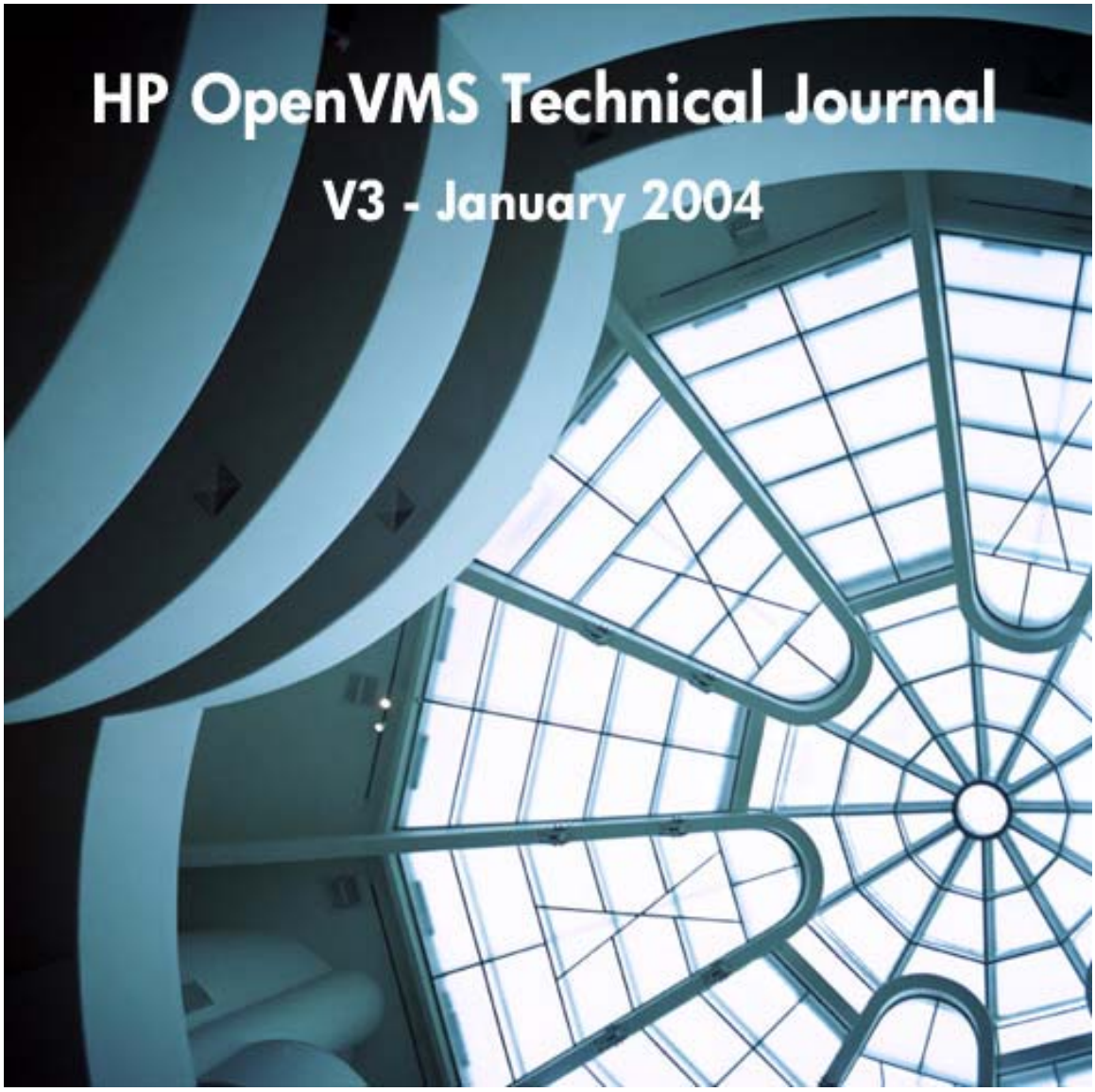

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Programming with TCP/IP – Best Practices

Matt Muggeridge
TCP/IP for OpenVMS Engineering

> "Be liberal in what you accept, and
> conservative in what you send"
>
> Source: RFC 1122, section 1.2.2 [Braden, 1989a]

Overview

A seasoned network programmer appreciates the many complexities and pitfalls associated with meeting the requirements of robustness, scalability, performance, portability, and simplicity for an application that may be deployed in a heterogeneous environment and a wide range of network configurations. The TCP/IP programmer controls only the end-points of the network connection, but must provide for all contingencies, both predictable and unpredictable. Therefore, an extensive knowledge base is required. The TCP/IP programmer must understand the relationship among network API calls, protocol exchange, performance, system and network configuration, and security.

This article is intended to help the intermediate TCP/IP programmer who has a basic knowledge of network APIs in the design and implementation of a TCP/IP application in an OpenVMS environment. Special attention is given to writing programs that support configurations where multiple NICs are in use on a single host, known as a multihomed configuration, and to using contemporary APIs that support both IPv4 and IPv6 in a protocol-independent manner. Key differences between UDP and TCP applications are identified and code examples are provided.

This article is not the most definitive source of information for TCP/IP programmers. There are many more topics that could be covered, as is evident by the number of expansive text books, web-sites, newsgroups, RFCs, and so on.

The information in this article is organized according to the structure of a network program. First, the general program structure is introduced. Subsequent sections describe each of the phases: Establish Local Context, Connection Establishment, Data Transfer, and Connection Shutdown. The final section is dedicated to important general topics.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Structure of a Network Program

All network programs are structured in a similar way, regardless of the complexity of the service they provide. They consist of two peer applications: one is designated as the server and the other is the client (see Figure 1). Each application creates a local end-point (*socket*), and associates (*binds*) a local name with it that is identified by the three-tuple: *protocol, local IP address,* and *port number*. The named end-point can be referenced by a peer application to form a connection that is uniquely identified in terms of the named end-points. Once connected, data transfer occurs. Finally, the connection is shut down.

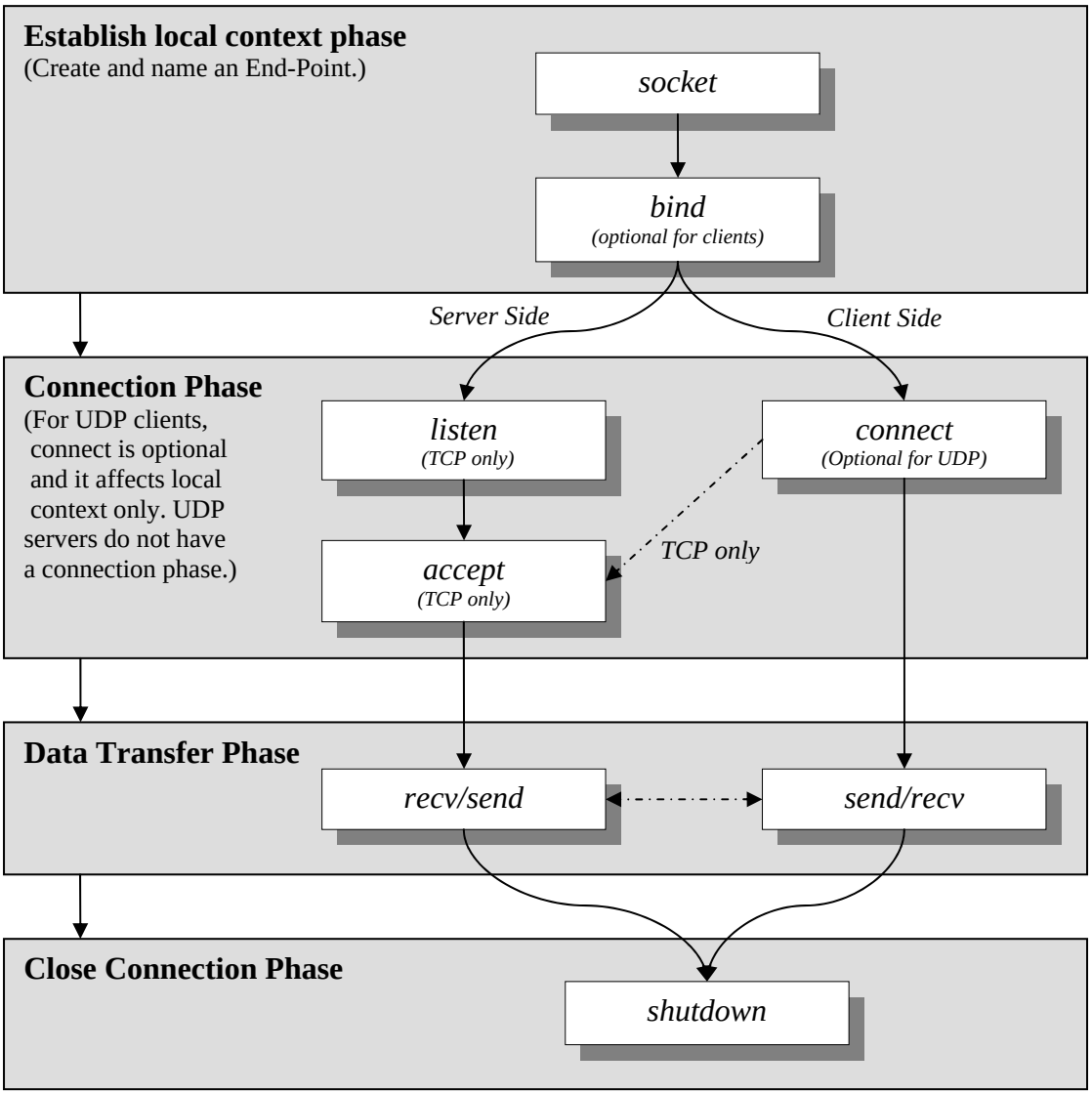

Figure 1 Structure of a Network Program

You have to take special measures to support multihomed configurations for UDP applications. In addition, by using the modern API's, network programs can readily support both IPv4 and IPv6. Writing applications that are largely independent of the version of the internet protocol (IPv4 or IPv6) requires the use of simple address conversion APIs.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The structure of any network program is independent of the API. Here it is described in terms of the BSD API. Most TCP/IP applications use the BSD sockets API, which was introduced with BSD V4.2 in 1983. OpenVMS programmers also have the option of using the $QIO system services, which may be preferable, especially when designing an event-driven application.

The remainder of this document is divided into sections that match the structure of a network program, as shown in Figure 1.

Establishing Local Context Phase

Establishing local context begins with deciding on the most appropriate transport protocol: TCP, UDP, or both. In addition, requirements for establishing local context differ based on whether the application is the server or the client. For instance, a server should be able to accept incoming connections directed to any IPv4 or IPv6 address configured on the system. A server design (TCP or UDP) may require multiple threads to accept incoming connections on the same port and address for either or both TCP and UDP. A TCP server must avoid the TCP TIME_WAIT state so that it can be restarted instantly. The client (UDP or TCP) must be aware that a server may be identified by a list of addresses and should connect to the more preferred address. These points are discussed in more detail in the following subsections and are summarized in the list below:

- Select the Appropriate Protocol – UDP or TCP
- Providing for UDP Behaviors
- Creating an End-Point
- Naming an Endpoint
- Servers Explicitly Bind to a Local End-Point
- Clients Implicitly Bind to Their End-Point
- UDP Servers Enable Ancillary Data
- Servers Reuse Port and Address
- UDP Servers Enable Ancillary Data
- Management of Local Context Phase

Select the Appropriate Protocol – UDP or TCP

One of the earliest decisions a TCP/IP programmer must make is whether the application will use a *datagram* or *stream* socket type. This decision determines whether the transport protocol will be UDP or TCP, because UDP uses the datagram socket type and TCP uses the stream socket type.

The four socket types are compared in Table 1. Only the *datagram* and *stream* socket types are discussed in this article. The *raw* socket provides access to underlying communication protocols and is not intended for general use. The *sequenced* socket is not implemented by TCP/IP Services for OpenVMS. The SCTP protocol (RFC 2960) uses sequenced sockets.

Table 1. Socket Types and Characteristics

	RAW	*DATAGRAM*	*STREAM*	*SEQUENCED*
Bidirectional	✓	✓	✓	✓
Reliable			✓	✓
Sequenced			✓	✓
No Duplicates			✓	✓
Record Boundaries		✓		✓

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The User Datagram Protocol (UDP) is connectionless, and supports broadcasting and multicasting of datagrams. UDP uses the *datagram* socket service, which is not reliable; therefore, datagrams may be lost. Also, datagrams may be delivered out of sequence or duplicated. However, record boundaries are preserved; a `recvfrom()` call will result in the same unit of data that was sent using the corresponding `sendto()`.

In a UDP application, it is the responsibility of the programmer to ensure reliability, sequencing, and detection of duplicate datagrams. The UDP broadcast and multicast services are not well suited to a WAN environment, because routers will often block broadcast and multicast traffic. Also because WANs are generally less reliable, a UDP application in a WAN environment may suffer from the greater processing overhead required to cope with data loss, which may in turn flood the WAN with retransmissions. UDP is particularly suited to applications that rely on short request-reply communications in a LAN environment, such as DNS (the Domain Name System) or applications that use a polling mechanism such as the OpenVMS Load Broker and Metric Server.

The Transmission Control Protocol (TCP) is connection-oriented; provides reliable, sequenced service; and transfers data as a stream of bytes. Because TCP is connection-oriented, it has the additional overhead associated with connection setup and tear-down. For applications that transfer large amounts of data, the cost of connection overhead is negligible. However, for short-lived connections that transfer small amounts of data, the connection overhead can be considerable and can lead to performance bottlenecks. Examples of TCP applications that are long-lived or transfer large amounts of data include Telnet and FTP.

Providing for UDP Behaviors

UDP is designed to be an inherently unreliable and simple protocol. Do not expect errors to be returned when datagrams are lost, arrive out of sequence, dropped, or duplicated. You may find it necessary to overcome these behaviors. It is the responsibility of the UDP application to detect these conditions, and it must take the appropriate action according to the application's needs. At some point, you may be duplicating the behavior of the TCP protocol in the application, in which case you should reconsider your choice of protocol.

Creating an End-Point

In a BSD-based system, the socket defines an end-point. It is a local entity and is used to establish local context only. The end-point is created using the BSD `socket()` function. See Example 1.

```
int socket(int domain, int type, int protocol);

Where the arguments are:
      domain – may be either AF_INET (IP version 4) or AF_INET6 (IP version 6)
      type – field may be either SOCK_STREAM (TCP) or SOCK_DGRAM (UDP)
      protocol – set to zero, because the protocol is implied by the type argument
```

Example 1 Creating an End-Point

Naming an Endpoint

An end-point is uniquely identified by its name. The name is defined by the *protocol, local IP address,* and *local port number* using the `bind()` function, as shown in Example 2.

The server-side application must `bind()` a name to its socket so that clients can reference the service. It is not recommended for the client-side application to call `bind()`. When a client does

© Copyright 2004 Hewlett-Packard Development Company, L.P.

not explicitly call `bind()`, the kernel will implicitly bind a name of its choosing when the application calls either `connect()` for TCP, or `sendto()` for UDP.

```
int bind(int socket, struct sockaddr *address, int address_len);

Where the arguments are:
      socket – value returned by calling the socket() function
      address – socket address structure
      address_len – length of socket address structure

Note that the "address" structure and "address_len" value are best initialized by
calling getaddrinfo().
```

Example 2 Naming an Endpoint

Servers Explicitly Bind to a Local End-Point

A service is identified by its *protocol, local IP address,* and *local port number.* The application advertises its service as either TCP or UDP on a specific local port number. (When a service binds to a local port number below 1024, the process requires one of the following privileges: SYSPRV, BYPASS, or OPER.) A server application should be capable of accepting connections on all IP addresses configured on the host, including IPv4 and IPv6 addresses.

Binding to all addresses is easiest to achieve by binding to the special address known as INADDR_ANY (IPv4) or IN6ADDR_ANY_INIT (IPv6). However, by using the protocol-independent APIs, the differences between IPv6 and IPv4 become less relevant. The server can readily be programmed to accept incoming TCP connections (or UDP datagrams) sent to any interface configured with an IPv4 or IPv6 address.

Use `getaddrinfo()` to return the list of all available socket addresses, (see Example 3). This function accepts hostnames as alias names, or in numeric format as IPv4 or IPv6 strings. In a multihomed environment, this may be a long list. For instance, a system configured with IPv4 and IPv6 addresses will return a socket address for each of the following protocol combinations: TCP/IPv6, UDP/IPv6, TCP/IP and UDP/IP.
Example 4 demonstrates the method for establishing local context for each of the socket addresses configured on a system, independent of IPv6 or IPv4.

Clients Implicitly Bind to Their End-Point

Whereas a server must explicitly `bind()` to its local end-point so that its service may be accessible, a client does not advertise a service. Hence, a client is able to use any local IP address and local port number. This is achieved by the client skipping the `bind()` call, (see the client path in Figure 1). Instead, when a TCP client issues `connect()`, or a UDP client issues `sendto()`, an implicit binding is made. The bound IP address is determined from the routing table and the order of addresses configured on an interface. The local port number is dynamically assigned and is referred to as an *ephemeral* port.

Note that the ephemeral port numbers are selected from a range specified by the following `sysconfig inet` attributes [Hewlett-Packard Company, 2003c]:

`ipport_userreserved` (specifies the maximum ephemeral port number)
`ipport_userreserved_min` (specifies the minimum ephemeral port number)

© Copyright 2004 Hewlett-Packard Development Company, L.P.

These values can be modified with the following command:

```
$ sysconfig –r inet ipport_userreserved=65535 ipport_userreserved_min=50000
```

Servers Reuse Port and Address
You should be aware of the following limitations with respect to servers binding to their local port and local IP address.

By default, the server's local port number and local address can be bound only once. Subsequent attempts to bind another instance of the server to the same local port and local address will fail, even if it is using a different protocol. This is a problem for servers that must advertise UDP and TCP services on the same port.

If a server performs an *active close*, the TCP state machine [Stevens, 1994] forces it into the TIME_WAIT state which, amongst other things, prevents an application from binding to the same local IP address and local port number. An active close is performed by the peer application that first issues the `shutdown()`, which causes TCP to send a FIN packet (see Figure 3). The peer that receives the FIN packet performs a *passive close* and is not subject to the TIME_WAIT state. The TIME_WAIT state lasts for at least twice the maximum segment lifetime (`sysconfig inet` attribute `tcp_msl` [Hewlett-Packard Company, 2003c].). By default this is 60 seconds. For the duration of the TIME_WAIT state, a subsequent attempt to `bind()` to the same local address

```
int getaddrinfo(const char *nodename, const char *servname,
                const struct addrinfo *hints, struct addrinfo **res);

Where the arguments are:
      nodename – string representing nodename as an alias or numeric format
      servname – string representing service name or port number
      hints – filters addresses with matching fields in addrinfo structure
      res – linked list of returned addresses
```

Example 3 Obtaining Addresses with Protocol Independent API

and local port number will return an error (EADDRINUSE). The service would be unavailable for at least 60 seconds. You can prevent the server from going through the TIME_WAIT state by having the client perform the active close, which causes no problems because the client will obtain a new ephemeral port for each invocation. Despite having a well-designed client, a server will still perform an active close if it exits unexpectedly or is forced to issue the active close for some other reason, and you must avoid this situation.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int sd[MAX_SOCKS]; /* one per TCP/IPv6, UDP/IPv6, TCP/IP, UDP/IP */
char *port, *addr = NULL;
struct addrinfo *res, hints;

port = argv[1]; /* port number as a string – must not be NULL */
if(argc == 3) addr = argv[2]; /* hostname – NULL implies ANY address */

memset(&hints, '\0', sizeof(hints));
hints.ai_flags = AI_PASSIVE; /* if usrreq.addr NULL, sets sockaddr to ANY */

err = getaddrinfo(usrreq.addr, usrreq.port, &hints, &res);
if(err) {
   if(err == EAI_SYSTEM) perror("getaddrinfo");
   else printf("getaddrinfo error %d - %s", err, gai_strerror(err));
   return 1;
}

i = 0;
for(aip = res; aip; aip = aip->ai_next) {
   if(aip->ai_family != AF_INET && aip->ai_family != AF_INET6) continue;

   /* create a socket for this protocol */
   sd[i] = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
   if(sd[i] < 0) {perror("socket"); return sd[i];}

   err = socket_options(sd[i], aip); /* set SO_REUSEADDR, SO_REUSPORT etc. */
   if(err == -1) {perror("socket_options"); return 1;}

   err = bind(sd[i], res->ai_addr, res->ai_addrlen);
   if(err == -1) {perror("bind"); return 1;}

  /** perform other per-socket work here – e.g. maybe create threads etc **/

   if(i == NUM_ELT(sd)) {printf("Insufficient socket elements\n"); break;}
   i++;
}
freeaddrinfo(res);
```

Example 4 Server Establishes Context for All Addresses

To overcome these issues, you must modify the server's socket to allow it to rebind to the same address and port number multiple times and without delay. This is implemented as a call to `setsockopt()`, as shown in Example 5.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int on = 1;

/* allow server to reuse address when binding */
err = setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char *)&on, sizeof(on));
if(err < 0) {perror("setsockopt SO_REUSEADDR"); return err;}

/* allows UDP and TCP to reuse port (and address) when binding */
err = setsockopt(sd, SOL_SOCKET, SO_REUSEPORT, (char *)&on, sizeof(on));
if(err < 0) {perror("setsockopt SO_REUSEPORT"); return err;}
```

Example 5 Setting Socket Options to Reuse Port and Address

UDP Servers Enable Ancillary Data

In a multihomed environment, a UDP server requires special care when replying to a request. It may reply to a client using any appropriate interface, setting the outgoing source address to that interface. That is, the reply source address does not have to match the request's destination address.

This creates problems in environments protected by a firewall that monitors source and destination addresses. If a packet that has a reply source address that does not match the request's destination address, the firewall interprets this as address spoofing and drops the packet. Also, a client using a connected UDP socket will only receive a datagram with a source/destination address pair matching what it specified in the `connect()` call. Therefore, the server socket must be enabled to receive the destination source address and the server must reply using that address as the reply source address. Sample code that enables a socket to receive the destination address information is shown in Example 6. This is an area where there are differences between IPv6 and IPv4, so they must be treated individually.

```
int err, on = 1, len = sizeof(on);

/* UDP should reply using dst address of the request */
if(ai->ai_protocol == IPPROTO_UDP) {
   if(ai->ai_family == AF_INET) { /* must be IPv4 - enable recvdstaddr */
      err = setsockopt(sd, IPPROTO_IP, IP_RECVDSTADDR, (char *)&on, len);
      if(err < 0) {perror("setsockopt IP_RECVDSTADDR"); return err;}
   }
   else { /* must be IPv6 - enable recvpktinfo and pktinfo */
      err = setsockopt(sd, IPPROTO_IPV6,IPV6_RECVPKTINFO, (char *)&on, len));
      if(err < 0) {perror("setsockopt IP_RECVPKTINFO"); return err;}

      err = setsockopt(sd, IPPROTO_IPV6,IPV6_PKTINFO,(char *)&on, len);
      if(err < 0) {perror("setsockopt IP_PKTINFO"); return err;}
   }
}
```

Example 6 UDP Servers Enable Ancillary Data

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Management of Local Context Phase
The number of sockets that can be opened is limited by the OpenVMS `CHANNELCNT sysgen` parameter, with one channel per socket. In addition, starting with TCP/IP Services for OpenVMS V5.4, a new `sysconfig net` subsystem attribute `ovms_unit_maximum` can extend the limit. The ephemeral port range is defined by the `sysconfig inet` attributes `ipport_userreserved and ipport_userreserved_min.` The `sysconfig` attributes are discussed in more detail in [Hewlett-Packard Company, 2003c].

Connection Phase

The connection phase has a different meaning depending on whether TCP or UDP is being used. In the case of TCP, the establishment of a connection results in a protocol exchange between the peers and, if successful, each peer maintains state information about that connection. Similarly, when a TCP connection is shut down, it results in a protocol exchange that affects a change in state of each peer. (See [Stevens, 1994] for more information about the TCP state machine.) Before establishing a connection, a TCP server application must issue `listen()` and `accept()` calls. The TCP client application initiates the connection by calling `connect()`.

UDP, on the other hand, is a connectionless protocol and the connection phase is optional. However, a UDP socket may be connected, which serves only to establish additional local context about the peer's address. That is, a UDP connect request does not result in any protocol exchange between peers. When a UDP socket is connected, the application will receive notifications generated by incoming ICMP messages and it will receive datagrams only from the peer that it has connected to. In other words, if a UDP socket is *not* connected, it is unable to receive notifications from ICMP packets and it will receive datagrams from any address. In fact, it is common for the UDP client to connect the server address, while the UDP server never connects the client address. A UDP `connect()` is similar to binding, where `bind()` associates a *local address* with the socket; `connect()` associates the *peer address* with the socket.

Because connecting TCP sockets is different from connecting UDP sockets, they are treated separately in the following subsections:

- TCP Connection Phase
- Optional UDP Connection Phase
- `connect()` and Address Lists
- Resolve Host Name Prior to Every Connection Attempt
- Server Controls Idle Connection Duration with Keepalive
- TCP Server's Listen Backlog
- Management of Connection Phase

TCP Connection Phase
TCP connection establishment defines a protocol exchange, often referred to as the "three-way handshake," between client and server. The API calls and resulting protocol exchange are shown in Figure 2.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

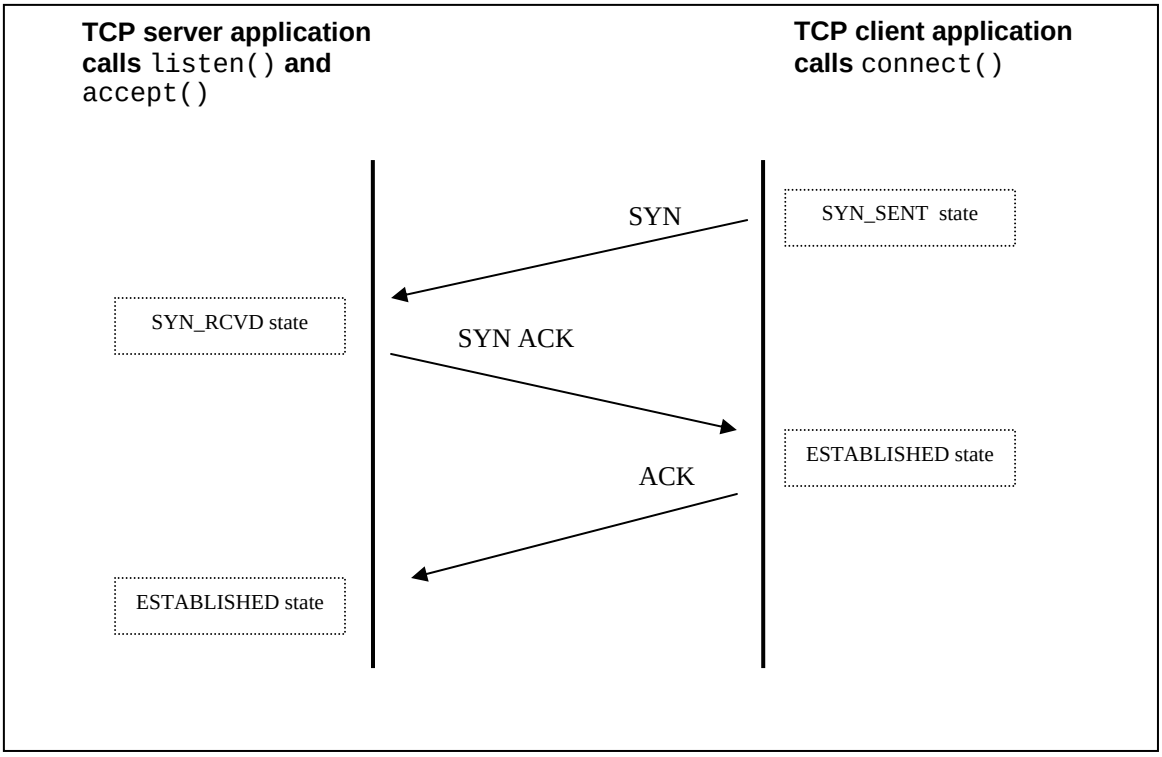

Figure 2 TCP Connection Establishment - "Three-Way Handshake"

Before a server can receive a connection, it must first issue `listen()` and `accept()`. These are illustrated in Example 7.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int listen(int socket, int backlog);

Where the arguments are:
      socket – value returned by calling the socket() function
      backlog – maximum number of outstanding connection requests
```

```
int accept(int socket, struct sockaddr *fromaddr, int *fromlen);

Where the arguments are:
      socket – value returned by calling the socket() function
      fromaddr – socket address of the peer we're accepting the connection from
      fromlen – length of the socket address

The return value is a new socket descriptor that can be used for data transfer between
the client and server.

Note that if the peers address details need to be recorded, they are best decoded with
getnameinfo().
```

Example 7 TCP Server Connection Phase

Once the server is ready to accept incoming connections the TCP client initiates the connection by calling `connect()`:

```
int connect(int socket, struct sockaddr *toaddr, int *tolen);

Where the arguments are:
      socket – value returned by calling the socket() function
      toaddr – socket address describing the peer to connect to
      tolen – length of the socket address

Note that if the toaddr and tolen fields may be initialized by calling getaddrinfo().
```

Example 8 TCP or UDP Client Connect Phase

Optional UDP Connection Phase

Unlike the TCP client where the `connect()` call is required, a UDP client may optionally call `connect()`. The API for connecting a UDP socket is the same as that used to connect a TCP socket, as shown in Example 8. During a UDP `connect()` the destination address is bound to the socket; therefore, when sending a UDP datagram, it is an error to specify the destination address with each datagram. To transmit data over the connected UDP socket, use the `sendto()`function with a NULL destination address, or use the `send()` function.
A connected UDP socket does not change the behavior of UDP as a connectionless protocol. There is no protocol exchange when a UDP socket is connected or shut down, and there is no

© Copyright 2004 Hewlett-Packard Development Company, L.P.

state machine. Connecting a UDP socket affects the local context only. A connected UDP socket allows the kernel to deliver errors to the user application as the result of received ICMP messages. Unconnected UDP sockets do not receive errors as a result of an ICMP message. For example, when a connected UDP socket attempts to send a datagram to a host without a bound service, the ICMP "port unreachable" message is returned and the kernel reports this to the application as a "connection refused" error. The client application is alerted that the service is not running.

As a result of connecting a UDP socket, the programmer must not specify the destination address with each datagram, because it is already bound to the socket. Unlike `bind()`, which binds a local name (IP address and port) to a socket, the UDP `connect()`call binds the remote name (IP address and port) to the UDP socket.

Because a UDP server must accept incoming datagrams from many remote clients and a connected UDP socket limits the communication to one peer at a time, the UDP server should not use connected sockets. Furthermore, a multihomed UDP server may reply with a source address that differs from the client's destination address. If the client is using a connected UDP socket, then datagrams that do not match the address in the connected UDP socket will not be delivered to the client. See Example 20 for programming a UDP server in a multihomed environment.

`connect()` and Address Lists

Host names are often stored in a name server as DNS aliases. It is not unusual for a DNS alias to be represented by multiple IP addresses, usually for the purpose of offering higher availability or load sharing [Muggeridge, 2003]. When an application resolves a host name, the resolver will return the list of addresses associated with that host name. The address list is usually sorted with the most desirable address at the top of the list.

The client should call `getaddrinfo()` (see Example 9) to retrieve the list of IP addresses and then cycle through this list, attempting to connect to each address until a successful connection is established. (RFC 1123 Sec 2.3. [Braden, 1989b])

Resolve Host Name Prior to Every Connection Attempt

Addresses are not permanent – they can change or become unavailable. For example:

- A system administrator may add or remove addresses during a migration exercise.
- High availability configurations using the Load Broker/Metric Server are designed to dynamically update the DNS alias with a modified address list [Muggeridge, 2003].

Therefore, it is highly recommended that applications never cache IP addresses. When the client connects or reconnects, it should resolve the server's DNS alias each time, using `getaddrinfo()`. This makes the client resilient to changes in the servers' address list. (See Example 9).

Keep in mind that DNS may also be caching bad addresses. Even if your application performs the name-to-address conversion again, it may receive the same obsolete list. Some strategies for ensuring the DNS alias lists are current include making an address highly available with failSAFE IP, or dynamically keeping the DNS alias address list current using Load Broker/Metric Server. For more information, on this refer to [Muggeridge, 2003].

Server Controls Idle Connection Duration with Keepalive

Because a server assigns resources for every connection, it should also control when to release the resources if the connection remains idle. A polling mechanism used to ensure the peer is still connected can be used to keep the connection alive. TCP has a "keepalive" mechanism built into the protocol; however, UDP does not.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
char *srv_addr, *srv_port;
struct addrinfo *srv_res, *ai, hints;

srv_addr = argv[1];  /* server address */
srv_port = argv[2];  /* sever port     */

memset(&hints, '\0', sizeof(hints));
hints.ai_family   = usrreq.family;
hints.ai_socktype = usrreq.type;

/* get remote address info */
err = getaddrinfo(srv_addr, srv_port, &hints, &srv_res);
if(err) {
   if(err == EAI_SYSTEM) perror("getaddrinfo");
   else printf("getaddrinfo error %d - %s", err, gai_strerror(err));
   return 1;
}
for(ai = srv_res; ai; ai = ai->ai_next) {
   /* AF_INET and AF_INET6 only */
   if(ai->ai_family != AF_INET && ai->ai_family != AF_INET6) continue;

   sd = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
   if(sd < 0) {perror("socket"); continue;}  /* try next socket */

   err = socket_options(sd, ai);
   if(err) {perror("sockopt"); continue;}  /* try next socket */

   /* use connected UDP sockets if userreq.connected is set */
   if(ai->ai_protocol == IPPROTO_TCP || usrreq.connected) {
       err = connect(sd, ai->ai_addr, ai->ai_addrlen);
       if(err == -1) {perror("connect"); continue;} /* try next socket */
   }
   break; /* use first successful connection */
}
if(err == -1) {printf("No connection"); return 1;}

/** data transfer phase **/
```

Example 9 Client Connects to Each Server Address Until Success

TCP is a wonderfully robust protocol that can recover from lengthy network outages, but this can result in zombie connections on the server. A zombie connection is one that is maintained by just one of the peers after the other peer has exited. For example, a cable modem may be powered off before the client application shuts down the connection. Because the modem has been powered off, the TCP client cannot notify the server that it is shutting the connection. As a result, the server-side application unwittingly maintains the context of the connection. This is not unusual with home networks. Without any notification of the client being disconnected, the TCP server will maintain its connection indefinitely.

There are a number of ways to solve this problem. At the application level, a keepalive message can be transferred between peers. When a peer stops responding for a configurable number of keepalives the connection should be closed. Alternatively, the system manager can enable a system-wide keepalive mechanism that will affect all TCP connections. This is controlled using

© Copyright 2004 Hewlett-Packard Development Company, L.P.

the following `sysconfig inet` atrributes: `tcp_keepidle`, `tcp_keepcnt`, and `tcp_keepintvl` [Hewlett-Packard Company, 2003c].

These system configuration parameters are useful when an application does not provide a mechanism for closing zombie connections. An application must be restarted to pick up changes in the keepalive `sysconfig` attributes.

Because UDP provides no way to determine the availability of its peer, you can implement a keepalive mechanism at the application level for this purpose.

TCP Server's Listen Backlog

The TCP server's listen backlog is a queue of connection requests for connections that have not been accepted by the application. When the connection has been accepted by the application, the request is removed from the backlog queue. The length of the backlog is set by the `listen()` call. If this backlog queue becomes full, new connection requests are silently ignored, which may lead to clients suffering from timeouts on their new connection attempts.

When a TCP application issues a successful `connect()` request, it results in a "three-way handshake" (shown in Figure 2). When the `connect()` request is unsuccessful, you have to provide for all potential failures.

Note from Figure 2 that the client and server enter the ESTABLISHED state at different times. Therefore, if the final ACK is lost, it is possible for the client to believe it has established a connection, while the server remains in SYN_RCVD state. The server must receive the final ACK before it believes the connection is established.

Consider the impact that this protocol exchange has on the peer applications. For example, after the first SYN is received, the TCP server tracks the connection request by adding the peer details to an internal socket queue *(so_q0)*. When the final ACK is received, the peer details are moved from *so_q0* to another internal queue *(so_q)*. The connection state is freed from *so_q* only when the application's `accept()` call completes. (For more details see [Wright, 1995].)

The socket queues will grow under any of the following conditions:

- The rate of incoming SYN packets (connection requests) is greater than the completion rate of `accept()`.
- The final ACK is slow in arriving, or an acknowledgement (SYN ACK or ACK) is lost.
- The final ACK never arrives (as in the case of a SYN flood attack).

If the condition persists, the socket queues will eventually become full. Subsequent SYN packets will be silently dropped (that is, the TCP server does not respond with SYN ACK segments). Eventually, the client-side application will time out. The client timeout will occur after approximately `tcp_keepinit/2` seconds (75 seconds by default).

The length of this socket queue is controlled by several attributes. You can specify the queue length in the `listen()` call. This can be overridden with the `sysconfig` attributes `sominconn` and `somaxconn`, but the server application must be restarted to use these system configuration changes. Restarting a busy server may not be practical, so it is better to treat this as a sizing concern, and ensure that the `accept()` call is able to complete in a timely fashion, given the rate of requests and the length of the listen queue.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Management of Connection Phase
The state of a connection can be viewed using the commands:

- `netstat`
- `tcpip show device`

These commands are described in [Hewlett Packard, 2003b].

The following `sysconfig` attributes of the `socket` subsystem affect the connection phase: `sominconn, somaxconn, sobacklog_drops, sobacklog_hiwat, somaxconn_drops, tcp_keepalive_default, tcp_keepcnt, tcp_keepidle, tcp_keepinit, tcp_keepintvl` [Hewlett-Packard, 2003c].

Data Transfer Phase

The data transfer phase provides the means of transferring data between the peer applications and, as with the connection establishment phase, there are different choices to be made depending on whether TCP or UDP is being used. The topics covered in this section include:

- Data Transfer APIs
- Avoid MSG_PEEK When Receiving Data
- Avoid Out-Of-Band Data
- TCP Data Transfer – Stream of Bytes
- UDP Data Transfer – Datagram Units
- UDP Datagram Size
- Understand Buffering
- Management of Data Transfer Phase

Data Transfer APIs
Several functions can be used for data transfer. The choice of function depends on whether the socket is connected or unconnected. A TCP socket must be connected, whereas a UDP socket may be connected or unconnected (see Example 8). A connected socket may transmit data with `send()` or `write()` and receive data with `recv()` or `read()`. The `send()` and `recv()` functions support a "flags" argument that `write()` and `read()` do not. Unconnected sockets require functions that support a "destination address" in the API. These functions include `sendto()` or `sendmsg()`, and `recvfrom()` or `recvmsg()`. The list of examples that demonstrate these API calls are:

- Example 10 Connected Socket Data Transfer APIs
- Example 11 Unconnected UDP Sockets Data Transfer APIs
- Example 12 UDP Data Transfer
- Example 13 TCP Receive Algorithm
- Example 14 UDP Receive Algorithm
- Example 15 Retrieving and Modifying Socket Options

For connected sockets the `recv()` and `send()` functions are shown in Example 10.

For unconnected sockets, the destination address of the peer must be sent with each message. Similarly, when receiving each message, the peer's address is available to the application. The `sendto()` and `recvfrom()` functions support the peer's address, as show in Example 11.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Note that for `sendto()`, the peer's destination address is specified by the arguments `dstaddr` and `dstlen`, which should be initialized using `getaddrinfo()`. For `recvfrom()`, the peer's address is available in the `fromaddr` argument and can be resolved with `getnameinfo()`.

```
int recv(int socket, char *buffer, int length, int flags);

int send(int socket, char *message, int length, int flags);

Where the arguments are:
      socket – value returned by calling the accept() function
      message – buffer containing data to be sent
      length – length of the message to send
      flags – sender may control transmission of the message.
```

Example 10 Connected Socket Data Transfer APIs

```
int sendto(int socket, char *message, int length, int flags,
           struct sockaddr *dstaddr, int dstlen);

int recvfrom(int socket, char *message, int length, int flags, struct sockaddr
*fromaddr, int *fromlen);

Where the arguments are:
      socket – socket descriptor for data transfer
      message – buffer containing data to be sent
      length – length of the message to send
      flags – sender may control transmission of the message
      dstaddr – socket address of destination
      dstlen – length of dstaddr
      fromaddr – socket address of peer from where the data was sent
      fromlen – length of fromaddr
```

Example 11 Unconnected UDP Sockets Data Transfer APIs

For UDP sockets, (regardless of whether they are connected or unconnected) the `sendmsg()` and `recvmsg()` routines may be used. These routines provide a special interface for sending and receiving *ancillary data*[1], as shown in Example 12. A socket may be enabled to receive ancillary data with `setsockopt()`. A special use of the received ancillary data is shown in Example 17 through Example 21.

Note that the `sendmsg()` and `recvmsg()` functions are particularly important in a UDP server application in a mulithomed configuration; see section "UDP and Multihoming" on page 21.

[1] UDP over IPv4 ignores ancillary data for `sendmsg()`.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int sendmsg(int socket, const struct msghdr *message, int flags);

int recvmsg(int socket, struct msghdr *message, int flags);

Where the arguments are:
      socket – value returned by calling the accept() function
      message – describes the data to be sent and ancillary data
      flags – sender may control transmission of the message.
```

Example 12 UDP Data Transfer

The `msghdr` structure contains a field, `msg_control`, for ancillary data. IPv4 currently ignores this field for transmission. IPv6 uses the ancillary data as described in RFC 3542 [Stevens et. al., 2003].

The `recv()`, `recvfrom()`, and `recvmsg()` APIs support a `flags` argument that can be used to control message reception. One of the options is MSG_PEEK, which allows the application to peek at the incoming message without removing it from the socket's receive buffer. Another option is MSG_OOB, which supports the processing of out-of-band data. These features are not recommended, as explained in the next two sections.

Avoid MSG_PEEK When Receiving Data

With today's modern networks and high-performing large memory systems, MSG_PEEK is an unnecessary receive option. The MSG_PEEK function looks into the socket receive buffer, but does not remove any data from it. Keep in mind that the objective of any network application is to keep data flowing through the network. Because the MSG_PEEK function does not remove data from the receive buffer, it will cause the receive window to start closing, which applies back pressure on the sender and can result in an inefficient use of the network. In any case, after a MSG_PEEK, it is still necessary to read the data from the socket. So an application may as well have done that in the first place and "peeked" inside its own buffer.

Avoid Out-Of-Band Data

Out-of-band (OOB) data provides a mechanism for urgently delivering a single byte of data to the peer. The receiving application is notified of OOB data and it may be read out of sequence. It is typically used for signaling. However, out-of-band data cannot be relied upon and is dependent on the implementation of the protocol stack. In many BSD implementations, if the out-of-band data is not read by the application before new out-of-band data arrives, then the new OOB data overwrites the unread OOB data. Instead of using OOB data for signaling, a better approach is to create a dedicated connection for signaling.

TCP Data Transfer – Stream of Bytes

Possibly the most common oversight of TCP/IP programmers is that they fail to realize the significance of TCP sending a stream of bytes. In essence, this means that TCP guarantees to deliver no more than one byte at a time; no matter how much data the user application sends. The amount of data that TCP transmits is affected by a wide variety of protocol events such as: send window size, slow start, congestion control, Nagle algorithm, delayed ACKS, timeout events and so on [Stevens, 1994], [Snader, 2000]. In other words, data is delivered to the peer in differently sized chunks that are independent of the amount of data that is written to TCP with each `send()` call. For a TCP application, this means that, when receiving data, the algorithm must loop on a `recv()` call. See Example 13.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
while((nbytes = recv(sd, streambuf, sizeof(streambuf), 0)) > 0)
{
   /* assemble the TCP byte stream into a message buffer */
   if(msg_assembled(message, streambuf, nbytes)) perform_action(message);
}
if(nbytes == -1) perror("recv");
```

Example 13 TCP Receive Algorithm

UDP Data Transfer – Datagram Units

UDP delivers datagram units in the same way they were sent, preserving record boundaries. Sample code showing how a UDP application may receive data is shown in Example 14.

```
if((nbytes = recvmsg[1](sd, &mhdr, 0)) == -1) perror("recvmsg");
```

Example 14 UDP Receive Algorithm

UDP Datagram Size

For robustness and responsible memory usage, do not assume the maximum size of a datagram when allocating buffer space within the application data structures. Avoid sending datagrams much larger than the maximum transmission unit, (MTU), because lost IP segments will require retransmission of the entire datagram. Also, limit the datagram size to be less than the socket option SO_SENDBUF.

Example 15 describes the APIs used to retrieve and modify the socket options. The maximum size of a datagram can be determined by calling `getsockopt()` with the `level` and `option_name` arguments set to SOL_SOCKET and SO_SENDBUF, respectively.

```
int getsockopt(int socket, int level, int option_name,
               void *option_value, socklen_t *option_len);

int setsockopt(int socket, int level, int option_name,
               char *option_value, socklen_t option_len);

Where the arguments are:
      socket – socket handle as returned by the socket() call.
      level – protocol layer for socket option
      option_name – socket option
      option_value – address of buffer to store the size of the socket send buffer
      option_len – address of integer to store the length of data returned
```

Example 15 Retrieving and Modifying Socket Options

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The maximum size of a socket buffer can be controlled by system-wide variables. These can be viewed and modified with the `sysconfig` utility. For example, to view these values you can use the following command [Hewlett-Packard Company, 2003c]:

```
$ sysconfig –q inet udp_sendspace udp_recvspace
```

To set the `udp_recvspace` buffer to 9216 bytes, use:

```
$ sysconfig –r inet udp_recvspace 9216
```

Changing these attribute settings will override programs that use the `setsockopt()` function to modify the size of their respective socket buffers.

Understand Buffering
A TCP or UDP application writes data to the kernel. The kernel stores the data in the socket send buffer. A successful write operation means that the data has been successfully written to the socket send buffer; it does not indicate that the kernel has sent it yet.

The procedure then involves two steps:

1. The data from the send buffer is transmitted and arrives at the peer's socket receive buffer. In the case of TCP, the delivery of data to the peer's socket receive buffer is guaranteed by the protocol, because TCP acknowledges that it has received the data. UDP provides no such guarantees and silently discards data if necessary. At this point, the data has not yet been delivered to the peer application.
2. The receiving application is notified that data is ready in its socket's receive buffer and the application reads the data from the buffer.

Because of this buffering, data can be lost if the receiving application exits (or the node crashes) while data remains in the receive socket buffer. It is up to the application to guarantee that data has arrived successfully at the peer application. TCP has completed its responsibility when it notifies the application that the data is ready.

Management of Data Transfer Phase
The following `sysconfig` attributes of the `socket` subsystem affect the data transfer phase: `tcp_sendspace, tcp_recvspace, udp_sendspace, tcp_recvspace, tcp_nodelack` [Hewlett-Packard, 2003c].

Connection Shutdown Phase

A connection is bidirectional; consequently, each side of the connection may be shut down independently. The following topics are described for the connection shutdown phase:

- TCP Orderly Release
- Management of Connection Shutdown

The `shutdown()` function is shown in Example 16.

```
int shutdown(int socket, int how);

Where the arguments are:
      socket – socket created for data transfer
      how – describes the direction to shutdown
```

Example 16 Connection Shutdown API

© Copyright 2004 Hewlett-Packard Development Company, L.P.

TCP Orderly Release

An application may not know how much data it will receive from a peer; therefore, each peer should signal when it has finished sending data, as in a telephone conversation in which both parties say "goodbye" to indicate they have nothing more to say. Similarly, a receiving application should not exit before it has received the signal indicating the last of the data. TCP applications can make use of the *half-close* to signal that a peer has finished sending data. When both peers have signaled this, the socket may be closed and the application can exit.

When a TCP application issues a `shutdown()` on the sending side of the socket, it results in the protocol exchange as shown in Figure 3. The TCP FIN packet is queued behind the last data. Because a connection is bidirectional, it requires a total of four packets to shut down both directions of a connection. The side that first issues the `shutdown()`on the sending side of the socket performs an *active close.* The side that receives the FIN performs a *passive close.* The difference between these is important, because the side issuing an *active close* must also wait in the TIME_WAIT state for 2 x MSL[2]. Some socket resources persist during the TIME_WAIT state. Because it is more critical to conserve server resources than client resources (see page 28), it is better practice to ensure the client issues the active close. See Servers Reuse Port and Address, on page 6, which discusses avoidance of the TIME_WAIT delay.

It is possible to `shutdown()` the receive side of a socket, but this is of little use, because shutting down the receive side does not result in a protocol exchange. In practice, the send direction of the socket is the more appropriate to shut down. Further attempts to send data on that socket will return an error. The peer reads the data until there is no more data in the receive socket buffer. When TCP processes the FIN packet, it closes that side of the connection and a subsequent `recv()` will return an error. This signals the receiver that the peer has no more data to send.

UDP applications do not provide for any protocol exchange when `shutdown()` is called. Instead, the programmer must design a message exchange that signals the end of transmission.

[2] MSL = Maximum Segment Lifetime

© Copyright 2004 Hewlett-Packard Development Company, L.P.

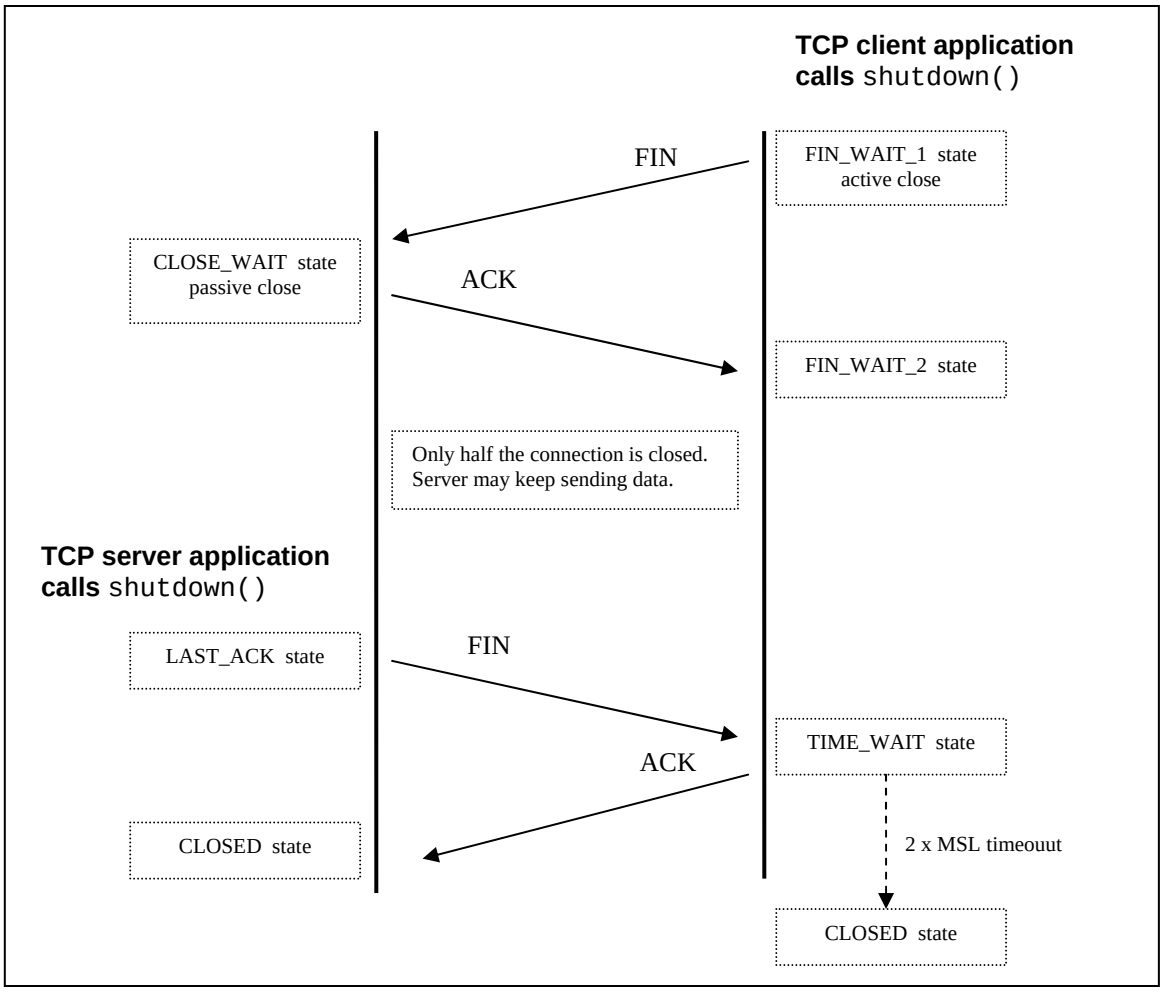

Figure 3. Closing a Connection - *4-Way Handshake*

Management of Connection Shutdown

The following `sysconfig` attributes of the `inet` subsystem affect the connection shutdown phase: `tcp_keepinit, tcp_msl`, [Hewlett-Packard, 2003c]. The network manager may also shut down a connection using the `TCPIP DISCONNECT` command, [Hewlett-Packard, 2003b].

Miscellaneous

A range of topics do not readily fit into the specific phases described in Figure 1, Structure of a Network Program. These are treated in the subsections below, including:

- UDP and Multihoming
- Concurrency In Server Applications
- Conservation of Server Resources
- Simplifying Address Conversion

UDP and Multihoming

Multihomed hosts are becoming more common. Recent features in TCP/IP Services for OpenVMS make it more desirable to configure a system with multiple interfaces. Depending on the configuration, this can have the following advantages:

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- Load-balancing of outgoing connections over interfaces configured in the same subnet
- Increased data throughput
- IP addresses can be protected when using failSAFE IP
- Multiple subnets per host

With these benefits, additional concerns arise for the TCP/IP programmer:

- Name to address translation can return a list of addresses (see "connect() and Address Lists," page 12).
- UDP servers ought to set the source address in the reply to be the same as the destination address in the request (see next subsection).

Each UDP datagram is transmitted with an IP source address that is determined dynamically[3]. The algorithm for choosing the IP source address is performed in two steps:

1. The routing protocol selects the most appropriate outbound interface
2. The subnet's primary address configured on that interface is assigned as the IP source address for the outbound datagram.

In a multihomed environment with multiple interfaces configured in the same subnet, this can result in successive datagrams with different IP source addresses. This becomes even more likely in a configuration that uses failSAFE IP, where the primary IP address may change during a failover or recovery event [Muggeridge, 2003].

An extract from RFC 1122 section 4.1.3.5 [Braden, 1989a] says:

```
A request/response application that uses UDP should use
a source address for the response that is the same as
the specific destination address of the request.
```

An extract from RFC 1123, section 2.3 [Braden, 1989b] says:

```
When the local host is multihomed, a UDP-based request/response
application SHOULD send the response with an IP source address
that is the same as the specific destination address of the UDP
request datagram.
```

If this recommendation is ignored, the application will be subject to the following types of failures.

1. Firewalls may be configured to pass traffic with specific source and destination addresses. In a single-homed host this does not present a problem, because a client will send a datagram to the server's IP address and the server will reply using that IP address in its source address field. When a second interface is configured with an address in the same subnet, the IP layer can choose either address as the reply source address. To solve this problem, either the firewall needs to be reconfigured or the addresses need to be reconfigured. If the UDP server application is written to always set the reply to the RECVDSTADDR then it will be more robust to changes in the network configuration.
2. Clients that use connected UDP sockets will only receive UDP datagrams from the server if its address matches the value stored in the UDP's connected socket.

The implementation differs depending on whether it is an AF_INET (IPv4) or AF_INET6 (IPv6) socket. However, the algorithm is essentially the same:

1. Enable the socket to receive ancillary data.

[3] Unless the server binds to a specific IP address, which is not recommended as described in "Servers Explicitly Bind to a Local End-Point," page 5.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

2. Receive the ancillary data that describes the destination address.
3. Reply using the source address that was received in the ancillary data.

Because IPv6 and IPv4 differ in the type of ancillary data that is needed, some general type definitions will help with making the implementation protocol-independent. These type definitions are shown in Example 17.

```
#include <sys/socket.h>
#include <net/in.h>

typedef union _recvdstaddr_u { /* ancillary data - IPv4 recvdstaddr */
   struct cmsghdr cm;
   char   ia[_CMSG_SPACE(sizeof(struct in_addr))];
} recvdstaddr_t;

typedef union _recvpktinfo_u { /* ancillary data - IPv6 recvpktinfo */
   struct cmsghdr cm;
   char   pktinfo[_CMSG_SPACE(sizeof(struct in6_pktinfo))];
} recvpktinfo_t;

typedef union _ancillary_u  { /* ancillary data - general control structure */
   recvdstaddr_t recvdstaddr; /* IPv4 */
   recvpktinfo_t recvpktinfo; /* IPv6 */
} ancillary_t;
```

Example 17 Ancillary Data Type Definitions for IPv4 and IPv6

Assuming the socket has been enabled to receive ancillary data (see Example 6) the sample library function in Example 18 can be called to receive IPv4 or IPv6 ancillary data.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int udp_recvdstaddr(int sd, void *buf, int buflen,
                    struct sockaddr_storage *from,
                    ancillary_t *recvdstaddr, unsigned int *controllen)
{
   struct msghdr mhdr; /* structure used to receive ancillary data */
   struct iovec iov[1]; /* data buffer to */
   int nbytes = 0, buflen = sizeof(buf);

   mhdr.msg_name = (char *)from; /* IP address of sender */
   mhdr.msg_namelen = sizeof(*from);
   mhdr.msg_iov = iov;  /* vector for scatter read */
   mhdr.msg_iovlen = 1; /* one buffer for scatter read */
   mhdr.msg_control = (char *)recvdstaddr;  /* ancillary rx data */
   mhdr.msg_controllen = *controllen;
   mhdr.msg_flags = 0;

   iov[0].iov_base = buf; /* data we receive from the peer */
   iov[0].iov_len = buflen;

   /* read data and ancillary data */
   if((nbytes = recvmsg(sd, &mhdr, 0)) == -1)
   {perror("recvmsg"); return -1;}

  *controllen = mhdr.msg_controllen;
   return nbytes;
}
```

Example 18 Receiving Ancillary Data

The implementation for sending a datagram with a specified source address varies depending on whether it is AF_INET (IPv4) or AF_INET6 (IPv6). The IPv6 socket interface simplifies this, (see RFC 3542 [Stevens et. al., 2003]). However, IPv4 ignores any ancillary data associated with the `sendmsg()` call, so it is necessary to create a separate reply socket and bind it to the desired local source address. A library function for doing this is shown in Example 19.

Notice that IPv6 readily supports ancillary data for UDP transmit, so the AF_INET6 case (Example 20) is straight forward (just a few lines of code). The AF_INET case, however, requires more than 20 lines of code and an additional five system calls (including the `udp_reply_sock()` routine) to perform the same action.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int udp_reply_sock(struct sockaddr_in *sin) /* needed for AF_INET only */
{
   int rsock, on = 1;

   if((rsock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
   {perror("reply socket"); return -1;}

   if(setsockopt(rsock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) == -1)
   {perror("reply setsockopt"); return -1;}

   if(setsockopt(rsock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on)) == -1)
   {perror("reply setsockopt"); return -1;}

   if(bind(rsock, (struct sockaddr *)sin, sizeof(*sin)) == -1)
   {perror("bind"); return -1;}

   return rsock;
}
```

Example 19 Receiving Destination Address from UDP

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int udp_sendsrcaddr(int sd, void *buf, int buflen, struct sockaddr_storage *dst,
                  ancillary_t *srcaddr, unsigned int controllen,
                  unsigned int localport)
{
   struct msghdr mhdr;  /* ancillary data message header */
   struct cmsghdr *cmsg = NULL;
   struct iovec iov[1]; /* vector used to reference data buffer */
   struct sockaddr_in sin;
   int rsock;
   int nbytes;

   /* init message header */
   mhdr.msg_name = (char *)dst; /* destination IP address */
   mhdr.msg_namelen = dst->ss_len;
   mhdr.msg_iov = iov;  /* vector for gather write */
   mhdr.msg_iovlen = 1; /* one buffer for gather write */
   mhdr.msg_control = (char *)srcaddr;  /* buffer pointing to ancillary data */
   mhdr.msg_controllen = controllen;
   mhdr.msg_flags = 0;

   /* init data buffer */
   iov[0].iov_base = buf; /* data we receive from the peer */
   iov[0].iov_len = buflen;

   switch(dst->ss_family) {
   case AF_INET:
      /* sndmsg() for IPv4 ignores ancillary data, so must bind to new socket */
      sin.sin_len = dst->ss_len;
      sin.sin_family = dst->ss_family;
      sin.sin_port = htons(localport);

      /* assumes RECVDSTADDR is only element of ancillary data */
      cmsg = CMSG_NXTHDR(&mhdr, cmsg);
      if(cmsg) { /* if there is ancillary data to send then copy it */
               /* save IPv4 source address to sin */
       memcpy(&sin.sin_addr, CMSG_DATA(cmsg), sizeof(struct in_addr));

       /* create a new socket and bind to the local address in sin */
       if((rsock = udp_reply_sock(&sin)) == -1)
       {printf("reply_sock failed\n"); return 0;}
      }
      else rsock = sd; /* no ancillary data - use same socket */

      /* send data using replysock */
      if((nbytes = sendmsg(rsock, &mhdr, 0)) == -1)
      {perror(errno); return -1; }

      if(cmsg) close(rsock); /* finished with newly created reply sock */
      break;
   case AF_INET6:
      if(controllen == 0) mhdr.msg_control = NULL;

      /* call sendmsg to force IP source address */
      if((nbytes = sendmsg(sd, &mhdr, 0)) == -1)
      {perror(errno); return -1;}
      break;
   }
   return nbytes;
}
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Example 20 UDP Reply with Source Address Set to Request's Destination Address

A UDP echo server might make use of these library functions in the following way:

```
len = sizeof(recvdstaddr);

if((bytes = udp_recvdstaddr(sd, buf, buflen, &rmtaddr, &recvdstaddr, &len)) <= 0)
{perror ("udp_recvdstaddr"); return -1;}

if(udp_sendsrcaddr(sd, buf, bytes, &rmtaddr, &recvdstaddr, len, port) == -1)
{perror("udp_sendsrcaddr"); return -1;}
```

Example 21 UDP Echo Server Using Correct Source Address

Concurrency In Server Applications

A server application must typically respond to many incoming connections in a concurrent manner. Concurrent handling of connections can be achieved by several different methods:

- Within a single-threaded application by using either `select()` or `$QIO()`
- Multithreaded application using pthreads
- Multiple processes, which may require additional interprocess communication (IPC)

The major difference between `select()` and `$QIO` is that `select()` will not return until one of its socket descriptors is ready for I/O. When `select()` returns, the application must poll each descriptor to determine which one caused it to return. In contrast, an asynchronous `$QIO()` will return immediately after having queued an AST routine that is called back when data is available. The argument passed to the AST routine uniquely describes the connection, so there is no need to perform further polling.

To measure the order of program complexity, consider an application with *"n"* connections. Because an algorithm using `select()` must poll each descriptor, this results in an algorithm with a complexity of *O(n)*. For asynchronous `$QIO()`, the argument passed to the AST routine describes the channel that has become ready. Hence, the `$QIO()` algorithm has a complexity of *O(1)*, which is far more efficient when *"n"* is large. Also, because the AST interrupts the mainline of processing, the `$QIO()` solution provides two code paths within the process (an AST code path and a process priority code path), although only one of these code paths will be active at a time.

In high-performance applications that handle many connections, consider using a multithreaded implementation. There are many options for designing a multithreaded solution. The choice of approach depends on many factors, including such concerns as overhead of process creation, program complexity, and type of service. For a discussion on choice of concurrency, see [Comer, 2001].

Separate processes operate similarly to a multithreaded application, except that in a multithreaded application, all threads share the same address space. This allows each thread to access global data directly, usually with the aid of mutex locks. Separate processes each have their private address space; so if there is a need to share data between the processes, a separate IPC mechanism must be implemented.

If the same TCP listen socket descriptor is used by multiple threads, OpenVMS will deliver new incoming requests to each of the listening threads in a round-robin fashion. Separate processes

© Copyright 2004 Hewlett-Packard Development Company, L.P.

may also share the same socket descriptor, provided it has first set the SO_SHARE socket option.

Conservation of Server Resources
Typically, a host provides a multitude of disparate services to many clients, with each service competing for system resources. Conserving server resources improves scalability and robustness. This is especially important in environments where the server may be exposed to attacks from poorly written clients or malicious clients.

Simplifying Address Conversion
Legacy IPv4 applications use a variety of BSD API functions to convert between a presentation form of an IP address and its binary format. Because a presentation form of an IP address may be in either *dotted-decimal* or *hostname* format, an application should first try the *dotted-decimal* form, and, if that fails, try to resolve the hostname, (RFC 1123, section 2.1 [Braden, 1989b]). Typical APIs include: `inet_addr(), inet_ntop(), inet_pton(), gethostbyname(), and gethostbyaddr()`.

With the introduction of IPv6, the various forms of IP addresses grew and the legacy APIs proved to be inadequate. Consequently, these APIs have been superceded by new and more powerful protocol-independent APIs. They are `getaddrinfo(), getnameinfo()`, and `freeaddrinfo()` (RFC 3493, section 6.1 [Gilligan et. al., 2003]). A new protocol-indendent structure used to describe socket addresses is `struct socket_storage.` RFC 3493 also describes `inet_pton()` and `inet_ntop()`, but these may also be replaced with `getaddrinfo()` and `getnameinfo()`, respectively.

An additional benefit of the `getaddrinfo()` function is that it returns an initialized socket address structure and other fields (embedded in `struct addrinfo`) that may be used directly in the `socket()` and `bind()` calls. This simplifies the code and makes it independent of the differences between IPv4 and IPv6 addresses. For example, a server application that is willing to accept connections from UDP/IP, TCP/IP, UDP/IPv6, and TCP/IPv6 might use the code in Example 22.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
int sd[MAX_SOCKS]; /* one per address for: UDP/IP, UDP/IPv6, TCP/IP, TCP/IPv6 */
char *port, *addr = NULL;
struct addrinfo *res, hints;

port = argv[1]; /* port number as a string – must not be NULL */
if(argc == 3) addfrr = argv[2]; /* hostname – NULL implies ANY address */

memset(&hints, '\0', sizeof(hints));
hints.ai_flags = AI_PASSIVE; /* if usrreq.addr NULL, sets sockaddr to ANY */

err = getaddrinfo(usrreq.addr, usrreq.port, &hints, &res);
if(err) {
   if(err == EAI_SYSTEM) perror("getaddrinfo");
   else printf("getaddrinfo error %d - %s", err, gai_strerror(err));
   return 1;
}

i = 0;
for(aip = res; aip; aip = aip->ai_next) {
   if(aip->ai_family != AF_INET && aip->ai_family != AF_INET6) continue;
   /* create a socket for this protocol */
   sd[i] = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
   if(sd[i] < 0) {perror("socket"); return sd[i];}

   err = socket_options(sd[i], aip); /* set SO_REUSEADDR, SO_REUSPORT etc. */
   if(err) {perror("sockopt"); return 1;}
         err = bind(sd[i], res->ai_addr, res->ai_addrlen);
   if(err == -1) {perror("bind"); return 1;}

   if(i == NUM_ELT(sd)) {printf("Insufficient socket elements\n"); break;}
   i++;
}
freeaddrinfo(res);

/*
** TCP sockets prepare to accept incoming connections.
** UDP sockets are ready for data transfer.
*/
```

Example 22 Use of `getaddrinfo()` to Retrieve Address List

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Summary

A network programmer is responsible for compensating for any unforeseen events that may affect the successful transfer of data, with the least impact to resources and performance of the systems involved.

To take advantage of a multihomed environment, you must be careful to use a method that ensures that no data is lost; even if a NIC fails during the transaction.

In the current environment of mixed IPv4 and IPv6 implementations, it makes sense to use the APIs that guarantee communication in either world.

With a solid basis of understanding about the way TCP/IP networking operates, you can ensure that your applications make the best use of the available resources in any type of environment.

This article has described a broad variety of best practices for the TCP/IP programmer in terms of the structure of a network program which was described in four phases: establish local context, connection establishment, data transfer, connection shutdown.

Establishing local context deals with selecting the appropriate protocol, creating and naming endpoints and preparing the socket for various functions depending on whether it is a client or server.

The connection establishment phase is different for TCP and UDP, where TCP connection establishment is controlled by a state machine and peers undergo a protocol exchange. A UDP connection affects local context only and merely associates the peer's address with the socket. Before a client attempts to connect to a server, it must resolve the server's hostname using the modern API, `getaddrinfo()`. The impact of a server application not being able to keep up with connection requests was also discussed. Once connected, the peers should monitor the connection with keepalive polls. Where TCP provides a keepalive mechanism, UDP leaves this up to the responsibility of the application.

Data transfer for TCP, a stream socket type, is often misunderstood. It is emphasized that TCP guarantees to deliver no more than one byte at a time and it is up to the receiving application to assemble these bytes into messages. On the other hand, UDP uses a datagram socket type which delivers datagrams in the same way they were sent. However, UDP was designed to be inherently unreliable and simple, so it is the repsonsibiltiy of the UDP application to provide for UDP behaviors.

Shutting down a connection for TCP applications should be done using the orderly release method, where the send side is shut down first, which notifies the peer that no more data will be sent. It is important to realize that the peer that initiates the shut down is forced to close the connection through the TCP TIME_WAIT state.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

For More Information

As well as the specific references described below, there are many web sites, newsgroups, and FAQs dedicated to TCP/IP programming. Web-based search engines, such as http://www.google.com, provide a critical tool for locating information for the TCP/IP programmer.

Braden R. T., ed. 1989a., "Requirements for Internet Hosts -- Communication Layers", RFC 1122, (Oct.)

Braden R. T., ed. 1989b., "Requirements for Internet Hosts -- Application and Support", RFC 1123, (Oct.)

Comer D. E., Stevens, D. L., 2001. *Internetworking with TCP/IP Vol III: Client-Server Programming And Applications Linux/POSIX Sockets Version.* Prentice Hall, New Jersey.

Gilligan, R., Thomson, S., Bound, J., McCann, J., and Stevens, W. 2003. "Basic Socket Interface Extensions for IPv6", RFC 3493 (Feb).

Hewlett-Packard Company, 2001. *Compaq TCP/IP Services for OpenVMS, Sockets API and System Services Programming.* (Jan.)

Hewlett-Packard Company, 2003a. *HP TCP/IP Services for OpenVMS, Guide to IPv6.* (Sep.)

Hewlett-Packard Company, 2003b. *HP TCP/IP Services for OpenVMS, Management.* (Sep.)

Hewlett-Packard Company, 2003c. *HP TCP/IP Services for OpenVMS, Tuning and Troubleshooting*, (Sep.).

Muggeridge, M. J., 2003. *Configuring TCP/IP for High Availability.* OpenVMS Technical Journal V2.

Snader, J. C., 2000. *Effective TCP/IP Programming: 44 Tips to Improve Your Network Programs.* Addison Wesley, Reading, Mass.

Stevens, W., Thomas, M., Nordmark, E., and Jinmei, T. 2003. "Advanced Socket Application Program Interface (API) for IPv6", RFC 3542 (May).

Stevens, W. R., 1994. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Reading, Mass.

Wright, G.R., and Stevens, W. R., 1995. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison Wesley, Reading, Mass.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

TimeLine-Driven Collaboration with "T4 & Friends": A Time-saving Approach to OpenVMS Performance

Steve Lieman **steve.lieman@hp.com**, Performance Group, OpenVMS Engineering, Nashua, NH, USA

This OpenVMS Technical Journal article gives you the background story on TimeLine Collaboration (**TLC)**, sketches the universal ingredients of a timeline-driven approach to OpenVMS performance, highlights the original and the current capabilities of the "upstream" and "downstream" **Friends of T4**, and details the central role played by growing reservoirs of TimeLine Collaboration format data.

We provide many concrete examples of the ways in which the graphical representations of timeline performance data can help solve a variety of performance concerns by extracting the meaning and value from the collected data. Numerous examples drawn from the OpenVMS GS1280 Proof Point Project (P3) have been included. We will show how you can benefit by getting started along a similar timeline-driven path beginning with T4 data collection and how you can customize this to meet your specific needs. The timeline-driven path is one that promotes collaboration by means of the efficient collection, sharing and exchange of vital performance data.

So Much Data, So Little Time, So Much at Stake.

It began three years ago with the creation of a timeline data extraction tool named **T4** (Total Timeline Tracking Tool). **T4** converted previously recorded MONITOR data into a reusable text file in Comma Separated Value (**CSV**) format. Since then, HP OpenVMS Engineering has been evolving, improving, and extending the value of T4. For example, we created a T4 kit that added vital timeline data from other independent sources to the generated MONITOR data. We continued by developing an interconnected series of timeline-driven tools, techniques, and tables that help extract the maximum value from the timeline data. These included features for automating the creation of detailed timeline history, for synchronizing performance data captured from independent sources, for readily adding new timeline collector sources, and for rapidly visualizing and reporting on timeline behavior.

This growing collection of cooperative capabilities has proven to be universal in scope and readily extendable while fostering and encouraging a collaborative approach to any performance situation. These developments, now codenamed **T4 & Friends**, have produced visible productivity improvements and dramatic time-saving for OpenVMS Engineering's performance efforts including our extensive cooperative performance work with customers and partners. T4 & Friends is all about the time efficient creation and use of timelines including: their capture, charting, synchronization, comparison, visualization, sharing and especially collaboration.

TimeLine Collaboration (TLC) Format

The **T4 & Friends** approach has, as its central core, a single, common, uncomplicated, universal format that we have chosen for representing the timeline statistical values that have been collected or extracted. We refer to this as **T**ime**L**ine **C**ollaboration (**TLC**) format. The T4 extractor for MONITOR data generates TLC output. In addition to its extractor for MONITOR data, the T4V3x kit now includes five other collectors and extractors. Each generates output in TLC-format.

Better yet, any upstream collector or extractor can do the same and readily store its vital timeline data in TLC-format. These will be the upstream "**Friends of T4".** There are also downstream "**Friends of T4"** that take TLC-format data and carry out value-adding actions such as synchronizing data from independent sources, visualizing timeline changes, comparing data from two different time periods, applying expert rules, or graphically reporting results.

Widening Use Among the OpenVMS Community

Not surprisingly, a growing number of OpenVMS customers and partners have seen these benefits first hand and have begun to follow our lead. They have done this by turning on their own historical timeline data collection, by generating data in TLC-format (beginning with the base T4V3x kit), by turning on their own new collectors or extractors that generate TLC-format data, and by structuring their own unique timeline-driven approach.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Components of our **T4 & Friends** developments and growing banks of historical data in the core **TLC-fo**rmat are now routinely employed on some of the largest, most important OpenVMS production systems around the world including vital systems receiving our highest Gold and Platinum Support Levels.

Access to the T4V3x Kits

The latest T4V33 kit that supports creation of historical timeline data now ships with OpenVMS Alpha Version 7.3-2 making this essential foundation block for collaborative, timeline-driven performance work more readily and widely accessible in 2004 to OpenVMS customers. The T4V32 kit is available for public web download for those running earlier versions of OpenVMS on AlphaServer systems. For many, the easiest way to obtain T4's capabilities is to use HP Services' **System Health Check** (**SHC**) offering for OpenVMS. SHC now includes a T4-driven collection of TLC-format data coupled with a growing list of expert rules that assess system performance health based on that data.

The Background Story: Timeline-Driven Performance Work

Timelines Are Key. Timelines Apply Universally.

While this has not always been the case, timelines are now at the center of most performance work we undertake today within OpenVMS Engineering. This applies to our internal efforts as well as our many interactions with customers and partners. These include:

- Tuning
- Stress testing
- Benchmarking runs
- System sizing estimates
- Checking for unexpected side effects
- Spare capacity evaluation and estimation
- Troubleshooting and bottleneck identification
- Dealing with reported cases of performance anomalies
- Validating that recommended changes really made a difference
- Estimating the headroom added by the newest models of hardware
- Characterizing the relative performance behavior of new versions of software

Timeline Charts Have Explanatory Power.

One of the most important ways that a timeline-driven approach improves any performance project is that timeline data are naturally and inherently graphical. It is always possible to make your performance findings visible to yourself (for analysis). It then becomes possible, post-analysis, to select a small set of key visual outputs and craft a **visual performance explanation** to share with others. This is especially powerful when the analyst can sit side-by-side with one or more interested parties and explain a carefully selected visual timeline while pointing to the natural features of the graphic and interacting with the other viewers. The interested parties in these cases could just as easily be technical or non-technical, because visual explanation has been proven to work well with both audiences.

NOTE

The graphs shown in the examples and the conclusions presented as to what they mean come from a thorough analysis of a much larger set of data and the observation of the timeline behavior of many many variables. The charts shown and the conclusions don't stand on their own. We fully expect that different analysts looking at the same timeline data might come to a somewhat different set of conclusions. The beauty of the approach presented in this document is that the timeline data so gathered and saved in timeline history data banks is readily reusable and is intended for collaborative use. It serves as the core reference point on which to build, discuss, and defend a set of hypotheses that help explain the patterns observed in the data. Our assumption is that that we always save the underlying core timeline data so we can return to it in case any question arises or if we decide to re-analyze the situation based on new information that has come our way.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Figure 1 gives a simple example of the potential explanatory power of timeline graphics.

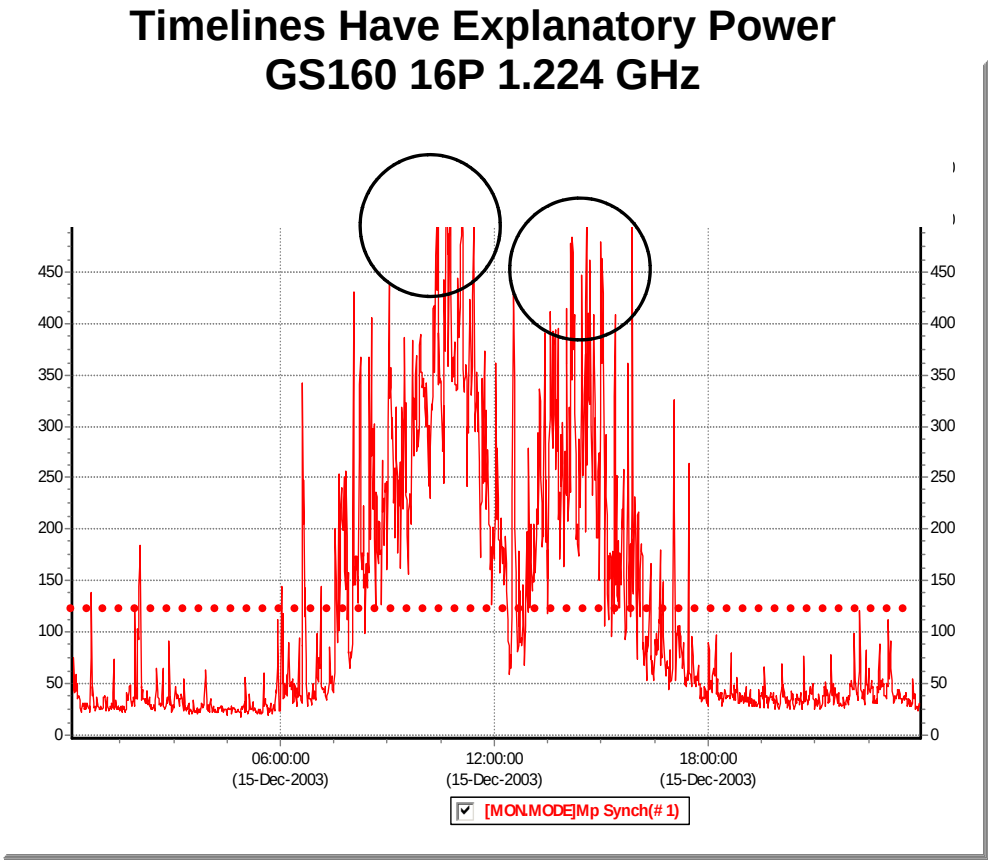

Figure 1 – Timelines Have Explanatory Power

MPsynch on a large GS160 system with 16 CPUs. Note the circled morning and afternoon peak periods and the lunchtime lull. A timeline graph such as this makes it easy to zoom in and focus your attention on the peak periods. The average, as shown by the dotted line, is about 120%. Use of the average would of course be very misleading for a metric that behaves in this fashion.

Timelines Are Everywhere. Timelines Are Us. Timelines Foster Collaboration.

Almost every stakeholder (and everyone we might need to communicate with) is already familiar and can easily grasp timeline graphics. Most of us have already seen innumerable stock market price charts and other examples of timeline graphics. Visual timeline processing is a powerful built-in human skill that requires virtually no training to tap.

Within OpenVMS Engineering, we have time and again found that the more we use timelines for our performance work, the better, faster, and easier it becomes for us and for everyone else involved to understand a given performance issue.

Timelines Are Old Hat, But…

Of course, the importance of timelines and their central role in conducting effective system performance work has been known for many years. It's nothing new.

The reasons for widespread timeline use are far ranging: Real systems are complex. The workload mix on live production systems can change radically in the course of a few minutes. Here are a few questions, answers, and example graphics to clarify this point.

QUESTION: How can you identify when those changes occur or which mix is in play when a bottleneck appears?

ANSWER: Timelines graphics can help bring this story and the necessary distinctions into sharp focus.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Figure 2 gives a simple example how a timeline graph can reveal changes in mix.

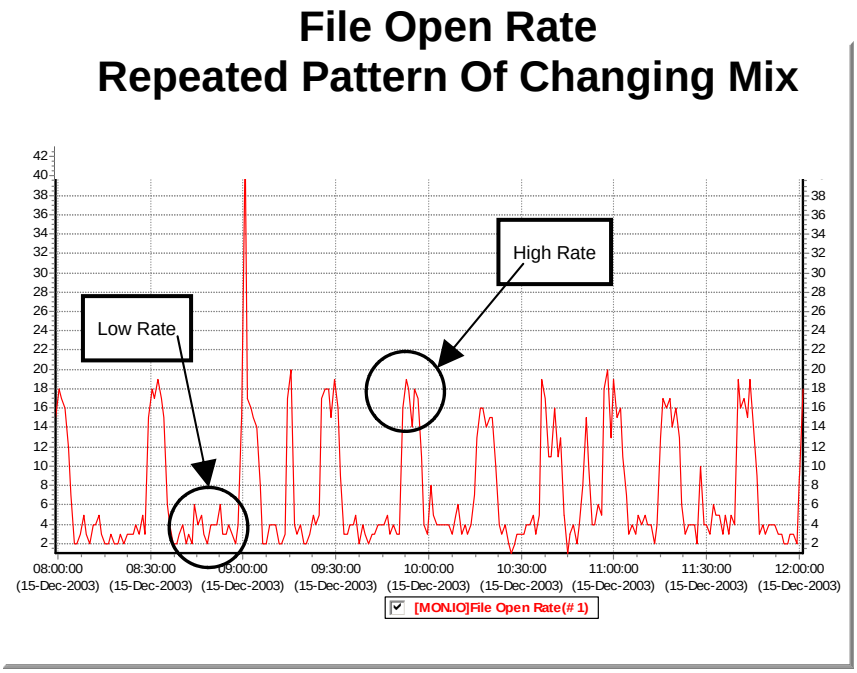

Figure 2 - Change in Mix

From the same large GS160 system, we have zoomed in on the period between 8 AM and Noon and focused on the metric for File Open Rate.

Note the repeated, somewhat regular way that the mix of work changes between a low rate and high rate. During the same period, other key variables such as Buffered I/O rate showed a much reduced range of variation in behavior.

QUESTION: Resource bottlenecks may last only for a few minutes at a time. How can you find those periods and zoom in for further analysis? How will you know which resources are most related to system slowdowns?

ANSWER: Timelines literally let you see when the peak periods begin and end for every measured resource. By comparing those peaks for the resource to the periods when slowdowns were noticed, you can detect whether there is a plausible relationship between the observed peak and the slowdown.

Figure 3 shows both the way in which the graphics reveal trends in the data and the specific way you can use a graph to identify the most important periods for further, detailed inspection. Identifying the peak period is essential if we want to avoid the danger of being misled by average values.

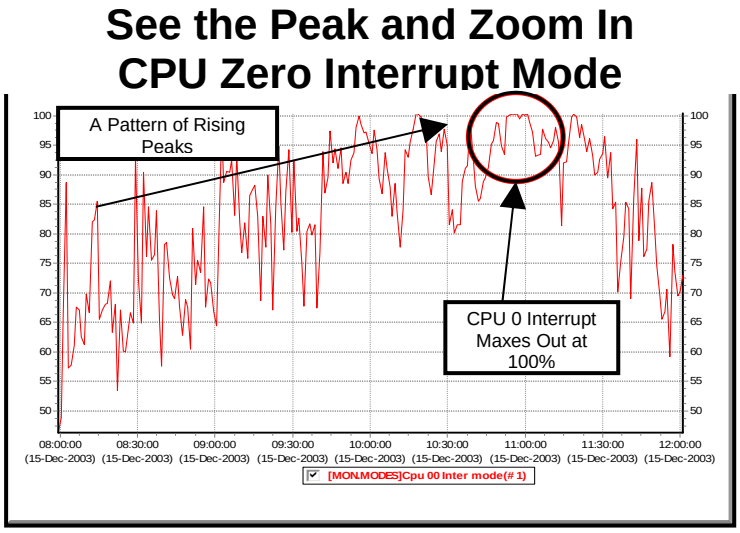

Figure 3 – See the Peak and Zoom in

Watching CPU zero interrupt use during the period from 8 AM to Noon, we can readily see the peaks getting higher and higher until just before 10 AM when we hit 100% for the first time. This vital metric maxes out for a sustained period around 11 AM and has dramatic impact on other key variables as shown in the next chart.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Other types of charts can be constructed using timeline data as a base. Scatter plots sometimes prove invaluable for highlighting patterns that may not be obvious from the time series view. Figure 4 is a simple example of the ways in which scatter plots sometimes bring hidden patterns into view.

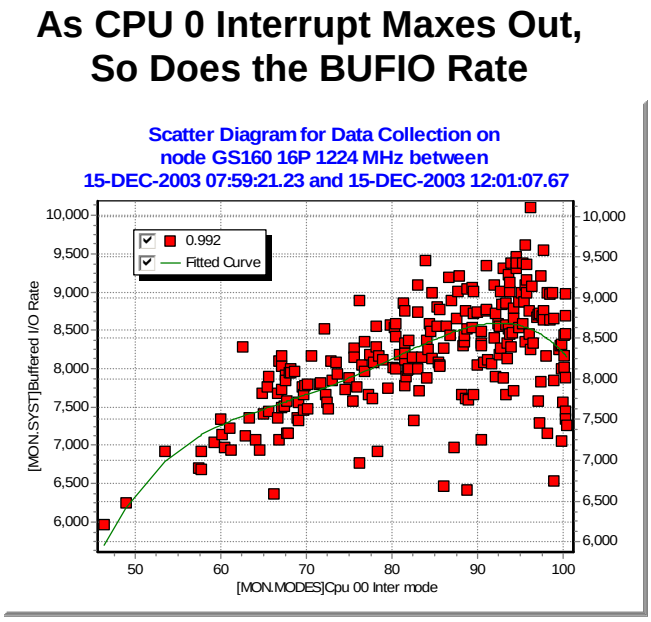

Figure 4 – As CPU 0 Interrupt Maxes out, So Does the BUFIO Rate

This chart is a scatter plot for CPU 0 Interrupt (X-axis) compared to the BUFIO rate (Y-axis) for the period from 8 AM to Noon. Some non-linear effects can be read from this graph. As CPU 0 Interrupt gets closer and closer to 100%, each increment is capable of driving fewer and fewer added Buffered I/Os.

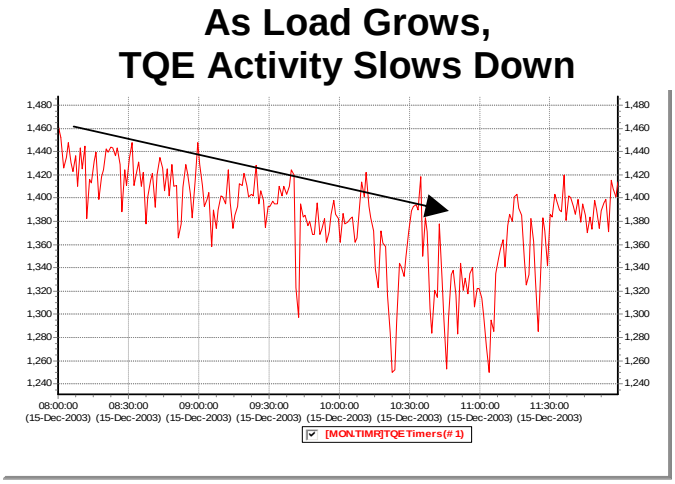

Figure 5 -- As Load Grows, TQE Activity Slows Down

Side effects: As Interrupt load on CPU 0 and Buffered I/O goes up, we notice that in the same period, TQE activity actually slows down. It's quite likely [but not certain] that the actual demand on TQE services actually increased during this same period. If so, any users dependent on TQE processing would have experienced increased response time while those busy doing Buffered I/O monopolized scarce (zero sum) resources such as CPU Zero interrupt.

QUESTION: Something that improves performance for one class of users may have the side effect of hurting another class of users. How can you clearly see both sides of the story?

ANSWER: Multi-dimensional timelines can often help bring this picture of side effects into clear focus. Figure 5 above draws from the same data as the previous examples and shows that Timer Queue Entry (TQE) activities appear to slow down as the total system load increases.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Figure 6 shows the scatter plot of for TQE under the influence of rising CPU busy. In this case, we see both a reduction in TQE activity and a non-linearity in the shape of the curve.

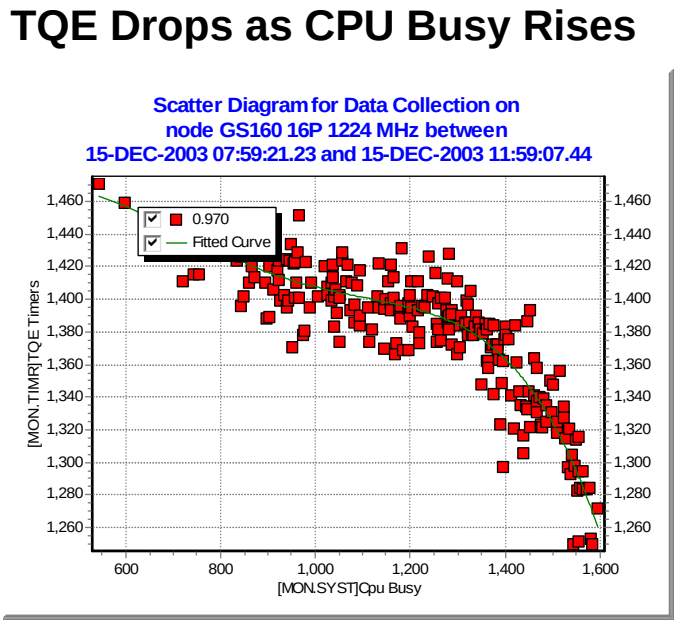

Figure 6 -- TQE Drops as CPU Busy Rises

Another way to look at the side effects equation is with scatter plots. The scatter plot of TQE rate (Y-Axis) vs. CPU Busy (X-Axis) shows a clear and steady reduction in TQE activity as CPU Busy approaches its maximum of 1600% for this system.

Seven Limiting Factors

Historically, as OpenVMS has evolved over the past twenty-five+ years, many expert performance analysts have used timeline-driven approaches with excellent effect. A number of tools and capabilities have grown up to help support this work. Some analysts have made fine use of the timeline capabilities that are built into existing tools. Others have harvested timeline data in their own unique ways and rolled their own downstream tools for manipulating, graphing and reporting with good results.

While there is nothing new in saying that timelines are essential to success, in OpenVMS Engineering, we found that our desire to conduct timeline-driven, collaborative performance work with our customers was impeded by seven key factors.

Factor 1. There was substantial risk that vital timeline performance data would not automatically be collected on customer systems before a performance issue arose. Collection would start only after some problem or issue appeared and the time to solution was systematically delayed.

Factor 2. The best of the existing tools for timeline capture, analysis, and reporting were frequently unavailable on the system in question or for use by those who most needed them.

Factor 3. Where the timeline tools were available and where timeline histories were captured, it was still common to discover severe productivity limits and a high time cost of using these expert tools effectively. In these cases, there was much we might have done for analysis or reporting that was left undone or incomplete due to real world time and cost constraints on these activities.

Factor 4. There was a substantial startup cost to learn to take advantage of the complexity of the best of the existing timeline tools. Effective use for analysis and reporting typically required achieving a rather high level of expertise. For many systems that had the tools, their local ability to do first level analysis was severely impeded by these expertise constraints.

Factor 5. To a large degree, the best of the existing tools imposed limitations on analysis and reporting to those fixed and unchanging capabilities that had been designed into the tools in the first place. Only those willing and able to program new capabilities on top of the existing set were able to invent new approaches.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Factor 6. The data created from one set of collectors did not play well with data from other important collectors. This incompatibility substantially limited sharing of data and extendibility of methods.

Factor 7. There appeared to be several ways in which the existing timeline tool set was still wed to 1980's technologies and mindset. The 1980's was a period when memory, disk, and CPU power was rather more scarce and expensive than today. It was also a time when human expertise was both more abundant and less costly.

Abundance & Scarcity. Computing Power is Abundant. Time is Scarce.

We noted especially how existing built-in and handcrafted timeline capabilities failed to take full advantage of the current abundant desktop and mobile computing power. In our new century, it is nearly universally true that desktop and laptop personal computing power is overflowing. Meanwhile, the scarcest resource for performance analyst work today always seems to be the analyst's personal time. System managers and performance engineers have less time to handle more complexity on an ever-increasing number of systems.

Improving Our Timeline-Driven Work. Improving How We Use Our Own Personal Time.

Over the past three years, OpenVMS Engineering has been looking for new and improved ways to make timelines even more useful in our own performance work. We were especially interested in the idea of being able to use any new approaches wherever and whenever we needed them on all customer and all internal OpenVMS systems. We needed universal tools and methods.

We also had a keen interest for developing new approaches that would make our work more and more productive, that would take full advantage of the abundance of desktop computing power, and that would compensate as best as possible for the scarcity of our own time. Consequently, we have developed and evolved our timeline-driven approaches with these two simultaneous goals in mind:

1. Be highly sensitive to questions of an analyst's use of their scarce time.
2. Freely use abundant resources to save time.

Not wanting to re-invent the wheel, we have also kept our eyes open for other universal timeline tools and techniques that might work cooperatively and collaboratively together and that also kept the scarcity of analyst time clearly in mind. The T4 & Friends path we have taken stands squarely on the shoulders of those who have come before.

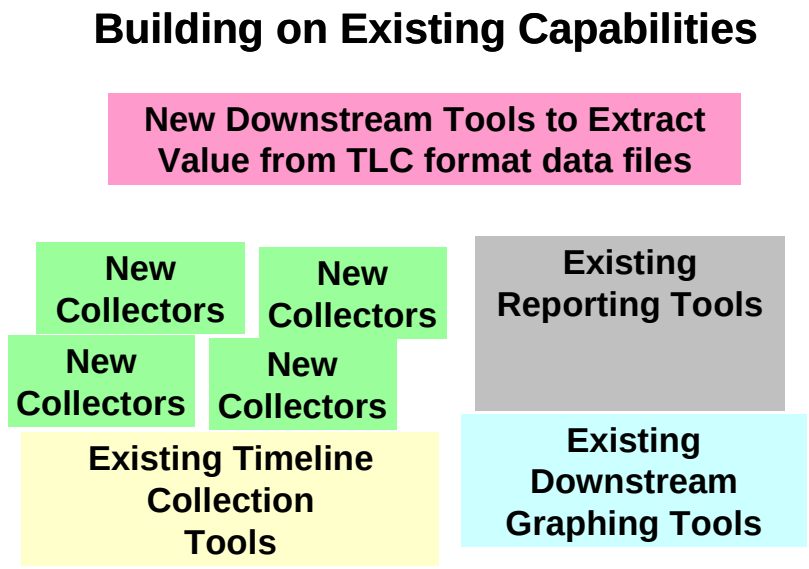

Figure 7 -- Building on Existing Capabilities

Many of the building blocks for timeline collaboration are already in place, readily available, and easy to get started. Because of the universal TLC-Format data, new capabilities can be added as needed with incremental investments.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

If a Tree Falls in the Forest …

Just as there are many different types of trees in a mature forest, important timeline data of many different kinds exist for the taking on all systems. It is the raw material needed to carry out any performance assignment. Response time and throughput metrics for essential business transactions represent an especially important type of data that's potentially collectable. When this kind of data is available, it almost always plays a fundamental and primary role in any performance work that follows.

Missing Data

Unfortunately, the natural state of affairs is that response data and other key timeline data that can help you most with your performance concerns is available only for a fleeting instant. Some of the most vital classes of data are never even collected. In many cases, crucial events transpire and essential information about those events is not even counted.

Private Formats

In other cases, timeline data is saved and stored in some complex internal or private format and then accessible only in a restrictive fashion using a non-extendable and often time-consuming set of methods. For example, MONITOR is a great tool for capturing a wide swath of important timeline data and saving it in its own internal format as a MONITOR.DAT file. Unfortunately, MONITOR's built-in tools for downstream processing of the timeline patterns captured in these DAT files are limited.

Over Reliance on Averages

Another problem can arise even when the timeline data is dutifully collected. If the only use of the data is to roll it into long-term averages, the underlying time-dependent complexity will be hidden forever. These averages or the total counts since the system was last booted may be readily available, but how that important quantity changed over time is lost. Without detailed timeline data taken at an appropriate resolution, you will never know when peaks or valleys occurred, how long they lasted, how high they reached or how low they fell, or how steady or erratic the overall behavior has been. And you will never be able to examine which factors moved together during the peaks – a known key to unraveling and revealing possible cause and effect relationships.

The Incredible Disappearing Timeline

We commonly see many collector tools that capture essential timeline data, display it to a screen in text format, or sometimes even use a striking graphical format, and then destroy earlier timeline data as each new sample arrives. It's what I call "The Incredible Disappearing Timeline." This screen data can be exceedingly handy for anyone who is watching in real time. However, a few seconds or a few minutes later that screen data is lost forever. Perhaps some key facts are gleaned and remembered by those individuals who are watching intensely, or a screen print is captured of a particularly revealing sample. Sadly, if you walk away and something important happens, you will simply miss it. Worse, even when we literally tie ourselves to the console, we can monitor only a small number of items at a time – typically between about 3 and 7 items for ordinary mortals. If something interesting happens in a vital statistic we are not fortunate enough to be watching (it could even be on the screen), we are simply out of luck. Some clumsy workarounds to this problem do exist. For example, you might automatically write/log the scrolled screen display to an output text file. Later on, you can review that text output. If you find yourself doing this all the time, you will likely be tempted to write a program that parses the text and extracts the timeline information you need.

This default state of affairs for carrying on a timeline-driven approach to performance work leaves a lot to be desired. There is a timeline analogue to the question:

"If a tree falls in the forest and no one is there to hear it, does it make a sound?"

QUESTION: If vital timeline data from many different independent and important sources is ready for harvesting, and no one is there to catch any of it and save it, can this data ever help you at all in your performance work?"

ANSWER: Maybe the timeline data for each source makes a sound when it falls uncaught, but no one will ever hear of it again.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Down The Drain

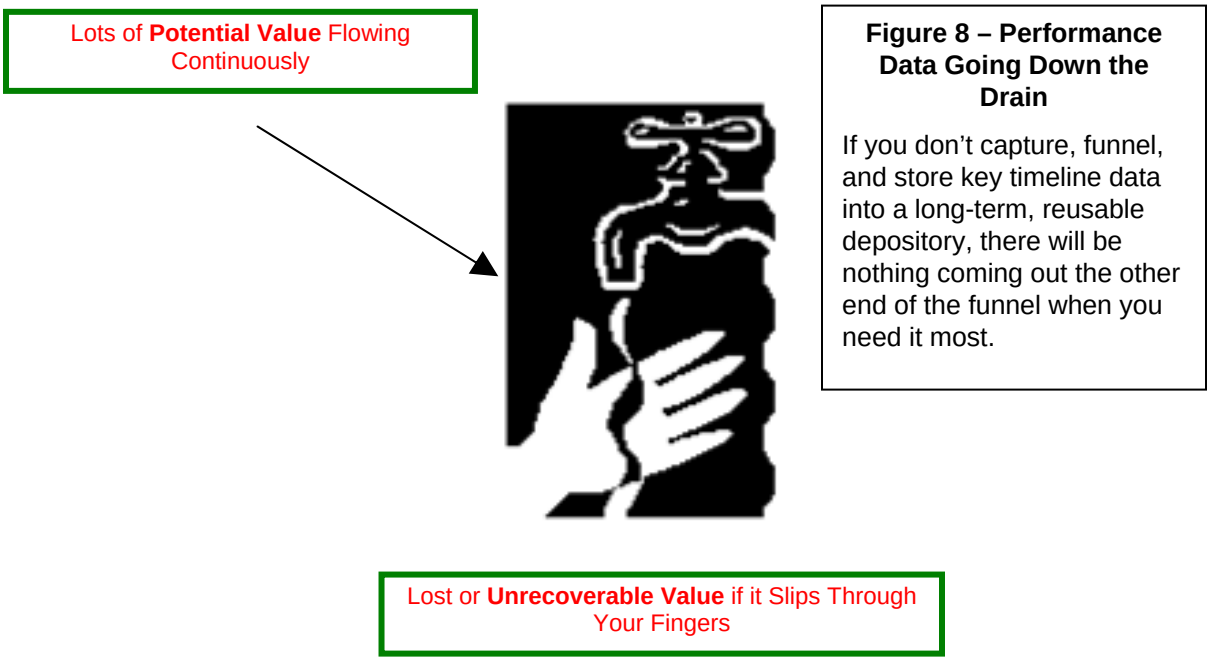

Figure 8 – Performance Data Going Down the Drain

If you don't capture, funnel, and store key timeline data into a long-term, reusable depository, there will be nothing coming out the other end of the funnel when you need it most.

Timeline Data is the Raw Material for All Complex Performance Work.

Unless key timeline data is captured at appropriately detailed resolution, time-stamped, archived for historical reference, and converted to a publicly reusable format, it is likely to be lost forever, subsumed into long term (and often misleading) averages, or only available for experts through rigid, time-consuming, non-extendable interfaces, or clumsy, time-consuming workarounds.

Half of the work with T4 & Friends has centered on solving this set of problems – capturing the raw timeline data and saving it in a readily reusable, sharable format so it's latent potential would be available when needed.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

T4V1: The Original T4 – A Good Beginning is Half the Work

T4 (originally an acronym for **T**om's **T**errific **T**imeline **T**ool) is the name given to an internally developed OpenVMS performance tool. The original T4V1, written in DCL, was an automated **extractor** of the latent timeline data previously hidden within MONITOR.DAT files.

T4V1 converted MONITOR's natural timeline data for about thirty key statistics into a two-dimensional timeline table. Each row represented exactly one sampling period and each column represented exactly one of the key statistics.

The original T4V1 capability then saved the two-dimensional timeline data in a readily reusable Comma Separated Value (CSV) file. In CSV format, the resulting file was then suitable for further analysis and graphing with tools such as Microsoft® Excel (one of the first universally available downstream "**Friends of T4**").

In what follows, we will refer to the CSV files created by T4 collection as **TLC** (**T**ime **L**ine **C**ollaboration) **format files**. Collectors other than T4 can readily create timeline files in **TLC-fo**rmat. Then they can also benefit from the full range of available downstream timeline tools that have been designed to extract the maximum value from this very special kind of data.

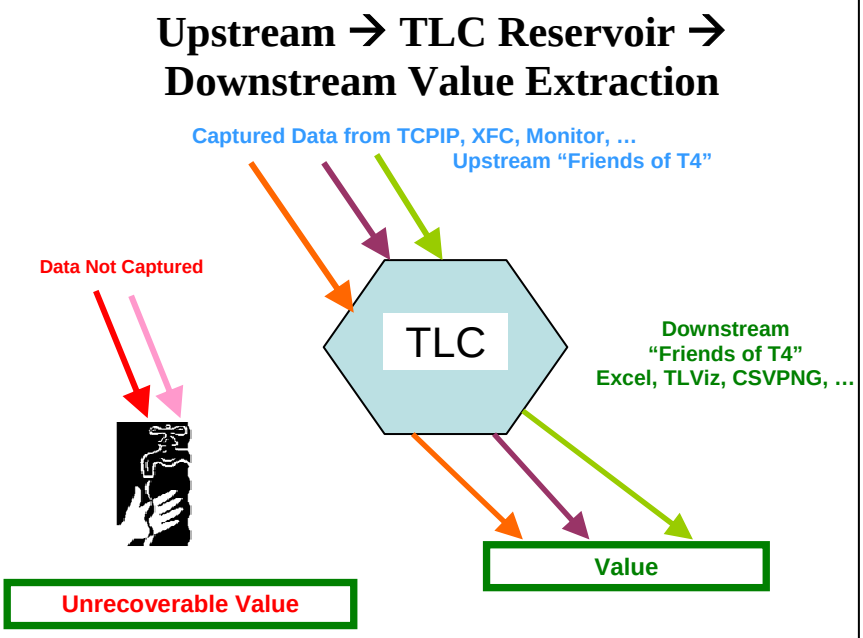

Figure 9 – Creating Value from Timeline Data

Data collected upstream from many sources (for example MONITOR, TCP/IP, ORACLE collectors or extractors) is channeled into a TLC reservoir and deposited for long-term historical storage. Later, selected TLC data is drawn off and fed into downstream tools to extract value. Any number of upstream collectors can be added as they become available. Each such collector further raises the potential value of the TLC data bank. Unfortunately, if key statistics do not have their timeline behavior captured, the potential value of that timeline behavior is unrecoverable. Downstream tools such as Excel, TLViz, and CSVPNG can be used to extract value from TLC historical depositories, as you will see with the many examples that follow.

While what we accomplished with T4V1 and our first use of TLC-format files may sound somewhat simplistic, it nevertheless produced a surprisingly large order of magnitude productivity gain for our performance work within OpenVMS Engineering. Our first use was with some collaborative customer benchmark tests we were conducting. We were comparing the performance of the GS140 to the GS160 on a complex workload that simulated actual load variations and that exhibited intricate time-dependent behavior.

We wanted to compare relative performance of the two systems during relatively short-lived peak periods. We had discovered that the averages over the entire test did not adequately explain the behavior during these peak periods. We knew we had to examine the time-series behavior to fully understand what was happening.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The original T4V1 automated what had been a clumsy and time-consuming manual process. We had previously been forced to use this expensive manual approach to visualize the relationships in time-dependent changes among the most important OpenVMS variables. Unfortunately, because of time constraints, there were many important cases where we did not have the luxury to carry out these expensive manual steps regardless of how much they might have helped.

Because **any** OpenVMS system could record and save periodic sample data in MONITOR.DAT files, once we had the original T4V1 extractor, we could turn that MONITOR data into a universal TLC-format that permitted ready and rapid visualization of that system's time series behavior. Every OpenVMS system could now create timeline data in a universally re-usable format with a relatively modest effort – a big step forward for us.

We fed TimeLine Collaboration (TLC) format CSV data from MONITOR into Excel. Then, we were able to use Excel's graphing capabilities to examine the time-varying nature of thirty important OpenVMS system performance variables. We had pre-selected these 30 variables for a proof-of-concept trial of the T4 and TLC approach.

More Than Just MONITOR Data. Timelines Work With Business Data.

On this same benchmark, we used a combination of DCL, the OpenVMS SEARCH utility, and a customer utility that reported on cumulative application performance and rigged a rudimentary way to capture and save timeline results for customer response time and throughput of key transactions – these were the vital business statistics that the customer was using to evaluate the results of this benchmark project.

By putting these two streams of timeline data together into a single spreadsheet and graphing the relationships between response, throughput, and key system variables, we were able to complete a thorough analysis of the benchmark and its peak period behavior that otherwise would have been unachievable. Figure 10 gives an example of how bringing business and system data together adds to our understanding.

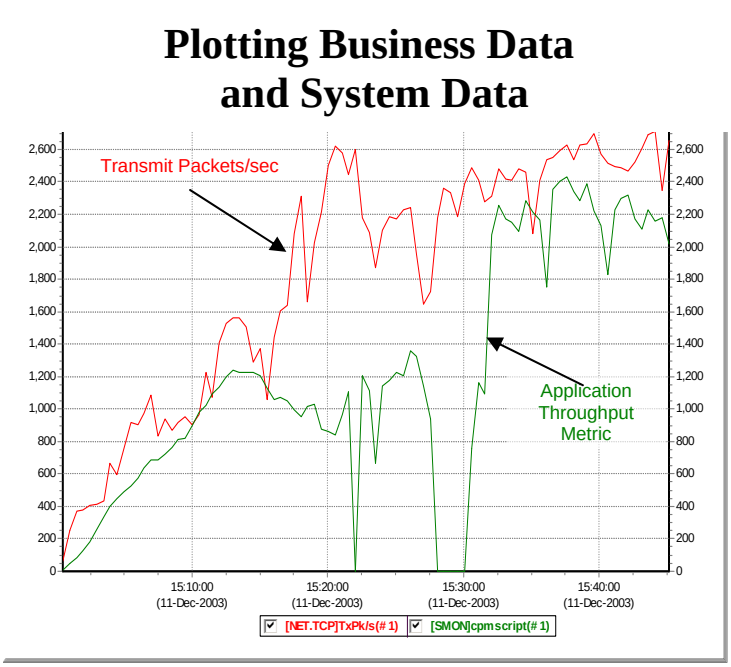

Figure 10 -- Plotting Business Data and System Data

This chart is taken from an intense benchmark. The Red Timeline represents TCP/IP packets transmitted per second. The Green Timeline represents an important customer measure of throughput. Note how they both track each other until about 15:15 and then again after 15:30. However in the 15 minutes in between, throughput for the customer throughput metric falls off sharply and in fact goes to zero on several occasions. A chart like this will tell you (rather loudly) that something very important happened in those 25 minutes. These kinds of graphical results serve as wake-up calls. It's likely you will want to or need to learn more about what really happened so you can explain the drop in throughput. This would be a time to review the other timeline statistics in your TLC data for the same period.

Since that first benchmark and proof-of-concept of T4 and TLC, a lot has happened. What we are now calling "T4 & Friends" has taken on a central role in OpenVMS Engineering's performance work – especially our collaborative efforts with OpenVMS customers and partners. This has been especially so when dealing with complex, live, mission-critical production workloads on the largest OpenVMS systems.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

T4 and Friends includes the many subsequent improvements, extensions and elaborations to the original timeline data extractor and collectors combined with a growing set of downstream capabilities for extracting even more value from the universal TimeLine Collaboration (TLC) files so created.

T4 & Friends today includes many different synchronized collection utilities gathering literally hundreds of columns of important performance data and saving it in a universal TLC-format. T4 & Friends also includes a growing list of capabilities and methods for manipulating, analyzing, learning about, charting and reporting on TLC data contained in ready-to-use CSV files.

With the successful use in this first benchmark project, the Original T4 and the subsequent post-processing with Excel was added as a standard part of our OpenVMS Engineering performance tool kit and found wider and wider use as time went on.

The ability to turn important OpenVMS MONITOR performance data from any OpenVMS system into a visual timeline graphic was one of the key driving forces in the early development of the Original T4.

We have traveled far since these early steps with T4 including the development of highly automated and time-saving approaches for comparing two sets of data in a series of single Before-and-After graphs. Figure 11 is an example of one such chart. We will return to this idea later in more detail, for example in Figures 25 and 26.

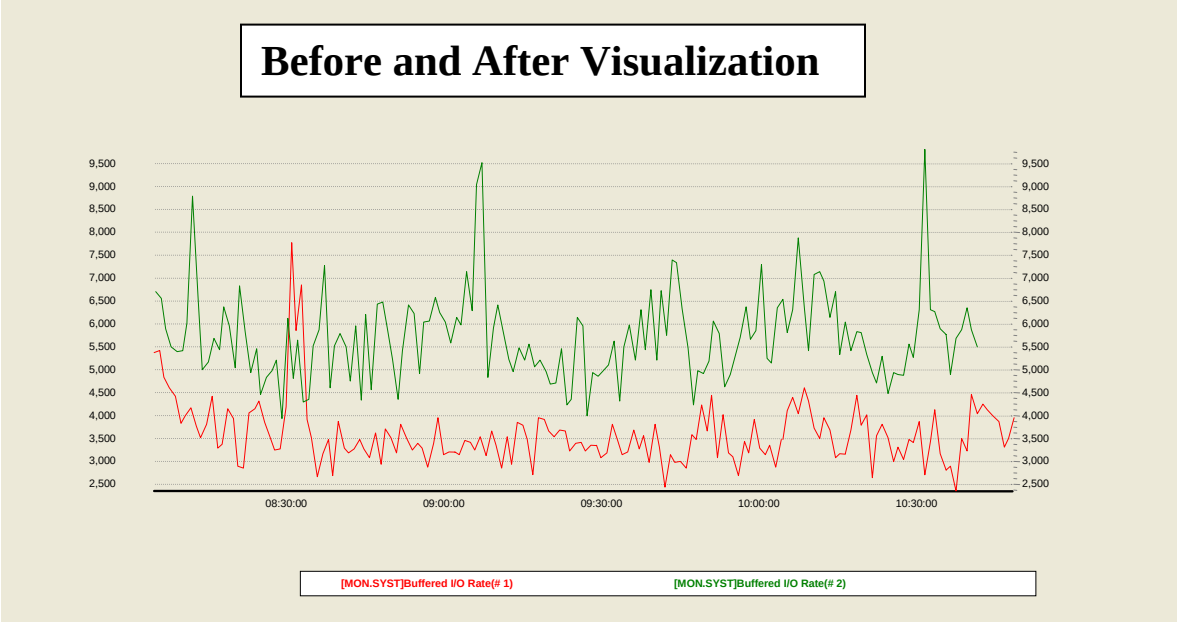

Figure 11- Before-and-After Visualization

In this example, we have mapped timeline data for Buffered I/O (a throughput variable in this particular case) taken from two different runs. This is a classic **before-and-after** comparison. It's one of the many downstream steps you can take once your data is transformed into the readily viewable TLC-Format. Note in this example how the throughput rate for the Green timeline regularly exceeds the Red timeline by approximately 1.5X. Try estimating this yourself using visual comparison and visual math.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Using the T4 Tool Kit to Get Started with TLC

What is in the T4 Tool Kit?

The original T4 extractor has since evolved into version T4V32 and version T4V33 kits. Each kit consists of collectors, extractors, synchronizers and other capabilities. We'll use "T4" or "T4V3x" or "The T4 Kit" to refer collectively to these current versions.

The T4 Kit has now become a mainstay of all OpenVMS Engineering performance work. This includes widespread use as part of collaborative projects with customers and partners on their most important systems. Most recently, the T4 kit proved to be an essential data gathering arm that fed the success of the GS1280 Performance Proof Point (P3) Project (as presented in the November 2003 ENCOMPASS webcast).

Six Collectors – More than a Dozen Views

Over the past three years, T4 has evolved and improved so that it now draws data from **six different independent sources** and literally hundreds of variables, giving us a more complete view of the underlying performance drivers that can impact OpenVMS systems. Each collection source produces its own two-dimensional TLC-format table as output. The six collectors offer more than a dozen separate views of data, each view with its own unique set of individual metrics.

Other enhancements in the T4 kit include automation of historical archiving, automated integration and synchronization of the separate TLC-format CSV files, DCL-driven control programs suitable to individual customization, and the optional ability to Zip and mail resulting TLC data. These added features have continued to make the T4 kit an ever more useful productivity enhancer and time saver for anyone interested in OpenVMS performance.

Because of the straightforward structure of the DCL code for launching and managing six collectors, new collectors can be added and synchronized quite readily as they become available.

Accessing the T4 Kit.

The T4V32 kit is now publicly available for download from the web at

http://h71000.www7.hp.com/OpenVMS/products/t4/index.html

The T4V33 kit is now shipping automatically in the **SYS$ETC** directory in OpenVMS Alpha Version 7.3-2. Both T4V32 and T4V33 are suitable for AlphaServer systems running Version 7.2-2 or higher.

For earlier versions on Alpha and for use on VAX, the T4V2A version (written in DCL) is available from the OpenVMS freeware CD at:

http://h71000.www7.hp.com/freeware/freeware50/t4/

Versions of T4 collection will also be provided for use on HP OpenVMS Industry Standard 64 Evaluation Release Version 8.1 for Integrity Servers.

System Health Check (**SHC**) has added T4-based timeline collection, analysis and reporting to its extensive list of OpenVMS capabilities. SHC is offered by HP Services' Mission Critical and Proactive Services and is widely used on customer OpenVMS systems with Gold and Platinum Support.

The latest SHC version for OpenVMS includes new performance rules based on the TLC data captured by T4. SHC is a suite of assessment tools and services that provide a thorough, broad assessment of customers' computing environment by identifying security, performance, configuration and availability problems before they can impact the customers' critical operations. SHC assessments are executed against sets of best practice system management rules.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Those OpenVMS AlphaServer customers already signed on to the SHC service have the option of using the embedded T4 kit to turn on historical timeline data collection and archiving on their systems. For more information on SHC check out:

http://www.support.compaq.com/svctools/shc/

Increasing T4 Kit Use on Production OpenVMS Systems

With this widening public availability, many OpenVMS customers and partners have taken the T4 kit and begun applying it. Some are using it with default settings to create performance histories for their most important OpenVMS nodes. Others are taking advantage of the kit's natural extendibility and have been customizing it for their own use with excellent personal results. We are starting to receive feedback outlining some of our customers' most useful extension ideas. We plan to consider these for possible inclusion in future versions of the standard T4 collection and historical archiving kit.

Collect First, Ask Questions Later.

There's the old saying from Western Cowboy movies about "'shooting first and asking questions later." We feel the same way about collecting and saving timeline data, about turning on timeline history creation now and deciding later how best to use it, and about how best to leverage the **TLC** (**T**ime**L**ine **C**ollaboration) format data so collected.

Don't Delay.

There are many ways one might create readily reusable TLC historical data for your OpenVMS systems. Pick the one that works best for you. Whatever you decide, we strongly suggest you don't delay in beginning a TLC collection process. For a wide variety of OpenVMS system statistics, the T4 kit is readily available for this purpose and we suggest you consider it as one of your options to get a quick start.

We recommend you collect and save TLC data even if you are not going to look at the data right away or anytime soon, or even if you don't know yet what you want to look for.

We recommend you collect TLC data even if you don't yet have the ideal downstream tools you have dreamed of to post-process this data into graphs and charts and squeeze the maximum value out of it.

We recommend you collect TLC data even if you don't have anyone locally available who is a whiz at Excel or at SQL queries or at writing your own special new utilities that extract value from TLC data by building on top of existing graphical or statistical packages.

Business Statistics Deserve the Very Best Treatment.

We recommend that you also begin to build TLC collectors or extractors for your most important business metrics and add these into the historical mix. In many cases, these statistics are the ones that are most important to your success. They deserve the very best treatment.

Like other statistics, the timeline behavior of essential business statistics will change over time, will show peaks and valleys and perhaps sudden dramatic changes, repeated cyclical patterns, or unusual erratic behavior.

If you are not already capturing this timeline behavior for your business metrics, the value you might have extracted is unrecoverable. We suggest you find a way to get some kind of timeline capture turned on as soon as is practical for you. Building your own collector that directly generates TLC data would be the most desirable choice if it turns out to be at all possible.

You may already be capturing timeline business data and saving it in a non TLC-format. If, however, this vital data has for all practical purposes proven inaccessible to you due to time, money, complexity,

© Copyright 2004 Hewlett-Packard Development Company, L.P.

access, or expertise, we suggest you consider finding a way to extract a TLC version of that data and integrate it with our other key TLC metrics.

Compatible Use of TLC with Other Timeline Utilities

Some of you may already be collecting timeline history in non-TLC-formats using other OpenVMS utilities. This is great. Don't stop. Continue to use these tools and the timeline data they provide to extract value that benefits your systems.

In addition, please note, that a substantial number of customers and partners are already successfully creating TLC data (using the low-overhead, non-disruptive T4 kit) on systems that are running other performance timeline collection software. The T4 kit's low overhead at its default setting of 60-second sampling means that you can use it virtually anywhere, including in conjunction with these other performance tools.

Once you turn on TLC collection on your most important systems, we believe you will discover that the TLC approach based on T4 & Friends offers you capabilities that enhance and extend any you may already be using – especially in the areas of saving your precious time and letting you look at some key performance variables not otherwise available.

We would enjoy learning about your experiences and impressions as you follow up on some of these ideas. You are welcome to forward your thoughts on TimeLine Collaboration to the author.

An Automatic Payoff of Substantial Size

Once you begin creating TLC histories, you dramatically change your ability to collaborate with and communicate with OpenVMS Engineering and with OpenVMS Ambassadors about the performance issues that are most important to you. While there are some other wonderful performance tools out there, within OpenVMS engineering we have radically diminished their use. Wherever possible, we have switched to TLC based on T4 & Friends for our collaborative efforts with customers and partners.

Wherever possible, we have switched to TLC for our collaborative efforts

The main reasons are the ease of T4 collection, the increasing number of new upstream collectors that extend our view, and the growing power of the downstream Friends of T4. Together, this combination helps us generate more and better TLC data and extract more value from the accumulating reservoir of results. We have found this approach to be an order of magnitude more efficient for us in our personal time use. We think you will find similar savings to be true for you once you get started.

The TLC-based model is open-ended and readily extendable, as you will learn below. This will simplify synchronization with your most important business data and with data from other collectors and extractors.

For all these reasons, when it comes to timeline data we recommend that you "Collect first and ask questions later". You won't be sorry.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

What is TLC-format Data?

Two-Dimensional TimeLine Collaboration Tables

TLC-format data is simply a two-dimensional table of timeline data saved in CSV (Comma Separated Value) format. These files obey a few basic rules. By convention, each row represents **exactly one** time interval; each column represents **exactly one** important performance variable. The first column, known as "Sample Time", contains the date and time at the end of each interval. Think of these files as being in **TLC Normal Form** (**TNF**)

> **WARNING: Not all CSV files are in TNF. For example, files which generate multiple rows of data for a given time period do not satisfy the TNF requirement of exactly one row per sample period. Such files are in CSV format, but cannot be readily graphed by tools such as Excel. The timeline data is there, but further time-consuming programming or manipulation is required to make it useful. If you want your timeline data to be readily and immediately re-usable by downstream Friends of TLC such as Excel, you must make sure that it obeys the basic rules above.**

For historical reasons, many TLC-format files have a total of four header lines with the important fourth line being the column header. In these files, the first line contains comment information in a CSV format. The second line contains only the calendar date at the start of sampling. The third line contains only the time of day for the first sample. Future versions may loosen these rules about headers and make them more universal – in particular by making the second and third lines optional. If you are going to create new TLC-format files, consider sticking with the original T4-style format for now, as all of the downstream friends of TLC-format data are then fully available to you.

Sample Time	Variable 1	Variable 2	Variable 3	Variable 4	…
10:21	37	58	107	19	
10:22	44	51	128	12	
10:23	29	74	103	25	

TLC Normal Form (TNF) - A fragment of a TLC-format two-dimensional table

A Standard, Widely Accepted, Universal Output Format

The widely used CSV file format is the current output standard for TLC-format two-dimensional tables. What this means is that a wide range of existing tools ranging from spreadsheets to databases have **built-in capabilities** for reading TLC-format files automatically. This opens the door for the full power of these already available tools to be applied to any TLC-format data. This can be extremely handy if you or someone on your team happens to be a whiz with Excel or an SQL giant.

Conversion to or from other useful two-dimensional timeline formats is also readily possible with minimal programming. The two-dimensional underlying format is a completely universal way to represent any and all timeline data from any source.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Open-Ended, Synchronizable Data

The standard CSV format means that data from other important timeline sources can be integrated and synchronized whenever it becomes available. By definition, each TLC-format CSV file contains internal timestamps that allow the possibility for later synchronizing timeline data from multiple independent sources.

Synchronization is especially helpful in making sense of the most complex performance situations

Our experience within OpenVMS Engineering tells us that synchronization is especially helpful in making sense of the most complex performance situations. The current T4V32 and T4V33 kits contain a utility (APRC – APpend ReCord) and the DCL code to drive it that automatically combines the timeline data from the current set of six independent timeline collectors in the kit while carrying out some rudimentary synchronization steps.

Readily Programmable Data

The easy to read and understand CSV format also means that new tools can be written that load the TLC-format data into memory with zero or minimal programming. Then, precious programming time can be employed in carving out new capabilities and methods. These could cover the range of examining, manipulating, graphing, analyzing, or reporting on the timeline data.

Ready programmability is not a theoretical property of TLC. Our experience in OpenVMS Engineering and in HP Services over the past three years has yielded impressive results with the creation of tools such as TLViz and CSVPNG and other downstream Friends of T4 as we will learn below.

A Universal Approach

By obvious convention, the T4 kit turns all of its timeline data into TLC-format CSV files. The good news is that it is possible and not difficult for **any** collector of timeline data to automatically create T4-style or TLC-format output on the fly as each timeline sample is captured. For example, four of the six current collectors directly generate their timeline output in CSV format.

This is, as they say, an SMP problem: a Simple Matter of Programming (or perhaps a Simple Matter of Priorities).

Alternatively, timeline data in **any other internal format** can (without huge difficulty) be extracted and converted into TLC-format – another “SMP” problem. The T4 kit includes an extractor that creates timeline columns for logins and logouts. This extractor uses the log data time-stamped in the standard OpenVMS Accounting Log File, searching that file for logins and logouts, accumulating the numbers for each sample period, and then writing the records to the CSV file row by row. This extractor is quite interesting in that it also adds some extra value on the way out by counting the number of logins and logouts of short duration, for example those less than 1 minute in length and those less than 5 minutes in length.

The kind of approach we used with the OpenVMS Accounting Log is readily replicatable to other log files. It could be applied to a whole raft of other collection utilities to extract selected variables not otherwise available and to turn their timeline data into the universal TLC-format.

Follow-up Questions for Timeline Data in Log files

The questions below may help you identify some opportunity areas to extract vital log file data that is not readily available to you today.

Question 1. What vital timeline data relevant to the mission critical purposes of your OpenVMS systems is currently locked inside some log file to which you have potential access?

Question 2. What is the internal format of that log file?

Question 3: What would be the estimated cost to build an extractor that grabbed key statistics from that file and turned them into TLC-format data?

© Copyright 2004 Hewlett-Packard Development Company, L.P.

NOTE: This log file data might now be captured by another system such as the one that supports your company's telephone exchange. Many systems log each transaction with a time-stamped record. Don't overlook this potentially invaluable source of vital performance data that can complement TLC data from other sources.

Timeline Data in Text Files

Another possibility, albeit a somewhat clumsy one, is to capture timeline data as a series of repeated time-stamped entries in a text file and then later automate the parsing and processing of that file to turn key metrics into CSV format. This is important if there are impediments to working directly on the collector program to have it write out the TLC-format CSV file itself. We have done this quite successfully with a number of prototype collectors to capture some vital stats that would otherwise not have been easily available to us in timeline format. While messy, this kind of collector can be readily and speedily constructed whenever needed at modest cost.

Follow-up Questions for Timeline Data in Text Files

The questions below may help you identify some opportunity areas for you to first create and then extract vital data from text files where such data is not readily available to you today in a reusable format.

Question 1. What statistics that are vital to the operation of your OpenVMS systems might you capture as time-stamped entries in a text file? For example, you might do this using a DCL script with a timer loop and an embedded call on an application command that gives total throughput counts for key functions.

Question 2. What's the estimated cost to you to build a suitable extractor to parse this text file and transform the vital statistics into a TLC-format?

Because the TLC-format CSV files are but one of many possible universal ways to format timeline data, note that other SMP programs could be readily constructed (as needed) to transform any TLC-format data into a chosen alternative universal format such as XML or a set of Oracle, Rdb, or MySQL tables.

The Bottom Line for TLC-Format Data is that it is Readily Reusable

TLC-format data can be used as a universal approach to timeline data. Any timeline data from any source (including but not limited to any OpenVMS performance data collector) can be converted to TLC-format data.

Many have already taken up the call. TLC-format collectors or extractors have been written for performance data from Oracle, Rdb, and from customer business statistics such as response time, throughput, internal application queueing, or even such things as sales volume attained by clerks using the OpenVMS system. More collectors and extractors generating TLC-format data are sure to follow as the universal T4 & Friends timeline-driven approach and its benefits become more widely known among the OpenVMS community.

The bottom-line payoff from TLC data is that it is ***readily reusable*** in ways that promote communication and collaboration. This means that it is:

- Readily programmable for new purposes as they are thought up.
- Readily or even instantly viewable.
- Readily synchronizable with other TLC data sources.
- Readily comparable to other TLC data sets.
- Readily extractable into reduced form.
- Readily sharable.
- Readily publishable in documents, presentations, and to the web.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

T4 & Friends – What is Available Today?

As use of the T4 tool kit proliferated, as more and more TLC-format histories were generated, and as the range of uses of the T4 tool kit and TimeLine Collaboration (TLC) format two-dimensional timeline tables widened, it became clear that capabilities beyond those offered by Excel's processing of CSV files could prove immensely helpful. Within OpenVMS Engineering, we discovered quite quickly that not everyone who wanted to use a timeline-driven approach was comfortable with using Excel as his or her main post-processing, value-extraction engine.

With more and more valuable timeline data beckoning, a growing number of other tools, methods, techniques, and approaches have evolved and have proven successful in helping OpenVMS Engineering enhance our timeline-driven approach to performance. We have come a long way from the Original T4.

Here's an abbreviated summary of current capabilities (as of early 2004) to give you a taste for ways in which T4 & Friends might prove directly useful to you.

The T4 Tool Kit (T4V32 and T4V33)

The T4V32 and T4V33 tool kits are a good place to start your exploration of T4 & Friends. See the Readme.txt file in the kit for full details. These kits include scripts that automate historical timeline collection using six separate collectors or extractors, as well as utilities for synchronization, mail distribution, and near real-time snapshots.

You can examine the Readme.Txt file and download the T4V32 tool kit from
http://h71000.www7.hp.com/OpenVMS/products/t4/index.html

The T4V33 tool kit is now shipping with the release of OpenVMS Alpha Version 7.3-2. You can find the kit in the SYS$ETC directory.

T4V3x collection can be a useful adjunct to your existing performance management program, and it co-exists peacefully and with low overhead with all other major OpenVMS performance data collectors. Whatever performance collectors you depend on today, we recommend that you also consider turning on low overhead T4 history creation.

Creation of such a TLC history will automatically open the way for your improved collaboration with OpenVMS Engineering in the future, whenever this might be valuable, useful, or even necessary for you. Of course, as you learn more about T4 & Friends, you will also find that your growing T4 timeline history (building automatically day by day) will powerfully extend your ability to manage performance on your most important systems and complement your existing performance capabilities and tools.

The T4V3x kit includes the T4EXTR.EXE utility – a classic example of a timeline extractor. T4EXTR converts the raw MONITOR.DAT files to CSV format and generates literally hundreds of columns of data. T4EXTR can be used manually as needed to re-examine the same raw MONITOR data and extract additional and more detailed columns of data. For example you can use it manually to find out about specific named process use or about RMS use for files for which you have turned on RMS monitoring. This utility includes several options for customizing and selecting which columns you wish to generate (ALL, CPU, DISK, PROCESS, SCS, RMS).

Two DCL scripts,T4$CONFIG and T4$COLLECT (called "HP_T4_V32" in the T4V32 kit), map a default approach for collecting data every day and transforming it into TLC-format CSV history files. Because these are written in straightforward DCL, many T4 Kit users have already found these scripts to be readily customizable for specific local purposes. These scripts allow you to automatically launch all current collectors and then to combine data from the different collectors into composite CSV files.

These scripts provide low overhead monitoring by using a default 60-second sampling interval. They also include a rough automatic synchronization of data from different collectors, some rudimentary storage management, optional mailing, and, most importantly, the automatic creation of a detailed long-term timeline history for that OpenVMS node in TLC-format.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The current list of collectors/extractors in the kit includes:

- MONITOR (T4EXTR)
- XFC
- Dedicated lock manager
- TCP/IP system wide traffic
- Network adapters traffic (you can request one collector for each such adapter)
- Login and logout activity (using the standard accounting log and an associated extractor)

The combination of MONITOR and T4EXTR alone deliver more than a dozen different views of OpenVMS performance, each view with many independent statistics. For example, the **SYSTEM View** includes such statistics as CPU IDLE, MPSYNCH, KERNEL, BUFFERED IO; and the **LOCKING View** includes such statistics as CONVERTS and ENQUEUES.

Customizing the T4 Tool Kit Scripts

As you look at the T4 kit's two controlling DCL scripts and the ways in which each new collector is included, you will likely conclude that adding your own new collector (for example, one for your most important business statistics) will be a relatively straightforward extension whenever you are ready to do so. You won't be sorry if you consider making some investment in understanding the capabilities of these two DCL scripts.

Industry Standard or Other Widely Available Friends of TLC-format Data

With the Original T4 extractor, we fed its output data in TLC-format CSV files into Excel. Then we tapped Excel's many capabilities for manipulating the columns of timeline data, for calculating averages and percentiles, and for creating a virtually infinite variety of individually customized timeline graphics representing the findings of our analyses. This proved a great advantage to those experienced with Excel and provided an immediate positive visual payback for our investment in the MONITOR to TLC extractor program.

Since then, others have taken TLC data (including, but not limited to T4 captured data) and fed it into a variety of databases (Microsoft® Access, Oracle 9i, Oracle Rdb, MySQL). They then used their expertise with those tools to query and report on the growing storehouse of timeline data.

Others have used OpenVMS utilities such as FTP, MAIL, COPY, Zip, Unzip, Uuencode and appropriate DCL wrappers to move their TLC data exactly to where they wanted it for further processing. TLC-format CSV files typically benefit from relatively high compression ratios when being zipped for transfer.

TLC data have been transformed into WMF (Windows Meta File) graphic files and then imported into Microsoft® Word or PowerPoint for explaining the results. Various drawing and annotation tools come in handy in customizing the results. The illustrations in this article are an example of what you can do with the additions of arrows, text boxes, or circles.

TLC data has been converted to PNG (Portable Network Graphics) images and embedded in HTML web pages. Using Apache Web Server, Mozilla and PHP (all running on OpenVMS), TLC-format graphical outputs can now be published on demand to the web by those comfortable with OpenVMS' extensive web-based capabilities.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Bottom Line – Readily Reusable

Any TLC-format CSV file from **any** source is **instantly usable and readily reusable** by widely known, widely available utilities for analysis, reporting, data transfer, or publishing. TLC tables can be readily converted to graphic timeline images and the images to industry standard graphic output formats such as WMF and PNG. These images, in turn, can be incorporated in desktop publishing documents or even published dynamically to the web. In other words, once key timeline data is converted to the universal TLC Normal Form (**TNF**), everything we can imagine doing with this timeline data is possible and some of those things are immediately available for the asking.

Figure 12 gives a highly simplified picture of the direct transformation from TLC-format data to visual image.

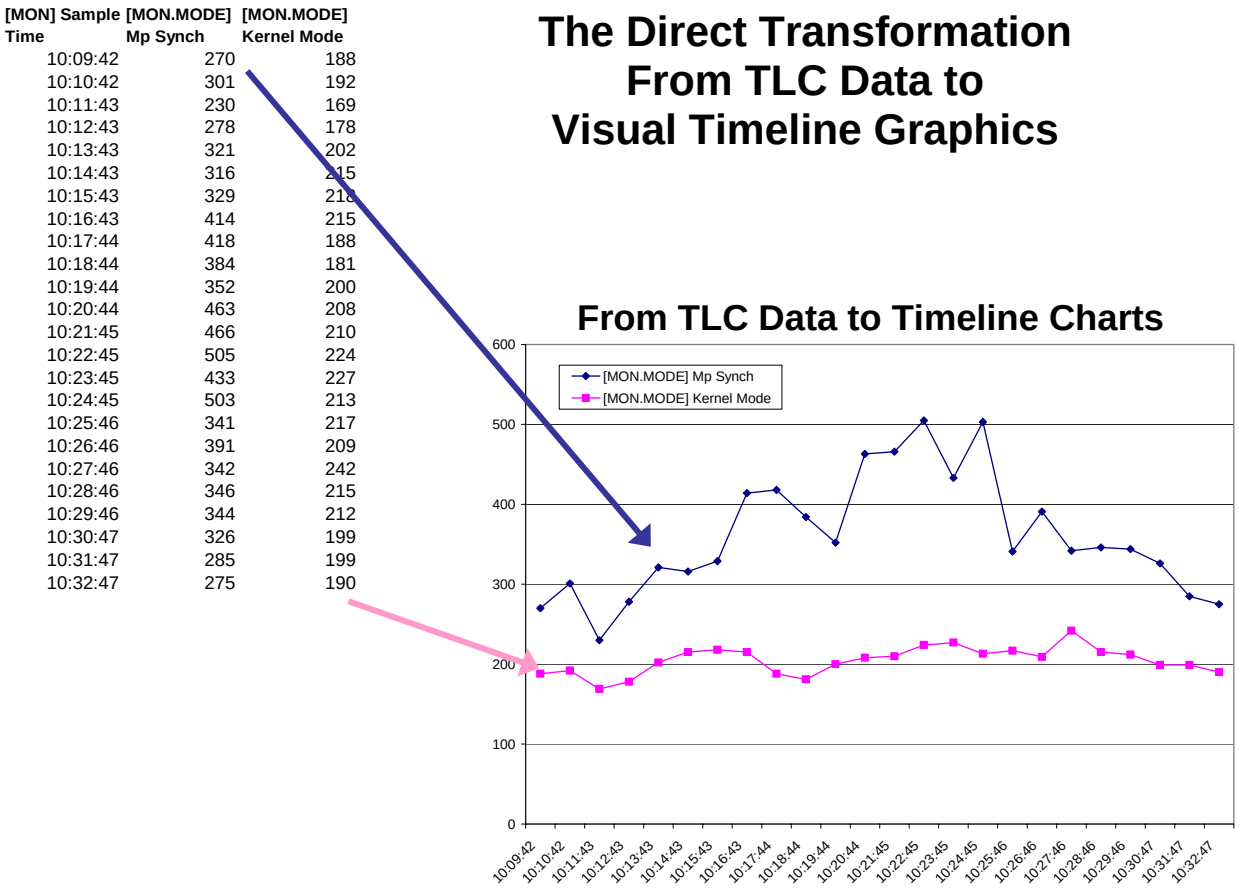

Figure 12 – The Direct Transformation from TLC Data to Visual Timeline Graphics

The three-column table at the left of this chart is a fragment of a TLC data file created by a T4 collection kit. The timeline graph at the right is a direct mapping from the columns of data into visual format created in less than a minute using Excel. TLC data is instantly visualizable.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

HP-Developed Downstream Utilities

The "upstream" collection of performance timeline data and the filling of a large reservoir of online storage with TLC history files are, of course, not ends in themselves. As the reservoir of TLC data grows, the **potential** value of that data bank grows with it. To turn that potential value into actual value requires selectively drawing off some of the data and flowing it downstream into mechanisms whose very purpose is to extract that value.

TLC tables can be readily converted to graphic timeline images and the images to industry standard graphic output formats

Excel was our first such downstream tool, and its use with TLC data created a powerful proof point. It demonstrated the large potential value of historical timeline data when saved in a readily reusable format. Since then, HP has undertaken a series of independent development efforts of downstream utilities to help extract even more value from performance timeline data saved in TLC-format. These include: extensions to the System Health Check offering from HP Services, and two other **internal use** HP utilities.

- A productivity-enhancing, interactive, timeline visualization tool (TLViz).
- A powerful command-line driven tool for manipulating and charting TLC data (CSVPNG).

System Health Check Service. Automated T4 collection is now a standard part of the improved System Health Check (SHC) service used by many OpenVMS customers. The new SHC automatically runs a T4 collection, creates TLC-format CSV files, applies a set of expert rules, and then reports and graphs the results as part of the overall SHC output. So if you are already using SHC, you are already making good use of and benefiting from T4 collection. For more information about SHC, please contact your local HP Services representative or check out: the following web site:

http://www.support.compaq.com/svctools/shc/

TLViz and CSVPNG. TLViz stands for TimeLine Visualizer, and CSVPNG stands for an automatic CSV to PNG (Portable Network Graphics) converter. TLViz and CSVPNG are interesting in their own right as well as being an excellent demonstration of the wide range of possible value-extracting downstream uses that can be made of TLC-format data. We have used these extensively with tremendous effect. They have dramatically changed the way OpenVMS Engineering does its most important performance work. These tools have demonstrated how easy it is to unlock some of the value captured by TLC-format universal timelines.

Once key timeline data is converted to TLC, everything we can imagine doing with this timeline data is possible and some of those things are immediately available for the asking.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

TLViz (TimeLine Visualizer) – A Visual Demonstration of What's Possible

TLViz is an excellent example of what we mean by a "friend of T4." TLViz is an **internal** tool developed and used by OpenVMS Engineering to simplify and dramatically speed up the analysis of TLC-format CSV files and to assist the subsequent reporting and sharing of our findings with others.

The combination of T4 timeline-driven collection and TLViz has literally changed our lives in OpenVMS Engineering. TLViz is a Microsoft® Windows PC utility (written in Visual Basic and using TeeChart software as its graphics engine). TLViz permits the analyst to carry out the most common graphical functions on these large timeline data sets with the fewest possible keystrokes when compared with alternative methods that we have tried. Within OpenVMS, we estimate that TLViz personally gives us an order of magnitude speedup and productivity increase in our own analysis work with this kind of highly multi-dimensional timeline data drawn from multiple sources. Figure 13 is an example of a TLViz output that tells a powerful before-and-after story.

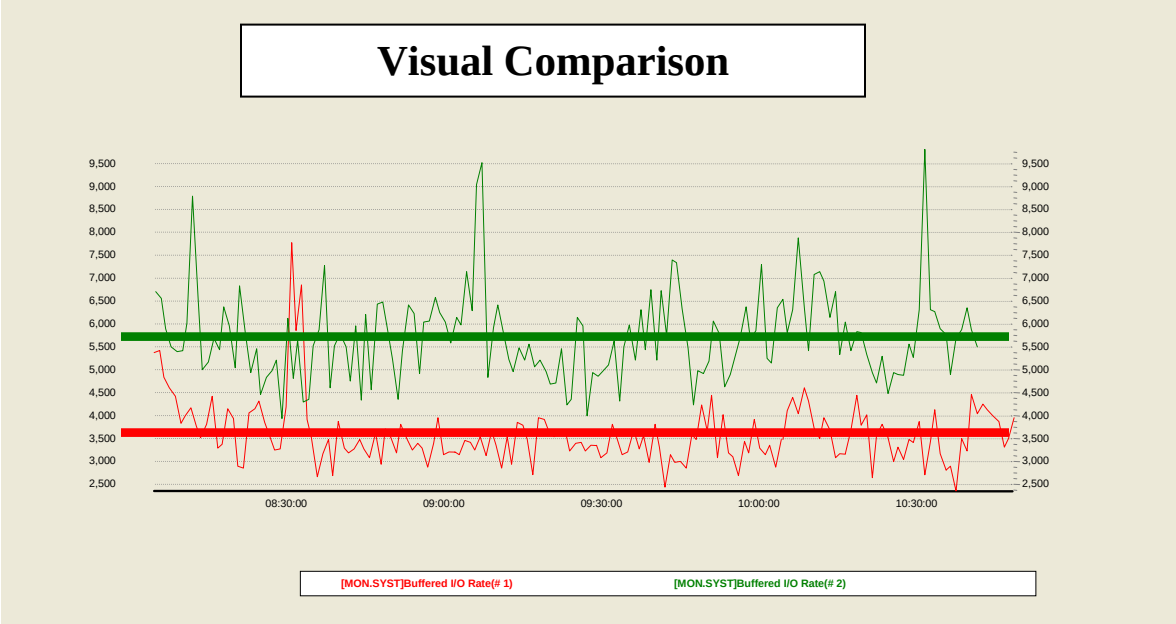

Figure 13 – Before-and-After Visualization

This graph is an example of one of many possible outputs that can be generated by TLViz. The solid red and green lines were manually added later to aid the visual comparison process and add to the chart's visual explanatory power. They were drawn by hand to represent the approximate average value for the two different cases. We did this based on visual inspection and visual approximation of "typical" behavior during the period. Of course, because all the underlying data for each of the plotted timeline points is still available in the TLC-Format files for these two measurement periods, the actual exact average could be readily computed and plotted as needed.

When TLViz opens up a TLC-format CSV file, the names of each performance variable appear in a standard Windows selection box. The names of the performance variables are drawn from the column header for each column of timeline data in the TLC file. By simply clicking once on the performance variable in which you are interested, the visual timeline graph for that variable appears immediately in the graphing window.

With the CTRL-key held down, you can decide exactly which set of variables to map together. Or you can use the arrow keys to move, one graph at a time, through the literally hundreds of variables – getting a quick overview of the underlying patterns – all in a matter of minutes.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

TLViz includes features such as mouse-driven zoom, scrolling, stacking, unstacking, correlating, automatic scatter plots between pairs of variables, column arithmetic, and saving a zoomed-in selected set of rows and a subset of the selected columns to a new, more compact TLC-format CSV file.

Reporting with TLViz

Whenever you see a display you want to save, TLViz lets you add your own title and then export it to a named WMF file for later use. Consider creating a special sub-directory for each analysis session where you can conveniently save the graphs you generate. This feature has proven to be a powerful memory aid. It is also a wonderful collaboration feature as these WMF files often form the basis for the creation of reports and presentations (further downstream) that will share the results of analysis with a wider audience. TLViz allows several other output formats for these graphics. We have found through experience that the WMF outputs work the best. They offer clean graphics with no perceptible loss of resolution, they work well with the standard Microsoft tools such as PowerPoint, and they are relatively well compressed compared to other formats. Documents and presentations containing many such graphical elements also tend to show excellent further compression when these files are zipped for transfer.

TLViz also allows you to open up to five TLC-format CSV files. It then automatically overlays the results for you each graph you select. For example, selecting the column header for Buffered I/O Rate, TLViz would graph the Buffered I/O Rate timeline for each of the currently open files. Figure 14 shows a simple example of this feature.

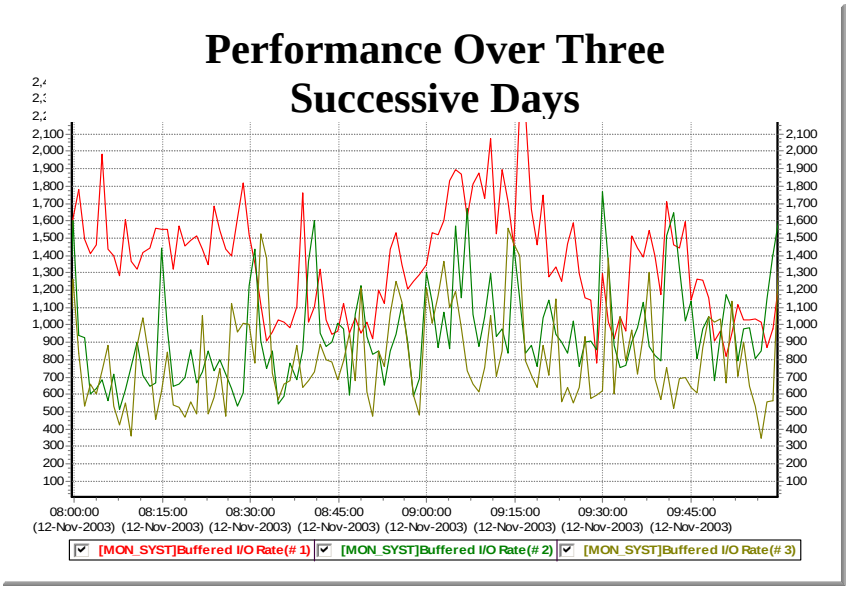

Figure14 – Performance Over Three Successive Days

Comparing Buffered I/O between 8 AM and10 AM over three successive days on the same system, it's clear that the throughput was highest on the first day (Red). The second and third days were about the same, with a slight edge to the second day (Green).

Before-and-After with TLViz

Arguably, the most useful case we have seen is using TLViz' multiple file open feature to do rapid **Before-and-After** analysis when the system under investigation has experienced some form of important change – for example, a significant slowdown on a live production system. This could also be useful for looking at upgrades to new versions of software or hardware, for quantifying the benefits from a series of tuning changes, or for understanding the impact on key resources caused by the introduction of a new application workload.

T4 collection tools, TLC-format data and TLViz' Before & After feature, were instrumental in the success of our GS1280 "Marvel" Performance Proof Point (P3) approach as presented at the ENCOMPASS webcast in November 2003. We plan to apply a similar P3 approach (built on T4, TLC, TLViz, CSVPNG and other friends of T4) to new performance situations. Performance proof points are popular because they help us all more clearly understand and more accurately quantify the actual benefits of performance change. A P3 approach lets us set expectations more precisely for performance improvements. We see

© Copyright 2004 Hewlett-Packard Development Company, L.P.

near term application of the P3 approach as customers with heavy TCP/IP loads and scaling bottlenecks on large production systems upgrade to OpenVMS Alpha Version 7.3-2 and TCP/IP Version 5.4 with its new scalable kernel option.

If you haven't already tried a visual before-and-after approach to look at differences captured in timeline data, we cannot recommend it to you too strongly. A very similar Before-and-After approach can be achieved with CSVPNG, with Excel, or other similar tools. The key to success is to be careful selecting suitable sample days to be **representative** of the before-and-after, and then looking at many independent graphical comparisons. If something has changed by as much as 5%, it will show up in the graphs in a way you won't be able to miss.

Synergy between T4 Collection, TLC-format Data, and TLViz

As we noted earlier, the standard format for TLC data includes several header rows. The first such row is reserved for comment information saved in a CSV format. The latest T4V3x tool kits make good use of this first row of the TLC table to store details about the measured OpenVMS system. These include:

- The AlphaServer node name
- The version of OpenVMS in use
- The version of TCP/IP in use
- The number of CPUs
- The amount of memory employed
- The sampling interval used
- The version numbers of the T4 kit components.

TLViz makes these background details captured at the time of measurement available to the viewer through use of a "Properties" selection from the "File" pull-down menu.

TLViz' Properties feature works for any file in TLC-format. So if you begin to create your own TLC files with your vital business metrics, remember to put background information relevant to that data in the first row, so that it will be available for future review. This might include version numbers of key application or database software or other attributes and properties that are highly specific to your business environment.

Other Uses of TLViz' Multiple File Open Capability

In addition to the powerful before-and-after visualization, TLViz' multiple file open capability can also be used to compare and contrast performance as it changes from Monday to Friday. It can also be used to examine the relative load on different nodes in an OpenVMS cluster for a given day.

Side-by-Side Collaboration Using TLViz

TLViz has proven to be a remarkably powerful tool for side-by-side collaboration. It has allowed us, in selected cases, to have two or three performance analysts work together analyzing a particularly complex problem. The GUI interface, the ability to point to graphical details on the screen, to make suggestions for adjusting what to look at and then seeing the new picture in a few seconds has led to some excellent synergistic problem solving.

TLViz has also proven to be a great way to share results with others quickly without producing a formal report. The basic model looks like this. Having previously studied the situation and noted the important factors, an analyst can use TLViz to project important graphs one by one to the audience. Audience questions can trigger the analyst to shift gears and bring up a graph or a series of graphs that helps answer the question, and then resume with his or her presentation. When you are presenting interactively and live to an audience, the ability to point at features as needed and as driven by the discussion can often add an incredible benefit beyond what can be pre-packaged in a report that cannot possibly anticipate every question. Think of it as a blackboard or whiteboard with built-in automation.

The success of these collaborative approaches with TLViz has a lot to do with the speed at which new graphs can be created. TLViz offers a model GUI design that has transformed the most important activities you might want to carry out on timeline data and timeline graphs into single keystrokes or mouse clicks.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

CSVPNG – Command-Line Driven Capabilities

CSVPNG (CSV to HTML and PNG converter) is a newer, command-line driven utility with many options. CSVPNG runs on both OpenVMS and in DOS on Windows PCs. Like TLViz, it allows you to directly open one or more TLC-format CSV files. Its original capability, which led to the CSVPNG name, allows it to open a TLC-format data file, specify a set of selected columns to be graphed, and then to automatically generate an HTML page with PNG embedded graphics for each selected column. Figure 15 is one of many outputs possible through use of CSVPNG.

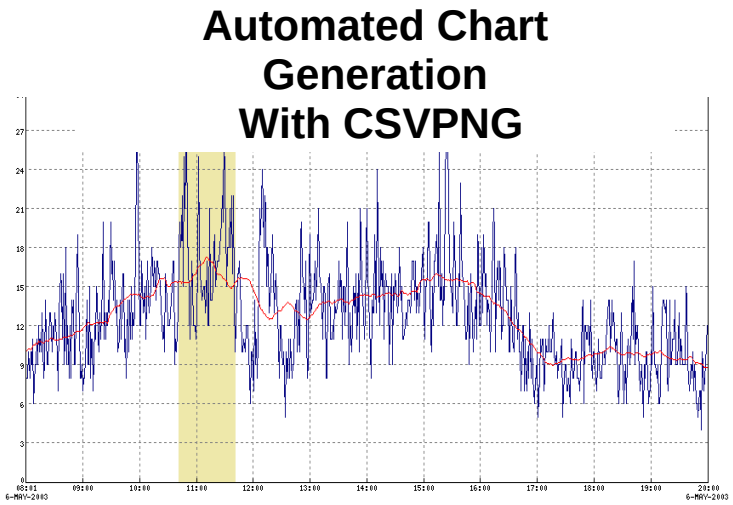

Figure 15 –Automated Chart Generation with CSVPNG

This sample output from CSVPNG shows a moving average for MPsynch in red and the peak one hour period for MPsynch in the shaded area. The individual MPsynch samples are shown in blue.

CSVPNG includes the following capabilities:

- Graphing multiple variables in a single chart
- Opening multiple TLC-format files and overlaying the results
- Calculating and displaying moving averages
- Identifying peak periods
- Carrying out correlation calculations
- Performing column arithmetic
- Applying user-written expert rules.

CSVPNG also has powerful capabilities for "slicing and dicing" a TLC-format data file and producing a much more compact file as output. For example, you could use it to select only the data from the peak one hour period and reduce it further so it outputs only your personal favorite list of 17 performance variables.

Because it is command-line driven, CSVPNG is programmable and already some have demonstrated how CSVPNG combined with Apache Web Server, Mozilla, and PHP (all running on OpenVMS) could dynamically publish T4 timeline graphs from an OpenVMS web-enabled server node. We have even seen demonstrations of near real-time graphical reporting by combining all these tools.

We expect that CSVPNG will become much more widely used for our OpenVMS Engineering performance work in the coming year as we all become more familiar with the full range of its capabilities and potential for extracting value from TLC data. It's not hard to predict that CSVPNG is likely to be at the leading edge of our continuing enhancements to the TLC approach in this coming year.

Use of Internally Built Tools Outside of OpenVMS Engineering

TLViz and CSVPNG have dramatically changed the way OpenVMS Engineering does its most important performance work as visual diagnostic tools, as visual collaboration tools, as visual presentation tools, and as incredible productivity enhancers and time savers during analysis and reporting.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Seeing our success, a number of customers and partners have made their own business case for gaining access to these tools and are reporting back to us the productivity gains they have achieved by using them. If you are interested in learning more about TLViz or CSVPNG, please send your request for more information to the author, steve.lieman@hp.com .

If you already feel you have a strong business case for gaining access to either of these *internal-to-HP* "Friends of T4", please forward your request to the author.

As an analogue to the saying "if you build it, they will come", with TLC-format data, we believe that "if you collect it, they will build." TLViz and CSVPNG demonstrate that once you place important data in universal, readily programmable form, it truly is an SMP problem to take the next incremental steps.

Consequently, we will not be surprised to see other powerful downstream utilities come into existence over the coming months to do even more magical things with the rich storehouse of data now being accumulated.

Other Friends of T4

Building Your Own Downstream Tools

TLViz and CSVPNG are powerful proofs of the concept that the universal data in TLC-format timeline files is readily programmable in ways that start to squeeze the value from this rich timeline data source. Many other uses of this data are possible and potentially even more valuable. We hope some of you will consider constructing your own downstream tools that extract further value from TLC-format data and that you will share the results of your success with us.

It's important to note that both TLViz and CSVPNG did not burst, full-fledged onto the scene. They both started with a basic set of capabilities and then added new features based on feedback from early users. Both of them built upon existing very powerful graphical utilities and did not try to duplicate those capabilities but rather just provided an interface to expose the power of graphing to everyone without further programming.

Refer to the section detailing TLC-format data for the definitions you will need to start building these tools. Further details are available in the README.TXT file included in the latest T4V3x tool kits.

And while you are thinking about what tools you can build, don't forget to turn on your TLC timeline history collection today if you haven't already done so. That way, you'll have the detailed TLC history data you need once your new tools are ready.

Other T4 Style Data Collectors

As we noted earlier, the current T4V32 or T4V33 umbrella utilities actually drive six independent collectors or extractors and then later combine, integrate, and synchronize data from all of them into a single, consolidated TLC-format CSV file with literally hundreds of individual variables represented. T4V3x automates a consistent start time, end time, and sampling interval for all six streams of data. We plan to add new standard collectors to future versions of T4Vxx as these become available.

Even better, **anyone** with important performance data can write their own TLC-format collector or extractor and create their own completely universal TLC-format CSV files in TLC Normal Form (**TNF**).

The minute that you turn your key performance data into universal **TNF**, all of the old and all of the new and evolving downstream capabilities for manipulating, analyzing, graphing, and reporting automatically become available to you. For example, you might consider converting application response time data for key business transactions into a reservoir of TNF data and then reaping the downstream benefits.

Even better, if you follow some simple, standard rules for TLC-format collectors (see next section for details), your newly minted data will line up with and synchronize with and be combinable with all the data from the standard T4 collection utilities and with any other collector that also plays by these same rules. We have barely scratched the surface.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Synchronization Reveals Relationships that Truly Matter

We simply cannot stress how important and valuable the synchronization of data from multiple collectors can be for any type of performance situation with which you will be faced in the coming year. This is especially true when business data such as response time or application throughput can be combined with underlying system statistics, with network statistics, and with database statistics. The ability to rapidly identify cause and effect relationships that really matter is increased many fold. Identification of the most important time periods to zoom in on is also greatly aided by bringing multiple views of data into play.

Although the TLC based T4 & Friends approach is relatively new, many important unique collectors have already been built and integrated with T4V32 data with excellent results by those who have taken that path. These include:

- Key Oracle statistics from Oracle 7, 8, and 9
- Key Rdb statistics
- Customer application response data
- Customer application throughput data

Several OpenVMS customers have already created single composite pictures of performance on their most important mission-critical systems that include their vital response data, their key database statistics, and the full set of current T4V32 statistics from its six collectors. They have used the standard APRC utility included in the T4V3x kit for this purpose.

APRC simplifies combining and synchronizing data captured from multiple sources and is as readily available for your specially built collector as it is for the current six standard T4V3x collectors.

When it comes to TLC-format collectors, literally anything you can think of is possible. We're hoping to see many such new TLC-format collectors pressed into useful service and (where possible) shared in the coming months.

Many important statistics do not yet have their own, easy to use collector. Can you help?

Don't forget to turn on your TLC timeline history collection today.

The minute that you turn your key performance data into universal TNF, all the downstream capabilities for manipulating, analyzing, graphing, and reporting automatically become available to you.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Building Your Own TLC-format Collector

Here are the rules you will need to follow so that your TLC-format collector will generate data that can be synchronized with other TLC-format data sources.

Your collector requires four input parameters:

- Start time
- End time
- Sample interval duration (default for T4V3x is 60 seconds)
- Output file name for TLC-format data in CSV format

No Drift Make sure that your collector's samples do not drift later and later. This insures that your collector will have exactly the same number of samples for a given measurement session as other **no-drift** TLC collectors. Unfortunately, some collectors we have known do in fact drift. For example, there are some collectors where each sample starts some given number of seconds (for example 60 seconds after the end of the previous one). Because it always takes at least a small amount of time from the beginning to the end of each sample, the next sample actual completes in the specified number of seconds plus a little more after the end of previous sample. Over time, these small delays mount up and drift occurs. While in most cases, you can live with this, drifting makes synchronization more difficult. For new collectors, drifting must definitely be avoided for best results. **Good News:** The long-standing drift experienced by those using MONITOR has been eliminated with OpenVMS Version 8.1. Those running OpenVMS Alpha Version 7.3-2 or earlier will experience some MONITOR drift and should be prepared to deal with it by exercising caution downstream during analysis. A workaround is available for use on earlier versions of OpenVMS. Please contact the author for details if you feel this would be helpful to you.

There will be a number of header rows (currently 4 for historical reasons). The first row of TLC-format files includes important comments about the measurement session in a CSV format. The second row has the calendar start date of the measurement. The third row has the start time of the first sample.

The last (currently 4th) header row will be the column headers naming the individual variables being measured. This will also be a comma-separated list of values. The first column header in this row, by convention, includes the text "Sample Time" for TLC-format data files.

Many TLC collectors and extractors have used an initial text string in square brackets to identify the collector. For example, the collector for XFC data uses "[XFC]" as its initial string for each measured dimension such as Read I/Os per second. Similarly, many TLC collectors or extractors that have multiple views include the view name in the initial bracketed string. For example, MONITOR has many views such as SYSTEM, IO, LOCKING, and PAGING. The starting string for the MONITOR SYSTEM view would look like: "[MON.SYST]". This would be followed by the metric name, for example "Buffered I/O Rate". While this convention is not mandatory for TLC-format data, it has proven useful in practice. For example, more than one collector might use the same text string such as "I/ O per second" for one of its metrics. Without the initial string identifying the collector, you would end up with duplicate column names.

Each sample period will generate exactly one row in the output. These samples will start in the fifth row using the current definitions for TLC-format

Note:
The idea of one row per sample period is absolutely vital if you want to create readily reusable and instantly visualizable results.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Commas will separate entries for each measured variable. If a particular variable is not available for a sample, a comma will be written nevertheless as a place holder for that variable. This insures that each logical column of data in the resulting two-dimensional table lines up and represents exactly one variable. In other words, the Nth variable always will end up in the Nth+1 column

The first column will have the date and time as its data value. We have chosen the convention that the time of a sample represents the time **at the end of the sample period**. To insure proper synchronization of data from multiple sources, make sure that you also use the interval end time as your timestamp value.

Ideally, samples from each TLC collector will start on major time boundary consistent and modular to its chosen sampling interval. For example, when using sixty-second sampling, it proves helpful to start exactly on a one-minute boundary. Ideally, samples will follow a **No Drift** pattern and, therefore they will be evenly spaced, without any missing samples. Where samples from multiple independent collectors are available, it will also be helpful if they all start and end at the same time and use the same sampling rate. By making collection properties uniform, downstream synchronization is more readily realized. However, the world is often imperfect, samples are missed, collectors drift, starting times may differ, and so on. When extracting value downstream from reservoirs of TLC data, it always pays to be on guard for imperfections in the gathered data. This is a fruitful area for future TLC enhancements and improvements.

Wherever possible, limit the TLC files you create to a maximum of 255 columns since Excel and MS ACCESS have a 255-column limit. Tools such as TLViz and CSVPNG can easily handle a much larger number of columns (1000 or more). If you consider building your own downstream tool to extract value from TLC files, it will be best if you make sure that you can handle higher numbers of columns. We have already introduced and are looking into some further changes in T4 kit collection to keep its standard number of columns generated under Excel's 255-column limit. Excel offers some useful features that are not available in other tools, and it’s always best when we don’t have to reinvent the wheel.

Rules for generating output files for extractor programs are identical. With extractors, the actual data collector may be **event driven and continuous** rather than having a start time, end time, interval approach to collection. For example, T4V3x uses time-stamped event data from the continuously collected accounting log file to compute the number of logins and logouts in a given period and then writes the result to its TLC-format data file.

In many cases, new TLC-format collectors and extractors have been written in rather short order. For example, the first Oracle 7 collector prototype with about 20 key variables was completed overnight. Other collectors have started with a small number of variables collected in version 1 and then readily added new metrics to the collection process as time went on.

What collector do you want to build?

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Techniques for TLC-format Data

Let's say you have begun to create daily timeline TLC-format CSV files for your most important systems and that you have even written a few of your own collectors to capture and save a timeline history of your most important business metrics. What kinds of things can you do with data like this that help you manage your performance situation even better than in the past?

The three primary reasons for wanting to capture timeline data in the first place are the assumptions that:

1) Key performance metrics vary over time.

2) The mix changes in unpredictable ways.

3) Averages can often be misleading.

Now that you have timeline data, the obvious first step is to observe how all your key indicators change over time. For most people this means converting the columns of data in TLC-format into visual, colorful, easy to understand graphs.

Graphing Single Indicators, One by One

It's of course possible to create very complicated timeline graphs with dozens of variables represented. There is so much data, so little time, and so much at stake. We have found that the place to begin in almost all situations is to look at the shape and pattern of the timeline curve for each key variable you have collected. We recommend that **Step One** simply be: graphing single metrics one at a time as shown in Figure 16.

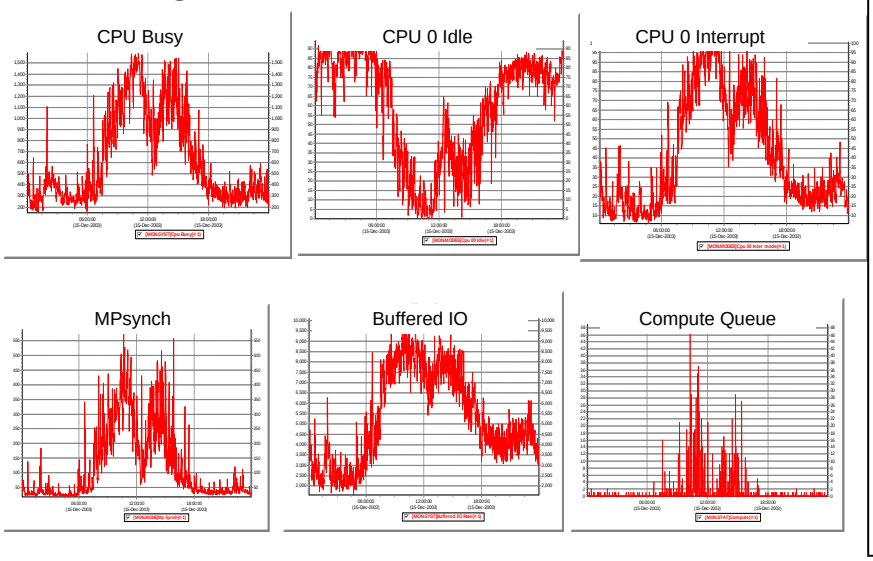

Figure 16 – Looking at TLC Variables One at a Time

Here we use Edward Tufte's concept of **"small multiples"** (see Envisioning Information, Graphics Press, 1990) to look at six key performance variables - one at a time, and all at the same time. These were taken over a 24-hour period from T4 data on a heavily loaded GS160 system with 16 CPUs. Note how four graphs show similar morning and afternoon peaks and lunchtime lull, while the CPU Zero Idle graph shows a reverse pattern. This data was captured as part of our GS1280 Proof Points Program and shows the "Before" case. This particular signature pattern (high MPsynch, heavy interrupt on CPU zero, high buffered I/O rate) is a classic sign of a system that is likely to benefit substantially upgrading to the GS1280.

Here's what to look for: You are going to look for peaks and valleys and their duration. You might attempt to visualize an average value. Be on the watch for short-lived spikes of behavior and whether these show some kind of periodic repetitive pattern. Also, watch for square-wave patterns and pay attention to rising and falling trends. We've already shown an example of visualizing the average and

© Copyright 2004 Hewlett-Packard Development Company, L.P.

quite a few examples of peaks and valleys. The next two illustrations (Figures 17 and 18) show examples of these two cases: a systematic short-lived spike and a repeating square-wave pattern.

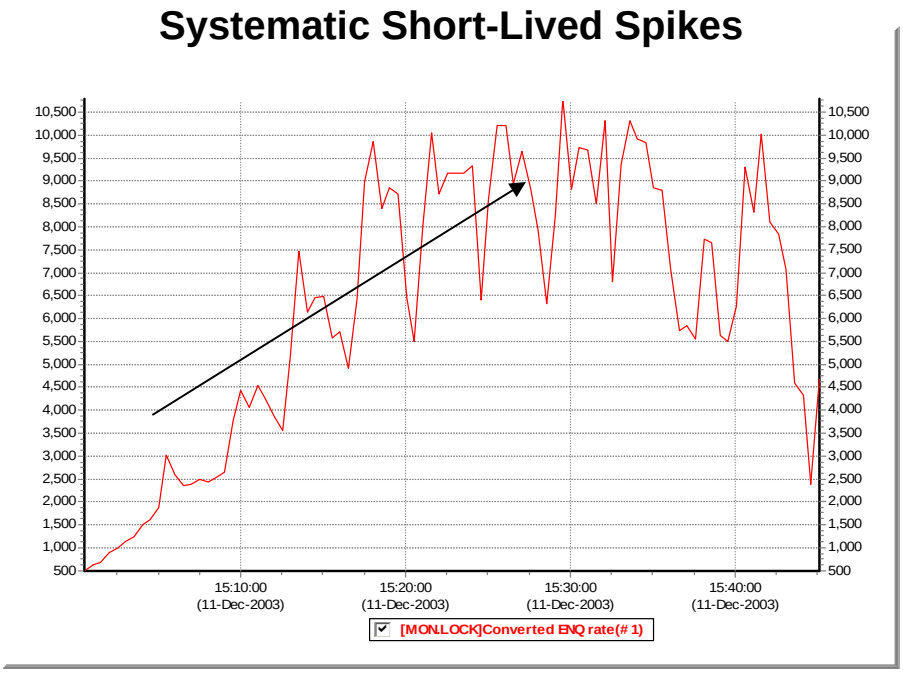

Figure 17 – Systematic, Short Lived Spikes

This chart shows the Converted Enqueue Rate from a heavy benchmark run. The peaks occur about 4 minutes apart and last for multiple samples (30- second sampling in this case). The valleys also come about 4 minutes apart but typically last for only a single sample. Systematic spikes like this are important, because they usually signal some significant underlying cyclical behavior. Also notice the rising trend from 15:00 to 15:30.

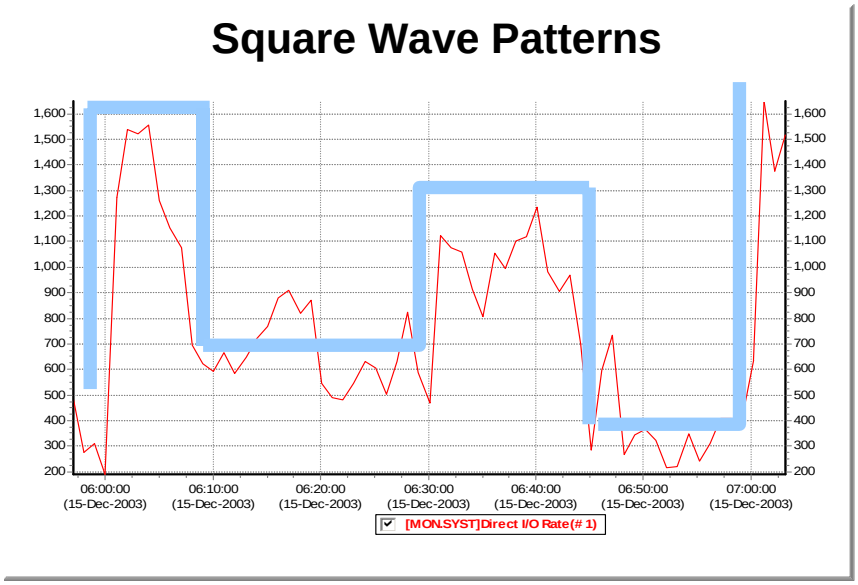

Figure18 – Square Wave Patterns

Square waves are frequently important in performance analysis activities. They typically indicate the start and end of distinct periods – periods when ***"THE MIX"*** of the type of work has changed or where sudden changes in load have kicked in. We have found in our analysis that it's often useful to focus on only one such period at a time. Identification of square wave behavior for key variables gives you the information you need to make sure you are not averaging data from distinct periods together. This chart shows several successive, measured, square-wave patterns with the Red Timeline. The pale blue hypothetical square wave is shown for reference. Real data is messy, but with a little practice, it's pretty easy to identify these kinds of square wave patterns. SUGGESTION: Check back on some of the previous graphs to see what other square waves you can discover.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Plugging in Local Intelligence

There will always be some data and information and background intelligence about your system that is not captured by any one of your current set of TLC-format collectors. For example, perhaps Monday is always the busiest day of the week for you because a certain kind of backlog builds up every weekend. Or your disk activity is highest Friday night, because that's when you always do your full backups. Or your company just ran a full-page nationwide advertising spread that has quadrupled your call volume on your inquiry line.

Because it is your system and your local intelligence information, you and your local colleagues are likely to be the only ones who are aware of this very special kind of data. As you are looking at the metrics collected, you will want to bring this very specific local knowledge into the picture as you try to make sense of what you are seeing.

As you move from graph to graph, it's often useful to posit hypotheses that try to explain and connect the patterns you are beginning to see and relate them to your specific knowledge of the local system.

Quite often a similar pattern will appear in many graphs, for example, a predictable lowering of activity during the lunch hour, and a gradual tapering off at the end of the day, followed by a heavy load when overnight processing jobs kick in, followed by an early morning quiescent period before the workday begins. Figure 19 is an example of how you might use timeline charts to identify similar patterns

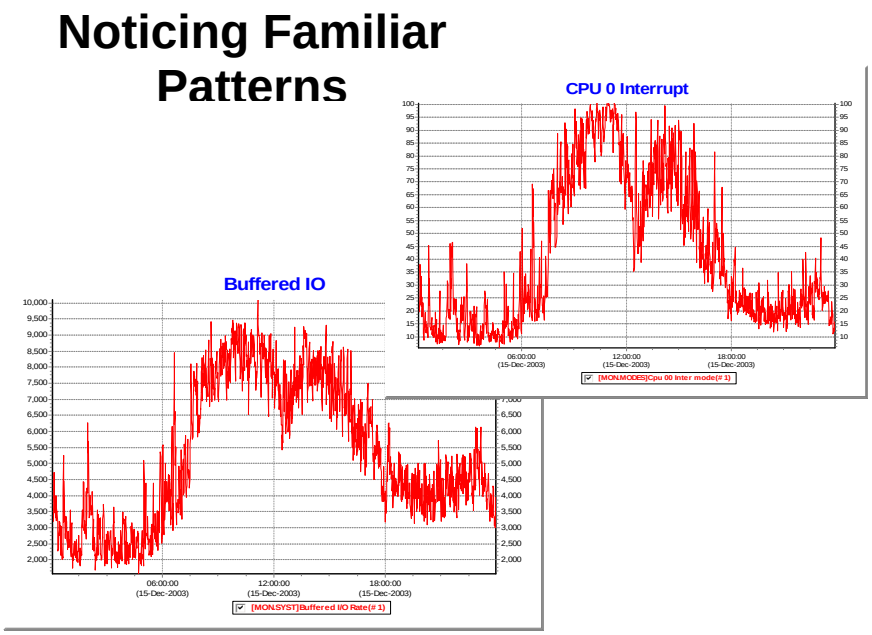

Figure 19 – Noticing Familiar Patterns

Here we examine Buffered I/O and CPU Zero Interrupt activity for a full 24-hour period. The obvious morning and afternoon peaks and lunchtime lull we have already discussed. Look closer in these charts, and you will notice the slight but steady increase in load from 9PM till about 11:15 PM, and a sharper drop off before midnight. You will also notice that the load from 6PM to midnight is higher and more stable than the load from midnight to 6AM. These patterns are important. They can help you know where and when to look and assist you in identifying unique periods of behavior that would benefit from individual analysis rather than being clumped or averaged with other dissimilar periods.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Forming More Complex Pictures

After you have taken stock of key variables one by one, you have mentally prepared yourself for the next step and are ready to create more complex pictures of the data to test your hypotheses. The natural thing to do here might be to graph several variables together at the same time and see if their shapes overlap or move more or less to the same rhythm. You may want to zoom in on peak periods to have a closer look at the behavior at that time. In some cases, you may want to stack several metrics together to create a composite view.

With the right downstream tools such as TLViz, this analysis work can be done independently or collaboratively. Groups of two or three can work simultaneously on the toughest problems. In our small group work within OpenVMS Engineering and with our customers, we have repeatedly proven the collaborative benefits that accrue when you the have real-time ability to work directly with the TLC data to: visualize timeline patterns, point out graphical details to each other, discuss openly their meaning and significance, and propose and then check out hypotheses by graphing other related data.

You might also want to try scatter plots between two related metrics. When doing this, keep your eye peeled for non-linear relationships (see Figures 20 and 21) that might reveal a potential bottleneck. You might also watch for signs of multi-modal distributions of these two variables indicating that the "mix" had changed.

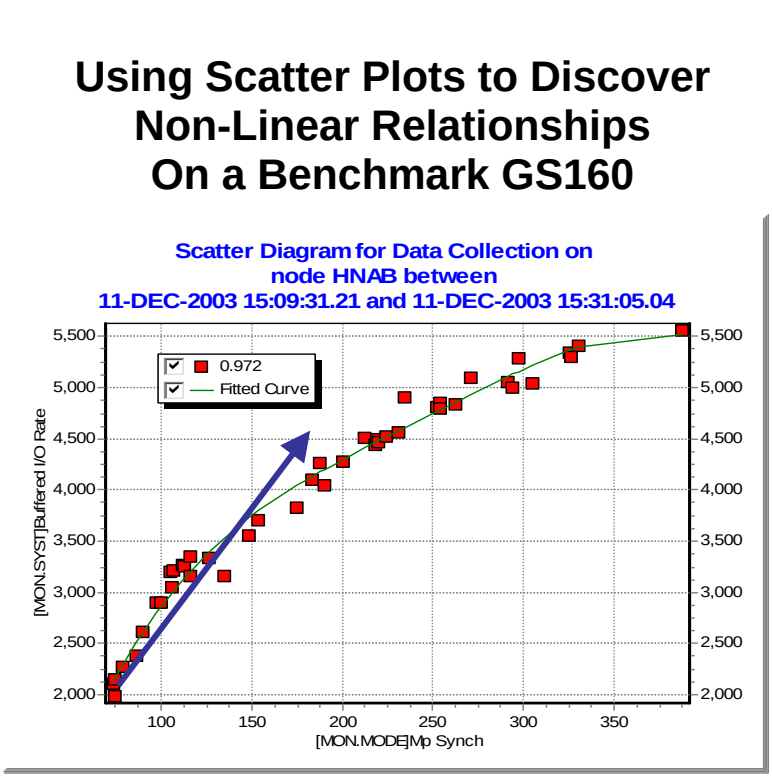

Figure 20 – Using Scatter Plots to Discover Non-Linear Relationships On a Benchmark GS160

Sometimes it's helpful to suppress the time-dependent aspect of TLC data when looking for relationships between metrics. Scatter plots often provide a new view that's not obvious when looking only at the individual timelines. In this chart from a GS160 benchmark run, we plotted Buffered I/O against MPsynch and discovered substantial non-linear behavior. The red squares represent the individual samples plotted. The green line is a curve fitted to the data points. The blue arrow has been added to point out the direction that linear behavior would have taken. Non-linear behavior like this is often a sign of diminishing returns. In our GS1280 Proof Point Project, this pattern proved another signature for recognizing systems that would benefit from upgrades to the GS1280.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

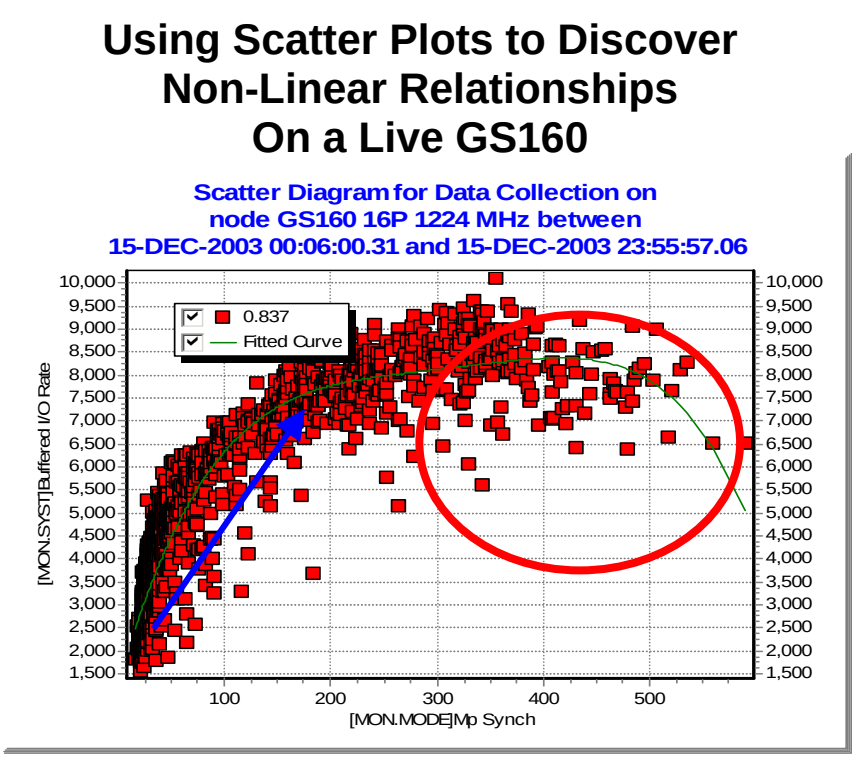

Figure 21 – Using Scatter Plots to Discover Non-Linear Relationships on a Live GS160

This chart plotting Buffered I/O against MPsynch from a live, heavily loaded production GS160 system is even more revealing. The red squares again represent the individual samples plotted, the green line is a curve fitted to the data points, and the blue arrow points out estimated linear behavior. The red circle has been added to bring your attention to what happens when MPsynch pushes past 300%. Notice that the maximum achievable Buffered I/O rates actually begin to diminish. This particular system benefited mightily from its upgrade to the GS1280 as a result of the GS1280's proven ability to handle MPsynch demands so much more efficiently than previous hardware platforms.

If the data is quite erratic, you might want to create a moving average to smooth things out. That way, the overall pattern might be more easily detected. Figure 22 gives an example of how moving averages can sometimes simplify the overall picture.

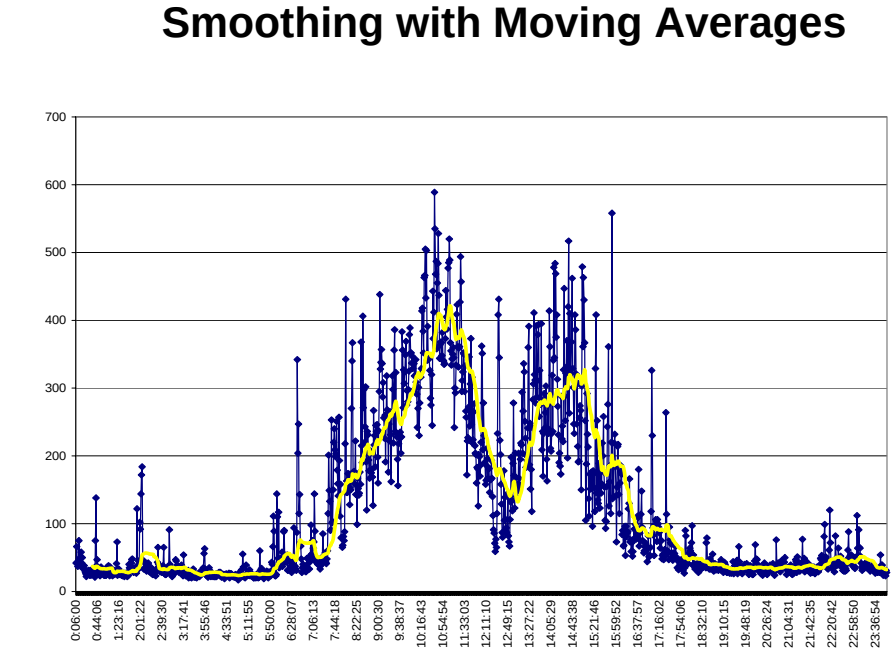

Figure 22 – Smoothing with Moving Averages

This chart created with Excel shows rather erratic MPsynch data from the production GS160 as a blue timeline with a 30-minute (noticeably smoother) moving average added in yellow.

When you see an irregular pattern like this, you might also want to consider capturing data more frequently to see if there is some underlying, explainable, square-wave pattern at work

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Or, you might want to do some column arithmetic (for example, divide metric 1 by metric 2 to create a new normalized metric). This can sometimes prove extremely revealing and help you better understand the points when the mix changes.

For example, if you took total CPU busy and divided by the rate of business transactions per second, you would get a metric that represented CPU consumption per transaction (see Figure 23). Typically, you will want to represent this value in either milliseconds or microseconds used per transaction. Of course, it may not be technically correct that all those CPU cycles actually are directly used for the measured transactions but that does not invalidate this approach. When the mix stays about the same, our experience shows that the ratio of CPU to transactions would also stay about the same. When the mix changes, the ratio will often change, sometimes radically. When the mix returns to the original value, the ratio will change back to its original value and range. This can produce some striking square wave patterns that are clear visual indicators of a changing mix.

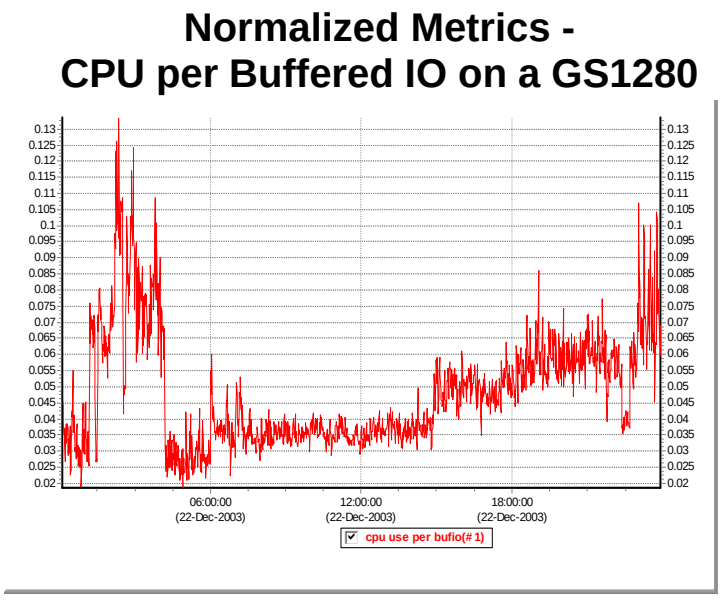

Figure 23: Normalized Metrics – CPU Per Buffered I/O GS1280

This chart shows the normalized metric of CPU (in hundredths of a second) per Buffered I/O on a large production GS1280 system. This is an excellent example of normalized metrics exhibiting square wave patterns that then let you see when the load and mix have taken a dramatic change. Notice the very steady behavior from 7AM until about 3PM and the radically different behavior from Midnight to 6AM. The steady daytime pattern is a strong indicator that the overall mix of activity stayed nearly the same despite what we know from other charts to be rather large swings in load, morning peaks, lunchtime lulls and so on.

The overnight picture is a strong indicator of a series of changes in the overall mix of work.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Figure 24, puts the normalized CPU per BUFIO data to good use by combining it with the Before-and-After technique to reveal patterns that might otherwise have not been detected.

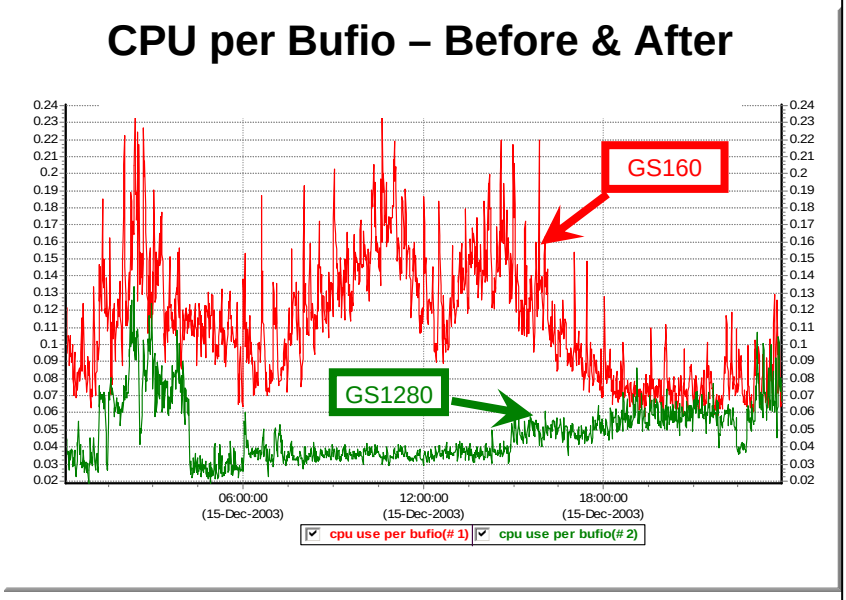

Figure 24: Before-and-After with Normalized Metrics

This before-and-after chart shows the same normalized metric of CPU per Buffered I/O (in hundredths of a second) for a GS160 that was upgraded to a GS1280. This chart will repay careful study. Note how the lunchtime lull pattern appears for the GS160 (red) but not for the GS1280, despite the fact that from throughput metrics such as BUFIO per second we know that both systems had very the classic lunchtime lull loading pattern. What hypotheses can you come up with that explain this difference?

Visual Memory – Your File System is Your Friend

As you go along, you will find that certain graphs you create stand out and tell an important part of the story you are forming in your mind. You are going to want to save these graphs. These might be saved as named files in WMF, PNG, GIF, JPEG or some other convenient graphic format for later reference. If you are working in a tool like Excel, you can save the key graphs on their own separate and named worksheets. Whatever tool you are using for analysis, remembering the most important graphs is an absolutely key step of any such project.

For example, most of the timeline charts in this report were created using TLViz to generate named WMF files. These files were later inserted onto individual PowerPoint slides. Titles, arrows, and circles, were added as further annotation to produce a slide that could then be inserted into the Word document for this article. The time-saving properties of TLViz to help you find exactly the graphs you want (the graphs that tell the story about what is happening), coupled with time-saving ways that TLViz can help you remember this story have proven indispensable on this project as they have on dozens of others over the past several years.

When you are done with your analysis, you will often find that you have a dozen or more such graphs and that you can come back to a data set, weeks later, and remember where you left off in pretty short order by reviewing your full set of visual reminders of the work you have already done.

Reporting on Your Findings

Once you have completed your analysis, you will often have a need to present your results to others inside and outside your organization. This usually works best when you select a simplified subset of your visual memory system that fits your audience. Different audiences quite commonly have different needs. What succeeds and communicates well with a technical audience of performance experts might not work as well for a non-technical audience, for example, those with application access to the system who depend on its rapid response to get their job done.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

While you may have used dozens of graphs to figure out, in depth, what actually happened, once your analysis is complete, perhaps only two or three graphs with accompanying captions and explanation will be needed to make your case with others. The remainder of your collection of graphs will be there as a visual reminder that can be called on as a strong backup for the case you are making whenever further proofs are needed.

You may wish to come back and customize or annotate the graphs or create new copies for these purposes, but you will likely find that a wide audience of both technical and non-technical people will readily understand your handful of timeline graphical explanations.

Bottom line: TLC-format data and the timeline graphs that flow readily from them can be a powerful visual story-telling mechanism that will get your point across with maximum effect in the minimum time. This can be done in documents and PowerPoint presentations. With a tool such as TLViz, the key timeline data sets can also be shared visually and interactively to tell the story and to foster further discussion with audiences large and small.

Understanding Change: The Power of Before-and-After

With TLC-format data collected and saved in a historical archive each day, you are perfectly positioned to deal with the changes over time on your most important systems. These changes could be intentional such as upgrades to new software versions, or such things as inexplicable slowdowns that have been thrust on you from who knows where.

We have found a simple Before-and-After technique to be incredibly powerful and full of explanatory magic in these situations. The basic method is to use local knowledge of the system in question to select a ***representative*** day from "BEFORE" the change, and a ***representative*** day from "AFTER" the change.

> NOTE: What a ***representative*** day mean is, of course, always subjective and often difficult to assess. Making good choices depends on explicit knowledge of what's happening on site with the systems being measured. Until you have proven experience that your assessment of "***representative"*** is accurate, we advise a cautionary approach where you would sanity check your results by selecting several days from each period and examining and comparing before-and-after results of each of the days within each period.

The data for key indicators from the Before-and-After TLC-format data sets could then be overlaid, one metric at a time.

The overlay of exactly two timelines brings into play our powerful human abilities for visual averaging, for visual comparison, for visual arithmetic, and for visual hypothesis formation and sanity checking. If something has changed that impacts performance, that difference will show up clearly in the overlaid before-and-after timeline graphs for at least some (but probably not all) of the key metrics. And the metrics where the changes are most dramatic will invariably give you clues to what is really going on and why. Figures 25 and 26 on the next two pages give examples of the power of this approach drawing from two GS1280 proof points.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The Power of Before & After Upgrading a GS160 to a GS1280

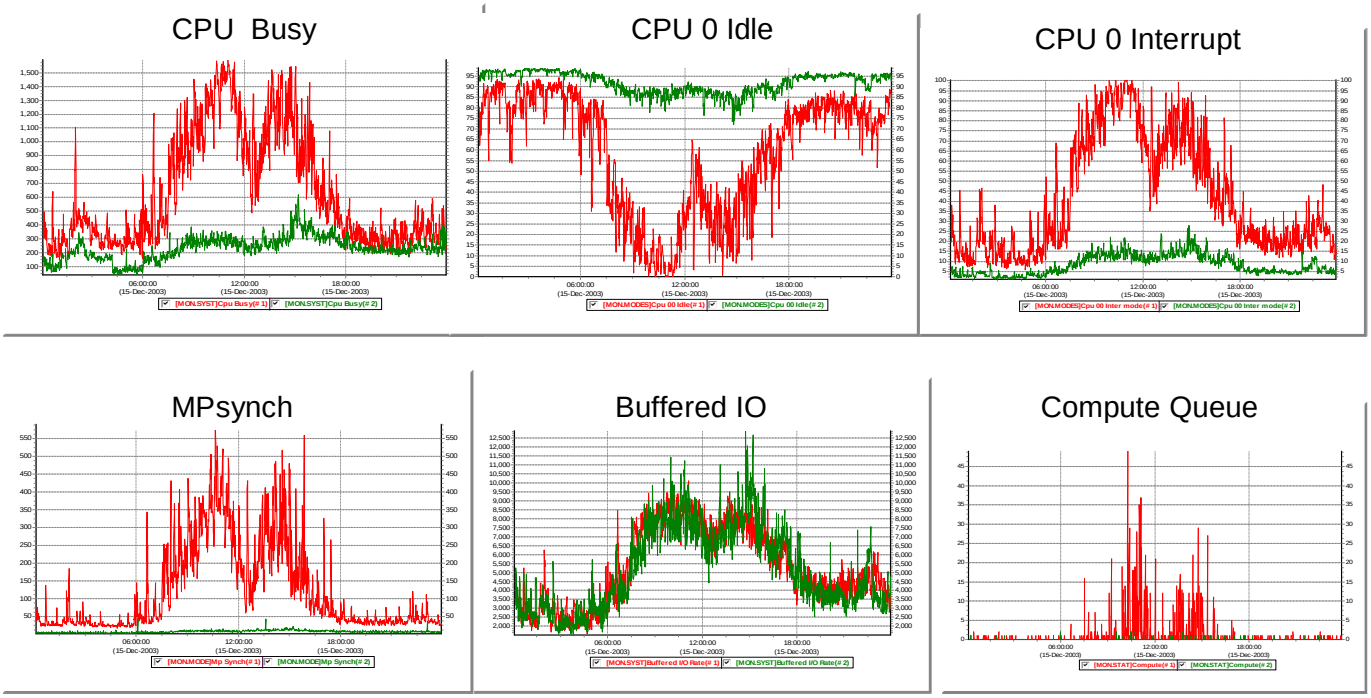

Figure 25 -- The Power of Before & After: Upgrading a GS160 to a GS1280

This set of charts shows the amazing night and day differences we observed when a heavily loaded GS160 (red) was upgraded to a GS1280 (green). This set of charts represents one particular classic case we have seen over and over as part of our GS1280 Proof Points Project (P3). Note the huge drop in CPU busy during the peak periods (UPPER LEFT), the virtual disappearance of MPsynch and its non-linear effects (BOTTOM LEFT – the green line for the GS1280 is almost invisible on the bottom of the chart), and the creation of substantial spare capacity on CPU 0 (UPPER CENTER). In this case, the underlying workload demand had not changed during the upgrade so the Buffered I/O chart (BOTTOM CENTER) shows virtually identical behavior. What's happened here is that substantial spare capacity (perhaps 3X or more) for additional future work has been created by the upgrade and this customer can now begin adding load to this system to match the needs of their growing business.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The Power of Before-and-After: Part II Another Classic Signature Pattern

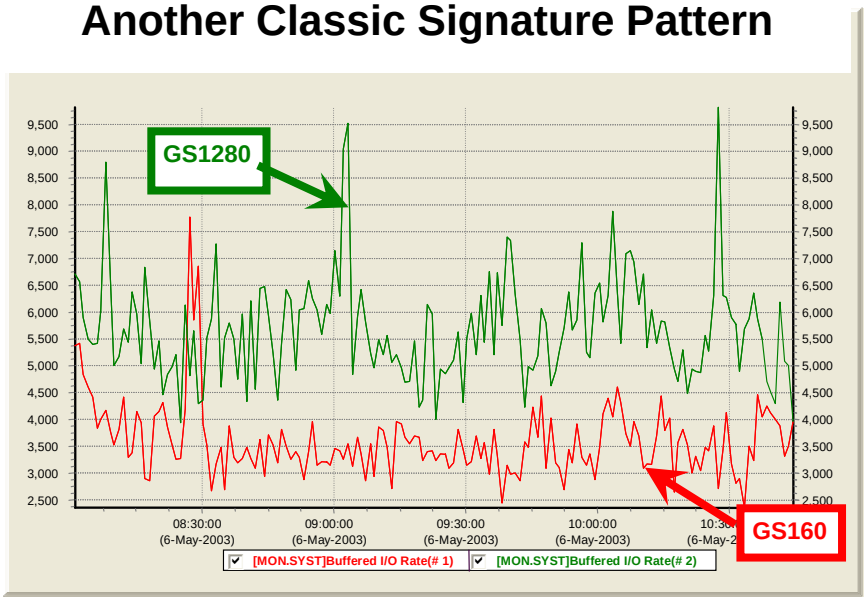

Figure 26: The Power of Before-and-After - Part II – Another Classic Signature Pattern

Again, drawing from our GS1280 P3 work, we compare changes in Buffered I/O throughput for a production system upgrade from a GS160 to a GS1280. In this case, the GS160 was maxed out, and the customer already had a substantial backlog of demand during the peak periods that just could not be completed in a timely manner. With the upgrade to the GS1280 came not only large increases in spare capacity (as shown in the previous example), but also an immediate payback in higher throughput (approximately 1.5X) to handle the existing workload backlog.

If you haven't already tried the before-and-after technique, we strongly recommend that you check out how this approach can help you more fully understand the changes taking place on your most important systems.

The GS1280 Performance Proof Points (P3) Project borrowed heavily and benefited mightily from this approach as demonstrated with many of the examples shown in this article. For more details about the GS1280 P3, please contact the author.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Other Uses of TLC Data

Problem Solving After the Fact with T4 History

Those who remember the past are not condemned to repeat it.

If you have a case of an unintentional slowdown, it's possible that that the slowdown has been gradual and that no one had noticed or complained until now. By selecting a number of older data sets and comparing them to the current performance, it is often possible to pinpoint the exact day when the change began to creep in. This can help you figure out what the primary cause was that triggered the slowdown by checking your system logs for that time period to see what changes were introduced.

Your history of TLC-format data on your most important mission critical systems can help you determine exactly when a problem first surfaced (after the fact). This is yet another reason to turn on timeline collection in advance. You'll never know when you will need it next. Those who remember the past are not condemned to repeat it.

TLC Data is Ripe for Further Analysis

In addition to all the wonderful graphical things you can do with the column upon column of TLC-format data, these columns of numbers are wonderful raw materials for a host of powerful analytical, mathematical approaches to performance.

For example, using a tool such as Excel, you can find averages, medians, minimum, maximums, and percentiles of any stripe with relative ease. You can compute moving averages of any duration or have them automatically graphed. You can create histograms with varying bucket sizes or carry out extensive column arithmetic to develop important normalized data.

Using Excel or CSVPNG, you can add up CPU busy or Direct I/O rate across all nodes in a cluster.

With Excel, CSVPNG, and TLViz you can also carry out linear correlation and discover which metrics appear to be most closely related to each other in behavior. With any of these three tools and other similar tools, you can zoom in on a particularly interesting peak period and recalculate correlation for that window.

With Excel, you can discover the peak hour or peak half hour for a particular variable. CSVPNG lets you automate the peak hour calculation.

And if there are certain systematic calculations that you find useful to repeat with each data set, the same data will readily yield to customized programming in the language of your choice once you decide what works best for you.

If you have a many-month history of TLC-format data for all the nodes in your cluster, you might want to consider analyzing changing trends that are showing up day-by-day, week-by-week and month-by-month. No automatic tools exist as yet to transform TLC data in this fashion but such capabilities appear to be just one or two short steps away.

Once you have data in **TNF**, the possibilities for analysis are endless and only limited by the interplay of your time, your imagination, your aptitude with Excel, SQL, and other available tools, and your programming skill to craft new tools.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

From TNF to TNF

One of the other important things you can do with TLC-format data is to transform one TimeLine Collaboration Normal Form (**TNF)** file into another. For example, you might want to automatically select only a handful of key metrics from each standard T4 CSV file and create a new file that showed only those metrics. Or, you might want to see only the time period from 9:30 to 11:00 and save a new file that included only those 90 rows of data. Or, you might want to carry out both row and column trimming to create a particularly compact new data set of the most important variables for the most important time period.

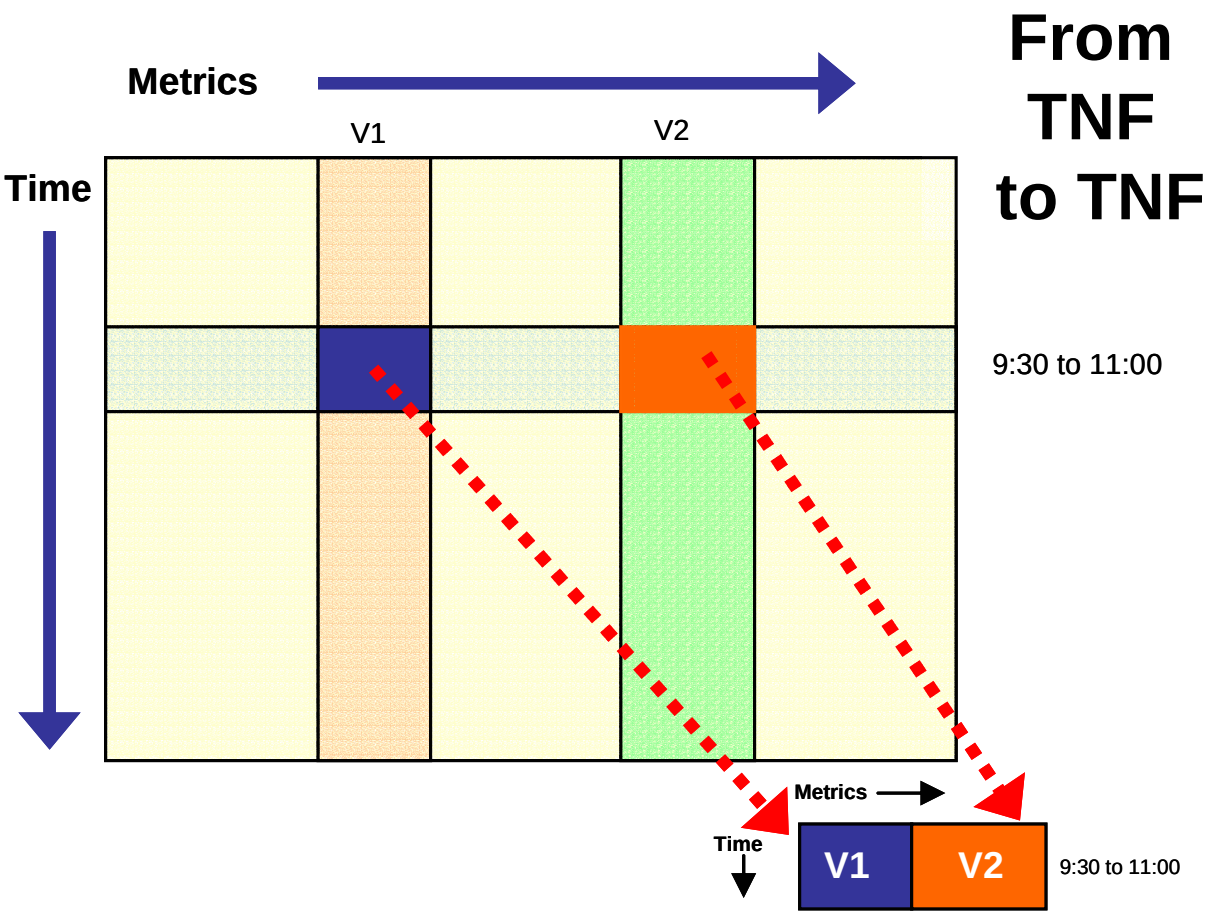

Figure 27: From TNF to TNF

Here we schematically illustrate the process of transforming one large TNF file into another, much smaller file. The total time might be a full day, but we are interested only in the period from 9:30 to 11:00. The total file might contain hundreds of metrics, but our interest lies only in the two metrics, V1 and V2. This kind of approach can be particularly useful when dealing with data from a large number of systems or when wishing to exchange key data sets with others using email or web access.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

All these kinds of steps are readily programmable when you start with files in TLC Normal Form. These newer, more compact files are well suited to email or for upload to central depositories for long-term archival storage.

When doing analysis and reporting, we have found that by trimming down our files to their key elements, the later portions of our analysis work are enhanced and speeded on their way.

In manipulating these files, we have talked earlier about combining and synchronizing rows of data drawn from a series of TLC-format data files. This happens automatically during the standard T4V3x collection step. The same idea could be used to create a composite file that showed key statistics from all nodes on a cluster as well as the aggregate across the cluster.

For other purposes, you might want to consider creating new TLC-format data by appending new rows to the bottom of an existing file. For example, you might try combining all the data for a full week of 60-second samples into a super-sized file containing 7 TIMES 1440 (or 10,800) rows of data.

The possibilities for downstream manipulation of TLC-format files to meet special new purposes are virtually unlimited.

What Customers and Partners and OpenVMS Engineers Have Already Done

OpenVMS customers, partners, ambassadors, and engineers have already carved out some of the possibilities for downstream use of TLC-format data. Here is a short list of some of the paths already in use or where the proof-of-concept has been demonstrated.

- **Near real-time data** – Harvesting data from collectors in near real-time, updating TLC-format CSV files, feeding the results to automatic graphing programs, and publishing them to the web on OpenVMS servers.
- **Consolidation** – Consolidating TLC-format data from dozens of systems and making it readily available at a central location to simplify and reduce the cost of performance management.
- **Massive Synchronization** – Integrating TLC data from T4V3x tool kit with Oracle TLC data, customer TLC response data, and customer TLC throughput data into a single, powerful, multi-dimensional composite picture of OpenVMS performance on mission critical production systems.

Possible Collaborative Improvements in the Future

Many opportunities await. While we have been making improvements and additions for over 3 years, there appears to be a long list of opportunities for squeezing even more value out of the T4 & Friends collaborative approach by adding new and important collectors to extend the standard collection kit, by crafting new downstream tools, and by encouraging an ever widening number of OpenVMS customers to join the party.

Some possibilities for the coming year include:

- Automatic trending over time (weekly, monthly, or yearly trends).
- Automatic consolidation and reporting of cluster-wide data.
- Development of a growing collection of expert rules for automatically examining large storehouses of collected data.
- Automation of central depositories for those with many systems to manage.
- Built-in capabilities for near real-time reporting.
- Built-in hooks to simplify publishing to the web.
- A whole host of new and powerful collectors that give us visibility into vital, but currently missing data and the addition of some of these new data streams to an updated T4 collection kit.
- Integrating some key collectors such as one for Rdb into the standard kit.
- More powerful capabilities for creating explanatory visual stories.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- Automatic report generation.
- Simplified adding of customer business timeline metrics to the standard T4 kit.

We would love to hear your ideas, plans, and accomplishments at extracting more and more value from TLC-format data.

Summary

Thanks to the widening use of the available T4 tool kits, we are now seeing steadily filling reservoirs of valuable TLC data on more and more OpenVMS production systems. Coupled with the increasingly numerous and powerful downstream Friends of T4, these databanks have so far produced dramatic, visible productivity gains for OpenVMS Engineering, for OpenVMS Ambassadors, and for an increasing number of OpenVMS customers and partners. The net result has been a dramatic improvement in our ability, our speed and our productivity in extracting real value from performance timeline data.

Progress in advancing this universal timeline-driven performance work on OpenVMS continues. Significant contributions are now beginning to flow in from our customers, partners and ambassadors. The year 2004 holds promise that we can build on the gains achieved in the past three years with our universal collaborative approach to OpenVMS Performance issues.

With the public availability of T4V2, T4V32, T4V33, and SHC, all OpenVMS customers and partners can now begin to join in, benefit from the advances, and contribute to future enhancements in collection and in value extraction.

The following points are worth noting:

- ✓ Any important source of timeline data can be harnessed to produce potentially synchronizable TLC data (either directly or by later extraction).
- ✓ TLC data is readily reusable and readily programmable.
- ✓ Complex systems exhibit complex, time-dependent behavior that often can best be understood by visually examining that behavior.
- ✓ The timeline graphics created in this way are accessible and understandable by both technical and non-technical viewers. The timeline graphics encourage communication, discussion, and collaboration among all the interested parties. We have found that tools such as TLViz make it possible for those with the most at stake in maintaining good system performance to conduct a first level review of the data on their own, and to question results presented to them, even if they are not themselves performance analysis experts.
- ✓ The growing reservoirs of TLC data mean that when questions arise, it's always possible to go back to the actual data for second opinions and further analysis.
- ✓ The time-saving T4 & Friends approach described in this article and all of the upstream and downstream tools we have developed have grown out of this set of principles combined with the central idea to use the currently abundant resources so as to conserve the scarcest resource – our personal time.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Acknowledgements – The Human Friends of T4

The rapid evolution of T4 & Friends owes its success to the work and creativity of many individuals. This has been a powerful on-going collaborative effort by the human "Friends of T4" including Tom Cafarella, Kevin Jenkins, Ian Megarity, Pat McConnell, Chris Brown, Matt Muggeridge, Paul Lacombe, Pat Moran, Michael Austin, Grant Hayden, Norm Lastovica, Debess Rogers-Grabazs, Guy Peleg, a whole raft of OpenVMS Ambassadors including Ray Turner, Dave Foddy, Kevin Fitzpatrick, Aage Ronning, and Jiri Kaspar.

Many, many others have played or are still playing active roles in the process. This includes: Sue Skonetski, David Klein, Kent Scheuler, Tom Moran, Peter Provencher, John Defilippo, Jim McLaughlin, Craig Showers, Christian Moser, Greg Jordan, Richard Bishop, Melanie Hubbard. Thank you all. Your efforts have really made a difference for OpenVMS.

Here's a brief rundown on some individual contributions to this evolving story.

Tom Cafarella (OpenVMS Engineering) wrote the Original T4 extractor tool in DCL. This gave us the proof-of-concept that we needed to continue. Tom has been an active user of T4 and TLViz and has recently created a prototype no-drift workaround for the current versions of MONITOR.

Ian Megarity and Kevin Jenkins (both from OpenVMS Engineering) demonstrated that with some effort, the Original T4 was reusable by porting it to a new system they were studying. Ian Megarity wrote the first code for combining TLC-format CSV files from different sources and built many new TLC-format collectors to grab key performance statistics not available from MONITOR. Ian then went on to create the powerful T4EXTR utility for squeezing even more timeline data out of MONITOR.DAT files. He followed with T4V32 and T4V33, greatly extending the scope of variables transformed into TLC-format tables with a total (today) of six TLC-format collectors, automated history creation, and many other advances you will find when you download the latest kit.

Kevin Jenkins actively applied each new T4 & Friends capability as he worked with OpenVMS customers on their most important performance concerns. His ideas on possible extensions and improvements often showed up in the next generation T4 kits and everyone benefited.

Ian Megarity amazed us all within OpenVMS Engineering when he revealed his Windows-based TLViz program for analyzing, graphing, and visually reporting on TLC-format data. It proved to be an instant success. It gave the analyst at least an order of magnitude productivity gain compared to doing similar analysis, graphing and visual reporting using a tool such as Excel. Many important activities that take multiple complex sets of keystrokes and mouse actions with Excel have been replaced by single simple keystrokes or single mouse clicks with fantastic time-saving benefits to the analyst.

Kevin Jenkins was again an early adopter and vigorous advocate of TLViz in his work. He found himself frequently opening up a pair of TLViz windows and comparing the graphs from one TLC-format data set with the graphs from a second TLC-format data set. Kevin's suggestion to Ian to let TLViz open up more than one data set and to overlay the timelines automatically has to rank as the single best T4 & Friends idea so far. Ian implemented this idea, literally overnight for the first prototype, and the resulting extremely powerful and easily generated ***Before-and-After*** graphs have proven they are worth their weight in gold for analysis and for explaining findings to others.

Pat McConnell (OpenVMS Engineering Performance Group Supervisor) has been another early adopter of T4 & Friends and a key management supporter of the work we have done. Pat has also prototyped several new tools and developed proofs of concept for new approaches to managing TLC data and making the best use of it. Pat has been instrumental in supporting advanced development efforts and experiments for possible new T4 collectors and extractors. In his own performance work in the eBusiness space, Pat has experimented with automated report writing driven by TLC data with output of text and graphs created in PDF. Pat has also begun to integrate the wider use of T4 collection into our QTV testing prior to release of new versions of OpenVMS.

Chris Brown (OpenVMS Director of Strategy) was an early and continuing management supporter of our T4 and TLC work as we used and developed successive versions of T4 while troubleshooting serious customer performance problems. Chris has been active in encouraging customers to turn on T4V3x

© Copyright 2004 Hewlett-Packard Development Company, L.P.

collection, so as to improve OpenVMS Engineering's abilities to collaborate with these customers about their most serious performance issues and concerns.

Matt Muggeridge (OpenVMS Engineering, Networking) turned the idea of creating a TLC-format collector for key TCP/IP data into reality in a few short weeks and worked with Ian Megarity to integrate this new collector into the standard T4V3x kit. Matt's work is a perfect model for how to build a new collector on top of existing capabilities of your particular layer of software and then design it so it fits like a glove with the T4 rules of the game. Matt has also been an active user of T4 collection as part of his network performance work, including a willingness to experiment with some of our barely tested new versions.

Pat Moran (HP Mission Critical Proactive Services and also an OpenVMS Ambassador) has made a tremendous recent contribution to the T4 & Friends repertoire with the creation of his command-line driven CSVPNG utility for massaging and creating new value out of TLC-format data sets. The powerful set of features and capabilities Pat has built into CSVPNG are sure to make this relatively new Friend of T4 into a big winner in the coming year. Pat has also done excellent proof-of-concept work with the use of PHP, APACHE, MOZILLA and CSVPNG to publish TLC-format data to the web and to explore doing this not only for historical data but also for near real time data as well.

Paul Lacombe (OpenVMS Engineering Director) provided the T4 team the management support and commitment and encouragement we needed to continue to develop and enhance T4 & Friends as a natural part of our day-to-day jobs. In the early days of T4, all of the OpenVMS engineers involved reported up to Paul and that made a big difference. Now, thanks to Paul's support, many others outside Paul's group are plugged in and are contributing to the evolving process. We are now, together, extracting more and more value from the TLC data being created on some of the most important and most mission-critical systems in the world.

Michael Austin (Cerner, Remote Hosted Operations) has proven how readily our best customers and partners can customize the basic T4 kit pieces and harness the power for their own special needs. Michael is doing amazing things with TLC data and with a variety of ever more powerful downstream tools. These are useful in their own right and serve as a demonstration and proof-of-concept for further extensions and elaborations to the base-level capabilities in the future. Michael has also made numerous suggestions for possible future enhancements to T4 collection and for useful downstream capabilities. For more details on what Michael is thinking about and doing related to T4, launch a GOOGLE GROUP search on "T4 VMS Austin" or on "What is T4 and what can it do for you"

Grant Hayden (Oracle) built the proof-of-concept Oracle Timeline collector, literally overnight. Thank you Grant for building the first TLC-format, database collector. Working together with Oracle and with one of our customers, we integrated data from this Oracle TLC collector with customer business data on sales volumes and with T4 collected system data from OpenVMS. This combination helped us delve deeply into a particularly complex and difficult set of performance problems.

Norm Lastovica (Oracle) demonstrated how TLC-format CSV files could be readily extracted from data captured by Rdb's standard monitoring utility RMU /SHOW STAT. As part of a collaborative Rdb-VMS benchmarking project, we have been making excellent recent use of this extractor --merging the Rdb TLC data with TLC data from T4 to create a more complete picture of our tests.

Debess Rogers-Grabazs (OpenVMS Engineering Performance Group) built many of the risky experimental and advanced development tools to test proof-of-concept for some of our next generation TLC collectors, extractors, and downstream value extraction utilities.

Guy Peleg (OpenVMS Engineering and an OpenVMS Ambassador) proved to be an active user of T4 and TLViz with a willingness to test some of our most experimental versions. Guy has had excellent success in using T4 and TLViz as a powerful customer collaboration tool.

A whole raft of OpenVMS Ambassadors were early adopters and supporters of T4 including Ray Turner, Dave Foddy, Kevin Fitzpatrick, Aage Ronning and Jiri Kaspar and many more.

Sue Skonetski deserves special thanks for helping us share the T4 story and the evolving TLC capabilities with so many customers and partners at Technical Updates and Bootcamps over the past two years. Those interactions helped provide much of the material for this article. Thank you, Sue.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

David Klein and Kent Scheuler (Cerner) – Dave and Kent were early adopters of T4 collection/extraction technology in their Performance Benchmarking lab. Dave and Kent have also been instrumental in the rollout of a Cerner unique timeline-driven approach that makes use of underlying T4V3x capabilities.

Tom Moran and Peter Provencher (OpenVMS Engineering) – Tom and Peter were active users of T4 technology in our HP/Cerner performance lab. This lab is a crucible for performance work with extremely heavy workloads and many unique and stressful tests. The continuous use of T4 technology in this lab built up a huge number of hours of airtime use for T4V32 collection and increased our confidence in the robustness and safety of using T4 collection more widely.

John Defilippo and Jim McLaughlin (OpenVMS Engineering) actively encouraged widespread use of T4 collection among the Cerner customer base and within Cerner itself. Lots of the success of our collaborative efforts to advance Cerner's unique timeline-driven approach to performance rests on John and Jim's shoulders.

Craig Showers (OpenVMS Engineering Customer Capabilities Lab). Craig has been an active voice encouraging those customers and partners who come in and use his extensive lab facilities to include a T4 timeline-driven component in the work whenever performance concerns were involved in the testing, Craig has been instrumental in spreading the word about the value of T4 to many new customers and partners.

Christian Moser (OpenVMS Engineering) deserves credit for his amazing work building a series of trace-based event collection tools that run in SDA and capture detailed views of vital, low-level OpenVMS performance metrics. These include, for example, his Spinlock Trace utility which can measure such things as which spinlocks are held for the highest percentage of time. We are exploring ways to begin to roll up this data (now that it has been revealed by Christian's trace tools) into a timeline-based approach, so we can examine the relationships between spinlock activity, other OpenVMS statistics and any other key metrics captured by other TLC-format collectors. Watch this space in the future for some more about this promising area of activity and collaborative synergy. Christian has also just added a No-Drift fix to MONITOR for OpenVMS Version 8.1, correcting this long-standing shortcoming.

Greg Jordan (OpenVMS Engineering) for his extension adding the ANALYZE capability to the Spinlock Trace utility. This automated some messy calculations and made key spinlock metrics such as IOLOCK8 hold time percentage immediately visible. We're now experimenting with a prototype TLC-format extractor that takes output from SPL ANALYZE and turns it into TLC-format CSV files.

Richard Bishop (OpenVMS Engineering) for his very recent addition to SDA of a WAIT command in OpenVMS Version 8.1. In the future, this will make the writing of TLC-format collectors that feed on SDA data much simpler, much more readily controllable at very high resolution, and substantially lower overhead. We anticipate we will be creating some prototype collectors that take advantage of this new feature as part of our performance work for OpenVMS on HP Integrity Servers. Because SDA has visibility into many important OpenVMS metrics that tools such as MONITOR never even dreamed of, there is lots of future potential to tap.

Melanie Hubbard has joined the T4 & Friends family in the last few months and has had the opportunity to help a growing number of OpenVMS customers using Oracle to consider putting T4V3x to use on their systems in a variety of different situations.

If I have forgotten to mention your contribution, my apologies. Please drop me a line and let me know so I can update this historical record and correct it in future work.

For More Information

For more information about T4 & friends, or if you have interest in gaining access to some of our internal downstream tools, please contact the author at steve.lieman@hp.com.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Cluster Test Manager (CTM) for OpenVMS

Richard Stammers
Software Engineer, OpenVMS

Overview

This paper discusses CTM, the OpenVMS Cluster Test Manager. CTM is one of the principal tools used internally for testing OpenVMS in large and high-risk cluster configurations. CTM consists of two parts: the software that is used to manage and control the running of the tests, (known as the CTM test harness), and a variety of tests that are specifically designed and written to be run by CTM (known as CTM test modules).

This paper discusses the CTM test harness, and discusses in general the goals, strategy, and internal design of CTM. Later papers may discuss various CTM test modules.

Introduction

Testing OpenVMS can be very challenging. Not the least of these challenges is the sheer volume of work that is involved in thoroughly testing everything. This is particularly true in a cluster environment, where the possible permutations and combinations of hardware types, versions of the operating system software, and all the other software and their versions, can be gigantic. Providing the means to expedite such a large volume of testing work was one of the primary reasons why CTM was developed.

CTM is not a test as such. Rather, CTM provides a framework, or "test harness," in which a large variety of tests can be efficiently controlled and managed in an OpenVMS Cluster environment. CTM provides a comprehensive means for managing such testing efforts on large and complex OpenVMS Clusters, where simply using DCL commands or BATCH to run the tests would be impractical. Using CTM hundreds of tests can be distributed across a cluster, and can be started and stopped with just a single command. CTM test runs can involve several thousand varied test processes when CTM is used on large test clusters.

Of course, testing involves more than merely running a lot of tests. The quality and nature of the tests are crucial. However, the way that the tests are sequenced and managed can be just as important to the quality of the testing as the tests themselves. With so much testing work to be done, the testing process can be very much a race against time, so it is essential to use the available testing time as efficiently as possible. In general, problems are much more likely to occur when a system or cluster under test is in a state of flux and is subjected to frequent changes, as against being in a "smooth" or steady state with respect to the tasks that are running. OpenVMS software has a natural tendency to smooth out the load on a system. This is very desirable for normal operations, but it might not give rise to the best conditions for testing. For example, if a group of tests are run continuously for, say, 10 hours and do not detect a problem, this does not in general provide 10 times the confidence level that running those same tests for just 1 hour would provide. In fact, the 10 hours of testing time can generally be used to better effect by constantly varying and modifying the tests that are run during the 10-hour test period. CTM provides the means to do this, "stirring the mix" of the testing, so to speak, by permitting repeated and continuous starting, stopping, and modification of the text mix throughout the test period.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Finding problems is only half the battle. The cause of a problem has to be isolated and identified so that it can be fixed. This can be very challenging in a large and complex cluster, especially particularly because the various components and subsystems of the OpenVMS operating system are so highly interrelated. It might seem rather a simplistic approach, but often the most viable initial testing strategy is to subject the systems under test to extreme stress testing. That is, subject the systems to a veritable barrage consisting of as large and complex and varied a sequence of tests that they can reasonably be expected to sustain. The idea is to "bring them to their knees," as it were, just to "see if anything breaks." When a problem is encountered, it can require considerable knowledge and skill to identify its cause. The test manager has to be able to meet this challenge, providing the means whereby the subsequent testing can be organized and sequenced to progressively home in on whatever is causing the problem and to provide the information required to fix it. This means that CTM not only has to provide the means to run a very large and complex mix of tests, but also has to provide the capability to be extremely precise in the selection and sequencing of tests in order to identify and fix problems. A corollary to this is that, with such an extreme degree of interrelationship between the components, if *anything* is changed or fixed, then *everything* needs to be retested.

Overview of CTM

CTM is designed to operate in a cluster environment. In general such clusters may comprise a mix of VAX and Alpha nodes, with CTM providing seamless functionality and testing capability across both the VAX and Alpha platforms. Similarly, CTM provides backwards compatibility for all the previous versions of OpenVMS that are supported and might be present in the cluster. At the time of this writing, a new version of CTM, providing similarly seamless capabilities for HP OpenVMS Industry Standard 64 Evaluation Release Version 8.1 (OpenVMS I64) is in development. CTM can run on a wide range of cluster sizes, varying from clusters comprising a single node through to clusters comprising 60 or more nodes with more than 600 storage spindles.

Perhaps the most fundamental aspect of the design of CTM is that it views the cluster as a whole as a collection of testing resources. In this context, a "testing resource" is any item of hardware within the cluster that can be used for testing. The current CTM implementation has two main categories of testing resources: nodes (single CPU systems or SMP systems) and storage devices (disks or magnetic tapes). This concept of testing resources is key to the whole design. CTM automatically maintains a dynamic database of testing resources that are available on the cluster on which it is running. A separation is made between the hardware resources on the cluster that CTM is permitted to use for testing, and those resources that are protected from CTM. All the resources on the cluster are potentially available for use by CTM. The test engineers are provided the means to dynamically control those resources that are to be used and those that are to be protected. For example the system disks are usually protected – in general, it is not a very good idea to do I/O testing on the system disks!

The tests that are run by CTM have to be specifically written to work with the CTM test harness, and are referred to as CTM test module. At present there are about 25 such test module in regular use, and these are constantly being updated and added to. Again, the concept of test resources is fundamental to a CTM test module. Within CTM, test modules are characterized as requiring certain test resources in order to run. For example, CTM_HIGH_IO is a disk I/O test that repeatedly writes, reads and verifies data to disk. It is characterized as requiring one CPU resource and one

© Copyright 2004 Hewlett-Packard Development Company, L.P.

disk resource. Similarly, CTM_TCP is a TCP/IP test that causes TCP/IP packets to be exchanged between CPUs within the cluster. It requires two CPU test resources.

The interaction between the resources that are available to CTM in the cluster and the resources that are required by each of the tests is the basis of the CTM design. The cluster resources are described in two databases – those that are available to CTM, and those that are protected from CTM. A weak pun is used for the naming of these databases that are called, respectively, the CARE (CTM Active Resource Entry) and DONT databases. The resources that each of the s module require are maintained in what is referred to as the Test Module Data Base (TMDB). An additional file, the load-numbers file, contains a numeric description of the performance characteristics of each all the different resource types that could be present in the cluster. Using the information in these files and databases, CTM can automatically start up, load balance, and manage large numbers of tests with very brief commands from the test engineer. The following command examples illustrate this.

CTM Command	**Action**
$ CTM START CTM_HIGH_IO /process=100	Starts 100 copies of the CTM_HIGH IO test, automatically balancing the load across each of the nodes and disks that are available in the cluster.
$ CTM START CTM_HIGH_IO /perdisk=4	Starts 4 copies of the CTM_HIGH IO, targeting each disk in the cluster, and automatically selecting the CPUs that will perform each of the tests.

In fact, the CARE and TMDB databases contain quite a detailed characterization of the test resources that are available on the cluster and the test resources that are required by a CTM test module. This information provides the test engineer with a considerable range of control for specifying how tests are to be run, ranging from the examples shown where CTM is used to automatically select and load balance the resources that are used, to detailed selection of the characteristics of the resources that the tests will use.

Test engineers interact with CTM through the CTM command center. The CTM command center provides an extensive command-line interface that lets test engineers control the starting and stopping of tests, monitor tests in progress, control test resources in the cluster, generate test reports, and perform all the other functions associated with controlling CTM and the running of CTM tests within the cluster. For convenience and redundancy, and because the component nodes within the cluster may be physically separated, CTM permits any node in the cluster to function as the command center. In fact, there may be multiple command s active on the cluster at any one time (one per node in the cluster). However, at any given time only one command center can act as the master command center, which is capable of issuing action commands that change the state of CTM, such as starting or stopping tests or modifying the characteristics of the test resources within the cluster. Other, nonmaster command s may be active but are confined to issuing interrogative commands, such as generating reports or getting information about active tests.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Organizing and identifying CTM tests

It is not uncommon for a CTM test run to involve running 3000 or more test processes. With so many tests being run it is essential that each tests is uniquely identified so that it can be properly managed and identified. CTM uses a job and process number scheme to identify the tests that it runs. In CTM nomenclature, each line to the command center that starts up test processes is referred to as a job. Job numbers revert to 1 each time CTM is restarted on the cluster. When a job involves starting multiple test processes, each test process within the job is allocated a sequential process number, starting at 1. Hence, the job and process number are always unique for all test processes that CTM is running at any point in time. These unique job/process numbers are referred to CTM as a DPID.

In the preceding example, if the command to start up 100 CTM_HIGH_IO processes were the first job, this would generate 100 test processes with DPIDs 00010001, 00010002, through 00010064 (hexadecimal), respectively. CTM uses the DPID for a variety of purpose. For instance, it is used as the OpenVMS process name on the nodes on which the test runs, is used to generate the names of whatever files the test process create, and is used as part of any data patterns that the test uses.

Viewing CTM test results

In general, every test process generates a test log. All test logs are placed in an area defined by the CTM$LOGS directory, which is accessible to every node on the cluster that is running CTM. The DPID is used to provide a unique name for each test log. The log usually contains header information that identifies the test, the test resources it was using, and the parameters that were used to invoke the test, followed by whatever other log information the test may generate during the test.

Once a CTM test is started, it runs until it is either explicitly stopped by the test engineer, or until a fatal error condition is detected. If a fatal error condition is detected (for example, a data corruption) the test usually generates an error or corruption report and then stops.

As they run, the CTM tests also generate performance and other types of information about their test run, which they periodically send via a mailbox to the TELogging process. This process exists on every participating CTM node in the cluster. It is responsible for adding the information that the test tasks send to it into the TEL data files, which are a common repository for information on all the tests that are running or have been run by CTM on the entire cluster

CTM provides a utility known as the TRU (TEL Reporting Utility) that can be run on any node in the cluster at the behest of the test engineers. The TRU generates reports based on the information in the TEL data files, and permits this information to be organized and presented by a variety of different criteria, such as time, or device or device type, or node or node type. This allows the test engineers to see what is happening on the cluster as a whole as well as providing the means to monitor the performance of individual systems under test, devices, or test processes.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Data patterns

There are no real constraints on the functionality of the tests that can be run within CTM, as long as the test complies with both the conventions required to communicate with the CTM test harness, and the needs and inventiveness of whoever is writing the test. However, a theme that is common to many of the tests is the transfer of data. For example, the CTM_WIDEST test involves transferring data between different areas of memory; the CTM_HIGH_IO test involves transferring data to and from disk files; and the CTM_TCP test involves transferring TCP/IP packets between different nodes on the cluster. The common theme is that a data pattern is generated, the data is transferred, and the results of the transfer are checked and validated. Data corruption is probably the most serious and most dangerous problem that can ever occur – much worse, in fact than a process or system failure because of the potential to destroy user data without warning. Thus, great importance is attached to the data patterns that are used, and the requirements for the data patterns are quite demanding:

- The data pattern should provide the best possible chance of detecting corruptions.
- Wherever feasible, the data pattern should provide as much evidence and as many clues as possible as to what caused the corruption
- The data patterns should allow quick construction and verification, so that as many test iterations as possible can be performed within a given time.

Many techniques are employed within the CTM tests to meet these requirements. In general, each test uses its unique DPID within the data pattern so that if a crossover of data occurs between test tasks, the source of the offending data can be identified. For example, if multiple CTM_TCP tasks are exchanging packets, and a packet is erroneously sent to or received by the wrong recipient, the source of the bad data can be identified. Similarly, sequence numbers are often used within the data patterns so that dropped data can be identified. When writing to disks, the block number and position of the data fields within the blocks are usually incorporated into the patterns in order to provide more complete information about the nature of the corruption. Various schemes are used to vary the positions and alignment of the data buffers with respect to OpenVMS pages and disk blocks.

Speed is also of the essence, with respect to both creating and verifying the data patterns. The usual technique is to build all the data patterns or as many of them as possible at the start of the test so that time is not wasted on this during actual test iterations. Fields within the data patterns are usually organized and aligned on quadword or longword boundaries so that the verification can be done very briskly, and routines to do the compares and verification are frequently written in assembly language for the same reason.

When elements of the data patterns are common between successive transfers, the transfer may not actually take place properly, but because of the stale data in the receive area, a corruption might go undetected. Poisoning the receive areas before the transfer is one way of avoiding this, but this technique is avoided in the CTM tests because of the time it takes. Instead, a technique of alternating the data that is transferred with the 1s complemented form is used. In this way, literally every bit of the transferred data is changed on each successive transfer, and the performance hit is taken only once, up front, when two forms of the data pattern – the “true” data and the 1s complemented form are built.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Tracking down problems

Normally, a CTM test runs until it is explicitly stopped by the test engineer, or until a fatal error is detected. Almost all errors that the tests detect are considered to be fatal.

If a fatal error condition (such as, a data corruption) is detected, the test usually generate an error report and then stops. The nature of the error report depends on the nature of the test, but where the test involves a data transfer, a corruption report is always generated. To simplify the debugging process, the corruption reports usually show the expected data alongside the actual data, with the incorrect or corrupted fields flagged so that they can be readily identified. As described earlier, the data patterns often involve a 1s complemented form of the "true" data, and because humans are not very good at reading 1s complement, a translated hexadecimal form of the data is provided for convenience.

For some tests, a corruption report is not always sufficient to debug the problem. For example, with the CTM_TCP test or the CTM_LOCK_IT test (a CTM test that stress tests the Distributed Lock Manager), a "history of events" leading up to the error is often required to debug what is going wrong. In these instances, the test maintains a ring buffer that describes events leading up to the failure, which is output before the test is stopped.

In extreme cases, a crash dump may be needed to debug the problem, and for some tests a parameter is provided that is used to that specify that a bugcheck is to be performed when the problem is detected. If a bugcheck is needed, then the ensuing crash dump is most valuable if it is produced as soon as possible after the problem is detected. In these instances, no reports are generated.

In very extreme cases, a bugcheck might be required on multiple nodes on the cluster in order to debug a problem. For example, a CTM_HIGH_IO test might be running on one node in the cluster, targeting disk I/Os to a disk that is served by a different node on the cluster. When a problem is detected it might require a timely crash dump of both these nodes in order to figure out what is going wrong. CTM uses a technique known as "triggering" to accomplish this.

The way that triggering works is that a "trigger arming" routine is provided with CTM that permits a node to arm itself on a trigger. There are 256 such triggers available. In the preceding example of the CTM_HIGH_IO test, the node that was serving the disk would be armed on a trigger. When a node arms on a trigger, it uses DLM to take out shared access to a lock that is associated with the trigger, and specifies a blocking asynchronous system trap (AST) to fire when this lock is lost. The blocking AST routine performs the bugcheck. When the node that is conducting the CTM_HIGH_IO test detects the corruption, it takes out exclusive access to the lock in question, thereby causing both the blocking AST to fire on the node that is serving the disk and a timely crash dump to be generated.

Tracking down problems in a large and complex cluster can sometimes become very complicated indeed, particularly if the problem is intermittent and occurs very infrequently. The problem might be caused by any one of a number of components, or by the interaction between multiple components that are distributed across the cluster. Triggering permits multiple nodes in the cluster to be made aware very quickly when a test process on any given node detects a problem. In such instances multiple different nodes can each be armed on a number of triggers. Even a crash dump

© Copyright 2004 Hewlett-Packard Development Company, L.P.

might not be sufficient to figure out what is going wrong. For this reason, the functionality that is invoked by triggering on the nodes can be changed to do things other than bugcheck by changing the functionality within the blocking AST that is fired when a node is triggered. For example, the blocking AST might be modified to output a data pattern that can be identified on a data scope or bus analyzer when a problem is detected and the node is triggered.

Varying test mixes and sequencing tests.

As suggested previously, the most efficient use of test time -- certainly in terms of surfacing particularly nasty problems -- is often achieved by varying the loads and changing the mix of tests that as much as possible on the systems under test. Although it's difficult to be very specific about what constitutes a "nasty problem," such problems leave little doubt as to their nature when they have to be fixed. They also seem to have certain common characteristics. For instance, they tend to happen intermittently and infrequently. Often the way they manifest themselves is not directly related to what caused them and provides few clues as to what is actually causing them. The problem may not be detected until some time after the original problem event occurred by which time the context that caused the problem has changed. Sometimes they happen as a result of an unusual combination of circumstances or interactions between components that are otherwise thought to be very solid and reliable. The effects can be very serious -- data corruptions, system failures, and so on.

The observation that these types of problems are most likely to be found by varying the testing mix as much as possible is borne out by both experience and empirical results. However, it is not hard to also think of theoretical reasons why this should be the case. A large OpenVMS Cluster is an amazingly complex system, and looking for problems in it can be likened to searching an immense combinatorial tree of all the possible states that the cluster can be in. The greater the variety of the tests and more the tests are mixed, the greater the area of the tree that will be searched. Hence, the increased probability that a problem will be found. On a more prosaic level, these types of problems often occur when unusual events occur, such as infrequent timers expiring, queues becoming full, or buffers overflowing. Continually stirring the test mix and spiking the loads tends to make these types of things happen more often than when the systems under test are left in a relatively constant state.

CTM provides the capability to mix tests, spike loads, and otherwise "torture" the systems under test by means of what are referred to as pulses. Logically, a CTM pulse can be thought of as a square wave that alternates indefinitely and with a certain specified periodicity between two states – blocking and free. A CTM pulse is shown schematically in Figure 1.

The pulses are implemented by means of the CTM pulse utility, which enables up to 256 such pulses to be defined and made available to every node in the cluster. Each pulse is completely independent as to when it starts and the amount of time it will subsequently block and free tests. When a CTM job is started, any of the defined pulses can be specified, with the effect that all the test processes associated with that job will stop and start in accordance with the specified pulse.

The pulse functionality is implemented by means of DLM. The pulse utility creates a lock that is associated with each pulse that is defined. It blocks by converting to exclusive access on the lock, and frees by converting to null access on the lock, in accordance with the timings specified in the definition of the pulse. Tests processes that have been instructed to synchronize to the pulse

© Copyright 2004 Hewlett-Packard Development Company, L.P.

precede each test iteration with an attempt to convert to shared access to the lock, and hence run only when the pulse is free and are stopped when the pulse blocks.

Because each job can use any one of 256 different pulses, there is considerable flexibility for varying the mix of tests that are running at any one time. In addition, because the state of the locks is so rapidly communicated across the cluster, they provide a means to generate very sharp spikes in the load to which the systems under test and the cluster as a whole are subjected.

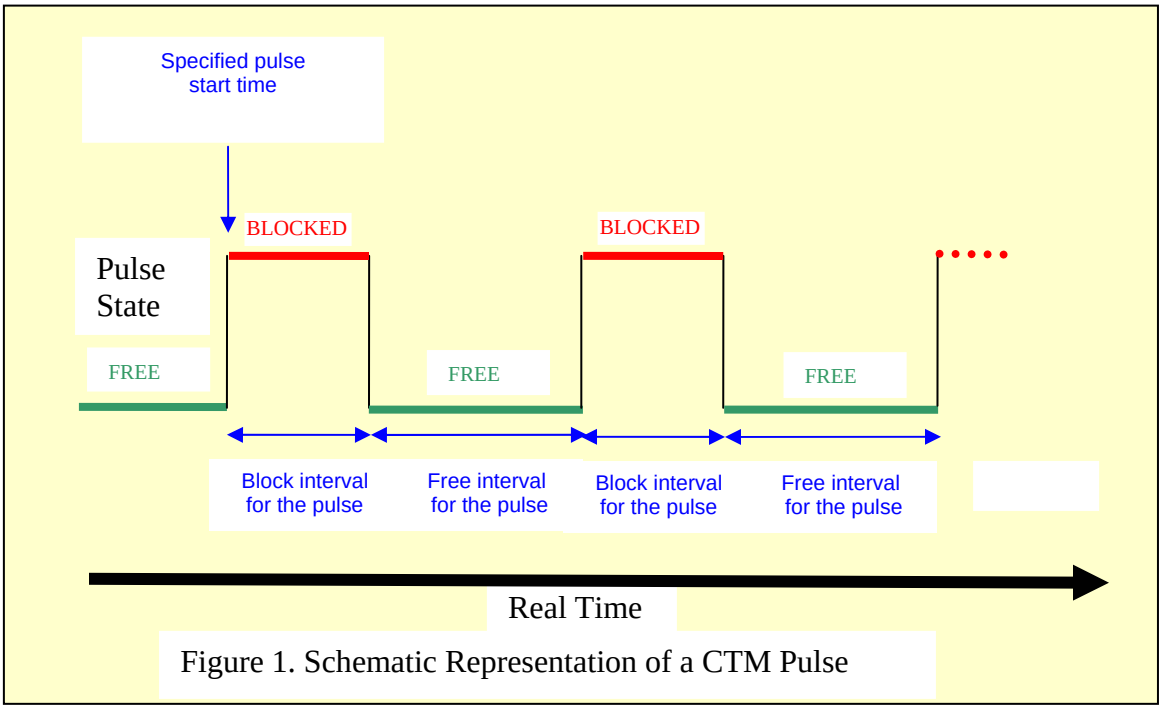

Figure 1. Schematic Representation of a CTM Pulse

Pulses can also be used as a diagnostic aid to isolate and identify components and combinations of components that cause problems. This is accomplished by successively "switching in" tests that use different components until the problem is detected, thereby allowing the culprits to be determined by the process of elimination.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Overall CTM design

Figure 2 provides a schematic view of the design and principal components of CTM.

In order to provide the highest degree of fault tolerance, the CTM harness software is based on a heterarchical rather than a hierarchical design. That is, there is no single "control" node or control process within the cluster that controls the running of the tests. Therefore, CTM as a whole is not vulnerable to failures on any given node in the cluster. Instead each node participates as a CTM peer within the cluster and each is capable of and required to perform all the tasks required by CTM. This redundancy is very important because CTM is often used to test high-risk clusters where some of the components are in early testing.

A participating CTM node always has the CTM server process, and the TELogger (Test Event Logger) process running. The CTM command center process is run whenever a participating node is explicitly called upon to act as the command center. The TRU (Test Report Utility) process is run whenever a participating node is explicitly called upon to generate a test report, and the Test Module Data Base (TMDB) utility is run whenever a participating node is called upon to modify or interrogate the TMDB.

The CTM related processes that run on participating CTM nodes communicate and share information by means of common databases and directories that are pointed to by logical names that are common to all participating CTM nodes in the cluster. Access to these databases and directories is controlled and synchronized, as required, by the Distributed Lock Manager (DLM). DLM is also used extensively for signaling and synchronization between the various processes that constitute CTM running on each of the participating nodes.

Note that the CTM "databases" are not databases in the usual sense, but rather are specially constructed RMS files. The OpenVMS File Definition Language (FDL) utility is used to create many of them.

CTM databases and related files

A number of directories, databases and files are involved in the implementation of CTM within a cluster, and are shared by and common to all the participating CTM nodes. The principal ones are as follows.

CTM$LIBRARY
This area contains the load-numbers file mentioned earlier, and all the executables, command files, and other files that constitute the CTM test harness. In general, there are separate versions of each of these files for the VAX and Alpha platforms. When any CTM-related process is to be run, the type of platform is identified and the appropriate version for that platform is used.

By convention, any corruption and error reports that a test generates are usually placed in this area as well.

The OpenVMS I64 version of CTM that is in development is very similar as regards the directories, databases and files that are used, but instead contains versions of the CTM test harness files that are appropriate for the OpenVMS I64 platform.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

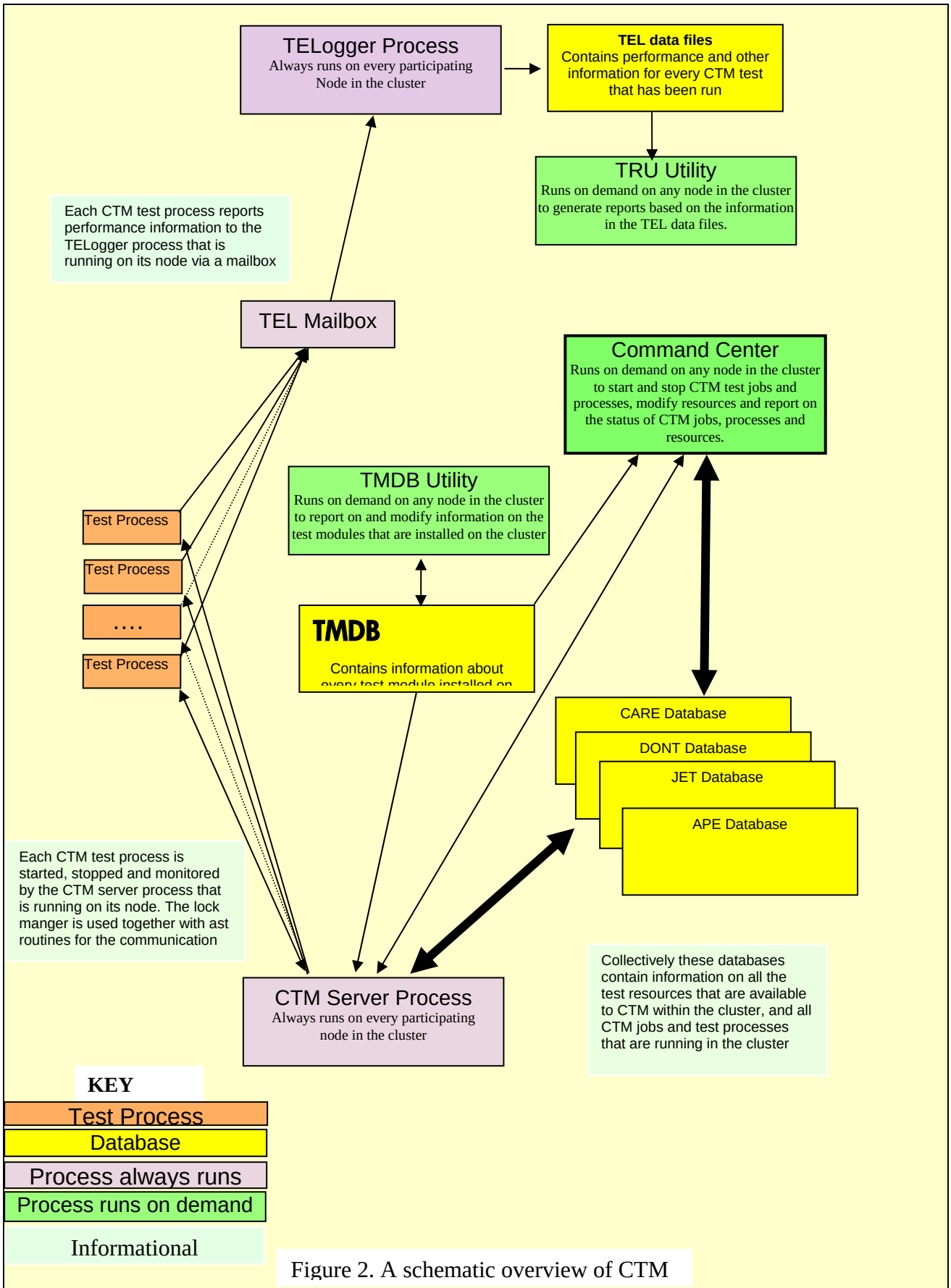

Figure 2. A schematic overview of CTM

© Copyright 2004 Hewlett-Packard Development Company, L.P.

CTM$TMROOT
This area contains all the executables, command files, and other files that are required for whatever CTM test module are installed on the system. Again, in general there are separate versions of each file for the VAX and Alpha platforms; when any CTM test module related process is to be run, the type of platform is identified and the appropriate version for that platform is used.

Similarly for the OpenVMS I64 version of CTM that is in development this area contains versions of the CTM test module that are appropriate for the OpenVMS I64 platforms.

CTM$DATABASES.
This area contains the major CTM databases that are used by the cooperating CTM processes within the cluster to record the test resources that are available to CTM, the test jobs and test process that are in progress, and the test resources that are being used by them. Access to these databases is protected and controlled by means of DLM.

CARE and DONT Databases
These databases are used by CTM to record the test resources that are available within the cluster and to record whatever cluster resources are protected from CTM. CTM automatically "probes for" and maintains a record of whatever test resources are available. Protected resources are explicitly controlled by the test engineers, either from the command center or by appropriately modifying the CTM startup sequence.

APE and JET Databases
The Job Entry Table (JET) database and the Active Process Entry (APE) database are used by CTM to record the tests that are in progress and the test resources they are using, and are maintained automatically by CTM. As described earlier, in CTM nomenclature each command issued at the command center is considered a job, and the various test processes that are started for that CTM job are considered processes within that job. The job/process number is always unique for all the test processes CTM is running at any point in time.

There is no correspondence between a job and a process as understood by OpenVMS, and a CTM job or process – other than each CTM job/process gives rise to a unique OpenVMS process on whatever node or nodes involved in that test. These tests take the DPID as the OpenVMS process name.

Loosely speaking, the JET database contains information about the jobs that CTM is running, and the APE database contains information about the processes that CTM is running and the test resources they are using. However, there is a lot of cross-pointing and shared information between the CARE, DONT, JET, and APE databases.

CTM$TMDB
This area contains the TMDB database that describes all the test module that are installed on the cluster, a characterization of the resources they require, and other information pertaining to how they are to be run.

TEL Data Files
As each test process runs, it may periodically report metrics on the performance of the test as well as other information about the progress of the test. All such information is collected in the TEL, or Test Event Logging, files. All TEL data from all the tests that have run since CTM was installed is kept

© Copyright 2004 Hewlett-Packard Development Company, L.P.

in TEL data files. This data is cleared only when CTM is reinstalled or when the database is explicitly reinitialized.

CTM$LOGS
This area is a repository for the logs that each of the tests produces. The test logs are named simply by the unique DPID of the test process to which they relate.

In addition, each of the CTM server process that are running on all the participating nodes in the cluster maintain their own logs, which are also kept in this area.

CTM test harness software

Normally, on a cluster where CTM is to be used, a CTM$STARTUP command file is invoked as part of the system startup procedure for every node in the cluster that is to participate in the CTM testing. This command file defines the CTM related logicals as described earlier, and installs the CTM server and the TELogger images on each of these nodes.

Subsequent logging in to any of these nodes as CTM invokes a LOGIN.COM file that uses the Command Definition Utility (CDU) to define the command CTM to run the CTM command center process and to define the syntax of all CTM commands. The commands and their syntax are quite extensive, reflecting all the functionality that is provided by the CTM command center.

Similarly, the commands and the syntax for the TMDB utility (which is used to modify and update the Test Module Data Base) and for the TRU utility (which is used to generate test reports) are defined at this time.

A general mechanism and technique is employed by which all the cooperating CTM processes signal and communicate with each other across the cluster. DLM is used extensively, and each cooperating node creates a series of locks whose names are predicated on the node's name in the cluster. Each node prepares itself to be signaled by taking out shared access to its own locks, specifying blocking ASTs that are to fire when they are signaled. Other nodes that wish to signal that node then initiate communication by taking out exclusive access to the appropriate lock for the node they wish to signal, thus causing that nodes blocking AST to fire.

The CTM Command Center Process
This process can be run on any participating node in the cluster. It provides a command-line interface (CLI) that allows the test engineer to control and interrogate CTM. The first instance of this process being run becomes the CTM command center master. As such, the CLI provides a command set that provides the following functionality.

- Starting (booting) and stopping CTM as an active participant on each nodes in the cluster
- Making test resources available to CTM or protecting test resources from CTM
- Mounting and dismounting the storage devices that will be used as test resources by CTM
- Starting and stopping the CTM jobs and test processes
- Interrogating CTM about what test resources are available, the CTM jobs and test processes that are running, and the test resources they are using
- Generating reports from the TEL data files

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Booting CTM

As the command center master, the CLI provides a command set that permits CTM to be stopped and started (booted) as an active participant on each node in the cluster that has successfully completed the CTM$STARTUP sequence. Normally, the first operation that is performed at the command center is to boot CTM on one or more nodes in the cluster.

Booting CTM involves several stages, one of which is to start up the CTM server and TELogging process on each node to be booted. The general mechanism by which CTM processes use DLM to signal each other across the cluster is used. Another task involved in booting CTM is to create or update the CARE database (the database of resources that are available to CTM). This is done by a sequence of wildcarded calls to SYS$GETSYI and SYS$GETDVI to identify the resources that are available on the cluster. Similarly, the JET and APE databases are also initialized to show that there are no jobs currently being run by CTM.

Controlling CTM resources

Requests to change the protection of test resources are handled by making the appropriate changes to the CARE and DONT databases. If the changed protection affects a resource that is in use by a test that is already running the status of that test can also be modified in the APE and JET databases. Commands to mount and dismount storage devices are handled in a similar fashion, and the node on which the device is being served is signaling to perform the mount or dismount in batch.

Starting and stopping CTM jobs and test processes

To start a job, the command center master is involved in varying amounts of work, depending on which test resources the test engineer has unambiguously specified in the command. The resource requirements for the specified test are first established from the TMDB. If all of these resources are unambiguously specified in the command, then details of the job are placed in the JET database, and the node that was specified is signaled to start the test processes. If the test resource is not specified clearly, the command center must determine what test resource to use. It attempts to load balance by referencing the tests that are already running (as described in the APE and JET databases), the load-numbers file, and various internal algorithms. Once the test resources to be used are identified, details of the job are then placed in the JET and the selected nodes are signaled to start the test processes. Handling commands to stop test processes is much easier; the nodes that are running the test processes are simply identified from the JET database, and then are signaled to stop the test processes.

Interrogating CTM and generating reports

When the command center is already running, any subsequent invocations of CTM give rise to a nonmaster version of the command center. As nonmaster the command CLI provides a reduced command set that is limited to generating reports and interrogating CTM about the resources that are available and about the jobs and test processes that are running. Requests for information about CTM resources and jobs and processes that are running are handled by examining the CARE, APE, and JET databases. Requests for reports are handled by calling the TEL Reporting Utility to generate the requested report based on the contents of the TEL database.

The CTM Server Process

This process runs on every participating node in the cluster on which CTM has been booted. Its primary purpose, based on commands signaled from the command center, is to stop and start CTM

© Copyright 2004 Hewlett-Packard Development Company, L.P.

test processes on the node on which it is running, and to periodically monitor whether these tests are still alive.

Starting a CTM test process involves several stages. The TMDB for the test process is examined, and the required test resources are verified and are assigned to the test process. The OpenVMS context in which the test is to run is established and is described to the test by a series of process local symbol definitions. For example, the DPID that the test is to use is defined as a local process symbol. Similarly, if the test is required to create data files, a directory for these files is created on behalf of the test and is also defined as a process local symbol. Finally, the test process is started by creating an OpenVMS process to run a DCL command file that is specific to the test.

The CTM server process is responsible for maintaining the status in the APE of all the test processes that are running on the node. Because there can be so much work occurring on the test nodes, they are often extremely busy and there can be significant delays in the communication between the CTM server process and the test processes it is running. After each test process is started, it is "pinged" periodically by the CTM server process to ascertain that it is still alive. When the first correct response to the ping is received back from the test process it is flagged as RUNNING. If a test process fails to respond to the pings in a timely fashion, it is eventually flagged as MIA. When a test encounters a fatal error and stops, the CTM server process is notified and the test is flagged as DEAD. Stopping a test at the behest of the command center can also involve significant delays if the system is very busy. The test's ping is changed to a request to stop, and the test is flagged as FIP (finish in progress). When the test process eventually stops, the CTM server process is notified and the test is flagged as FINISHED.

As long as CTM is booted and running on a node, a secondary function of each CTM server process is to wake up periodically to verify that the contents of the CARE, APE, and JET databases are accurate and up to date. Access to each of these databases is controlled by the acquisition of DLM locks. If the state of the locks indicates that a CTM server process on another node is verifying the databases, no action is taken, and CTM relies on the functionality of the peer server process to maintain their accuracy. If not, the databases are "walked" and verified by issuing a series of SYS$GETSYI, SYS$GETDVI and SYS$GETJPI system service calls to verify that the contents of the databases are accurate and up to date.

The TELogger Process

This process runs on every participating node in the cluster on which CTM has been booted. Its function is to accept messages, such as performance metric information from the test tools, and to write that information to the TEL data files.

The Test Module Data Base Utility

The TMDB utility is used to create, maintain, and report information about the CTM test module that are installed on the cluster. It is run on demand, and it can be run on any participating node in the cluster. It handles such information as the name of test module, a characterization of the test resources used by each of the test module, and other information pertaining to how the tests are to be run.

The TRU Reporting Utility

This utility is used to create generate reports from the data that is contained in the TEL database. It is run on demand, and can be run on any participating node within the cluster.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

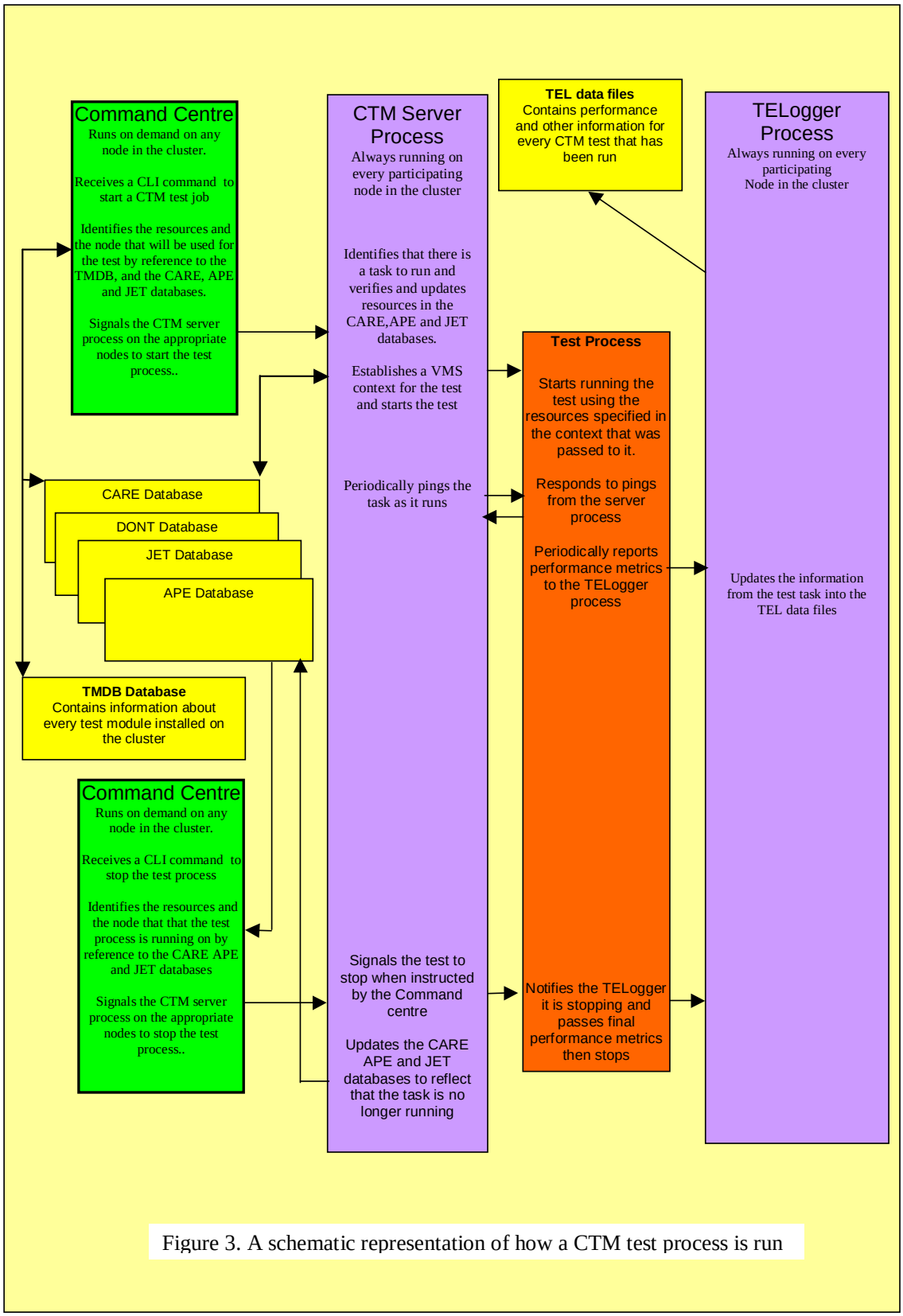

Figure 3. A schematic representation of how a CTM test process is run

© Copyright 2004 Hewlett-Packard Development Company, L.P.

CTM Test Modules (Test Processes)

Figure 3 provides a schematic representation of how a test process is run by CTM.

The primary function of a CTM test module is, of course, to perform whatever actual testing functionality is required. Several such test modules have been mentioned in the foregoing description of CTM. For example, the CTM_HIGH_IO test performs disk I/O testing by means of repeatedly writing, reading back, and then verifying data patterns to and from disk. Because CTM test modules have to be written specifically to work within the CTM test harness, they also must perform certain additional tasks.

All test processes are initiated from the CTM command center, which resolves any ambiguity concerning the resources that the test is to use, including the node on which the test is to run. The CTM command center then signals the CTM server process on the selected node to start the test. The CTM server process on the signaled node establishes an OpenVMS context for the test process and then starts it.

In order to interface with CTM, the test module must use the test context provided by the CTM server process by using parameters that are passed to the test by the CTM server process. These parameters establish the process logical names that describe them. By convention, the DPID that is established by the CTM server process and that is passed as a process logical name is used in building whatever the data patterns the test may use. Similarly, the directories and file names that the test module uses must comply with what the CTM server process tells the test to use.

The test is also required check in periodically with the CTM server process and to respond periodically to pings from the CTM server.

As the test process runs it periodically sends the TELogging process that is running on that node performance metric information for the test. The TELogging process places this information in the TEL data files, where it can be accessed and reported on in the context of other activity taking place on the cluster. This confers a great deal of standardization to the reports that are generated and results in a considerable saving of effort by eliminating the onerous task of writing lengthy report-generating programs for each test.

Assuming that the test does not encounter a fatal error, it is eventually stopped by an explicit command from the CTM command center. In turn, the CTM server process signals the test task to stop. Upon receiving this signal, the test task makes one final report on the overall test metrics to the TELogger and then stops. Once the test process has stopped, the CTM server process updates the various CTM databases and files to reflect that the test is no longer running.

Summary

After many years of use, there can be no doubt that CTM is an invaluable tool for testing OpenVMS on large clusters, and that it provides power and flexibility that goes beyond what could be provided by any single test. The TEL Reporting Utility (TRU) provides reports and clusterwide overviews of the testing and test history that are invaluable and that would not otherwise be possible.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Although this paper began by talking about the challenges involved in testing on OpenVMS, the greatest challenge of all was left unstated. That challenge is to continue to maintain the highest levels of quality and reliability for which OpenVMS is renowned. The hope and expectation is that CTM will continue to help meet this challenge in the years to come.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Digital Signature in Automatic Email Processing: A Customer Case Study

Authors:

Francesco Gennai

Network Systems and Services team leader, ISTI-CNR, Pisa, Italy

Marina Buzzi

Technologist, IIT-CNR, Pisa, Italy

Revised by Giovanni Vischio, HP Italy, OpenVMS Ambassador

Overview

Today many services can be requested over the network by sending an electronic form or message to the service provider. If the data is coded in syntax that can be read by computers, it is possible to automate its interpretation, elaboration, and storage, thus speeding up data processing and reducing human error.

A digital signature can be associated with Internet messages to guarantee sender authentication, message integrity, and nonrepudiation of origin. The verification process for incoming digitally-signed messages is usually performed by the email client on behalf of the end user. However, if a digital signature is applied to data that is subjected to automatic elaboration in order to maintain the above-mentioned benefits, it may be convenient to automate the verification process as well. Our idea is to implement the verification process in the email server rather than the client.

This paper describes our experience in designing and implementing software to automate the verification of digitally-signed messages and web forms in order to simplify the registration of Internet domains under the .IT Top Level Domain.

Introduction

Verification of incoming digitally signed messages is a critical operation that is usually performed by the receiver using an email client (receiving agent). If the verification process is unsuccessful (that is, the "from" header field does not match any of the email addresses present in the associated certificate), the client alerts the user. However, the user plays a fundamental role in the entire process by doing the following:

Correlating the message content with the sender (usually a relationship exists between sender and content)

Authorizing the client to upload or remove certificates of trusted Certification Authorities (CAs)

Configuring the client to retrieve the CAs' Certificate Revocation Lists (CRLs) to ensure the correctness of the verification process.

In system design, it is very important to evaluate the purpose as well as the context in which the digital signature is applied. It is obvious that the automation of the verification process is applicable only within certain contexts. Unless a semantic relationship exists between data and sender that

© Copyright 2004 Hewlett-Packard Development Company, L.P.

only the receiver can understand, the verification process can be transferred to automatic systems, which can process a large number of messages in a single time unit.

The Message Verify (MV) system has been designed to simplify the registration of Internet domains under the .IT Top Level Domain (TLD), performed by the Italian Registration Authority[1] (RA).

Usually ISPs (or maintainers) register domain names on behalf of third parties (organizations, associations, individuals, and so forth) and also take charge of technical maintenance duties such as managing DNS for the requested domain. Each domain name registration requires a two-step procedure:

- Fill out and send a letter of assumption of responsibility (LAR) for the use of the domain, signed by the registrant (using fax or surface mail)
- Send one electronic mail[2] message containing technical and administrative data needed for the registration (domain name, organization, administrative contact, and so forth).

The email can be automatically generated by filling out a web form or manually composed as a text file with well-established fields conforming to a predefined syntax (for example, name:value). The elaboration of these data is automatic. The software for data processing is triggered upon message reception. It performs user access control and verifies syntactical and technical correctness of data (for example, it checks DNS configuration). If any control fails, an automatic error notification is created and sent to the sender. User authentication is performed by means of a password assigned to the ISP. This password permits the user to access the web form for the domain registration. Alternatively, the password can be ciphered and included in the plain text message. With the same password, an ISP can view the status of its' ongoing registrations using web interfaces.

The MV system introduces the use of digital signature instead of user/password authentication. The use of digitally signed messages offers strong user authentication (compared to the user/password mechanism) and guarantees data integrity as well as nonrepudiation of origin, which is fundamental for proving data trustworthiness in case of legal problems (for example, between the RA and its customers).

An alternative to signed emails is the use of digitally signed web forms. Both technologies are valid. Using an email server, which is a "store and forward" service rather than an interactive service like the web, allows the masking of any processing delay and accidental server unavailability from the user. However, web service offers the user more awareness of the session progress (for example, data transmission and elaboration).

Our study embraces both the design and development of an automatic system for verifying digitally-signed messages and web forms as well as the activation of security mechanisms to maintain an adequate degree of operational security and reliability.

The MV System

The MV system was designed to be transparently placed between the message's arrival and the automatic elaboration of its content. Transparency requires that the pre-existing legacy software for automatic elaboration remain unchanged. This basic requirement was the first constraint on the system design. Our system automatically verifies the digital signature associated with an incoming

[1] The Italian Registration Authority at `http://www.nic.it/`

[2] The email must be sent from the provider/maintainer within ten working days from the date the RA received the LAR; otherwise the domain name becomes available for a new assignment.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

message before transmitting data to the process for automatic elaboration. If the verification is successful, the request is accepted and elaborated; otherwise it is rejected and a notification is automatically sent to the sender.

First we studied the automation of the digital signature's verification process. This required studying integration of Public Key Cryptography Technology with the electronic mail system and understanding security service mechanisms based on cryptography.

Another fundamental question was how to carry out the function performed by the user interacting with the client software (that is, to choose trusted CAs and download certificates or CRLs). To maintain control over the verification process, all functions the user performs by means of an email client must be assigned under the control of a system administrator using a simple web interface.

To perform the verification process, the receiving agent extracts a great deal of information from the message: the cryptographic algorithms applied for signing the message (one-way hash function, encryption algorithm), the sender certificate, and the CA certificate chain (if present). The verification environment is quite complex. This can be explored in detail by following activities and studying documents by the IETF working group: S/MIME Mail Security. However the verification process can be logically divided into two main functions:

Recognize message MIME parts containing protected data. The RFC 1847 "Security Multi-parts for MIME: Multipart/Signed and Multipart/Encrypted" specifies how to apply security service to MIME body parts. It adds two new content types: Multipart/Signed and Multipart/Encrypted. Both of these content types contain two body parts: one for the protected data and another for the control information necessary to remove the protection. RFC 2630 describes the Cryptographic Message Syntax used to digitally sign, digest, authenticate, and encrypt messages, and the RFC 2633 defines the application/pkcs7-mime and application/pkcs7-signature MIME types used to transport S/MIME signed messages.

Apply the verification process to the extracted MIME parts. Mechanisms are needed to retrieve and validate certificates. The RFC 2632 specifies basic rules to be implemented by receiving agents to correctly verify a signed message. In addition, the RFC 3280 outlines the format and semantics of certificates and certificate revocation lists for the Internet PKI. RFC 2560 describes procedures for processing certification paths in the Internet environment.

Operative Environment

The operative environment consists of a 2-node SCSI cluster sharing an external disk array. Each node is a COMPAQ AS 800 5/500 (1 GB of RAM, 4GB internal disk) running OpenVMS Version 7.3-1. The external storage is a Digital RAID7000 configured as follows:

- 2 RAID5 string of 6 x 36.4 GB
- 1 RAID5 string of 3 x 18.2 GB
- 1 RAID5 string of 6 x 9.1 GB

The MV system is composed of numerous software modules written in DCL, and utilizes the cryptography libraries of OpenSSL (v. 0.9.6d, the openssl.org standard distribution). It interacts with the PMDF electronic email/FAX system (v. 6.2, http://www.process.com/) and the HTTP OSU Web server (v. 3.9c, Ohio State University Web Server) running on both the cluster nodes.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The OpenSSL toolkit has been fundamental for system development. OpenSSL offers libraries for developing cryptographic software and for managing PKI objects (certificate manipulation, basic CRL manipulation, basic CA management, signing, encrypting, decrypting, verifying, and so forth) as well as line-mode tools. By using these commands, the MessageVerify system performs most controls including signature verification, control of valid paths to trusted CAs, verification of accessibility and validity of CRLs, control of expiration, and revocation of certificates.

The Architecture

The digital signature process works as follows:

A user sends a digitally-signed message to a mailbox configured for automatic elaboration of message content.

A unique identifier is assigned to the incoming message.

The digital signature is verified.

If verification is successful, the signed part is extracted and sent to the elaboration process.

If verification fails because of a temporary error (for example, a CRL is not available), the message is queued for future attempts until an established threshold is reached (see Configuration Parameters).

If verification fails because of a permanent error (for example, no digital signature is present), an error notification is automatically sent to the sender and the system administrator.

The system is composed of software modules, databases, and web interfaces for administration and control. Each module, which can be executed in parallel on each node of the cluster, performs one specialized function. Figure 1 shows the logical scheme of the Message Verify system, where processes (that is, modules in execution) are represented by rectangles and databases are represented by ellipses.

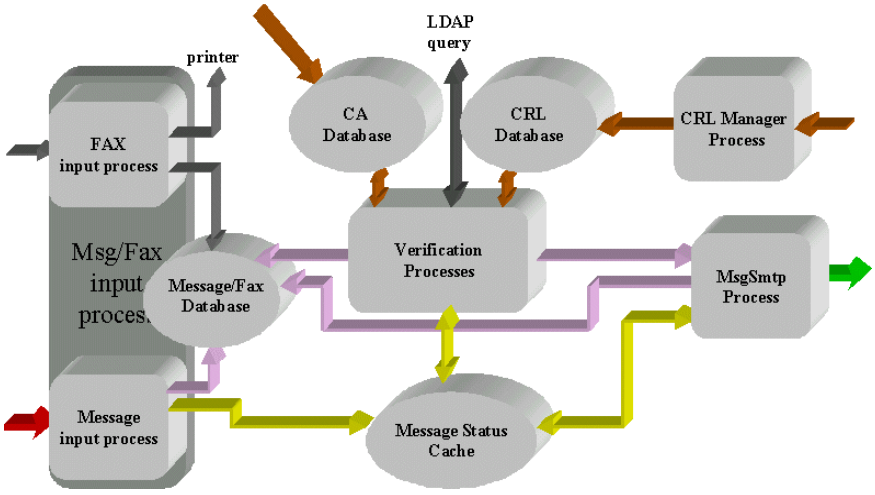

Figure 1. MV System's Logical Scheme

Modules perform the following functions:

© Copyright 2004 Hewlett-Packard Development Company, L.P.

The **Message Input** process performs preprocessing of messages to extract data necessary for subsequent elaboration, synchronization between the MV system and fax-RA system (receiving LAR) to generate the unique identifier, and message queuing.

The **Verification** process is the heart of the MV system. It verifies the signature's validity and adds four header fields to the message to specify the certificate Issuer Distinguished Name, the certificate Subject Distinguished Name, the certificate serial number, and the concatenation between the verification process return code and the message global identifier. In this way the stored messages keep information about the result of the verification operation.

The **MsgSmtp** process composes the final message, removing the MIME part that includes the digital signature, and sends it toward the RA systems. Messages are processed in parallel on a 2-node cluster, which means they may be out of sequence after verification. The process reorders the messages in the correct temporal sequence, according to the message status present in the Message Status Cache, before sending them to the automatic elaboration process (see Figure 2).

The **CRL Manager** process automates the management of the CRL's database and keeps it up to date. The module itself notifies the administrator of any errors or warnings (for example, the proximity of the CRL's expiration date).

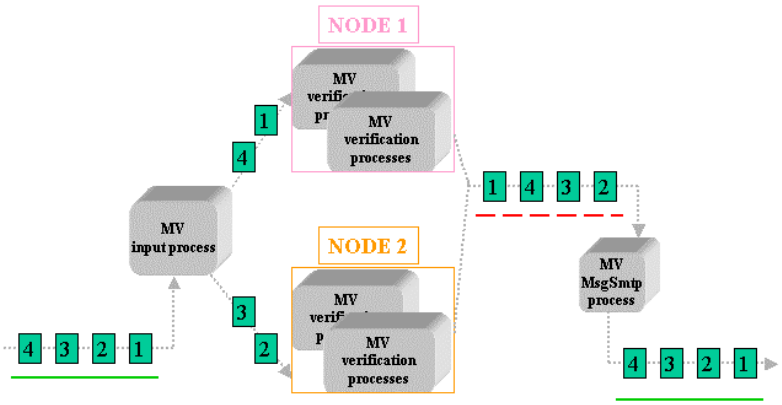

Figure 2. Message Flow

The MV system utilizes the following databases:

- **Message/Fax Database**
  The message/fax database includes both messages and requests received by means of fax, stored as postscript files. A global identifier is assigned to each message or fax entering the system (by the message or fax input processes), thus maintaining the temporal sequence of the requests; this step is an important factor in being able to resolve collisions on requests for the same domain name.

- **CA Database**

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Certificate of trusted CAs are added to (or removed from) the CA database by a system administrator. The system can use certificates issued by any CA after its own and any sub-CA certificates have been loaded in the database. The loading can occur by one of the following means:

HTTP session with the server web where the CA certificate is published
File upload from the administrator's local disk

- **CRL Database**
  CRLs are automatically downloaded (using the Certificate's CRL Distribution Point field or one system configuration parameter) by the CRL Manager process (which uploads the local CRL database). LDAP query is possible but not yet implemented.

- **Message Status Cache**

The message status cache is used for greater efficiency and separates process information from that related to fax and messages. It maintains temporary information on the message's status as the message undergoes elaboration.

Management and control of the system is performed using web interfaces.

Notifications and Errors

The verification process can fail for several reasons, including the following:

The "from" header field does not match any of the email addresses present in the certificate (*subjectAltName* field and *emailAddress* attribute of distinguished Name field).
A path to a trusted root CA does not exist.
It is not possible to retrieve a CRL.
CRL is not valid.
CRL is expired.
Certificate is expired.
Certificate is revoked.

Failure can be temporary or permanent. For temporary failures (for example, CRL is not available) the system makes a fixed number of retries until it reaches a threshold that is configured through the administrative interface. Both notifications and errors are inserted into the *Subject* header field and into other internal message fields (From:, Global ID:, Message ID:, Previous Message ID:, Error-type:, Number of retries:, Cert Issuer:, Cert Subject:).

System Extension

Recently, a new component (Form Input Process) was added to the system to accept signed web forms that contain the domain registration data. Using a Netscape browser (4.04 or higher), it is possible to apply a digital signature to a web form. Netscape V4.04 and higher include a JavaScript method (crypto.sign.Text) that enables the user to apply a digital signature to input data. Conforming to the PKCS7 and to the cryptographic message syntax (which in part extends the

© Copyright 2004 Hewlett-Packard Development Company, L.P.

PKCS7), Netscape utilizes the external signature modality. The resulting signed-data structure does not contain the original data, which must be dealt with separately. In this case, the session is interactive and the user receives acknowledgement of the signature verification process along with a unique identifier, which can be used to access the state of the request.

To synchronize the unique identifier among messages, fax, and web forms (in a cluster environment) required rewriting the algorithm of allocation of the identifier generators (msgid, faxid and globalid). The new algorithm can manage any level of conflict of access to the identifier generator. When there is a conflict in resource allocation, it is simpler and correct to generate an error message to the interactive user (by web access) rather than interfere with the messaging or fax processes. Based on these considerations, the algorithm can favor allocation requests arriving from message or fax processes by "interrupting" the concurrent CGI/HTTP process.

Web Interfaces

The MV system can be fully configured and monitored using the web. The user or administrator interface requires user authentication using one of the following access modes:

Normal login (for operators) furnishes access to the message/fax archive, which is useful for domain name modification, monitoring, or removal, and for supporting help desk activities. It permits searches based on unique a identifier or date, visualization and reprinting of faxes and messages, and access to verification process results.

Privileged login allows the system administrator to do the following:

- Add and remove CA certificates.
- Manage the CA certificates database, and display certificate fields.
- Display the log file of the CRL manager process in short and extended format.
- Configure and monitor the system.

The configuration interface has been designed to be highly configurable to adapt easily to changes in organization policy.

Related Work and Security Issues

The problem of reducing the cost of bulk cryptographic operations (especially public key operations, which are computationally expensive) has been treated in numerous papers. Two approaches are widely studied: remotely keyed encryption and server-aided cryptography.

Remotely keyed encryption efficiently achieves encryption by sharing the computational load between a fast but untrusted device and a slow, trusted device. Blaze introduced remotely keyed encryption schemes (RKESs) to support cryptographic applications that require high bandwidth, such as encryption of online multimedia contents, in a secure, low bandwidth, smart card environment (where users' private keys never leave the smart card). In such schemes, the load of communication and computation required of the smart card is independent from the input size [Blaze96]. Further studies to define the security of other RKESs [Lucks97, Blaze98, Lucks99] and other applications have been accomplished. In particular, Weis applied remotely keyed encryption in a java card environment [Weis 2000].

Server-aided cryptography refers to the ability to aid small and resource-limited devices in expensive computation. Jakobsson introduced this term

© Copyright 2004 Hewlett-Packard Development Company, L.P.

while studying how to decrease the local computational cost by transforming a heavy task into a large set of small subtasks carried out by a set of external servers, and performing security analysis of the proposed scheme [Jakobsson01]. By using this scheme he generated inexpensive server-aided batch signatures, showing improved efficiency for groups greater then 20. Recently, Ding applied server-aided cryptography in smart but resource-limited devices such as PDAs, cell phones, and palm pilots [Ding02]. In particular, Ding generated server-aided signature (SAS) while implementing fast certificate revocation, which is fundamental for the accuracy of the verification process. This approach is based on using a partially trusted server, which acts as a security mediator (SEM) executing the main load of the computation. Two half signatures, independent but interrelated, are generated by the SEM and the user to create a SAS signature. A relevant characteristic of this scheme is its online nature permitting fast revocation of signing capabilities, which limits damage from potential compromises. This scheme appears to be suitable for many mobile infrastructures. However, centralization of the architecture and incompatibility of the SAS signature with other signature types, make it impractical at the moment. (SEM is a single point of failure if it is subjected to DoS attack, performance bottleneck, and so forth.)

Berson et al. proposed providing cryptographic operations as a network service [Berson01]. Authors designed and built a centralized cryptoserver equipped with hardware cryptographic accelerators to provide public key operations to clients using their internal network. Because the cost of the accelerator is shared by a large number of clients, this system allows clients to benefit from hardware speedups while reducing CPU load and the cost of a single cryptographic operation. This approach differs from the previous ones because it implies that the client must trust the cryptoserver with knowledge of his private key. This delegation mechanism practically simplifies the design of the user-server interaction, although it poses a security risk of private key storage and protection. Berson observed that outsourcing cryptography inherently raises questions about the trustworthiness of the computation. However, cryptographic operations can be split into two categories: those requiring knowledge of the client's private key (that is, decryption and digital signature), and the other needing only the client's public key (publicly distributed by means of his certificate), which are encryption and verification of digital signature. The first category poses security issues (of secret key storage and protection) and requires more attention than the second category. Thus, instead of building a complete cryptoserver, it is simpler to use a subset of cryptographic operations (encryption and digital signature verification) and limit automation to these. In this case, assuming that the communication between the client and server is secure, the cryptographic operations have the same degree of trustworthiness as performing the computation locally [Berson01].

Our approach is similar to Berson's but is less ambitious. The MV system automates only one cryptographic function, which does not require the use of secret keys. Although our system has this limitation, it is simpler, cheaper (being based on free cryptographic software), and standard – compared to previously discussed schemes – and is interoperable with any type of signature. Our goal is to improve the quality of service offered to a large set of widespread users. In our scenario, automation of the verification process is possible and appropriate because signed data do not contain any semantic information that only the receiver is able to understand. Therefore, the main security issue of the MV system is its integrity. If a system is violated, the entire service can be compromised. For example, if the legacy software receives input data that bypasses authentication, it makes a registration invalid.

In the 2001 annual security report, CERT affirms: "… the ability to attack a system depends on the 'global' security of the Internet." For example, a DoS attack takes advantage of other weak points in the network. This implies that at the moment there is no way to implement a totally secure policy, but it is important to create a strategy and use combined technologies to fend off intruder attacks.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

According to this approach, in order to protect the system, different contra-measures have been applied to build protection levels with the following features:

***Software design*:**

The system is designed to be failure-tolerant and robust.
The MV system and the legacy software are within the same security perimeter.

Operating environment:

Managing network services in an OpenVMS[3] environment reduces the recurring security problems typical of other platforms and enhances compatibility between different versions of software. The benefits of using an OpenVMS system equate to those of Unix and Windows systems, with the notable advantage of being one of the least-attacked systems on the Internet today.

The system is carefully configured to include access control, logging of user sessions, auditing of security-relevant events, ensuring that all features and services that are not explicitly required are disabled, and so forth.

Network:

Protection of the network perimeter includes the following features:

Blocking the transfer of executable files on the Internet gateway or email server
Configuring servers to disable features and services that are not explicitly required
Configuring routers and firewalls to enable traffic only to authorized servers and ports
Running and maintaining updated anti-virus software

In addition, Intrusion Detection Systems, which conduct analyses of live network traffic, and Honeypots (usually single systems that emulate systems, known services and vulnerabilities, or create jailed environments) can contribute to monitoring network and system activities and detecting attacks.

Be sure to read security bulletins and promptly apply software patches to the operating system and applications.

Currently, a drawback of the proposed solution is the cost of certificate creation and distribution as well as help desk support. The solution is applied to ISPs, which manage nearly all domain name registrations[4]. This kind of automation will be suitable for everyone when PKI technology further penetrates e-society.

Performance

Before buying expensive cryptographic hardware and software, it is important to evaluate the real load of the system or application to verify whether high performance is necessary. In our context, we operated under the following conditions and assumptions:

In general, email-based applications are not required to be as fast as interactive applications such as web access.

To test the system, the Italian Registration Authority (RA) set up its own CA and issued certificates only for a restricted set of ISPs, thus employing a low load of the MV system.

[3] The security protection provided by OpenVMS Operating System has been evaluated by the National Computer Security Center, receiving a C2 rating.
[4] Direct contract with final users (domain name requesters) are possible, but discouraged.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

To evaluate system performance we undertook a real test in our operating environment. The test was split into two parts: verification, and message delivery to the software for automatic elaboration. In operating system terminology, a job unit can involve the execution of one or more modules. In our case the job includes all message elaboration between its arrival in the system and its exit. A script was set up to create and send a load of 2000 signed messages to the system.

Test results show that better performance can be achieved by activating a single job in each node of the cluster, with the total execution time of 1hour and 4 minutes. The activation of more jobs on the same node (for example, multiple processes) does not produce additional advantages, but provokes a slight increase in total execution time, probably due to competition of processes for access to resources. The average number of domain registrations per month in 2003 was 15,573.

Considering that our test was performed with the cluster fully operational, the results show that the system should be able to process the current monthly load in less than 9 hours. This implies that we do not need to invest in expensive cryptographic hardware and software.

Summary

The Message Verify system was developed to permit automatic elaboration of digital signature in a flow of messages. Our experimentation revealed the following numerous advantages:

Increased quality of service. Digital signature offers strong authentication (compared to the weak "user and password" mechanism), data integrity, and nonrepudiation of origin.

Increased efficiency of service compared to signature verification using an email client.

Transparency of service for RA operators. The service is user-friendly and training is not required.

Reliability of the verification process and document management.

Low cost, using free cryptographic software.

As an additional benefit, the use of certificates and secure mail has permitted users to become familiar with these technologies, which have wider application fields. Although the proposed solution implies the initial cost of certificate creation and dissemination, in the future, when all citizens have their own certificate to access numerous secure online services, this kind of tool can be used to obtain strong authentication of sensitive services (for example, accessing critical resources) directed to a large audience, such as that offered by public administrations and health departments.

References

Berson, T. et al, 2001. Cryptography as a network service. Proceedings of Eighth Annual Network and distributed System Security Symposium. San Diego, USA. http://www.isoc.org/isoc/conferences/ndss/01/2001/INDEX.HTM.

Blaze, M. 1996. High-bandwidth encryption with low-bandwidth smartcards. Proceedings of the Fast Software Encryption Workshop. Lecture Notes in Computer Science. No. 1039, pp 33-40.

Blaze, M. et al, 1998. A formal treatment of remotely keyed encryption. Proceedings of EUROCRYPT98. Lecture Notes in Computer Science. No. 14039, pp 251-265.

Ding, X. et al, 2002. Experimenting with Server-Aided Signature. Proceedings of Ninth Annual Network and distributed System Security Symposium. San Diego, USA. http://www.isoc.org/isoc/conferences/ndss/02/proceedings/index.htm

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Jakobsson, M. and Wetzel S., 2001. Secure Server-Aided Signature Generation. Proceedings of the Public Key Cryptography Conference, Cheju Island, Korea, pp. 383-401.

Lucks, S. 1999. Accelerated remotely keyed encryption. Proceedings of the Fast Software Encryption Workshop. Lecture Notes in Computer Science. No. 1636, pp 112-123.

Lucks, S. 1997. On the security of remotely keyed encryption. Proceedings of the Fast Software Encryption Workshop. Haifa, Israel, pp 219-229.

Weis, R. et al, 2000: Remotely Keyed Encryption with Java Cards: A Secure and Efficient Method to Encrypt Multimedia Streams. Proceedings of the IEEE International Conference on Multimedia and Expo (ICME). NY, USA, pp. 537-540.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Native 64-bit Virtual Addressing for Oracle Rdb Row Caches Capability and Performance

Norman Lastovica
Oracle Corporation, Relational Technology Group, Rdb Engineering
Oracle New England Development Center, Nashua, New Hampshire

Oracle Rdb Release 7.1.2 Row Cache Enhancements

Oracle Rdb for OpenVMS Alpha Release 7.1.2 introduces two significant enhancements to the Row Cache feature: Snapshots in Row Cache and Native 64-bit addressing support for Row Cache. These features provide significant additional database performance and capability increases by further avoiding disk I/O and database page locking operations and by permitting significantly more data to be easily cached in memory.

Using the snapshots in Row Cache feature allows applications to approach zero disk I/O operations per transaction by reading and updating database rows entirely within memory while using the after-image journal to provide persistent storage data protection. Native 64-bit addressing support for Row Caches permits a vast number of database rows to be cached, limited solely by the amount of memory available in the computer.

This paper provides an introduction to the Native 64-bit addressing support for the Row Cache feature in Oracle Rdb Release 7.1.2. In addition, the results of large system performance and capability tests are described.

Row Cache Background

Introduced in Oracle Rdb Release 7.0, the Row Cache feature allows faster and more efficient access to database rows (both table data and index information). An Oracle Rdb Row Cache is a section of globally accessible memory that contains copies of database rows. This cache provides the ability to store, fetch, and modify frequently accessed rows in memory, avoiding disk I/O and locking. Database rows remain in memory even when the associated page has been removed from the local or global buffer pool. Row caching provides the following advantages:

- Reduced database read and write I/O operations
- Reduced database page locking operations
- Reduced CPU overhead for accessing a database row in cache
- Improved response time
- Efficient use of system memory resources for shared data

When a row is requested from the database, Oracle Rdb first checks to see if the requested table or index is mapped to an existing Row Cache. If a Row Cache is mapped, Oracle Rdb then checks to see if the requested row is in the cache. If the row is found in the Row Cache, the row is retrieved directly from the cache. If the row is not in the cache, Oracle Rdb checks the page buffer pool (either a local, per-process buffer pool or a global, shared buffer pool). If the row is not in the page buffer pool, Oracle Rdb locks the database page and initiates an OpenVMS I/O request

© Copyright 2004 Hewlett-Packard Development Company, L.P.

(either via $QIO or $IO_PERFORM) to retrieve the database page containing the row. Assuming space is available, the requested row is then inserted into the Row Cache.

When a modification is made to a cached row (either table data or index structures), if the original database page containing the row is locked for update in the process's page buffer pool (in such case, the page will have to be written back to the database anyhow), the modified data is written back to the database and to the cache. However, if the original database page is not locked for update or is not found in the process' buffer pool, the modification is made directly into the cache without incurring an additional database I/O or page locking.

Multiple Row Caches can be active for a database. A Row Cache must be available to, and shared by, all processes attached to the database. Using OpenVMS Galaxy cluster configurations, processes on different nodes within the galaxy share Row Caches that are stored in "galactic" shared memory.

No application changes are necessary when utilizing the Oracle Rdb Row Cache feature. The following are requirements for using the Row Cache feature:

- After-image journaling must be enabled for the database
  - Journals must be created
  - The fast commit feature must be enabled
- The Row Cache feature must be enabled
- Cache slots must be reserved
- Caches must be defined
- All database users must have access to shared memory. This can be accomplished by either:
  - Setting the database parameter NUMBER OF CLUSTER NODES to 1, or
  - Setting the database parameters GALAXY SUPPORT IS ENABLED and SINGLE INSTANCE

Existing VLM Capabilities in Oracle Rdb

The Oracle Rdb VLM (Very Large Memory) feature was created for Rdb Release 7.0 to allow access to more than 32 bits worth of virtual address space. This interface was implemented because, at that time, programs on OpenVMS Alpha did not have the ability to directly access memory beyond the 32-bit virtual address space.

The traditional VLM implementation within Oracle Rdb manipulated the OpenVMS Page Table Entry (PTE) mapping pointers for a small number of virtual pages in 32-bit process virtual address space (called "windows") to access physical pages allocated directly from OpenVMS. This technique allowed a large amount of physical memory to be accessed by changing memory pointers at run time.

While effective, this original VLM implementation did have some drawbacks. Performance was reduced, slightly, during the manipulation of the page table entries, as CPU cycles were required to unmap and remap a page of memory. And the intimate knowledge of OpenVMS internal memory management structures required additional maintenance efforts and was considered an issue for porting Oracle Rdb to the OpenVMS Itanium architecture.

Further, when using the VLM feature in prior versions of Oracle Rdb, Row Cache control structures were always stored in 32-bit virtual address space. The location of these control structures depended on whether shared memory for the Row Cache was designated as "process" or

© Copyright 2004 Hewlett-Packard Development Company, L.P.

"system." When using the SHARED MEMORY IS PROCESS attribute, the control structures were stored in a process global section in P0 (32-bit program region) virtual address space. When using the SHARED MEMORY IS SYSTEM feature, the control structures were stored in S0S1 (32-bit system region) virtual address space. This virtual memory in system (S0S1) virtual address space is allocated via the LDR_STD$ALLOC_PT system routine. Physical memory is allocated, one page at a time, by the MMG_STD$ALLOC_PFN_64 system routine.

Limitations Within a 32-bit Virtual Address Space

The Row Cache feature was historically limited to something less than 33 million total cached rows per database. This limitation was due to storing control data structures in 32-bit virtual address space. Even with the LARGE MEMORY IS ENABLED attribute, some data structures still were located in 32-bit virtual address space, ultimately leading to this restriction. As databases and application performance requirements have grown, this limitation has become a concern. In addition to Row Cache size limits, competition for the 1 GB process P0 virtual address space (shared among application code, application data buffers, database code, database buffers, Row Caches, and so on) and competition for the 2GB system S0S1 virtual address space (used by OpenVMS as well as for Row Caches and global buffers using the SHARED MEMORY IS SYSTEM feature) caused excessive compromise in design and performance.

Native 64-bit Virtual Addressing for Row Caches

OpenVMS introduced native support for accessing the 64-bit virtual address space defined by the Alpha architecture and has provided additional memory management features since 1995. The OpenVMS operating system and current Alpha architecture implementations support 8 TB (terabytes) of virtual address space.

Starting with Oracle Rdb Release 7.1.2, the Row Cache feature utilizes the native 64-bit virtual addressing support within the Alpha processor and OpenVMS operating system. Row Caches are always created in the OpenVMS P2 **(**64-bit program region) virtual address space.

Performance benefits of the native 64-bit addressing for Row Caches are realized by allowing vastly larger Row Caches to be created and by avoiding performance penalties related to the previously available Row Cache use of VLM capabilities within Oracle Rdb. Additionally, configuration and management of very large Row Caches (those with many or large cache slots) is simplified by always utilizing OpenVMS global sections.

There are no visible user or application effects (beyond potential performance improvements) due to the Oracle Rdb implementation of native 64-bit virtual addressing for Row Caches.

The Row Cache memory mapping attributes LARGE MEMORY IS ENABLED and SHARED MEMORY IS SYSTEM have been replaced with the RESIDENT attribute. Although these clauses are now deprecated, these attributes are internally considered as synonyms for RESIDENT.

If the RESIDENT attribute is specified (implicitly or explicitly) for a Row Cache, the cache will be created as a memory-resident global section. If the section is at least as large as the number of CPU-specific pages mapped by a single page table page (approximately 8mb), it will be created utilizing OpenVMS "shared page tables" (potentially with granularity hints).

OpenVMS *shared page tables* for memory-resident global sections reduce physical memory consumption by allowing multiple processes to share one set of page table entries for the global section. These shared page tables can save a significant amount of physical memory for large global sections with many users. The Alpha processor's implementation of *granularity hints* allows

© Copyright 2004 Hewlett-Packard Development Company, L.P.

ranges of pages to be mapped by a single translation buffer entry within the processor, leading to improved translation buffer hit rates and utilization.

Considering the reduction in physical memory consumption provided by *shared page tables,* and the performance advantages provided *granularity hints*, Oracle recommends that Row Caches be configured with the RESIDENT attribute when possible. It is important, however, to avoid "starving" the system for memory by creating resident caches that consume memory that would be needed by, for example, application program working sets.

Global sections are created with the $CRMPSC_GDZRO_64 system service by the first process to reference a Row Cache. Subsequent access to a created cache by other processes is with the $MGBLSC_64 system service. Row Cache global sections are always mapped in P2 virtual address space (either in the default P2 region or a specified region if shared page tables are being created). Global sections are named including a system-unique database specification along with a unique Row Cache identifier within the database.

When a memory resident global section is created, OpenVMS writes invalid global PTEs (Page Table Entries) to the global page table. When the global section is mapped, invalid page table entries are placed in the page table; physical memory is not allocated until the pages are referenced. To ensure that all pages are valid and resident in memory, Oracle Rdb accesses each page when a memory resident section is first created. The first access to the page causes a page fault (due to the invalid PTE) and OpenVMS then initializes the physical page with binary zeros and the page is then valid for all users of the global section.

Specify Physical Memory by RAD

On AlphaServer GS-series systems that support OpenVMS Resource Affinity Domains (RADs), Row Caches that are configured with the RESIDENT attribute can be individually designated with a preferred RAD for memory allocation. By carefully managing the home RADs for database users with critically high performance requirements and the physical memory allocated for Row Caches used by these processes to appropriate RADs, performance can potentially be increased by reducing "off-RAD" memory references. Note that the RAD attribute for a Row Cache simply specifies a hint to OpenVMS about where memory should be physically allocated. It is possible that, if the memory is not available, it will be allocated from other RADs in the system.

Restrictions

This change to utilize native 64-bit virtual addressing is specific to the Row Cache feature in this release of Oracle Rdb.

With Oracle Rdb Release 7.1, the maximum total size of any individual Row Cache is 2,147,483,647 rows. This restriction may be relaxed in a future Oracle Rdb release. A database may be configured with many Row Caches so the total number of cached rows for a database is generally limited by the physical memory available on the system.

System Parameter Considerations

Shared memory sections using the LARGE MEMORY IS ENABLED or SHARED MEMORY IS SYSTEM features were previously not created as OpenVMS global sections and were not directly affected by the various system parameters controlling global section allocation. Because all Row Cache shared memory sections are now created and mapped as global sections, it is possible that the GBLSECTIONS, GBLPAGES and GBLPAGFIL system parameters may have to be increased in order to map large caches that previously relied on the LARGE MEMORY IS ENABLED or SHARED

© Copyright 2004 Hewlett-Packard Development Company, L.P.

MEMORY IS SYSTEM features. Note that GBLPAGES and GBLPAGFIL are dynamic system parameters (can be updated without rebooting the system) while the updates to the GBLSECTIONS parameter requires that the system be rebooted for the change to take effect.

When mapping Row Caches or opening databases using the GALAXY SUPPORT IS ENABLED feature in an OpenVMS Galaxy environment, a "SYSTEM-F-INSFMEM, insufficient dynamic memory" error may be received. One source of this situation is running out of OpenVMS Galaxy "Shared Memory regions" (a future OpenVMS release is expected to provide a more specific error message). For OpenVMS Galaxy configurations, the system parameter GLX_SHM_REG specifies the number of "shared memory region" structures configured into the Galaxy Management Database (GMDB). While the default value of 64 (for OpenVMS versions through at least V7.3-2) may be adequate for some installations, sites using a larger number of databases or Row Caches will likely find the default insufficient. Refer to the Oracle Rdb Release Notes for additional information about configuring an OpenVMS Galaxy environment for Oracle Rdb.

The RMU /DUMP /HEADER command can be used to display the physical global section sizes for Row Caches configured in the database. Oracle also offers a Row Cache sizing and configuration tool available for download from Oracle Support's MetaLink.

A Test Using Very Large Caches

To help test and demonstrate the capabilities of a rather large amount of cached data, a test database with one billion (10^9; 1,000,000,000) rows and associated indexes in Row Caches was created. This database was configured with 10 tables each horizontally partitioned across 5 storage areas. Each table consists of 10 quadword (64-bit integer) columns. One pseudo-ranked sorted index on a single column was created for each table. 100,000,000 data rows of random content were then loaded into each table. Each table required approximately 1.2 million index nodes at 2000 bytes each (index depth of 5 levels).

For each table, two caches were configured: one for the rows and one for the index nodes. Simulating an environment planning for future database growth, caches for the database rows were configured with 101,000,000 "slots" per table and the caches for the index nodes were configured with 1,515,000 "slots" per index. The caches for the indexes were somewhat over-sized as compared to the caches for the data rows. This was intentional to simulate a typical production application where the DBA would allow for modifying index key values. Changes to the index keys could cause index node splitting, leading to additional index nodes. The caches for indexes were also configured specifying the Row Cache "no replacement" attribute to enable an optimization that helps avoid cache slot contention for heavily accessed database objects.

Taking advantage of the "Snapshots in Row Cache" feature of Oracle Rdb Release 7.1.2, each cache was created with 50,000 snapshot "slots" to allow read-write transactions to store snapshot records for modified rows directly into cache (avoiding disk I/O and database page locking). By eliminating snapshot and database I/O when modifying cached records, the application should more fully utilize the system CPU resources, leading to reduced transaction duration. All caches were created specifying the "SHARED MEMORY IS PROCESS RESIDENT" attribute to allow use of OpenVMS shared page tables and granularity hint regions (reducing physical memory consumption and improving performance).

The total amount of physical memory utilized for Row Caches for this configuration was approximately 202GB (representing both overhead and usable data storage).

Data content for each table was generated by a program that wrote records of random values into an OpenVMS mailbox; the mailbox was read by an RMU /LOAD session that populated the table.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

After populating the caches with the data row and index node content, a synthetic workload was run. This workload represented database server processes that would continuously execute database update transactions, with each transaction choosing a table at random and selecting and modifying a random row in the table. Due to the initial data distribution pattern, approximately one transaction in a thousand would find the specified row non-existent and would insert the row into the database. Thus, every transaction either updates an existing database row or, occasionally, inserts a new row into the database.

The workload program was written in C using the Oracle Rdb SQL precompiler. In order to keep the test program development effort limited, and to help ensure a reasonable percentage of non-database activity on the system, the SQL update and insert statements are generated at run-time using the dynamic SQL interface. The technique of using entirely dynamic SQL for the workload may be revisited in the future, as the CPU cost to completely interpret and compile the SQL statement for each operation may be an excessive burden on performance of this test and limited overall throughput. The Oracle Rdb SQL language provides the ability to, at run time, compile and later repetitively execute statements. This approach would be expected to save some CPU time.

To ensure high levels of performance for writing to the after-image journal (AIJ), the database was configured to use the optional AIJ Log Server (ALS) process. The ALS process removes the task of writing to the AIJ from the user processes by performing all AIJ write operations for a database. The ALS is optimized to avoid journal locking operations, and does "double buffered" asynchronous I/O operations to the after-image journal to reduce effective I/O latency.

As the test application would be modifying or inserting records for every transaction, a large number of locking operations were expected during the test. To provide for efficient system scaling, OpenVMS was configured to use the optional dedicated CPU lock manager. This feature provides significant performance increases for heavy lock operation loads when many CPUs are present in the system by eliminating contention for locking data structures. A single CPU is dedicated to locking operations for all processes on the system with very low overhead.

System Configuration

The test system utilized a Hewlett-Packard AlphaServer GS1280 running 32 Alpha EV7 processors at 1150 MHz with 256GB of physical memory. OpenVMS Alpha Version 7.3-2 base level X9Z0 was installed and disk volumes were configured as RAID-5 arrays in an EVA12000 storage server accessed through KGPSA controllers. As disk I/O loads were expected to remain quite low during this test (ultimately less than 5000 I/O write operations per second), the configuration and absolute performance of the storage subsystem was not a significant concern and little analysis or tuning was performed or required.

Initial Test Results – More than 13,000 Database Transactions Per Second

To simulate a large multi-client database server environment, multiple copies of the workload were run at the same time. Instances of the program were added (to a total of 60) until the system was effectively CPU bound (over 98% utilization). All database server workload instances solely executed read-write transactions and updated or inserted a single row per transaction.

At this workload level, the peak database transaction rate was 13,228 transactions per second with significantly less than 1 disk I/O request per transaction. System-wide disk I/O rate was approximately 3000 per second and the system-wide locking operation rate (new lock requests, promotions, demotions and releases performed by the OpenVMS lock manager) was approximately 150,000 per second.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Second Test

In an attempt to increase the transaction rate, a second experiment was run with database snapshots disabled and the Oracle Rdb "commit to journal" feature enabled. Avoiding writing snapshots to the Row Caches should reduce CPU consumption (in a typical production system, this would not often be a viable configuration because it causes read-only transactions to get promoted to read-write transactions and some operations such as online database backup are prohibited while snapshots are disabled). The "commit to journal" feature reduces database root-file I/O by allowing transactions to allocate transaction sequence numbers from the database in groups of up to 1024 at a time, rather than one per transaction.

The database was also configured to allow Oracle Rdb to use the OpenVMS "Fast I/O" feature. Setting the "buffer object enabled" attribute for database file, root file, after-image journal file, and recovery-unit journal file I/O operations enables this feature within Oracle Rdb. The "Fast I/O" feature provides a streamlined I/O interface that can reduce CPU usage by simplifying the I/O path and performing buffer locking and probing once (typically at application startup), rather than for each I/O request.

To enable multiple statistics global sections to be automatically created by Oracle Rdb, OpenVMS was also configured to enable RAD support on the GS1280. By using many statistics global sections, memory contention for statistics counters maintained by Oracle Rdb is reduced. When Oracle Rdb detects that a system is configured with RAD support when a database is opened, a statistics section is created in each RAD. As user processes attach to the database, they select a statistics global section to be used based on the process's "home" RAD.

Second Test Results – One Million Database Transactions Per Minute

With the reduced overhead due to the elimination of database snapshots, the "commit to journal" feature, and the "FAST I/O" feature, the same test workload peaked approximately 17,000 database transactions per second. This was considered a significant data point in that it represents just over 1 million database transactions per minute for this particular workload.

Results Analysis and Updates

The performance indicators in this test caused some amount of interest between both HP and Oracle engineering staffs. Further lab testing revealed some areas in the Rdb engine that could be further optimized. In particular, some alignment faults (cases where the instruction stream was expecting to operate on naturally aligned data cells but caused a fault when the data was not aligned) were eliminated from the code.

The application itself was also enhanced to pre-compile all possible update statements during program startup. Then at run time, the correct pre-compiled statement was executed during each transaction. This technique avoided having to compile the dynamic SQL statement for each update, saving CPU cycles.

Third Test

Each release of Oracle Rdb version 7.1 is currently shipping as two variants: one compiled for all Alpha processors and one compiled for Alpha EV56 and later processors. The EV56 variant includes code compiled to utilize the Alpha byte-word instructions. As an internal performance experiment, the Rdb code was compiled explicitly for the EV67 and later Alpha processors. This configuration allowed the language compilers to produce an Alpha instruction sequence that was

© Copyright 2004 Hewlett-Packard Development Company, L.P.

optimal for the EV67/EV68/EV7 processors in both use of available instructions and the scheduling of instructions for the quad-issue Alpha processor.

A 32 processor GS1280 with 128GB was configured and made available for a second week of testing in January of 2004. OpenVMS V7.3-2 was installed with the experimental compilation of Oracle Rdb along with the enhanced workload program. The OpenVMS feature RAD support was not enabled for this test.

A number of experiments were run with the updated configuration. Areas of interest included measuring performance with varied numbers of CPUs to determine how effectively the system scaled from 8 to 32 processors, the effect of the Rdb AIJ Log Server (RDMALS) process, impact of the Record Cache Server (RDMRCS) process checkpoint operations, and so on.

Third Test Results – 1,791,480 Database Transactions Per Minute
A sustained measured database transaction rate of 1,791,480 for an interval greater than one minute was demonstrated (representing 29,858 transactions per second). Rates of over 30,000 transactions per second were sustained over 15 second intervals.

Further Analysis

In addition to the performance indicators, the ability to run these workloads on a large multi-processor system also provided the opportunity for Oracle Rdb Engineering and Hewlett-Packard OpenVMS engineering to measure and review various performance indicators. The results of such analysis helps provide direction for further areas of investigation for performance enhancements for future releases of both OpenVMS and Oracle Rdb.

Observations and Recommendations

Using caches as large as those in the tests requires some operational considerations. First, it can take quite a while for the Row Cache Server (RCS) process to create and map (and, by extension, initialize) these large global sections. The Database Recovery process (DBR), as well, can run for a longer time because it must scan all cache slots in order to make sure that the failed process had not reserved any cache slots. It may be possible to reduce these effects, to some extent, in future releases of Oracle Rdb.

In the same way, the RMU/SHOW STATISTICS utility can take a long time to accumulate statistics information for very large caches. Some of the displays require that all cache slots be examined (to determine the amount of space used in a cache, or to count the number of reserved rows, for example). This can require a significant amount of CPU time and the actual screen refresh interval will be quite slow.

When closing, backing up, or analyzing a database, the RCS process must write all modified rows back to the physical database. When using huge caches with many modified rows in cache, the RCS may take quite a while to do this writing. Planning ahead for a shutdown or backup may allow you to perform an RMU /SERVER RECORD_CACHE CHECKPOINT command prior to the scheduled shutdown or backup time. This allows the RCS to get a “head start” writing modified rows back to the database.

Additional Information

For more information about Oracle Rdb performance and configuration options, you may wish to refer to the following documents available from Oracle Corporation:

- *Oracle Rdb Release 7.1.2 Release Notes*

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- *Oracle Rdb7 Guide to Database Performance and Tuning*
- *Oracle Rdb7 Guide to Database Design and Definition*

For detailed information about the Alpha processor or OpenVMS memory management and programming, you may wish to refer to the following documents available from Hewlett-Packard:

- *OpenVMS Programming Concepts Manual*
- *HP OpenVMS System Services Reference Manual*
- *OpenVMS Calling Standard*
- *HP OpenVMS System Manager's Manual*
- *HP OpenVMS Alpha Partitioning and Galaxy Guide*
- *Alpha Architecture Handbook*
- *Alpha Microprocessor Hardware Reference Manuals*

Acknowledgements

Running the performance experiments would not have been possible without the expert assistance of Craig Showers and Bill Gabor at Hewlett-Packard's OpenVMS Enterprise Solution Center in Nashua, New Hampshire for providing access to the system and storage configuration along with system and operational support. Special thanks as well are due to Jeff Jalbert and Tom Musson of JCC Consulting, Bill Gettys, Bill Wright, Ian Smith, Ed Fisher, and Paul Mead of Oracle Rdb Engineering, and to Christian Moser, Greg Jordan, Karen Noel, Sue Lewis, Kevin Jenkins, Steve Lieman, and Tom Cafarella of Hewlett-Packard OpenVMS engineering for providing outstanding technical support and enthusiasm.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Inheritance Based Environments in Stand-alone OpenVMS Systems and OpenVMS Clusters

By Robert Gezelter, CDP, CSA, CSE
Software Consultant

Introduction

The standard OpenVMS user environment conceptually rests upon two related, yet distinct logical hierarchies:

- the four hierarchical levels of the logical name environment[1] and
- the execution of the system-wide and user-specific login profiles[2]

The classification of users into groups, the separate identification of privileged user groups (those identified as "System"), and all other users, suggests a natural hierarchical structure. While these hierarchies are more than what many other operating systems provide, they still are not fully reflective of many environments. The organization, the applications, and the computing environment of today's corporate organization are often more complex than a single organization with easily identifiable, disjoint groups.

Many user environments are far richer in diversity than they appear at first glance, reflecting users' different collections of applications, their roles, and their responsibilities.

Computing environments are more than single instances of processors, memories, and peripherals. The most critical elements of a well-configured OpenVMS environment reside in the logical environment created by the system manager and application architect, not in the specification of the bare hardware and software environment. The exact processor (VAX, Alpha, or Itanium) and peripheral configuration of the system is far less important.

User environments are hierarchically nested. For example, an individual user is typically a member of a group. In its turn, the group is a part of a company. Thus, an individual's application environment depends upon that person's place in the organization. Similarly, the group or department's environment is merely an instance of the standard company-level

---

[1] **LNM$SYSTEM** (composed of **LNM$SYSTEM_TABLE** and **LNM$SYSCLUSTER**, which in turn really translates to **LNM$SYSCLUSTER_TABLE**), **LNM$GROUP**, **LNM$JOB**, and **LNM$PROCESS** are part of the user's context created as part of the processing performed by **$CREPRC**.

[2] The system-wide login command file is located in **SYS$MANAGER:SYLOGIN.COM**; by default, the user-specific login is in the user's default directory. The user's login can alternatively be placed in any private or shared file that is accessible to the user through setting the appropriate fields of the user's record in the system UAF.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

environment, tailored to the specific functions performed by that department. In a service bureau or Applications Service Provider (generally referred to as an "ASP") environment, where multiple companies (or several sibling companies) share a system, there are also company-wide environments, which are shared by all users at an individual company but differ to some extent between different companies.

A review of the mechanics of what happens when a user connects to an OpenVMS system is appropriate at this point.

When a user logs on to an OpenVMS system, **LOGINOUT.EXE** creates a basic operating environment consisting of:[3]

- the default directory, **SYS$LOGIN**
- the device upon which the default directory is located, **SYS$LOGIN_DEVICE**
- a scratch device/directory, **SYS$SCRATCH**
- the default device characteristics established by the command procedures executed as part of the login process
- the logical name environment, which is a hierarchical list of names and translations of names. The logical name tables are searched in a variety of situations when commands and programs access a variety of resources, most commonly files and queues. Generally speaking, translation continues until no more translations are possible, each iteration starting again from the beginning. This feature is dramatically different from the one-pass symbolic parameters available on other systems. A user process's actual logical name context is built by **LOGINOUT.EXE** and includes:
    - the command files executed as part of login processing
    - the job logical name table, specific to the job, referred to by the name **LNM$JOB**[4]
    - the group logical name table, applicable to all users who share the same UIC Group number, **LNM$GROUP**[5]
    - the system logical name table, **LNM$SYSTEM**

---

[3] For simplicity, we refer to the basic case of an interactive user. The start of a network or batch produces similar results and follows a similar path. Creating processes that are neither interactive, batch, nor network may create a similar environment, or may result in a slightly truncated environment (e.g., whether the **/AUTHORIZE** option is used on the **RUN/DETACH** command). Environmental truncation has important implications for the proper operation of applications and facilities.

[4] **LNM$JOB** is itself a logical name, whose translation resides in the **LNM$PROCESS_DIRECTORY** as **LNM$JOB_XXXXXXXX**, where XXXXXXXX is the eight-digit hexadecimal address of the Job Information Block (see Goldenberg, Kenah, Dumas, "VAX/VMS Internals and Data Structures, Chapter 35, page 1073, first footnote)

[5] **LNM$GROUP** is itself a logical name, whose translation resides in the **LNM$PROCESS_DIRECTORY** as **LNM$GROUP_GGGGGG**, where GGGGGG is the six-digit octal UIC group number (ibid.)

© Copyright 2004 Hewlett-Packard Development Company, L.P.

    - the cluster logical name table, **LNM$CLUSTER_WIDE**[6]

- other process characteristics established as a part or consequence of login processing.

These capabilities are generic, and straightforwardly support the construction of environments based upon a simple cluster belonging to a single entity with groups of users, each with their own profiles. However, environments are often not as straightforward. Traditionally, complex environments have been implemented by explicitly invoking the particulars of each environment from the individual user's **LOGIN.COM** file. A user's specific environment is achieved by manually adding elements to the user's login profile. This method requires inordinate maintenance, is excessively fragile, and is virtually impossible to manage.

[6] Introduced in OpenVMS Version 7.2 (OpenVMS Version 7.2 New Features Manual, Order#AA-QSBFC-TE, July 1999)

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Philosophic Basis

OpenVMS provides a straightforward approach, founded upon the same architectural principles as the operating system itself, which dramatically simplifies the creation of customized environments. This can be accomplished with a minimum of complexity and maintenance effort, and a high degree of manageability and scalability.

The basis of the simplification is the realization that while environments differ dramatically, they do so in an orderly, systematic way, and those differences can be implemented with minor, manager-level changes.

An approach based upon axes of variation, leveraging the strengths of OpenVMS and its hierarchical logical name structure, is more robust, more auditable, and more maintainable than the traditional explicit enumeration approach.

Environments can be characterized by multiple, independent axes of variation. For the purposes of this paper, we will consider five axes of variation, although the concept can be easily extended to address further axes of variation.

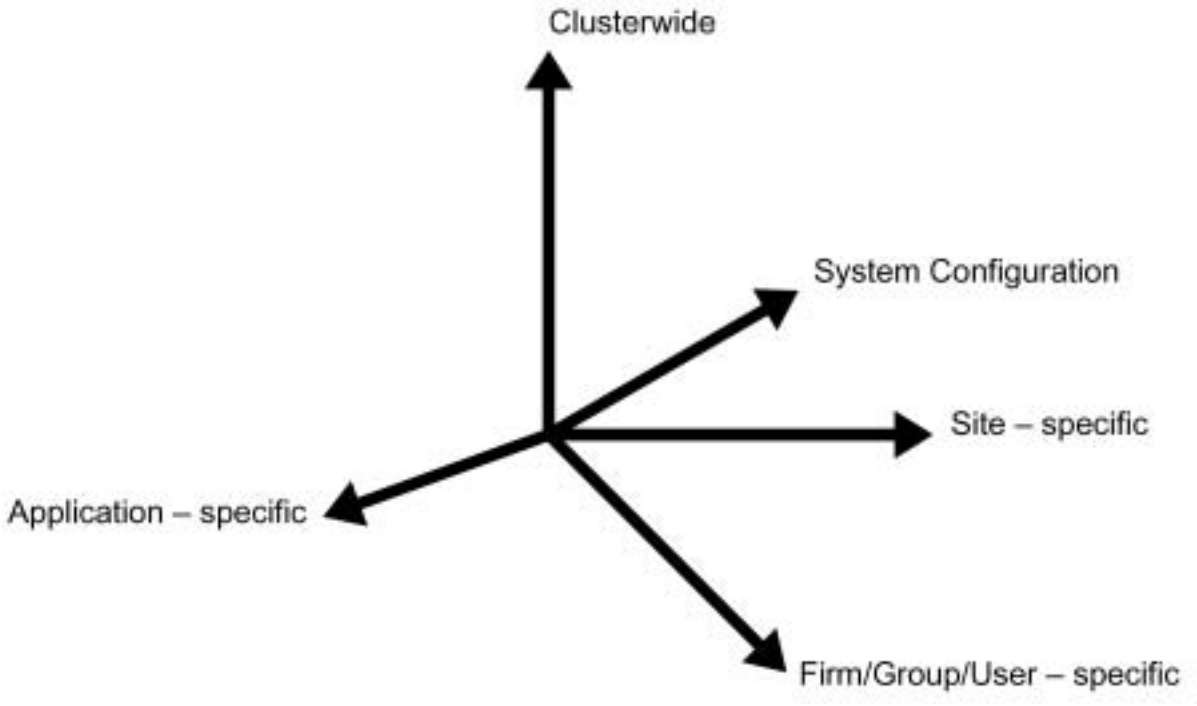

Figure 1 – Five illustrative independent axes of variation

The axes are considered independent; changes in the names comprising a single axis do not imply changes in different axes. There is also no need to restrict name translation iterations to a single axis.

There are five axes of variation referred to in this paper, namely:

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Clusterwide Variations

Clusterwide variations reflect those differences that define the individual cluster as a whole.

Clusterwide variances are best dealt with using the clusterwide logical name table.[7] The clusterwide local name table is automatically included in the logical name resolution path, **LNM$FILE_DEV**, at a lower precedence than that of the system-wide logical names contained in **LNM$SYSTEM**.

System Configuration Variations

System specific variations reflect the particular needs of a particular system and its hardware.

System specific variations are best dealt with by adding appropriate logical name definitions to the standard system logical name table, **LNM$SYSTEM**.

Site-Specific Variations

Site-specific variability reflects the connection and capability differences that exist on a specific site that is a member of a multi-site OpenVMS Cluster system.

While this axis of variation is not explicitly supported by OpenVMS, it can be added to OpenVMS by the addition of a new, system-wide logical name table inserted in the search path (**LNM$FILE_DEV** in **LNM$SYSTEM_DIRECTORY** or **LNM$PROCESS_DIRECTORY**) between the systemwide and clusterwide logical name tables.

Firm/Group/User-Specific Variations

Some variations are specific to an individual's organization, or place within the organizational hierarchy.[8] OpenVMS traditionally recognizes the group/department and user hierarchy with UIC-based protection and the existence of the group logical name table. This existing axis of variation can be extended and enhanced through slight adjustments to the login processing and the logical name search path.

Application-Specific Variations

Lastly, some environmental parameters are specific to a particular application. They may overmap similar parameters from similar applications, but differ in that each user of the application must have definitions in his or her logical name search path providing the value for these parameters. Thus, applications can be completely parameter-driven by their environment, with all of the implications of hierarchical defaulting.

---

[7] Ibid

[8] Within this paper, we refer to this as firm, group, and user. Additional levels (e.g., division or region) can easily be accommodated by similar means.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

One Parameter, One Line of Code

Another overall principal is that each definition should appear, like a common subroutine, only once. Duplicate definitions are a major element in maintenance complexity and costs, as well as an ongoing source of errors.

Dependency

The inclusion of all the definitions in the logical name search path allows different logical names to be phrased in terms of other logical names in the search path. As an example, the location of a file may be expressed as a logical name definition:

```
$ ASSIGN/PROCESS SYS$SCRATCH:VEHICLES.DAT DATABASE
```

which itself includes a reference to a definition in the user's process logical name context (to the logical name **SYS$SCRATCH**). In turn, **SYS$SCRATCH** may include a reference to the logical name **DISK$SCRATCH** which might be defined in the group's logical name table. **DISK$SCRATCH** could also be defined in the system logical name table**, LNM$SYSTEM**.

This is the same approach used by OpenVMS itself. Many logical names are explicitly or implicitly dependent on the definition of **SYS$SYSROOT**[9] or **SYS$SYSDEVICE**. In practice, multiple dependencies of logical names incur a generally insignificant cost.[10]

---

[9] Most of the **SYS$** logical names are expressed in just such a way, in terms of **SYS$SYSROOT**. For example, **SYS$SYSTEM** is defined as **SYS$SYSROOT:[SYSEXE]**. **SYS$MANAGER** is similarly defined as **SYS$SYSROOT:[SYSMGR]**. **SYS$SYSROOT** is defined as a search list including both the system-specific and clusterwide OpenVMS systems directories. Expanding this scheme to include additional levels beyond system-specific and clusterwide is straightforward.

[10] The processing costs associated with the rapid opening and closing of files and other operations is far more significant. In any event, the labor costs and inflexibility of the approaches required to save the logical name translations are far greater. In applications where thousands of records are processed, the cost of a few extra logical name translations is negligible.

Migrating logical names from private copies in **LNM$PROCESS** to **LNM$JOB** or other logical name tables with a wider purview increases performance by reducing the need to copy large numbers of process-private logical names during **SPAWN** operations.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Inheritance

Successive dependence, illustrated in the previous section, is a powerful technique. Defaulting, a concept implicitly familiar to OpenVMS users in the guise of default file types, and the traditional hierarchy of logical name tables, is a far more powerful mechanism than generally realized. Viewed as a form of inheritance, defaulting is also a mechanism for expressing the variability of user environments.

It has previously been remarked that each axis of variability embodies a hierarchical series of qualifications within the axis. On the individual axis, users are members of groups; groups are members of firms. On the cluster axis, the highest level is the cluster as a single entity; within the cluster, there are sites; within the sites, there are individual cluster nodes.[11]

Hiding the Physical and the Organizational

The purpose of isolating the physical and organizational aspects of a user's environment is the same as the more familiar OpenVMS concepts of disk space management (virtual blocks) and memory management (virtual memory). In the case of disk space and memory management, the purpose is to free the application from managing the details of a specific processor or device environment.

Adapting the same philosophy to users' logical environments similarly allows the re-organization of users and their hardware platforms without the need to explicitly re-engineer each and every reference to the environment.

[11] One could also argue that within nodes, there are individual realizations of nodes on particular hardware. For example, the node **ALPHA** could run at different times on either an ES40 or with a pre-configured alternative configuration for an Alphaserver 1200 as a backup. It is straightforward to implement such an environment, but does not affect the overall discussion.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

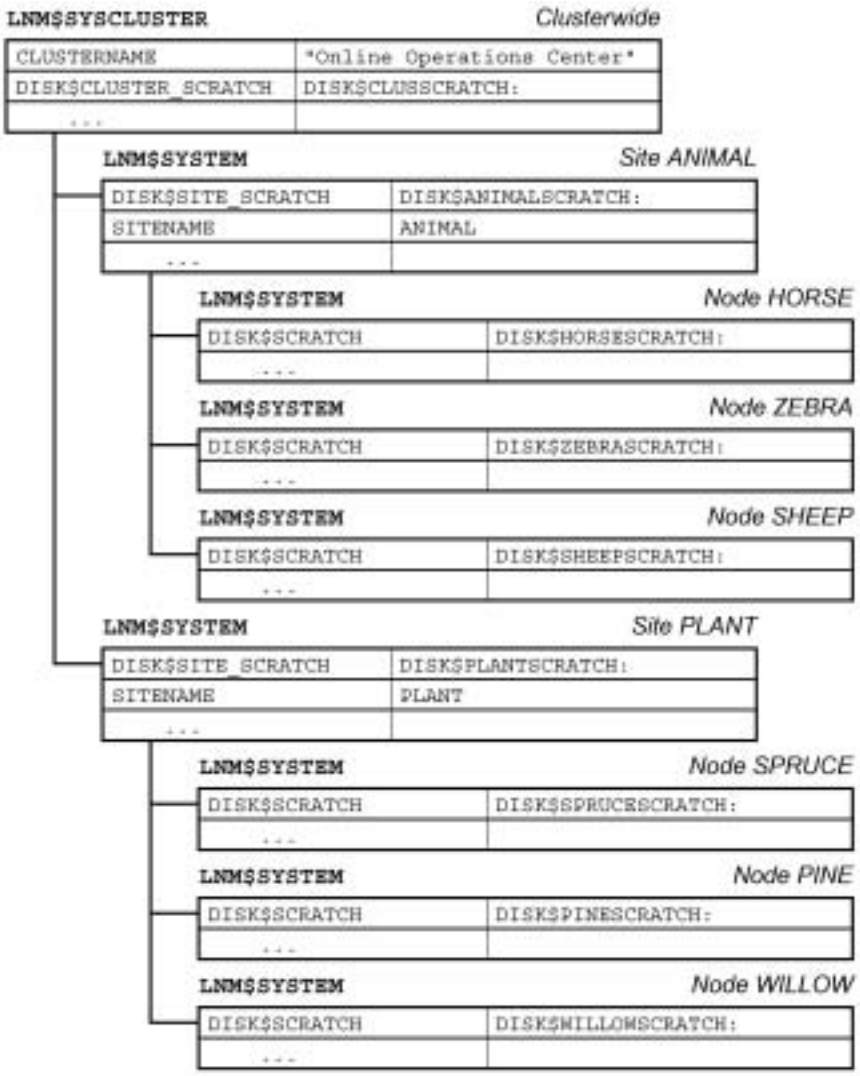

Figure 2 – Hierarchical dependencies and inheritance – Cluster/Site/System

© Copyright 2004 Hewlett-Packard Development Company, L.P.

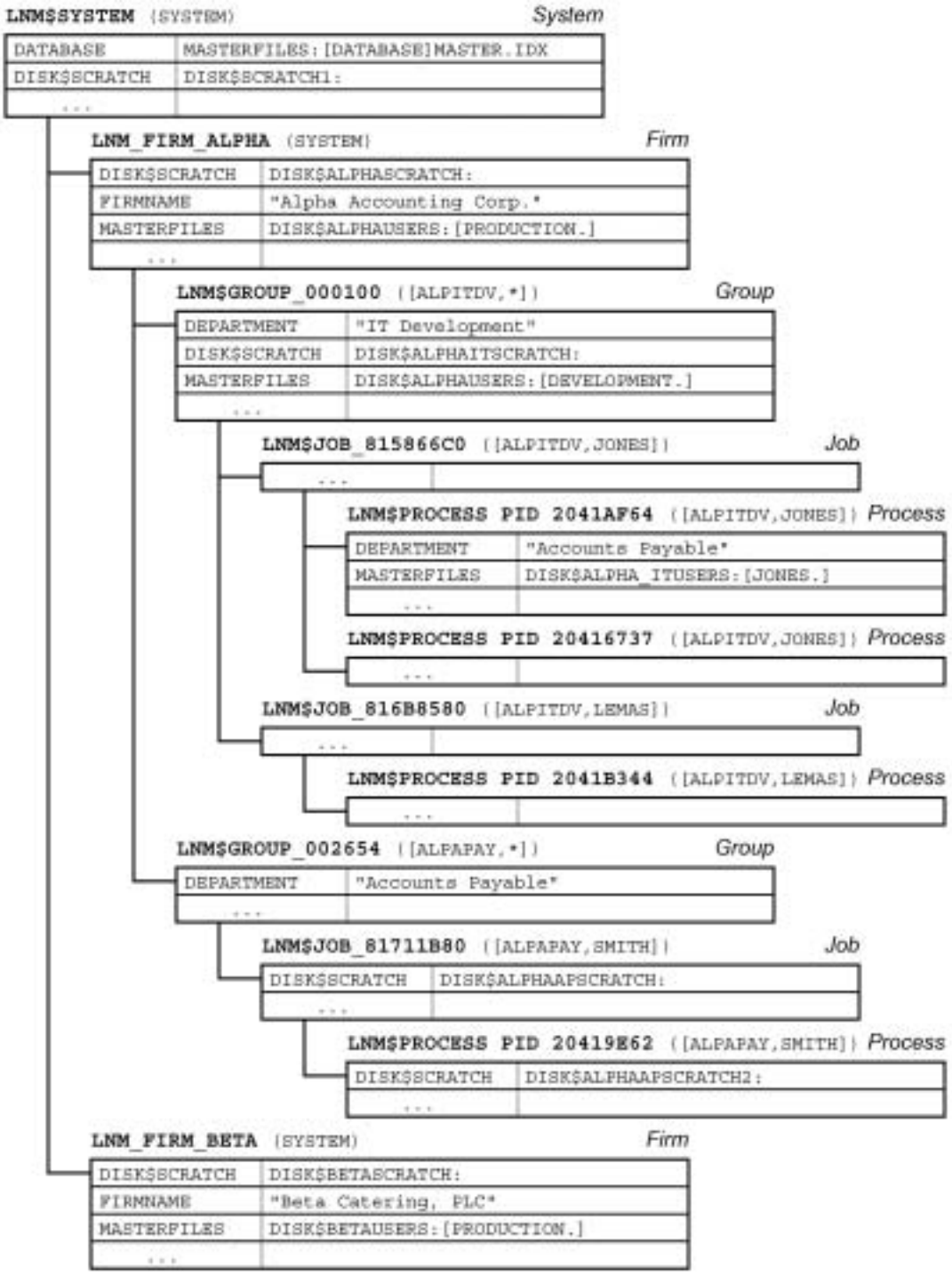

Figure 3 – Hierarchical Dependencies and Inheritance – Firm/Group/User

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Physical Configurations

Simple Configuration

Simple systems are employed for a variety of reasons, including cost and space. A large, complex system may not be justified or economically feasible for small organizations or for small applications. A small system may also be used in a large organization or for a large application to support development or to prototype applications.

Small systems, such as a DS10 or similar small workstation or server, are frequently used for testing installation, startup, shutdown, and restart procedures. These operations are extremely disruptive to a large production environment.

The affordability of relatively inexpensive, small OpenVMS platforms permits developers and maintainers to perform these highly disruptive tests with minimal impact on production systems and with a high degree of certainty that the full-scale tests will be successful.

Small systems are also used for projects in the proof-of-concept stage, where economics can make the difference between feasibility and infeasibility.

Advanced Configuration

Larger configurations present more options than small systems. While small systems may be restricted to one or more directly attached disks, a larger configuration may include a mix of directly attached disks (directly attached to the integral SCSI adapters on systems such as the GS-series and ES-series), local disks (CI or SAN attached), and remote SAN or network-attached storage. Each of these storage categories has different operational and performance characteristics.

Different user groups may be assigned to login environments with different attributes, depending upon a multitude of factors. Some factors will be technical in nature (such as a need for large scratch areas, or a need for shadowed and/or mirrored storage) and some will be organizational or political in nature (one department may have contributed the funds for a particular storage facility). In either event, the environment for one group of users (or in some environments, a particular user) will determine the need for that group’s default environment to differ from some other group’s default environment.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

User Disks

During login processing, **LOGINOUT.EXE** determines a user's default disk from the contents of the user's login profile, located in the user authentication file (referred to as the UAF). **LOGINOUT.EXE** populates **SYS$LOGIN** and **SYS$LOGIN_DEVICE** in the user's job logical name table as executive mode logical names (making them available to and usable by privileged images).

Scratch Space

Similarly, **LOGINOUT.EXE** uses the same information to populate the contents of **SYS$SCRATCH**.

Access to Data

When creating a user's process, **LOGINOUT.EXE** also attaches a series of rights list identifiers to the process. These identifiers come from two sources: a set of identifiers that reflect the origin of the process (e.g., **BATCH**, **INTERACTIVE**, **REMOTE**) and those identifiers associated with the user's UAF entry in **RIGHTSLIST**.[12]

[12] The rights list identifiers associated with a process are used to determine access rights to system resources beyond those granted through the normal System/Owner/Group/World and privilege access mechanisms. The details of the access checking scheme and how identifiers are used is described the "OpenVMS Guide to System Security".

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Default Inheritance

While not often appreciated, the OpenVMS login processing provides an easy and natural idiom for using at least three levels of inheritance, clusterwide, system, and group.

Concealed, rooted logical names,[13] coupled with the conventions for directory names in FILES-11, provide a natural naming convention and structure for inheritance, as illustrated in Figure 4.

Architectural Concepts

Architects understand that building designs must deal correctly with the forces of nature. A realizable design must always address the realities of construction, the details of the physical materials, and the techniques of fabrication. Designs do not exist in a vacuum, devoid of context.

Systems architecture is a hybrid discipline. In some sense, computer software is totally malleable, purely a creation of the mind of its creator. However, some architectural principles do apply. One such principle is that the overall architecture will only work if the details are correct, both architecturally and in the resulting implementation. For example, Digital's RT-11 (with the "**`DK:`**" device) and Microsoft Windows™ (with the "**`C:`**" device) have been hobbled when compared to a system, such as OpenVMS, that hides the identity of the system device behind a logical name.

The principles that apply to the quality of pictures when enlarged or reduced have their analog in the world of OpenVMS configurations. Reducing the size of a picture makes detail imperceptible, but it is still there. When enlarging a picture, detail cannot be invented.

The logical environment used for a larger, clustered system, can be easily reconfigured to transparently provide access to the same resources in a smaller workstation or development system (in effect, the analog to reducing a picture – no loss of picture quality occurs). However, an environment designed for a small system is frequently not suitable when used with a larger configuration (the analog of enlarging a picture – the picture loses sharpness and clarity as it is enlarged). The inherent problem is that the environment designed for the small system does not address the issues that occur in larger configurations.

To an even greater degree, different members of a cluster, whether a conventional OpenVMS Cluster system or a distributed, multi-site disaster tolerant OpenVMS Cluster system, offer the same challenge along a different axis.

[13] Much has been written on the use of concealed, rooted logical names, including this author's articles on OpenVMS.org, accessible directly or though the author's www site at http://www.rlgsc.com/publications.html.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

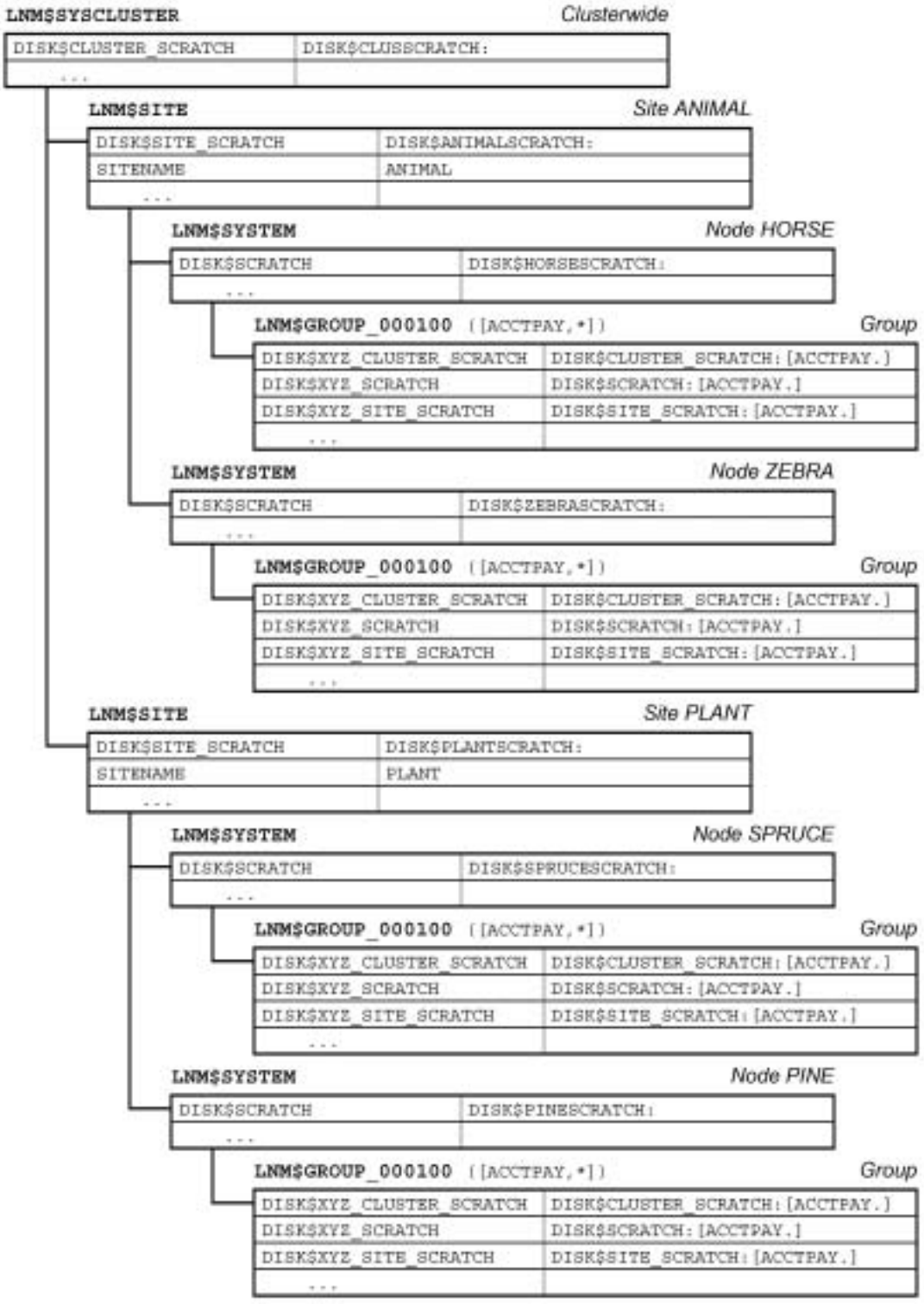

Figure 4 – Inheritance with rooted, concealed logical names.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Node	Translations according to Hierarchical Logical Name			
	Translation of SITENAME	Translation of DISK$SCRATCH	Translation of DISK$SITE_SCRATCH	Translation of DISK$CLUSTER_SCRATCH
HORSE	ANIMAL	DISK$HORSESCRATCH	DISK$ANIMALSCRATCH	DISK$CLUSSCRATCH
ZEBRA	ANIMAL	DISK$ZEBRASCRATCH	DISK$ANIMALSCRATCH	DISK$CLUSSCRATCH
SPRUCE	PLANT	DISK$SPRUCESCRATCH	DISK$PLANTSCRATCH	DISK$CLUSSCRATCH
PINE	PLANT	DISK$PINESCRATCH	DISK$PLANTSCRATCH	DISK$CLUSSCRATCH

Table 1 – Translations for the Accounts Payable Group ([ACCTPAY,*]) differ depending upon the contents of the hierarchical logical name table structure illustrated in Figure 4.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Developing a logical environment that is transparent across different systems requires care. In a clustered environment, the different member nodes must be fully interoperable, while still being appropriate for the different members and/or sites comprising the cluster. In both stand-alone and clustered systems, the logical environment must effectively embrace all operational issues without imposing unneeded presumptions of organizational structure or hardware configuration on any of the affected users (general users, developers, or system managers).

Issues

Different systems have their own issues, many of which are addressed through the judicious use of different elements in one of the login files, or the environment created by one of the elements of the STARTUP process.

Physical Machine Characteristics

Significant differences exist among system models. A small DS10 may be limited to an internal disk, and perhaps a small external storage shelf. A large GS1280 may have access to a complex SAN, as well as multiple local disks. Intermediate sized systems will be variations on the theme, with a hierarchy of storage, ranked by size, speed, latency, integrity, and cost.

Applications Impact

File placement depends upon use. Data files that are used throughout a cluster must be completely accessible throughout. On systems attached to a multi-site SAN, the straightforward choice for widely shared files is on SAN-mounted volumes.

Conversely, there are other files on the other extreme of the spectrum. Process-private temporary files are used only within a given image or process. A scratch file used by the SORT/MERGE utility is a common member of this class of files. Scratch files have no context or meaning outside of the moment, and need not be accessible to any other member of the cluster. Nor do they need backup, shadowing, or any other data protection scheme. If the process terminates for any reason, software or hardware failure, the files will, of necessity, be recreated when the process is restarted.

Often, temporary files are of substantial size. Placing them on remotely shadowed volumes, or volumes with full backup support, is a common source of overall system performance problems.

Physical Location

One of the strengths of OpenVMS is that it allows the programmer and system manager to generally ignore the actual location and configuration of mass storage. However, like any other virtualization scheme, this information cannot be ignored; performance and other issues merely move from one level to another.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

It is tempting, but overly simplistic to consider the use of a SAN as a solution to all of the performance questions which bedevil large configurations. While SAN technology is quite effective, it cannot change the laws of physics. Some correlation between the function of mass storage and its location is extremely beneficial to system performance.

Implementation Aspects

These issues, concepts, and concerns may seem abstract or theoretical. This is far from the case. Examining the issues and concepts in the context of a particular example will help bring them into focus.

Consider a firm with a disaster-tolerant OpenVMS Cluster system located at two sites, with two systems at each site. To illustrate the full range of issues, let us assume that each of the four systems is different, ranging from an enterprise-class machine at the high end (e.g. a GS80 or GS1280), to departmental-class machines (e.g., ES4x) to small servers (e.g., DS10/DS20). A SAN is deployed at each site, and the systems take full advantage of the mirroring and shadowing facilities of OpenVMS and the storage controllers to provide mass storage.

User-Specific Configurations

The default value for **SYS$SCRATCH** created by **LOGINOUT.EXE** is the user's default directory, as contained in the UAF. This is far from an optimal decision for several reasons:

- the user's default directory will likely be on a volume that is mirrored (within a site) and/or shadowed (between sites). While conceptually simple, the reality is that there is an overhead associated with both local mirroring and remote shadowing. In the case of shadowing, the reality is that the inter-site interconnect has a finite bandwidth far less than that available either within the computer room itself or on the system's local interconnect.

- scratch files are frequently high activity files.

- scratch files may be very large.

- scratch files generally have little meaning outside of the particular process or job that created them.

A straightforward way to address this situation, while simultaneously using the configuration to best advantage, would be to relocate the scratch directory somewhere else. One obvious place is locally connected disks, namely disks connected directly to the individual system. Whether the scratch disks support all processes on the machine or a single user is irrelevant. The critical issue is access to an appropriate scratch area through the **SYS$SCRATCH** logical name as translated from within a user's individual process context.

So far, it seems quite simple. While this issue can sometimes be addressed at the level of an individual user, it is more appropriate to address it in a hierarchical fashion. Scratch areas can generally be described on a group (department), set of departments, firm, or system basis, with only a small customization for the individual user.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Group-Specific Configurations

At the highest level of the scratch volume, create a series of directories, one for each department. These directories need to be accessible to all members of the group, either through rooted, concealed group logical name table entries for **DISK$SCRATCH**, or through rooted, concealed system logical name table entries for **DISK$GROUP_SCRATCH**, or other means.

In the **SYLOGIN.COM** file, we can redefine **SYS$SCRATCH** in the job logical name table (**LNM$JOB**) to point to the correct location.[14]

Thus, each group of users will seem to be pointing at their own scratch volume. For example, in the System or Firm Logical Name Tables we have definitions for **DISK$SCRATCH** as follows:

Group	Scratch Device	Value of **DISK$SCRATCH** in Group Logical Name Table
ITDevelopers	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[ITDEVELOPERS.]**
Accounting	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[ACCOUNTING.]**
Operations	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[OPERATIONS.]**

Table 1 – System/Firm definition variations in translation of DISK$SCRATCH

The preceding may appear rather obvious, and indeed it is a rather simple example. However, suppose that as the system activity increases, we realize that the space and performance requirements of the scratch space for accounting have been underestimated. We decide to allocate a dedicated stripe set to servicing the large scratch space requirements of the Accounting group. With the above structure, only a single logical name

---

[14] The **ASSIGN/JOB DISK$GROUP_SCRATCH:[USERNAME] SYS$SCRATCH** command defines a supervisor mode name in **LNM$JOB**. The definition of **SYS$SCRATCH** generated by **LOGINOUT.EXE** is in executive mode. In most user situations, this difference is of no import.

If the dichotomy caused by having the supervisor and executive mode logical names pointing to different directories is an issue, it is possible to enable the **CMEXEC** privilege in the user's default privilege field in the UAF (but not in the authorized privilege field). **SYLOGIN.COM** will then execute with the **CMEXEC** privilege initially enabled. In this case, **SYLOGIN.COM** should immediately redefine the **SYS$SCRATCH** logical name in **LNM$JOB** as an executive mode logical name, and then downgrade the process by removing the **CMEXEC** privilege with the **SET PROCESS/PRIVILEGE=(NOCMEXEC)** command. Alternatively, a small privileged image could perform the same functionality with more restrictions.

If done properly, the preceding is not a security hazard. However, one should exercise prudence. In most cases, **SYS$SCRATCH** can be defined as a supervisor mode logical name with no ill effects.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

needs to be modified, namely the definition of **DISK$NODESCRATCH** in the Accounting group's group logical name table as shown below.

Group	Scratch Device (value of **DISK$SCRATCH**)	Value of **DISK$SCRATCH** in Group Logical Name Table
ITDevelopers	**DISK$NODESCRATCH**	**DISK$SCRATCH:[ITDEVELOPERS.]**
Accounting	**DISK$NODESCRATCH1**	**DISK$SCRATCH:[ACCOUNTING.]**
Operations	**DISK$NODESCRATCH**	**DISK$SCRATCH:[OPERATIONS.]**

Table 2 – Translation of DISK$SCRATCH when System/Firm level definitions are overridden in the Accounting Department Group Logical Name Table.

The system does not need to be restarted, only the users in the Accounting group must be removed from the individual cluster member momentarily while the logical name is changed and the files migrated to the new location.[15] It is important to note that the impact of the change is very limited. Since the changes only affect the realization of the conceptual environment on the actual environment, only the system management group needs to be involved with this change.[16] Users will be unaffected, and in many cases unaware of the change, provided that they do not examine configuration-specific information not directly relevant to their applications.[17]

Site-Specific Configurations

The preceding discussion addressed user- or group-specific scratch file locations. A similar technique can be used to identify a site-specific resource, such as a site-local scratch disk.

In its most primitive sense, a site-specific scratch location can be identified by the creation of a single name identifying the site that the system is associated with.

---

[15] Though not strictly required for scratch files, it is highly recommended. Copying the current contents of the old scratch device to the new scratch device during the changeover should not take long, and will prevent many problems for users who use **SYS$SCRATCH** as a storage place for transient files (e.g., test files generated during debugging sessions).

[16] An automatic procedure is useful for generating the group directory trees from a reliable roster of group members (e.g., an automatically generated and parsed listing of the system UAF).

[17] While users can use system services and DCL lexical functions to see the actual difference in their environment, there is no normal reason for them to do so. Done properly, the differences should be transparent.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

SYS$MANAGER:SYLOGIN.COM (or the group logins, where appropriate) can then construct the logical name values appropriate for a particular site.

This approach allows the flexible provisioning of multiple levels of resources in a logically consistent manner. For example, three levels of scratch space can be made available to users and applications, each with different characteristics, as follows:

Name	Characteristics	Restrictions
DISK$SCRATCH	Machine local	Only available on a single system
DISK$SITE_SCRATCH	Site local	SAN connected, topologically local to each system
DISK$CLUSTER_SCRATCH	Clusterwide	SAN connected, shared with all cluster members

Table 3 – Scratch Resources by Location, Connection, and Accessibility

Company-Wide Defaulting

In a similar fashion, a hierarchical environment can be exploited to reduce the complexity and redundancy of applications. Suppose several sibling organizations share a hierarchically structured environment.[18] The same techniques traditionally used to support the differences between groups can be used to separate and support the parallel environments.

The parallel environments may represent subsidiaries or divisions of the same organization using common applications, different departments within a division, or customers of a service bureau. From an environmental perspective, the similarities far outweigh the differences.

In creating a hierarchical structure to bridge department-level differences, we would use the group logical name table **(LNM$GROUP**). Users, who belong to different, yet parallel environments, require us to create a new category or level,[19] for example Firm (in this narrative, we will refer to it as **LNM_FIRM**).

[18] Conceptually, it does not matter if the actual systems involved are separate stand-alone systems maintained by common managers or applications developers, a single consolidated server, or an OpenVMS cluster comprised of many individual systems.

[19] The logical name hierarchy in a baseline OpenVMS is cluster, system, group, job, and process.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Logical names reflecting the company-wide environment would be inserted into **LNM_FIRM**. In **SYS$MANAGER:SYLOGIN.COM** we would insert **LNM_FIRM**[20] in the logical name search path (**LNM$FILE_DEV** in **LNM$PROCESS_DIRECTORY**) between the group and system-wide logical name tables. Each group would be uniquely identified with a particular firm.[21]

It is admittedly simplistic, but this structure allows the creation of parallel application environments with minimal effort and minimal code differences between different branches of the tree.

The same process can be used to implement testing environments. It is admittedly expansive, but this approach can leverage the seemingly simple OpenVMS UAF, rights list, and logical name facilities to support large numbers of parallel development, test, and production environments for similar yet separate organizations on a single integrated OpenVMS Cluster system.

Company-Wide Constant Data

It is obvious that brief information specific to a particular sibling company can be stored in **LNM_FIRM**. Examples of such information are the name of the firm or the locations of company-wide resources or files.

Application-Wide Defaulting

The same principles that apply to individual subsystems apply to individual users or groups of users. The names of files and commonly used constants can be contained in an application-specific logical name table, and that name table can be inserted in one of the active search paths.[22] The benefits of this technique include:

- easier maintenance – only one copy of the definition to use
- fewer logical names in the process or job logical name tables[23]
- faster logins as the command files defining the logical names need not be executed at each login

---

[20] Paralleling the design of the OpenVMS logical name facility, **LNM_FIRM** would be a name located in **LNM$PROCESS_TABLE** containing a pointer to the name of the actual firm-wide logical name table for that particular process. The protection on **LNM_FIRM** must also be set appropriately.

[21] Each group belongs to an identifiable firm. Thus, it is possible to identify the correct firm-wide logical name table through a group-wide login script, the contents of the group logical name table, a rights list identifier, or a file in a common group-wide directory.

[22] If an entire community makes use of a particular application, it may make sense for that application's logical name table to be included at a higher level in the hierarchy than an individual user (e.g. the firm or group).

[23] Admittedly, memory consumption for logical name tables is not the concern for system performance that it once was. However, 200 logical names for each user on a large machine is still a potential performance issue when each of several hundred users defines a full complement of the names.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- faster **SPAWN** operations, as the voluminous process logical name table need not be copied to the sub-process each time a process is spawned.

Logical names used by different applications should not overlap. If the logical names used by pre-existing applications overlap, changing the sequence of name tables in **LNM$FILE_DEV** can be used to resolve the problem.

System Resources

The location of scratch space is but one example of an instance where the location or identity of system resources can be managed through the use of logical names.

Naming Conventions

Names that will be used globally should be named separately from names that are unique to a particular process or application. Care should be exercised to allow the same mechanisms to be used by different groups or ISVs. One possible way to avoid naming conflicts is to use the registered Internet domain names as the leading part of the logical name.

Summary

Presenting a conceptually consistent, although not necessarily identical user-perceived environment, is a powerful approach when implementing OpenVMS systems, whether stand-alone or as members of OpenVMS Cluster systems.

Hierarchical environments provide a powerful way to express the differences while retaining common elements. The source of the differences does not matter. The differences may be matters of mass storage configuration, as in the **SYS$SYSR00T** hierarchy used by OpenVMS itself, or the differences may reflect the management structure of the organization. Hierarchical environments allow the differences between disparate systems, and the differences in organizations, to be hidden from users and applications. The greater the disparity between the underlying systems or organizations, the greater the leverage of using different, yet conceptually identical, environments to provide users and applications with a perceived identical computing context.

Using inheritance to dynamically instantiate logically identical user environments on dramatically different systems simplifies system management, reduces the cost of system management, and increases system availability. This method can be of even greater use to the end-user and system manager than to the base operating system and layered products.

Bibliography

HP OpenVMS DCL Dictionary: A–M, © 2003, Order #AA-PV5KJ-TK, September 2003

HP OpenVMS DCL Dictionary: N–Z, © 2003, Order #AA-PV5LJ-TK, September 2003

© Copyright 2004 Hewlett-Packard Development Company, L.P.

HP OpenVMS Systems Manager's Manual, Volume 1: Essentials, Order #AA-PV5MH-TK, September 2003

HP OpenVMS Systems Manager's Manual, Volume 2: Tuning, Maintaining, and Complex Systems, Order #AA-PV5NH-TK, September 2003

HP OpenVMS System Services: A–GETUAI, © 2003, Order #AA-QSBMF-TE, September 2003

HP OpenVMS System Services: GETUTC-Z, © 2003, Order #AA-QSBNF-TE, September 2003

OpenVMS Guide to System Security, Order # AA-Q2HLF-TE, June 2002

OpenVMS User Manual, Order #AA-PV5JD-TK, January 1999

OpenVMS Version 7.2 New Features Manual, © 1999, Order #AA-QSBFC-TE

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 1", http://www.openvms.org/columns/gezelter/logicalnames1.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 2", http://www.openvms.org/columns/gezelter/logicalnames2.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 3", http://www.openvms.org/columns/gezelter/logicalnames3.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 4", http://www.openvms.org/columns/gezelter/logicalnames4.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 5", http://www.openvms.org/columns/gezelter/logicalnames5.html

Goldenberg, R, Kenah, L "VAX/VMS Internals and Data Structures – Version 5.2", © 1991, Digital Equipment Corporation

Goldenberg, R, Saravanan, S "VMS for Alpha Platforms: Internals and Data Structures, Preliminary Edition, Volume 3, © 1993, Digital Equipment Corporation, ISBN#1-55558-095-5

Goldenberg, R, Dumas, D, Saravanan, S "OpenVMS Alpha Internals: Scheduling and Process Control", © 1997, Digital Equipment Corporation, ISBN#1-55558-156-0

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

VAX Emulator on HP's Latest AlphaServer Products Extends Life of Legacy OpenVMS VAX Systems

Robert L. Lyons
Systems Consultant, Resilient Systems, Inc.

Overview

Strong synergies in the latest technologies from Hewlett-Packard and Software Resources International promise not only a reprieve for remaining OpenVMS VAX systems but also a clear path to 21st century platforms. OpenVMS VAX users can now take advantage of ever-faster chip speeds and massive storage platforms available on the latest HP AlphaServer systems. This article describes tests of the Software Resources International CHARON™-VAX/AXP *Plus* emulator on an HP GS1280, the most powerful of the recently released EV7-based AlphaServer systems. CHARON-VAX/AXP *Plus* emulates an OpenVMS VAX system to replace VAX hardware systems with modern OpenVMS Alpha systems. The emulator runs as an application on the Alpha system, providing a 'virtual' OpenVMS VAX system that can directly execute OpenVMS VAX system operatives, the layered software, and user applications without requiring conversion.

The tests show that the combined strengths of the CHARON-VAX/AXP *Plus* and the HP GS1280 AlphaServer provide server consolidation and multi-architecture cluster support while significantly enhancing the performance of OpenVMS VAX system functions and applications running on the Alpha hardware. The tests demonstrate that a 16-way GS1280 running multiple instances of CHARON-VAX/AXP *Plus* can deliver the equivalent of an OpenVMS VAX 3198 or VAX 7610 on each GS1280 Alpha processor, while achieving nearly linear scaling. In addition, the software/hardware configuration minimizes the floor space required: only one Alpha footprint for up to 15 VAX systems emulated on the 16-way GS1280. Testing also shows that the emulator software running on a GS1280 can replace multiple high-end VAX systems. With the recent release of the 64-way GS1280, the potential for the coupling of the CHARON-VAX/AXP *Plus* and GS1280 products is spectacularly promising.

Background

This article describes the results of tests performed by Resilient Systems at Hewlett-Packard's Littleton, Massachusetts laboratory in the spring of 2003, using hardware and test suites provided by OpenVMS Engineering. The tests were performed on a 16-way GS1280 AlphaServer running multiple instances of CHARON-VAX/AXP Plus.

The test suite was the same one that OpenVMS Engineering used in previous decades to test new VAX hardware designs. To ensure proper execution of the VAX instruction set, the tests verify conformance of the new hardware to expected test results. The comprehensive suite exercises nearly every VAX instruction, including all three-operand VAX instructions as well as single, double, and floating-point calculation speeds. For some instructions, the tests revealed that the CHARON-VAX/AXP Plus emulator was more than ten times faster than any real VAX processor.

In addition to this suite, Resilient Systems used the VUPs Calculator utility to test individual CPU performance in processing a mix of fixed and floating point instructions. To test scalability of CHARON-VAX/AXP Plus on multiprocessor configurations, Resilient used standard Dhrystone tests because the results are much more granular than those produced by the VUPs Calculator. Resilient Systems first determined peak Dhrystones on a single CPU and then ran simultaneous Dhrystone tests on multiple instances of CHARON-VAX/AXP *Plus* in multiple *n-way* GS1280 configurations,

© Copyright 2004 Hewlett-Packard Development Company, L.P.

with up to 18 instances on a 16-way GS1280. Resilient Systems repeated these tests over three days, and then calculated the average Dhrystone performance. Resilient Systems then used the standard formula to convert Dhrystones to VUPs, and produced detailed graphs to show the results (the graphs are shown later in this article in Figure 4 and Figure 5).

The complete results from the OpenVMS Engineering test suite; the CPU, disk, and network performance tests; and the CHARON-VAX/AXP *Plus* scalability tests are available from Resilient Systems.

The next two sections describe the benefits of (1) the latest release of HP AlphaServer products (and the GS1280 model in particular) and (2) the CHARON-VAX/AXP *Plus* emulator, respectively. The third section to follow provides details about the cumulative performance benefits of these two products combined.

The Marvels of the GS1280 AlphaServer

Based on the new EV7 processor, the GS1280 AlphaServer, as well as the smaller ES47 and ES80 models, deliver an unprecedented combination of performance, scalability, and system reliability. The products deliver numerous architectural advancements over previous switch-based, non-uniform memory access (NUMA) systems such as the 32-way GS320.

High-Performance Chip Technology and Scaling Advancements

All elements required for symmetric multiprocessing now reside on a single chip. In addition to an on-chip L2 cache, two on-chip memory controllers provide exceptional memory bandwidth. In an industry-first achievement, an on-chip router connects AlphaServer processors directly to one another. This "switchless" mesh design results in a very high interconnect bandwidth of up to 64 CPUs. SPEC_rate 2000 tests on a 32-way GS1280 system proved that the GS1280 can achieve nearly 100% linear scalability. In other words, as the number of processors used on the GS1280 increases, the performance increases at a constant rate.

The I/O performance and scaling improvements of the GS1280 over the GS320 are equally impressive. The GS1280 provides flexibility in configuring I/O, from one I/O chip per system to one I/O chip per processor. The result is a platform with linear scaling in I/O, yielding eight times the I/O bandwidth of the GS320. Moreover, the GS1280 Lego™ block design of hot-swappable components results in a robust platform with 15% to 30% improvement in Mean Time Between Failure (MTBF) over the previous generation of AlphaServers. Available in multiple processor (*n*-way) configurations, the enterprise-scale AlphaServer GS1280, along with the departmental and workgroup ES80 and ES47 models, provide significant performance and reliability improvements over the earlier GS320 and ES45 models.

Partitioning Enables Support of Mixed-Architecture Clusters in a Single AlphaServer Box

With the new ES47, ES80, and GS1280 AlphaServer products, HP introduced support for hardware partitions. Hardware partitions permit multiple instances of the OpenVMS operating system to run concurrently in physically separate parts of the computer. Such a configuration facilitates the dedication of partitions to specific applications, with the ability to tune and secure each partition to the specific demands of its application set. By effecting the partition of the system into multiple independent Alpha processors, this new feature facilitates the deployment and execution of multiple instances of CHARON-VAX/AXP *Plus*. CHARON-VAX/AXP *Plus* can be run as an application on one or more of the CPUs in a processor partition, or across multiple CPUs in multiple partitions.

This allows construction of a variety of mixed VAX and Alpha configurations, all within a single system cabinet. For example, by using the 8-way AlphaServer depicted in Figure 1, we can partition it into three separate Alpha systems, forming a consolidated mixed-architecture

© Copyright 2004 Hewlett-Packard Development Company, L.P.

configuration as shown in Figure 2. The Alpha system constructed from the five CPUs (shown in yellow) could be configured to run multiple CHARON-VAX/AXP *Plus* instances.

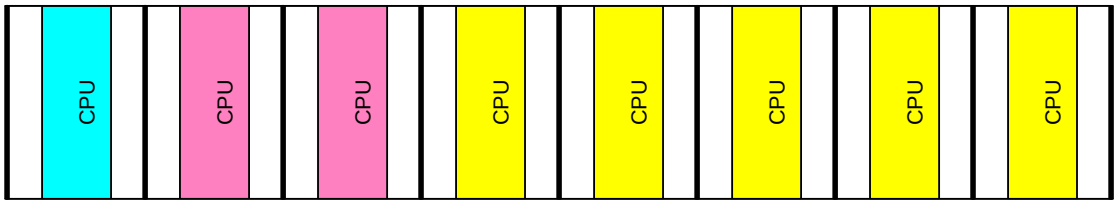

Figure 1 Multi-CPU Alpha Processor

The other two AlphaServer partitions could be added to create the 6-node, mixed-architecture cluster, shown in Figure 2. In this configuration, three of the eight available CPUs would run as actual Alpha nodes — one single CPU node (blue) and one dual-CPU multi-processor node (pink). The remaining five CPUs (yellow) would run four instances of CHARON-VAX/AXP *Plus* as four VAX nodes, with the fifth CPU (labeled in the figure as the "Alpha Management CPU") fielding user interrupts and managing disk and network I/O as described in the next section.

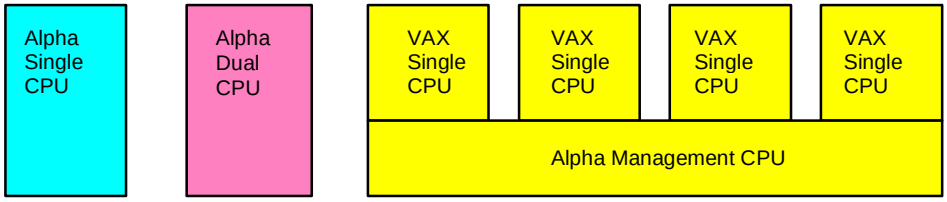

Figure 2 Mixed VAX and Alpha Cluster Example

The CHARON-VAX/AXP *Plus* Benefits for OpenVMS Customers

The new VAX-on-Alpha emulator from Software Resources International takes full advantage of the revolutionary improvements of HP's EV7-based AlphaServer systems. Software Resources International specializes in migrating operating systems and applications to modern platforms (for example, migrating OpenVMS Alpha systems to OpenVMS I64 systems based on the Intel Itanium architecture) and developing hardware emulators for PDP and VAX processors. The emulators are mathematical models of the hardware architecture; written in C, they run as ordinary applications on modern platforms.

Figure 3 illustrates how the Software Resources International CHARON-VAX/AXP *Plus* emulator running on an OpenVMS Alpha system replaces the OpenVMS VAX hardware, providing the same operating system functionality and application support. The OpenVMS VAX software, layered software, and user applications are installed onto the CHARON-VAX/AXP *Plus* emulator which is running on the AlphaServer, which in turn is running its own copy of OpenVMS. With CHARON-VAX/AXP *Plus*, no conversion of code is needed. Simply use BACKUP/IMAGE to transfer existing OpenVMS system and application binaries to the CHARON-VAX/AXP emulator running on the OpenVMS Alpha system, as if you were simply moving from one VAX model to another.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

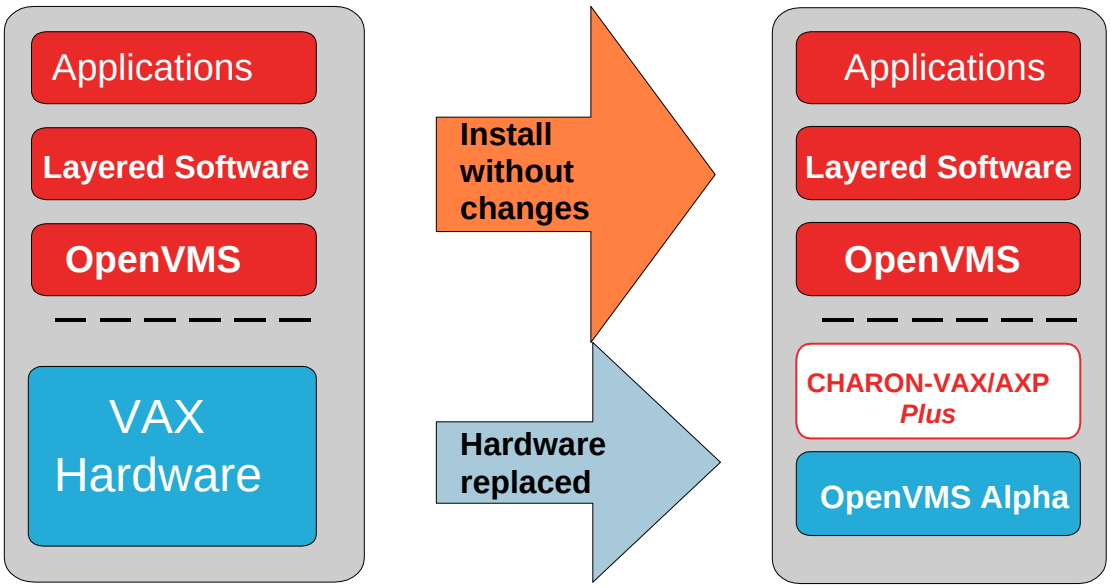

Figure 3 CHARON-VAX/AXP *Plus* on an HP AlphaServer Easily Replaces VAX Hardware

The software architecture of the CHARON-VAX/AXP *Plus* emulator consists of two threads — one thread to execute the emulator and a second thread to field interrupts, run the scheduler, manage resources, handle I/O to storage devices, and manage network I/O. While you can run both threads on the same processor, for optimum performance the emulator thread should have 100% of a CPU available to it. The second thread, automatically assigned to a separate CPU when one is available, requires a fraction of the compute power available to it.

CHARON-VAX/AXP *Plus* is the second-generation of Software Resources International's VAX hardware emulator for Alpha. The first emulator modeled a MicroVAX 3600. The new emulator provides the functionality of a VAX 3100 Model 98 hardware system, complete with up to 512 MB memory, dual SCSI storage buses, and a 10/100 Mbps Ethernet network.

Combined with the scalability and reliability of the ES47, ES80, and GS1280 AlphaServer products, the sophisticated instruction preprocessing now provided by the CHARON-VAX/AXP *Plus* emulator has significantly increased the viability of preserving business-critical VAX applications by means of VAX emulation. As a result, one or many low to mid-range MicroVAX processors can be replaced by entry-level ES47s. Testing has now shown that the CHARON-VAX/AXP *Plus* emulator running on an *n-way* GS1280 can replace one or more high-end VAX processors, such as VAX 77*xx*'s or VAX 78xx's.

Significant Results: Proven Performance Worth Emulating

Specifically, the testing conducted by Resilient Systems at HP's Littleton, Massachusetts laboratory proved that a 16-way GS1280 running CHARON-VAX/AXP *Plus* delivers the equivalent of a VAX 3198 or VAX 7610 (over 36 VAX units of performance (VUPs)) on each CPU of the AlphaServer system.

Even more impressive, the remarkably efficient CHARON-VAX kernel (0.5 MB) achieved the same scalability as the underlying GS1280 hardware when running multiple instances of CHARON-VAX/AXP *Plus*. As the graph shows in Figure 4, the compute power (measured in VUPs) obtained by running multiple instances of the VAX emulator scaled nearly linearly. In other words, each additional emulator instance adds nearly the same amount of compute power to the cumulative

© Copyright 2004 Hewlett-Packard Development Company, L.P.

stacked bar graph even though the instances are all competing for resources from the same Alpha management CPU.

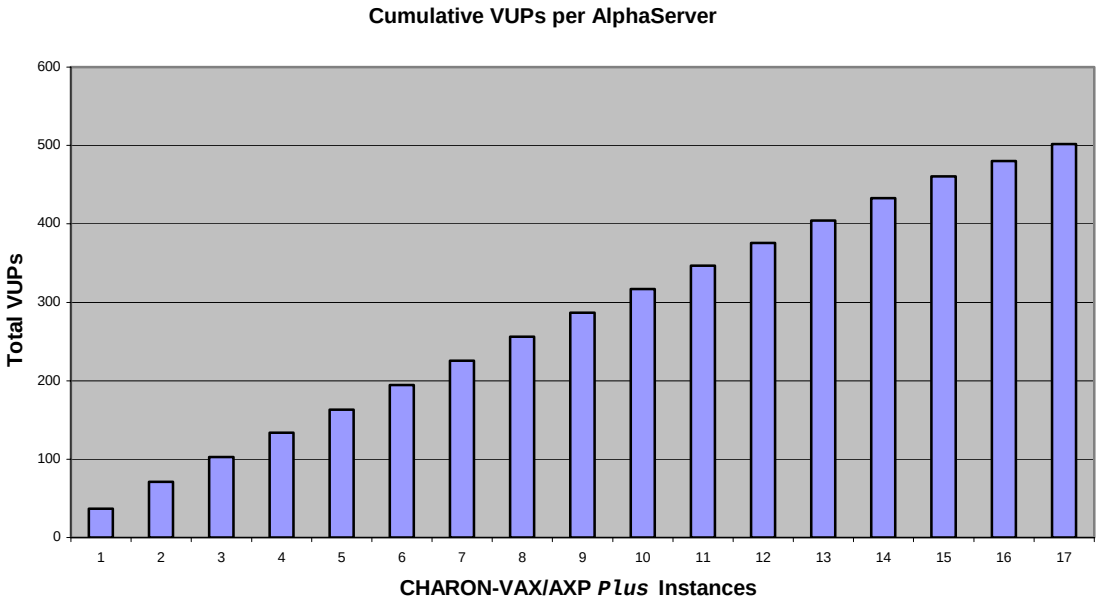

Figure 4 Compute Power of Multiple Instances of CHARON-VAX/AXP *Plus* Scales Nearly Linearly

The remarkable synergy between the hardware architecture of the HP GS1280 AlphaServer and the software architecture of CHARON-VAX/AXP *Plus* produced an optimum configuration of 15 instances of the emulator on a 16-way AlphaServer, with the 16th CPU managing resources for the other 15.

Specifically, tests proved each instance of CHARON-VAX/AXP *Plus* delivers an average of 32 VUPs on an AlphaServer with 16 CPUs, each CPU independently running an instance of the emulator. As the graph demonstrates in Figure 5, performance remained above 30 VUPs per CPU except when the number of CHARON-VAX executables exceeded the physical number of CPUs.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

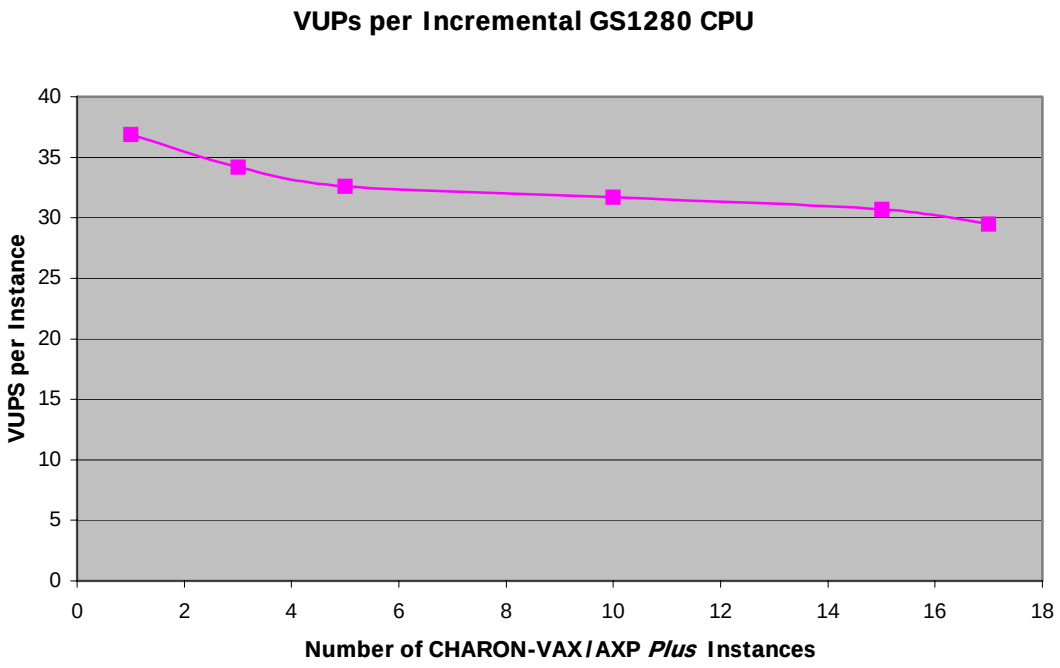

Figure 5 Incremental VUPs per Instance of the CHARON-VAX/AXP *Plus* Emulator

The Practical Benefits of Consolidated Power

The results of these tests clearly indicate that multiple individual VAX servers or VAXstations could easily be consolidated on the same GS1280 AlphaServer host. **Server consolidation** offers many benefits to VAX sites with multiple systems, including reduced footprint and power consumption, and greatly reduced hardware maintenance costs. In addition, the superb reliability and MTBF of the GS1280 AlphaServer reduces staffing requirements and dependence on increasingly scarce VAX/VMS system engineers, while also reducing the risk of business disruption due to malfunction of aging hardware.

Similarly, **single platform clusters** can now be created — an entire cluster of existing VAX processors could be recreated as multiple cluster members of the same cluster, all residing and managed on a single GS1280 host.

Alternatively, the configuration could be aggregated and then spread over redundant GS1280 systems to attain the highest possible availability through the independence of separate hardware systems. The benefits would include all the benefits of server consolidation described above, plus the failover capability inherent in OpenVMS Clusters.

Note that in either scenario — server consolidation or single platform clusters — you must carefully calculate your VUP performance needs to ensure that the number of CHARON-VAX/AXP *Plus* instances on the GS1280 stays within the recommended limits. The chart in Figure 6 illustrates the cumulative capacity (in VUPs) of the various AlphaServer platforms, with the maximum capacity being provided by the GS1280 (16 instances producing approximately 460 VUPs).

© Copyright 2004 Hewlett-Packard Development Company, L.P.

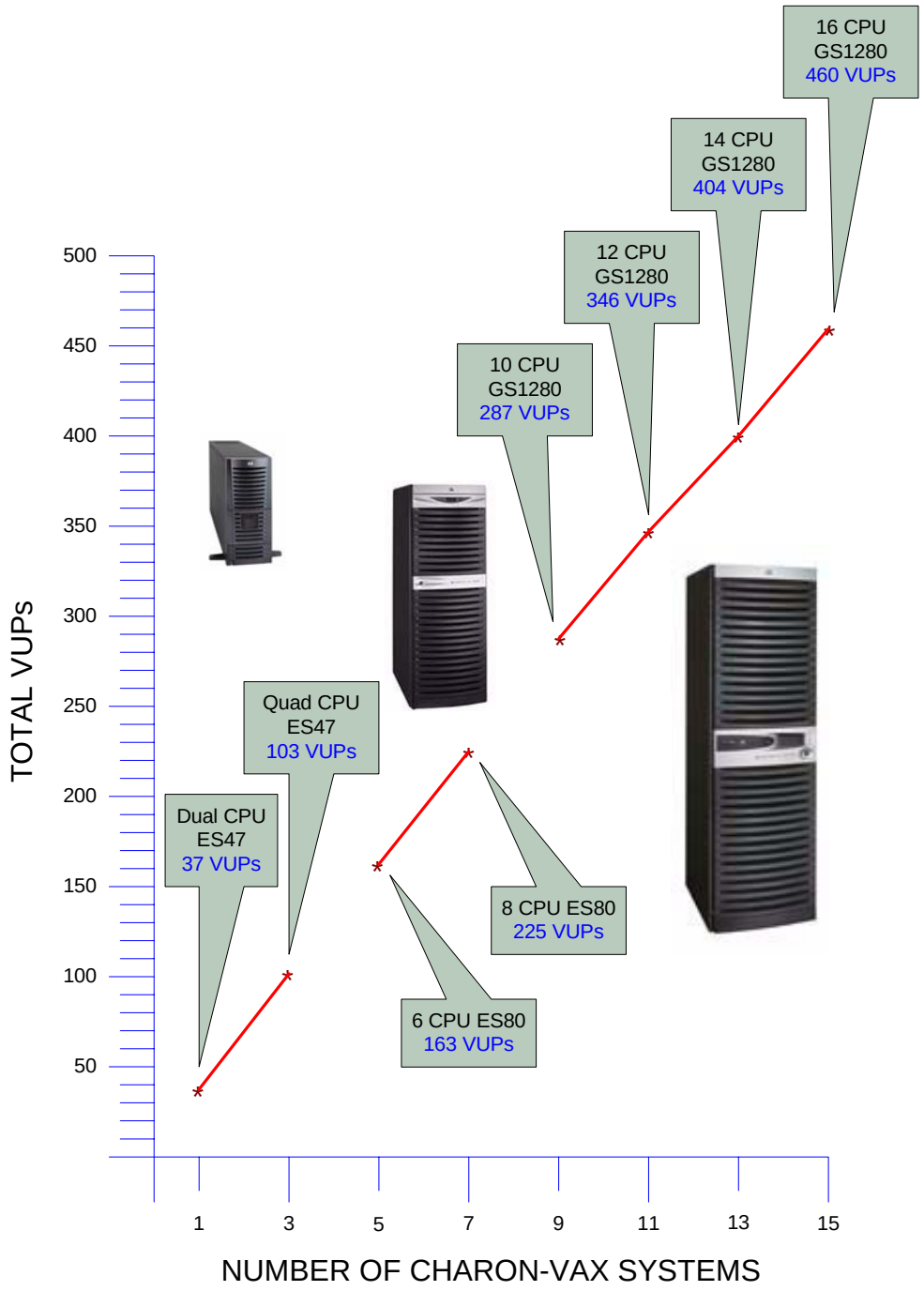

Figure 6 Cumulative Capacity of Various AlphaServer Platforms Running Multiple Instances of CHARON-VAX/AXP *Plus*

CHARON-VAX/AXP Emulator I/O Capacity Keeps Pace with Hardware

As more evidence of the synergy between the GS1280 hardware and the CHARON-VAX/AXP *Plus* software, tests show that the bandwidth available to the emulator is nearly identical to what is physically attached to the AlphaServer host. Repeated testing showed that native Alpha disk transfers achieved 4.47 MB/sec when accessing a local SCSI disk versus 4.45 MB/sec for the CHARON-VAX/AXP *Plus* emulator when accessing the same physical disk. In other words, emulator

© Copyright 2004 Hewlett-Packard Development Company, L.P.

overhead is less than 1% for tasks such as disk-to-disk file copy operations or OpenVMS backup transfers.

The tests prove that customers are now able to assimilate high performance storage subsystems, such as Fibre Channel, into a legacy OpenVMS VAX configuration. The CHARON-VAX/AXP *Plus* software increases storage capacity by transforming VAX physical disks into disk image files on the replacement platform. Now, with the HP AlphaServer's support for robust storage technologies, the CHARON-VAX/AXP *Plus* software enables critical VAX applications to take advantage of both storage and I/O throughput capacities that were unimaginable in the heyday of the VAX processor.

The integrated Ethernet adaptor emulator provided with the CHARON-VAX/AXP *Plus* product is pivotal to integrating an instance of the emulator with other DECnet nodes and cluster members, or by means of IP, with corporate LANs and WANs. This channel also provides user connectivity through Telnet and third-party terminal emulators. Thus, it is a key component of a VAX replacement configuration. When the adaptor emulators were set in tests to match the 10baseT adapter of a VAX system, Resilient Systems observed data rates through the network device at over 1.8MB/sec for sustained data transfer, and near the full 10MB/sec possible for message transfer. You can use 100 Mbps Ethernet adapters with the current version of CHARON-VAX/AXP *Plus*, but these were not tested. Operation at 100 Mbps requires an Alpha SMP host with a CPU frequency of at least 1 GHz. Network throughput can be tuned individually for specific protocol classes (for example, DECnet, TCP/IP, or OpenVMS Cluster communication).

Summary

The combination of HP's AlphaServers and Software Resources International's CHARON-VAX/AXP *Plus* software opens the way for many OpenVMS VAX owners to achieve the performance and reliability experienced using modern-day server hardware platforms. OpenVMS VAX owners can now consolidate servers and clusters to preserve critical OpenVMS VAX applications and maximize performance and reliability while minimizing the space required for hardware.

For more information about Resilient Systems, Inc., go to http://www.resilientsys.com.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Structured Programming in Assembly Language

Author: Dick Munroe
Cottage Software Works, Inc.

Overview

The stored program computer in its modern form was developed in the late 1940s. About 15 minutes after the first program was written (using patch panels and/or toggle switches on the front panel) engineers started looking for ways to make writing programs easier. Some of the initial attempts at making programming easier were what are now considered third generation languages, e.g., Fortran II. However that still left the problem of writing low level (close to the hardware) code. Fortran and the other languages developed at the time were unsuited to the problem of dealing directly with hardware. As a result, low level languages, more or less universally called assembly languages, were developed. These languages shared a number of common characteristics. These are:

- One-for-one mapping from operands in the language to machine instruction
- Direct access to hardware resources,such as registers, I/O mechanisms, etc.

And they made programming at the lowest level of the computer substantially easier. They worked.

Over time, these assembly languages acquired additional mechanisms for making programming easier, most notably a macro processing capability through which programmers could extend the macro language. Most frequently "macros" were used to capture frequently-used code sequences, for example, saving registers at the entrance to routines, restoring them at return, etc. Ultimately the purpose of a macro was to reduce the opportunity for making mistakes. By putting common instruction sequences into macros, a whole class of errors was eliminated making the writing of assembly language programs more reliable and faster.

At the same time, substantial research was being done at the higher levels of programming (what most would consider application programming) resulting in a variety of more or less general purpose programming languages, Fortran 4, PL/1, Algol, C, Pascal, and a veritable tower of Babel of others. All of these had interesting features and approaches with respect to how programs were written. For example, Fortran included statements that were equivalent to the ubiquitous compare/branch assembly language instructions (IF (A .EQ. B) GOTO …). This was probably one of the very first uses of a "design pattern" in the history of modern computing.

Most interesting of all was the gradual elimination of the branch instruction in many of the languages (notably Pascal which takes the elimination of GOTO to ridiculous lengths). The GOTO or branch language construct was replaced by a variety of flow of control constructs such as:

- IF THEN ELSE
- FOR loops
- DO WHILE/UNTIL loops

The "elimination" of the GOTO and the use of the alternative flow of control constructs became known as "structured programming." Structured programming is probably the single biggest contributor to the quality and quantity of code produced since the 1960s. There are several reasons for this and they will be discussed later.

However, assembly language programmers (and there are a lot of us, though fewer now with the advent of the RISC machine and the use of higher level languages for OS and driver development) largely missed the advantages of structured programming.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

This article discusses the benefits of structured programming and how to do structured programming in assembly language, specifically Macro-32. The techniques discussed here have been used in a number of real time environments on a variety of platforms.

What is structured programming?

Well there are a lot of answers to that question. The most common one is "goto-less" programming.

In fact, "structured programming" is little more than an enforced discipline that encodes information directly into the "structure" of a program that makes some of the characteristics of the program easier to understand by a programmer other than the original author. This is also useful if the original author returns to that program after a substantial hiatus.

In the case of most higher level languages, the discipline is enforced by the language. For example, Pascal literally has no GOTO in the language. C and C++ do have GOTO but also have flow-of-control constructs that mitigate their use. In both languages, it is possible to write well structured, easy to understand programs. It is also possible to write well structured, difficult to understand programs. For good examples of such things, see the winners of the obfuscated C contest held yearly. The obfuscated C programs all work, many do useful things, and all are virtually incomprehensible by design.

So, goto-less programming is not a panacea. What use is it then?

Without discipline, none.

What structured programming enables is the ability to encode additional information into the body of the program that makes it easier to understand. Specifically it let the programmer encode a visual representation of the flow of control. In turn, this lets the programmer use extra pieces of his/her brain to understand the program. The additional information is encoded by indenting the bodies of the various flow-of-control constructs so as to make them stand out visually. Thus the visual cortex is engaged to help understand the program. Humans evolved using their eyes to detect predators and a substantial portion of the brain is dedicated to visual processing. As a consequence, anything which adds visual information to a program makes understanding the program easier because more of the brain is used when working with the program.

Of course the visual information must be added consistently, otherwise the programmer's eyes get confused and the additional information can be obscured. The guidelines for adding visual information to programs are relatively straightforward. Basically, every flow-of-con trol structure must be introduced consistently. The code executed with the flow of control must be indented to show the scope of the flow-of-control construct. This indentation must be large enough to separate the code visually while not being so large so that the eye "skips" over the indented code as though it were divorced from the flow of control. In general, indents of less than three spaces is too little and more than 5 spaces is too much. However, the human brain is enormously flexible and as long as the rules for encoding flow-of-control information are consistently followed within a program, practically anything will work.

Encoding is therefore largely a matter of personal style. In the open source community, there are at least a dozen different popular structured programming styles. In my experience, Digital Equipment Corporation and the code generated in support of its many products and product lines was unique in that the same structured programming style was used for many of them. Today much of the programming consistency is supported by languagesensitive editors such as EMACS and LSEDIT. Virtually every integrated development environment (IDE) also enforces one or more programming styles by being sensitive to the indentation in use at the time in the code being

© Copyright 2004 Hewlett-Packard Development Company, L.P.

written. All modern programming editors can be customized to support virtually any programming style.

Again, the assembly language programmers have largely missed out on the advantages of modern editors and IDEs bcause assembly language itself doesn't provide any direct support for structured programming.

So what are the benefits of structured programming in assembly language and how can it be supported?

Benefits of Structured Programming

One common error in assembly language is a property of the flexibility of assembly language. Assembly language programming is inherently untyped. The assembly language programmer may treat any given piece of data as any type, byte, word, longword, ASCII, EBCDIC, null terminated, counted string, etc. By itself, structured programming doesn't deal with this source of errors. OpenVMS (and before it, RSX-11M) used naming conventions to denote data types, byte, word, longword, text, etc. The strict use of naming conventions provides an easy visual mechanism for the programmer to make sure that the type of comparison matches the data type being compared.

Another common error is also a function of data typing. In particular, comparisons against all the common data types can usually be done in either signed or unsigned modes. Since the difference between a signed branch versus an unsigned branch is frequently a single character (for example, in Macro-32, BGTR versus BGTRU for branch on greater than versus branch on greater than unsigned), it is easy to forget that a data type is unsigned, or to simply make a typing error and leave off the "U". Again, by itself, structured programming doesn't help with these errors but naming conventions help facilitate the discovery and correction of such errors.

Another common error is not getting the sense of the comparison correct. I program in assembly language on a variety of machines. Some machines test their operands from left to right (A < B) when usin g comparison instructions. Others test their operands from right to left (B < A). Switching from machine to machine can lead to programming errors simply from forgetting details of the machine architecture. Again, structured programming, by itself, doesn't help here either.

The place where structured programming **does** help is understanding the flow of control through a maze of assembly language instructions. Properly designed, the tools to do structured programming in assembly language will help with the other problems as well.

Needless to say, I'm not the first to think of this. During the development of the Record Management System (RMS) on the PDP-11, Ed Marison, et al., developed a package of macros that addressed virtually all of the defects of assembly language for the PDP-11. Unfortunately, this package of macros (known as Super Mac) took forever to assemble, but the increased programmer productivity and higher quality in terms of number of bugs was felt, correctly, to more than offset the amount of time it took to assemble any given portion of RMS.

I developed and have used a similar macro pachage for 20 years now on a wide variety of embedded systems (PDP-11, Z8000, Motorola 68K) and any number of driver development projects (mostly on OpenVMS). This macro package focused mostly on what I feel are the largest problems associated with assembly language programming, specifically, exposing the structure (flow of control) of a program written in assembly language.

Introducing Simple Mac

The simple structured macro package (Simple Mac) has virtually eliminated my most common errors in assembly programming and has substantially improved my ability to revisit and understand programs that I've written years ago. Since I'm a pragmatic programmer (use what you need

© Copyright 2004 Hewlett-Packard Development Company, L.P.

when you need it), Simple Mac also makes it easy to spot where I **don't** use proper structured programming by making it possible to use labels only where unexpected branches occur rather than everywhere a branch destination is required.

Listing 1 shows a basic structured program using Simple Mac.

```
    .LIBRARY /SMPMAC.MLB/

    SM32INIT


    ONE:       .BLKL
    TWO:       .BLKL

    START.MODULE
    SMPMAR_EXAMPLES:

    IF #1 SET.IN R0
    THEN
        MOVAB ONE,TWO
    ELSE
        MOVAB TWO,ONE
    END IF

10$:

    BEGIN BLOCK_TEST
       IF RESULT IS VC LEAVE BLOCK_TEST
       MOVAB ONE,TWO
     END BLOCK_TEST

    IFW R0 EQLU #0 GOTO 10$

    IFL <ADDL ONE,TWO> IS PLUS THEN <MCOML TWO,TWO>

    REPEAT
       MOVL ONE,TWO
       IF TWO GEQL ONE AND TWO NEQU #-1 NEXT
        MOVL TWO,ONE
    END

    DECRU ONE FROM #43 TO #-44 BY #13
        MCOML TWO,TWO
        NEXT
        MCOML TWO,ONE
    END

    DECRU ONE FROM #43 TO #-44 BY R0
       MCOML TWO,TWO
       NEXT
       MCOML TWO,ONE
    END

    DECR ONE FROM #43 TO #-44 BY #13
       MCOML TWO,TWO
       NEXT
       MCOML TWO,ONE
    END

    DECR ONE FROM #43 TO #-44 BY R0
       MCOML TWO,TWO
       NEXT
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
   MCOML TWO,ONE
 END

DECRU ONE FROM ONE TO #-44 BY #1
   MCOML TWO,TWO
   GOTOW 10$
   MCOML TWO,ONE
END

DECR ONE FROM ONE TO #-44 BY #1
   MCOML TWO,TWO
   LEAVE
   MCOML TWO,ONE
END


   REPEAT
          ON.ERROR THEN <GOTOW 10$>
   DECRS ONE TO #13 BY #25

   REPEAT
          ON.NOERROR LEAVE MCOML TWO,TWO
   DECRU ONE TO #13 BY #25

   REPEAT
          ON.ERROR THEN <GOTOW 10$>
   INCRS ONE TO #13 BY #25

   REPEAT
          ON.ERROR THEN <GOTOW 10$>
   INCRS ONE TO #13 BY R0

   REPEAT
          ON.NOERROR LEAVE
          MCOML TWO,TWO
   INCRU ONE TO #13 BY #25

   REPEAT
       ON.NOERROR LEAVE
       MCOML TWO,TWO
   INCRU ONE TO #13 BY R0
   REPEAT
       ON.ERROR THEN <GOTOW 10$>
   DECRS ONE TO #0 BY #1

   REPEAT
       ON.ERROR THEN <GOTOW 10$>
   DECRS ONE TO #0 BY R0

   REPEAT
       ON.NOERROR LEAVE
       MCOML TWO,TWO
   DECRU ONE TO #0 BY #1

   REPEAT
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
        ON.NOERROR LEAVE
        MCOML TWO,TWO
    DECRU ONE TO #0 BY R0

    REPEAT
        ON.ERROR THEN <GOTOW 10$>
    INCRS ONE TO #13 BY #1

    REPEAT
    ON.NOERROR LEAVE
        MCOML TWO,TWO
    INCRU ONE TO #13 BY #1


    SCASE R0 FROM 10 TO 30
    SET
      $CASE OUTRANGE
           MCOML ONE,ONE
      END

      $CASE 10 TO 15
           MCOML TWO,TWO
      END

      $CASE 16 TO 20,10$
      $CASE INRANGE
           MOVL R0,ONE
      END
    END

    SCASE R0 FROM 10 TO 30
    SET
$CASE 10 TO 15
           MCOML TWO,TWO
      END

      $CASE 16 TO 20,10$

      $CASE INRANGE
           MOVL R0,ONE
      END
    END

END.MODULE
.END
```

Listing 1
Simple M ac example program.

The Simple Mac example program doesn't do anything except demonstrate that using Simple Mac allows the programmer to focus on the implementation instead of worrying about how to implement the flow of control thorough the program. In the above example, the necessary code to actually implement the flow of control would substantially outnumber the actual executable code in the program (not a normal situation, but for complex programs this can appear to be the case). Additional documentation in the form of comments, discussion about the purpose of the program,

© Copyright 2004 Hewlett-Packard Development Company, L.P.

why each block exists and what each statement is doing would also appear in a real program, adding to the ease of maintanence.

I have used Simple Mac in many environments. Of course, its principal use is in the development of Macro-11 and Macro-32 programs. I've written many device drivers for OpenVMS compatible with both the VAX and AXP versions of the system using Simple Mac. I've written system services and a variety of other applications in assembly language using Simple Mac. I've also developed embedded systems using Simple Mac on processor architectures other than the 16 and 32 bit Digital/Compaq/HP machines. In these cases, the macro processing capabilities of the assembler in the development environment was not sufficient to implement Simple Mac directly. Under these circumstances I found it necessary to write a preprocessor that converted the Simple Mac statements into assembly language which were then processed by the embedded system's development environment. By leaving the Simple Mac statements embedded in the generated assembly language source files, debugging was straightforward.

Since developing Simple Mac 20+ years ago, I've written several hundred thousand lines of assembly language on several different processor architectures. Use of Simple Mac has virtually eliminated the most common of my programming errors in assembly language and substantially improved my ability to maintain the assembly language code that I've written.

Conclusion

Like all tools, Simple Mac must be used where and when appropriate. Most of the benefits of Simple Mac are simply copies of capabilities inherent in all modern high level languages. Given the choice between implementing in assembly language and any higher level language (save possibly Cobol) I will always choose a higher level language. But when, for whatever reasons, it's necessary to write code in assembly languaqge, I use Simple Mac.

In summary, use of Simple Mac, along with good naming conventions and strong programming discipline can significantly improve programmer productivity and reduce maintenance costs for projects written in assembly language.

Simple Mac syntax elements

Module	A group of assembly language source lines which begin with a START.MODULE, end with an END.MODULE, and [may] include one or more Simple Mac statements.
Module declaration	A callable unit (CALL/CALLS/CALLG).
Macro statement	Any valid assembly source statement, **except** one of the Simple Mac statements.
Block statement	Any of the block-structured statements: BEGIN, REPEAT, CASE, IF-THEN-ELSE, REPEAT, etc.
Block type	LOOP BLOCK CASE Segment name INNER OUTER REPEAT INCR DECR
Segment name	1-15 character symbolic name given to a

© Copyright 2004 Hewlett-Packard Development Company, L.P.

	program segment by a BEGIN statement.
label	Any valid MACRO address label.
condition	operand relation operand operand SET.IN/CLR.IN operand operand ON.IN/OFF.IN operand operand MASK.ON/MASK.OFF RESULT IS relation <macro-statement> IS relation
Conditional expression	condition condition AND condition condition OR condition
Asm constant expr	Any assembly time constant expression. It must be possible to evalute the expression at assembly time, not link time.
Case range expression	asm-cons-expr TO asm-cons-expr[1] asm-cons-expr
$Case range expression	case-range-expression INRANGE OUTRANGE <case-range-expression, …>
Operand	Any valid assmembly language operand.
Status	ERROR NOERROR

[1] the first asm constant expression must be less than the second

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Relation	EQ/EQL EQU/EQLU NE/NEQ NEU/NEQU GT/GTR HI/GTU/GTR U GE/GEQ HIS/GEU/GEQU LT/LSS LO/LTU/LSSU LE/LEQLOS/L EU/LEQU MINUS ZERO PLUS CC CS VC VS SET.IN /ON.IN CLR.IN/OFF.IN MASK.ON MASK.OFF	Signed Equal to Unsigned Equal to Not Equal To Not Equal To Greater than Greater than Unsigned Greater than or Equal to Greater than or Equal to Unsigned Less than Less than Unsigned Less than or equal Less than or Equal to Unsigned Sign bit set Zero bit set Sign bit clear Carry Clear Carry Set Overflow Clear Overflow Set Bit set in Bit off in Bit(s) on in the masked operand Bit(s) off in the masked operand

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Type	B (byte) W (word) L (longword) F (float, currently not implemented) Q (quadword, currently not implemented) O (octaword, currently not implemented)
Sign	S (signed) U (unsigned)

A conditional expression is true if:

- it is a single condition and that condition is true.
- it is an OR expression and either of the conditions is true.
- it is an AND expression and both conditions are true.

It is false otherwise.

The IF, UNTIL, and WHILE statements operate on the specified data types when evaluating a conditional expression. If a type is unspecified, the default type is word.

The SCASE statement operates on the specified data type when evaluating a range, otherwise word entities are used. Float values are not valid for case ranges.

The IS operation in a condition tests the settings of the current condition codes. If the first operand is the reserved word RESULT, then the current setting of those codes is tested, otherwise the first operand is assumed to be a macro statement. This macro statement is executed and the resulting condition codes are tested.

SET.IN/CLR.IN and ON.IN/OFF.IN in a conditional expression refer to a bit in the second operand, as selected by the first operand.

MASK.ON/MASK.OFF in a condition expression refer to a collection of bits in the second operand, as masked by the first operand.

A Simple Mac source file contains one or more modules. SIMPLE-MAC statements may only appear within a Simple Mac module (START.MODULE/END.MODULE).

A program block consists of one or more assembly language statements delimited by a starting statement and an END statement.

- Conditional blocks begin with IF or SCASE statements.
- Loops begin with REPEAT, INCR, or DECR statements.
- Program segments are started by BEGIN or $CASE statements.

Multiline conditional blocks and program segments must be terminated by an END statement. Loops can be terminated by an END, UNTIL, or WHILE statement. Note that THEN and ELSE statements do not terminate a block.

Single line IF-THEN, IF-LEAVE, IF-NEXT, and IF-GOTO statements do not constitute a conditional block and do not require an END statement.

Single line $CASE-range-expression, label statements do not constitute a program segment.

Simple Mac statements

BEGIN segment-name

Assigns the specific symbolic name to this program segment. The symbolic name can then be used in LEAVE statements to exit the code contained within the block. BEGIN blocks may be nested.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
BEGIN CONTROL
   CALL INIT
   CALL CSIOV1
   ON.ERROR LEAVE CONTROL
   CALL PROCES
   CALL CLEAN
END
```

SCASE[type] operand FROM case-range-expression

To avoid confusing the Macro-32 compiler, the CASE instruction has been renamed SCASE.

This stands for Simple Mac CASE. This is the only distinction in syntax between SMPMAC.MAC (Macro-11) and SMPMAC.MAR (Macro-32). Many processors do not directly implement a case instruction. The SCASE macro provides the hooks by which one may be impleme nted.

Provides an EXTREMELY fast dispatch mechanism to a variety of possible alternative processing paths. This function is expensive in memory since the speed is achieved using a dispatch table. If the value of the case operand is outside the specified range and no OUTRANGE action is specified, the case falls through to the end. If no action is specified for some set of values of the operand, the case falls through to the end (remove the $CASE INRANGE in the example and values 0 to 3 would fall through the CASE).

```
SCASE I FROM 0 TO 7
SET
  $CASE OUTRANGE,CASE.ERROR       Go here if out of range.

  $CASE 4 TO 7                    If 4 <= i <= 7 do this block
       CALL ON.HIBIT
  END

  $CASE INRANGE
       CALL OFF.HIBIT             Otherwise do this block.
  END
END
```

$CASE $case-range-expression[,label]

Identifies which value or range of values will be processed by the following block. If a label is specified, the $CASE causes a jump to that label to occur. A $CASE without a label argument defines the beginning of a block. The block may be exitted via a LEAVE CASE. See the SCASE example above for usage.

DECR[sign] loop-index FROM start TO end BY decrement

Initialize the loop index with the start value, and repeat the loop until the value of the loop index is less than the stated end value. The test for termination is either SIGNED or UNSIGNED depending on the value of the sign argument (S or U). The default is SIGNED comparison. The loop index must be of type long. This loop is a 0 trip loop, that is, if the initial value of the loop index is less than the end value, the loop is not executed. The termination test is performed at the top of the loop. If the lexical value of the loop-index and the start value are the same, no loop initialization is generated.

If the loop index is identical to the initial value, the loop assumes that the last thing done before entering the loop was to set the loop index. No special code is generated to test things. Since the 0 trip versions of these loops assume the loading of the loop index, you MUST either load the loop

© Copyright 2004 Hewlett-Packard Development Company, L.P.

index just before the DECR/INCR instruction or issue a processor appropriate comparison instruction (TSTL or CMPL for Macro-32) to get the condition codes set up properly.

ELIF

ELIF[type] conditional-expression
[THEN]
    conditional-block
[ELSE
    conditional-block]
END

ELIF is syntactic sugar equivalent to:

```
ELSE
       IF[type] conditional-expression
              …
       END
END
```

except that a single END statement to a series of IF-THEN-ELIF-THEN-ELIF statements will terminate all statements including the introducing IF. The purpose of the ELIF statement is to avoid generating excessive indentation with nested ifs. Excessive indentation can make a program very difficult to read and obscure the flow of control.

All forms of IF are also valid with ELIF, e.g., ELIF-LEAVE, ELIF-GOTO, etc. When used in this fashion, the ELIF terminates the current conditional block, and NO END is necessary since these forms of IF statement have no ELSE clauses.

```
IFW A EQ #14
THEN
       …
ELIF A EQ #19
THEN
       …
ELIF A EQ #23
THEN
       …
ELSE
       …
END                              Conditional block is EXPLICITLY
                                 terminated.
IFW A EQ #14
THEN
       …
ELIF A EQ #23 GOTO DONE          Conditional block is IMPLICITLY
                                 terminated.
```

IF

IF[type] conditional-expression
[THEN]
    conditional-block

© Copyright 2004 Hewlett-Packard Development Company, L.P.

[ELSE
        conditional-block]
END
Executes the subsequent conditional block if the specified conditional expression is true, otherwise, it executes the optional ELSE conditional block. Type can be any of the primitive types supported by the processor architecture. For Macro-32 these are Byte, Word, Longword, Quadword, and Octaword, e.g. IFB, IFL, IFW, …

```
IFB (R1)+ GT #0
THEN
       MOVL R0,-(R3)
       INCL COUNT
       CALL NEWLIN
ELSE
       CALL ERROR
END
```

IF[type] conditional-expression THEN <macro-statement>

Executes the specified macro statement if the conditional expression is true. There are no restrictions on the macro statement. It can be anything from a single instruction, to a subroutine, to a macro-invocation in its own right. The macro-statement can, in fact, be another Simple Mac statement.

```
IFW R0 LT #MAX THEN <CALL SWQR5>
IFB R5 LSSU #TABEND THEN <ADD #3,R5>
```

IF[type] conditional-expression LEAVE block-type

Transfers control to the end statement of the specified block type. LEAVE searches for the innermost block that satisfies block-type. For example, you can exit a repeat loop from within a case block or a BEGIN block from within an IF block. If a segment name is used in a LEAVE, the named segment is exited. If the block type OUTER is used, the outermost block is exited, independent of block type. If the block type INNER is used, the innermose block is exited, independent of block type. The default value for block-type is INNER.

```
REPEAT
       IF A EQ #END
       THEN
              …
              IF (R0) EQL #0 OR RO GTRU LEAVE LOOP
              …
       ELSE

              …
        END
       …
   END
```

IF[type] conditional-expression GOTO label

If the conditional expression is true, control is transferred to the specified label. This operation isn't frequently needed, but is here for those times when a good old GOTO is just what's needed.

```
IF #ERROR SET.IN CONTROL THEN GOTO ABORT
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

END [COMMENT]

Terminates the current loop, conditional, or program segment. The optional comment can be used to match end statement with block start statement, improving readability.

```
BEGIN
      …
END BEGIN
```

END.MODULE

Terminates the current module. A single file can contain more than one module.

GOTO[type] label

Transfers execution to the specified label. A GOTOB uses a branch instruction, a GOTOW uses a jump. Unlike the other instructions, the default type for GOTO is B rather than W. The implementation of this statement is rather heavily slanted towards the VAX processor architecture and may need to be modified on other architectures. In that event, I suggest that the GOTOB statement be used to indicate transfer of control to somewhere "close" visually and GOTOW to someplace "far away."

INCR[sign] loop-index FROM start TO end BY increment
END

Initialize the loop index with the start value, and repeat the loop until the value of the loop index is less than the stated end value. The test for termination is either SIGNED or UNSIGNED depending upon the value of the sign argument. The default is SIGNED comparison. The loop index must be of type word. This loop is a 0-trip loop. The termination test is performed at the top of the loop. If the lexical value of the loop-index and the start value are the same, no loop initialization is generated. This statement is biased towards the VAX implementation for loops.

See also DECR.

LEAVE block-type

Transfers control to the end of the specified block type.

See IF conditional-expression LEAVE block-type for examples.

NEXT

Transfer control to the next iteration of the loop. Control is transferred to the loop termination tests.

ON.ERROR THEN <macro-statement>
ON.ERROR GOTO label
ON.ERROR LEAVE block-type

Perform the specific action if the low bit of R0 is clear. This is an implementation of the OpenVMS-specific error-condition code covention in which error codes are returned in R0 and the low order bit is cleared. Other processors and systems have other conventions.For example, PDP-11 operating systems generally returned success and failure by clearing and setting the carry-condition flag. The Macro-11 implementation of Simple Mac reflects this difference.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

ON.NOERROR THEN <macro-statement>
ON.NOERROR GOTO label
ON.NOERROR LEAVE block-type

The opposite of the ON.ERROR statement.

REPEAT
END

REPEAT UNTIL[type] condition
END

REPEAT
UNTIL[type] condition

REPEAT WHILE[type] condition
END

REPEAT
WHILE[type] condition

REPEAT
DECR[sign] loop-index TO value BY decrement

REPEAT
INCR[sign] loop-index TO value BY increment

Perform the loop UNTIL the condition is true, WHILE the condition is true, or until the loop-index is less than or equal to the specified value (for decrement repeat loops) or is greater than or equal to the specified value (for increment repeat loops). The termination conditions are tested at the place where they appear in the loop. If at the top, they are tested before the loop begins and the loop is a 0-trip loop; if at the bottom, they are tested when the loop terminates and the loop is a 1-trip loop. The loop index must be initialized before entering the loop and must be of type word. This will vary from processor architecture to architecture.

START.MODULE

Defines the beginning of a module. Initializes the state of Simple Mac.

For more information

Dick Munroe – my resume and contact information, I'm looking for work, contract or permanent.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Developing Backup Strategies That Work

Ted Saul
OpenVMS Product Competency Center

Overview

Today's computing environment today demands that backup strategies be finely tuned and designed to meet the specific needs of each company. It is no longer simply a matter of running a backup each day with the intent of being able to restore system files should a disk failure take place. A data protection strategy must be in place to enable your company to continue to do business in the event of any of the following:

- A lost file caused by accidental deletion or data corruption
- A lost disk perhaps caused by a hardware failure
- A total system failure where a server becomes inoperable and the total environment will need to be recreated
- A catastrophic site loss caused by nature or hostile activity

Also affecting the criticality of today's backups is the ever-growing list of governmental regulations that require that certain data be available and accessible for a specific period of time. Consideration must be given to the new types of media that are available and their life expectancies. Thought needs to be put into the redundancy of backups that must be retained and where they will be stored. Safeguards should be put in place to ensure that data that needs to be restored will be available and restorable with the least downtime possible. As well, there are steps to be taken to make it possible to restore a system as quickly as possible.

How can you be sure your backup strategy works? Will you be able to recover from a catastrophic event ranging from a disk failure to a complete loss of site computing due to natural disaster or terrorist activity? Developing a disaster-tolerant backup strategy is one of the most important tasks that can be carried out in the IT environment. The design must consider the nature of your business, the size of your budget and the speed with which the system must be recovered. You can take specific steps to ensure the integrity of your backups and that all required data can quickly be restored in the event of a catastrophic outage. This article will help system managers review current strategies to ensure they meet the needs of their company and answer the many questions that arise in building such a strategy. It is also a starting point for new computing environments to help guide the design of their backup strategy.

Understanding your Environment and Defining Data Zones

The first step in a solid backup strategy is to have a good understanding of your data environment. This knowledge goes beyond simply executing a SHOW DEVICE command to see what disks are on your system; it extends into logically grouping your data for backup purposes. You can design *data zones* to group certain types of data together for backup protection. For example, static data that experiences very little change may fall into one zone while data changing frequently may belong to another. These data zones may be physically set up by disks or arrays of disks, which makes for easy viewing and understanding. As an alternative, they may be "virtual" groups established by directories with like data. This strategy can be documented on paper or by software but must be easily interpretable.

You will need to decide what data zone scheme works best for your company. A simple example of how three data zones might work follows.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- **Data Zone 1:** Consists of the data that is critical to the business. This zone is targeted for more frequent backups and placed at a high priority on the schedule. The files in this zone may change quite often and may be used for transaction processing. These crucial files may also not be easy to recreate from scratch, such as a system disk.
- **Data Zone 2:** Consists of files such as your system disk. Although they are clearly important to your system, these files may not change often. System files may also be backed up as a part of a standalone backup scheme. Backups of this zone will need to be scheduled after an upgrade of the operating system or applications as well as when ECOs have been applied. Consideration should be given to administrative changes that may be made, such as changes in Authorize, the DECNET configuration files, or other such files that may be modified as a part of system management. Note that you need to decide whether retention of files such as the Operator log and error log normally located in system areas are considered critical to your business.
- **Data Zone 3:** This group of files may include those that can be easily recreated in case of loss, files that are not required to recover an application, and user temporary files that have no effect on the running system. Some of these may be candidates for backups managed by users themselves, assuming the hardware resources are available.

Identifying data zones will also help set up the schedule for backups. It will be evident what files need to be backed up first each night to ensure their completion. A non-critical zone may be skipped should there be a hardware outage or an unplanned error that reduces the backup window.

Data Zone Backup and Restore Policies

Once you have organized the data into workable groupings, you can develop your backup and restore polices. Assessing your environment and answering a number of questions can accomplish this.

How much data is there to be backed up in each zone?

To identify required backup resources, you will want to calculate how much data you have in each of the defined zones. To see individual file sizes use the command:

$ dir/size

```
Directory SYS$SYSROOT:[SYSMGR]

ACCOUNTNG.DAT;6         1915
B.DAT;1                    1
COPY.DAT;1                31
CRASH.DAT;1               60
FILE_TO_EMAIL.DAT;2        2
IOGEN$PREFIX.DAT;1         9
K.DAT;2                   41
KERRY.DAT;1                1
MIKE.DAT;1                 4
MONITOR.DAT;3              5
.
.
.
.
Total of 15 files, 275 blocks.

Grand total of 2 directories, 43 files, 5414 blocks.
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

To capture the entire directory and a total use the following command:

$dir/size/total

```
Directory SYS$SYSROOT:[SYSMGR]
Total of 784 files, 1846347 blocks.
Directory SYS$COMMON:[SYSMGR]

Total of 303 files, 54898 blocks.
```

Divide your number of blocks by 2000 to obtain an approximate number of megabytes. With OpenVMS Alpha Version 7.3-2, you can use the following command to obtain the number of kilobytes for each file and the total for all subdirectories:

$ dir/size=units=bytes [.*] *.dat

```
Directory SYS$COMMON:[SYSMGR]

ACMEXCPAR_GENVAR.DAT;1   4KB
AMDS$DRIVER_ACCESS.DAT;1
                         9KB
APACHE$CONFIG_DEFAULT.DAT;1
                         4KB
DECW$AUTOCONFIG.DAT;1   22KB
DECW$FS_CONFIG.DAT;1     4KB
DECW$RGB.DAT;1          18KB
VMS$AUDIT_SERVER.DAT;1
                         9KB
VMS$IMAGES_MASTER.DAT;1 22KB
VMSIMAGES.DAT;1          4KB

Total of 9 files, 99KB
Grand total of 2 directories, 15 files, 3.33MB
```

$ dir/width=file=30/size=units=bytes *.dat

```
Directory SYS$SYSROOT:[SYSMGR]

ACCOUNTNG.DAT;1                  3.94MB
APACHE$CONFIG.DAT;5                 4KB
IOGEN$PREFIX.DAT;1                  4KB
VMSIMAGES.DAT;1                    18KB

Total of 4 files, 3.97MB

Directory SYS$COMMON:[SYSMGR]

ACMEXCPAR_GENVAR.DAT;1              4KB
AMDS$DRIVER_ACCESS.DAT;1            9KB
APACHE$CONFIG_DEFAULT.DAT;1         4KB
DECW$AUTOCONFIG.DAT;1              22KB
DECW$FS_CONFIG.DAT;1                4KB
DECW$RGB.DAT;1                     18KB
VMS$AUDIT_SERVER.DAT;1              9KB
VMS$IMAGES_MASTER.DAT;1            22KB
VMSIMAGES.DAT;1                     4KB
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
Total of 9 files, 99KB
Grand total of 2 directories, 13 files, 4.06MB
```

With this information and the expected throughput of your drive and controller, you can get an idea of how large a backup window will be needed to complete backup of each data zone.

Contingency plans can be put into place should a problem arise during one phase of the backup. This might include canceling the backups of a low priority zone or shifting to another set of tape drives. Tracking how much data is to be backed up will ensure that you have enough tape volumes available for your backups. This will help prevent the surprise of suddenly needing a new tape volume and finding your backup waiting on a tape when you arrive in the morning. It will also help those responsible to make sure enough tapes are loaded into a tape library before the backup run begins. As an added benefit, sudden jumps in the amount of data being backed up may uncover wasted disk space caused by an event such as the undetected growth in an application log or transaction file.

How should each data zone be protected?

OpenVMS is known for its non-stop processing and continuous uptime. This can present some challenges when it comes to making sure all these data are backed up. Tape backup is only one option of many available to companies today when it comes data protection. Other options to setting up a disaster-tolerant environment include:

- Hierarchical Storage Management (HSM) Software to allow for the shelving and un-shelving of files to and from near-line storage. The near-line storage may be backed-up as needed, preventing the need to back up the online data.
- Volume Shadowing for OpenVMS that provides an implementation of RAID technology to ensure that data remains accessible in the event of media deterioration or of controller, device, or interconnect failure. Shadowing plays a vital role in disaster-tolerant cluster configurations by allowing for duplicated data and configurations over long distances. Members of shadow sets may be taken off-line to be backed up while the others remain online and available for processing.
- RMS Journaling to allow for the ability to provide a transaction oriented file-level journaling operations for RMS files. When used in conjunction with redundant sites, a disaster tolerant environment can be created. The standby site can then be used as the backup source while leaving the on-line site running at optimal levels.
- Application-based transaction protection provided with products such as RTR and ACMS.
- Hot Backups provided by the vendors of databases. These backups write data while the database is in an open mode, allowing processing to continue.

The key to a successful backup policy is to understand how quickly a file may need to be restored and ensuring that the data is reachable in that amount of time. This means that if a disk goes bad and a hospital patient file is suddenly corrupted, most likely that file will be needed almost immediately. In this case, volume shadowing may have been a good strategy to implement so that a secondary member can be brought into place quickly. However, if shadowing is unavailable, a quick find of the last backup of the patient file will need to be located to start the restore. A user file that contains information that is needed on a project next week, however, perhaps can wait until the tape is brought back from off-site storage to have the file restored by an operator at a scheduled time.

At some sites, it is nearly impossible to shut down processing for any amount of time to perform a standalone or OpenVMS backup. If this is the case, one of the above solutions may be best for you. However, a great number of OpenVMS sites can carve out a backup window and are able to

© Copyright 2004 Hewlett-Packard Development Company, L.P.

send their data directly to tape. These users should move on to define the schedule for backing up their data zones.

How often should each data zone be backed up?

Depending on the number of transactions taking place in your application, this calculation may vary. Some businesses may be taking in thousands of records per hour and the effect of an outage and perhaps three to four hours of lost processing might be devastating. This decision will be based on the trade-off between the restore time and any downtime that may be required to execute a backup. For example, if your processing takes place between 8:00 and 5:00 each day, you may lose more data as the day goes on. Will it cost you more to delay processing in the middle of the day to perform some type of backup or will there be more loss by requiring a restore of data and re-entry of data from more hours lost the day of the outage and restore time?

Also affecting this trade-off will be:

- The speed of the equipment used for backups and restores. New efficient tape devices allow for the random selection of files and directories from tapes that may be already loaded in an automated tape library. When used in conjunction with a backup application solution, the files can be easy to locate and the process can be done almost automatically and unattended.
- The location of the volumes used to store the backed-up data. If the latest tapes have been sent to an offsite location or service, the length of time it will take to retrieve these tapes must be calculated. This should be factored into the cost of downtime to recover a system.

A common schedule template is to do a Weekly Image with Daily Incremental backups. In its simplest form, it allows for a complete backup of the system one day a week followed by a backup of files that have been changed each day. To conserve tape space, the incremental backups can be stored on a defined set of tapes that can be reused. The image backup volume set may be sent off-site for protection.

What devices do you have to back up each data zone?

When it comes to tape devices, there are two issues to consider when looking at where to store your data:

- The speed of the device.
- The reliability of the device.

You may want to send your most critical data to your fastest device but consider its trustworthiness. If the device consistently shows numerous errors caused by verification of reads and writes, not only will performance suffer but also the level of data integrity may be questionable. The availability and age of the associated media needs to be considered as well. If your tape device has been consistently using its Compact III tapes for any extended period of time, data written to the tapes may be compromised.

When should the backup of each data zone take place?

By gathering the information of how much needs to be backed up (Question 1), how often it needs to be done (Question 3) and what devices are available for the backup (Question 4), you can start to define your backup window. If you find this window overlapping with production time, you will have to take steps to reduce its size. This can be accomplished by using techniques such as the following:

- Using an Image / Incremental pattern of backups. The Image may be written on the weekend when system availability is greater.
- Off-loading low priority data zones to weekends.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- Reducing the iterations of backups taken against the system disk. This may require the "lock-down" of the system disk and scheduling times of system management work to correspond with the next backup.
- Increasing the number tape devices available to handle the backup.
- Replacing slower tape devices with newer, faster devices and interconnects.
- If backups are dependent on an operator schedule, implement an unattended backup scheme using automated tape libraries and software to manage the backups.
- Use OpenVMS Backup to a disk and copy the savesets in a separate operation.
- Implement Volume Shadowing to allow for the temporary breaking of shadow sets.
- Obtain redundant hardware for appropriate disaster tolerance.

What is best for your site will depend on your budget, backup resources available and the ability of your business to absorb computer outages.

How long do backups need to be retained?

Decide how long your data must be maintained. This length will depend on the industry such as Healthcare, Securities Trading and Department of Defense. Be sure to investigate the legal requirements of keeping your data. Data zones may also be formed by retention lengths required on groups of files.

Keep in mind the type of media that you are using for backup. Choose a medium that will have a life expectancy consistent with the data you are saving. Keeping multiple copies of time-critical data may also be something to consider.

What Will Be Your Data Restore Policy?

First, who will be responsible for the restore? Be sure that person understands the directory structure and the privileges required to write to those directories. This person must be able to find documented processes for restores quickly and be able to execute them confidently.

Before any restore takes place, you should determine why the failure took place so as to prevent the incident from reoccurring. If the failure was caused by a hardware failure, be sure that alternative hardware is in place. If an application or software has failed, be sure the problem has been fixed or an appropriate workaround is in place. A user that has inadvertently deleted a file should be coached to ensure that the behavior is not repeated. Data restoration can prove to be very expensive making it imperative that it is not done unnecessarily.

Timing of restores is important as well. Those performing the restores need to be aware of conflicts that might arise, including overlap with scheduled backups, or attempts to restore at a time of heavy system use. They must also be able to find a tape in a vault. A review of all types of file restores should take place regularly to ensure that operations personnel are familiar with the procedures. One does not want to generate another interruption of the system by attempting to restore at the wrong moment, for example.

Your restore policy needs to define how to find files in your collections of backup tapes. A backup application can be quite helpful in this role as it can provide catalogs that record which tape volume a file or disk is located on as well as the current location of the tape. Depending on the complexity of your backup environment, a backup listing may be used to provide similar data. It is important to have some type of organization to your multitude of tapes in order to locate the needed object quickly.

A last area to consider is whether or not there are predictable instances where files or directories will need to be restored. For example, testing data gathered through the month may need to be restored at one point in order to process and generate summary reports. Such restores can be scheduled as a part of IT processing and, if necessary, plans can be made to bring appropriate volume sets back onsite.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

How will backup and restore failures be addressed?

Develop a plan on how to troubleshoot backup and restore failures. An operator should know exactly where to look first, whether it is looking for error lights on a tape drive or for the existence of log files. A backup should not simply be restarted unless some explanation for the problem can be presented. Otherwise valuable time may be wasted as well as allocation of more volumes than necessary. Operators should also be aware of how to handle unusual Opcom errors that are generated. Replying incorrectly may cause valid data to be overwritten or lost. Phone numbers and pagers should be readily available for operators to use in order to escalate their problem. Support organization numbers should be available along with a list of the pertinent log files that may be required.

Vaulting

Vaulting is the process of moving retained data to a location away from your systems. In case of catastrophic outage such as fire, earthquake or other, these volumes can be brought back in to rebuild a system. Often vaulting is handled by a third-party service that provides the pickup and delivery of tapes on a pre-determined schedule.

For vaulting to work successfully, a schedule of when tapes are to move off site and back on site must be developed. Begin by thinking of a worst-case scenario where your environment is severely compromised shortly after a backup has been done. Where do you want those most recent backup tapes to be located? If they are located on site with your system, they may be lost as well. However, if they have been shipped offsite, they will be safe.

Consider the following when designing a Vault scheme:

Is there an offsite facility available to be used as the vault?

The vault needs to be a secure location away from the main computing site. It is also good to put the tapes into a fireproof vault to guard against fire, explosion and water damage. Be sure that where you send your tapes will be secure not only from harm but also from theft and unauthorized access. The data on critical zone tapes must be considered crucial to the existence of the company.

Is there an offsite service to be used?

Reputable vaulting companies will pickup your tapes on a scheduled day while returning expired tapes as well. They are experts in ensuring that stored data is safe from the natural elements of the regions and can provide facilities against known threats in the area. The environment of their vaults should also encourage the lifetime of the media by providing appropriate cooling and humidity levels.

How fast can volumes be retrieved?

It is equally important to know how quickly your volumes can be retrieved from their offsite location whether they are managed internally or protected by a third-party service. The success of your restore policy will depend on having the needed tape volumes in hand when its time to start the return of the data.

Backup Verification and Testing

The moment of an emergency restore is not the time to find out if your backups have worked successfully. It is better to perform disaster recovery drills regularly. This will require some free space to be done without affecting current production machines. A test or backup machine makes a good candidate for this type of work. A simple test of a backup is to mount a tape in a drive an

© Copyright 2004 Hewlett-Packard Development Company, L.P.

issue a BACKUP/LIST against the tape. Files should be displayed. The command will need to be reissued for each saveset on the tape.

Practice runs on getting tape volumes back from offsite services should also be executed at appropriate intervals. These may be when your staff has seen a significant turnaround or there have been changes at your offsite service.

Special Considerations

There are a few areas that should not be forgotten. Not all these areas apply to all business sites, but they are all too easily overlooked.

Databases – Do you need to use a vendor's backup product to back up their database? This may allow you to execute hot backups in order to get saved data while the processing continues and the database is still open. However, the format of the data on tape may not be the same as written by OpenVMS Backup. It may require that the application is running during a restore.

One strategy for database backups includes using a disk-to-disk-tape process (D2D2T). In this case:

- The database is shut down.
- The Vendor Software is used to write their saveset to disk using their own format.
- The database is restarted and processing continues.
- OpenVMS backup is used to write the database savesets to tape.

A disk-to-disk backup will go more quickly than a tape backup, thus allowing for a faster return of the database to an available state.

Standalone Backups – Booting a system into standalone backup and running an image backup is the only method to ensure that a system disk is backed up safely. This procedure should take place regularly while taking into consideration changes that have been made to system files. These changes may include updates to user accounts, DECNET files and TCP/IP host files.

Note: The use of /ignore=interlock during an online backup does not ensure that all files will be available from a restore. Standalone backup is the only method by which OpenVMS engineering ensures a bootable disk can be created from a backup.

Remote nodes – Nodes that may be located outside of the cluster or normal processing area may have an impact on backup schedules should they be using cluster backup resources such as libraries and tape drives. This type of backup needs to be looked at to identify how much it may require the use of the cluster resources and be scheduled in such a way that it does not interfere with other backup streams. Products that allow tape drives to look local to a network node can introduce challenges of their own and should not simply be assumed to work in conjunction with local backups without conflict.

Backup Consolidation – Products such as Hewlett Packard's SaveSet Manager (SSM) have the ability to combine multiple backups into one. This product can take a series of incremental backups and merge them with an image restore essentially taking a week's worth of tapes (5 or 6) into one. This strategy may be used to reduce the number of tapes sent to long-term off-site storage by combining backups and reducing the costs of tapes necessary to provide site data protection.

Backup Applications

Backup applications such as the Archive Backup System and Storage Library System can prove helpful in managing your backups. Common features found in these applications include:

- Ability to set up backup policies to assist with load balancing.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- Scheduling of backups to execute at required times.
- Cataloging of data on backup volumes for easy location.
- Interaction with robotic libraries for unattended operations.
- Easy restore operations including scheduling during off-hours.
- Notifications of backup failures.
- Vaulting services.
- Writing of multiple savesets on single tapes.

Many of these features address issues that have been mentioned previously. Though the initial configuration of this type of application can appear to be overwhelming, once set into place the operations are typically hands-off. Documenting how to handle problems within the application and where to look for logging information becomes more of a priority. It is also important to have posted where support will come from in case of problems.

Conclusion

There is obviously no generic "how-to" manual for setting up OpenVMS backups. There are however many basic issues to watch out for. Taking the time to establish appropriate backup strategies that meet corporate standards can prevent serious loss of time and money should the worst scenario come about. Backups are the insurance policy for computing departments. A backup strategy should be developed carefully and reviewed to ensure that it provides adequate coverage at all times.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Best of Ask the Wizard

Steve Hoffman "Hoff"
OpenVMS Consulting Engineer

Upward Compatibility and OpenVMS Releases

Full upward binary compatibility of all user-mode applications is a central goal of HP OpenVMS Engineering, and a key OpenVMS customer expectation.

On OpenVMS Alpha systems, OpenVMS Engineering assumes, works for, tests for, and targets upward-compatibility of user-mode applications; applications that are specifically using only documented interfaces, that are lacking latent bugs, that are not using position-dependent coding constructs, and that are not specifically dependent on the OpenVMS version number itself or on a version-dependent product.

Like OpenVMS Alpha, OpenVMS VAX releases are expected to provide upward-compatibility of all valid user-mode code. On OpenVMS VAX V6.0 and later systems, OpenVMS Engineering further and additionally assumes that privileged-mode software application code such as device drivers -- again code that is using only documented interfaces, lacking latent bugs, and not otherwise version-dependent nor position-dependent -- will also be upward compatible.

To validate upward-compatibility, OpenVMS Engineering uses a variety of mechanisms. Examples of these mechanisms include source code reviews, statistical quality controls, an extensive regression test suite, tools which verify the values of constants and symbols, and programs for distributing early copies of OpenVMS releases via software developer kits (SDKs) and targeted field test efforts. All of these mechanisms strive for the early identification of compatibility problems, and for the rapid remediation of regressions.

User-mode or privileged-mode application code containing latent bugs, position- or version-dependent code, or that are using undocumented interfaces may well require rework or rebuild as ECO kits are applied or as OpenVMS is upgraded. Application code that fails to consistently check for return status values, that has implicit (and undocumented) assumptions of completion, synchronization, or any of a host of other latent bugs can and occasionally does fail after OpenVMS upgrades. An introduction to some of the more common latent coding errors is

© Copyright 2004 Hewlett-Packard Development Company, L.P.

included in topic (1661) of the HP OpenVMS Ask The Wizard site at the following URL: http://www.hp.com/go/openvms/wizard/

Privileged-mode code may require rework across major OpenVMS Alpha releases.

That said, HP OpenVMS Engineering has broken upward compatibility in a few product areas and on a few occasions, the removal of Display Postscript from DECwindows being a salient example. (While not specifically a change in OpenVMS itself, this change has tripped at least one layered product.) Latent problems within an Alpha code generator also triggered compatibility problems with the Alpha Architecture and with associated OpenVMS upgrades; OpenVMS and Alpha implementation upgrades caused the generated (application) code to fail on Alpha 21264 (EV6) and later. (Qv: the SRM_CHECK tool and information available at http://h71000.www7.hp.com/freeware/freeware40/21264/.) There are incompatibilities which can be caused by (deliberately) tightened handling of the file delete permissions starting at OpenVMS V6.0; security enhancements which broke a few applications and which also correctly prevented errant file deletion operations from occurring. But all this written, an application image that was built for the VAX/VMS V1.0 release is and will remain a part of the OpenVMS VAX regression test suite; upward compatibility is a central goal of HP OpenVMS Engineering.

The following lists show the types, contents, and intended customer audiences for various OpenVMS releases:

Note: full upward-compatibility of user-mode code -- that code using documented interfaces and lacking latent bugs -- is expected across OpenVMS upgrades.

Major release ("dot-zero" release)

- OpenVMS Alpha V7.0, OpenVMS VAX V6.0
- Key designator: the "dot zero"
- Contains new features, new capabilities
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration after the upgrade
- Often includes changes to kernel-mode data structures.
- Can require changes (recompile or reassembly, possibly recoding) of kernel-mode (privileged-mode) code

© Copyright 2004 Hewlett-Packard Development Company, L.P.

- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

Minor release ("dot" release)

- OpenVMS Alpha V7.1, OpenVMS VAX V6.2
- Key designator: the "dot not-zero"
- Contains new features, new capabilities
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration
- Few (compatible), or no, changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

Maintenance release ("dash" release)

- OpenVMS VAX V5.5-2, OpenVMS Alpha V7.1-2
- Key designator: the "dash"
- Not a vehicle for new features or new capabilities. These releases are intended for new hardware, as a roll-up of one or more maintenance fixes, to establish a common code base, or a combination of these.
- Can include new hardware support
- User-mode code using documented interfaces is expected to operate without alteration
- Few (and compatible) or no changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- The full battery of OpenVMS tests are run on this release
- The release is a general release, and is targeted for use on all supported processors
- Customers with software services contracts receive this release automatically

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Limited Hardware Release ("LHR" or "hardware" or "H" release)

- OpenVMS Alpha V6.2-1H3, OpenVMS Alpha V7.1-1H2
- Key designator: the "H"
- Not a release vehicle for new features, new capabilities, nor bug fixes -- the hardware release is intended solely for new hardware and new configuration support
- Includes new hardware support
- Includes only those ECO kits that are directly affected by the work for new hardware support -- only those ECO kits and changes specifically required to prevent regressions are included in this release
- User-mode code using documented interfaces is expected to operate without alteration
- Few (compatible) or no changes to kernel-mode data structures
- Kernel-mode (privileged-mode) code using documented interfaces is expected to continue to operate without alteration
- A limited battery of OpenVMS tests are run on this release
- The release is not a general release, and is targeted for use only on configurations including specific new hardware
- Customers must explicitly order this release

Special-Purpose Releases

- V7.2-6C2
- Key designator: other letters
- OpenVMS releases intended solely for specific customers, specific applications, or for specific and targeted purposes
- Can be a predecessor, variant, or descendant. No particular relationship to other OpenVMS releases can be assumed
- Compatibility may or may not be provided during any subsequent OpenVMS upgrades; some OpenVMS upgrades may require full application rebuilds for applications built against these specialized releases
- This release is not a general OpenVMS release, and is typically not shipped to contract support customers

© Copyright 2004 Hewlett-Packard Development Company, L.P.

OpenVMS System and Password Security

Various recent posting in the comp.os.vms newsgroup have recalled the attached piece of ancient OpenVMS programming trivia, once published in Phrack. (The attachment is from Phrack, Volume Three, circa June 1, 1989, and potentially also published elsewhere.) This trivia was posted as part of a discussion of the relative sensitivity of lists of usernames on OpenVMS systems.

In this newsgroup context, a Microsoft Windows-based password cracking tool, "John The Ripper", has been referenced. Specifically, the availability of new code within this tool intended to target OpenVMS password security. Depending on the intent of the particular tool user, "John The Ripper" is either intended for the cracking OpenVMS passwords, or for the testing of OpenVMS system password security. This and all similar tools require access to the contents of the OpenVMS password database (SYSUAF.DAT) as the input into the password attack scheme. Password attacks such as those used by "John The Ripper" are not particularly new (though clearly these techniques can and do improve, and the processor cycles that are available to crackers for these brute-force attacks clearly also increase over time), as there are and have been various password-cracking tools targeting OpenVMS over the years -- dictionary attacks, brute-force attacks, and research into the passwords likely used by the target user akin to that of the movie Wargames. ("Joshua" and "CPE1704TKS", for those wishing to collect the relevant movie trivia.) Native OpenVMS-based implementations of password-cracking tools are readily available.

As for this particular "John The Ripper" attack, the OpenVMS break-in evasion mechanisms effectively eliminate direct risks from these tools. Evasion prevents a user from repeatedly guessing passwords, locking out access temporarily or even permanently. Unfortunately, some application programs will also prompt for passwords, and provide either cleartext password storage or will not perform break-in evasion. (See sys$acm on V7.3-1 and later for a relevant and useful programming interface.)

Assuming that the contents of SYSUAF.DAT are not exposed to (untrusted) users on the running OpenVMS system, and evasion is not disabled, direct password attacks are effectively negated. Potential password exposure can be via unsecured or unencrypted BACKUP media libraries, or via unprotected and unencrypted network transfers, or untrusted privileged users. (These considerations are covered in the OpenVMS security manual.)

© Copyright 2004 Hewlett-Packard Development Company, L.P.

OpenVMS itself does not store cleartext passwords, only hashed passwords. There is no known way to reverse the password hashing algorithm; a Purdy polynomial. Various tools will perform what are referred to as dictionary attacks; trying combinations of cleartext passwords looking for a matching resulting hash value. Various other tools will search for occurrences where users and or applications have stored cleartext passwords within disk files.

The Purdy polynomial provides for large changes in the output quadword hashed value for small changes in the input. More importantly, the polynomial is also particularly difficult to reverse; to calculate the input values based on the output hash. Further, to prevent tools such as "John The Ripper" from simply building a library of pre-calculated hashed passwords, OpenVMS implements a value known as a salt. This value is unique to each username, and is incorporated into the input into the hash. And OpenVMS implements both a list of previously-used password hash values for each user, and a site-extensible dictionary of prohibited words. These two mechanisms prevent the reuse of passwords, and help the users avoid selecting weak passwords. Site-extensible password filtering is also available, allowing the security manager to further tailor the password processing.

Details of the Purdy polynomial and particularly of the OpenVMS password-hashing algorithm are deliberately available; the strength of the scheme is derived from the mathematics of the algorithm, and not from obscuring the particular implementation. The user interface is the sys$hash_password service, and C and Macro32 implementations of the password hash are widely available.

As for discussions of the exposure of lists of usernames on an OpenVMS system, no major formal security evaluation standards -- the NCSC "Rainbow" series security evaluation standards, the European ITSEC standards, or otherwise -- indicate that or believe that the exposure of lists of usernames is a security problem. Various of these standards do specifically mention such things as exposing classified project names or other sensitive data as usernames. But there is an expectation that usernames can and will be exposed. For instance, the surnames of workers can be available from various sources such as telephone lists or vehicle registrations, and surnames are often used as usernames.

On OpenVMS, usernames are not considered and not maintained as security-relevant objects, and there are various ways to acquire lists of usernames on OpenVMS, including (but not limited to) MAIL messages, PHONE commands, the finger command, SHOW USER and SHOW SYSTEM commands, and various other DCL commands, network applications, and programming interfaces.

© Copyright 2004 Hewlett-Packard Development Company, L.P.

OpenVMS as well as all other general-purpose operating systems provide what are known in the security world as covert channels; with ways to acquire or transfer secured information through unsecured channels. The SHOW SYSTEM command is an example of a covert channel on OpenVMS, as individual users can select a process name. As this process name is easily visible, some amount of information can be communicated among users. Accordingly, one the many aspects of an operating system security evaluation is an evaluation of the bandwidth of the covert channels; of the volume of security-critical information that can be transmitted through each covert channel.

In the case of the attached program, the covert channel involves finding entries in the rights list that (often, but certainly not always) match usernames. The attached adduser.pas program uses $idtoasc to look for these matching values. (Unlike John The Ripper, the attached adduser.pas tool provides only lists of usernames (assuming, as is common system management practice, that there are identifiers that match the usernames) and does not attempt to find the password -- nor does the attached example program have any way to find the hashed password value that is stored in the SYSUAF database.)

Another unfortunately common covert channel is the telephone. One of the most common password attacks involves social engineering, not any particular knowledge of the operating system security. Attackers can and commonly do locate and contact system support staff, will falsely identify themselves, will claim that a particular existing user's password has been forgotten, and will request that the password be reset. Shared usernames are another obvious target, and this security attack is one of the reasons why all shared usernames are strongly discouraged. If the attacker's password request is honored, system security is accordingly compromised, whether or not the target username is privileged.

Another potential covert security exposure can be an application that prompts users for passwords, that operates with privileges, or that operates in privileged processor modes. Such application can potentially compromise OpenVMS system security, and these applications can and will be targeted by attackers. This because users will tend to choose the same or similar passwords on a system, meaning that the insecure storage of a password within an application can compromise the OpenVMS password mechanisms. Applications that prompt for and subsequently verify passwords against application-private or against OpenVMS password storage can themselves also be targeted simply for password verification, particularly if the application does not provide for evasion. While OpenVMS break-in evasion prevents "John The Ripper" and similar tools from directly operating against OpenVMS password prompts, similar attacks can be

© Copyright 2004 Hewlett-Packard Development Company, L.P.

made against any application that prompts for passwords but that does not implement evasion. (As stated earlier, please see the OpenVMS V7.3-1 sys$acm system service.)

Information related to "John The Ripper" and pointers to the full Phrack article with the attached alluser.pas example program are available on the web.

A discussion of the relative weakness of password-based authentication is available at the OpenVMS Ask The Wizard area, with the core topic on this likely being (4612). Related topics include (1461), (1645), (4303), (4778), (5508), (6080), (6328), (7818), and others. The Ask The Wizard area is available via the following URL:

http://www.hp.com/go/openvms/wizard/

What does all this mean to most OpenVMS users and most customer sites? Please update your password dictionary to contain all words of local, regional, or site-specific significance, and please educate your users on the selection of appropriately-secure passwords. Locally-written password filters are also a potential option, and can reduce the ability of users to select weak passwords. Also train your users and particularly your support staff in related password vulnerabilities, and particularly also to avoid exposing passwords over the telephone or via cleartext email to remote (and thus what must always be assumed insecure) systems. Alternatively, consider configuring and using generated passwords, challenge-response, biometric, or other more secure identification and authorization mechanisms. Please also maintain the current OpenVMS ECOs and (as many security exposures involve a network) also maintain current network product versions and ECOs.

The following program, published in Phrack, was written by "Deep Thought of West Germany:"

```
{
* alluser.pas - get names of all users
* by Deep, 1989
* This program is freely redistributable as long no modifications are made
* DISCLAIMER: I take no responsibility for any use or abuse of this
*             program.  It is given for informational purpose only.
*
* program history:
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
* 04-May-89   started
* 02-Jun-89   clean up of code
}
[inherit ('sys$library:starlet.pen')]
program alluser(input,output);

  type $word      = [word] 0..65535;
       $byte      = [byte] 0..255;
       $quadword  = record
                      lo,hi : unsigned;
                    end;
       $uquad  = record
                      lo,hi : unsigned;
                    end;
var
  id: unsigned;
  status, status2: integer;
  length: $WORD;
  attrib,context,context2,context3: unsigned;
  ident, ident2: unsigned;
  name: varying [512] of char;
  holder: $uquad;

begin

writeln('Alluser - use at your own risk!');
status := SS$_NORMAL;
{ id = -1 selects next identifier }
id := -1;
context := 0;
while (status <> SS$_NOSUCHID) do
   begin
   { find next identifier }
   status := $idtoasc(id,name.length,name.body,ident,attrib,context);
   if (status <> SS$_NOSUCHID) then begin
      write(pad(name,' ',16));
      if (ident div (65536*32768) > 0) then
         { it's a rights-list, so print the hex-value of the identifier }
         begin
         writeln(oct(ident,12));
         context2 := 0;
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.

```
        context3 := 0;
        { find all holders of this right }
        repeat
          holder := zero;
          status2 := $find_holder(ident,holder,attrib,context2);
          if (holder.lo <> 0) then begin
              ident2 := ident;
              { get UIC and username }
              status := $idtoasc(holder.lo,name.length,name.body,ident2
                  ,attrib,context3);
              write('                ',pad(name,' ',16));
              writeln('[',oct(holder.lo div 65536,3),','
                  ,oct(holder.lo mod 65536,3),']');
             end;
        until (holder.lo = 0);
        end
      else
        { it's a UIC, so translate to [grp,user] }
        begin
        writeln('[',oct(ident div 65536,3),',',oct(ident mod 65536,3),']');
        end;
      end;
   end;
end.
```

© Copyright 2004 Hewlett-Packard Development Company, L.P.