Cluster architectures and S/390 Parallel Sysplex scalability

by G. M. King
D. M. Dias
P. S. Yu

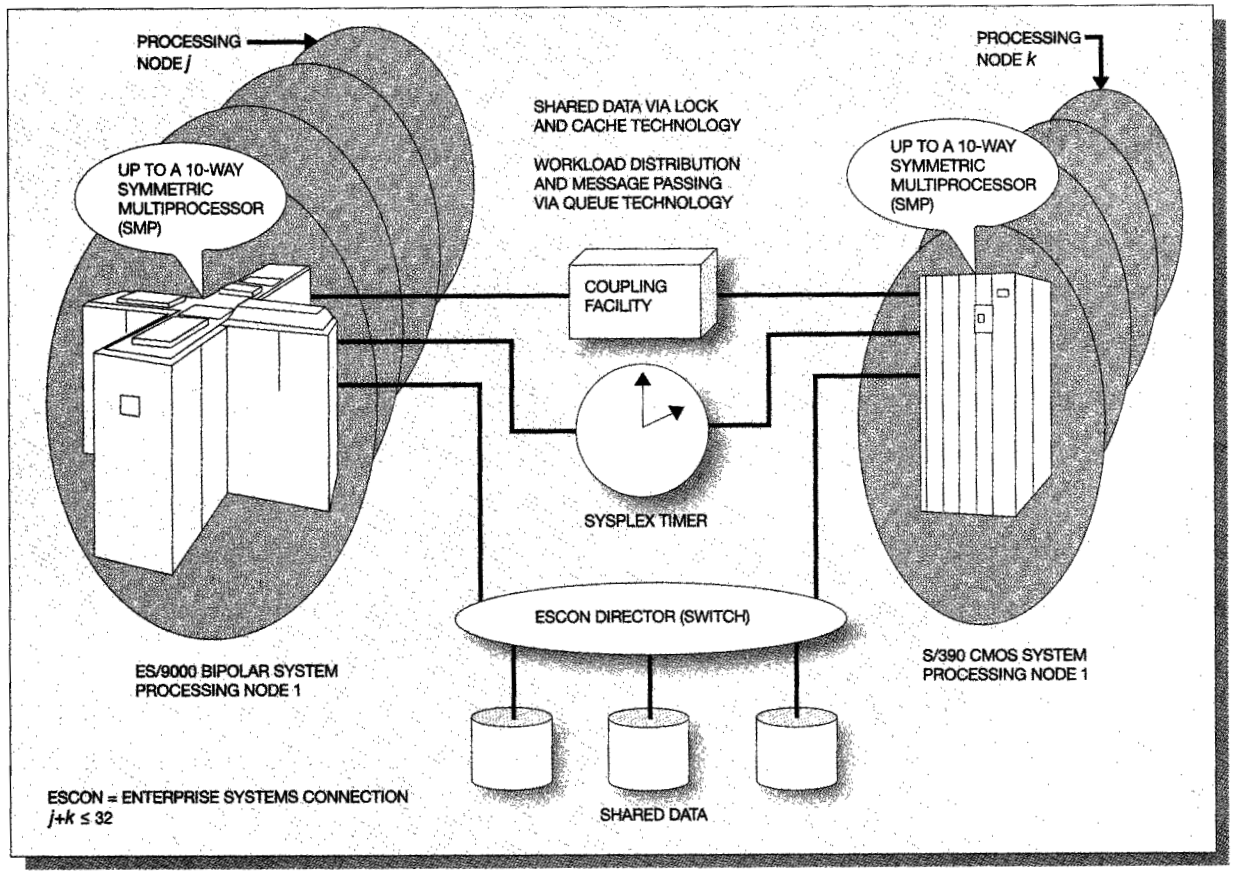
Supporting high transaction rates and high availability for on-line transaction processing and emerging applications requires systems consisting of multiple computing nodes. We outline various cluster architectures and describe the factors that motivate the S/390® Parallel Sysplex™ architecture and its resulting advantages. We quantify the scalability of the S/390 Parallel Sysplex and show that the transaction rate supported is close to linear as nodes are added to the system. The key facet of the S/390 Parallel Sysplex architecture is the coupling facility. The coupling facility provides for very efficient intertransaction concurrency control, buffer cache coherency control, and shared buffer management, among other functions, that lead to the excellent scalability achieved. It also provides for effective dynamic load balancing, high data buffer hit ratios, and load balancing after a failure.

The transaction processing rates that need to be supported have been growing beyond those that can be supported by a single computing node. With the exponential growth in the traffic on the Internet and the World Wide Web (Web), which includes allowing end users to browse and place orders on the Web, the transaction processing load is likely to grow even larger. Thus, multinode cluster architectures are needed to support these environments. Furthermore, these applications need to provide high availability, which can be supported by cluster architectures.

In this paper, we outline the various cluster architectures and their characteristics. We describe the factors that motivated the design of the IBM System/390* (S/390*) Parallel Sysplex*, and outline its key advantages as compared to other cluster architectures. We then quantify the scalability of the Parallel Sysplex design. We show that the transaction rate that can be supported is close to linear in the number of nodes in the Parallel Sysplex. We also show that, even for a Parallel Sysplex composed of heterogeneous nodes, excellent dynamic load balancing among the nodes can be achieved. We further show that the processing overhead incurred in a multinode Parallel Sysplex is small and that the amount of I/O per transaction can actually be reduced in a multinode Parallel Sysplex as compared to a single node system.

Figure 1 illustrates the basic Parallel Sysplex architecture. Further details of the Parallel Sysplex design can be found in Reference 1. Each node in the Parallel Sysplex can have a single CPU, or can consist of a symmetric multiprocessor (SMP). Each node runs a separate copy of the operating system; the processors in an SMP node run a single operating system, OS/390* (previously known as MVS) on the IBM S/390, and have a shared main memory. The nodes in the Parallel Sysplex have shared disks, which can be accessed directly from each node. Large complexes of shared disks can be configured, using the

Figure 1 Parallel Sysplex system model



Enterprise Systems Connection (ESCON*) switch.[2,3] The key facet of the Parallel Sysplex architecture is the *coupling facility*, which is shared by the nodes in the Parallel Sysplex, as illustrated in Figure 1. The coupling facility provides very efficient intertransaction concurrency control, buffer cache coherency control, a global shared buffer, and other services, which are described further in the section on the S/390 Parallel Sysplex architecture. These services of the coupling facility are the primary factors that lead to excellent scaling, which is quantified in the section on Parallel Sysplex performance. The shared buffer provided in the coupling facility also leads to high shared buffer hit ratios that can actually reduce the I/O rate per node as the number of nodes in the Parallel Sysplex increases, as quantified later. Dynamic load balancing is achieved by providing shared job queues maintained in the coupling facility, as detailed and quantified later.

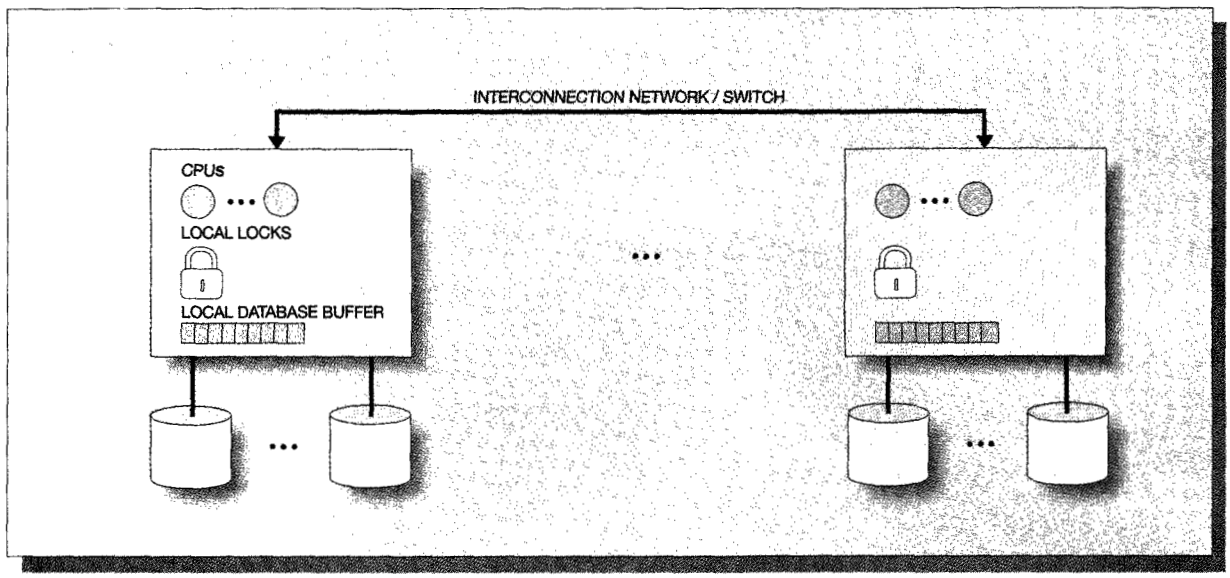
The other principal cluster architectures include the shared-disk architecture,[4–6] the shared-nothing or partitioned architecture,[7,8] and the virtual shared-disk model.[9] These architectures are outlined in the next section and are then qualitatively compared. Next, the Parallel Sysplex performance and scaling is quantified, followed by concluding remarks.

Cluster architectures

In this section we describe the principal cluster architectures for supporting scalable commercial applications such as on-line transaction processing (OLTP) and parallel database systems. These architectures can also be used to support various other emerging applications such as scalable Web[10] and video servers.

There is a major division between cluster architectures because of differences in the ability of the nodes

Figure 2 Partitioned or shared-nothing architecture



to access the disks. In one camp we have a partitioning scheme where each node has access to only a subset of the disks. In the other camp lies the disk-sharing topology where each node has access to all disks. The disk-sharing architecture additionally introduces a number of key choices that further define it. These choices include:

1. An update concurrency control method
2. A database buffer cache coherency control
3. A provision for shared memory among the nodes

The principal composition of the choices that define each cluster architecture is presented here. The performance effects and trade-offs resulting from these choices are discussed in the next section.

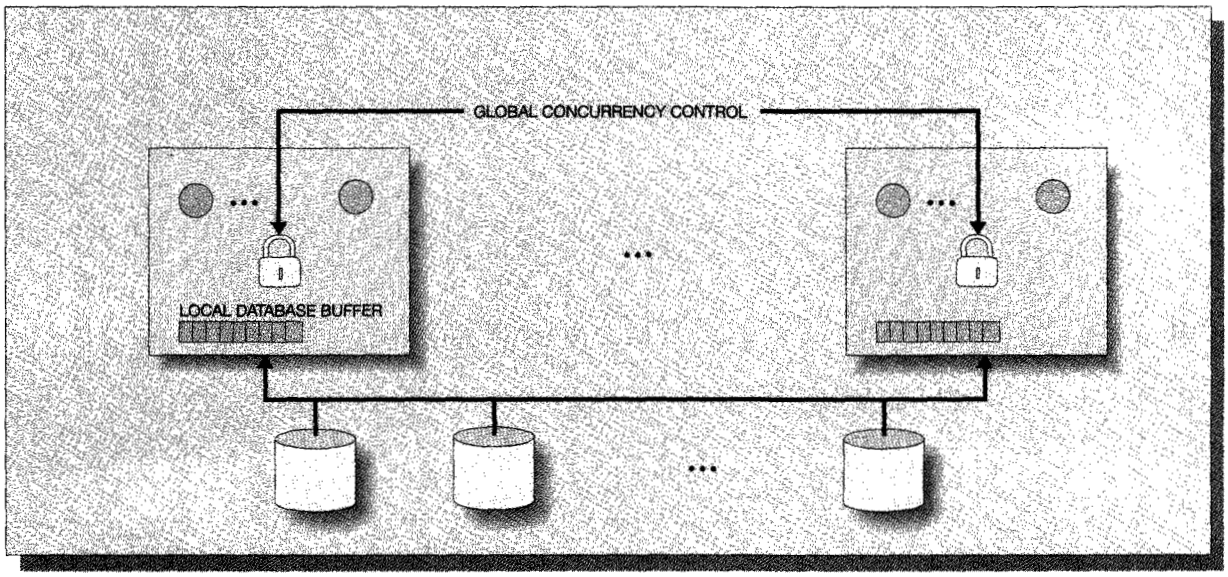
Partitioned or shared-nothing architecture. In this architecture,[7,8,11] illustrated in Figure 2, the disks and the database are partitioned among the nodes in the cluster. There are two flavors of data partitioning: In the function-shipping model, if a transaction running on a node needs to access data located at a remote node in a cluster, a remote function call is made to the node at which the data reside;[8] the remote node makes a local database call to retrieve the data, and the results are shipped back to the requesting node. In the I/O shipping model,[11] remote data are accessed by making a remote I/O request to fetch the required disk block from the remote node. I/O shipping is similar to the virtual shared-disk model described later, and shares many of its characteristics.

In the function-shipping model, the node that owns the data partition obtains local locks on the data accessed on behalf of the remote transaction. Thus, no global locking is needed. The database buffer cache is also located at the node that owns the data, and thus, there is no buffer coherency problem. A single transaction may access and update data located on several nodes. At transaction commit, a two-phase commit is needed between all the nodes in the cluster involved in updates on behalf of the transaction.

The function-shipping model is used in the Gamma Database[12] and the DB2* (DATABASE 2*) Parallel Edition (DB2PE),[13,14] among other systems. The IBM Scalable POWERparallel System*[15] running DB2PE has this architecture.

Shared-disk architecture. The shared-disk architecture is illustrated in Figure 3.[6,16,17] Essentially, all the nodes in the clustered system have direct access to (some or all) of the disks on which shared data are placed. Each of the nodes in the cluster has a local database buffer cache. In order to maintain consistency of the database with transactions running on

Figure 3 Shared-disk architecture



the different nodes in the cluster, a global (i.e., system-wide) concurrency control protocol is needed. Further, in order to maintain coherency among the local database buffer caches at the nodes in the cluster, global buffer coherency control must be enforced.

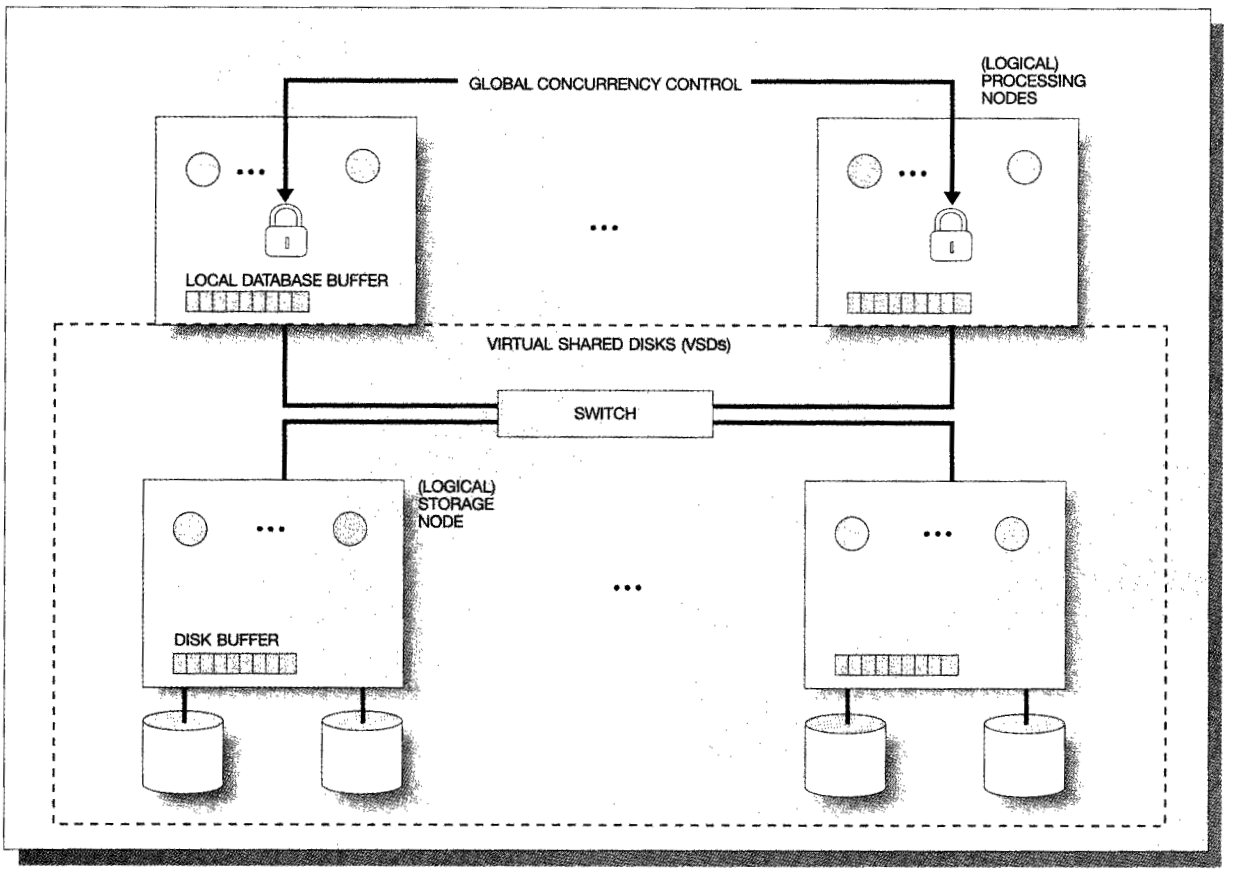
The global concurrency control could be either distributed or centralized.[18] Distributed concurrency control using a locking protocol is often used. One of the earliest distributed concurrency control protocols was implemented for Information Management System (IMS*) data sharing.[4] The protocol was referred to as "pass the buck"; the lock space was mapped to hash classes, and lock contention was detected at the hash class level by passing the "buck," with information on locks at the hash class level, among nodes. In case of contention at the hash class level, contention resolution at a finer granularity was done.

Distributed concurrency control was also used in the Digital Equipment Corporation VAXcluster**.[5] The lock space was partitioned among director nodes. Once a node obtained a lock on an object, it became the master, and subsequent lock requests to the object were referred by the director to the master node. A similar distributed lock manager is used by the Oracle Parallel Server and is implemented in the IBM Highly Available Cluster Multiprocessor (HACMP).

In addition to concurrency control, the database buffer caches located at each node in the cluster need to be kept coherent. The simplest method for buffer coherency is the so-called broadcast invalidation protocol, in which buffer blocks at all remote nodes are invalidated when a block is updated at the node holding the corresponding update lock. Broadcast invalidation has a high overhead, particularly when the number of nodes in the system is large. To reduce the overhead for buffer coherency control, integrated concurrency-coherency control schemes are available and are described in Reference 19. Essentially, when a global lock is obtained, information is also provided by the integrated concurrency-coherency controller on whether the local buffer copy of the corresponding page at the requesting node is valid. Further issues related to concurrency and coherency control are discussed in the section on qualitative comparisons.

Virtual shared-disk model. The virtual shared-disk (VSD) model[9] is illustrated in Figure 4. Here, we have a blending of the partitioned and shared data schemes. As in the data partitioning model, the disks are partitioned among the nodes in the system. There is a set of logical storage nodes to which the disks are connected and a set of logical processing nodes on which database transactions run. (Logical storage and processing nodes may reside on the same

Figure 4 Virtual shared-disk architecture



physical processor.) I/O requests to disks connected to the storage nodes are trapped in the disk driver and shipped to the storage nodes. The required data block(s) are retrieved by the storage nodes and returned to the requesting node; the storage nodes have all processing done at interrupt level, leading to improved efficiency. By comparison, a typical network file system processes requests by daemons at the equivalent of the storage nodes. By doing all storage node processing at interrupt level, VSD achieves an order of magnitude better performance than typical network file systems. When there is affinity of data access at the logical processing nodes, the logical processing and storage nodes are typically combined on a physical node. The storage nodes may also have a memory buffer cache for recently accessed disk blocks directed to that storage node.

The VSD model is similar to the I/O shipping model outlined earlier. The difference is that the VSD model is completely transparent to the parallel database system running on the processing nodes. Thus, to the database, the VSD model is identical to the data-sharing model. Furthermore, the concurrency and buffer coherency models used are the same as that for the shared-disk model. I/O shipping is typically done at the buffer manager layer of the database, which also results in higher overhead for remote call processing.

The VSD model is used on the IBM Scalable POWERparallel System[15] running the Oracle Parallel Server.

S/390 Parallel Sysplex architecture. We outline aspects of the Parallel Sysplex architecture to the extent required by subsequent sections of this paper; further details can be found in Reference 1. The S/390 Parallel Sysplex architecture is illustrated in Figure 1. As can be seen, the shared-disk topology is used as each node has access to all disks. Thus, the ex-

cellent availability attributes of a shared-data architecture are present. Additionally, the nodes in the system have access to one or more coupling facilities. It is through the use of the functions provided by the coupling facility that the S/390 Parallel Sysplex is able to achieve excellent scalability beyond the capabilities of other data-sharing architectures.

The coupling facility consists of hardware and microcode to support the S/390 Parallel Sysplex architecture extensions. Coupling facilities are attached to S/390 processors using high-speed coupling links that typically provide data transfer rates of up to 100 MB per second. Most commands issued to a coupling facility complete in several hundred microseconds or less. The coupling facility supports three general types of functions, or behavioral models: lock, cache, and list. Each of these models has an associated "structure" that resides in the storage of the coupling facility.

The coupling facility lock model provides the mechanism to address the global concurrency issue with shared-data schemes. The lock structure contains lock table entries that are mapped to data blocks or records by the database manager software. Each lock table entry contains shared or exclusive indicators for each interested system. When a system wishes to obtain a lock for a data item, generally one quick trip to the coupling facility allows the lock to be granted (based on the requested and current state of the lock). Should the requested lock create an incompatible state, this lock "contention" is recognized, and further processing by the interested lock managers is required. Typically, over 99 percent of the time the lock can be granted immediately.

The coupling facility cache model is used to address the database buffer coherency issue, enabling each node in a sysplex to locally cache frequently referenced data items from globally shared databases. The cache structure consists of two parts: directory entries and optional data elements. A directory entry exists for each unique data block held in the local buffer pool of any system, or in a global pool. Each directory entry contains an indicator as to which systems currently have copies of the data block. In the protected storage of each system, a bit vector is defined mapping a bit to each local data buffer. Upon reading a data block from a shared database, the database manager registers the data block (by name) and its associated local bit-vector offset to the coupling facility directory entry, and sets the local bit-vector value to indicate a valid data block. Should the database manager on another system now update this data block, the update is communicated to the coupling facility. The coupling facility interrogates its directory and sends "invalidate signals" to the systems currently holding copies of the data block. An invalidate signal causes the local bit-vector bit corresponding to the data block to be set to indicate an invalid state; note that the act of setting the bit is handled by the coupling link hardware with no interruption or impact to work running on the system. Whenever a database manager wishes to use a locally buffered data block, a simple bit test indicates the validity. Should the test indicate an invalid state, the database manager will refresh its local copy of the data block. The data blocks themselves may also be stored in the optional data element part of the cache structure in the coupling facility. If this is done, a local copy of a data block that is found to be invalid may be refreshed from the updated copy found in the coupling facility. Many shared-data architectures degrade because of the overhead of buffer invalidation. It can be seen that the coupling facility cache model successfully and efficiently solves this problem.

The coupling facility list structure provides a general-purpose queuing construct useful for a wide variety of applications. A list structure consists of one or more list headers and list elements. Elements may be added or removed from lists using LIFO/FIFO (last-in-first-out/first-in-first-out) or key-sequenced ordering. Programs can register an interest in lists and be notified when a list makes a transition from empty to nonempty. List structures are used for intersystem messaging and workload distribution. For example, instances of a workload manager across the Parallel Sysplex may periodically exchange performance status information through the use of a list structure. This information is then used for dynamically routing transactions away from overutilized systems to underutilized systems.

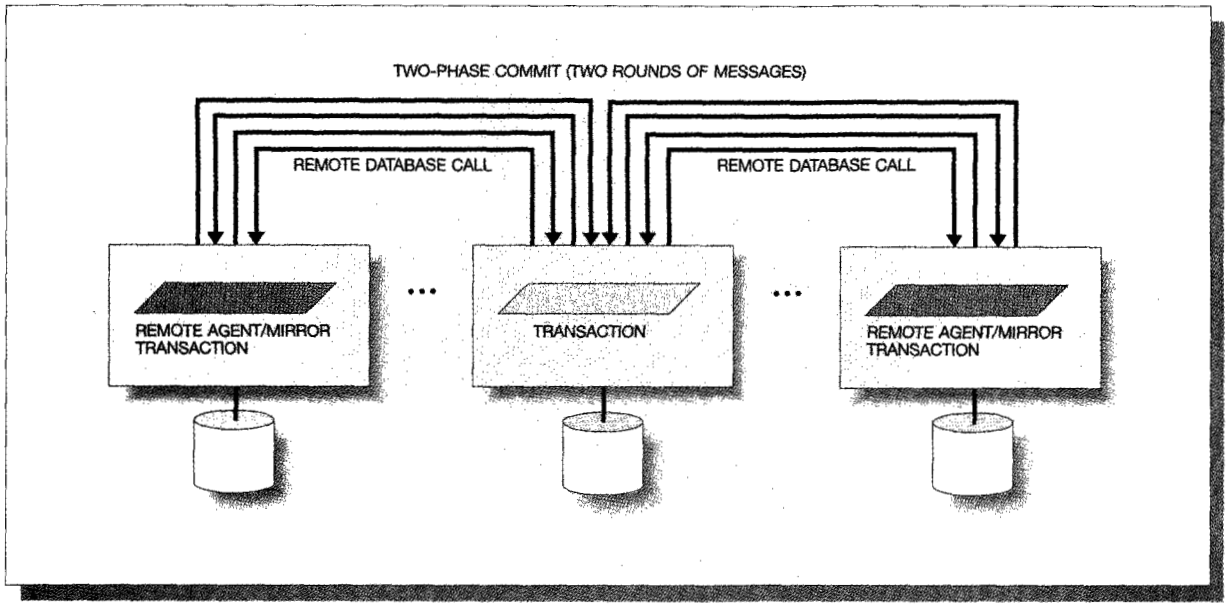
Qualitative comparison

We examine factors that impact the scalability of the cluster architectures, outlined in the previous section, and discuss how they motivated the Parallel Sysplex design.

Several contending factors affect the comparative performance of these approaches to clustering. They include:

- Coupling or clustering efficiency
- Buffer hit ratio

Figure 5 Remote database call and two-phase commit for data partitioning



- Load balancing under normal operation
- High availability and load balancing after failure

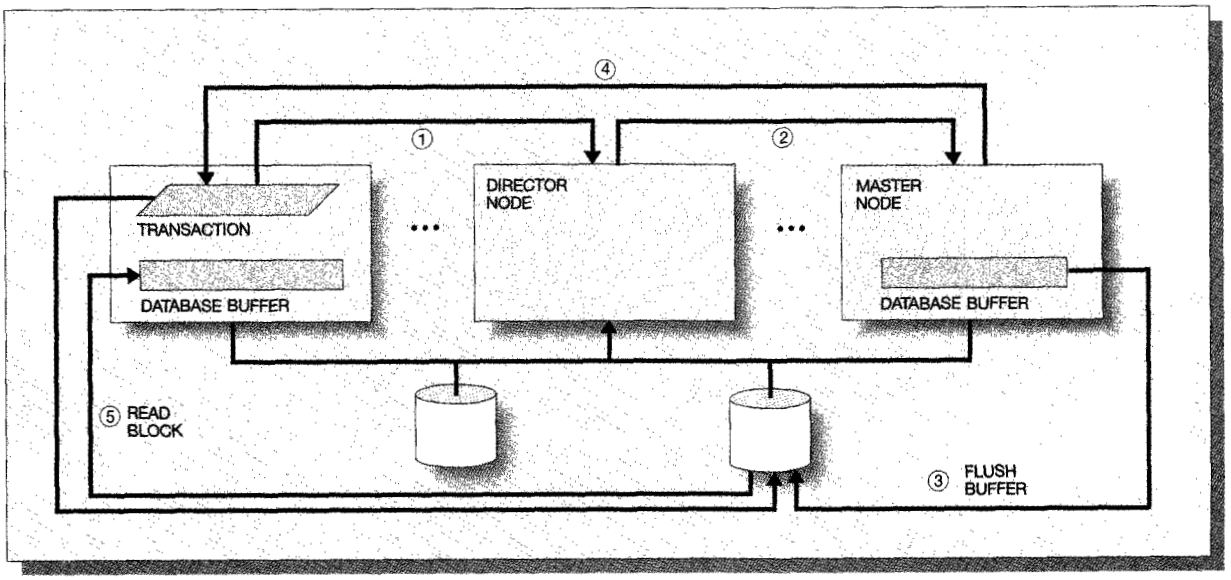
Coupling efficiency. The coupling efficiency is defined as the ratio of the throughput achieved by a multinode cluster to that of an ideal system with a throughput that is linear in the number of nodes. The coupling efficiency is determined by several factors, including the CPU overhead due to cluster protocol processing, any additional I/O operations, possible locking delay due to contention, and additional latency for some operations in the cluster.

For the data partitioning architecture, additional costs are incurred for remote requests to access nonlocal databases, as illustrated in Figure 5. The data are partitioned among the nodes in the cluster and can be accessed only by the node at which the data reside. Consequently, if a transaction running at a node needs to access data located at another node, a remote database call is made, which entails communications overhead and delay. In addition, the requesting node must perform a task switch since the remote request processing has a delay greater than twice the task switch time. Executing the database call at the remote node requires allocating an agent, or a so-called mirror transaction, to process the call, with concomitant overhead.[20] Finally, since multiple nodes may process database requests for the same transaction, all the nodes involved in processing update requests on behalf of the transaction must be part of a two-phase commit process. Two rounds of messages are exchanged during the two-phase commit operation. However, the number of nodes in the two-phase commit is bounded by the number of update requests in a transaction.

For the data-sharing approach, as illustrated in Figure 6, global concurrency control is needed, and additional overhead is incurred to obtain and release global locks. This is in contrast to the data partitioning architecture where the database is partitioned and hence all lock requests to the data are local to the node owning the corresponding database. That is to say that under data partitioning, the lock information can be maintained locally within local memory of the node, whereas data sharing requires global locks.

The global concurrency control can be implemented either via distributed locking or by a lock assist, as in the Parallel Sysplex coupling facility. The distributed approach generally relies on standard communication protocol to request and release lock requests.[4,5] Figure 6 illustrates the flow for a typical

Figure 6 Distributed global locking for disk sharing



distributed concurrency control protocol as in Reference 5; if the node on which a transaction runs is not the master for the required lock (see previous section), obtaining the global lock typically involves a message to the lock director and then to the current lock master, which then grants the lock. This operation entails a large processing overhead and delay to obtain a lock.

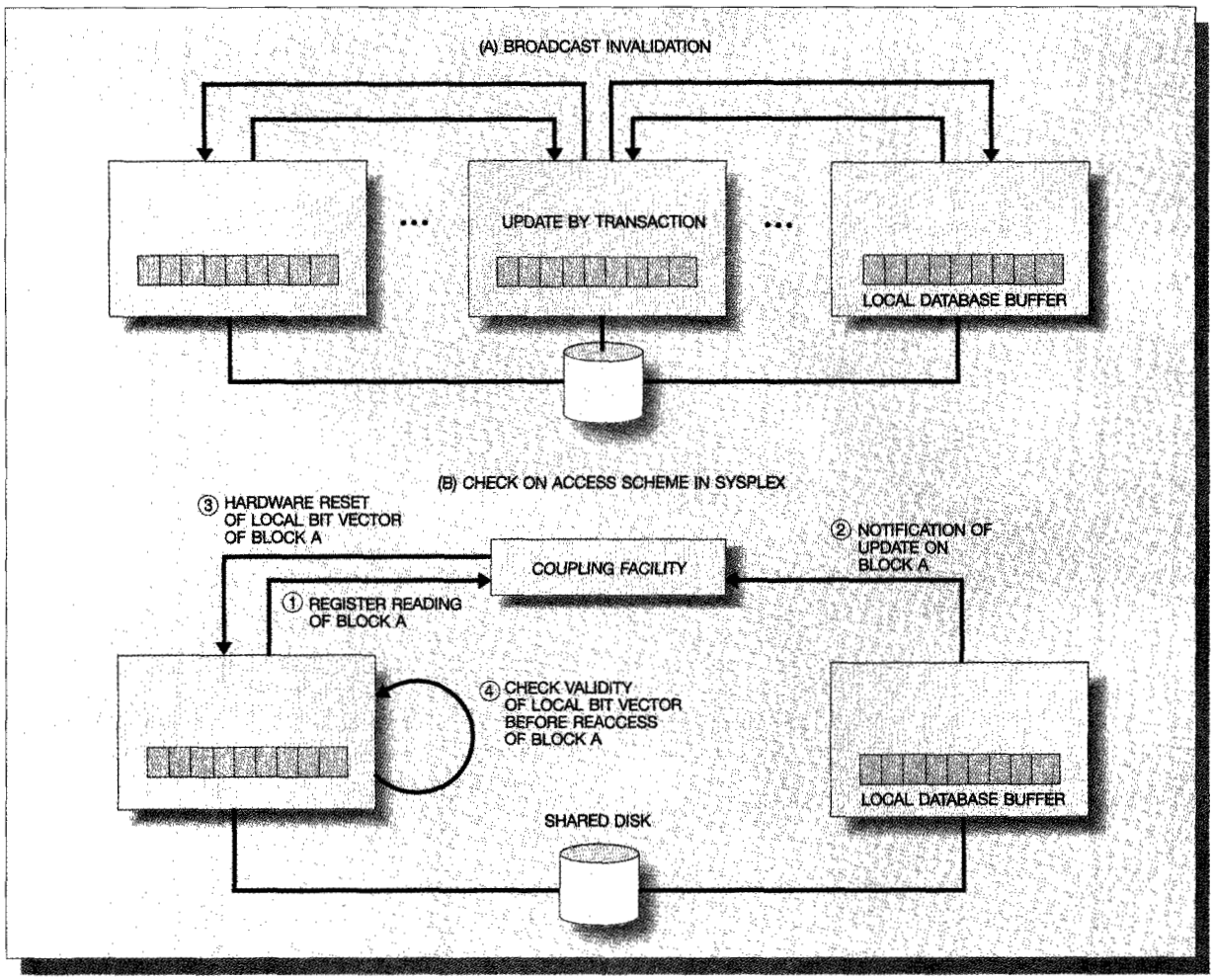
The lock assist approach, as in the Parallel Sysplex coupling facility, uses specialized hardware and protocols or instructions to obtain global locks and can reduce the locking overhead significantly.[6,21,22] This approach is demonstrated by the high efficiency of the coupling facility, as shown subsequently.

The locking protocol can be either eager (release locks on transaction commit) or lazy (release locks only on conflict).[23,24] Lazy protocols are adequate if there is a low degree of sharing, but they have a high overhead and delay for moderate and high degrees of sharing. Typical distributed locking methods use lazy lock release. Consequently they perform reasonably well at low degrees of sharing, as in benchmarks like the Transaction Processing Performance Council benchmark C (TPC-C**),[25] but have a large overhead and delay for many real workloads. *The coupling facility uses eager locking in order to handle varying degrees of sharing*, since many applications when directly ported to a cluster have high degrees of sharing.

The granularity of locking for global locks can be at either the physical block level or record level. Block level locking can result in false contention, especially for hot spots.[6] *The coupling facility supports record-level locking to avoid false contention.* Typical distributed locking protocols use block-level locking.

Another effect is that of database buffer cache coherency control, as outlined earlier. Since a data block can be present in the database buffers of more than one node, when an update occurs, all other buffered copies become obsolete. Coherency control needs to be provided to invalidate the obsolete granules in the local buffers and maintain the coherency of the buffer contents. There are several alternative approaches to coherency control. One scheme is referred to as broadcast invalidation,[26,27] illustrated in part A of Figure 7. When one node updates a block, all other nodes are informed that the block, if held in the local buffer of the other nodes, is invalid. It has several implications. First, an overhead is associated with sending the buffer invalidation messages. If this information is broadcast to every node, the overhead grows linearly with the number of nodes. Second, as a result of the update and buffer invalidation, another node that uses the updated granule

Figure 7 Buffer coherency protocols



will have a local buffer miss, which results in an increase in I/O activity for the shared-disk architecture and a shared buffer access if shared global memory is present.

An alternative approach, referred to as the check-on-access scheme in References 5 and 19, avoids the broadcast overhead by providing a mechanism to track the validity of in-memory data granules and having each node explicitly check the validity of a data granule upon access. This approach can substantially reduce the message overhead for invalidation and is especially suitable for a large number of nodes. Since there are no explicit invalidation messages, invalid granules continue to stay in the local buffer and are not detected until reference time. It reduces the buffer hit probability a little; however, in Reference 26, a performance study shows this reduction of buffer hit probability to be small, especially when the data access pattern is skewed (i.e., nonuniform), as is often observed in transaction processing environments.[28]

As described in the section on the S/390 Parallel Sysplex architecture, *the Parallel Sysplex coupling facility combines the check-on-access scheme with a granular and nondisruptive cross-invalidate mechanism to efficiently provide database cache coherency*. The Parallel Sysplex buffer coherency method is illustrated in part B of Figure 7: A node registers reading a data

block to the coupling facility (1); on update of a block, the coupling facility is notified (2); the coupling facility selectively resets a local bit vector (in hardware) only on nodes that have registered interest in the block (3); when a block is reaccessed, the local bit vector is checked for validity (4).

In the shared-disk architecture, when a data block needed by a transaction is cached in the database buffer of another node, the data are typically obtained by writing the block to disk from the cache and reading the data from disk at the requesting node, as illustrated in Figure 6. This caching leads to large overhead, delay, and increased disk bandwidth, especially for hot shared data, leading to so-called "ping-ponging." *In the coupling facility, hot shared data are cached in a shared global buffer cache.*[29,30] As a result, the efficiency is high, even with high degrees of sharing, and the ping-ponging penalty is low, as quantified subsequently.

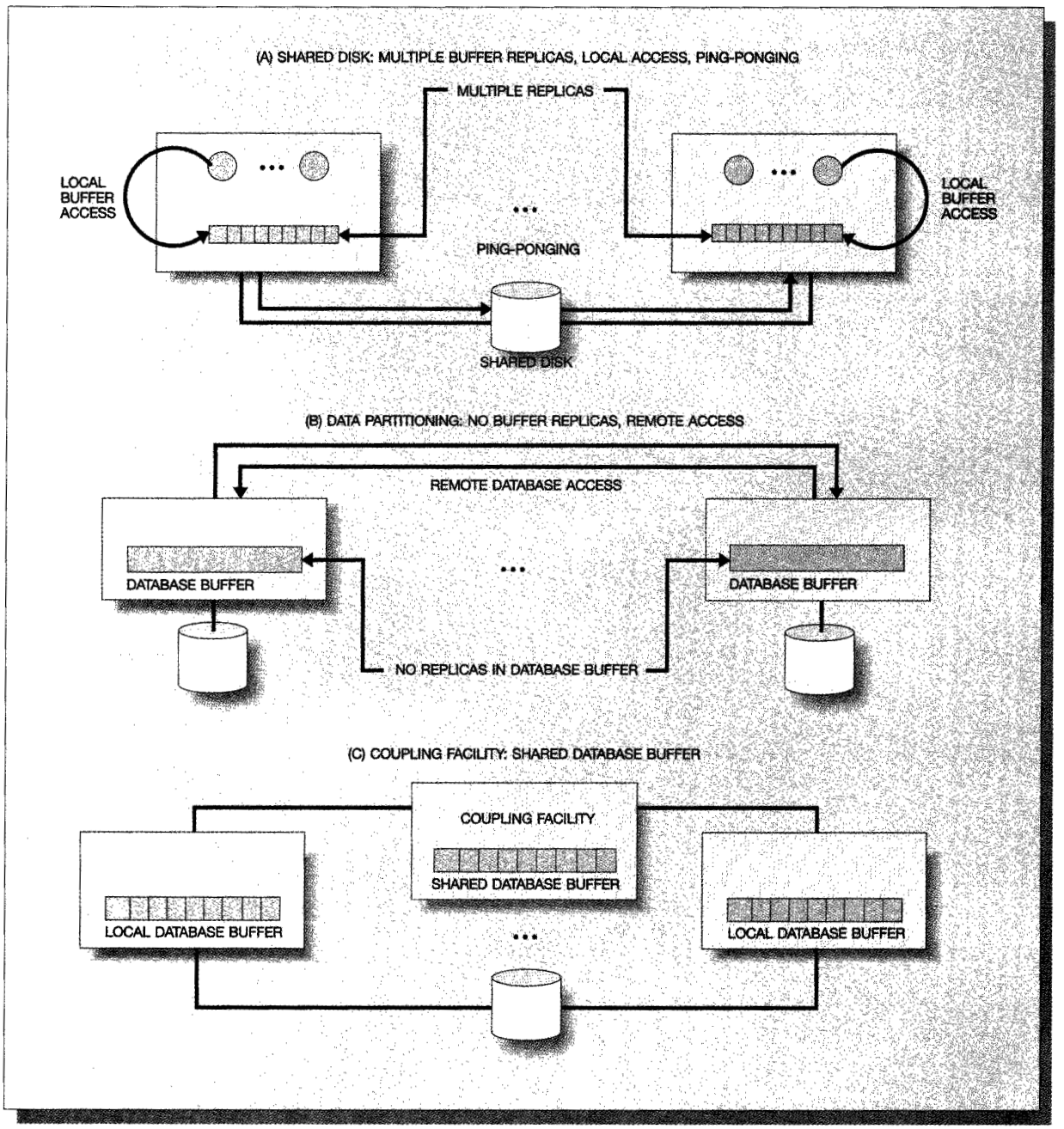
Buffer hit probability. Next, we consider the issue of efficient buffer usage. In a shared-disk environment, the buffer management is decentralized, and each buffer manager makes independent decisions on which granules to buffer. In essence, there is no cooperation or division of work among the local buffer managers. If the transactions are routed to the nodes in a round-robin fashion without exploiting the transaction affinity in various database partitions, each local buffer manager will observe similar data reference patterns and try to buffer similar data blocks. This action could lead to significant replication of data blocks among the nodes, as illustrated in part A of Figure 8.

This situation is in contrast to the data partitioning architecture, where the database is partitioned among the nodes, and each node caches data blocks from its own partition. (Note that the buffering under data partitioning is similar to that under a single node system where multiple buffer pools and different partitions use different buffer pools.[31]) Even without the invalidation effect, the local buffer hit ratio is expected to be lower under the shared-disk approach for the same total buffer size, since the shared-disk approach uses the local buffer to cache the entire database instead of caching just one partition of the database (as is done under the data partitioning architecture). However, there is a trade-off in that a remote database call is required to access data in a remote buffer for data partitioning, rather than local buffer access using the shared-disk approach, as illustrated in part B of Figure 8. The buffer hit ratio degradation of the local buffer under the shared-disk architecture can be alleviated somewhat by using affinity-based routing, as explained in the next subsection. Further, with the data partitioning architecture, the memory is distributed among the nodes in the cluster. A single large memory has a higher buffer hit ratio than partitioned memories of the same total size.

The shared database buffer cache provided in the coupling facility has advantages over either the shared-disk or data partitioning architectures. Data that are shared among the nodes can be placed in the shared buffer, leading to a higher buffer hit ratio than a partitioned buffer and avoiding replication in multiple local database buffers, as illustrated in part C of Figure 8. In Reference 32, a performance study on a global shared buffer in a data-sharing environment is conducted to understand the trade-offs of local and global shared buffers. It is found that proper usage of a global shared buffer can substantially improve the overall buffer hit probability and transaction response time addressing the replication and invalidation issues in the local buffers under data sharing. Comparison of the buffer hit probabilities under the different architectures can be found in Reference 33. Quantification of the improvement in the buffer hit probability that accrues to the shared buffer in the coupling facility is given in the section on Parallel Sysplex performance.

Load balancing. In a transaction processing environment, there are generally multiple transaction classes and relations (or physical databases in a hierarchical database).[8] Each transaction class may exhibit affinity to certain relations, or partitions of relations. For instance, in the Transaction Processing Performance Council benchmark B (TPC-B**),[34] transactions have affinity with the associated (bank) branch with which the account is associated. Similarly, in the TPC-C benchmark,[25] transactions have affinity with the associated warehouse. Affinity clustering[33,35] is the process of partitioning the transactions and relations into affinity clusters (ACs) according to their database references. The entity consisting of the relations associated with each AC is referred to as a DB (database) cluster. Transactions associated with an AC are routed to the same node. Additionally, under data partitioning, each DB cluster is assigned to the node where its associated transactions are executed. Under data-sharing environments, the database partitioning is logical (rather than physical) as pages of each DB cluster are buffered (referenced) from its associated node. This reduces the CPU over-

Figure 8 Buffer issues



heads under data partitioning, caused by reduced remote database accesses and two-phase commits, and improves the local buffer hit probability and ping-ponging for the shared-disk architecture. It is not always easy and sometimes impossible to partition a database to form affinity clusters and at the same time balance the transaction load at all nodes. In Reference 33, the effect of affinity clustering on the per-

formance of the different coupling architectures is studied, and the Parallel Sysplex-like architecture is found to be most robust to workload partitionability.

To illustrate the partitionability of real workloads and the degree of affinity (i.e., fraction of local database accesses) that can be achieved, we examine some workloads from DB2 and IMS database environments. In Reference 36, a relational database workload is analyzed for a production DB2 system that runs an accounting-type application in a petroleum company. There are 323 relations or tables and 305 application plans or transaction classes. However, in the two-hour measured interval, more than 90 percent of the transactions executed come from seven plans, concentrating the database references on a few relations. Thus, partitioning the load evenly beyond a moderate number of ACs is difficult.

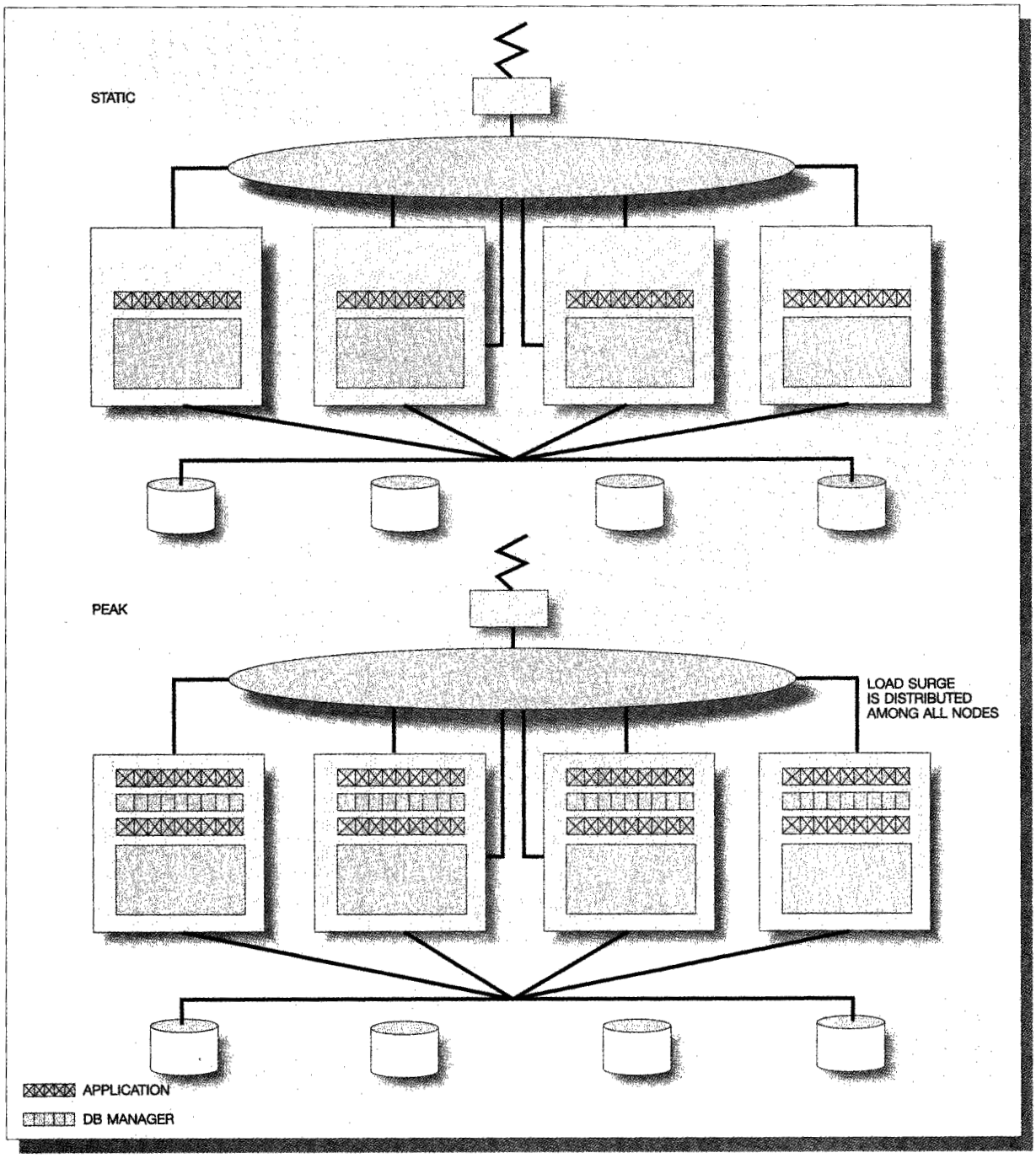
In References 6 and 8, two transaction workloads from production IBM IMS environments are studied. The first system is an on-line planning database system with 103 physical databases and 143 transaction classes. The second system is a parts inventory database system with 158 physical databases and 176 transaction classes. Traces of database (DL/I, or Data Language 1) calls were obtained from the two systems and analyzed. These calls are referred to as workload 1 and workload 2, respectively. Let N be the number of partitions into which the workload is split. In workload 1, a large fraction of the transactions show a strong affinity to one physical database and must be assigned to the same AC. These transactions constitute more than 20 percent of the transaction load. For $N > 5$, the partitioning into ACs becomes skewed since one AC with more than 20 percent of the load will be larger than the other ACs. That is to say, the workload is not evenly partitionable for $N > 5$. For a smaller number of nodes, the transactions can be split into N ACs, where the load imbalance among the nodes is within 5 percent. Furthermore, the affinity attainable, i.e., for each AC the fraction of database references that can be made to stay within the affiliated DB cluster, tends to go down as the number of nodes increases. For $N = 2$, the transactions can be partitioned such that the affinity of each AC is around 0.95, whereas for $N = 3$ and 4, the affinities deteriorate to 0.82 and 0.74, respectively. In workload 2, partitioning skew is again observed, where 32 percent of the transaction load falls onto one AC. That is to say, for $N > 3$, the partitioning will be skewed, as the load on one of the ACs becomes larger than the other ACs. For smaller values of N, the transaction load can be split into N ACs, where the load imbalance among the nodes is within 5 percent. Again we observe the deterioration of affinity as the number of nodes increases. For $N = 2$, the transactions can be partitioned to achieve an affinity of 0.75, whereas for $N = 3$, the affinity deteriorates a bit to 0.68.

These case studies indicate that it is difficult with real workloads to form affinity clusters with low fractions of remote database calls, and simultaneously achieve a balanced load among the nodes in the system. Therefore, it is important to achieve high coupling efficiency when any transaction may run on any node in the cluster. We show later that the S/390 Parallel Sysplex achieves excellent scalability under these conditions.

Dynamic load balancing. We now examine the impact of dynamic load changes on these architectures. Let us assume that at steady state the workload is perfectly partitionable and balanced, and consider the case of a sudden load surge in one of the transaction classes. For example, the most active stocks in the stock market may change from period to period, depending upon which companies become takeover targets. The corresponding database partition would have a sudden surge in load. For data sharing, a front-end router can be employed to spread the load surge across all nodes, as illustrated in Figure 9. Certainly, by doing this, the buffer hit probability at each node decreases. The reduction comes from two factors. First, in each of the nodes except the surge node (i.e., the node affiliated with the transaction class with the load surge), the data blocks in the buffers come from two partitions: the original partition logically assigned to the node and the partition corresponding to the surge transaction class. The second factor is the cross-invalidation effect on the data blocks from the partition affiliated with the surge load. This factor potentially may affect the surge node more than the other nodes as the buffer in the surge node contains only the data blocks from the partition accessed by the surge transaction class and, hence, is more susceptible to invalidation.[37]

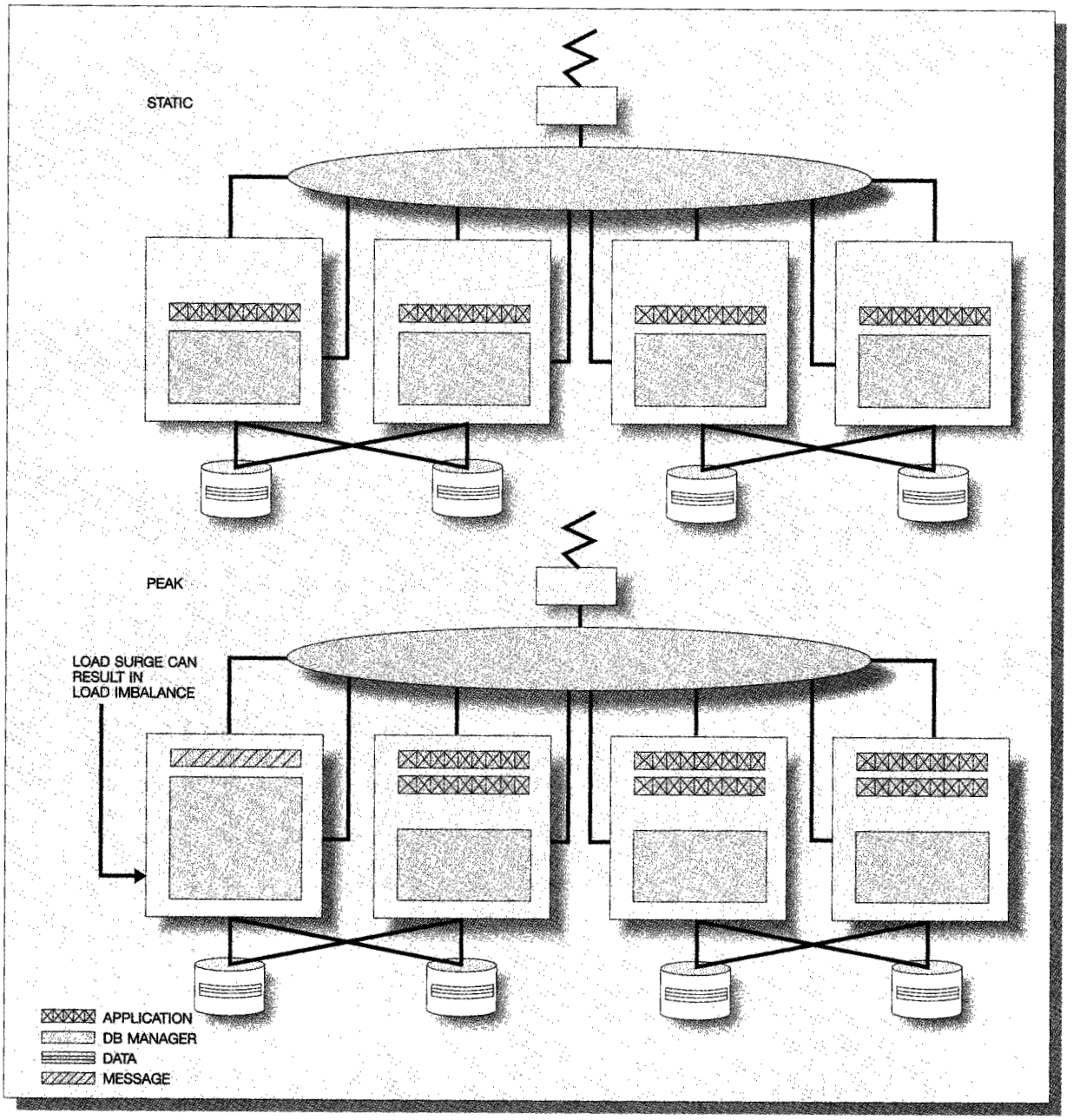
Thus, the stronger the affinity among transactions in terms of the data accessed, the more severe is the impact of ping-ponging of the shared data among nodes. Consider, for example, if the TPC-B benchmark[34] transactions to the same branch are routed to different nodes due to a load surge, then the branch and teller blocks will ping-pong between nodes. For disk sharing, the ping-ponging and dis-

Figure 9 Data sharing—surge in transaction load



tributed concurrency control overhead limits the load that can be moved away from nodes with a heavy load. *The efficiency of the coupling facility for heavily shared data, and for locking, allows for excellent dy-*

Figure 10 Data partitioning—surge in transaction load



namic load balancing by moving load away from hot nodes with relatively small overhead.

For data partitioning, load sharing is difficult, as illustrated in Figure 10. The processing load associated with each transaction can be divided into two parts. One is application processing and the other is database call processing (labeled as DB manager load in the figure). The database call processing must be done at the node owning the database partition

that the call is referencing or updating, whereas the application processing can be done at any node selected by the front-end router. This phenomenon under data partitioning is referred to as the *database call server bottleneck*.[37] If we try to spread the transaction application processing of the surge transaction class to other nodes, the database calls still have to be sent back to the surge node; thus, as illustrated in Figure 10, the surge node gets a disproportionate load. Furthermore, there is additional communications overhead for shipping the database calls and two-phase commit overhead for updates. This added overhead reduces (or nullifies) the effectiveness of load sharing. In Reference 37, a performance comparison of different cluster architectures for handling system dynamics showed that the data partitioning architecture is least able to cope with load variation and requires a large amount of contingent capacity to be put aside as compared with the data-sharing architecture. The data-sharing architecture can also improve load balancing at the disk level.[38]

High availability and load balancing after failure. We now examine the fault tolerance to single node failures. We consider the case where one of the CPUs has failed, while the disks are still accessible. For data partitioning, we assume that the straightforward paired backup strategy is used where each active node is paired with another active node to form a backup pair using twin tail disks,[39] as shown in Figure 11. (There are certainly other backup scenarios. Using a single spare node to back up any failed node is another possibility. However, this would require direct connection to all disks, which is more of a data-sharing flavor, and is thus not considered here. Also, database replication and disk-mirroring strategies that require more disk space and overhead are not considered here.) When a CPU fails, the disks of the failed node are switched over to its backup node. Now the backup node needs to serve database calls to two partitions: its own partition and the partition of the failed node. The dual serving role has implications for both the buffer hit probability and CPU load. The buffer hit probability decreases as the buffer now has granules from two partitions. As the database call processing of the two partitions cannot be off-loaded to other nodes, it results in a server bottleneck, as illustrated in Figure 11.

For the data-sharing architectures, the transaction class originally destined for the failed node will be spread across all the other nodes, as shown in Figure 11. However, if there is affinity among the data accessed by transactions directed to the failed node (as in the TPC-B example above), distributing the load across the remaining nodes can lead to ping-ponging (as discussed before) resulting in some reduction in buffer hit probability. *The coupling facility provides efficient operation even when the load of the failed node is redistributed* because of its ability to efficiently support workloads where transactions are randomly routed among nodes in the cluster. In Reference 37, the contingency capacity requirement to sustain a single node failure under different architectures is studied. It was found that the data partitioning architecture requires substantially more contingent capacity as compared to the data-sharing architecture.

Summary. In summary, the key advantages of the Parallel Sysplex use of the coupling facility are:

- Low coupling overhead due to efficient integrated concurrency and coherency control and access to the shared buffer in the coupling facility
- Excellent dynamic load balancing due to the ability to route load to any node in the cluster with low coupling penalty
- High availability due to the ability to rebalance the load among the remaining nodes, with excellent load balancing after a failure
- High buffer hit ratio due to shared buffer, potentially with a higher buffer hit ratio with a larger number of nodes

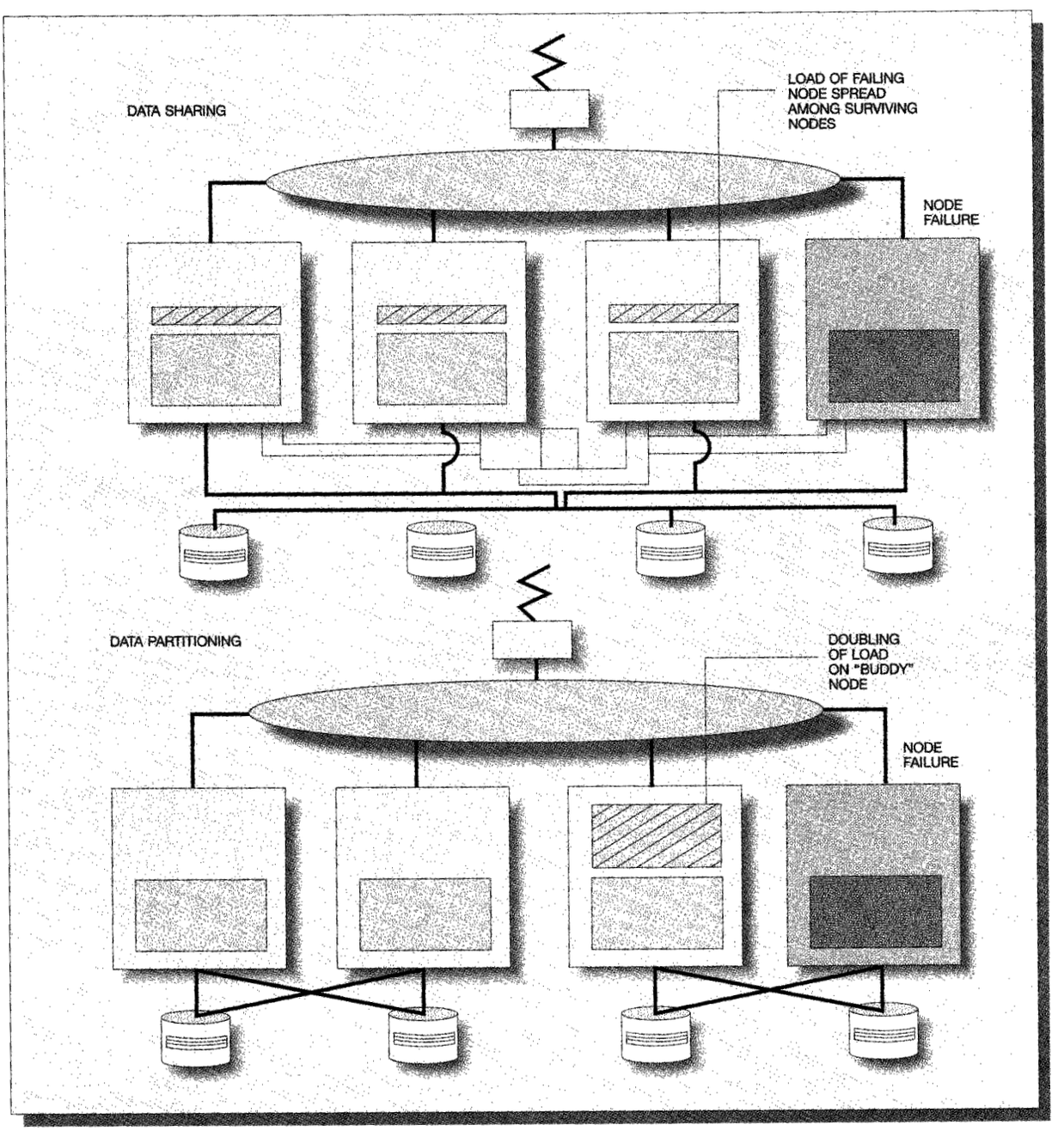
These advantages of the Parallel Sysplex design are quantified next.

S/390 Parallel Sysplex performance and scaling

A set of laboratory benchmarks[40] was conducted to exhibit the excellent performance and scalability characteristics of the S/390 Parallel Sysplex. Additionally, data were collected from several in-production Parallel Sysplexes running mission-critical applications. From the results of these measurements, we quantify the aspects discussed in the previous section that illustrate the key advantages of the Parallel Sysplex.

Laboratory benchmark environment. The measured hardware configurations consisted of two, eight, and sixteen S/390 9672 CMOS (complimentary metal-oxide semiconductor) systems coupled by one (for the 2- and 8-system tests) or two (for the 16-system test) S/390 9674 CMOS coupling facilities. Two different software configurations were measured. The CICS-TM/IMS-DB configuration included the Customer

Figure 11 Fault tolerance



Information Control System (CICS*) transaction manager (TM), the CICSPlex* Systems Manager, and the IMS database manager. The IMS-TM/DB2 configuration included the IMS transaction manager and the DB2 database manager. The workloads reflected mixes of on-line transaction processing (OLTP) transactions covering diverse business functions including warehousing, reservations, and banking applica-

Table 1 Normalized capacity for 2-, 8-, and 16-node Parallel Sysplex from benchmark data

	Number of Systems in the Parallel Sysplex		
	2	8	16
CICS-TM/IMS-DB	1.00	3.89	7.40
IMS-TM/DB2	1.00	3.87	NT

NT = not tested

tions. All databases were fully shared and accessible to all systems. There were no transaction or data affinities; thus, any transaction could run on any system.

Simulated user sessions were evenly distributed among the S/390 systems. Each session randomly selected a script of transactions to submit, and random-duration think times were inserted between the transactions. For the CICS-TM measurements, the CICS transaction manager (under guidance from the CICSPlex Systems Manager) would decide either to process transactions locally or to route the transactions to other systems within the Parallel Sysplex for best workload balancing. For the IMS-TM measurements, transactions were processed on the system to which the session was connected.

Scalability and coupling efficiency. Table 1 shows the normalized capacity (based on the transactions processed per second in the two-system Parallel Sysplex) for each measured point.

With use of the CICS-TM/IMS-DB measurements, a comparison of the normalized capacity to the "ideal" capacity at the 8- and 16-system points may be made to highlight the scalability of the Parallel Sysplex. Growing a Parallel Sysplex from two systems to eight systems increases the potential capacity by a factor of four. In reality, a factor of 3.89 capacity growth was measured, meaning that with the addition of six systems, the Parallel Sysplex realizes 97.25 percent (3.89/4) of the ideal! In scalability terms, adding the six systems results in a loss of only 2.75 percent or approximately 0.5 percent per system added. Likewise, growing the Parallel Sysplex from two systems to 16 systems yields a 7.4-fold increase in capacity, or 92.5 percent of ideal. This also averages to approximately 0.5 percent cost for each of the 14 systems added. The IMS-TM/DB2 8-system measurement shows similar excellent scalability results.

Most of the cost of moving from a single-system non-data-sharing configuration to a multisystem, fully data-sharing configuration occurs in the transition to a two-system Parallel Sysplex. It is during this transition that the cost to access records in shared databases is increased by the activity to the lock, cache, and list structures on the coupling facility to manage update serialization and local buffer coherency. Additionally, coupling facility activity on behalf of workload balancing is activated. Once this cost is paid, there is very little increase in overhead to connect additional systems as shown by the scalability measurements.

In order to observe this transition cost in production environments, data have been gathered from several Parallel Sysplexes that are running mission-critical applications. Table 2 summarizes data from four production Parallel Sysplexes. The number of OS/390 images is the number of nodes in the Parallel Sysplex. "CF access per CPU second" gives the rate of activity to the coupling facility, normalized by the total processing time across the Parallel Sysplex. This value provides a relative measure of the access intensity to the shared databases of each Parallel Sysplex. The final column gives the approximate percentage of the capacity of the Parallel Sysplex that is being consumed for activity related to coupling the systems. Note the high correlation between the coupling overhead and the access rate to the coupling

Table 2 Cost of coupling using the sysplex architecture from customer production data

Customer	Relative Size of Parallel Sysplex (9672-R61 = 1.0)	No. of OS/390 Images	CF Access per CPU Second	Approximate Percentage of Capacity
1	10	4	101	11
2	11	3	86	10
3	12	8	84	9
4	13	2	49	7
5	24	11	93	10

Table 3 Normalized database I/O per transaction from benchmark data

	Number of OS/390 Images			
	1	2	8	16
IMS DB	1.00	1.03	1.08	1.13
DB2	1.00	.96	.94	NT

Table 4 Dynamic load balancing for 8-node homogeneous Parallel Sysplex from benchmark data

CEC	Normalized-to-Average	
	CPU Busy	Transactions per Second
R61	.99	.99
R61	.99	1.00
R61	.99	1.00
R61	.99	1.00
R61	1.00	1.00
R61	1.00	1.00
R61	1.01	1.00
R61	1.03	1.01

Table 5 Dynamic load balancing for 9-node heterogeneous Parallel Sysplex from benchmark data

CEC	Normalized-to-Average	
	CPU Busy	Transactions per Second
R61	1.00	.93
R61	1.00	.94
R61	1.00	.94
R61	1.01	.95
R61	1.01	.95
R61	1.01	.95
R61	1.01	.95
R61	1.02	.95
821	.94	1.44

facility. The data indicate that the cost of coupling systems using the Parallel Sysplex architecture is approximately 10 percent.

I/O per transaction. In the section on qualitative comparisons, we outlined the advantage of having a shared database buffer cache among nodes in the cluster. This advantage is quantitatively illustrated in Table 3. The table shows the database I/Os per transaction, normalized with respect to that for a single system. Data are presented for the database managers of both IMS-DB and DB2. The data for IMS-DB show that the I/O activity per transaction, which is proportional to the inverse of the buffer hit ratio, increases gradually with the number of systems in the Parallel Sysplex. For a 16-system Parallel Sysplex, the I/O activity per transaction increases by about 13 percent. The reason for the increase in activity, which implies a decrease in the buffer hit ratio, is that IMS-DB did not store the updated data blocks in the coupling facility cache structure. Thus, invalidated local copies were refreshed from disk.

The data for DB2 in Table 3 show that the normalized number of I/Os per transaction actually falls with the increase in the number of systems in the Parallel Sysplex. The reason is that DB2 places updated data blocks in its coupling facility cache structure. Consequently, the total system buffers are used more efficiently, leading to lower numbers of I/Os per transaction, or a higher buffer hit ratio. Placing updated data in the coupling facility also allows deferred updates to disk, while releasing the locks at transaction commit time. This action prevents coupling degradation due to increased update I/O activity and lock contention. Another advantage is that highly shared data that are frequently updated are found in the coupling facility; this condition alleviates the degradation caused by ping-ponging of updated pages between systems in the Parallel Sysplex. By contrast, the (pure) shared-disk architecture has a high overhead because of ping-ponging.

Dynamic load balancing. Table 4 shows the load balancing achieved in a laboratory benchmark of a Parallel Sysplex composed of eight homogeneous systems (CEC, the central electronics complex). The table shows the CPU utilization of each system, relative to the average CPU utilization across the Parallel Sysplex. Also shown is the transaction rate, again relative to the average transaction rate per system. As can be observed, the workload was balanced very evenly across the systems.

Now, what happens when a "faster" system joins the Parallel Sysplex? Table 5 shows the results when a ninth system is added to the Parallel Sysplex. However, this system has a potential capacity that is 60 percent larger than each of the original eight systems. We find the workload is redistributed, achieving a fairly even balance of the CPU utilizations across the now heterogeneous Parallel Sysplex. The faster system automatically begins to process a share of the workload consistent with its capacity.

Conclusion

With the rapid advance in microprocessor speed and reduction of cost per MIPS (millions of instructions per second), coupling multiple microprocessors to support high transaction rates and high availability for OLTP and emerging applications becomes increasingly attractive. In this paper, we considered various cluster architectures, including shared-disk, partitioned (shared-nothing), virtual shared-disk, and Parallel Sysplex architectures, and examined the trade-offs between these architectures. The factors that motivated the System/390 Parallel Sysplex architecture and its key advantages were then described and quantified. Specifically we showed that:

- The Parallel Sysplex architecture results in close to linear growth in the transaction rate with the number of nodes in the Parallel Sysplex. For instance, in a benchmark environment, when growing the system from two to eight nodes, the Parallel Sysplex realized more than 97 percent of the ideal. Even considering the overhead in going from a single node to a multinode cluster environment, the cost of coupling for customer production systems has been observed to be around 10 percent. This high efficiency is due to the coupling facility, which provides very efficient intertransaction concurrency control, buffer cache coherency control, shared buffer management, and shared job queues.
- The shared buffer cache in the Parallel Sysplex coupling facility results in efficient sharing, and can actually reduce the I/O activity per transaction with a larger number of nodes.
- The Parallel Sysplex exhibits excellent load balancing even for heterogeneous systems. Furthermore, superior dynamic load balancing is supported because of the ability to route load to any node in the Parallel Sysplex with low coupling penalty.
- Another key advantage of the Parallel Sysplex design is that, after a failure of a node, the load can be rebalanced among the remaining nodes. It contrasts to the partitioned architecture, where load balancing can be constrained by static database partitioning.

Although these advantages of the Parallel Sysplex are considerable for many environments, for some environments and workloads the other cluster architectures do offer appropriate solutions.

Acknowledgment

We would like to thank the referees for their comments that helped us to significantly improve the paper.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Digital Equipment Corporation or the Transaction Processing Performance Council.

Cited references

1. J. M. Nick, B. B. Moore, J.-Y. Chung, and N. S. Bowen, "S/390 Cluster Technology: Parallel Sysplex," *IBM Systems Journal* **36**, No. 2, 172–201 (1997, this issue).
2. S. A. Calta, J. A. deVeer, E. Loizides, and R. N. Strangwayes, "Enterprise Systems Connection (ESCON) Architecture—System Overview," *IBM Journal of Research and Development* **36**, No. 4, 535–551 (July 1992).
3. J. C. Elliot and M. W. Sachs, "The IBM Enterprise Systems Connection (ESCON) Architecture," *IBM Journal of Research and Development* **36**, No. 4, 577–591 (July 1992).
4. J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, "IMS/VS: An Evolving System," *IBM Systems Journal* **21**, 490–510 (1982).
5. N. Kronenberg, H. Levy, and W. D. Strecker, "VAXcluster: A Closely-Coupled Distributed System," *ACM Transactions on Computer Systems* **4**, 130–146 (May 1986).
6. P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. W. Cornell, "On Coupling Multi-Systems Through Data Sharing," *Proceedings of the IEEE* **75**, No. 5, 573–587 (May 1987).
7. M. Stonebraker, "The Case of Shared Nothing," *IEEE Database Engineering* **9**, No. 1, 610–621 (1986).
8. D. W. Cornell, D. M. Dias, and P. S. Yu, "On Multisystem Coupling Through Function Request Shipping," *IEEE Transactions on Software Engineering* **SE-12**, No. 10, 1006–1017 (October 1986).
9. C. R. Attanasio, M. Butrico, C. A. Polyzois, S. E. Smith, and J. L. Peterson, *Design and Implementation of a Recoverable Virtual Shared Disk*, Research Report RC 19843, IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598 (1994).
10. D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A Scalable and Highly Available Web Server," *Proceedings of COMPCON '96: Technologies for the Information SuperHighway*, IEEE Computer Society Press, Los Alamitos, CA (February 1996).
11. D. W. Cornell, D. M. Dias, P. S. Yu, and A. Thomasian, "Performance Comparison of IO Shipping and Database Call Shipping: Schemes in Multisystem Partitioned Databases," *Performance Evaluation* **10**, No. 1, 15–33 (October 1989).
12. D. J. Dewitt et al., "The Gamma Database Machine Project," *IEEE Transactions on Knowledge and Data Engineering* **2**, No. 1, 44–62 (March 1990).
13. C. K. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G. P. Copeland, and W. G. Wilson, "DB2 Parallel Edition," *IBM Systems Journal* **34**, No. 2, 292–322 (1995).
14. C. K. Baru, G. Fecteau, A. Goyal, A. H. Hsiao, A. Jhingran, S. Padmanabhan, and W. G. Wilson, "An Overview of DB2 Parallel Edition," *SIGMOD Record* **24**, No. 2, 460–462 (June 1995).
15. T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 System Architecture," *IBM Systems Journal* **34**, No. 2, 152–184 (1995).
16. E. Rahm, "Design of Optimistic Methods for Concurrency Control in Database Sharing Systems," *Proceedings of the 7th International Conference on Distributed Computing Systems*, Berlin (September 1987).

17. A. Bhide and M. Stonebraker, "A Performance Comparison of Two Architectures for Fast Transaction Processing," *Proceedings of the 4th International Conference on Data Engineering*, Los Angeles, CA (February 1988), pp. 536–545.
18. P. S. Yu, D. M. Dias, and S. S. Lavenberg, "On the Analytical Modeling of Database Concurrency Control," *Journal of the ACM* **40**, No. 4, 831–872 (September 1993).
19. D. M. Dias, B. R. Iyer, J. T. Robinson, and P. S. Yu, "Integrated Concurrency Coherency Controls for Data Sharing," *IEEE Transactions on Software Engineering* **SE-15**, No. 14, 437–448 (April 1989).
20. *Customer Information Control System/Virtual Storage (CICS/VS): General Information Manual*, GC33-0155, IBM Corporation; available through IBM branch offices.
21. J. T. Robinson, "A Fast General Purpose Hardware Synchronization Mechanism," *SIGMOD Record* (1985), pp. 122–130.
22. N. S. Bowen, D. A. Elko, J. F. Isenberg, and G. W. Wang, "A Locking Facility for Parallel Systems," *IBM Systems Journal* **36**, No. 2, 202–220 (1997, this issue).
23. A. Dan and P. S. Yu, "Performance Analysis of Coherency Control Policies Through Lock Retention," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA (June 1992), pp. 114–123.
24. C. Mohan and I. Narang, "Efficient Locking and Caching of Data in the Multi-System Shared Disks Transaction Environment," *Proceedings of the International Conference on Extending Database Technology*, Vienna (March 1992).
25. S. T. Leutenegger and D. M. Dias, "A Modeling Study of the TPCC Benchmark," *SIGMOD Record* **22**, No. 2, 22–31 (June 1993).
26. A. Dan and P. S. Yu, "Performance Analysis of Buffer Coherency Policies in a Multi-System Data Sharing Environment," *IEEE Transactions on Parallel and Distributed Systems* **4**, No. 3, 289–305 (March 1993).
27. A. Dan, D. M. Dias, and P. S. Yu, "The Effect of Skewed Data Access on Buffer Hits and Data Contention in a Data Sharing Environment," *Proceedings of the 16th International Conference on Very Large Databases*, Brisbane, Australia (August 1990), pp. 419–431.
28. A. Dan, P. S. Yu, and J-Y. Chung, "Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability," *VLDB Journal* **4**, No. 1, 127–154 (1995).
29. M.-S. Chen, P. S. Yu, and T.-H. Yang, "On Coupling Multiple Systems with a Global Buffer," *IEEE Transactions on Knowledge and Data Engineering* **8**, No. 2, 339–344 (April 1996).
30. E. Rahm, "Use of Global Extended Memory for Distributed Transaction Processing," *Proceedings of the 4th International Workshop on High Performance Transaction Processing*, Asilomar, CA (September 1991).
31. J. Z. Teng and R. A. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance," *IBM Systems Journal* **23**, No. 2, 211–218 (1984).
32. A. Dan, P. S. Yu, and D. M. Dias, "Performance Modelling and Comparisons of Global Shared Buffer Management Policies in a Cluster Environment," *IEEE Transactions on Computers* **43**, No. 11, 1281–1297 (November 1994).
33. P. S. Yu and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," *IEEE Transactions on Knowledge and Data Engineering* **6**, No. 5, 764–786 (October 1994).
34. *Benchmark Handbook*, J. Gray, Editor, Morgan Kaufmann Publishers, San Mateo, CA (1991).
35. P. S. Yu, D. M. Dias, D. W. Cornell, and B. R. Iyer, "Analysis of Affinity Based Routing in Multi-System Data Sharing," *Performance Evaluation* **7**, No. 2, 87–109 (June 1987).
36. P. S. Yu, H.-U. Heiss, S. Lee, and M.-S. Chen, "On Workload Characterization of Relational Database Environments," *IEEE Transactions on Software Engineering* **18**, No. 4, 347–355 (April 1992).
37. P. S. Yu and A. Dan, "Performance Evaluation of Transaction Processing Coupling Architectures for Handling System Dynamics," *IEEE Transactions on Parallel and Distributed Systems* **5**, No. 2, 139–153 (February 1994).
38. K. L. Wu, P. S. Yu, J. Y. Chung, and J. Z. Teng, "A Performance Study of Workfile Disk Management for Concurrent Mergesorts in a Multiprocessor Database System," *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich (September 1995), pp. 100–109.
39. J. A. Katzman, "A Fault-Tolerant Computing System," *Proceedings of the 11th International Conference on System Science* (1978), pp. 85–102.
40. *System/390 MVS Parallel Sysplex Performance*, SG24-4356-01, IBM Corporation (March 1996); available through IBM branch offices.

General references

H. Boral et al., "Prototyping BUBBA, A Highly Parallel Database System," *IEEE Transactions on Knowledge and Data Engineering* **2**, No. 1, 4–24 (March 1990).

A. J. Borr, "Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing," *Proceedings of the 7th International Conference on Very Large Databases* (1981), pp. 155–165.

DBC/1012 Database Computer System Manual Release 2.0, Document No. C10-0001-02, Teradata Corp. (later acquired by NCR) (November 1985).

C. Mohan and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine Granularity Locking in a Shared Disks Transaction Environment," *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain (September 1991), pp. 193–207.

C. Mohan, I. Narang, and S. Silen, "Solutions to Hot Spot Problems in a Shared Disks Transaction Environment," *Proceedings of the 4th International Workshop on High Performance Transaction Systems*, Asilomar, CA (September 1991).

Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex, Technical Report, GC28-1208-00, IBM Corporation (April 1994); available through IBM branch offices.

Accepted for publication January 15, 1997.

Gary M. King *IBM S/390 Division, 522 South Road, Poughkeepsie, New York 12601 (electronic mail: garyking@vnet.ibm.com).* Mr. King is a Senior Technical Staff Member of the S/390 division, consulting on all aspects of system performance. He joined IBM in 1974 and has been involved in the design and evaluation of the system resource managers, most notably in the area of storage management. For the past seven years his efforts have focused on clustered system performance, particularly the S/390 Parallel Sysplex and its exploiters. He has received six Outstanding Technical Achievement and Outstanding Innovation Awards in a variety of areas, including storage management, data compression, and performance analysis. Mr. King received a B.S. in mathematics from the University at Albany, State University of New York, in 1972 and an M.S. in computer science from the Pennsylvania State University in 1974.

Daniel M. Dias *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: dias@watson.ibm.com).* Dr. Dias received the B. Tech. degree from the Indian Institute of Technology and the M.S. and Ph.D. degrees from Rice University, all in electrical engineering. He currently manages the Parallel Commercial Systems group at the Research Center, which performs exploratory systems architecture, design, and analysis, with a focus on reducing these ideas to working prototypes and products. His current research includes scalable and highly available information servers including Web, video, and other servers for network-centric computing, Java™ collaboratory, Internet commerce, parallel transaction and query processing, highly available clustered systems, and performance analysis. Dr. Dias has published more than 100 papers. He has won two best paper awards, IBM Outstanding Innovation and Technical Achievement Awards, five Invention Achievement Awards, and Research Division Awards. He holds 13 U.S. patents, with seven additional patents pending.

Philip S. Yu *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: psyu@watson.ibm.com).* Dr. Yu has been with IBM Research since 1978. Currently he is manager of the Software Tools and Techniques group. One focus of the project is to develop algorithms and tools for Internet applications, such as a Web usage mining tool. His current research interests include database systems, data mining, multimedia systems, parallel and distributed processing, disk arrays, computer architecture, performance modeling, and workload analysis. Dr. Yu received the B.S. degree in E.E. from National Taiwan University in 1972, the M.S. and Ph.D. degrees in E.E. from Stanford University, in 1976 and 1978, respectively, and the M.B.A. degree from New York University in 1982. He has published more than 200 papers, and he holds or has applied for 36 U.S. patents. He is a Fellow of the ACM and the IEEE and was an editor of *IEEE Transactions on Knowledge and Data Engineering*. Dr. Yu has received a number of honors including Best Paper Award, two IBM Outstanding Innovation Awards, an Outstanding Technical Achievement Award, a Research Division Award, and 17 Invention Achievement Awards.

Reprint Order No. G321-5642.