The z990 first error data capture concept

S. Koerner
R. Bawidamann
W. Fischer
U. Helmich
D. Klodt
B. K. Tolan
P. Wojciak

Superior availability is one of the outstanding features of modern zSeries® machines, among the most highly rated of any existing computer platforms in this reqard. Many features contribute to this characteristic, some in hardware, some in software. This paper describes the first error data capture (FEDC) concept in the zSeries 990. The concept is used for both zSeries integration efficiency and its ability to gain field data for problem determination in the user environment. FEDC is not a single function, but part of all internal software (firmware) in the z990. This paper explains the overall concept and implementation details of the various internal functional layers (subsystems).

Introduction: z990 FEDC concept

The first error data capture (FEDC) concept is a feature of the z990 series of computers that significantly contributes to its high availability characteristics. The basic concept is that at the occurrence of any failure in the system, FEDC gathers and saves enough data for later analysis to assist in creating a solution. Technically, FEDC involves all firmware layers in the z990 system. The backbone functionality is similar for all subsystems. However, the function set of each subsystem is tailored to its needs. The benefits of the concept are twofold.

The first advantage occurs during the system bringup and integration phase. The efficiency of system integration is directly related to whether the problems can be solved with the data already available—or whether a re-creation on the test floor is necessary. A good FEDC implementation would, in most cases, deliver a sufficient set of data for developers to work with and solve the problem. During the development of the z990 program, the FEDC functionality delivered the correct data on the first attempt in 70% of all test-floor-found problems. This reduced the system integration time of a zSeries* system significantly.

The second effect is in the field. Here, it is of importance to get the right set of data when the problem occurs, since we cannot expect the customer to allow the problem to be re-created for the purpose of additional data collection.

In the following sections, the z990 FEDC technical implementation is described, as are the field behavior and the experiences gained during system integration. In all of the events that lead to insufficient data, the cases are examined in detail to determine the root cause and whether improvements to the FEDC implementation have to be made.

z990 system components relevant to FEDC

The z990 system structure supported by the FEDC concept comprises the host firmware—I/O subsystem (IOSS), processing unit subsystem (PSS), and logical partitioning (LPAR)—and the corresponding I/O adapter firmware (IOPs), together with the power, service, and control network (PSCN) controller and the support element (SE) (**Figure 1**). The IOSS firmware handles all I/O work on the host, the PSS firmware is part of the z/Architecture* and supports the instruction execution. The PSCN and SE belong to the control structure of the system [1–3].

All subsystems send their logs, traces, and dumps to the SE, where they are permanently stored. Collecting and storing the data is done by the use of a common FEDC infrastructure. In the following sections, the specific implementation and features of the z990 FEDC subsystems are described.

Support element

The SE is a system management console implemented on a notebook computer (IBM ThinkPad*). Two of these notebooks are installed within a z990 machine. Both keep their dynamic data synchronized at all times. The second SE is there for redundancy; it dynamically takes over the

0018-8646/04/$5.00 © 2004 IBM

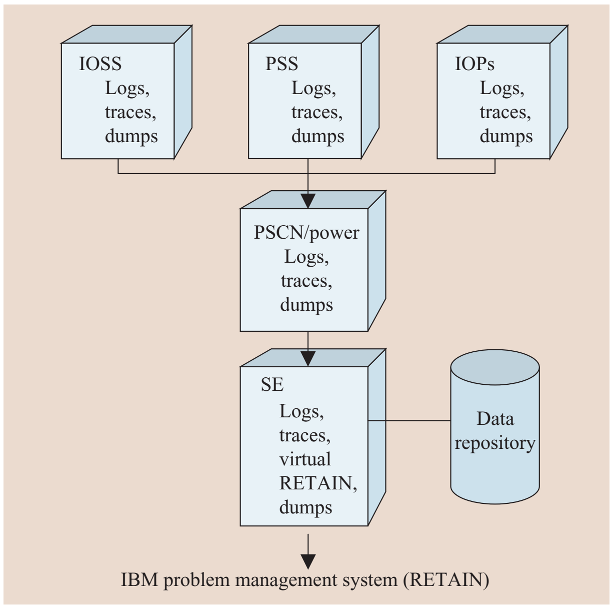

Figure 1

The first error data capture (FEDC) subsystem structure in the z990. (IOSS = I/O subsystem; PSS = processing unit subsystem; IOP = I/O processor; PSCN = power, service, and control network; SE = service element.)

workload from the first one in case of a problem. The fact that each SE has its own hard disk that can store large amounts of data is an important prerequisite for effective FEDC and a great advantage. As the destination of all FEDC data, the SE features an extensive set of FEDC management functions, described below.

Background

The reliability of the zSeries systems has improved over time [1, 4]. Through this period, the reliability of firmware components remained relatively stable, but, due to the increased reliability of the hardware, firmware failures represented an increasingly larger percentage of overall system failures in the field. Additionally, due to the increasing complexity of the firmware components, the cost of thoroughly testing the firmware was also increasing. A high percentage of test resources was being spent on recreating failures in order to collect sufficient failure data for the developer to solve the problem. Additionally, for machines installed in field environments, it was not practical to request time on a customer's system to re-create a failure, even if insufficient data was captured at problem occurrence.

Although FEDC has been part of zSeries products for many years, until recently there was no coordinated procedure for how this data was collected and transmitted to IBM. The common point was transmitting the system log file to IBM on problem call home. The standards were that each failure-describing log entry contain a system reference code (SRC) that indicated the source of the failure, and that the data include information on the suspected field-replaceable unit (FRU) (i.e., the failing hardware part that must be replaced). What other information was captured and stored in the log file was left up to the individual developer.

It was recognized that this was not the most effective way of dealing with an issue critical to the successful debugging of failures on the system. Therefore, FEDC became an area with specific focus, and the FEDC infrastructure was born, with a recognition that it had to meet certain criteria:

- It had to be centralized so that all components could use the same set of tools.
- It had to be easy to tailor it to each failure symptom.
- It had to be easily modifiable so that information learned in the testing phase could be quickly incorporated into the process.
- It had to provide a basic level of FEDC for all failures.

The infrastructure had to be reliable and always had to deliver the available data without introducing new failures into the system. It had to be efficient in that it could not introduce timing delays into the failure recovery path, since this could lead to additional problems. It also had to collect the data as close to the time of the initial failure as possible and provide a mechanism to store the data until it could be successfully offloaded.

System log file

From the beginning, the system log was (and in many cases still is) the cornerstone of the FEDC infrastructure. The three main purposes of the system log are

- Journaling of all significant system events.
- Problem data repository for automatic problem analysis (auto PA).
- FEDC.

The auto PA feature and FEDC have a similar background. The first is there to attempt, insofar as possible, to automatically determine and recover a problem, the latter to gather additional data whenever auto PA fails and the developer must analyze the failure to determine the exact cause of the problem.

The type and amount of data gathered for both purposes are very different. Auto PA requires a predefined data structure upon which it can apply its rules set and analyze, correlate, and prioritize a set of problem-related information. FEDC data has to be much more exhaustive and, in general, is a much larger set of data than that needed for auto PA. The system log file for the z990 has a 30-MB capacity. One log entry can have a size up to 128 Kb. While this may seem very generous, it proves very valuable by providing a history of system events that may well reach back several months.

The FEDC infrastructure provides tailored interfaces for each component. Most of the FEDC data is received on the SE together with an event for the system log. Components that are external to the SE can send FEDC data via the power, service, and control network (PSCN).

For the most part, these external components send only raw data to the SE, where that information is structured and split up into the part that will be posted to the system log and the part that will be stored in individual files on the SE hard disk. This is done to offload as much of the workload as possible from the host to avoid negative impact on overall system performance.

When posting an event in the system log, the component can decide whether or not that event must be handled by auto PA. All events forwarded to auto PA must bring along an SRC that describes as exactly as possible the type of error. For hardware failures, this SRC is supplied with a reference code extension that provides information on the possibly affected hardware FRU(s). These reference codes and the structure of the reference code extension are maintained centrally for all z990 systems to guarantee uniqueness across the z990 series. The IBM support organization can link each SRC to its owning developer and route the FEDC data to that person for analysis.

Log file housekeeping

For a pure journaling log, a simple wraparound (when the buffer is full, overwrite the oldest entries) would be sufficient. However, since the system log is used for problem analysis, a more sophisticated housekeeping approach is needed; i.e., events directly related to problems must be kept longer in the log file than other events. A pruning algorithm was implemented that prioritized the log events according to their long-term importance. This allows PA to correlate subsequent service actions and avoid redundant *call homes* for the same problem. The pruning algorithm kicks in whenever a log event would increase the size of the system log above its allowed maximum. The pruning reduces the size of the log to, at most, 50% of that maximum. Previous back versions of the system log are kept on the SE; these are not sent home but can be collected manually in case the data in the latest system log is not sufficient for analysis. However, this is relatively rare.

SE tracing

One important tool for debugging firmware problems is the tracing of program steps. On the SE, this is done through a central wraparound buffer in memory. There is a common routine that any C or C++ program can call to create a trace entry. The SE coding conventions require tracing of the entry and exit of every major function. Additionally, a trace can be taken at any critical point in the code that the developer deems would be helpful in debugging a code problem. Since the memory buffer for tracing is a shared resource used by all SE applications, care has to be taken to ensure that the correct level of tracing is done without adding spurious entries that would cause more pertinent entries to be wrapped out. For example, a trace entry within a loop would be undesirable.

Whenever a log is taken, the developer has the option of including the SE trace information with the log. In addition to being able to retrieve a subset of the entire buffer, a subset of the trace entries created only by the calling process and a subset created only within the calling process and thread can be requested.

In addition to the system trace buffer, the SE provides for a separate private trace buffer. This private buffer is reserved for long-running applications, such as power-on reset. Depending on the amount of tracing done, the system trace buffer typically wraps in a matter of seconds, while the debugging of some functions requires data on their program flow from the beginning. Instead of filling the system log with snapshots of data that are not needed for the successful passes through the routines, the developer can use the private trace buffer and, through the utilities described below, save this longer-running process trace on error indications. The total sum of all independent trace buffers amounts to 36. The time to create a log entry is in the range of microseconds.

Table-driven data collection

In the early years of the zSeries, the system log was the only data repository for FEDC. Over time, however, the size of FEDC data increased with the complexity of the systems to a point at which the amount of data needed grew beyond a realistic log file size. Algorithms were then put in place that could detach the FEDC data from larger log entries and write them to separate files on the SE hard disk. The log entry would keep only a reference to that detached data file.

In other cases, data is needed that was not part of the event, but would already be available on the SE hard disk, e.g., machine configuration data or system status information. A key enhancement was the creation of a table-driven approach that enabled attaching the necessary files whenever needed; for certain groups of log events, files can be collected and added to the FEDC data. Also from these tables it is possible to invoke SE functions that can generate this data from dynamic system information at the time the log is written. One prominent example is the SE trace collection mentioned above.

The table-driven approach is very flexible and makes quick changes to the FEDC dataset possible without having to alter and recompile actual SE code. This proved especially useful during the early system development phase when these FEDC tables had to be updated and enhanced rather frequently.

Whenever a log event is posted to the system log, the FEDC tables are scanned for related entries. Functions are called and files are scheduled for collection. If the automatic problem analysis determines that the event should be called home, it collects the FEDC files for all events related to the problem.

Another important enhancement to the FEDC infrastructure was the creation of a *virtual RETAIN*. (RETAIN—which stands for *remote technical assistance information network*—is the IBM service organization problem management system. It is a repository used for long-term retention of FEDC information from IBM Enterprise-class machines.) Instead of waiting to collect the files until a connection is made to RETAIN, these files are compressed and put into a virtual RETAIN directory. Once the connection to RETAIN is made, the files from the virtual RETAIN directory are transmitted to the IBM service organization.

Because of limited bandwidth, there is a maximum allowed data size that can be transferred to RETAIN via modem. If the total size of FEDC data exceeds that maximum, only a defined subset is transmitted (the system log is always part of that subset), and the remainder of the data resides in the virtual RETAIN directory to be manually collected if necessary. The virtual RETAIN directories are deleted only after a problem is successfully serviced and closed.

Display tools for FEDC

Along with the FEDC infrastructure, appropriate tools had to be developed to ease collecting and attaching the failure data for the test floor and for viewing the data once it is collected. The most important tool for FEDC display is the system log displayer, because much of the FEDC information is still included there. The log displayer on the SE features a plug-in architecture for data formatting routines. These formatters are invoked on the basis of data format information stored within the log entries.

Besides the main system log displayer, there are several other viewers that expand the complex data structures contained in the log and trace files into human-readable format. The correct interpretation and user-friendly presentation of that data is crucial for debugging and development efficiency.

Cage controller and FEDC

The cage controller (CC) is an embedded controller based on an IBM PowerPC* processor. The current zSeries system z990 can have up to 16 CCs. Each controller has 51 MB of random-access memory (RAM) and two Ethernet interfaces. The controllers are connected with the SE via redundant internal Ethernet networks. The CCs themselves are also redundant. Each CC has a counterpart that waits in a warm standby mode to take over the job of the other CC if it fails or reboots for another reason (e.g., a code update). There are up to eight units in the z990 with such a pair of CCs [power subsystem, up to four central electronic complex (CEC) units, and up to three I/O units].

Other than a small flash memory that holds the basic I/O system (BIOS), the CCs have no local hard disk or flash memory or any other kind of persistent storage. Therefore, to store FEDC data from the CC on the SE, a working Ethernet is needed. This arrangement is not a problem for error scenarios in which the operating system is still fully functional, but in some cases—for example, if a parity check (memory, processor cache) fails, the CC memory might contain random data. If the CC tries to send something to the SE, it might hang or fail again (which opens the possibility of an endless loop) or it might do something completely unexpected, such as switching off the power of an I/O card or the CEC. Thus, the CC should not take any action except for a reboot. The most important thing is to keep the system running. The fact that a CC is rebooting is not important, and probably not noticeable, to the user, so the reboot is the safest action.

To obtain the FEDC data in this instance, a brute force method was chosen: The BIOS of the CC recognizes the error reboot and sends the complete CC memory content to the SE. The BIOS does a clean start "from scratch" (i.e., after a processor reset) and uses some reserved memory to avoid overwriting any memory used by the firmware. This way, the BIOS can safely use the internal network to send the dump to the SE. With this memory dump, all available data that was on the CC is saved, so all possible FEDC data can be analyzed in depth. A CC dump is the final "safety net" for firmware errors. The CC itself just does the dumping: It sends its memory content to the SE. After the CC dump is received on the SE, an entry is made in the system log that refers to the file with the CC dump. PA later puts the dump into virtual RETAIN.

A postmortem analyzer analyzes the dump and is able to extract the status of the CC operating system and the applications running on the CC at reboot time. The trace buffers and the logs in the log queue can be extracted as well.

Tracing on the CC resembles a flight recorder: Events can be recorded in multiple trace buffers. The major difference with tracing on the SE is the larger number of trace buffers. This is to avoid the traces of one component overwriting the traces of others. Another feature of CC tracing is optimized performance, due to the slower central processing unit (CPU) of the CC.

Traces on the SE are stored in a textual format. Creating this text takes considerable CPU time, and storing the strings consumes a lot of space. Usually a trace consists of static and dynamic data, e.g., a constant string in which some numbers are filled in at runtime. The CC tracing stores only the dynamic data (the numbers) and an identification for the constant part (the string). This way, one trace event is very small (24 bytes plus up to 36 bytes of dynamic data) and fast (around four microseconds on a 200-MHz PowerPC 405*). This reduces both time and space by 80%. A postprocessor on the SE or in the trouble ticket system translates the dynamic data and the identification into a human-readable text.

Some components such as device drivers run very frequently, while others have long delays between actions. If a component has to wait some time before it can assume and report an error (e.g., networking code that works with timeouts), another component could overwrite the traces of the reporting component and the FEDC information would be lost. Therefore, the CC provides several trace buffers, one or even two for each component. This has proven to be very valuable.

The CC can put a log into the system log on the SE. The logs are queued on the CC and transferred to the SE via the internal network. Some or all of the CC trace buffers can be attached to a log. These traces are stored on the SE as separate files and included in virtual RETAIN by PA (see above).

Logs are usually used to report broken or misbehaving hardware so that repair actions can be triggered or to report software (i.e., firmware) errors. Of course, the traces are important in the latter case for finding the errors, but even in the former case, they help in understanding what went wrong with the hardware.

CEC firmware FEDC techniques

The CEC firmware also incorporates FEDC techniques—such as tracing, logging, and memory dumps—as important design elements; these can be summarized as follows.

Tracing

A trace macro writes 8×64 bits of data to a dedicated wraparound buffer per processing unit (PU). The first two doublewords hold a time stamp, a PU identifier, a trace identifier, and the so-called *word one* trace data. The following 6×64 bits can be used for payload that the code designer specifies when calling the macro. Via an SE panel, it is possible to limit tracing to certain code areas and to one of four trace levels ranging from flow information to error information. This way, one can obtain just the information needed to analyze a problem without filling the buffer too rapidly and overwriting older trace entries. Tracing is usually used to track the code flow and save some limited critical data during code debugging and system bringup. It is turned off during normal customer operation because of its potentially large impact on system performance. Tracing is turned on for a limited time and for a limited set of trace identifications only if a system experiences problems during normal operation that make an error recovery run necessary. Traces collected by recovery are automatically transferred to the SE via a log entry added to the system log file. In all other cases, writing the buffer contents to the SE must be triggered by a manual action via an SE panel.

Logging

This mechanism stores larger amounts of data to permanent storage. Any CEC code can fill a buffer with data that is to be placed in the system log. This can be error information or state information. The buffer content must be transferred to the SE because the CEC has no permanent storage for its exclusive use. The transfer is managed by a call to the log handler facility and realized by the internal data transfer protocol.

Memory dumps (CECDUMP)

If the system encounters problems that do not allow further execution of the CEC firmware, a dump of the entire hardware system area (HSA) memory is written to the SE disk. This happens only in rare cases in which recovery is not able to handle the failure. It holds all system state information at the moment the CECDUMP was written.

These techniques ensure that all data necessary for debugging a problem is available or provides enough information to re-create the failure at an IBM site so that more data can be gathered by performance-impacting or highly invasive techniques (e.g., a system checkstop on a trace match).

CEC environment

The implementation of the above techniques must take into account some constraints imposed by the CEC environment. During normal user operation, the CEC provides computing power for all mission-critical applications and drives the I/O requests to the channel subsystem. The CEC firmware keeps track of the system state in its private control block data structure, located in HSA. This state information is shared by all PUs in the system and by the channel subsystem. Access to it must be very fast in order to provide the highest possible performance. Any benefit that might be gained by data

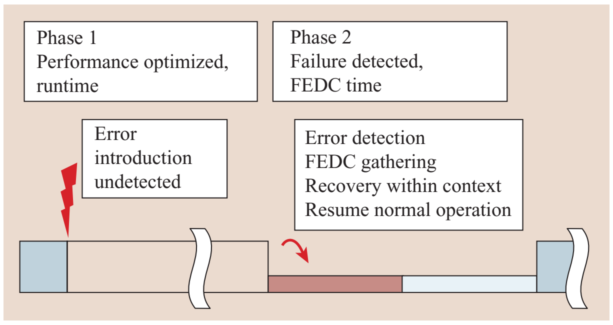

Figure 2

FEDC phases.

collection during normal operation (*flight recorder*; see the section below on new FEDC in CEC firmware) must be balanced against its impact on performance.

FEDC and recovery

System operation is different if a problem is detected. This places the system on the recovery path and should happen only in rare cases. Performance is of secondary importance to the goal of recovering from the error and gathering all relevant data related to the error. This approach ties FEDC in the CEC environment more closely to recovery than is the case in most subsystems, making it very important to detect errors as early as possible.

One of the first steps taken by recovery is to activate the internal host firmware trace facility that records the code flow and important symbols used by the present recovery run. Before any attempt is made to recover the error, some FEDC data is gathered and written to a log file. Depending on the type of the failure (hardware failure or code bug), hardware registers are scanned and/or internal state information is taken from the HSA.

After FEDC collection, recovery decides on an action plan and takes all necessary steps to recover from the error. During this process, the concept of secondary error data capture (SEDC) is introduced to complement the FEDC data. Each action performed by recovery writes data to a log that reflects the results of this action.

Several code functions that are required during normal operation also provide a version that must be used during a recovery run. This serves two purposes. First, the recovery version does not have to obey the high-performance requirements of the critical path. It is designed to perform more checking on all handled data, log this data for FEDC if problems are found, and provide error escalation mechanisms for all problems that can be conceived of during the code design. The second benefit is that this is a different code path that might work, even if the normal version fails because of a microcode error.

Failure types

If one does not take into account user errors, there are basically two different types of failures: failures resulting from hardware malfunction and failures resulting from coding errors. Hardware errors usually cause a recovery run that resets or fences the component from the system. FEDC and SEDC data are used to automatically identify the failing FRU and trigger its replacement by a call home. These failures cause temporary system degradation, but usually do not cause a catastrophic customer disruption.

Coding errors are much more critical because the type of failure and its consequences are not predictable. Recovery attempts to reset all involved hardware and code components and checks to see if normal operation can be restored. If this is not successful, the system is halted and a CECDUMP is triggered, as the most complete set of FEDC data that can be obtained.

Learning from previous errors

As a result of the restrictions that apply to FEDC data collection in a field environment, it is important to develop a feeling for error scenarios that are likely to occur. These scenarios can be translated into test cases to regression-test all new code using a CEC firmware code simulator [5]. The test case is used during unit test or as an error-inject scenario during bringup on the real hardware. Test cases based on FEDC data from previous error scenarios help identify new problems at an early stage of development. This makes FEDC a very important tool long before a system is shipped to the customer. **Figure 2** shows the FEDC phases.

New FEDC in CEC firmware

For some long-term problems, the scope of the system log has been found to be insufficient for tracking the cause of a problem to its origin. A problem may have been introduced days or weeks ago in an activity which, at that point, went errorless.

Traces are still the most often used method of capturing sequences of events. It is a debugging feature, but it becomes a FEDC feature when it is always available during runtime, be it during bringup or in the field. Having a single trace buffer available for use by all CEC internal firmware was considered insufficient, because particular internal functions have different frequencies and time spans.

These considerations led to a collection of FEDC features for CEC firmware code called *CEC debug data manager* (CDDM). CDDM consists of three components:

1. *Flight recorders* (FLRs), which are small internal trace buffers for each requesting function named. There are currently ten functions with separate buffers. Each PU has a separate buffer to avoid false sharing between PUs. The synchronized time of day on all PUs is the same. This time has a resolution down to about 4 ns and is used to generate a timestamp for every FLR entry. Thus, a later merge can be performed on all buffers, and the entry sequence can be reconstructed over the PU and buffer boundaries. The buffers wrap according to their size and use. The FLRs remain in memory. Every entry holds eight doublewords ($8 \times 8 = 64$ bytes) of binary data. To obtain a longer history, one of the following methods may be used.
2. *Text logs* (TLGs), or inline generated text. The logging feature is of great use during debugging, but its final use is limited conceptually to less than one text log per second. Bursts are supported. The text logs are sent to the SE to be saved in a separate subdirectory. They do not have the strong relationship to SRCs that the system log has. It is a repository for state/transition information useful for debugging CEC firmware situations and complements the error information stored in the system log. TLGs are written at useful debugging points, not necessarily error situations, but the resulting text log is waiting to be picked up if the data becomes important for a particular error situation.
3. *Dumps* (DMPs) of memory data into selectable files on the SE in that same subdirectory. In one such DMP, up to 512 KB of binary data may be saved. With this feature, the state of control variables and structures can be saved, as can the contents of one FLR buffer or an entire group of FLR buffers for all PUs. On the SE, the DMP data is appended to the selected file. If the size of the resulting file is above an individual limit, it is compressed, and a new file is started. The limits are controlled in a control file on the SE. Essentially, this ensures that CDDM data cannot overwhelm the hard disk capacity on the SE. Finally, there is the chance of loss of debugging data, but, because different applications use different files, every application can fit its own debugging requirements.

CEC firmware development now has a collection of FEDC features at hand for those situations in which an error has not yet occurred, but valuable data must be saved in order to facilitate analysis of problems that are not easy to debug when only the system log is available. The system log is used when an error is detected and an SRC is generated. Now, according to that SRC, further debug data may be appended to the system log. This data may very well contain portions of the saved CDDM data in the separate subdirectory on the SE.

CECDUMP improvement for CEC firmware

The already very valuable CECDUMP as a postmortem dump of the z990 memory state was a significant improvement. On former zSeries machines, only the memory contents were dumped to the SE; the architected state of the PUs remained unknown in the CECDUMP. This was a significant deficiency for some debugging situations. On the z990, a new mechanism is used to trigger all PUs to save their current state and their internal hardware trace data to memory, providing much more complete postmortem FEDC information. The CECDUMP is now invoked through a special machine check that spreads synchronously to all PUs, in both single-book and multibook z990 systems. Even hang situations are disrupted, and thus, the most complete system view including all PU states plus memory is generated. This also allows analysis of most of the problems between two or more PUs, which can be very hard to debug.

This improvement is amplified by a new browser tool to examine CECDUMPs and help in understanding the data content in a short time. **Figure 3** shows the general CEC system structure in relationship to FEDC. SCD and SCC together make up the L2 caches of the system.

Effect of virtual RETAIN on problem reproduction requirements

Each zSeries system has the ability to connect to RETAIN via a modem or high-speed network. Some limitations of this repository, such as the bandwidth of the upload link (phone line) and the maximum size of each file being uploaded, have been mentioned. In response to this, a longer-term repository, virtual RETAIN, resides on the SE hard drive and has been expanded for z990. Virtual RETAIN keeps the last 100 incidents' worth of FEDC information, regardless of size. A common compression tool reduces the size of each error dataset stored in virtual RETAIN. (Data sent to RETAIN over the modem is also compressed before transmission.)

Prior to the introduction of virtual RETAIN, there were instances in which FEDC information kept on the SE was overlaid by new data. This caused the data from the original failure to be lost, making diagnosis all but impossible. Another problem was that, for a given failure, many different files may have to be collected. In some cases, collection of this data was not automatic and would require several trips by an IBM Customer Engineer to the machine at the customer's data center. These same difficulties were frequently encountered by the engineering test team. The FEDC information was overwritten, pruned by the automatic log management process, inadvertently erased by the user prior to a new test, or the user had collected only a partial set of FEDC files when documenting the problem. In all of these cases, the problem inevitably would have to be re-created before

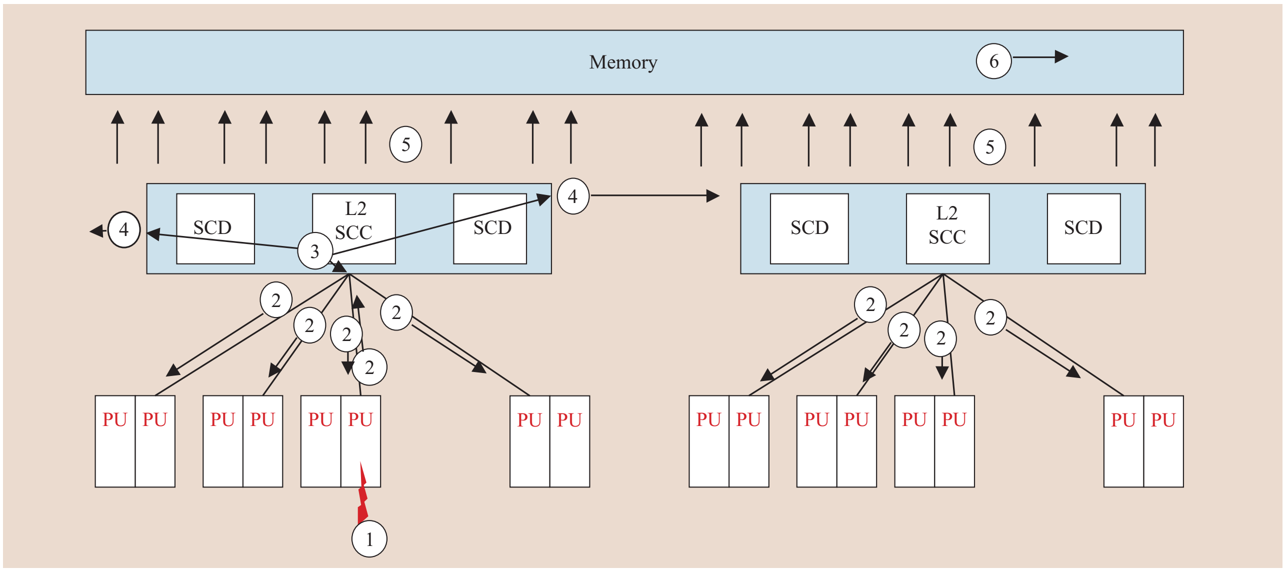

Figure 3

FEDC flow. 1. Firmware on any PU detects a fatal error. 2. Firmware signals a special machine check to the L2 ring. 3. The L2 control chip on one book starts a broadcast to all books in the system and its own PUs under full hardware control. 4. The broadcast travels on the interfaces. (On other books in the system, the broadcast reaches the PUs a few cycles delayed.) 5. All PUs write their PU state independently into memory. 6. The CECDUMP continues as usual to dump the data to the SE.

Table 1 Percentage of problems with missing minimum debugging data not attached and with missing minimum debugging data not available.

	9672 G05	*z900*	*z990*
Minimum debug data not attached to problem ticket	16.2	11.4	9.9
Minimum debug data not available	21.1	13.9	2.0

diagnosis could be completed, or in some cases, even be started.

With the current virtual RETAIN packaging scheme for FEDC, instances of overwritten data have been virtually eliminated. Also, there has been a marked reduction in instances in which the originator of a problem has not attached the minimum FEDC information. This, however, is really more a matter of educating problem originators on proper FEDC collection than anything to do with the code.

Table 1 shows an improvement trend with respect to percentage of problems written where the minimum FEDC is not collected by the problem originator. Virtual RETAIN was introduced after the 9672 G05 model introduction.

The log management process of the SE performs housekeeping tasks on logged data. The event log, in which much of the FEDC information is housed, was previously limited to 1.4 MB (the size of a floppy diskette), but it has been expanded to 10 MB on the z990. When the amount of logged information is about to exceed that limit, log management housekeeping performs two tasks: log compression and log pruning. Log pruning causes data to be discarded directly, contributing to lost FEDC information.

A look at the percentage of problems written in which the FEDC information was pruned out also shows a substantial improvement trend (Table 1). The larger event log is probably the single biggest contributor to this improvement.

zSeries machine FEDC effectiveness

Historically, on S/390* and zSeries servers, about 33% of the problems written (identified and logged) during engineering tests and from incidents in the field required additional data beyond that produced by FEDC to isolate and understand the root cause. This is termed the *more-data rate*, because that portion of problems written requires more data for resolution. As mentioned previously, insufficient data from a field machine can inhibit the ultimate resolution of a problem. Additionally, since a field machine is used 24 hours a day, seven days a week, there is rarely a chance to reproduce the problem on site. Thus, obtaining the failure data on first occurrence is paramount.

For the zSeries z990 machine, the early indications are that the overall more-data rate is about 25%. This means that instead of one in three problems requiring more data, the rate has been reduced to about one in four. Given the number of problems written during engineering tests for such a product, this translates to a significant savings in labor and machines. Further, because of the increased complexity of the z990 with respect to previous processor families, just maintaining more-data rates—let alone improving them—was a monumental challenge that required significant FEDC enhancements. **Table 2** reflects a comparison of the more-data rate for the 9672 G05, the z900, and the z990 families of machines per code subsystem.

Another measure can be drawn by looking into types of more-data requests. The person requesting additional data for a problem must select the reason for the more-data request. One of these reasons maps directly to insufficient FEDC. A comparison of the 9672 G05, z900, and z990 programs is shown in **Table 3**.

Another way to quantify more-data activity and improvements from one machine to the next is with the ratio

$$\frac{\textit{Total number of more-data requests}}{\textit{Number of problems with a more-data request}} = \textit{more-data requests per problem}.$$

This ratio is useful because higher numbers often reflect that the weaker FEDC of a subsystem is contributing to multiple requests for reproduction per problem; if the FEDC mechanisms are improving, this ratio should decrease. **Table 4** illustrates the improvement trend of the more-data requests per problem ratio. What may seem like small movements in the values represent thousands of hours of labor and machine time for the test organization, product engineering, and development.

Comparison of various metrics useful for analyzing the effects of FEDC on problem more-data rates demonstrates that those of the z990 system exceed those of prior zSeries machines. The improvements in FEDC for zSeries have evolved from one machine to the next. There is a direct correlation between better FEDC and lower problem more-data rates. In turn, this can be translated into real personnel and test machine time savings, representing significant test process productivity improvements.

Future direction and outlook

On the basis of experience gained during the last few zSeries programs and positive customer response, FEDC coverage will continue to increase by further analysis of the FEDC data gathered during the integration phase and in the field. Over time, this will lead to fewer more-data requests, which, in turn, can increase the integration efficiency and drive down costs. The tool suite that surrounds the FEDC functionality will be enhanced to make it easier and more error-free to use, and these changes will also include optimizing the human interface.

Table 2 Subsystem-based percentage of more-data requests. (Percentage of problems written by the Engineering Test and Product Engineering organizations during the initial testing and first three months after general availability that went into more-data status.)

Subsystem	*9672 G05*	*z900*	*z990*
Support element	28.2	31.7	25.0
Processor millicode	31.1	27.2	26.0
Processor i390	32.5	42.3	24.3
Channel	33.7	38.1	36.7
Power	19.8	12.9	16.9
PSCN	NA	32.6	20.9
LPAR	34.5	41.7	36.0
All hardware	23.7	22.2	26.4
SAK/PCX (Test software)	11.8	11.7	13.2
Totals (w/o SAK/PCX)	29.7	30.0	25.4

Table 3 Areas of more debug data requests due to insufficient FEDC (%).

Subsystem	*9672 G05*	*z900*	*z990*
Support element	31.3	32.7	25.4
Processor millicode	63.3	55.6	32.4
Processor i390	50.0	40.0	40.4
Channel	37.2	59.9	33.9
Power	19.2	9.1	9.5
PSCN	NA	27.4	21.0
LPAR	78.6	50.0	26.7
All hardware	47.4	51.4	25.5
SAK/PCX (test software)	62.5	42.2	35.7
Totals (w/o SAK/PCX)	43.6	43.1	30.2

Table 4 More-data per problem ratio comparison by subsystem.

Subsystem	*9672 G05*	*z900*	*z990*
Support element	1.57	1.59	1.44
Processor millicode	1.58	2.05	1.48
Processor i390	1.67	1.67	1.58
Channel	1.78	1.83	1.54
Power	1.37	1.34	1.32
PSCN	NA	1.54	1.47
LPAR	1.47	1.87	1.84
All hardware	1.36	1.68	1.39
SAK/PCX (test software)	1.6	1.25	1.32
Totals (w/o SAK/PCX)	1.60	1.63	1.50

As an integral part of the zSeries system, FEDC will influence eServer* series systems in the future.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. M. Mueller, L. C. Alves, W. Fischer, M. L. Fair, and I. Modi, "RAS Strategy for IBM S/390 G5 and G6," *IBM J. Res. & Dev.* **43,** No. 5/6, 875–888 (September/November 1999); see *http://www.research.ibm.com/journal/rd/435/mueller.pdf*.
2. D. J. Stigliani, Jr., T. E. Bubb, D. F. Casper, J. H. Chin, S. G. Glassen, J. M. Hoke, V. A. Minassian, J. H. Quick, and C. H. Whitehead, "IBM eServer z900 I/O Subsystem," *IBM J. Res. & Dev.* **46,** No. 4/5, 421–445 (July/September 2002); see *http://www.research.ibm.com/journal/rd/464/stigliani.pdf*.
3. F. Baitinger, H. Elfering, G. Kreissig, D. Metz, J. Saalmueller, and F. Scholz, "System Control Structure of the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 523–535 (July/September 2002); see *http://www.research.ibm.com/journal/rd/464/baitinger.pdf*.
4. L. C. Alves, M. L. Fair, P. J. Meaney, C. L. Chen, W. J. Clarke, G. C. Wellwood, N. E. Weber, I. N. Modi, B. K. Tolan, and F. Freier, "RAS Design for the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 503–521 (July/September 2002); see *http://www.research.ibm.com/journal/rd/464/alves.pdf*.
5. J. von Buttlar, H. Böhm, R. Ernst, A. Horsch, A. Kohler, H. Schein, M. Stetter, and K. Theurich, "z/CECSIM: An Efficient and Comprehensive Microcode Simulator for the IBM eServer z900," *IBM J. Res. & Dev.* **46,** No. 4/5, 607–615 (July/September 2002); see *http://www.research.ibm.com/journal/rd/464/vonbuttlar.pdf*.

Received September 22, 2003; accepted for publication November 25, 2003; Internet publication April 27, 2004

Stefan Koerner *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (koerners@de.ibm.com).* Mr. Koerner is a Senior Technical Staff Member in the IBM eServer z990 Hardware Development Group in the Boeblingen laboratories. He joined IBM in Boeblingen in 1981 after receiving an M.S. degree in electrical engineering from the Technical University of Furtwangen. He has held a number of positions in logic design, firmware development, and verification. He holds three patents and received an IBM Outstanding Innovation Award in 2001. Mr. Koerner is currently the technical leader for firmware verification in the IBM Systems and Technology Group.

Rainer Bawidamann *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (rainer.bawidamann@de.ibm.com).* Mr. Bawidamann received a diploma in computer science in 2000 from the University of Ulm, Germany. He joined the zSeries Firmware Development team of the IBM Systems Group in Boeblingen, Germany, after some work on network management tools. Shortly thereafter, he assumed the role of the FEDC focal point of the PSCN subsystem, a position he still holds. Mr. Bawidamann is currently focusing on FEDC for a common embedded system controller for future IBM eServers.

Wolfgang Fischer *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (fischerw@de.ibm.com).* Mr. Fischer studied electrical engineering at the University of Siegen and received his Dipl.-Ing. degree in 1985. He joined IBM in January 1986 in the Hardware Development Department for S/370 in Boeblingen. He has a patent and has held various positions in hardware and microcode development. Mr. Fischer is currently in microcode development, with RAS and FEDC responsibilities.

Ulrich Helmich *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (helmich@de.ibm.com).* Dr. Helmich received an M.S. degree in physics from the University of Sussex in 1995 and a Dipl.-Phys. degree in 1997 from the University of Tuebingen, Germany. After working as a research assistant for the Max Planck Society from 1996 to 1998 and as an IT consultant for Integrata from 1998 to 2000, he joined the IBM Development Laboratories in Boeblingen, Germany. Dr. Helmich's current responsibilities are zSeries I/O microcode development, including FEDC and error-recovery code.

Daniel Klodt *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (dklodt@de.ibm.com).* Mr. Klodt studied mathematics and computer science at the Technical University of Darmstadt, Germany. He received a Dipl.-Math. degree in 1995 and joined IBM Global Services in December that same year, initially working on new strategies in spare parts logistics for the EMEA central region. He has held assignments at the EMEA IBM Logistics Center in Amsterdam and at the IBM Support Center in Mainz, Germany. In November 2000 he joined the zSeries

Firmware Development team, where he has been developing code in the error handling and FEDC area. Mr. Klodt was assigned the FEDC focal point position in July 2003 for the next-generation zSeries server.

Brian K. Tolan *IBM Systems and Technology Group, 1701 North Street, Endicott, New York 13760 (tolan@us.ibm.com).* Mr. Tolan is a Senior Engineer working in the eServer platform architecture and design team responsible for serviceability issues across eServer. He graduated from Columbia University in 1981 with a B.S. degree in electrical engineering. He joined IBM Endicott that same year, working in the First Level Packaging Test Development Group. Mr. Tolan has worked in systems serviceability engineering development since 1988.

Paul Wojciak *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (wojciak@us.ibm.com).* Mr. Wojciak joined the IBM Engineering Systems Test organization in 1988 after receiving a B.S. degree in electrical engineering from Clarkson University. He has worked on more than a dozen different systems developed by the S/390 and zSeries design teams, with an emphasis on system hardware and firmware recovery testing and techniques. He has received several awards for his leadership and technical contributions to those product lines throughout that period. Mr. Wojciak, as a Senior Engineer, continues to enjoy devising new and creative ways of making the zSeries systems fail, all in the name of fault-tolerant computing.