Appendices

11. Appendix A - Glossary of Terms

BMC Firmware

Firmware running on the BMC.

BMC-based error logging

Error logging using firmware running on the BMC, and not involving the operating system or platform firmware (may use management core firmware)

Containment

Error containment prevents corrupted data from being used by applications on the system and prevents corrupted data from being transmitted over I/O (e.g stored to disk or sent over a network). In systems that had a bus-based architecture, resets could implement error containment since the reset was sent to all of the devices on the bus at once. Modern systems do not use buses and the lack of a reset wire means there it takes a potentially long time for resets to propagate. Modern systems therefore transmit containment signals via their interconnect fabric (e.g. UPI or HyperTransport). See poison and viral for example containment signals. Stopping an OS with an interrupt is not sufficient for containment. DMA may continue after the OS is interrupted and can send corrupted data to I/O.

CSP - Cloud Service Provider

The term Cloud Service Provider is intended to be a broad term for any organization that runs a large fleet of machines and provides services using that fleet. The use cases in this document are broad and are not strictly limited to organizations that provide cloud services. This proposal also supports use cases that include server operators that are not cloud providers.

Decoding

Decoding is the first step in analyzing an error log. The binary error log is parsed and broken into fields to which rules can be applied. A decoded error log is often in a human-readable, but machine parsable format such as JSON. Decoded error logs allow server fleet operators to analyze the errors and use ML tools to look for signals, for humans to consume the logs and to build dashboards for tracking failures in their fleet.

EL3 (Exception Level 3)

EL3 is the highest level of execution state for ARM CPUs, called Secure Monitor. It is a privileged mode that can only be entered by firmware and has some similarity to System Management Mode (SMM) on x86. It is used for managing security and other low-level features of the CPU.

Error Logging

Error logging is used very broadly to refer to collecting error data about a failure. Error logging is the process of latching information about an error in the chipset, collecting the error data in software and sending the data to tools where it can be analyzed. When software reads error logs, it often generates a binary structure such as a Common Platform Error Record(CPER) log that can be passed to other, higher level software for analysis.

Error Handling

Error handling is where software logs errors, analyzes them and determines how to minimize their impact. Error handling software can take actions like driving RAS features, telling the operating system to stop using resources and sending messages to get the server repaired.

Fault

A fault is something that causes hardware to malfunction. If the hardware detects the fault, it can log errors that aid in diagnosing the fault and servicing the hardware.

FIT(Failures in Time)

The Failures In Time (FIT) rate of a device is the number of failures expected in one billion (10^9) device-hours of operation. (E.g. 1000 devices for 1 million hours, or 1 million devices for 1000 hours each, or some other combination.)

FRU (Field Replaceable Unit)

A FRU is a part that can be changed in the data center in order to repair a broken system. Components like DIMMs or PCIe cards are FRUs. A DRAM soldered to a mainboard would not be considered a FRU; in that case, the mainboard would be considered a FRU.

FW-First error logging

Error logging using runtime firmware (e.g SMM). Firmware first error logging allows a server vendor to see all error logs from hardware, even when the vendor does not have a daemon or driver on the operating system. With

firmware-first error logging, the firmware can log hardware errors, determine whether it can address the error or whether it needs to expose the error to the operating system. If the firmware can handle the error, it can clear the error and the OS will not see it.

Hardware component

A component designed by a vendor. Examples of hardware components include CPU chips/SoCs, DRAMs and PCIe devices.

Hermetic error logs

Upon detecting an error, hardware should log all the relevant information and allow 'error monitoring agent' to read entire logs prior to updating the logs with any new error event. The error logs should be complete so that they can be analyzed without needing to gather additional data. Hardware should not taint the error logs by overwriting some part of logs before previous logs are read by the 'error monitoring agent'. This is important where error detection rate could be higher than the rate at which 'error monitoring agent' could read the logs, e.g., main memory corrected error logs.

Management core firmware

Firmware running on management cores on the processor.

Management infrastructure

Software, typically deployed by the server fleet operator, that manages the fleet operator's hardware. This software manages workloads, monitors the health of the fleet, drives repair actions and manages the flow of data. This software enables server fleet operators to automate the management of their fleet.

Node

A platform designed by a vendor integrating multiple components that run an operating system. These are sometimes also called compute nodes.

Platform firmware

Firmware running on mission-mode cores on the processor. On x86 CPUs, platform firmware is sometimes called the BIOS or UEFI firmware.

Operating system (OS)

An operating system running on the processor. The operating system may run in a virtual machine on a hypervisor or directly on the machine (bare metal).

OS-First error logging

Error logging using the operating system. This usually requires an agent that runs as a daemon in the operating system that communicates with external system management software.

Poison

Poison is an example of an error containment signal that can be transmitted over interconnect fabrics. Poison is used when the scope of corrupt data is known (e.g. an uncorrected memory error). The corrupted cache line is marked as being poisoned. When an application attempts to use poisoned data it is interrupted. When poison data is sent over I/O, either the I/O device needs to block the poisoned transaction or the I/O link needs to block it by bringing the link down (e.g. PCIe Downstream Port Containment). Poison must be tracked in each ECC domain; if poison is generated by memory UCEs, any cache that fetches it must remember that it received corrupted data so that any consumption of the poison is blocked. It is critical that hardware log errors when poison is generated since the server fleet operator will use those errors to determine what FRU needs to be replaced.

Poison Consumption

Poison Consumption implies software attempting to read data that is marked with poison signature. It does not imply that poisoned data is used by the software and instead indicates that software was blocked from using the poisoned data. Refer to the 'Poison' for more details of the data flow.

RAS Features

RAS features mitigate hardware faults. They can be implemented in hardware, software or a combination of both. Some examples of RAS features are memory ECC, DRAM Post Package Repair, marking pages as bad in the OS and poison recovery. Some RAS features like Post Package Repair(PPR) need to be triggered. For example, on a memory controller that supported run-time PPR, software might see a series of corrected errors, all from the same row address. That software could decide that PPR would mitigate the fault causing the corrected row errors and could trigger hardware to do the PPR.

Reporting

Signaling

Error signaling refers to the means by which software is notified that an error has occurred. Machine Check Exceptions(MCEs) or Corrected Machine Check Exceptions(CMCIs) are ways the hardware can let software running in the compute node know that an error has occurred. The BIOS on x86/x64 machines can be notified of errors via System Management Interrupts (though that has security and performance concerns). The CPU can send the BMC interrupts when an error occurs in order to trigger Out-Of-Band error logging.

Viral

Viral is an error containment strategy that is useful when the scope of the corrupted data is not known such as when there is a corrupted packet header or a timeout. When an error causes a loss of coherency, the scope of the corruption is not known. We cannot associate a signal with the affected transactions so we must signal the failure on all transactions. When a block in a chip goes viral, every transaction through the block is marked viral and any block that receives a viral packet also goes viral. As CPUs go viral, they either take an interrupt or halt. I/O is stopped when the I/O controllers go into viral mode. Viral support has typically only been supported on larger systems, but as core counts grow, we need to monitor the potential for SDC and ensure that data corruption is still contained.

12. Appendix B - Implementation Considerations

This section dives into some implementation considerations any BMC-based error logging solution would need to support.

Shared Error Logging Registers with Kernel and BMC

The error logging registers are read non-atomically; there are many registers we need to log for a particular error. Error logging registers frequently do not log the next error until the previous error log in them has been cleared. These overwritten rules are fundamental and are essential to getting a complete error log. There will be race conditions if something in both the kernel and the BMC are logging and clearing errors. They will get incomplete logs if the log is cleared while they are reading it and will get an incomplete picture of the errors that are occurring.

There are several options for enabling sharing:

- Firmware First error logging using firmware interrupts
    - Requires SMM on x86

    - Firmware copies error data to the HEST
    - The security and performance concerns with this solution make it undesirable
- Shadow registers
    - A set for the kernel and a set for the BMC
    - May get out of sync for CEs
    - UCEs will be easier since they are infrequent and we mainly care about the first one
- Embedded error logging microcontroller
    - Microcontroller reads logs from HW and sends them to the BMC
    - Need to ensure OS and microcontroller can both read the data without race conditions
    - Cannot change the programming model of the CPU for legacy OS's