Optimizing Interrupt Handling Performance for Memory Failures in Large Scale Data Centers

Harish Dattatraya Dixit, Fred Lin, Bill Holland, Matt Beadon, Zhengyu Yang, Sriram Sankar

Hardware Sustaining. Facebook.
Globally, there are more than 2.8B people using Facebook, WhatsApp, Instagram or Messenger each month.

Source: Facebook data, Q4 2019

*MAP - Monthly Active People
Contents

- Server Architecture
- Intermittent Errors
- Memory Error Reporting
- Interrupt Handling
  - System Management Interrupts (SMI)
  - Corrected Machine Check Interrupts (CMCI)
- Experiment Infrastructure
- Observations
Server Architecture

- **Compute Units**
  - Central Processing Unit (CPU)
  - Graphics Processing Unit (GPU)
- **Memory**
  - Dual In-line Memory Modules (DIMM)
- **Storage**
  - Flash, Disks
- **Network**
  - NIC, Cables
- **Interfaces**
  - PCIe, USB
- **Monitoring**
  - Baseboard Management Controller (BMC)
    - Sensors, Hot Swap Controller (HSC)
Intermittent Errors – Occurrence and Impact

**CPU**
- Machine Check Exceptions
- Bitflips
  - System Reboots
  - Data Corruptions

**DIMMs**
- Correctable Errors
  - Interrupt Storms, System Stalls
- Uncorrectable Errors
  - System Reboots

**Storage (Ex: Flash, Disks)**
- ECC Errors
- RS Encoding Errors
  - Retries
  - Input Output Bandwidth Loss

**Network (Ex: NICs, Cables)**
- CRC Errors
- Packet Loss
  - Retries
  - Network Bandwidth Loss

**Interfaces (Ex: PCIe, USB)**
- Correctable Errors
  - Retries, Bandwidth Loss
- Uncorrectable Errors
  - System Reboots
Memory Error Reporting

- CEs
- UCEs
- System Management Interrupts (SMI)
- Corrected Machine Check Interrupts (CMCI)
- mcelog
- System Management Mode
- Error Detection and Correction (EDAC)
- Firmware Config
- Kernel Driver
- OS Daemon

Kernel Driver
System Management Interrupts

SMI Trigger:
Memory correctable errors

SMI Handling:
- System Management Mode (SMM)
- Pause all CPU Cores
- Perform Correctable Error (CE) Logging
- Capture Physical Address of the error
- Return from SMM

Processor
- SMI
- NMI Handler
- Machine Check Handler
- OS Error Handling

Firmware
- Logging Handler

Platform
Corrected Machine Check Interrupts

CMCI Trigger:
Memory correctable errors

CMCI Handling:

1. Collect CEs from each core
2. Aggregate CEs
3. Log aggregated CEs (count per poll) [randomly assigned core]

CPU stall on 1 core

Invoke CMCI Handler
EDAC kernel driver for error data collection
Repeat every specified polling interval duration

EDAC

SMI
Firmware
Logging Handler
NMI Handler
CMCI Handler
Machine Check Handler
OS Error Handling
Failure Detection – MachineChecker

- Runs hardware checks periodically
- Host ping, memory, CPU, NIC, dmesg, S.M.A.R.T., power supply, SEL, etc.
Experiment Infrastructure

Failure Detection – MachineChecker

Failure Digestion – FBAR
- Facebook Auto Remediation
- Picks up hardware failures, process logged information, and execute custom-made remediation accordingly
Experiment Infrastructure

Failure Detection – MachineChecker

Failure Digestion – FBAR

Low-Level Software Fix – Cyborg
- Handles low-level software fixes such as firmware update and reimagining
Experiment Infrastructure

Failure Detection – MachineChecker

Failure Digestion – FBAR

Low-Level Software Fix – Cyborg

Manual Fix – Repair Ticketing
• Creates repair tickets for DC technicians to carry out HW/SW fixes
• Provides detailed logs throughout the auto-remediation
• Logs repair actions for further analysis
**Experiment Infrastructure**

**Production System Setup**

**Production Machines**

Configured with
- Step 1: SMI Mem. Reporting
- Step 2: CMCI Mem. Reporting

**Remediation Policy**

Swap Memory
at 10s of Correctable
Errors per second

**Benchmarks**

- Repro Memory Errors: Stressapptest
- Detect Performance Impact:
- SPEC (Perlbench)
  Fine grained stall detector
Experiment Infrastructure

Memory errors in a production environment are random occurrences, and have no fixed periodicity, seen in experimental error injection setup.
Observation 1: System Management Interrupts (SMI) cause the machines to stall for hundreds of milli-seconds based on the logging handler implementation. This is measurable performance impact to report corrected errors.

Application Impact
Example Caching Service

Impact of SMI due to CEs:

- CEs increase (6200 → 7300)
- Default configs, trigger SMIs for every n errors (n=1000)
- Stall all cores of a CPU for 100s of ms
- Application request efficiency drops by ~40%
Observation 2: Benchmarks like perlbench within SPEC are useful to quantify system performance. For variable events, we need to augment the benchmarks with fine-grained detectors to capture performance deviations.

Detect performance impact using benchmarking

Perlbench
- Compare scores with and without SMI stalls.
- Benchmarks return same scores

Stall detection
- CPU Stall duration: 100s of ms
- Fine-grained stall detection to observe CPU stalls

**Stressapptest:** Helps surface memory correctable errors due to bad DIMMs

No difference observed in scores with or without Correctable Errors (and the SMI stalls)
Minimizing performance impact using CMCI interrupts

CMCI Trigger:
Memory correctable errors

CMCI Handling:
1. Collect CEs from each core
2. Aggregate CEs
3. Log aggregated CEs (randomly assigned core)

- Invoke CMCI Handler
- EDAC kernel driver for error data collection
- Repeat every specified polling interval duration

CPU stall on 1 core
Observation 3: SMI are several times more computationally expensive than CMCI for correctable memory error reporting in a production environment.

SMI vs CMCI performance impact

SMI:
- Stall all cores
- Provide full physical address of the error

CMCI:
- Stall 1 CPU core

Graph:
SMI stall time vs CMCI stall time vs Number of Errors

Results hold for M1, M2, M3 machine types since the stalls are a function of error counts.
Observation 4: We see that with increased polling interval, the amount of time spent in individual aggregate logging by the EDAC driver increases.

Every Polling Interval
- Log aggregated CEs (randomly assigned core)
  - **CPU stall** on 1 core

Optimizing Polling Interval
- Tradeoff
  - Error visibility frequency vs **Individual CPU stall**
- Modify polling interval
  - Obtain maximum individual stall times per core
Observation 5: We see that with an increased polling interval for EDAC, frequent context switches are reduced. Hence the total time a machine spends in stalls will be reduced.

Every Polling Interval
• Log aggregated CEs (randomly assigned core)
  • **CPU stall** on 1 core

Optimizing Polling Interval
• Tradeoff
  • Error visibility frequency vs **Total CPU stall**
• Modify polling interval
  • Obtain total stall times
Observation 6: With increased polling interval for EDAC, we run the risk of overflow in error aggregation.

Every Polling Interval
- Log aggregated CEs (randomly assigned core)
  - CPU stall on 1 core

Optimizing Polling Interval
- Tradeoff
  - Error visibility vs CPU stalls
- Modify polling interval
  - Measure counter overflows and error count variations
Minimizing performance impact using CMCI interrupts

Recommendations:
- For measuring 10s of CEs per second, **use CMCI**
- At polling interval of ~37s
  - **Tradeoff:**
    - Error visibility
    - Maximized
    - Total Stall time
    - Minimized
Post Package Repair (PPR)

Memory Error Repair
- DDR4 Feature
- Remaps faulty cells to healthy cells in memory
- Requires physical address for performing PPR
  - SMI provides physical address of error.
  - CMCI doesn’t provide physical address.
- Hard PPR (Preferred)
  - Persistent across reboots
- Soft PPR
  - Not persistent across reboots

To overcome this,
Use a hybrid approach, CMCI in production flow, SMI in remediation flow
Hybrid Error Reporting Approach

- Production Machine (CMCI Interrupt)
- Change Interrupts (SMI to CMCI)
- Perform Hard PPR
- Run Benchmarks (Memory Stress)
- Reduce SMI trigger thresholds
- MachineChecker (If error > PPR threshold)
- Daemon
- Alert Manager
- FBAR
- Change Interrupts (CMCI to SMI)

Run periodically and collect output
Conclusion

SMI vs CMCI
- SMI's results in **stalls of 100s of ms** in production environments
- **Benchmarks can be augmented** to be sensitive to fine-grained stalls.
- **CMCI more efficient** for reporting memory errors in production.
- CMCI can further be **optimized by tweaking polling intervals**.

PPR
- **Hybrid implementation** to reduce perf impact in production, and obtain benefits of PPR
facebook | Thank you
facebook