RAS/mcelog Plugin High Level Design

Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Collection Interval should be configurable</td>
</tr>
<tr>
<td>2.0</td>
<td>Support for all Platform RAS Notifications/ Events</td>
</tr>
<tr>
<td>3.0</td>
<td>Support for all RAS counters</td>
</tr>
<tr>
<td>4.0</td>
<td>RAS MIB definition</td>
</tr>
<tr>
<td>5.0</td>
<td>Documentation on preferred configuration to enable RAS Memory features</td>
</tr>
<tr>
<td>6.0</td>
<td>Failure events must be detected within 10ms</td>
</tr>
</tbody>
</table>

Overview

The goal of this equivalence feature is to expose Reliability, Availability and Serviceability (RAS) features metrics and events provided by the platform to higher level fault management applications.

Design

mcelog plugin

Introduction

The purpose of this plugin is to send notifications and stats relevant to Machine Check Exceptions (MCE) when they occur. The plugin leverages the mcelog Linux utility to detect that an exception has occurred. mcelog supports a client server model and does the logging and accounting of exceptions when they occur. The plugin simply leverages the client protocol and the logging capabilities of mcelog to detect when an exception has occurred. Note, mcelog supports logging to syslog as well as a custom log file.

Mcelog

X86 CPUs report errors detected by the CPU as machine check events (MCEs). Most errors can be corrected by the CPU by internal error correction mechanisms (RAS Features). Uncorrected errors cause machine check exceptions which may kill processes or panic the machine. A small number of corrected errors is usually not a cause for worry, but a large number can indicate future failure.

When a corrected or recovered error happens the x86 kernel writes a record describing the MCE into an internal ring buffer available through the /dev/mcelog device. mcelog retrieves errors from /dev/mcelog, decodes them into a human readable format and prints them on the standard output or optionally into the system log. This device (/dev/mcelog) should be read in a regular cronjob by the mcelog program or instead of the cronjob the mcelog daemon.

Optionally mcelog can also take more options like keeping statistics or triggering shell scripts on specific events. By default mcelog supports offlineing memory pages with persistent corrected errors, offlineing CPU cores if they developed cache problems, and otherwise logging specific events to the system log after they crossed a threshold.

When an uncorrected machine check error happens that the kernel cannot recover from then it will usually panic the system. In this case when there was a warm reset after the panic mcelog should pick up the machine check errors after reboot. This is not possible after a cold reset.

When the panic triggers a kdump kexec crash kernel the crash kernel boot up script should log the machine checks to disk, otherwise they might be lost.

Note that after mcelog retrieves an error the kernel doesn’t store it anymore (different from dmesg (1)), so the output should be always saved somewhere and mcelog not run in uncontrolled ways.

http://www.mcelog.org/manpage.html

Plugin Configuration

Plugin configuration parameters description can be found at: https://github.com/collectd/collectd/blob/master/src/collectd.conf.pod

Example configuration for the plugin can be found at:

https://github.com/collectd/collectd/blob/master/src/collectd.conf.in
Plugin Functionality

The plugin does the following:

- Checks mcelog server liveliness, reports a failure if it’s not running or if it fails.
- Retrieve aggregated Memory Corrected and Uncorrected Errors from the client protocol (Submit event/stat).
- Retrieve other machine check exceptions from the log file (Submit event).

Sequence diagram

mcelog sample output

mcelog can relay information regarding MCEs in two ways: client protocol (Memory MCEs only) or through writing to a log file (syslog or other). The next sections show the sample output for each case. Please note, that anything written to a log file will be relayed by means of a collectd notification, while information received through the client protocol will be relayed by a notification or a statistic from collectd.

Client Protocol

[mcelog]# mcelog --client

Memory errors

SOCKET 0 CHANNEL 0 DIMM 0
DMI_NAME "DIMM_A1" DMI_LOCATION "NODE 0 CHANNEL 0 DIMM 0"

corrected memory errors:

2 total
2 in 24h
uncorrected memory errors:

0 total
0 in 24h

Log file

Hardware event. This is not a software error.
MCE 0
CPU 0 BANK 1
ADDR abcd
TIME 1475154705 Thu Sep 29 14:11:45 2016
MCG status:
MCi status:
Corrected error
Error enabled
MCi_ADDR register valid
MCA: No Error
STATUS 9400000000000000 MCGSTATUS 0
MCGCAP 1000c1d APICID 0 SOCKETID 0
CPUID Vendor Intel Family 6 Model 62

syslog

Oct 3 09:24:19 localhost mcelog: Hardware event. This is not a software error.
Oct 3 09:24:19 localhost mcelog: MCE 0
Oct 3 09:24:19 localhost mcelog: CPU 0 BANK 1
Oct 3 09:24:19 localhost mcelog: ADDR abcd
Oct 3 09:24:19 localhost mcelog: MCG status:
Oct 3 09:24:19 localhost mcelog: MCi status:
Oct 3 09:24:19 localhost mcelog: Corrected error
Oct 3 09:24:19 localhost mcelog: Error enabled
Oct 3 09:24:19 localhost mcelog: MCi_ADDR register valid
Oct 3 09:24:19 localhost mcelog: MCA: No Error
Oct 3 09:24:19 localhost mcelog: STATUS 9400000000000000 MCGSTATUS 0
Oct 3 09:24:19 localhost mcelog: MCGCAP 1000c1d APICID 0 SOCKETID 0
Oct 3 09:24:19 localhost mcelog: CPUID Vendor Intel Family 6 Model 62

Mcelog server output log file (or syslog, depending on mcelog server configuration provided in /etc/mcelog/mcelog.conf) may contain detailed error messages of many categories, including:

- Memory errors
- IO errors
- CPU errors
- QPI (Intel QuickPath Interconnect) errors
- System errors

All those error messages are processed by mcelog plugin and published as collectd notifications.

Logfile Valid identifier/value pairs

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU number [bank-nr]</td>
<td>Starts a record. Describes the CPU number, optionally followed by the bank number. Required. Can be followed by STATUS or BANK identifier-pairs.</td>
</tr>
<tr>
<td>CPU cpu-number: Machine Check Exception mcg-status-nr Bank bank-nr: status-nr</td>
<td>Starts a record. Alternative form of CPU that is output by the kernel, never by mcelog. Describes the CPU number, mcg-status, bank, status fields in one line</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BANK number</td>
<td>Describe the machine check bank. May be on the same line as CPU.</td>
</tr>
<tr>
<td>STATUS number</td>
<td>Describe the machine check bank. May be on the same line as CPU.</td>
</tr>
<tr>
<td>MCGSTATUS number</td>
<td>The mcgstatus field Will be decoded below.</td>
</tr>
<tr>
<td>RIP number</td>
<td>The program counter May be followed by !INEXACT! if the exact program counter cannot be determined</td>
</tr>
<tr>
<td>RIP segment:number (symbol)</td>
<td>The program counter in segment/offset format, with an kernel symbol May be followed by !INEXACT! if the exact program counter cannot be determined</td>
</tr>
<tr>
<td>TSC number</td>
<td>The CPU time stamp counter at the time of the event</td>
</tr>
<tr>
<td>ADDR number</td>
<td>The physical memory address of the error.</td>
</tr>
<tr>
<td>MISC number</td>
<td>The MCi_MISC register. Values are model specific. Will be decoded below.</td>
</tr>
<tr>
<td>PROCESSOR vendor:cpuid</td>
<td>The CPU vendor. Vendor identifier (1 = Intel, 2 = AMD) and the CPUID 1.EAX identifier.</td>
</tr>
<tr>
<td>MCGCAP number</td>
<td>The MCGCAP register. May be decoded below.</td>
</tr>
<tr>
<td>APICID number</td>
<td>The APIC ID of the logical processor where the error occurred</td>
</tr>
<tr>
<td>SOCKETID number</td>
<td>Describe the physical APIC-ID of the socket the error occurred on.</td>
</tr>
<tr>
<td>CPUID Vendor vendor Family family family Model model</td>
<td>Describe the CPUID version. Decoded version of PROCESSOR</td>
</tr>
<tr>
<td>TIME time [decodedtime]</td>
<td>Time when the error occurred, in time_t format. Optionaly followed by human readable time. Decoded version of TSC</td>
</tr>
</tbody>
</table>

**SNMP Support**

As traps are not supported, the only information relayed through SNMP will be memory error related:

- corrected memory errors
  - total error number
  - number of errors over a period of time
- uncorrected memory errors
  - total error number
  - number of errors over a period of time

**Considerations**

**Configuration Considerations**

BIOS configuration options associated with Errors.

Mcelog must be configured to run on the platform in daemon mode and logging capabilities must be enabled.

**Deployment Considerations**

Mcelog must also be configured to run on the platform

A custom RAS-MIB will be deployed and registered on the platform

**API/GUI/CLI Considerations**

**Equivalence Considerations**

**Security Considerations**
IPC between the server and client is through a UNIX socket.

**Alarms, events, statistics considerations**

MCEs must be relayed as alarms, and where possible if a statistic can be retrieved for the type of error than this should be relayed as a statistic also.

**Redundancy Considerations**

**Performance Considerations**

Not part of Telemetry so performance is Not Applicable, however, collectd and its plugins should not impact the overall performance of a workload running on the platform

**Testing Consideration**

Mcelog test suites cover large parts (but not all of) the kernel corrected machine check code at the software level. It does not exercise the actual error handling and correcting hardware or any BIOS components. To exercise these components you can:

- Use APEI error injection if your BIOS supports it. See examples in mce-test below.
- Use a known bad DIMM that throws errors in a test system
- Test on a live cluster with enough machines. Enough transistors imply plenty of errors.
- Use special error injecting DIMMs.
- Use extreme measures like hair dryers (not recommended, use at your own risk)

**Other Considerations**

Mcelog doesn’t support PCIe AER, so relevant errors will not be retrievable.

User must be root user to run collectd with mcelog plugin enabled.

**Impact**

The following table outlines possible impact(s) the deployment of this deliverable may have on the current system.

<table>
<thead>
<tr>
<th>Ref</th>
<th>System Impact Description</th>
<th>Recommendation / Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key Assumptions**

The following assumptions apply to the scope specified in this document.

<table>
<thead>
<tr>
<th>Ref</th>
<th>Assumption</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key Exclusions**

The following exclusions apply to the scope discussed in this document.

<table>
<thead>
<tr>
<th>Ref</th>
<th>Exclusion</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key Dependencies**

The following table outlines the key dependencies associated with this deliverable.
<table>
<thead>
<tr>
<th>Ref</th>
<th>Dependency</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mcelog daemon</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Collectd message log parser utility</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Issues List**

<table>
<thead>
<tr>
<th>Ref</th>
<th>Issue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The sample file format is proposed in section 1.1.1.1</td>
<td></td>
</tr>
</tbody>
</table>