August 20, 2018

A Firmware-first, Early Warning System to Detect Performance Anomalies and Provide Actionable Insights in User-visible Interfaces

Srinivasan Varadarajan Sahasranamam
Hewlett Packard Enterprise

Sandeep S Yelandur
Hewlett Packard Enterprise

Title: A Firmware-first, Early Warning System to Detect Performance Anomalies and Provide Actionable Insights in User-visible Interfaces

Abstract:

Application performance can be affected adversely due to many reasons. Some of these sources of performance degradation remain hidden from user visibility due to their very nature of occurrence, such as misconfigured hardware or firmware settings, or a faulty piece of hardware that is designed to keep the system functional, albeit on a reduced horsepower. Prevalent monitoring approaches rely on such performance degradations having persisted in the platform before they can be acted upon. In this paper, we propose a novel, “firmware-first”, rules-based approach for early detection of performance anomalies, both during the boot process and at runtime (where the anomalies may manifest due to autonomous recovery actions taken either in hardware or firmware), and propagating such anomalies to standard user-visible interfaces, to help customers make informed decisions before deploying their services.

Summary of invention:

This disclosure relates to a firmware-first approach, using linked performance rules, for early detection of anomalies affecting workload performance.

Platform differentiation defining enhanced TCE and TCO, comes not just from guaranteed performance, but also from the ability to detect and propagate any potential system anomalies that might degrade application performance, enabling customers to take corrective action and mitigate business risks. Prevalent monitoring approaches rely on performance degradations having persisted in the platform before they can be acted upon. In this paper, we propose a novel. “firmware-first”, rules-based approach to early detection of performance anomalies, both during the boot process and at runtime (due to autonomous actions taken either in hardware or firmware), and propagating such anomalies to standard user-visible interfaces, to help customers make informed decisions before deploying their services.

The Problem:

Based on data from internal benchmark runs, insufficient cooling degrades CPU-bound integer workload throughput by 4.5%. Non-cluster CPU configuration degrades CPU-bound integer workload throughput by 1%. Memory double refresh degrades memory-bound workload performance by 2% (1x vs 2x Refresh rate). Mixing DIMMs speeds degrades memory-bound workload performance by as much as 3%. Enabling HyperThreading on Intel x86 platforms degrades single-threaded performance by as much as 3%.

Customer support data indicates 14% of the reported cases to be server performance issues. Of those, about 13% resulted in a system board replacement. 95% of those board replacements were done without adequate troubleshooting.

Performance degradation affects business SLAs, and could mean revenue loss. Board replacement implies extended down time, compounding the business impacts and its associated effects on revenue. In both the scenarios described above, the issues could be triaged faster if there was enough data. And the issues could very well be prevented in the first place, if we had early indications of the performance problems.

Our Solution:

Figure 1 describes the high level solution architecture. Figure 2 describes the structure of an anomaly record. Figure 3 illustrates 2 example rules, one for a compute-centric workload and the other for a memory-centric workload.

The key components of the solution are the Anomaly Rules Database (ARDB), the Telemetry Data (TD), the Performance Anomaly Records (PAR), the Transport Layer (TL), the Lightweight Performance Analyzer (LWPA) and the RedFish Alerts (RA).

In the remainder of this section, we focus on a brief description of the differentiating aspects of the solution, viz., the linked performance anomaly rules, and its applicability to detection of configuration-induced performance anomalies during the boot process.

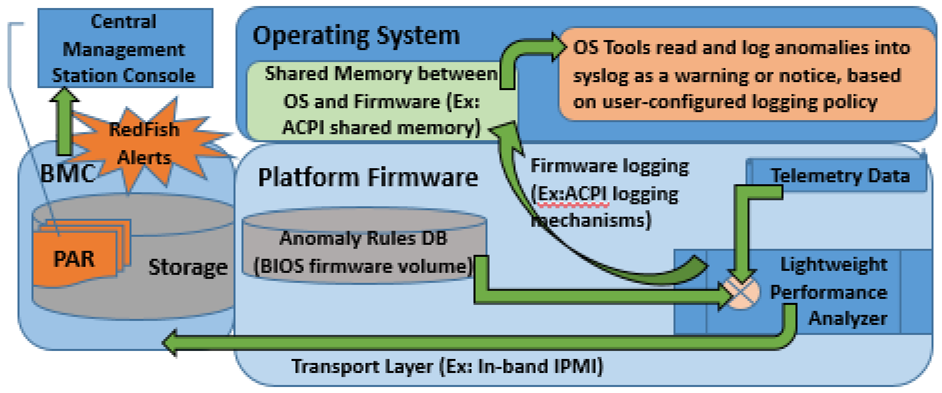

Central Management Station Console
Operating System
Shared Memory between OS and Firmware (Ex: ACPI shared memory)
OS Tools read and log anomalies into syslog as a warning or notice, based on user-configured logging policy
RedFish Alerts
BMC
Platform Firmware
Firmware logging (Ex:ACPI logging mechanisms)
Telemetry Data
PAR
Storage
Anomaly Rules DB (BIOS firmware volume)
Lightweight Performance Analyzer
Transport Layer (Ex: In-band IPMI)

Figure 1. Firmware-First Anomaly Detection Architecture

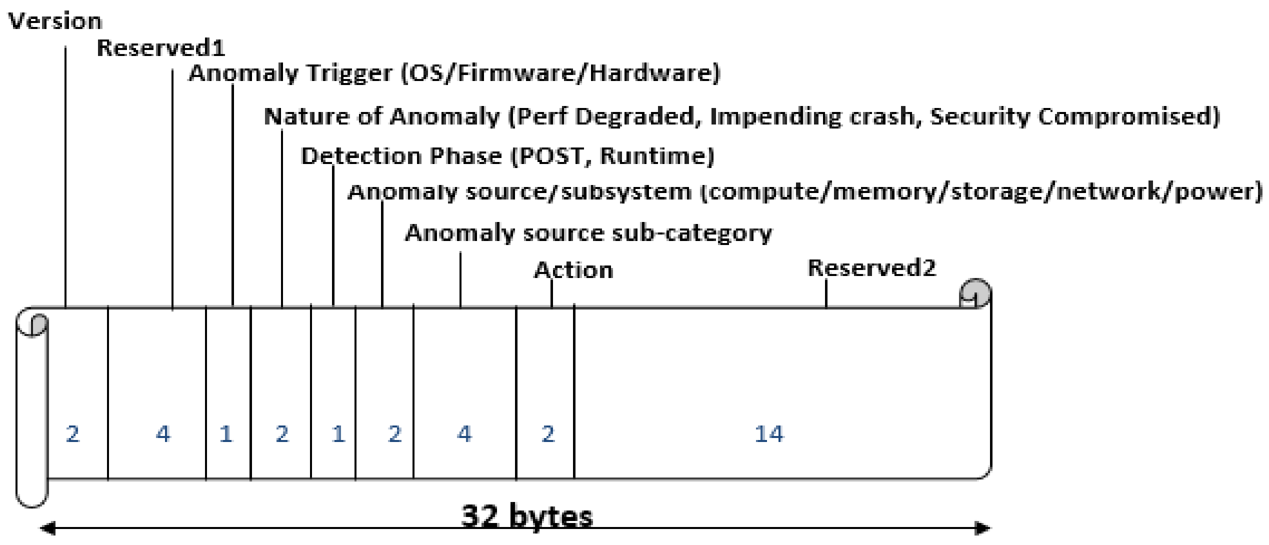
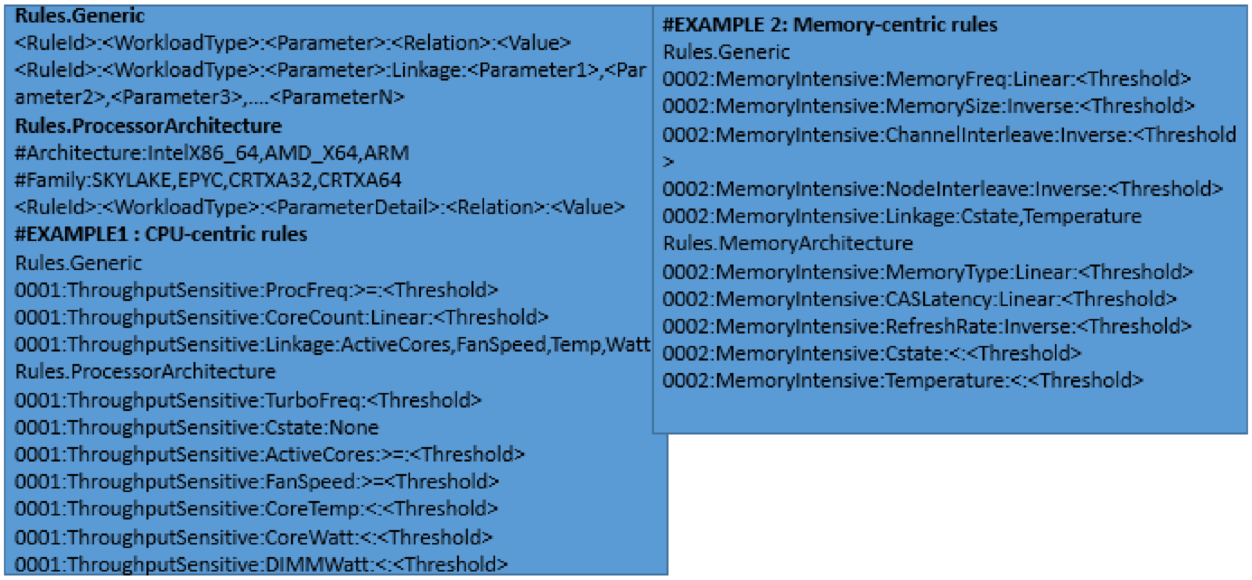

Version
Reserved1
Anomaly Trigger (OS/Firmware/Hardware)
Nature of Anomaly (Perf Degraded, Impending crash, Security Compromised)
Detection Phase (POST, Runtime)
Anomaly source/subsystem (compute/memory/storage/network/power)
Anomaly source sub-category
Action
Reserved2
2
4
1
2
1
2
4
2
14
32 bytes

Figure 2. Layout of an Anomaly Record



```
Rules.Generic
<RuleId>:<WorkloadType>:<Parameter>:<Relation>:<Value>
<RuleId>:<WorkloadType>:<Parameter>:Linkage:<Parameter1>,<Par
ameter2>,<Parameter3>,....<ParameterN>
Rules.ProcessorArchitecture
#Architecture:IntelX86_64,AMD_X64,ARM
#Family:SKYLAKE,EPYC,CRTXA32,CRTXA64
<RuleId>:<WorkloadType>:<ParameterDetail>:<Relation>:<Value>
#EXAMPLE1 : CPU-centric rules
Rules.Generic
0001:ThroughputSensitive:ProcFreq:>=:<Threshold>
0001:ThroughputSensitive:CoreCount:Linear:<Threshold>
0001:ThroughputSensitive:Linkage:ActiveCores,FanSpeed,Temp,Watt
Rules.ProcessorArchitecture
0001:ThroughputSensitive:TurboFreq:<Threshold>
0001:ThroughputSensitive:Cstate:None
0001:ThroughputSensitive:ActiveCores:>=:<Threshold>
0001:ThroughputSensitive:FanSpeed:>=<Threshold>
0001:ThroughputSensitive:CoreTemp:<:<Threshold>
0001:ThroughputSensitive:CoreWatt:<:<Threshold>
0001:ThroughputSensitive:DIMMWatt:<:<Threshold>
```

```
#EXAMPLE 2: Memory-centric rules
Rules.Generic
0002:MemoryIntensive:MemoryFreq:Linear:<Threshold>
0002:MemoryIntensive:MemorySize:Inverse:<Threshold>
0002:MemoryIntensive:ChannelInterleave:Inverse:<Threshold
>
0002:MemoryIntensive:NodeInterleave:Inverse:<Threshold>
0002:MemoryIntensive:Linkage:Cstate,Temperature
Rules.MemoryArchitecture
0002:MemoryIntensive:MemoryType:Linear:<Threshold>
0002:MemoryIntensive:CASLatency:Linear:<Threshold>
0002:MemoryIntensive:RefreshRate:Inverse:<Threshold>
0002:MemoryIntensive:Cstate:<:<Threshold>
0002:MemoryIntensive:Temperature:<:<Threshold>
```

Figure 3. Rules format example

Detailed description of the linked performance anomaly rules (the key aspect of this disclosure):

The rules are defined in two sections, a generic (platform-agnostic) section, and a platform (architecture-dependent) section. The generic section also defines how the performance parameters are linked, making it a key differentiation in the solution. For example, for a throughput sensitive workload, apart from the operating frequency, we also define what other parameters influence the peak performance, viz., wattage of the processor, the temperature around the CPU and the DIMMs, and the configured cooling (fanspeed). A change in any of these linked parameters can influence the operating frequency. For example, if the configured cooling is less (low fanspeed), or the observed ambient temperature is high, the engine would detect and flag them as degrading performance for cpu-bound workloads right during the boot process itself. Similarly, for a memory-bound workload, apart from the routine settings for memory, we also define linked settings like temperature and Cstate (on Intel x86 architecture). If an application is memory bound, allowing the processor to enter deep sleep states can cause the application to incur latency penalties if the memory controller is shutdown due to the processor entering the deeper sleep states. If deep sleep states are enabled and the workload type is memory-bound, the engine would flag potential performance degradation during the boot itself. Existing solutions monitor either one or two known parameters, but we use our knowledge of the platform to define how other platform attributes are linked to performance. And we apply this in the context of a known workload type to ensure best performance for that specific workload. This is where our solution differentiates itself from other routine monitoring solutions either in firmware or in the OS.

While we limit ourselves to configuration anomalies detected during boot (static anomalies), we would like to point out that the format of the rules allows the engine to monitor any linked parameters supported by the underlying platform, to determine potential degradations at runtime too and raise alerts. One such example would be the detection of memory performance degradation due to excessive errors on the DIMMs. Or power-supply failures (in a redundant power-supply configuration) that might reduce the available power by half, potentially degrading performance for bursty workloads during their peaks.

Here is a description of the rules format with concrete examples for ThroughputSensitive and MemoryIntensive workloads.

1. Workload type – ThroughputSensitive

   The generic section of the rules describe the architecture-agnostic elements of the processor and how they affect performance. For example, we state that this specific workload category is affected primarily by “core count”, “frequency” and other linked parameters like “# of active cores”, “Fan Speed”, “Temperature of the processor” and “Wattage consumption of the processor”. The rules define the thresholds and also the nature of the relationship. For example, the rule states that the performance has a linear relation to the “Core Count”, meaning, performance increases as the number of cores increase. The threshold for frequency is derived from the processor specifications (usually available from architected registers on the processor that can be read early during boot). The thresholds for the linked parameters are derived from the platform specifications (for the fan speed) or from the hardware specifications (for processor temperature and wattage).

   Similarly, the architecture-specific section lists performance parameters specific to a processor architecture. In the example discussed above, we cover Intel x86 processors. So we list parameters like C-State and TurboFrequency. Thresholds are again derived from the processor specifications from the vendor.

   The values of these parameters at any given point in time (be it boot, or at run time) are readily available for consumption as they are already collected for reporting. The LWPA uses this data to match against the pre-configured thresholds and flag potential performance degradations to standard user-visible interfaces.

2. Workload type – MemoryIntensive

   Similar to the processor-centric rules, we define architecture-agnostic and architecture-specific rules. Examples of architecture-agnostic rules are memory frequency, memory size and the interleave strategy. Examples for architecture-specific elements are CAS latency and Refresh rate. We also provide the link to processor parameters like C-State, since they have a direct bearing on the memory functionality (for example, allowing deep sleep states like C6 can cause the memory controller to opportunistically shutdown during periods of low traffic. But this can cause a large exit latency when processor needs to access memory via that specific memory controller).

   The values of these parameters are also readily available for consumption at any given time. The LWPA uses this data to match against the pre-configured thresholds and flag potential performance degradations

The linked performance parameters are derived from the historical knowledge base within the performance engineering team, leveraging the extensive tunings on various industry standard benchmarks and also from tuning customer systems for pre-sales engineers during RFP bids.

Competitive approaches (state of art):

Most solutions make use of OS agents to monitor, measure and act upon detected anomalies. This means borrowing compute cycles from the host processor, which could place additional compute demands on the system being monitored. This would simply add costs to the overall solution.

Our firmware-first approach makes it OS-agnostic, while still allowing OS based agents to consume notifications and take actions. The rules database is an extensible framework, allowing rules to be modified as hardware and firmware evolve. The use of performance anomaly records allows a common and standardized reporting format for performance anomalies, enabling platform firmware to retrieve data from devices like storage and networking and roll it up to system performance behaviors. An embedded solution minimizes latency penalties in alert notifications (inherent in cloud or appliance-based solution).

Disclosed by:

Srinivasan Varadarajan Sahasranamam and Sandeep S Yelandur – Hewlett Packard Enterprise