Trident

Automated system tool for collecting and analyzing hardware performance counters

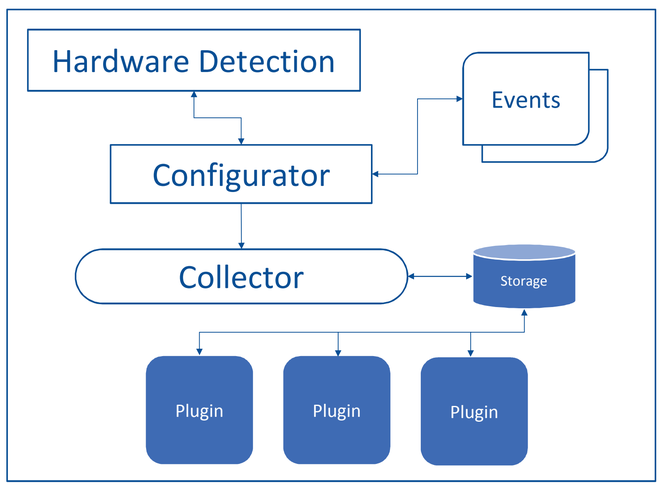
Servesh Muralidharan & David Smith

{servesh.muralidharan, david.smith}@cern.ch

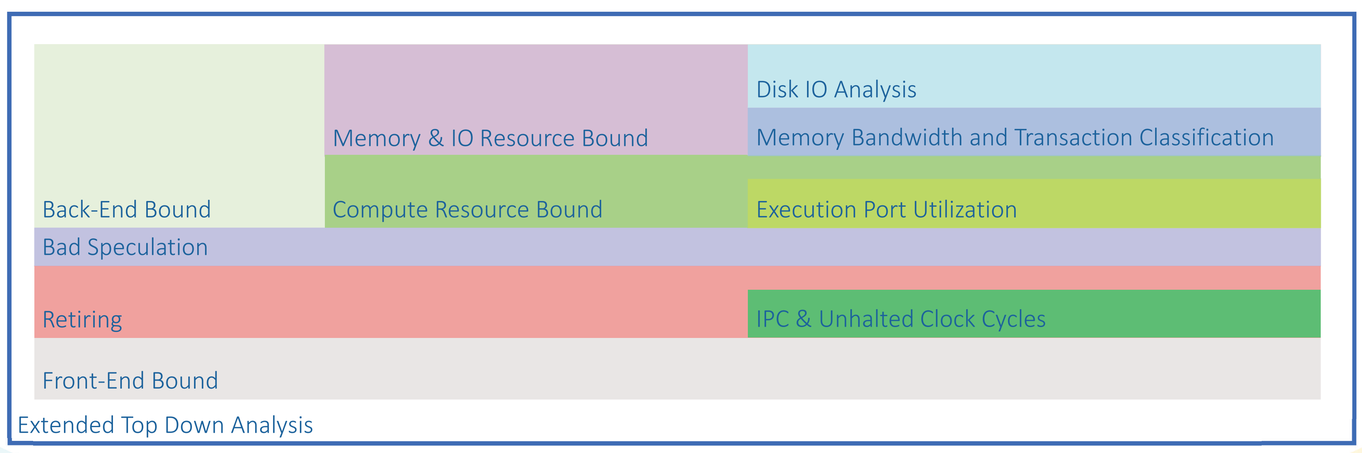
Trident is a system level performance monitoring tool that combines information from several CPU hardware counters and system counters. A three pronged approach of combining core, memory and IO metrics under a single analysis for node utilization is the basic principle of the tool. The hardware counters are monitored in such a way that it has a negligible impact on the performance of the application being executed thereby reducing the chance of significant "Observer effects". Finally we perform basic analysis using an extended top down approach to present the resultant data in several user friendly timeline trace graphs.

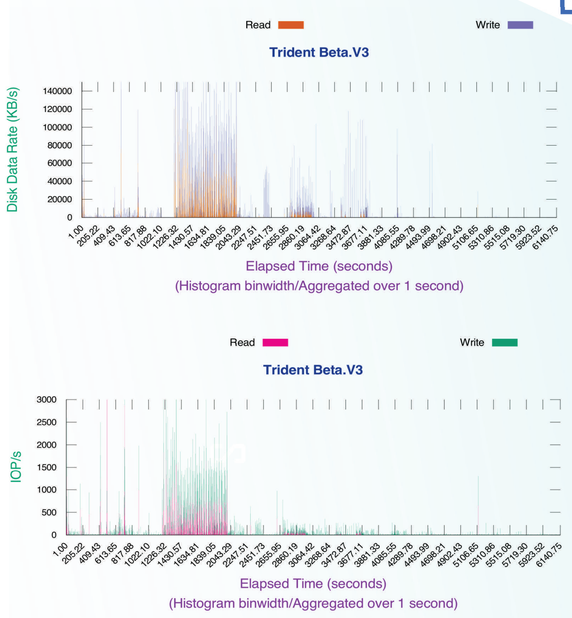
To collect the performance counters for core and memory we utilize Intel hardware counters built into modern server Xeon processors. A detection mechanism determines the runtime architecture enabling use of preconfigured counter masks. Metrics are collected at a specified interval. Currently for the IO subsystem we focus on the amount of data to/from block devices, IOPS and amount of time at least one request is outstanding. This IO information is gathered through the Linux proc filesystem. The collected counts are used to generate derived metrics.

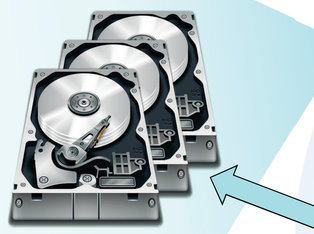
Monitoring and collecting performance metrics at near real time is intended to understand compute demands better and which changes can improve utilization. The raw metrics are often difficult to interpret, hence development of this tool to allow the scientific communities to both collect and interpret resource utilization data more easily.

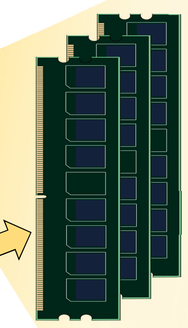


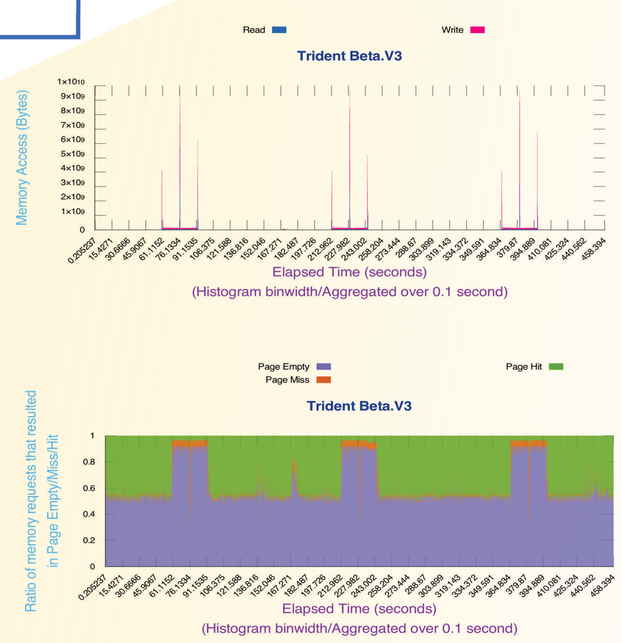
Conventional performance analysis tools often focus on optimization of large applications through identification of hotspots. However, several studies have shown that sometimes applications may not show localized hotspots, making it more difficult to benefit from such techniques. This can be due to complexity caused by a number of reasons. For instance the descriptive programming of complex physics processes which are expressed through many functions. Common patterns such as iterators or design principles of object oriented programming such as polymorphism can make the analysis of the situation more difficult for analyzers, such as an optimizing compiler. In short, a code base can become highly segmented and make it difficult to easily identify regions of the code which would benefit most from refactoring.







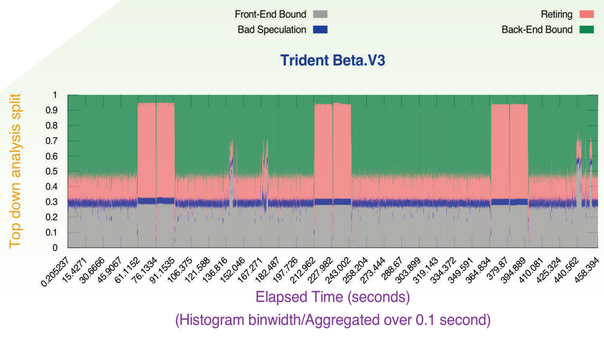


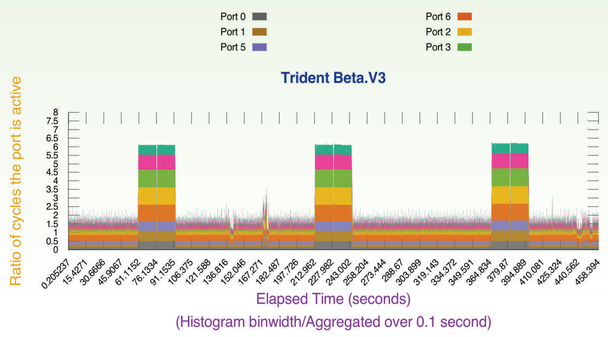


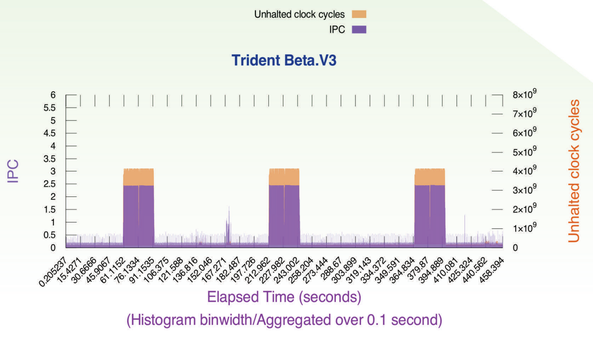
- Disk data transfer rate (bytes read or written)
  - Volume of data being stored or read
  - Can be compared to the specifications for the device and interface
- Disk data access rate (IO operations per second)
  - In combination with the number of bytes involved indicates the request size
  - Can be compared to the specification of the device
- Outstanding requests to the device
  - Gives an indication about the capability remaining to service more requests

- Top down characterization
  - Identifies the resources dominated by the workload
  - Front-End – fetch and decode program code
  - Back-End – monitor and execution of uOP once the dependent data operands availability
  - Retiring – Completion of the uOP
  - Bad speculation – uOPs that are cancelled before retirement due to branch misprediction
- Instruction Per Cycle (IPC) / Unhalted clock cycles
  - Denotes ratio of parallel instructions executed
- Execution Unit Port Utilization
  - Determines how many cycles the port was busy
  - Identifies broadly the pressure from different types of uOPs

- Memory bandwidth : Data to/from main memory
- Memory transaction : Page-Hit
  - Memory bank in open state with lowest access latency
  - Sequential memory access usually have high page hits
- Memory transaction : Page-Empty
  - Memory bank is idle and needs to be activated
  - Moderate access latency (Usually 2x of page hit)
- Memory transaction : Page-Miss
  - Memory access that closes a page in the same bank
  - Worst access latency (Usually 3x of page hit)







References

[1] A. Carvalho, "The New Linux 'perf' tools," presented at the Linux Kongress, 2010.

[2] Intel Corporation, "Intel® 64 and IA-32 Architectures Optimization Reference Manual," Intel.

[3] A. Yasin, "A Top-Down method for performance analysis and counters architecture," 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, 2014, pp. 35-44.

[4] V. Cuppu, B. Jacob, B. Davis and T. Mudge, "A performance comparison of contemporary DRAM architectures," Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367), Atlanta, GA, 1999, pp. 222-233.

[5] Linux kernel documentation for procfs, https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html

[6] Characterizing HEP Reference Workloads with Trident: HEPSPEC06 vs SPEC2017, https://indico.cern.ch/event/693206/contributions/3057529/attachments/1677925/2694603/HEPiX_Benchmarking_WG_29Jun18.pdf

Information Technology Department

Département des technologies de l'information