Round Robin Data Bases for Performance Evaluation of High Performance Applications and Clusters

Fernando G. Tinetti[1], **Leopoldo J. Rios**[2]
[1]Fac. de Informática, UNLP, Comisión de Inv. Científicas Prov. Bs. As., Argentina
[2]Instituto IMIT, CONICET-UNNE, Corrientes, Argentina

Abstract – *We introduce a tool for automating (or aiding) performance evaluation of HPC (High Performance Computing) applications by combining both, Round Robin Databases (also referred to as RRD) and performance data collected at runtime. RRD monitoring is at the base of several well-known and popular tools for cluster monitoring. We use take advantage of already existing RRD tools for collecting, processing, and presenting runtime data for scientific processing users. Scientific applications (and even the hardware used by those scientific applications) are assumed to be performance optimized, but it is not always the case. Thus, collecting and analyzing runtime information will help scientific users to decide new optimizations and/or runtime strategies. The primary focus will be on parallel applications running in clusters, i.e. distributed hardware, since they are the most complex to optimize given their different and varying computing and communications patterns.*

Keywords: Performance Evaluation, Round Robin Databases, High Performance Computing.

1 Introduction

Performance optimization and tuning has been done since the beginning of computer science up to these days [1] [2] [3] [4] [5]. Classical tools such as the well-known profilers are used in the first steps of optimization, but they not always provide enough and/or accurate information beyond plain (and statistical) running time/s. Currently, there are multiple types of HPC (High Performance Computing) applications running on production clusters as well as there are multiple possibilities or sources of monitoring information. At the lowest level is found the most accurate information provided by hardware event counters [6] [7]. At the highest level is the (mostly statistical data) collected by operating system (OS) for accounting (such as that found at Linux or OS X /proc).

Beyond abstraction levels, there are multiple ways of collecting data for analysis as well as there is no single place/tool to have every single piece of data needed for system and/or application performance analysis. Some tools are focused are "system wide" in that summarize or provide information regardless specific users and applications, such as Ganglia [8] [9]. Tools such as profilers are specifically designed (at least initially) for single application performance optimization and post-mortem data analysis [10] [11]. Most profilers are not meant for real-time monitoring and are focused on applications, not HPC systems such as clusters (either in hardware or used in a cloud facility). At most, several profilers are single system-wide, and there it should be necessary to aggregate several data sources to have a complete view of a parallel processing application, for example.

Multiple abstraction levels, tools, sources, and types of data usually lead to a complex scenario for HPC application programmers as well as those in charge of medium/long term decisions for HPC systems. This work is a first attempt to provide a single tool for the scientific related to HPC systems, so that it can be adjusted/configured to collect and visualize different data in a uniform way. The huge amount of data involved in performance monitoring will be handled in RRD taking advantage of a priori storage limits as well as the large amount and usefulness of tools for handling those data [12]. Actually, is can be considered that parallel application resource usage/monitoring is more complex than whole cluster monitoring, because a parallel application does not necessarily uses all the resources at the same time.

2 Basic Information

Overcoming the basic problem of real-time profiling (as opposed to the post-mortem data provided by profilers is relatively simple combining three tools/tasks:

1. Find and extract information in real-time (at runtime) from /proc filesystem.
2. Store data in a RRD.
3. Visualize/generate graphic charts and/or define further analysis.

Even when /proc information is system-wide, it is not rare that nodes in the cluster are "reserved" or "exclusively used" for a single process (via some resource manager), thus in that case the /proc (mostly) contains data of a single process. Beyond sampling, ell data is handled by RRDtool. Furthermore, data stored in RRD can be used for more specific analysis beyond graphic charts, depending on users' requirements.

Data extraction from /proc filesystem is relatively easy using standard tools such as grep/awk. Periodic samplig is also simple using OS tools such as at/cron. The most complex data management is data storage and handling, which is made in this proposal by RRDtool. The key of the integration between sampling and RRDtool usage is matching sampling period in order to avoid noise generated by "extra" interpolation. It is not reasonable to sample with a 4 hours rate if the analysis is going to be made in terms of minutes. Given that users have the application knowledge, the final tool will be configurable in terms of data collected and sample rate.

RRDtool provides a lot of facilities for handling sample data in general and those collected by monitoring HPC parallel/ distributed applications. Fixed size data storage and consolidation functions are specifically designed for time series data. Our proposed tool is intended to go beyond standard tools such as Ganglia in that it will show and consolidate information by users or by applications at several extra levels of details. At this point the tools already have implemented data collection by users in order to aid decision making to users and/or a HPC center manager/s. It is worth pointing out that even this basic information is not directly available using standard tools such as Ganglia, which require at least specific customization for handling and generating reports of this kind of data. Up to this point, the greatest contribution would be the tuned usage of well-known tools for clusters and applications analysis. The tuning is specifically related to two key aspects: 1) integration with the runtime system, so that measurements are made by application/user, and 2) user parametrization (as opposed to fixed by design) of data to obtain and/or selection of sampling data.

As a proof of concept, a synthetic application was monitored with our tool. The application is focused in memory: it requests and set to a fixed value 10MB blocks every 5 seconds. The total amount of RAM is 16GB, and the values selected for monitorization were: a) total memory, b) available memory, and c) memory used for cache (memcache in /proc filesystem). Fig. 1 shows the memory evolution that our tool is able to show from beginning to end of runtime. The units are KB, so 10M represents 10GB RAM (the total amount of RAM is 16GB). The application does not have any I/O, so memcache evolution is related only to OS decisions. At about 1/3 of the runtime, there is almost no free memory, but it is due to OS allocation/usage of memcache. The chart graph in Fig. 1 provides information for the user which lets to conclude that at that point in runtime if there is a problem for memory, it is not related to its application but related to OS memcache management. As expected, the application is able to keep running despite there is not "clean" free memory, since the OS is able to release memory from memcache depending of its I/O management. It is left to the user to analyze if its application performance is affected by the time there appears to be no free memory in spite of the fact it is not using all the memory and it is not doing any I/O. It would be possible to relate memcache usage to application runtime if other process activities are monitored, e.g. I/O operations.

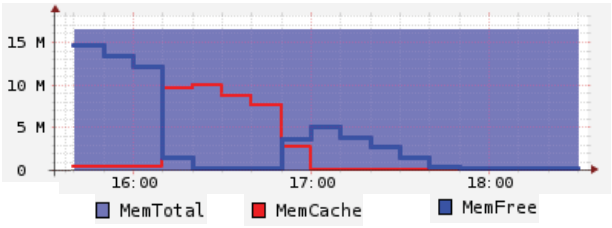

Figure 1: Monitoring Memory

Fig. 2 shows the last minutes of process execution, where the free memory decreases as the used by the process increases, coming to the point that there is more memory used by the process than free memory in the system. Also, Fig. 2 shows that we are able to identify per process memory assignment beyond the "system wide" information (which would be "free").

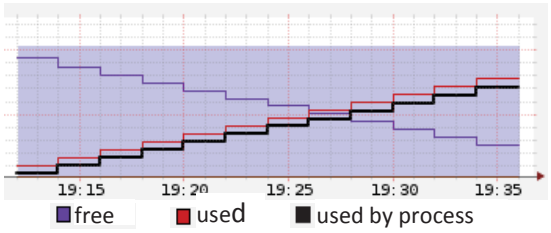

Figure 2: Free and Process Memory Evolution.

Fig. 3 shows another view of the memory usage evolution as well as how the free memory begins increasing after the process has ended.

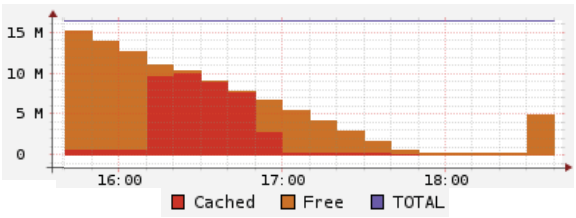

Figure 3: Memory Evolution.

The tool is now being enhanced to collect more specific data at runtime, just like that provided by hardware monitoring counters now available in standard processors [6] [7]. It is worth noting that information as that in Fig. 1 is not directly available in standard profilers or monitoring tools, beyond that it is expected to have even better/more specific data as the tools is enhanced. Also, summarizing this information in a per cluster base is straightforward, since the data is already available with the tool in its current state.

3 Parallel Application Data

Currently, Ganglia [8] is widely used for cluster monitoring and usage visualization in the HPC field. Classical computational resources involved in HPC are easily identified

in a cluster as well as in individual nodes. Once installed, Ganglia begins to collect data on each of the Grid / Cluster nodes, and data are stored in files RRD. Ganglia collects data referred to CPU and memory usage and network interconnection traffic. Large scale (in number of nodes) reports are possible and reports can be customized and sorted by date, for example. However, there are many interesting/ useful metrics missing, or for which there is a huge tuning work for them to be collected. Most of the missing information is related to identifying users and applications. Examples of missing information are (parallel) program using specific cores (or some fraction of cores) and the owners (users) of processes at runtime in a given time frame. As in the example given in the previous section, we are integrating different sources of monitoring data in order to provide better/more useful information than the currently available with standard tools. Also, given that the methodology (e.g. sampling, time series data) and basic tools (e.g. RRDtools) are maintained there is no overhead beyond that of current and *de facto standard* tools, thus maintaining quality of service in general.

Cluster/distributed resource managers such as Torque (aka Torque PBS –Portable Batch System) [13] are a very good source of valuable information for HPC users as well as cluster managers. Resource managers usually are a suite of applications that help to manage and organize the jobs in a cluster and/or grid. Users are able to interact in a rational and organized way with the computational resources, avoiding downtime, for example. Job submission, monitoring, and visualization are performed via simple commands. Most resource managers (including the aforementioned Torque) let users to specify required resources such as nodes, cores per node to be used (or processes to be started at each node), memory, and other specific details such as command name and (usually command line) parameters to execute a job. Statistical data provided by the resource managers are very useful. Most (if not all) current monitoring tools do not integrate such information, and we think it has to be integrated to those provided by the operating system, mostly because they refer to particular (parallel) jobs, directly related to specific applications and users.

Our monitoring tool currently integrates data provided by Torque as a proof of concept: we are able to take advantage data provided by resource managers. Thus, the information provided to users and system managers is enhanced, combining data provided by resource managers to that provided by the OS accounting system. Actually, the integration is made at the level of data stored and handled in RRD and by RRDtools, i.e. as time data series. As a first step, our tool provides further information beyond that provided by standard monitoring tools about CPU/cores usage. Specifically focused on parallel application performance monitoring, it is important to identify the user/s running processes on our cluster and the number of cores actually being used. Standard tools usually provide the % of cores and or nodes being used. Fig. 4 shows the evolution in time of cores utilization per user in a cluster.

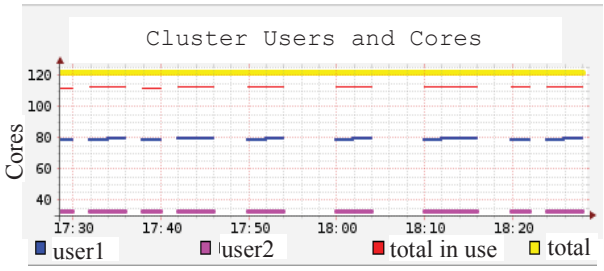

Figure 4: Cluster Cores Usage.

Fig. 4 shows/confirms that user1 is most likely running a parallel application (classical SPMD processing with a fixed number of cores) while user2 is running a sequential job. The parallel application can be definitely confirmed using data from the specific cluster resource manager (queue manager) being used to start cluster processing jobs. Fig. 4 also shows the total number of cores being used as well as the total number of available cores, as standard in cluster monitoring tools.

The monitoring tool is currently being enhanced by including core as well cluster optimization/utilization data maintaining their uniform handling in RRD-RRDtools. One the first data we are integrating is that provided by hardware performance counters. Initially, we will take advantage of tools such as perf [14] which, in turn, takes advantage of the perf_event Linux kernel interface [15]. Even when perf is very attractive since it does not require any source code change, it implies lack of runtime profiling, since it provides post-mortem data. Hardware performance event counter data can be collected, though, if the code is instrumented with libraries such as PAPI (performance API) [16] [17]. Thus, a runtime profile can be built with low level and precise hardware behavior if the code is specifically instrumented.

4 Conclusions and Further Work

We have been able to integrate multiple sources of data in order to monitor performance behavior and provide information for HPC applications and clusters. Collected data is handled via RRD (Round Robin Databases) so it is manageable in terms of bounded data storage. RRDtools provide highly useful ways of obtaining statistical and graphical data for programmers as well as HPC cluster managers. We have shown how our tool takes advantage of information provided by OS as well as cluster/queue resource managers in order to provide insightful data for HPC optimization and hardware resource planning. Even more, we are able to tune the kind of information specifically useful for parallel programmers having running applications in clusters.

We are working on several enhancements for our tool:

1. Adding user interface capabilities so that a user will be able to easily select among a set of data to be collected, date ranges, and data visualization way/s

2. Integrate post-mortem hardware event counters data provided by tools such as perf. Even when the way in which hardware data is integrated in RRD is straightforward, hardware dependence (i.e. specific events each micro-architecture has implemented) involves another level of parametrization the tool does not currently implement.

3. Integrate profiling with runtime hardware event counters data provided by code instrumentation and library such as PAPI. This is another level of parametrization, since it implies collection of data at runtime and reconstruction of program behavior (profile) from collected data.

5 References

[1] D. H. Bailey, R. F. Lucas, S. Williams, Eds., Performance Tuning of Scientific Applications, CRC Press, 2011.

[2] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, 5 Ed., Morgan Kaufmann Publishers, Inc. 2012.

[3] Fernando G. Tinetti, Andres More, "Hotspot: a Framework to Support Performance Optimization on Multiprocessors", Proc. PDPTA'15, H. R. Arabnia, H. Ishii, K. Joe, H. Nishikawa, H. Shouno, L. D'Alotto, G. A. Gravvanis, G. Jandieri, G. Sirakoulis, A. M. G. Solo, W. Spataro, F. G. Tinetti, G. A. Trunfio, CSREA Press, 2015, pp. 171-176.

[4] A. V. Aho, M. S. Lam, R. Sethi, and J. Ullman, Compilers: Principles, Techniques, and Tools, 2nd ed. Prentice Hall, 2006.

[5] J. A. Bilmes, K. Asanovic, C. Chin, and J. Demmel, "Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology," Proc. of the International Conference on Supercomputing, A. SIGARC, Ed., Vienna Austria, 1997.

[6] Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D, Dec. 2015.

[7] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," International Journal of High Performance Computing Applications, vol. 14, no. 3, pp. 189–204, 2000.

[8] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal, D. Pocock, Monitoring with Ganglia. Tracking Dynamic Host and Application Metrics at Scale, O'Reilly Media, 2012.

[9] Ganglia Monitoring System, http://ganglia.info/

[10] M. Honeyford, "Speed your code with the GNU profiler. Target the parts of your applications that take the most time", April 2006, IBM DeveloperWorks, Technical library GNU Gprof documentation.

[11] W. E. Cohen, Tuning Programs with OProfile, Wide Open Magazine, 2004, pages 53–62.

[12] http://oss.oetiker.ch/rrdtool/

[13] Adaptive Computing, Torque Resource Manager http://www.adaptivecomputing.com/products/open-source/torque

[14] perf wiki, https://perf.wiki.kernel.org, Linux profiling with performance counters.

[15] V.M. Weaver. "Self-monitoring Overhead of the Linux perf event Performance Counter Interface", IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2015), Philadelphia, Pennsylvania, March 2015.

[16] V. Weaver, D. Terpstra, S. Moore, "Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations," 2013 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, April 21-23, 2013

[17] F. G. Tinetti, M. Méndez, "An Automated Approach to Hardware Performance Monitoring Counters", CSCI The 2014 International Conference on Computational Science and Computational Intelligence (CSCI'14) March 10-13, 2014, Las Vegas, USA,