

OpenWLC: A Scalable Workload Characterization System

Hong Ong¹, Rajagopal Subramaniyan³, Chokchai Leangsuksun², R. Scott Studham¹

¹Computer Science and Mathematics Division, Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA

²Computer Science Department, Louisiana Tech University
Ruston, LA 71272, USA

³Department of Electrical and Computer Engineering, University of Florida
Gainesville, FL 32611, USA

Abstract

This paper discusses the design and development of a workload characterization system, known as OpenWLC, for large-scale computing systems. Large-scale hereafter implies computing systems with number of nodes ranging from thousands to hundreds of thousand. In such environment, it is critical to accurately monitor the system activities (workloads) and determine the potential performance bottlenecks that limit its effectiveness. Obviously, the challenge lies in providing a performance evaluation framework that can optimally and simultaneously meet the criterions of scalability, performance, extensibility, availability and minimum intrusiveness. OpenWLC is an attempt to meet this challenge. In particular, the OpenWLC framework employs a component-based multi-tier architecture coping with large amounts of monitored data during collection, storage, visualization and analysis stages. At the same time, OpenWLC aims to scale its framework up to and beyond 100,000 processors and to minimize its impact on system application performance by less than 1% (on an average). It is important to note that the extent of OpenWLC goes beyond typical monitoring software. It is indeed an integral environment developed for collecting performance data, workload modeling and system evaluation specially designed for large-scale computing systems.

1 Introduction

The Oak Ridge National Laboratory (ORNL) Leadership Computing Facility (LCF) hosts world-class computing infrastructures such as the Cray X1E and XT3. One of the main LCF's activities is to provide computing platforms that

scale up to thousands and tens of thousand nodes. Needless to say these large-scale computing systems are very difficult to manage due to their size and the complexity of the applications they run. Moreover, the Cray X1E and XT3 are relatively new architectures and thus applications need to be ported and optimized in order to fully utilize the capability of these systems. Consequently, a scalable workload characterization service is vital for a successful deployment and operation of the computing system.

Typically, system performance is determined by utilization of resources such as the CPU, network interconnects, and memory usage. One way to evaluate the effective utilization of resources is to deploy monitoring tools onto the system to collect performance data [1]. Subsequently, the performance data is post-processed to enable time-critical services and long running applications with process migration capabilities to distribute processes and tasks. In other words, resource-monitoring services should provide guidance for better performance evaluation, health and available resource location and usage information.

An ideal workload characterization (WLC) system should consist of a monitoring subsystem that is capable of collecting the system activities and handling large number of performance metrics. It should also provide a scalable repository for both data storage and detailed knowledge about the various aspects of the system's performance. In addition, a workload modeler subsystem should be capable of learning the behavior of a workload for verifying performance models developed for current and future capacity planning. Last but not least, the WLC must include a visualization subsystem that can display performance data meaningfully. Application developers may use the WLC

system to analyze the resource usage of an application, identifying performance bottlenecks or usage patterns which may suggest better algorithms. System administrators may use the workloads information to tune the system by adjusting critical parameters in order to improve performances or uphold certain site-policy.

Continuous monitoring of large-scale computing platforms is a challenging problem for different reasons. The size (number of parameters) of the system to be monitored is often beyond the scalability limit of many available tools. These tools are usually targeted for a particular application and are often proprietary products. As a consequence, the application developers cannot easily adapt them to suit their requirements. Moreover, the tools are usually difficult to configure, provide an inconvenient user interface, and impede scalability to a large number of nodes. In this paper we present a scalable workload characterization system, known as OpenWLC, for large-scale computing. Section 2 discusses previous work related to the problem of monitoring large-scale computing systems. In Section 3 the design goals of OpenWLC are illustrated and rationalized. Section 4 describes the architecture and design of OpenWLC. Finally, Section 5 summarizes the project status and roadmap.

2 Related Work

Considerable attention has been devoted to the problem of performance monitoring for large-scale computing systems [2-9,11,12,14]. These systems vary in terms of data collection strategies (i.e. polling or event driven) and system architecture (i.e. server-based, client-based, or task-based).

Most monitoring tools use proprietary collection protocol over TCP/IP links. One exception is SIMONE [9], which employs the standard SNMP protocol to build a large-scale, distributed monitoring system. Hierarchical monitoring approach has been explored in [14]. Such hierarchical, tree-based monitoring systems are particularly effective when the user is mainly interested in aggregating information about the cluster's status, such as the average load of all the machines, or the least utilized node of the cluster. The reason is that the information can be aggregated at each intermediate node of the hierarchy, thus avoiding the possible bottleneck

of a single node getting all the data from all hosts.

Clumon [3] is a cluster monitoring system that is built on a subset of SGI's Performance Co-Pilot (PCP) monitoring system to collect host data, and tracks jobs submitted to Portable Batch Scheduler (PBS) [15] queues. Clumon web portal allows retrieval of detailed job queue information and displays statuses of the PBS job schedule.

Supermon [4] is a centralized monitoring system that provides efficient and frequent data collection from the nodes of a Linux cluster. However, it requires Linux kernel patches for additional system calls, which provides system status information. A server program running on each machine collects system metrics and then passes them to requesting applications using a telnet-based network protocol. A possible drawback of this approach is the requirement of the kernel patch, and the fact that it requires a new development of the system call if additional parameters need to be monitored.

Hawkeye [5] is a cluster monitoring service developed as a part of Condor job management system. Hawkeye consists of Monitoring Agents and a Monitoring Manager. Monitoring Agent is placed at clients of the Condor cluster. Monitoring Agent collects client system status and, employs the same description form as other Condor services i.e., ClassAds, to send information to the Monitoring Manager. Monitoring Manager uses database as a system status repository for individual client computers. Users can use command interface, graphic interface and web portal to access data.

NWPerf [6] is also a centralized monitoring system for analyzing fine granularity performance metric data on large-scale supercomputing cluster. Architecturally, OpenWLC shares many design goals as NWPerf. However, OpenWLC design pays particular attention to resolving issues raised but not addressed in NWPerf, for example, fine-tuning overall resource usage, standardized API and GUI interfaces, and automated detection of performance anomaly.

Unlike most monitoring tools that focus only on data collection and hierarchy, OpenWLC incorporates visualization and workload modeler in its design to support modeling and analysis.

3 OpenWLC Project

3.1 Introduction

The performance evaluation of any computer system requires understanding of the system's workload. This is because the optimal performance that a computer system can achieve strictly depends on the match between workload and system characteristics. This is even more so for large-scale computing systems with hundreds of service nodes and thousands of compute nodes.

Indeed, in the production environment, workload characterization is a requirement for various resource management policies such as scheduling, capacity planning, workload balancing, scalability assurance, system tuning and configuration, and the construction of benchmarking suites. A well-defined benchmark suite, in particular, can be viewed as a synthesized, controlled workload, which can be ported to different environments to evaluate the system's performance. However, the evaluation of resource management policies, such as how to assist average users in utilizing the system, requires the knowledge of the system characteristics and the behavior of the users activities. If properly executed, workload characterization can aid in determining the relationship between workload and the Quality of Service (QoS)

3.2 Design Goals

OpenWLC aims to provide meaningful workload information that can be used as a guide in making decisions about the configuration and control of the available hardware and software resources. Unlike existing monitoring tools, OpenWLC is designed as an integral environment for supporting performance monitoring and workload characterizing.

To meet the OpenWLC project goal, we identified the following requirements for our workload characterization system:

Minimum overhead - OpenWLC should have little impact on the resource utilization. The service should not incur performance

degradation more than 1% on an average. Lower (possibly zero) overhead is possible with the use of hardware monitors, which are special hardware devices for data collection. However, hardware monitors are usually expensive, special-purpose and non-portable devices.

User interface - OpenWLC should be able to operate in batch mode, without any user supervision. At the same time, a suitable user interface should be provided.

Ease of use - OpenWLC should be easily configurable. If possible, the configuration file should be based on some standard notation, so the user is not required to learn a new one.

Scalability - OpenWLC should be as efficient as possible, and able to scale at least up to 100,000 processors.

Extensibility - OpenWLC should be able to deal with a wide range of different hardware devices and parameters to be monitored.

Availability - OpenWLC should continuously provide monitoring service in case of hardware or software failures. This capability not only ensures that the service is always available but also enables monitoring of both hardware health and software statuses.

4 OpenWLC Architecture

The system-wide scalability of the workload characterization service is deeply impacted with increasing number of processors or nodes in the system. Communication management in terms of the network load and data volume at aggregation/collection nodes, and data management in terms of storage and visualization are complicated. In lieu of addressing such scalability constraints, the OpenWLC framework is based on a multi-tier architecture (see Figure 1) that consists of a client core subsystem running on each compute node, a data collection subsystem running on the collection server, a visualization subsystem running on a graphic workstation, and a modeler system that supports formulation and verification of performance models. A prototype implementation is currently being actively developed.



Figure 1. OpenWLC Architecture

4.1 Client Core Subsystem

The client core subsystem is a lightweight daemon that dynamically loads metric collection modules at initialization and calls the collection routine at a specified interval. The modules return a list of data points to the main daemon. The collected performance data is then encoded and transmitted over the network on a multicast channel to the collection manager (see Section 4.2). The approach taken in implementing the client core subsystem is similar to NWPerf.

4.2 Data Collection Subsystem

In order to avoid a single centralized collection manager that may manifest a hotspot in the system, the collection manager is distributed among several designated nodes in the system. Distributed agents called the sub-collectors are responsible to collect data from a subset of nodes. The sub-collectors in turn report to a centralized collector node that is active and to another standby collector node. The standby node is in place for fault-tolerance and will take over if the active collector fails. The data reported by the sub-collectors to the collector is

aggregated and replicated. This hierarchical structure of client nodes reporting to sub-collectors that aggregate data and report to collector reduces the load on the collector node and also provides for better data aggregation.

The client nodes report to specific pre-assigned packet gatherers in the corresponding sub-collectors. The packet gatherers in turn supply the data to aggregation managers. The aggregation managers can be plugged with several aggregation algorithms such as maximum, minimum, average, sum and other user-defined functions.

The aggregated data from the sub-collectors are reported to the collector node and stored in the database.

4.3 Visualization Subsystem

The visualization subsystem is responsible to collect data in the raw format from the database in the collector and project data in any required form. The visualization subsystem interacts with the collector using a data subscription service that abstracts the collection system to the outside world. The data requests from the visualizer are queued at the collector and serialized. Similar to the aggregation manager, the visualizer is also

capable of providing different views of the system such as system monitor view, application monitor view, data analysis view, etc.

4.4 Modeler Subsystem

Performance monitoring of large-scale systems creates a dilemma. There exists a genuine need to collect detailed information to locate performance bottlenecks. However, collecting this data can introduce serious overheads that would impact the applications' overall performance. Meanwhile, existing monitoring tools imposes users with challenges in dealing with steep learning curve, significant volumes of complex metrics, graphs and tables that require a performance expert to interpret.

OpenWLC aims to address both of these problems, sheer data volume and ease of analysis simultaneously. Our initial investigation is to combine dynamic on-the-fly selection of what performance data to collect with decision support to assist users with the selection and presentation.

The dynamic selection approach permits the system to collect performance data that are relevant to its final analysis. Although this technique may give greater flexibility, it also requires many decisions with regards to what to collect and how to display the aggregated data. On the other hand, the decision-making process can simply be formulated as a data-mining problem. In particular, we are searching for: 1) how to recognize that the application is performing poorly, 2) how to identify the bottleneck, and 3) how to locate the problem when it occurs. To recognize that an application is performing poorly, one can include hypotheses about potential bottlenecks in an application. Then, relevant performance data must be extracted to test the validity of the hypothesis. If the hypothesis is true, one needs to identify a specific resource or a class of resources that cause the poor performance. Finally, performance problem should be located by continuously refining the aforementioned three "howto's" at each iteration step.

Although our search model executes dynamically at runtime, it will incorporate post-mortem semantics. This means that programmers can view their search as if it took place after the program had completed execution, and the data necessary to answer their performance questions has been collected.

Usability and portability are also to be considered. Programmers should not be forced to adopt a specific programming model. They also should not be required to manually instrument their application to collect performance data. The only exception is to re-compile an application. To permit users to quickly locate the cause of the bottleneck without having to look at extraneous details, our search model starts from a high level view and iteratively refines the detail about what is causing the program to perform poorly. Users can independently refine the three "howto's" of an application's performance. The search process can be seen as exploring a three-dimensional space, and at each step a movement in any direction is allowed. Finally, our search model should work for a variety of machine architectures and programming styles. This is a particularly challenging goal due to the diversity of machines.

In summary, the modeler subsystem is based on a grand vision of the search model as described above. The model subsystem is still in the embryonic stage. The details of the search model and its implementation status will be discussed in another paper [16].

5 Summary and Status

This paper describes the requirement, design and current status of a workload characterization tool, known as OpenWLC, for large computing systems. The aims of OpenWLC are 1) an analysis framework, scalable to 100,000 processors, for continuously collecting and aggregating performance information at every level of system's hierarchy, 2) a low-overhead collection protocol for coping with large amounts of monitored data, in the order of Terabytes, 3) not affect the user application performance by greater than 1% on the average, 4) an integrated runtime environment for both performance visualization and modeling. The extensibility of OpenWLC has been achieved through a modular design and an efficient implementation. A prototype implementation is currently being implemented and the status can be found in [16].

6 Acknowledgements

This project is supported in part by the Department of Defense HPC Modernization

Office, and the Advance Computing Research Testbed, Department of Energy. The authors also wish to thank Ryan Mooney, PNNL, for his expertise and discussion.

References

1. R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.
2. G A N G L I A project page. <http://ganglia.sourceforge.net/T>.
3. Roney, T., A. Bailey, and J. Fullop, Cluster Monitoring at NCSA. 2nd LCI Intl. Conference on Linux Clusters, 2001.
4. R. Minnich and K. Reid. Supermon: High performance monitoring for linux clusters. In *The Fifth Annual Linux Showcase and Conference*, Nov. 2001.
5. Hawkeye project page. <http://www.cs.wisc.edu/condor/hawkeye/>
6. Mooney, R. et al. NWPerf: A System Wide Performance Monitoring Tool Poster Session 31, Supercomputing 2004, Pittsburg, PA.
7. A. King and R. Hunt. Protocols and architectures for managing TCP/IP network infrastructures. *Computer Communications*, 23(16):15581572, Sept. 2000.
8. Puliafito and O. Tomarchio. Using mobile agents to implement flexible network management strategies. *Computer Communications*, 23(9):708719, Apr. 2000.
9. R. Subramanyan, J. Miguel-Alonso, and J. A. B. Fortes. A scalable snmp-based distributed monitoring system for heterogeneous network computing. In *Proceedings Supercomputing 2000*, Dallas, Texas, USA, Nov. 2000. IEEE Computer Society.
10. J. M. Anderson, W. E. Weihl, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, and C. A. Waldspurger. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, 15:357390, Nov. 1997.
11. R. Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software Practice and Experience*, 30:117, June 2000.
12. B. Tierney, B. Crowley, D. Gunter, J. Lee, and M. Thompson. A monitoring sensor management system for Grid environments.
13. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
14. P. Uthayopas and S. Phatanapherom. Fast and scalable realtime monitoring system for Beowulf clusters. *Lecture Notes in Computer Science*, 2131, 2001.
15. Ibeaus Bayucan, Robert L. Henderson , *et al*, "Portable Batch System External Reference Specification", MRJ Technology Solutions, May 1999.
16. The openWLC project web site- <http://www.openwlc.org>