Design for Run-Time Monitor on Cloud Computing*

Mikyung Kang[1], Dong-In Kang[1], Mira Yun[2],
Gyung-Leen Park[3], and Junghoon Lee[3,**]

[1] University of Southern California – Information Sciences Institute, VA, USA
[2] Dept. of Computer Science, The George Washington University, Washington DC, USA
[3] Dept. of Computer Science and Statistics, Jeju National University, Jeju, South Korea
{mkkang,dkang}@isi.edu, mirayun@gwu.edu,
{glpark,jhlee}@jejunu.ac.kr

Abstract. Cloud computing is a new information technology trend that moves computing and data away from desktops and portable PCs into large data centers. The basic principle of cloud computing is to deliver applications as services over the Internet as well as infrastructure. A cloud is the type of a parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources. The large-scale distributed applications on a cloud require adaptive service-based software, which has the capability of monitoring the system status change, analyzing the monitored information, and adapting its service configuration while considering tradeoffs among multiple QoS features simultaneously. In this paper, we design Run-Time Monitor (RTM) which is a system software to monitor the application behavior at run-time, analyze the collected information, and optimize resources on cloud computing. RTM monitors application software through library instrumentation as well as underlying hardware through performance counter optimizing its computing configuration based on the analyzed data.

Keywords: Run-Time Monitor, Cloud Computing, QoS, library instrumentation, performance counter.

1 Introduction

Cloud computing is a new information technology trend that moves computing and data away from desktops and portable PCs into large data centers. The basic principle of cloud computing is to deliver applications as services over the Internet as well as infrastructure. A cloud is the type of a parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically

* This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency (NIPA-2010-C1090-1011-0009)) and OPERA Software Architecture Project.
** Corresponding author.

T.-h. Kim, A. Stoica, and R.-S. Chang (Eds.): SUComS 2010, CCIS 78, pp. 279–287, 2010.

provisioned and presented as one or more unified computing resources. The large-scale distributed applications on a cloud require adaptive service-based software, which has the capability of monitoring the system status change, analyzing the monitored information, and adapting its service configuration while considering tradeoffs among multiple QoS features simultaneously.

Recently, multi-core and many-core architectures are becoming more and more popular due to diminishing returns from traditional hardware innovations such as caching and deep pipeline architectures. With more cores in a processor, it is easier to get performance gains by parallelizing applications than traditional approaches. In addition, traditional processors consume large amounts of power to achieve high performance by using high frequencies. By using multiple cores at a lower frequency, and consequently lower voltage, multi-core architectures can achieve higher performance with lower power consumption.

There have been many multi-core processors from commercial vendors [1][2][3]. Among them, Tilera Corporation offers three processor families with the largest number of cores on a general-purpose chip available on the market [4]. Boeing has developed a processor called MAESTRO, to be used in space, based on the first Tilera processor, TILE64 [5]. The TILE64 has 64 cores on a chip. Each core has a three-instruction-wide Very Long Instruction Word (VLIW) pipeline, memory management unit, L1 and L2 cache, so each core itself is a complete processor, which can run a complete operating system like Linux (although more commonly, a single operating system instantiation is used to control multiple cores).

We are to design Run-Time Monitor (RTM) which is a system software to monitor the characteristics of applications at run-time, analyze the collected information, and optimize resources on cloud node which is consisted of multi-core processors. The rest of the paper is organized as follows. In Section 2, the system architecture is briefly described. Our proposed Run-Time Monitor is described in Section 3. Implementation result is described in Section 4, and Section 5 concludes the paper.

2 System Architecture

Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) project began from California University at Santa Barbara, and mainly was targeting at building a private open-source cloud platform [6]. Now Eucalyptus is an open-source implementation of Amazon EC2 (Elastic Compute Cloud) and compatible with most business interfaces [7][8]. Eucalyptus is an elastic computing structure that can be used to connect the user's programmers to the useful systems and it is an open-source infrastructure using clusters or workstations implementation of elastic, utility, and cloud computing. Figure 1 demonstrates the topology structure of Eucalyptus resources. In this figure, the node controller is a component running on the physical resources. On each node, all kinds of entities of virtual machines can run. Logically connected nodes form a virtual cluster, and all nodes belonging to the same virtual cluster receive a command from the cluster controller and then report to the same controller. Parallel HPC applications often need to distribute large amounts of data to all compute nodes before or during a run [9]. In

a cloud, these data are typically stored in a separate storage service. Distributing data from the storage service to all compute nodes is essentially a multicast operation. This paper targets multi-core processor with a single node controller on each node. After receiving data and command, each node processes data while monitoring performance and optimizing resources. And then it returns the result to the cluster controller.

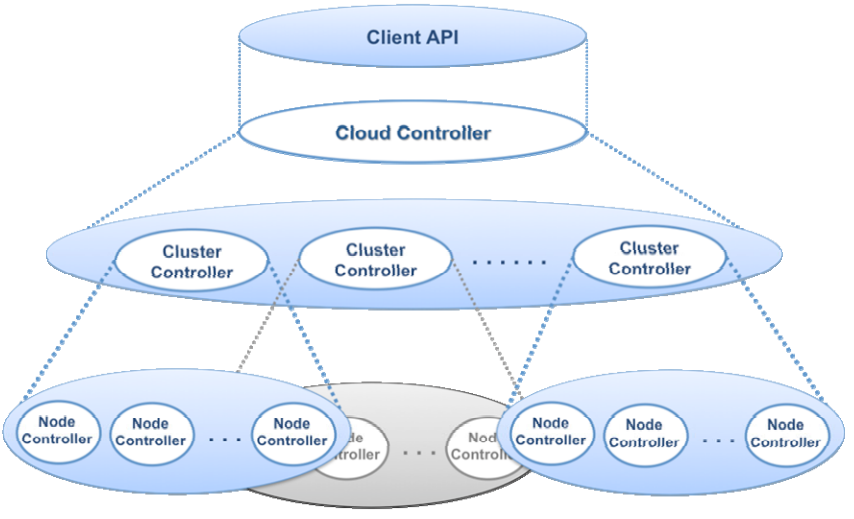

Fig. 1. The resource topology structure of Eucalyptus

3 Run-Time Monitor (RTM)

Run-time monitor is a system software to monitor the characteristics of applications at run-time. As shown in Figure 2, RTM monitors both application software and hardware. We implemented RTM through library instrumentation for software information and through Perfmon2/PAPI for hardware information. The collected software and hardware information can be used by Parallel Performance Analysis Tool and Run-Time system.

RTM implementation targets multi-core processors such as Tilera's TILE64 processor or MAESTRO processor. The implementation has been developed on top of a modified version of libraries such as MPI [10], pthread, iLib [5] and so on.

For the dynamic linking at runtime via preload, we exploited library interposition with the library instrumentation as shown in Figure 3. The interposition is a technique that allows an additional function to be automatically called whenever a library function is called. In RTM model, an interposed library layer was added so that original library modification/recompilation is not needed, no source is needed for anything, and no recompilation/redo on library version update (only on API change). So in each interposed library, needed information was collected after calling unmodified binary library.

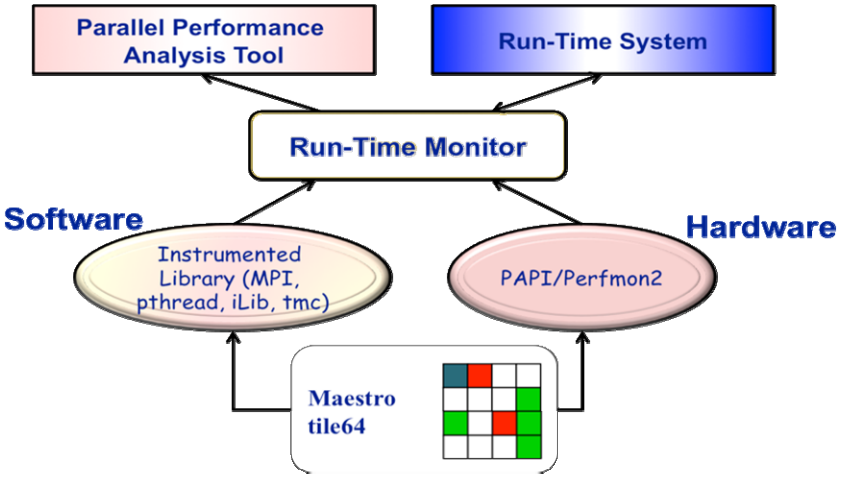

Fig. 2. Run-Time Monitor Overview

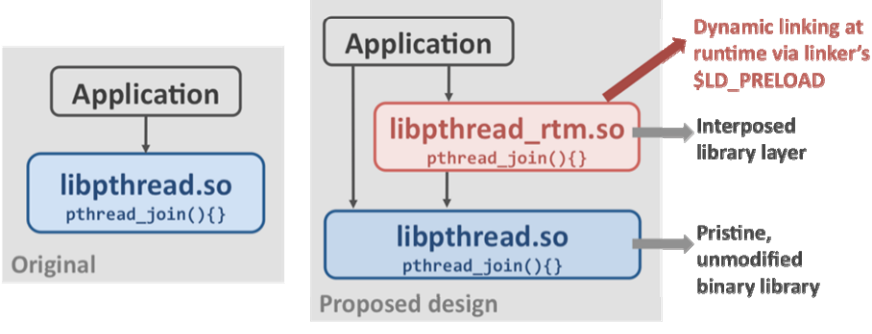

Fig. 3. Library Interposition

Let's see this sample interposed call, pthread_join, in Figure 4. In the application, pthread_join is called. Then, in the interposed library layer, unmodified original library is called and the return value was saved. The *dlsym* is a routine that gives the user direct access to the dynamic linking facilities. The *dlsym* allows a process to obtain the address of a symbol defined within a shared object previously opened by *dlopen*. If handle is *RTLD_NEXT*, the search begins with the *next* object after the object from which *dlsym* was invoked. The *pthread_join* is the symbol's name as a character string. And then other information needed for RTM was calculated, set, and saved in FIFO system. After saving necessary information, interposed library sends the information to the RTM server at the instrumentation layer. Basic library functions and inline/macro functions were instrumented for the interposition.

The RTM server and client concept is illustrated in Figure 5. After loading the RTM software and hardware servers, RTM client program begins to run. Whenever an event happens on each tile, RTM client sends the information to the RTM server

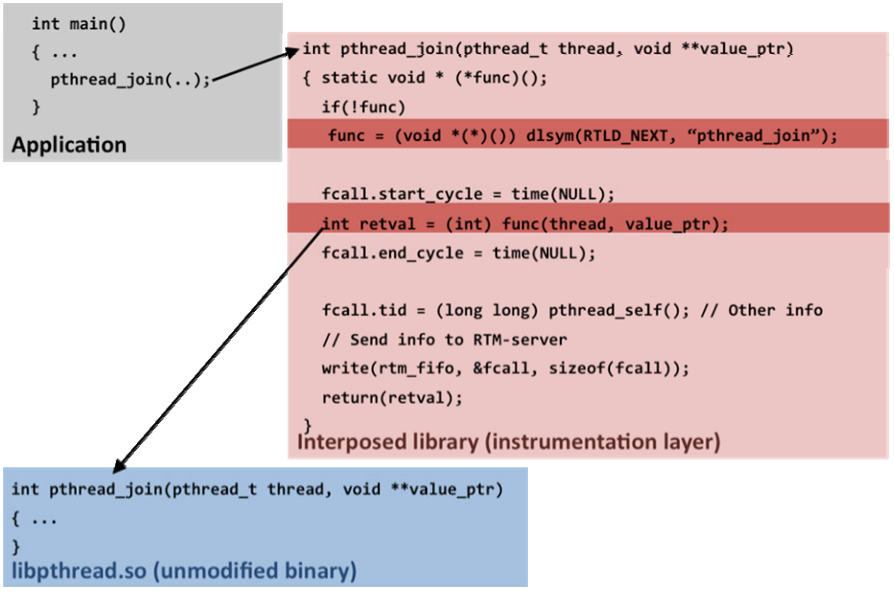

Fig. 4. Sample Interposed Call

using system FIFO. Then the RTM software server calculates the communication pattern and synchronization information providing task graph and synchronization graph XML files periodically. The RTM hardware server also collects hardware information from hardware client on each tile through Perfmon2.

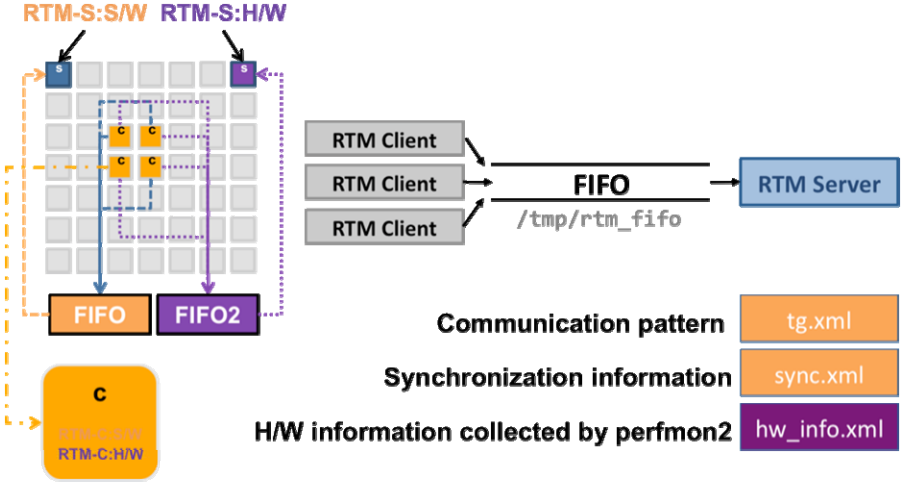

Fig. 5. RTM Server and Client

In the message passing model, source and destination rank and tile location, transferred data amount, timestamp for each event are saved whenever an event happens and then collected by monitoring tile periodically according to the pre-defined interval. Using statistical results for each (source, destination) pair/process calculated by the RTM software server, the user can know that task dependency and the load. At the hardware level, Perfmon2 counter values were collected on each tile, transferred to RTM hardware and then provided as an XML file. The performance counters such as ONE, MP_BUNDLE_RETIRED, TLB_EXC, HIT, L2_HIT, MP_DATA_CACHE_STALL, MP_INSTRUCTION_CACHE_STALL, MISS_I, MISS_D_RD, and MISS_D_WR, were used for the hardware information. This information is collected by way of multiplexing on each tile and sent to the RTM hardware server.

In the shared memory model, the event name (Barrier/Mutex/Lock/Conditional events), the number of occurrences for each event, Max/Min/Ave time of each thread/process for each event, process/pthread ID, and the address of each event group are saved whenever an event happens and then collected by monitoring tile periodically according to the pre-defined interval. Using statistical results for each process/pthread/event, the user can know the synchronization information. At the hardware level, it has no difference from the message passing model.

4 Implementation

The 64 cores are interconnected with mesh networks, while each processor executes at up to 866 MHz to achieve up to 443 billion operations per second. For supporting several libraries, we implemented MPI on the TILE64/MAESTRO based on MPI 1.2 specification. Also Perfmon2/PAPI was ported on multi-core architecture. Figure 6 depicts the RTM eclipse plug-in which can be used for analyzing the periodic hardware and software results on TILE64 or MAESTRO.

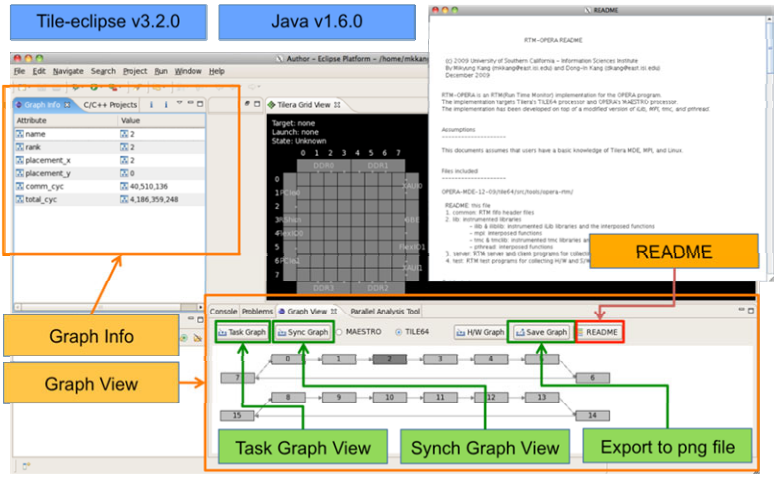

Fig. 6. RTM Graph View and Graph Info

Figure 7 shows the snapshot of the task graph for the communication pattern. As we mentioned in the previous section, the graph view and the related informaiton can be provided. We can know the event's source, destination, total count, data amount, distance, sender cycles, time, CPW (Cyles Per Word), bandwidth, and receiver cycles, time, CPW, bandwith.

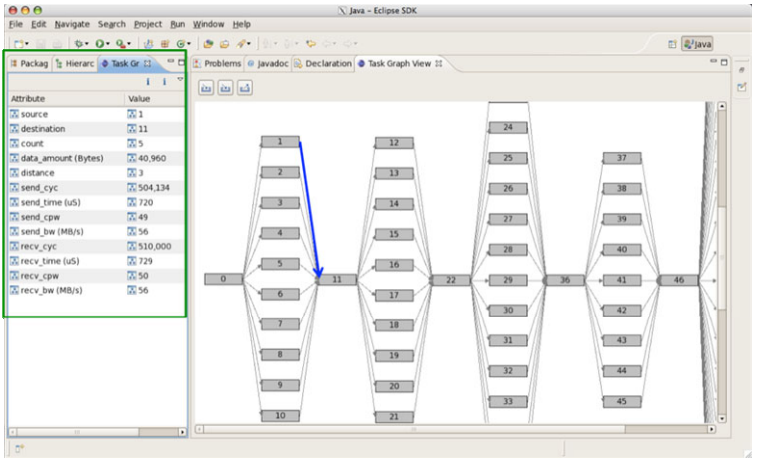

Fig. 7. RTM Task Graph

Figure 8 shows the snapshot of the sync graph for the synchronization information. The first and second images show the result when the <link to the event / event> is selected. We can know the rank, the number of occurrences, maximum/minimum/ average cycles, and the average execution time between each event.

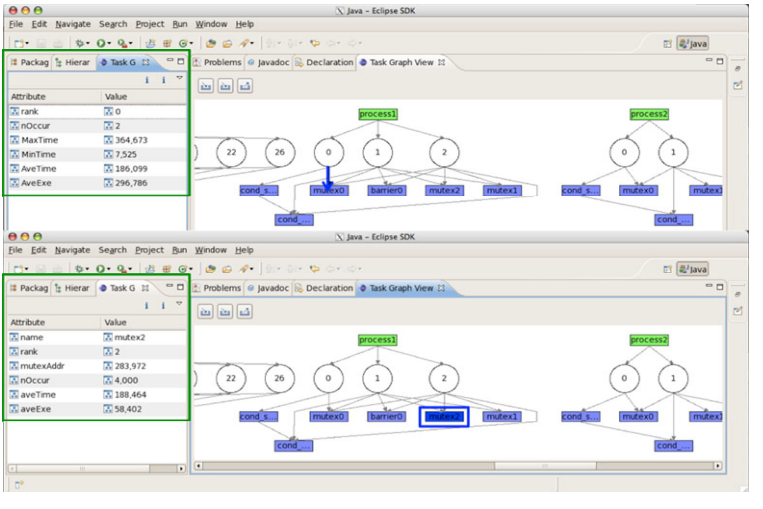

Fig. 8. RTM Sync Graph

The RTM hardware server gathers performance counter information and provides in a XML file. Using this XML file, user can know the current status which tile is running.

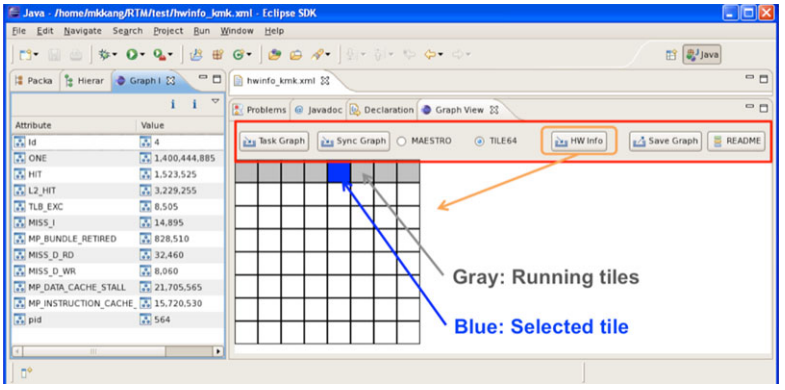

Fig. 9. RTM Hardware Info Graph

5 Conclusions

In this paper, we have designed Run-Time Monitor which is a system software to monitor the characteristics of applications at run-time, analyze the collected information, and optimize resources on cloud computing. RTM monitors both application software through library instrumentation and underlying hardware through performance counter optimizing its computing configuration based on the analyzed data. For future work, we are planning to develop a dynamic run-time self-morphing software framework on multi-core systems. It is expected to provide a framework for an automated optimization of the software with minimal overhead on multi-core systems. After all, the key feature of our framework is that 1) performance monitoring is detached from the applications to system-wide run-time manager, 2) application has a range of morphing at run-time, 3) run-time manager monitors the application's performance and morphs the application at run-time for either better performance or adapting to situation.

References

1. Feng, W., Balaji, P.: Tools and Environments for Multicore and Many-Core Architectures. IEEE Computer 42(12), 26–27 (2009)
2. Schneider, S., Yeom, J., Nikolopoulous, D.: Programming Multiprocessors with Explicitly Managed Memory Hierarchies. IEEE Computer 42(12), 28–34 (2009)
3. Multi-core processor (2009), http://en.wikipedia.org/wiki/Multi-core_processor

4. Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J., Mattina, M., Miao, C.-C., Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J., Zook, J.: TILE64 Processor: A 64-Core SoC with Mesh Interconnect. In: Proc. IEEE International Solid-State Circuits Conference (ISSCC), pp. 88–98 (2008)
5. Tilera Corporation (2010), http://www.tilera.com/
6. http://www.eweek.com/c/a/Cloud-Computing/Eucalyptus-Offers-OpenSource-VMwareBased-Cloud-Platform-100923/
7. Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W., Li, Q.: Comparison of Several Cloud Computing Platforms. In: International Symposium on Information Science and Engineering, pp. 23–27 (2009)
8. Hill, Z., Humphrey, M.: A Quantitative Analysis of High Performance Computing with Amazon's EC2 Infrastructure: The Death of the Local Cluster? In: 10th IEEE/ACM International Conference on Grid Computing, pp. 26–33 (2009)
9. Chiba, T., Burger, M., Kielmann, T., Matsuoka, S.: Dynamic Load-Balanced Multicast for Data-Intensive Applications on Clouds. In: 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, pp. 5–14 (2010)
10. Message Passing Interface Forum, MPI: A Message Passing Interface Standard (2009), http://www.mpi-forum.org/docs/