ABC: Dynamic Configuration Management for MicroBrick-based Cloud Computing Systems

Junghi Min, Hyungwoo Ryu, Kwanghyun La
SW R&D Center, Device Solutions, Samsung Electronics Co., Ltd.
{junghi.min, hyungwoo.ryu, nala.la}@samsung.com

Jihong Kim
Dept. of CSE, Seoul National University,
jihong@davinci.snu.ac.kr

ABSTRACT
In designing a high-performance cloud computing platform, it is important to support diverse system resource requirements of various cloud computing services/applications in a scalable fashion. In this poster, we propose an intelligent middleware for our prototype cloud computing system which automatically changes configurations of modules for high performance under varying cloud service/application workload. Our initial evaluation results show that efficient resource management during run time is a key enabling technique for developing high-performance cloud computing systems.

Categories and Subject Descriptors
C.1.4 [Parallel Architectures]; C.5.5 [Servers]

General Terms Management, Performance

Keywords
Middleware, Cloud System, Optimization

1. INTRODUCTION
In order for a cloud computing platform to support different types of applications efficiently, a cloud computing platform should be able to change quickly its configurations (possibly at multiple system abstraction levels) over changing resource requirements of various applications. As an initial attempt to build such a highly-flexible cloud computing platform, we have proposed MicroBrick [1], a basic building node for constructing a cloud computing platform, and built a prototype cloud computing platform, called MiBiP (MicrobBrick-based Big Data Platform). In this poster, we describe our early experience of optimizing MiBiP, focusing on the automatic resource configuration aspect of MiBiP. In particular, we propose an intelligent middleware for our MicroBrick-based cloud computing systems, called AutoBrickChanger (or, in short, ABC), which automatically changes configurations of MicroBricks for high performance under varying cloud service/application workload.

2. OVERVIEW of MiBiP
Figure 1 shows an overview of our prototype MiBiP. The current version of MiBiP consists of three boards within its chassis and each board has three MicroBricks. Three boards within MiBiP are connected via an Ethernet network. Each MicroBrick (as indicated by a red box within the board picture) has sixteen PCIe ports which are connected to the common PCIe switch. Each PCIe can be connected to a computing module or a storage module.[1] Since all modules are connected to each other via the common PCIe switch, very flexible intra-MicroBrick communications can be supported.

By changing the number of computing modules and storage modules for MicroBricks, different types of MicroBricks can better match to diverse resource requirements of cloud services. Furthermore, by dynamically changing groupings of computing modules and storage modules using a high-performance scalable PCIe switch, a MicroBrick achieves a high degree of the flexibility required by cloud storage systems. Although MiBiP can adapt to changing resource requirements, it is a key challenge in MiBiP to decide when and how to change configurations of each MicroBrick. A support for flexible dynamic configuration and resource management is the main responsibility of ABC.

3. RESOURCE MANAGEMENT in MiBiP
In order to realize the maximum performance potential of MiBiP, our proposed ABC layer needs to make two key decisions on a given cloud workload. First, ABC must understand the key requirements of the current cloud workload so that it can quickly decide the best configuration for each MicroBrick in MiBiP. Second, for a given configuration of a MicroBrick, ABC should decide how local resource (of the given MicroBrick) can be best utilized. In order to satisfy two key requirements of ABC, the proposed ABC was implemented in a hierarchical fashion. Figure 2 shows an overview of the proposed ABC for MiBiP. The ABC layer is divided into two sub-layers, Global ABC (G-ABC) and Local ABC (L-ABC). The G-ABC sub-layer is responsible for managing each MicroBrick's configuration while the L-ABC sub-

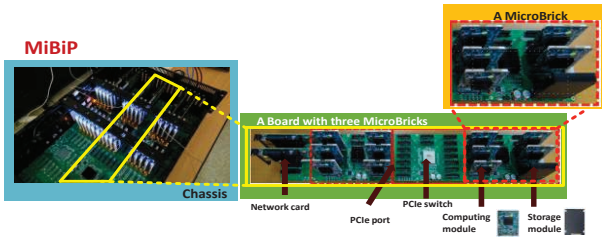


Figure 1: An Overview of MiBiP.

[1] Each computing module consists of a Quad Cortex A15 1.7-GHz ARM processor and a 256-GB mSATA SSD. A 400-GB NVMe SSD is used as a main shared storage module. A computing module of a MicroBrick runs Linux kernel v3.10.

Middleware'14: Demos and Poster, December 08 – December 12 2014, Bordeaux, France
ACM 978-1-4503-3220-0/14/12.
http://dx.doi.org/10.1145/2678508.2678521

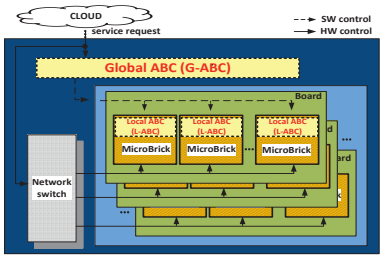


Figure 2: An organizational overview of ABC for MiBiP.

layer focuses on supervising resource managements within a given MicroBrick.

3.1 Global Configuration Management

One of the key resource management issues of MiBiP is how to configure each MicroBrick with computing modules and storage modules to meet diverse resource requirements of cloud services/applications. Since we are currently developing a specific configuration management policy of G-ABC, which takes into account of various workload characteristics, in this section, we present our early evaluation results on the impact of MicroBrick configurations on the system performance.

In our preliminary evaluation of MiBiP, we used HiBench benchmark [2] and the YCSB benchmark [3] with Cassandra (Distributed DBMS). The workloads of YCSB are described in Table 1. As shown in Figure 3, for Wordcount, configuration C1_H, C1, C2 show the lowest elapsed time, however, for Sort, the configuration C2 shows the best. Likewise, for Workload A, the configuration C1 shows the best throughput, however, for Workload B, the configuration C3 works the best. System performances for the same test case vary according to configurations. From this preliminary experimental result, we can see that there is no single best MicroBrick configuration for all test cases, and it is critical for G-ABC to change the configuration of each MicroBrick in an automatic fashion over varying resource requirements of cloud applications.

3.2 Local Resource Management

Once a MicroBrick's configuration is fixed by G-ABC, each MicroBrick needs to be optimized for a given configuration for high performance. For example, we observed that, for Cassandra, a default work distribution policy of Linux works so poor that no meaningful work progress is ever made after a short initial active interval. As shown in Figure 4 (a), four CPU cores are all actively utilized in the initial execution interval. However, in the second interval, only CPU core 1 is actively working while the rest of CPU cores are virtually idle. This specific case illustrates a strong need for local resource management within a MicroBrick. The main function of L-ABC is to fine-tune various performance impacting local parameters (such as a work distribution policy) within a given MicroBrick. Dynamic resource allocation is a well known problem [4]. As an optimization case, we developed a new work distribution policy which works considerably better than the

Table 1: Workloads of YCSB.

Workloads	Functional Ratio	Characteristics	Example
Workload A	Read/Update ratio 50/50	Update-heavy	Session store
Workload B	Read/Update ratio 95/5	Read-mostly	Photo tags
Workload C	Read/Update ratio 100/0	Read-only	User profile cache
Workload D	Read/Update/Insert ratio 95/0/5	Read-latest	User status updates
Workload E	Scan/Insert ratio 95/5	Short ranges	threaded conversations
Workload F	Read/Read-modify-write ratio 50/50	Read-modify-write	User database

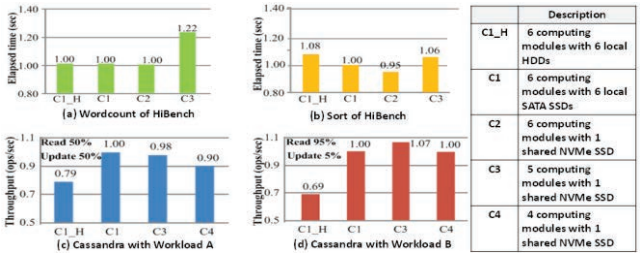
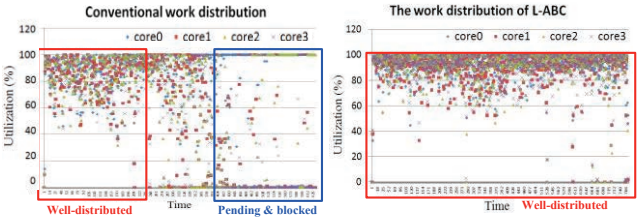


Figure 3: Performance variations under different configurations.



(a) Linux (v3.10) **(b) L-ABC**

Figure 4: A comparison of the CPU utilization.

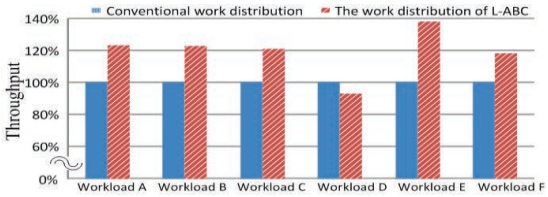


Figure 5: A comparison of the system throughput.

default policy when there are a large number of works that need to be distributed. Our new work distribution policy considers more system status information in distributing tasks among multiple cores. When an interrupt comes in, for example, our new policy checks the numbers of waiting works in the work queue, the number of active CPU cores, and the average per-CPU load. Figure 4 illustrates that the per-core CPU utilization is significantly improved under our new policy over the default work distribution of Linux. For our YCSB cloud workloads (described in Table 1), our new policy alone improves the system throughput by up to 38% as shown in Figure 5.

4. CONCLUSION

From our initial evaluation results of a MicroBrick-based cloud computing system, we strongly believe that efficient resource management during run time is a key enabling technique for developing a high-performance cloud computing systems. We are currently focusing on developing a more intelligent middleware including G-ABC and L-ABC.

5. REFERENCES

[1] Min, J., Min, J., La, K., Roh K., and Kim, J., MicroBrick: A Flexible Storage Building Block for Cloud Storage Systems, in Proc. of the USENIX Conference on File and Storage Technologies, Poster, 2014.

[2] HiBench. https://github.com/hibench.

[3] YCSB. https://github.com/brianfrankcooper/YCSB.

[4] Zhang, Y., and West, R., Process-Aware Interrupt Scheduling and Accounting, in Proc. of the IEEE International Real-Time Systems Symposium, 2006.