Procedia Computer Science
Volume 29, 2014, Pages 576–585
ICCS 2014. 14th International Conference on Computational Science

The K computer Operations: Experiences and Statistics

Keiji Yamamoto[1], Atsuya Uno[1], Hitoshi Murai[1], Toshiyuki Tsukamoto[1], Fumiyoshi Shoji[1], Shuji Matsui[2], Ryuichi Sekizawa[2], Fumichika Sueyasu[2], Hiroshi Uchiyama[2], Mitsuo Okamoto[3], Nobuo Ohgushi[3], Katsutoshi Takashina[3], Daisuke Wakabayashi[4], Yuki Taguchi[5], and Mitsuo Yokokawa[6,1]

[1] Advanced Institute for Computational Science, RIKEN, Japan
[2] Fujitsu Limited, Kawasaki, Japan
[3] Fujitsu Systems West Ltd., Osaka, Japan
[4] FUJITSU SSL Ltd., Kawasaki, Japan
[5] SAKURA KCS Corp., Kobe, Japan
[6] Kobe University, Kobe, Japan

Abstract

The K computer, released on September 29, 2012, is a large-scale parallel supercomputer system consisting of 82,944 compute nodes. We have been able to resolve a significant number of operation issues since its release. Some system software components have been fixed and improved to obtain higher stability and utilization. We achieved 94% service availability because of a low hardware failure rate and approximately 80% node utilization by careful adjustment of operation parameters. We found that the K computer is an extremely stable and high utilization system.

Keywords: Supercomputer, Job Characterization, Statistics of Failures, Language and Parallelization

1 Introduction

Supercomputers are indispensable tools for solving several social issues by conducting computer simulations in various fields of science and engineering, and are often used in academic research and in industry. Therefore, many countries focused their efforts on developing supercomputers to maintain national competitiveness.

Development of large-scale supercomputers is undoubtedly important; however, the management and operation of such supercomputers is a significant concern. Most supercomputers are requested to be stable in operations and to have easy-to-use environments.

More than a year has passed since we released the K computer to users on September 29, 2012. The K computer is a large-scale parallel supercomputer system developed by RIKEN Advanced Institute for Computational Science (AICS) in cooperation with Fujitsu Ltd.[1] It is expected to provide stable operations and high utilization by taking many user requirements into consideration so that it can be used efficiently in various research fields.

doi:10.1016/j.procs.2014.05.052

We have learned a significant amount about operational issues of the K computer since its official release. Some system software components have been fixed and improved to obtain greater stability and utilization. In this paper, we describe the K computer operational experiences and present some statistics with respect to utilization for a year.

2 System and power consumption

2.1 System

The K computer is a distributed memory supercomputer system consisting of 82,944 compute nodes and 5,184 I/O nodes, two kinds of file systems, control and management servers, and front-end servers. Each compute node is primarily composed of a CPU, 16 GB memory, and an interconnect LSI. The CPU is a SPARC64 VIIIfx[2] with 8 cores operating at a clock frequency of 2 GHz and 6 MB of shared L2 cache. Its peak performance and performance per Watt are 128 GFLOPS and 2.2 GFLOPS/Watt, respectively. Each core has four floating-point multiply-and-add execution units, two of which are operated concurrently by SIMD instruction. The Tofu[3] interconnect network is a six-dimensional mesh/torus network used for data communication among compute nodes. The interconnect controller (ICC) is directly connected to a CPU, and each ICC has ten routes connecting ten adjacent nodes. Multiple communication routes can be taken to communicate among nodes. From a programming perspective, one-, two-, or three-dimensional torus network topologies can always be used for a job; e.g., a torus network can be configured dynamically when a job is assigned to a portion of the K computer by indicating the required topology in a job script.

The operating system for nodes is Linux. Fortran, C, and C++ programming languages are provided for users to maintain conventional programming environments. Those compilers can automatically generate binary codes executable concurrently which are carried out by threads within the CPU. The K computer provides a three-level parallel programming model to attain high sustained performance. The first level of parallel processing is SIMD processing in the core, the second level is thread programing on the compute node supported by automatic parallelization or OpenMP[4] directives, and the third level is distributed-memory parallel processing programming with the Message Passing Interface (MPI)[5] library over several compute nodes. The MPI library of the K computer is based on an implementation of OpenMPI[6], in which several functions are specially implemented using native Tofu interfaces to achieve high performance[7]. The job scheduler is the Fujitsu Parallelnavi, which has been specifically optimized for the K computer. Some development tools, including XcalableMP[8] are installed

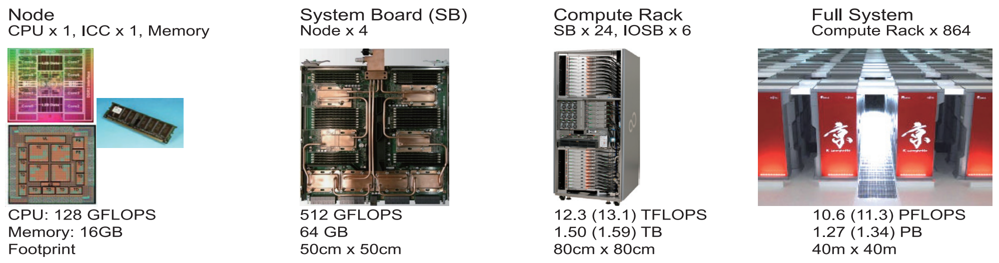

Figure 1: System configuration. (numbers in parentheses denote either total performance or total memory capacity, including those of I/O nodes)

under the support of developers.

Two kinds of file systems, a 30 PB global file system and an 11 PB local file system, are implemented based on the Lustre file system[9]. Users' permanent files are placed on the global file system. If a user submits a job to the K computer and files on the global file system are required to run the job, the files are moved to the local file system before the job starts by staging functions (staging-in). After the job ends, files created during the job on the local file system are moved to the global file system (staging-out). These file-staging operations are managed automatically by the job manager.

The system configuration is shown in Fig. 1. Peak performance, memory capacity, and footprint area of each level are also described. The K computer is designed as a "thin node" system in which a node contains one CPU. Four nodes are mounted on a system board (SB), and 24 system boards are installed into a compute rack. There are a total of 864 compute racks.

Reliability, availability, and serviceability (RAS) were carefully considered in when designing and constructing the K computer. The memory modules have ECC functions, which can correct one-bit errors and recognize two-bit errors. If a one-bit error is detected in the memory module during operation, the memory module is replaced as quickly as possible. When an SB failure occurs, the SB is stopped and replaced with a new SB immediately during operation. This takes approximately two hours. The system keeps working, except for the four nodes on the failed SB. The new SB rejoins normal operation as soon as it is available. Critical components such as a power distributing unit in a compute rack and control and management servers are duplicated and have hot-swap functionality to avoid one-point failures.

2.2 Power consumption

Recent parallel computer systems constructed with a tremendous number of components consume a large amount of electricity even though the power consumption of each component, such as the CPU, has been reduced due to technological development and innovation. The power consumption of large-scale computer systems is currently a significant issue. Power usage effectiveness (PUE) is a measure of how efficiently a computing system uses energy.

We have four buildings, i.e., the computer building, the chillers building, the substation supply building, and the research building. The electricity is fed by a few distribution lines to the floors of the buildings, and the amount of power consumed by each floor can be measured. We have classified electricity distribution lines into four blocks, i.e., "K computer," "air-handling units," "chillers," and "others". Figure 2 shows the average electric power for each block. Total electric power of the AICS facility is approximately 15,000 kW. The electric power of the K computer was approximately 12,000 kW (80% of the total) and has increased gradually over time. We have concluded that applications were highly optimized to the K computer and therefore it consumed more power.

We employ a tool that calculates PUE each hour to analyze the relationship between K computer operations and power consumption. This facilitates improving the management of the facility. Figure 3 shows the electric power and PUE for 60 hours from November 12 to November 14, 2013. PUE changed in the range 1.20 to 1.27 depending on the K computer job load. As can be seen, good PUE is obtained at AICS.

3 Failure statistics

Statistics for hardware and software failures were collected between October 2012 and September 2013. We have classified hardware failures into three groups according to specific hardware

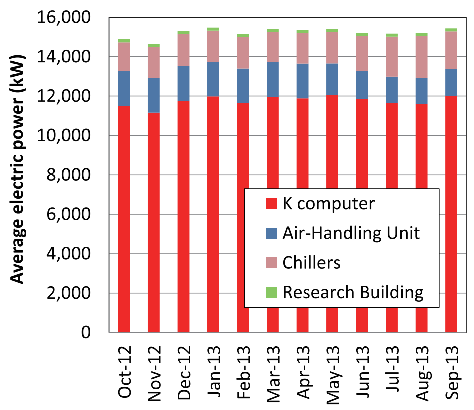

Figure 2: Average electric power between October 2012 and September 2013

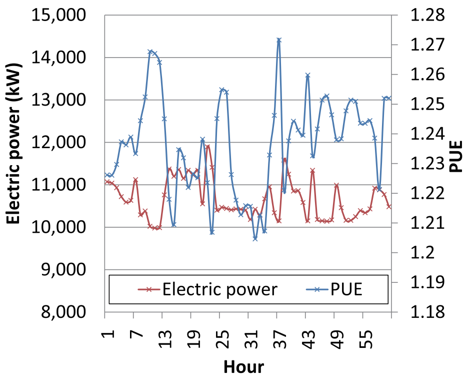

Figure 3: Electric power (red line) and PUE (blue line) for 60 hours from November 12 to November 14, 2013

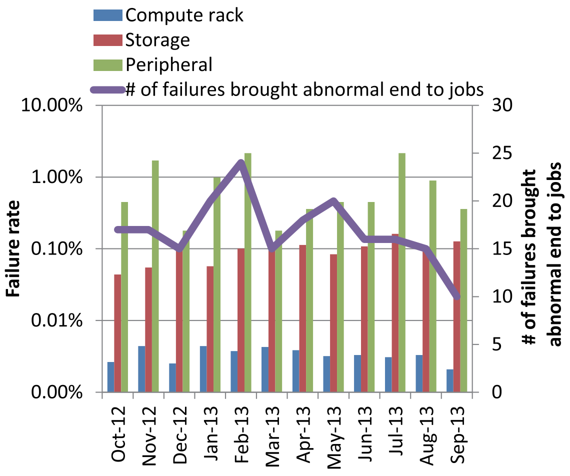

Figure 4: Failure rates in terms of hardware components and the number of failures resulting in abnormal termination of jobs

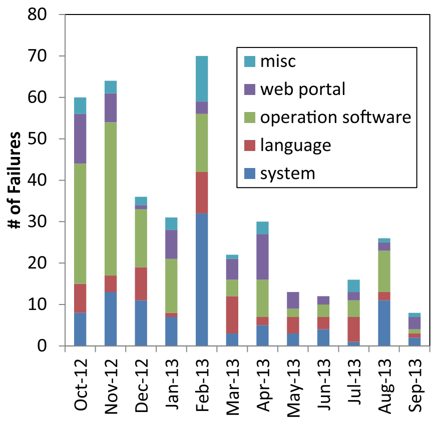

Figure 5: Number of failures in terms of software components

components, i.e., "Compute rack," "Storage," and "Peripheral" groups. The compute rack group includes the failures of parts inside the compute rack such as CPU, ICC, memory module, SB and rack (sensor, PCI interface, etc.). The storage group includes the failures of hard disk drives (HDD) and HDD controllers. The peripheral group includes the failures of parts other than compute rack and storage components such as network switches and servers. Figure 4 shows the failure rate of each group. The solid line denotes the number of failures that resulted in abnormal termination of jobs.

Compute rack system failure were caused by failures in the CPU, ICC, memory module, SB, system disk, etc. Among them, failures affecting running jobs are the failures of the CPU, ICC, and SB. The number of failures was less than 25 each month. It is obvious that the

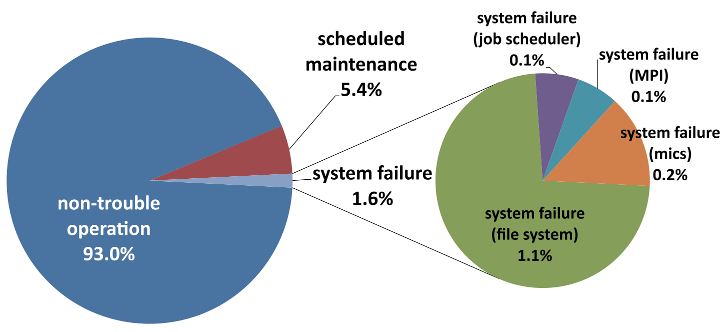

Figure 6: The K computer operation time between October 2012 and September 2013

RAS facilities of the K computer have resulted in a small number of failures. Storage and peripheral failures do not result in serious system down because of the duplication of servers and connection paths, and do not affect running jobs.In addition, the failure rate of compute racks is much less than that of the storage and the peripherals group.

Figure 5 shows the number of software failures. The number of failures for the first five months of official operation was relatively high. However, the number of software failures decreased after some modifications and improvements were applied in March 2013. Major software failures were caused in the system and operating software such as the file system and the job manager.

Figure 6 shows the rate of trouble-free operation time (93.0%), scheduled maintenance time (5.4%), and system failure time (1.6%) for 8,817 hours between October 2012 and September 2013. As can be seen, the system failure time is very short. The pie chart on the right shows a breakdown of system failures that were found in file systems, the job scheduler, MPI libraries, and others. As can be seen, the majority of system failures are caused by the file system. Some file system failures are caused by original bugs in the Lustre file system; the bugs have been reported to the Lustre community.

4 Job statistics

4.1 Job queues and job types

The current scheduling policy gives uses a fair chance for job execution. First-come and first-serve (FCFS) and back-filling scheduling are adopted. Submitted jobs are placed at an appropriate space in node-time scheduling space by the job manager. Compute nodes are divided into resource groups as job queues, and there are three groups, i.e., **Small** for jobs with 1–384 nodes, **Large** for jobs with 385–36,864 nodes, and **Huge** for jobs with 36,865–82,944 nodes. During normal operation, **Small** and **Large** resource groups are available and **Huge** is disabled. The jobs in **Huge** are processed during a designated period (contiguous three days per month).

Depending on the execution pattern, four job types are available: normal jobs, step jobs, bulk jobs, and interactive jobs. A normal job is a general batch job. A step job is a batch job that applies an execution sequence or dependency to jobs. A bulk job is a set of multiple normal jobs that are submitted simultaneously and executed with the different bulk numbers provided. An interactive job is a job that is executed interactively.

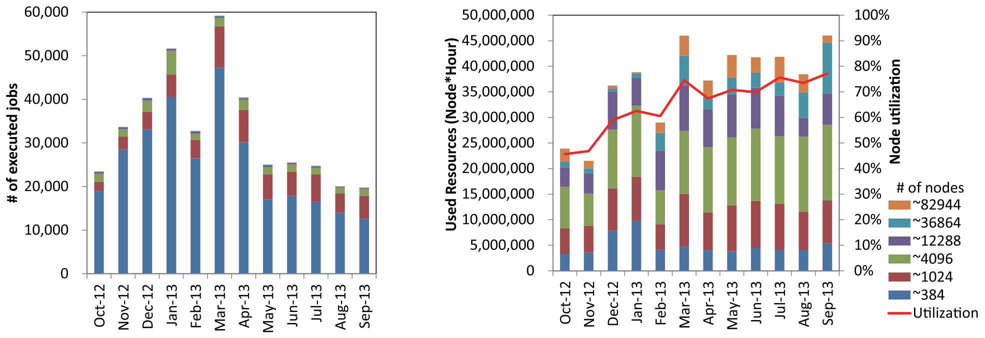

Figure 7: Number of executed jobs (left) and amount of used resources (right)

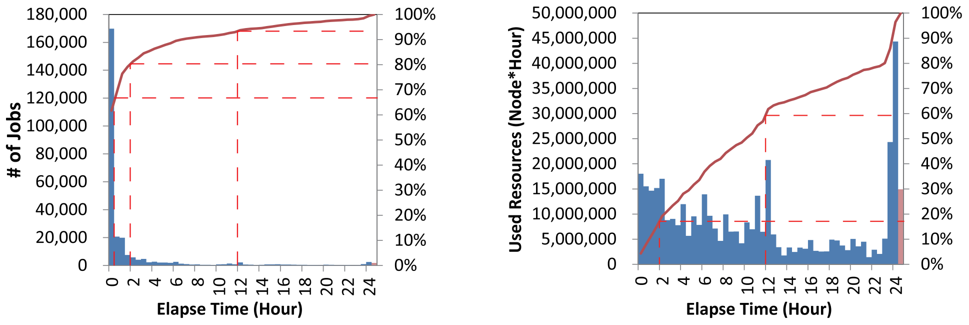

Figure 8: Number of executed jobs (left) and amount of resources used (right) by elapsed job time (rightmost red rectangle denotes jobs over 24 hours elapsed time; solid lines are cumulative values)

4.2 Job characteristics

The characteristics of jobs executed on the K computer have also been analyzed. The knowledge of jobs is important to decide operation policy for job scheduling. In addition, the results of the analysis can help determine optimal parameters for the operating system, file systems, job scheduler, etc. Figure 7 shows the number of executed jobs (left) and the amount of used resources (right) between October 2012 and September 2013. Used resources are the product of the number of used nodes and the elapsed time. The resources used in November 2012, February 2013, and August 2013 decreased compared with the resources used in the respective previous months. System maintenance was conducted in these months; thus, operation time was less than that of other months.

At the beginning of the official release, we provided one job queue for all jobs. We expected that small-size jobs would be processed during gap spaces between large-size jobs in terms of node-time scheduling space; therefore, node utilization had to be high. However, many small-size jobs disturbed the execution of large-size jobs, and node utilization was very low. Therefore, we decided to divide the single job queue into two queues **Small** and **Large** in the middle of February 2013. After this modification, most jobs submitted have become larger and require long time. During the first year of operation, we also tuned scheduling parameters, such as file staging timing and the maximum number of jobs executed simultaneously for each user group,

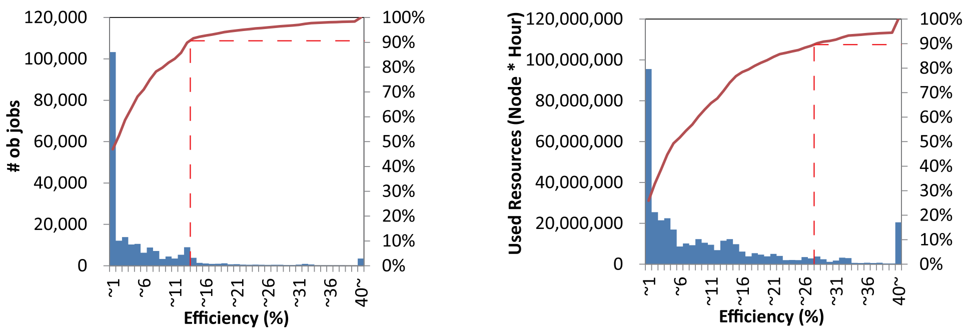

Figure 9: Number of executed job (left) and amount of resources used (right) by job efficiency (solid lines are cumulative values)

and achieved approximately 80% node utilization in September 2013.

We have analyzed job characteristics for approximately 275,000 jobs between October 2012 and September 2013, which used 420,000,000 *Node*Hour* computational resources. Figure 8 shows the number of jobs and used resources at a 30-minute interval of elapsed job time. The short jobs (under 30 minutes) account for 60% of all the jobs. Jobs under two hours account for 80% of all the jobs. However, jobs under two hours used approximately 20% of all computational resources. Although the number of long jobs (over 12 hours) represents 5% of all jobs, these jobs used 40% of all the computational resources. Figure 9 shows the frequency of sustained performance efficiency. Approximately 220,000 jobs, which used 369,000,000 *Node*Hour* computational resources are analyzed. The jobs under 13% efficiency account for 90% of the total number of jobs. 90% of all computational resources are used by jobs that have less than 28% efficiency.

4.3 Job waiting time

Figure 10 shows the average waiting time with respect to job sizes and elapsed times. This figure shows that the waiting time for jobs with longer elapsed time and a larger number of nodes is longer than that of jobs with shorter elapsed time and a smaller number of nodes. This trend represents our scheduling policy, which gives users a fair opportunity for job execution. The average waiting time in March and September was longer than that in other months. Each project was given appropriate compute resources for a year, and the resources were divided into two seasons: from April to September and from October to March of the next year. Users attempt to use up the remaining compute resources at the end of each season and submit many jobs in March and September. Therefore, the waiting time for jobs is longer in those two months.

5 Analysis of languages and parallelization

Information on the invocation of Fortran, C, and C++ compilers was collected between September 28, 2012 and November 30, 2013[1]. This information includes the number of invocations,

[1]Statistics were not logged between March 1, 2013 and May 14, 2013 due to a failure.

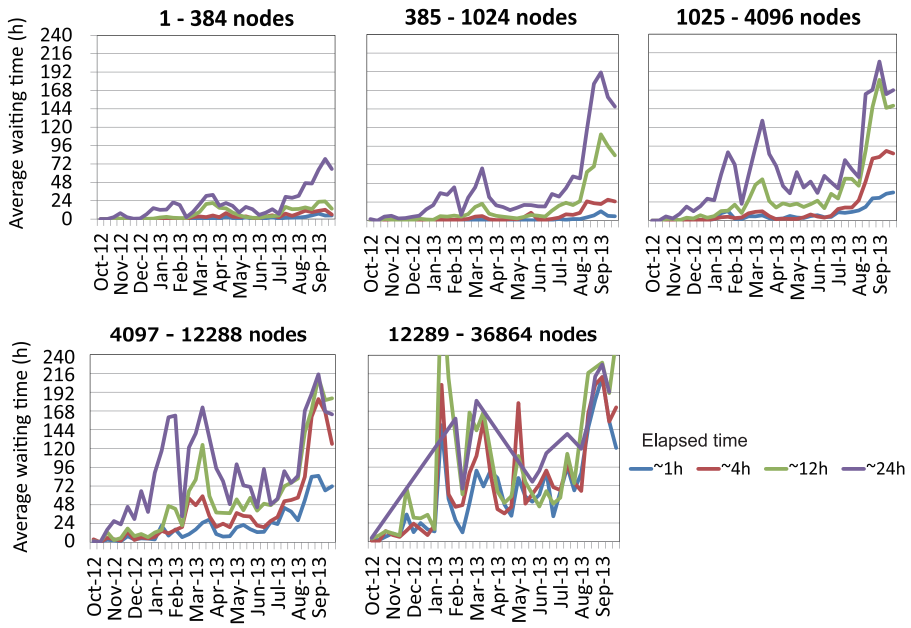

Figure 10: Average waiting time

compiler options specified by users, and linked libraries. In addition, information for each executed job was recorded.

Our final goal was to obtain knowledge on how choice of programming languages and parallelization methods affect the execution performance of programs; however, the information recorded was insufficient for this goal. Thus, we plan to extend the content of the information we collect. In this section, we show some of the results of the analysis based on the existing information.

5.1 Languages

Three programming languages, Fortran, C, and C++, are available on the K computer. Figure 11 shows the rate of invocations for each compiler. It can be seen that Fortran was dominant, although C and even C++ become quite popular in the area of HPC.

5.2 Thread parallelization

The advantage of intra-node thread parallelization can be taken with either OpenMP or a compiler's automatic parallelization. The OpenMP parallelization is invoked with a compiler option "-Kopenmp," and automatic parallelization is invoked with either "-Kparallel" or "-Kvisimpact." The proportions of invocations of each type of thread parallelization are shown in Fig. 12.

Pure OpenMP parallelization is invoked three times more frequently than pure automatic parallelization. Note that it is possible to mix OpenMP and automatic parallelization in a program, and that mixed parallelization is used more frequently than pure automatic parallelization. On the other hand, the case in which no thread-level parallelization is specified is

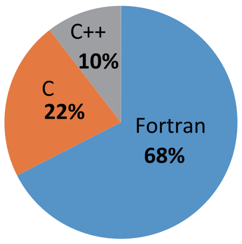

Figure 11: Number of invocations of each language compiler

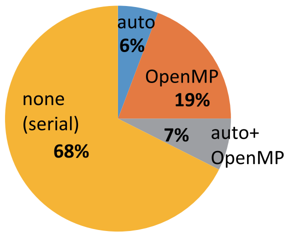

Figure 12: Number of invocations of thread parallelization

much more prominent than cases with any thread-level parallelization.

5.3 Inter-node parallelization

Both "hybrid-MPI" and "flat-MPI" can be used to apply inter-node parallelism. It is recommended that users exploit hybrid-MPI, particularly when their program runs on a large number of nodes, for the following reasons.

- Memory size
  As the number of nodes increases, the size of the communication buffer that the MPI library allocates implicitly and internally also increases.
- Efficiency of collective communications
  The number of steps of iterative algorithms for collective communications increases and efficiency decreases as the number of nodes increase.

We have analyzed the performance of hybrid- and flat-MPI based on the user-specified job parameters in the job scripts. These parameters include the following:

- *node*: the number of nodes allocated for the job
- *shape*: the number of nodes onto which the initial MPI processes are arranged
- *proc*: the number of the initial MPI processes

It can be said that the initial MPI processes are invoked on all nodes allocated for a job when $node = shape$ [2], hybrid-MPI is adopted when $proc = shape$, and flat-MPI is adopted when $proc = shape \times 8$ [3]. Note that, because of the limitations of logging capacities, performance statistics were saved only for the jobs that were neither bulk nor step jobs, were normally finished, and were executed without the performance profiler tools. The number of jobs and the average FLOPS value per node for hybrid- and flat-MPI parallelization between September 28, 2012 and October 31, 2013 are shown in Table 1.

It can be seen that approximately six times as many hybrid-MPI jobs have been executed, and the hybrid-MPI performance is nearly equal to that of flat-MPI, which is contrasts our prediction. We assume that hybrid-MPI has an advantage, particularly in very large-size jobs. However, analysis based on the number of nodes has not clarified this and it remains a future consideration.

[2] If users employ features of dynamic process creation (e.g., MPI_Spawn), then *shape* should be less than *node*.

[3] Note that cases in which just one node are used (serial execution), there is no thread parallelization and "moderate-MPI" (e.g., $2threads \times 4processes$ in a node) is ignored.

Table 1: Performance of inter-node parallelization

	number of jobs	average GFLOPS/node
hybrid	158,568	5.10
flat	25,820	5.11
others	19,717	3.45
total	204,105	4.94

6 Summary

The K computer is one of the most massively parallel supercomputers in the world. After operation began on September 29, 2012, we have faced and solved various issues and failures. In particular, file system failures have caused serious system service down, as was described in Section 3. Such service down rarely occurs in ordinary sized supercomputers. Despite the various troubles and failures described in Sections 3 and 4, we have achieved 94% service availability and approximately 80% node utilization, which are attributed to the low hardware failure rate and the high level RAS functions of the K computer.

The hybrid-MPI programming model is recommended for the K computer. However, according to utilization analysis in terms of language and parallelization, we have observed that many users choose to adopt the "flat-MPI" programming model. For more efficient utilization of the K computer, we must introduce users to hybrid-MPI via user support and training.

In future, we will continue to improve the stability and usability of the system and user environment. We hope that the experiences discussed in this paper will provide useful information for effective operation and utilization of large scale supercomputers.

References

[1] Mitsuo Yokokawa, Fumiyoshi Shoji, Atsuya Uno, Motoyoshi Kurokawa, and Tadashi Watanabe. The K Computer: Japanese Next-generation Supercomputer Development Project. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 371–372, 2011.

[2] Takumi Maruyama, Toshio Yoshida, Ryuji Kan, Iwao Yamazaki, Shuji Yamamura, Noriyuki Takahashi, Mikio Hondou, and Hiroshi Okano. Sparc64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing. *IEEE Micro*, 30(2):30–40, 2010.

[3] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers. *Computer*, 42(11):36–40, 2009.

[4] Leonardo Dagum and Ramesh Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, January 1998.

[5] The MPI Forum: CORPORATE. MPI: A Message Passing Interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 878–883. ACM, 1993.

[6] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, 2004.

[7] Tomoya Adachi, Naoyuki Shida, Kenichi Miura, Shinji Sumimoto, Atsuya Uno, Motoyoshi Kurokawa, Fumiyoshi Shoji, and Mitsuo Yokokawa. The Design of Ultra Scalable MPI Collective Communication on the K Computer. *Comput. Sci.*, 28(2-3):147–155, May 2013.

[8] Specification of XcalableMP, version 1.2: http://www.xcalablemp.org/spec/xmp-spec-1.2.pdf.

[9] Philip Schwan. Lustre: Building a File System for 1,000-node Clusters. In *Proceedings of the LINUX SYMPOSIUM*, page 9, 2003.