Vol.2021-HPC-178 No.13
2021/3/15

Energy Aware Scheduler of Single/Multi-node Jobs Exploiting Node Heterogeneity

Jiacheng Zhou[1] Fukazawa Keiichiro[2] Hiroshi Nakashima[2]

Abstract: Modern CPUs suffer from power efficiency heterogeneity, which can result in additional energy cost or performance loss. On the other hand, future supercomputers are expected to be power constraint. This report focuses on energy aware scheduling algorithms target on two situations. In single-node situation, workload consists of various single-node jobs, Combinatorial Optimization Algorithm saves energy by calculating a local optimal allocation plan with KM algorithm. In multi-node situation, power cap causes load unbalancing in multi-node jobs. Sliding Window Algorithm targets on reducing such unbalancing by sliding window. Proposed algorithms are evaluated in the simulation and real supercomputer environment. In single-node situation, Combinatorial Optimization Algorithm achieved up to 2.92% saving. For the multi-node situation, workload is designed based on real historic workload, and up to 5.36% saving was achieved by Sliding Window Algorithm.

1. Introduction

After a long period of exponential improvement in transistors density, Dennard scaling appeared to break down. According to Dennard scaling, with the density of transistor doubles, CPU frequency increases by 40% and the power reduces by 50%, which means the total power of a chip stays in a level the same and performance per watt grows in the same rate as Moore's Law [1]. However, in recent years, such performance improvement without increasing power consumption becomes harder due to the current leakage and high temperature of transistors at extremely small size. For example, in 2001, NERSC's 3 Tflops HPC system uses less than 400 KW of electrical power[2] while the top petascale system[*1] in 2008 consumes only 2 MW of power, which means 5 times higher power gives us a huge performance improvement of hundreds time. However, in November 2016, Sunway TaihuLight with 93 Pflops reaches 15 MW[3] to exemplify that the growth rate of performance per watt has been going down significantly. Since Dennard scaling is over, the performance improvement means power consumption increasing commensurately. Thus, the power consumption of future supercomputers may be restricted due to the facility capacity[4]. The US Department of Energy has identified the power management as a primary challenge for exascale systems, and set a goal that exascale systems operate under 20 MW[4].

On the other hand, as the size of transistors becomes smaller, processors with the same architecture show randomness in power efficiency between each other[5]. This kind of manufacturing variation causes the different power efficiency of computing nodes in HPC systems, and has been observed in supercomputer systems. Figure 1 shows the distribution of static power[*2] in Kyoto University's supercomputer system B, which is also known as Laurel 2 (Intel Xeon Broadwell, 18 cores, 2.1 GHz). X-axis is the static power of nodes normalized by its average, and y-axis is the number of nodes. From this figure, the static power of more than 800 computing nodes in Laurel 2 shows a large variation of 31% (87%–118%).

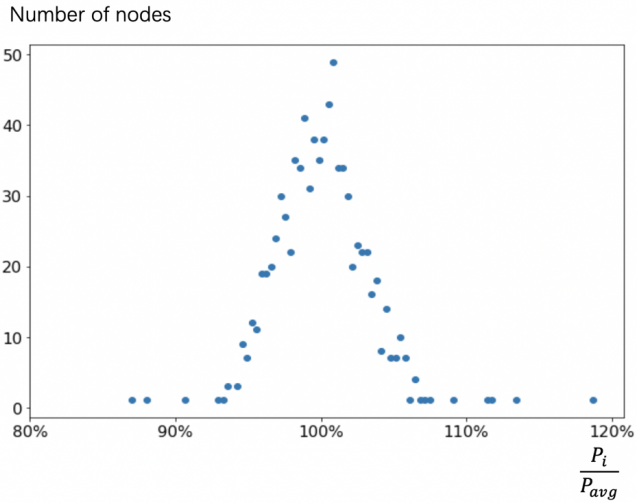

Fig. 1 Distribution of static power consumption in Laurel 2.

There are supercomputers consuming tens of WM at the maximum power in the operation. These systems are used to be designed to endure the maximum power consumption by preparing high power supply and expensive cooling system. However, such a maximum case rarely happens and this designing is continuously wasting energy. In order to reduce the cost, the power cap is another method to prevent the peak power from exceed-

[1] Graduate School of Informatics, Kyoto University
[2] Academic Center of Computing and Media Studies, Kyoto University

[*1] Systems capable for 1 Pflops.

[*2] Power consumed when nodes were idle.

ing the predetermined threshold. One problem of the power cap is that nodes can show significant performance variation under a cap[6] even with the same architecture. This variation is referred to as a node-level power/performance heterogeneity. Most parallel applications are designed to be load-balanced to maximum the performance of all computing nodes. Thus, the node-level performance heterogeneity may cause serious imbalance in parallel applications depending on nodes they are assigned to.

There is also another problem that will cause load imbalance, most applications in supercomputers mainly use part of components, such as CPU, memory and internal network[7]. For example, some computation-intensive applications consume only very little memory and internal network power. With such an intensive use of a particular component, the power/energy consumption of an application is determined by not only the node-level power efficiency but also component-wise efficiency. Thus, applications should be assigned to nodes which can perform better. In supercomputer systems, the node allocation plan is determined by the job scheduler. Thus, to achieve efficient power management, it is necessary to study energy aware scheduling considering node-level heterogeneity and property of jobs.

This report proposes two kinds of energy-aware scheduling algorithms for supercomputer systems. Combinatorial Optimization algorithm (COA) targets on the situation of scheduling different kinds of jobs, such as computation-intensive and memory-intensive jobs, using one node for each (single-node situation). Sliding Window algorithm (SWA) targets on reducing load imbalance caused by the performance heterogeneity among nodes executing the same type jobs under a tight power cap possibly using multiple nodes (multi-node situation). The common basic idea behinds two algorithms is using Power/performance Variation Table (PVT). PVT is a profile characterizing power efficiency of all nodes and can be built by test runs[6], [8] or historic data[9]. With information of power efficiency of each node, algorithms output the energy-efficient allocation plan depending on property of jobs.

This report are organized as follows. Section 2 first introduces several tools used in experiments, then explains the design of the scheduling simulator and two scheduling algorithms. Section 3 shows observed behaviors of ITO-A under different power cap and evaluates energy saving capability of scheduling algorithms. Related work are presented in Section 4 and Section 5 concludes this report.

2. Methodology

2.1 Power Measuring

The composition of power consumption in supercomputer system is very complex. However, the main power consumption still comes from CPUs and memories. Current technology can measure the power consumption of two generic components, identified as PKG for CPUs and DRAM for memories, with high accuracy[10]. Thus, energy saving capability of scheduling algorithms is evaluated by energy consumption of PKG and DRAM during the workload in this report. Power/performance heterogeneity occurs at transistor level. However, this report mainly focuses on job scheduling by which a set of nodes is allocated to a job. Hence, this report chooses node as the smallest unit of heterogeneity.

In order to simulate the power behavior of systems having node-level heterogeneity, the simulator needs to be able to characterise the power consumption of different jobs running on different nodes. One technique to achieve this is PVT. In this report, PVT is built by test runs of applications. PKG energy and DRAM energy data of test runs and evaluation runs is recorded by a power management tool named RAPL[*3]. For benchmark applications, STREAM and HPCG[*4] are used in single-node situation and HPCG is also used in multi-node situation. Test runs and evaluation are launched on Kyushu University's ITO-A supercomputer system.

2.1.1 RAPL

RAPL is a tool introduced in Intel Sandy Bridge processor family at first to provide energy model interfaces in its first generation. Then, it has been constantly enhanced in its successive generations and now provides more valuable interfaces. Ilsche et al. [11], Desrochers et al.[12] and Hackenberg et al.[13] verified the power information and showed that its accuracy has been much improved from Sandy Bridge to the modern state-of-the-art architecture.

In this report, RAPL is used for setting power cap and collecting power data of PKG and DRAM. The specification of RAPL covers DRAM power cap, but supercomputers supporting this functionality rarely exist. Thus, this report mainly studies the behavior of systems under CPU power cap in multi-node situation.

2.1.2 Benchmarks

The STREAM benchmark is used to measure a sustainable memory bandwidth and executes simple vector operations. In single-node situation, STREAM is chosen as the memory-intensive benchmark application. HPCG aims to create a new metric for the ranking of HPC systems and was exploited by Top 500 Supercomputer Site from November 2017[3]. Since HPCG mainly carries out sparse matrix-vector multiplications and vector updates, the evaluation in single-node situation picks it as the computation-intensive benchmark application. One execution of HPCG contains several iterations, and during each iteration of multi-process HPCG, several MPI communications and synchronizations are required between processes. Thus, it is also used for evaluating load imbalance caused by performance heterogeneity in multi-node situation.

2.1.3 ITO-A Supercomputer System

The specification of ITO-A is in **Table 1**. It is a subsystem of Kyushu University's supercomputer system and started its operation from January 2018. Its specification satisfies the demand of this report with its sufficiently large system scale, and according to experimental results node-level power/performance heterogeneity has been observed in it. For each benchmark used in the simulation and evaluation, power and execution time in the PVT are collected from the average value of more than 10 times test runs on all 2,000 nodes in ITO-A. The evaluation of real power consumption of each scheduling algorithms is also carried out on

[*3] Running Average Power Limit.
[*4] High Performance Conjugate Gradients.

Table 1 Specification of ITO-A

Machine	Fujitsu PRIMERGY CX2550/CX2560 M4
	System
Number of Nodes	2,000
Total Cores	72,000
Memory	384 TB
Peak Performance	6.91 Pflops (Double Precision)
Interconnect	InfiniBand EDR 4x (100 Gbps)
	Node
CPU	Intel Xeon Gold 6154 (Skylake-SP) 3.0 GHz (Turbo 3.7 GHz) 18 core x 2 / node
Peak Performance	3,456 Gflops / node (Double Precision)
Memory	DDR4 192 GB / node
Memory Bandwidth	255.9 GB/sec / node
Measurement Tool	Intel RAPL

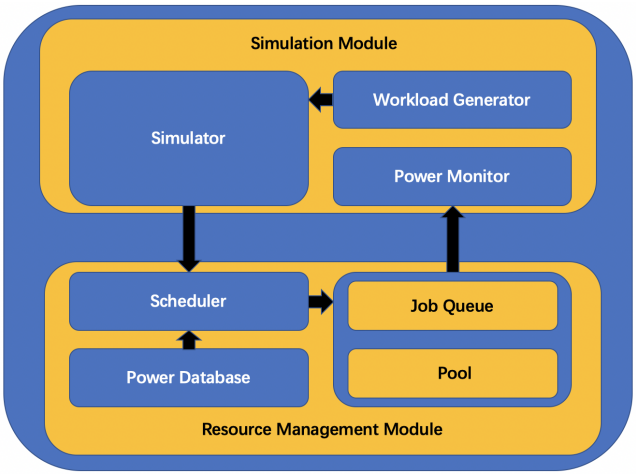

Fig. 2 Simulator architecture

ITO-A.

2.2 Multi-node Scheduling Simulation

In supercomputer systems, execution time of jobs varies from few seconds to hours, and computing nodes provided to each job are also limited. As a result, evaluating scheduling algorithms in real computing environments is inappropriate, even if such a challenge is allowed. Thus, a reliable and customizable simulation environment is necessary for studying the behavior of large-scale systems and evaluating scheduling algorithms. Scheduling simulation introduced in this report supports scheduling of multi-node jobs and is able to characterise power efficiency of each nodes by PVT.

Fig. 2 shows the architecture of the simulation configuration. In Simulation Module, Workload Generator creates a workload, which determines the timing to submit various kind of jobs to the Scheduler, depending on the workload configuration. Simulator submits jobs to Resource Management Module. Submitted jobs are added to the job queue in Resource Management Module, then Scheduler will calculate the allocation plan depending on the scheduling algorithm, jobs in the queue, available nodes in the node pool and PVT in the Power Database. Power Monitor simulates the job execution after scheduling, and records energy consumption.

2.2.1 Workload Generation

There are two states of computing nodes, one is executing a job and another is waiting for scheduler to assign a job. In this report, these two states are called busy and available. Depending on whether the system is busy or not, the energy saving capability of scheduling algorithms can be different. System utilization rate describes how busy the system is, and it is defined by the following:

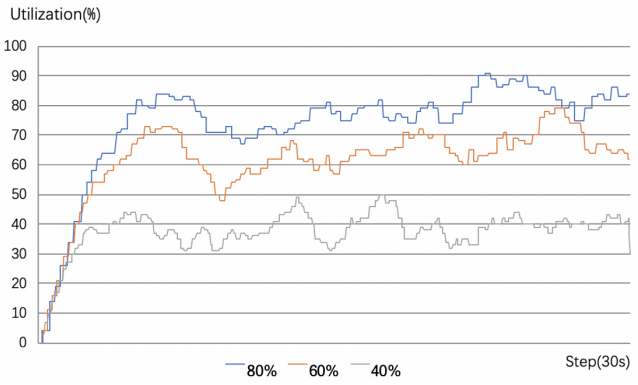

Fig. 3 Supercomputer utilization rate tracing of generated workload

$$U = \frac{N_{busy}}{N_{available} + N_{busy}} \tag{1}$$

where U means utilization rate, N_{busy} and $N_{available}$ present the number of busy and available nodes respectively. In the real supercomputer system, the utilization rate is not constant in time. Previous research[14] reported the utilization rate of ITO-A system has been traced for two weeks (from Jul 08 2018 15:10:37 to July 21 2018 03:30:40) and ranged from 40% to 90%. Thus, the generated workload is designed to keep the utilization rate fluctuating around a certain value according to the workload configuration, so that the behavior of the supercomputer can be simulated under different utilization rate. Figure 3 shows system utilization rate of generated workload in three configurations, in which system utilization rate is fluctuating around 40%, 60% and 80%, respectively.

2.2.2 Power Monitor

After the allocation plan is made during each step, the power monitor updates the power, elapsed time and total energy consumption. Since several researches have been carried out about how to predict the power and the influence of heterogeneity by power logs[6], [9], [15], simulation in this report is based on the assumption that the power consumption, execution time and influence of heterogeneity can be accurately predicted. Thus, the final power and execution time have the same value as in PVT. For multi-node jobs, the simulation assumes that the computational load amount of each node is perfectly balanced.

For a multi-node job j assigned to nodes $N_0, N_1, \ldots, N_{q-1}$, the total power P, execution time t and total energy E are calculated as follow:

$$P = \sum_{k=0}^{q-1} P_{j,k} \tag{2}$$

$$P_{j,k} = P_{j,k}^{PKG} + P_{j,k}^{DRAM} \tag{3}$$

$$t = \max(t_{j,k} : k = 0, 1...q-1) + t_{comm} \tag{4}$$

$$E = Pt \tag{5}$$

where $P_{j,k}$, $P_{j,k}^{PKG}$, $P_{j,k}^{DRAM}$ and $t_{j,k}$ is the node-level total power,

PKG power, DRAM power and execution time of the execution of a single-node job j_{single} on the node N_k. The job j_{single} is used to estimate node-level performance numbers of j which is considered as the weakly-scaled parallel version of j_{single}. For example, j is HPCG with problem size (256, 384, 256) requiring 8 nodes, corresponding j_{single} is HPCG with problem size (128, 192, 128) requiring 1 node. The time t_{comm} represents the total communication cost paid in j's execution. In multi-node situation, the execution time is difficult to be predicted by PVT, because the job execution time of one node is related to other nodes executing the job. Thus, the simulation assumes that the execution time of a multi-node job only depends on the node with the worst performance. To simplify the model, this report does not consider the relationship between the communication cost and power-efficiency of nodes, which means t_{comm} only depends on application, problem size and number of required nodes.

2.3 Heterogeneity Aware Scheduling Algorithms

This section introduces scheduling algorithms applied to the scheduler. The scheduler is a component of Resource Management Module, which consists of scheduler, power database, one FIFO[*1] job queue and one node pool. The architecture of this module allows each components easy to modify, in case new systems or scheduling policies have different requirements. PVT is saved in the power database, recording the average P^{PKG}, P^{DRAM} and t collected from test runs. Scheduler assigns jobs in the job queue to available nodes in the node pool. Currently four scheduling algorithms are applied to the scheduler:

- Naive: An application-unaware, heterogeneity-unaware scheduling, applied to single-node and multi-node situations. This scheduling algorithm always chooses available nodes with the smallest ID number. It is the baseline to evaluate the energy saving capability of other algorithms.
- Power Aware Algorithm (PAA): An application-unaware, heterogeneity-aware scheduling, applied to single-node and multi-node situations. In this algorithm, as long as there are enough available nodes, the scheduler assigns the earliest job in the job queue to the most power efficient nodes regardless of the state in the node pool and job queue. Power efficiency of nodes is ranked according to the average value of all benchmarks' predicted powers in PVT. It saves energy by using efficiency nodes as much as possible, and the energy saving capability has been reported in previous study[14].
- Combinatorial Optimization Algorithm: An application-aware, heterogeneity-aware scheduling, applied to single-node situation. It is the same as PAA if the same application are executed on all nodes. When scheduling various kinds of jobs, COA finds an optimal energy-saving solution by KM algorithms for jobs with different property. Scheduling policy not only depends on power efficiency, but also on the property of applications.
- Sliding Window Algorithm: An application-unaware, heterogeneity-aware scheduling, applied to multi-node situation. This algorithms only targets on multi-node situation under the power cap, and based on the assumption that execution time of a load-balanced multi-node job depends on the worst performance node. In multi-node situation, the performance of highly efficient nodes will be dragged down by other less efficient nodes executing the same job. SWA uses the sliding window so that performance gap between nodes running the same job is not too large. Current SWA in simulations and experiments is application-unaware, but it can be extended to application-aware version.

COA and SWA will be introduced in detail in Section 2.3.1 and Section 2.3.2.

2.3.1 Combinatorial Optimization Algorithm

PAA is an application-unaware scheduling algorithm and thus only considers the ranking of power efficiency. However, some nodes may show the high power efficiency when executing memory-intensive applications, and is not very efficient when executing computation-intensive applications. Thus, the allocation plan in PAA is not the best because PAA does not always choose the most suitable nodes according to the property of applications. Since there are usually two or more jobs in the job queue during each scheduling interval, the queue likely has two ore more jobs to be scheduled at the next interval in its head segment. With this information, COA computes an allocation plan that has minimum energy cost by KM algorithms.

Considering p single-node jobs $J = \{j_0, j_1 ... j_{p-1}\}$ assigned to q nodes $N=\{n_0, n_1 ... n_{q-1}\}$ $(q \geq p)$, the problem can be transformed into the optimal matching problem in a graph: giving a bipartite graph $G(V, E)$ $(V = J \cup N, E = J \times N)$, the weight of edge $w(j, n)$ is the energy consumption of job j running on node n. M $(M \subseteq E)$ is called a matching if $\exists (j, n) \in M$ holds for $\forall j \in J$; for $\forall (j, n) \in M$, $(j, n') \notin M$ and $(j', n) \notin M$ hold for $\forall n' \in N - \{n\}$ and $\forall j' \in J - \{j\}$, respectively. Then, energy consumption of matching M is defined as follow:

$$C_M = \sum_{(j,n)\in M} w(j, n) \tag{6}$$

The matching M with minimum C_M is called a minimum-weighted matching, which is also the minimum energy consumption allocation plan.

KM algorithm is one of the most popular algorithm that solves this assignment problem in polynomial time[16]. The whole process to compute the allocation plan with minimum energy consumption is described in **Fig. ??**. The basic idea of this algorithm is defining a label $l(v)$ for each jobs and nodes, and looking for the M^* that satisfies the following equation:

$$\sum_{(j,n)\in M^*} \{l(j) + l(n)\} = \sum_{(j,n)\in M^*} w(j, n) \tag{7}$$

When a job can not be matched, the algorithm will adjust $l(v)$ while keeping $l(j) + l(n) \leq w(j, n)$ ($\forall j \in J, \forall n \in N$), then try to match the job again. Assuming M^* is generated by the algorithm in Fig. **??**, then for any M the following inequality is satisfied:

$$C_M = \sum_{(j,n)\in M} w(j, n) \geq \sum_{(j,n)\in M} \{l(j) + l(n)\} \tag{8}$$

Note that, with definitions $N(M) = \{n : (j, n) \in M\}$ and $J(M) = \{j : (j, n) \in M\}$, following hold; $l(n) \leq 0$ for $\forall n \in N(M^*)$;

[*1] First In First Out.

```
input: J, N, PVT
output: M_min
initialize
∀ j_i ∈ J, n_j ∈ N, set w(j_i, n_j) according to PVT
for i in 0, 1...p − 1:
    l(j_i) = min(w(j_i, n_j), j = 0, 1...p − 1)
for k in 0, 1...q − 1:
    l(n_k) = 0
for i in 0...p − 1:
    while True :
        clear all marks
        g_min = infinity
        if(match(j_i)):
            break
        for v in marked j and n in the last match://adjust l(v)
            if v in J:
                l(v) = l(v) + g_min
            if v in N:
                l(v) = l(v) − g_min
return M

def match(j_i):
    mark j_i
    for k in 0...q − 1 ://traverse all nodes and find nodes that can be matched
        if n_k is marked:
            continue
        else:
            g = w(j_i, n_k) − l(j_i) − l(n_k)
            if g==0://g cannot be less than 0, if g==0, try to assign the job to the node
                mark n_k
                if n_k is not assigned://no other jobs will be assigned to the node
                    M.add((j_i, n_k))
                    return True
                else://another job will be assigned to the node
                    j* = current job assigned to n_k
                    if(match(j*))://try to assign the job to another node
                        M.remove((j*, n_k))
                        M.add((j_i, n_k))
                        return True
            else:
                g_min = min(g_min, g)
    return False
```

Fig. 4 Optimal allocation plan with KM algorithm

$l(n) = 0$ for $\forall n \notin N(M^*)$; and $\sum_{j\in J(M)} l(j) = \sum_{j\in J(M^*)} l(j)$. Therefore, the following is obtained to show the optimality of M^*:

$$\sum_{(j,n)\in M} \{l(j) + l(n)\} \geq \sum_{(j,n)\in M^*} \{l(j) + l(n)\} \quad = \sum_{(j,n)\in M^*} w(j,n) \quad (9)$$

Therefore, the allocation plan generated by COA consumes less energy than all other allocation plans. If the number of jobs in the job queue m is more than the number of available nodes n, only the earliest n jobs in the queue can be scheduled to obey the FIFO principle.

2.3.2 Sliding Window Algorithm

When power cap is not applied to nodes, the difference of CPU frequency between nodes is very small. Considering a q-node job j assigned to q nodes (n_0, n_1... n_{q-1}), let P, t and E be the power consumption, execution time and energy consumption of j's execution, respectively. Suppose q single-node weakly-scaled jobs $j_{single,k}$ ($k = 0, 1...q-1$) are assigned to the same q nodes, and let $P_{single,k}$, $t_{single,k}$ and $E_{single,k}$ be the power consumption, execution time and energy consumption of the job $j_{single,k}$, respectively. Since the relationship between communication cost and nodes is ignored and CPU frequency may be assumed independent of nodes in the case without power capping, following equations are obtained:

$$t = \max(t_{single,k}) + t_{comm} = t_{single,k} + t_{comm}, k = 0, 1...q-1 \quad (10)$$

$$E = Pt = \sum_{k=0}^{q-1} P_{single,k} t = \sum_{k=0}^{q-1} \{E_{single,k} + P_{single,k} t_{comm}\} \quad (11)$$

It is proved that the scheduling of multi-node jobs can be transformed to the scheduling of several single-node jobs based on Equation (2)–(5), and COA still works. In the situation with power capping, however, Equation (10) does not hold anymore, which means COA can not be applied to such situation. In this

report, SWA focuses on multi-node situation, where multi-node jobs are assigned to nodes under power caps.

In multi-node situation, the simulation assumes that the execution time is determined by the node with the worst performance when load-balanced multi-node job is executing on nodes with different computational performance. Thus, if good performance nodes (good nodes) and bad performance nodes (bad nodes) are executing the same multi-node job, the performance of good nodes will be dragged down by bad nodes. Since this performance loss is caused by the load imbalance within multi-node jobs, the load imbalance can be reduced if the scheduler chooses nodes of similar performance to execute the a multi-node job. Such allocation plan makes it more difficult to assign good nodes to a multi-node jobs, but these nodes still have chance to have smaller scale multi-node jobs or, more likely, single-node jobs fully exerting their performance if such small jobs are ready to run in the queue. Since single-node and small scale multi-node jobs are dominant, the allocation plan aware of node performance similarity will work well without degrading the utilization rate of good nodes.

SWA performs node assignment for a q-node job taking care of the performance similarity by sliding a *window* wider than q over all nodes, ranked by their computational performance, one by one from the ranking top to the bottom until q available nodes are included in the window. As exemplified in **Fig. 5** for an 8-node job, SWA successfully finds eight nodes whose performance is similar to each other, leaving two most efficient nodes which will be likely utilized immediately by a 2-node job or two single-node jobs at the head of the queue. Note that SWA can successfully terminate with more than q available nodes in the window only when they are found at the very beginning of the sliding. In this case it simply chooses most efficient available nodes.

As discussed above, Equation (10) does not hold under the power cap so that KM algorithm is difficult to be applied in multi-node situation, since $w(j,n)$ is not a constant value. Thus, scheduling algorithms used in this experiment of multi-node situation are application-unaware with a workload consisting of jobs for one particular application but with different problem size and the number of required nodes. However, there are some ways extending SWA to an application-aware version, for example, resorting nodes and updating the ranking before scheduling a new job. The detail of the application-aware SWA is discussed in Section 5.

3. Validation and Evaluation

3.1 Single-node Situation

COA focuses on scheduling single-node jobs with the different property to nodes with the different power efficiency. Thus, the power saving capability of COA not only depends on nodes, but also relates to applications. In this section, an experiment is carried out to verify the difference between the power behavior of benchmark applications. Then the same workload is executed on both the simulator and ITO-A to evaluate the accuracy of the simulator and power saving capability of scheduling algorithms. The evaluation is carried out under two different utilization rates.

3.1.1 Node-level Heterogeneity Verification

This section verifies that memory-intensive applications show different power behavior from computation-intensive applications. For verification, STREAM and HPCG are chosen as the memory-intensive and computation-intensive benchmark applications, respectively. Two benchmark applications are executed for 10 times on 2,000 nodes on ITO-A. The average value of PKG power and DRAM power are measured by RAPL. The scheme of experimental setup is briefly shown in **Table 2**.

Figure 6 shows results of the verification. To study the difference between benchmark applications, the PKG power and DRAM power of nodes are displayed separately and exceptional points have been removed. X-axis represents the ranking of 2,000 nodes, and is ordered by the average PKG/DRAM power of two benchmark applications. It should be noted that these two rankings are different and no clear relationship was found. Y-axis shows the average PKG/DRAM power consumption of 10 executions.

For the PKG power consumption, the variation in HPCG reaches up to 17W (107W–124W), about 14.4% of the average value. However, the PKG power consumption of STREAM shows a variation of 4W (119W–123W), only 3.3% of the average value. The PKG power consumption of these two applications among all nodes does not show a strong relation. In contrast, for the DRAM power consumption, tendencies of STREAM and HPCG are similar. The DRAM power of STREAM shows a 16W (60W–78W) variation, which is 23.5% of the average value. Similarly, the result of HPCG is 25.0% of the average value ranging in 30W–40W.

Two important facts can be inferred from the result. One is that a PKG-efficient node may not be a DRAM-efficient node since the DRAM power ranking of nodes has no obvious relationship with the PKG power ranking. Thus, application-aware scheduling is necessary because the power consumption not only depends on the ranking of nodes, but also depends on the property of applications. The other is that the variation of STREAM is much less than HPCG in terms of the total power consumption, which means assigning HPCG to a power-efficient node can save more power compared to STREAM. In contrast, even if STREAM is assigned to a power-inefficient node, the additional power consumption is relatively acceptable. This requires the scheduling algorithm not to consider the earliest job in the queue, but to trade off in all jobs that need to be assigned.

3.1.2 Simulation and Evaluation of Combinatorial Optimization Algorithm

The evaluation of scheduling algorithms was carried out both in the simulator and ITO-A. Only one representative workload is used due to the limitation of experiment time in ITO-A. The workload includes two different kinds of jobs, representing computation-intensive jobs and memory-intensive jobs. Without

Table 2 Experimental setup of node-level power heterogeneity verification

Node	2,000 nodes	
Benchmark	Node(Process/Thread)	Problem Size
STREAM	1 (1/ 36)	ARRAY_SIZE=6G
HPCG	1 (1/ 36)	X=128, Y=192, Z=128

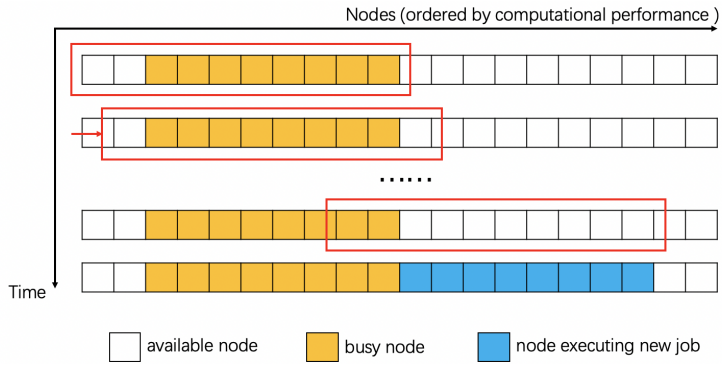

Fig. 5 Process of Sliding Window algorithm ($WindowSize$=10)

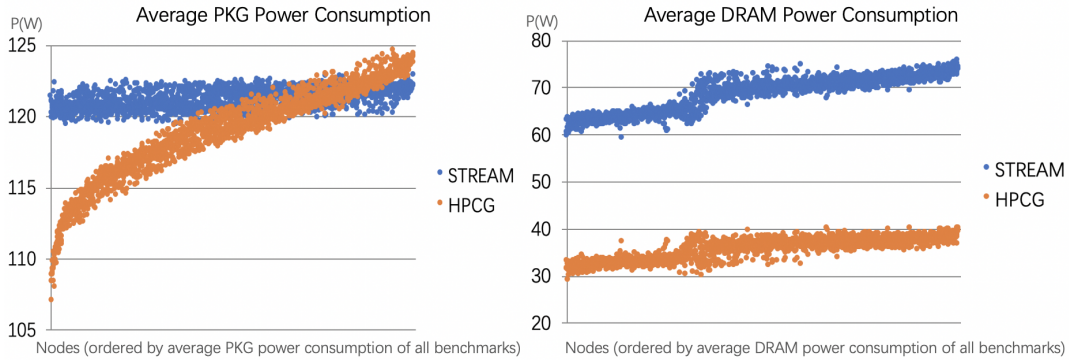

Fig. 6 Power consumption heterogeneity of STREAM and HPCG

Table 3 Experimental setup of the evaluation in single-node situation

Scenario	Utilization Rate	Node
Busy	80%	1,990 nodes
Free	40%	1,990 nodes
Benchmark	**Problem Size**	**Count**
STREAM	ARRAY_SIZE=6G	5,700
HPCG	X=128, Y=192, Z=128	3,400

All jobs use 1 node, 1 process and 36 threads for each

loss of generality, the total time for two benchmark applications in the workload is set to be similar. The evaluation is carried out under two scenarios (busy and free) with different utilization rates to verify how the utilization rate affects the energy saving capability. Workload and allocation plan applied to the simulation and ITO-A is the same in order to evaluate the accuracy of the simulator. In each scenario, three scheduling algorithms, Naive, PAA and COA, are applied. The experimental setup is briefly shown in **Table 3**.

To compare the energy saving capability of algorithms quantitatively, the energy saving rate of algorithm A in scenario S is defined as follow:

$$Saving_{A,S} = \frac{E_{Naive,S} - E_{A,S}}{E_{Naive,S}} \tag{12}$$

where $E_{A,S}$ is the total energy consumption of algorithm A in the scenario S. The comparison of scheduling algorithms in two scenarios is shown in **Fig. 7**. X-axis represents whether the energy consumption is from the simulation or real, Y-axis is the energy saving rate. Both PAA and COA show better energy saving capability in the free scenario. This is because both algorithms save power by using good nodes as much as possible. However, the busy scenario, in which almost all good nodes are busy, forces some jobs to be assigned to bad nodes. In all scenarios, COA saves more energy than PAA. As discussed above, the total power consumption variation of STREAM is less than HPCG, then COA can use this fact to save more energy when jobs must be assigned to bad nodes, while PAA cannot. Thus, the difference between two algorithms is more significant in busy scenario as shown in the figure, where COA saves 17% more energy than PAA.

To describe the difference between the simulation energy consumption $E_{simulation}$ and real energy consumption E_{real}, the error of the simulation is defined as follow:

$$Error = \frac{|E_{real} - E_{simulation}|}{E_{real}} \tag{13}$$

Figure 8 shows the error of simulations in different scenarios. The maximum error is only 0.68% of total energy consumption. Comparing with PAA and COA, the simulation error of Naive is much smaller. There are many possible explanations for this. For example, PAA and COA always assigning jobs to the same node may cause the overheat, then it forces CPU frequency to decrease, which results in the changes of power and worse accuracy[17].

3.2 Multi-node Situation under Power Cap

In this section, several experiments are carried out to verify the power behavior of multi-node jobs under power caps. Then three algorithms (Naive, PAA, SWA) are compared in the simulator and ITO-A. The workload used for evaluation is set to be similar to a historic workload in the real supercomputer.

3.2.1 Verification of Power Behavior in Multi-node Situation Under Power Caps

One feature of supercomputers under power caps is that node-level power heterogeneity is transformed into node-level performance heterogeneity. In this case, the variation of power is very

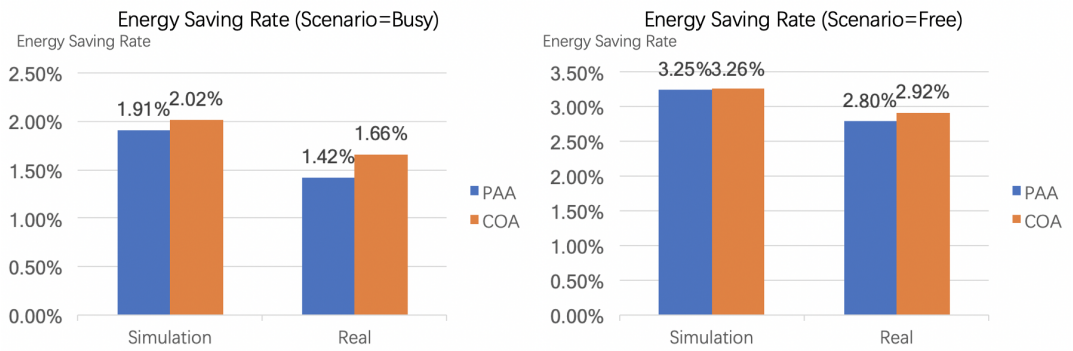

Fig. 7 Comparison of energy saving rate in single-node situation

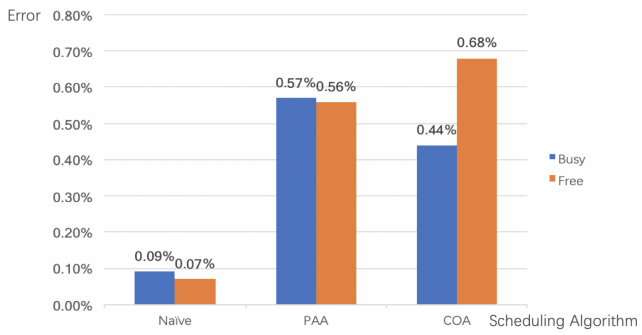

Fig. 8 Error of single-node simulations

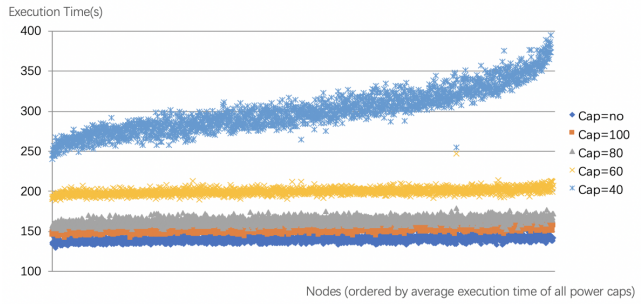

Fig. 9 Execution time of HPCG under different power caps

small, while nodes show different computational performances. The most important factor to determine the energy consumption is the execution time, which depends on the performance of nodes. To verify this, the performance of all nodes is measured under different CPU power caps. Each node executes single-node HPCG with one process, 36 threads and problem size of (104,104,104) for ten times. The same set of jobs has been launched for four times under different CPU power cap and one time without capping for comparison. The execution results are shown in **Fig. 9** and the power consumption is shown in **Fig. 10**.

For the execution time, all nodes show similar computational performances when there is no power cap. The execution time range from 130s–151s, its variation is about 15% of the average value, and no strong relationship with power heterogeneity is observed as discussed in Section 3.1.1. Thus, this variation may be caused by other reason instead of node-level heterogeneity, for example, the CPU frequency changing in the execution. As the power cap becomes tighter, the execution time becomes longer. When the power cap is extremely tight (40W), the execution time increases significantly, and shows clear relationship with the power efficiency of nodes. It is also observed that the execution time ranges in 250s–400s resulting in a large variation of almost 50% of the average value, much larger than the 14.4% variation of power consumption shown in Section 3.1.1. It can be inferred that the node-level heterogeneity becomes more serious under the tight power cap.

In contrast to the situation in Section 3.1.1, the PKG power consumption of all nodes are very close under each of CPU power caps. As for DRAM power, it is almost insensitive with the tightness of power capping and almost independent of PKG power as well. Thus, the power heterogeneity is smaller in this case, and the performance heterogeneity, which reflected in the variation of execution time, becomes the most important factor of total energy consumption. Following simulations and experiments are carried out under the power cap of 40W.

An important assumption in this work is Equation (4), which states that the execution time of multi-node job is determined by the worst performance node under the power cap. A small-scale experiment is carried out to study the execution time of multi-node jobs under the power cap, and to prove that it is possible to save energy by SWA. Figure 11 explains graphically the difference between allocation plans of PAA and SWA when assigning one multi-node job (j_1) and 4 single-node jobs ($j_2, \ldots, j_5$). The ID of the node is labeled from the ranking of its computational performance. In PAA, j_1 first arrives and is assigned to the best 8 nodes (node 1, 2, 11, 12, 21, 22, 23 and 24), then single-node jobs $j_2, \ldots, j_5$ are assigned to nodes 25, . . . , 28. In this situation, node 1 and 2 are both good nodes, but they cannot take their performance advantages because the execution time is decided by node 24. In SWA with the window size of 10 nodes, the job assigned to each of nodes 1, 2, 11 and 12 is the single-node job, which means there is no performance loss on these nodes. The multi-node job must suffer an unavoidable performance loss of some nodes, such as the node 21. However, the performance difference between the node 21 and 28 is much smaller than difference between the node 1 and 24. Hence, the performance loss of SWA is less than that of PAA.

30 nodes were selected according to the stratified sampling on ITO-A to verify the power consumption and execution time in the above situation. Figure 12 shows the experimental result. It is observed that, compared with PAA, SWA brings a small rise of 15s

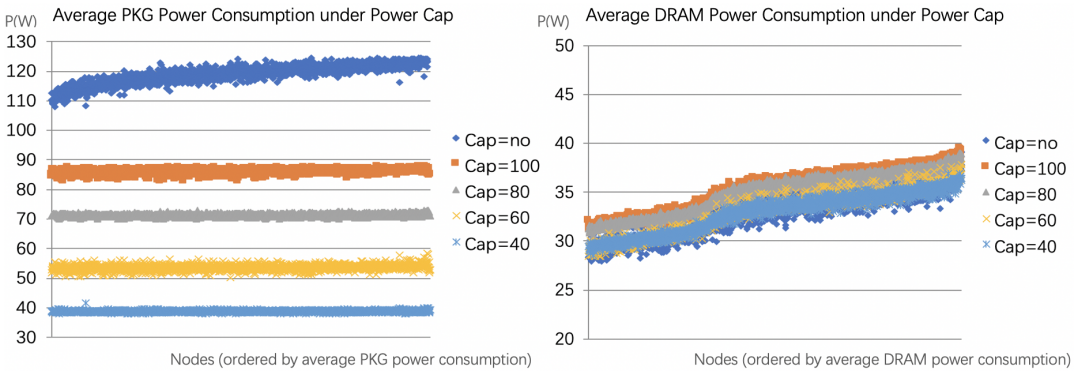

Fig. 10 Power consumption of HPCG under different power caps

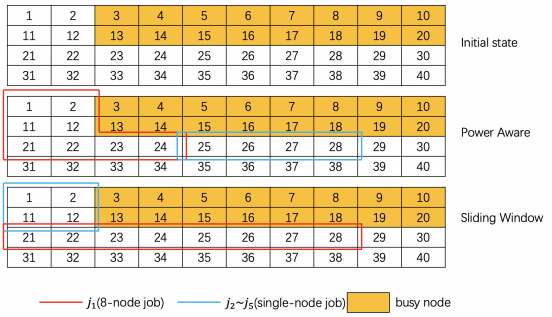

Fig. 11 Allocation plans of PAA and SWA

to the execution time of the multi-node job, while it significantly decreases the execution time of single-node jobs. For example, a single-node job assigned to the node 1 by SWA takes 65s shorter execution time than the node 25 chosen by PAA. Compared to PAA, the total energy consumption of SWA is reduced by 1.7%. Another important observation is that the difference of execution time between single-node jobs and multi-node jobs is relatively large and cannot be ignored, even though the problem size for each node is the same. Theoretically, the communication cost is related to the performance of nodes but the prediction of the communication cost is so complicated, thus that the current simulation does not consider the relationship.

3.2.2 Simulation and Evaluation of Sliding Window Algorithm

The power saving capability of SWA depends on the number and execution time of both multi-node and single-node jobs in the workload. This comes from that the number of jobs assigned to both good nodes and bad nodes is related to the arriving time, execution time and number of required nodes. Thus, in order to simulate the workload in real supercomputers faithfully, a historic workload of Laurel 2 was analyzed (from September 18 2019 00:00:00 to September 25 2019 00:00:00). Table 4 shows the result of classifying all jobs during this period according to the execution time and the number of required nodes. The number in the table is the total of classified jobs. It is observed that multi-node jobs are much less than single-node jobs, and the number of required nodes for most multi-node jobs is less than 8.

In the evaluation, four configurations of HPCG are used to represent jobs with different execution time and required nodes. The difference among these four configurations is shown in **Table 5**. The number of iterations is used to control the execution time of

Table 4 Classification of jobs in Laurel 2

	<360s	360s-3,600s	>3600s
1 node	10,472	7,749	10,437
2-8 nodes	253	172	516
>8 nodes	1	0	3

jobs, rather than using the problem size whose change also affects the execution time but causes a chaotic behavior in power consumption due to the complicated structure of HPCG. The number of jobs is set according to the historic workload in Laurel 2. Due to the resource constraints, it is difficult to run workload that lasts for more than an hour in ITO-A. Thus, comparing with the real execution time in Laurel 2, the execution time of jobs in the evaluation is cut down. Jobs with extremely short execution time ($\leq$ 20s) are discarded because the power consumption of these jobs cannot accurately be measured and is a very small proportion of the total energy consumption.

Fig. 13 shows the energy saving rate of PAA and SWA in the simulation and ITO-A. Compared to the energy saving rate without the power cap, the energy saving rate of PAA under the tight power cap becomes higher since the performance heterogeneity under the power cap of 40W is larger than the power heterogeneity without power cap. In free scenario, there are many available nodes with similar performance so that the number of jobs assigned to a mixture of good and bad nodes is small. Thus, the power savings of PAA and SWA is close in free scenario. However, since there are not many available good nodes to choose in busy scenario, PAA assigns more jobs to the mixture of good and bad nodes resulting in performance loss. Thus, SWA saves more energy than PAA in the busy scenario. Another observation is that both algorithms show better power saving capability in the free scenario, which is the same as single-node situation. The reason is that more available nodes in the free scenario mean more options for the scheduler, making it possible to assign more jobs to good nodes keeping bad nodes less busy, while in the busy scenario the scheduler has to assign jobs to bad nodes since the number of available good nodes is not enough.

In multi-node situation, the error of the total power consumption between simulation results and real results is larger than single-node situation, as shown in **Fig. 14**. The difference is caused by many factors. For example, the execution time of jobs is unstable under tight power cap and sometimes exceptionally long. Another possible reason, as discussed above, is that com-

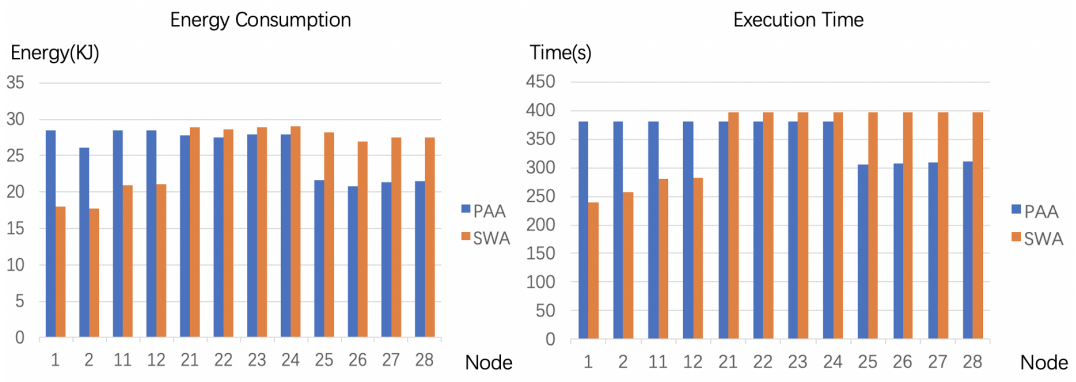

Fig. 12 Experimental result of allocation plans in **Fig. 11**

Table 5 Configuration of jobs in the simulation

Scenario	Utilization Rate	Nodes	Power Cap	
Busy	80%	665 nodes	40W	
Free	40%	665 nodes	40W	
Job	Node(Process/Thread)	Iterations	Number of Jobs	Benchmark
A	1 (1/ 36)	1 times	774	HPCG (104-104-104)
B	1 (1/ 36)	10 times	1,043	HPCG (104-104-104)
C	8 (8/ 36)	1 times	17	HPCG (208-208-208)
D	8 (8/ 36)	10 times	51	HPCG (208-208-208)

munication cost of multi-node jobs is related to the performance of nodes but not considered in the current simulation.

4. Related Work

Several studies have been carried out to reduce the additional energy consumption caused by power/performance heterogeneity in supercomputers. Inadomi et al. reported that the node-level power heterogeneity is transformed to the node-level performance heterogeneity under power caps, and introduced a power budgeting framework, which is based on the power variation estimation with PVT[6], [8]. Uno et al. also proposed a power budgeting framework based on power estimation, and discussed the scheduling from another viewpoint, which reduces performance loss by preventing the power of job from exceeding the power constraint[18]. A node-level power heterogeneity aware resource management was proposed in [14], the algorithm presented in which is the prototype of PAA in this report, and is the very fundamental version of the algorithms proposed in this report. Comparing with these works, this report takes different perspectives to reduce additional energy consumption caused by power/performance heterogeneity. COA focuses on both power heterogeneity and application factors by solving the optimal assignment problems, and SWA prevents the performance loss caused by tight power constraint by reducing load imbalance with the sliding window.

For power estimation, Inadomi et al. introduced a power model predicting power and performance variation by test runs[6], [8]. The approach presented in [9] and [15] used historic execution data, such as the username, number of required nodes and real/estimated execution time to estimate the power consumption. The power estimation in this report refers to the methods in these works.

5. Conclusion and Summary

This report first presented a newly developed a multi-node scheduling simulator that can be applied to nodes with different power efficiency and computational performance, then proposed two scheduling algorithms. COA is an application-aware scheduling algorithm targeting on single-node situation without power constraint, and saves the energy by solving the optimal assignment problem with KM algorithm. SWA reduces the load imbalance in multi-node jobs caused by the performance heterogeneity under tight power caps by a sliding window. These two algorithms are compared to Naive and PAA in the simulation and the real supercomputer. As a result, COA saved up to 2.92% energy saving rate compared to Naive in single-node situation. In multi-node situation, the best energy saving rate of SWA reached 5.36% compared to Naive under a power cap of 40W . It should be noted that 40W is a very strict power cap for computing nodes, and thus is not usually applied in real supercomputers. However, studying power behavior of supercomputers under such a power cap is necessary, because in future it is expected that even a relaxed capping will cause a more significant performance degradation in multi-node job execution due to the enlargement of semiconductor process variation according to the shrinkage of transistors.

In this report, SWA is application-unaware because the execution time of of a multi-node job for a node depends on other nodes involved in the job, making it hard to apply to KM algorithm to job scheduling. However, there are still some way to exploit SWA to application-aware version. One way is to resort the ranking of nodes according to the property of the job to be scheduled although it may result in a sparse distribution of available nodes. Another way is dividing nodes into a number of node pools by CPU performance and DRAM performance to

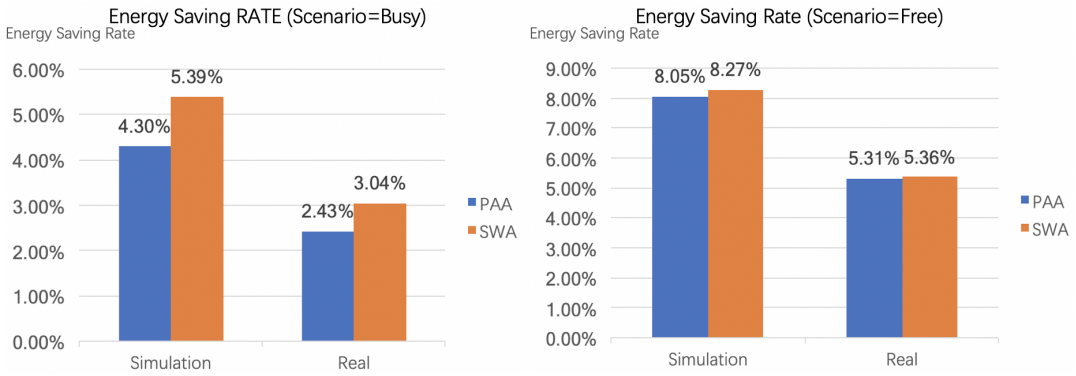

Fig. 13 Comparison of energy saving rate in multi-node situation

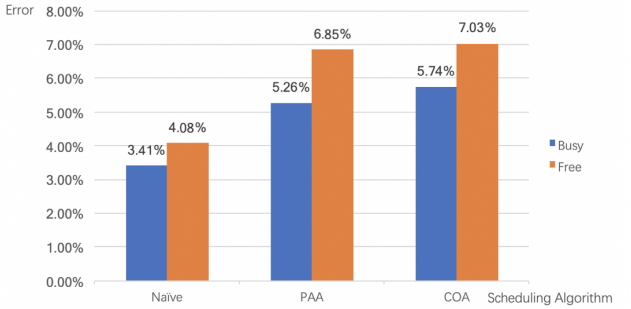

Fig. 14 Error of multi-node simulations

make sure all multi-node jobs are assigned to nodes in a pool with a similar performance. In this case, it is possible that the utilization rate of one node pool is very high and others are very low, resulting in some good nodes being idle for a long time.

The error between the power consumption in the real supercomputer and simulation is less than 0.68% in single-node situation. However, considering the amount of saved power, it is still hard to be ignored, and the error is even bigger in multi-node situation. The communication cost of multi-node jobs is also an important factor that makes results less faithful. Thus, the accuracy of simulation still needs to be improved by some techniques. For example, the granularity of power consumption and computational performance heterogeneity can be cut down to each core in the node, and the error of data collected from test runs can be reduced by strictly controlling the frequency of each CPU. Studying the relationship between the communication cost and performance of nodes is also necessary.

Finally, the workload in the evaluation is not perfectly same as the historic workload in Laurel 2, because it is generated in a stochastic process and applications in it are also different. When a workload includes jobs lasting for hours and jobs only executing for few seconds, it must be considered that whether SWA still saves energy or not.

Acknowledgements

In this research work we used the supercomputer of ACCMS, Kyoto University.This work was supported by “Advanced Computational Scientific Program” of Research Institute for Information Technology, Kyushu University.

References

[1] Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Commun. ACM*, 56(2):93–102, 2013.

[2] Shoaib Kamil, John Shalf, and Erich Strohmaier. Power efficiency in high performance computing. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.

[3] Performance development.

[4] Pete Beckman, Ron Brightwell, Maya Gokhale, Bronis R de Supinski, Steven Hofmeyr, Sriram Krishnamoorthy, Mike Lang, Barney Maccabe, John Shalf, and Marc Snir. Exascale operating systems and runtime software report. 2012.

[5] Shekhar Y. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[6] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David K. Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In Jackie Kern and Jeffrey S. Vetter, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 78:1–78:12. ACM, 2015.

[7] 稲富雄一, 和田康孝, 深沢圭一郎, 青柳睦, 近藤正章, 三吉郁夫, 井上弘士. 電力性能特性ばらつきを考慮した MPI 並列アプリケーションの性能最適化. **情報処理学会研究報告**, 2014-HPC-147(6):1–8, 2014.

[8] 稲富雄一, 井上弘士, 和田康孝, 深沢圭一郎, 上田将嗣, 近藤正章, 三吉郁夫, 青柳睦. 電力制約スーパーコンピューティングにおける製造ばらつき問題とその対策–大規模計算機システムを対象とした電力バジェット配分法の提案–. **情報処理学会研究報告**, 2015-HPC-150(27):1–8, 2015.

[9] 山本啓二, 末安史親, 宇野篤也, 塚本俊之, 肥田元, 池田直樹, 庄司文由. 過去の実行実績を利用したジョブの消費電力予測. **情報処理学会研究報告**, 2015-HPC-151(2):1–7, 2015.

[10] 小野美由紀, 山本昌生, 中島耕太. ソフトウェアの消費電力分析手法. **情報処理学会研究報告**, 2015-HPC-150(29):1–6, 2015.

[11] Thomas Ilsche, Robert Schöne, Joseph Schuchart, Daniel Hackenberg, Marc Simon, Yiannis Georgiou, and Wolfgang E. Nagel. Power measurement techniques for energy-efficient computing: reconciling scalability, resolution, and accuracy. *SICS Softw.-Intensive Cyber Phys. Syst.*, 34(1):45–52, 2019.

[12] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A validation of DRAM RAPL power measurements. In Bruce L. Jacob, editor, *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Alexandria, VA, USA, October 3-6, 2016*, pages 455–470. ACM, 2016.

[13] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the Intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 896–904. IEEE Computer Society, 2015.

[14] LE LI, Keiichiro FUKAZAWA, Hiroshi NAKASHIMA, and Takeshi NANRI. A node level performance/power efficiency aware resource management technique. *IPSJ SIG Notes*, 2018-HPC-166(3):1–7, 2018.

[15] 宇野篤也, 末安史親, 山本啓二, 肥田元, 池田直樹, 辻田祐一. ジョブの時系列電力変動の推定手法の検討. **情報処理学会研究報告**, 2018-HPC-167(20):1–6, 2018.

[16] Hong Cui, Jingjing Zhang, Chunfeng Cui, and Qinyu Chen. Solving

large-scale assignment problems by kuhn-munkres algorithm. 2016.
[17] M. Platini, T. Ropars, B. Pelletier, and N. De Palma. CPU overheating characterization in HPC systems: A case study. In *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pages 59–68, 2018.
[18] 宇野篤也, 末安史親, 山本啓二, 肥田元, 池田直樹, 辻田祐一. 消費電力の変動を考慮したジョブスケジューリングの検討. **情報処理学会研究報告**, 2018-HPC-167(5):1–6, 2017.