Evaluation of Impact of Noise on Collective Algorithms in Repeated Computation Cycle

HONGZHI CHEN[†1] REIJI SUDA[†1]

Abstract: With the scale increasing, the impact of OS noise has become a key limiter of the performance of supercomputer. In this paper, we focus on a common supercomputing structure, which repeatedly execute computation followed by a collective operation implemented by using butterfly algorithm. We introduce a new proposed algorithm by increasing the number of paths of data exchange and evaluate the impact of noise on conventional butterfly algorithm and the proposed one by using the LogGOPS model. We simulate three situations: collective using butterfly algorithm without noise, collective using butterfly algorithm with noise and collective using proposed algorithm with noise. And the results show that the proposed algorithm improved the performance by almost 40 percent.

Keywords: Noise, All-Reduce Operation, Repeated Computation Cycle

1. Introduction

1.1 Motivation

A decade ago, Petrini et al. [1] found that the performance was much worse than their anticipation when they were running a large-scale parallel application on ASCI-Q, an 8192 processor supercomputer. After that, many studies have shown that Operating System (OS) noise is one of the main causes of performance missing [1][2][3]. Today, with the advent of the newest champion of TOP500 List, Tianhe-2 [4], which has more than 8M cores, the impact of OS noise would be a key limiter with the scale increasing.

Generally, when processors computing collaboratively, sometimes they need exchange their data to do the further computing [10], and the most widely used operations are the all-reduce operations [11]. In this case, the slowest processor will slow the entire system down. Fortunately, noises on different processors in computing phase only affect the system once, but in communicating phase, they may affect some processors more than twice [12]. Since these computing communicating cycles will execute many times till the end of the application, the performance of the entire system will be reduced severely. [13]

Besides, it is evident that the main forms of OS noise are daemons and interrupts [3][5], so in order to mitigate the impact of noise, many systematic approaches have been proposed. One way to do that is reducing the frequency and the duration of noise by removing the useless system daemons or idle a processor to deal with noise. Another common systematic approach is synchronizing the noises across all processors in one computation system by modifying the schedulers of the processors [6][7][8]. However, the proposed algorithm approaches are less [9], so in this paper, we will propose one and introduce a way evaluate it.

1.2 Our Contribution

The main contribution of this paper is to propose a new algorithm of all-reduce operation. All-reduce operation is kind of leverage for noise influence, the impact of noise in all-reduce operation will be expanded several times stronger, so we focused on analyzing the butterfly algorithm, a widely used all-reduce operation, and based on the binary tree of data exchange in butterfly algorithm, then proposed the redundant exchange butterfly algorithm. By increasing the number of paths of data exchange, the impact of noise was mitigated. After that, we built a simulation model of supercomputing and showed that even if no noise occurs, the cost of the proposed algorithm is relatively low, but if noise occurrence is relatively heavy, the improvement of the proposed algorithm is significant.

1.3 Related Work

There are a lot of methods to mitigate the impact of noise [6][8]. The primitive way is idle a processor on each node to absorb noise, but people are unwilling to waste the entire computing ability of those processors. So many researchers try to increase the utilization of the supercomputer, and lots of systematic methods are proposed. One of them is reducing the frequency and the duration of noise by removing several system daemons which is unneeded by supercomputing, by which the impact of noise can be mitigated essentially [6][8]. Another common systematic approach is synchronizing the noises cross all processors in one computation system by modifying the schedulers of the processors, so that the cross- impact can be reduced, and the overall performance is improved [14]. Besides, there is a measure solely with compile and run time configurations in recent unmodified Linux kernel by using invasive approaches to remove the involuntary preemption induced by task scheduling [12][16][17].

1.4 Organization

Section 2 presents the butterfly algorithm and introduces a new proposed algorithm which can reduce the effect of noise, and analyzes them. Then we introduce the model for simulating the supercomputing in Section 3. Section 4 shows the result of the simulation of two algorithms. Eventually, we conclude the paper and discuss the future work in Section 5.

†1 Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo

2. Methodology

Consider a parallel application running on a supercomputer by using N processors. We assume $N = 2^K$ to simplify the problem.

2.1 Butterfly Algorithm

There are several algorithms to implement the all-reduce operation. In this paper, we focus on the particularly important one, butterfly algorithm, which is widely used in application on supercomputer. In this algorithm, each two processors have the same data after exchanging theirs, so after the i^{th} round, every 2^i processors have the same data. For all 2^K processors, it needs K rounds to finish the butterfly algorithm. **Algorithm.1** shows the pseudo code of the butterfly algorithm on each processor. Note that it only shows communications without showing operations.

Algorithm 1 (i^{th} processor)

```
1:  round = 0
2:  datagot[0] = 1
3:  datagot[1…K-1] = 0
4:  while round < K do
5:    while datagot[round] == 1 do
6:      tgt ← i xor (1 << round++) // Original Data Exchange
7:    end while
8:    while data_arrived do
9:      datagot[src_round] = 1
10:   end while
11: end while
```

The example shown in **Fig.1(a)** is the ideal situation. In each round, all processors start exactly at the same time. Since noiseless, they also end at the same time to start next round at the same time too. However, if some of the processors are delayed by noise like P3 in 2nd round, the associated processors need to wait for the delayed ones, so P1 need to wait for the end of noise on P3, that is the way through which the noise on P3 affects more than one processors, which shown in **Fig.1(b)**.

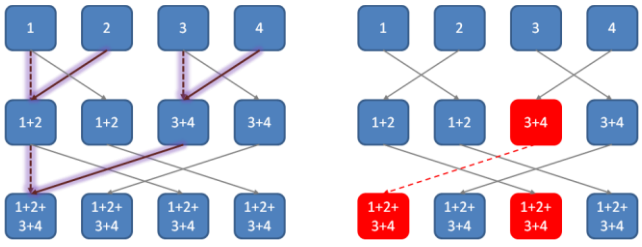


Fig.1 Example of butterfly algorithm

It should be notice that, if noises occur only in the beginning of the butterfly algorithm, all processors can receive the final result nearly at the same time, the time difference is less than several network latencies. But if noises occur in the middle of the butterfly algorithm on different processors in different rounds, the time difference of finishing butterfly algorithm of all processors will be huge and the performance might be strongly reduced. However, it is reported that OS noise mainly occurs approximately periodically [7]. So in the extreme situation, if all processors on one of the paths to any processor of all 2^k processors delayed one by one, the processor will be delayed K times, shown in **Fig.2**.

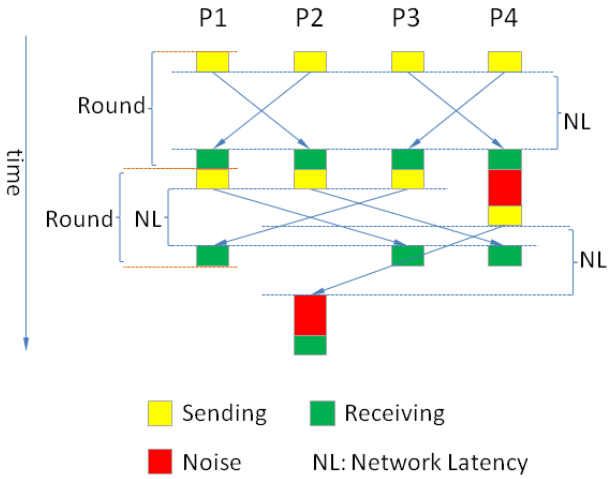


Fig.2 Noises affect more than once

2.2 Analysis

In the conventional butterfly algorithm, each processor obtains the final result from a single binary tree through K rounds of data exchange. In this way, if noise occurs, it will affect the processors which need its data round by round, finally leads to performance degradation, and the number of affected processors will be doubled after each round. However, it should be notice that, after the 1st round data exchange, each two processors have exactly the same data, and they will never communicate with each other anymore. Furthermore, in each round, the data on the two processors are always the same, so if one of them is delayed by noise, we can recover it by using the data in the other processor, but if the other processor is delayed by noise too, then the noises become unrecoverable.

2.3 Proposed Method

In order to recover the delayed processors before the noises become unrecoverable, we can obtain data from the other binary tree which have the same data but haven't be delayed. It's easy to know that the 1st round exchange target of each processor meets the requirements. So the proposed method is adding some redundant exchanges in the middle of the butterfly algorithm.

Fig.3 is a macroscopic example of the proposed method. The black lines are the data streams of traditional butterfly algorithm, but it fails two times shown by red dotted lines in the figure, one is in the second round, the other is in the third round. The green lines represent the redundant exchange data streams of our method. In our method, each processor has more than one binary tree to get the data needed. **Algorithm.2** shows pseudo code on each processor. RDE stands for Redundant Data Exchange, and ODE stands for Original Data Exchange in pseudo code.

It should be noticed that we do not synchronize processors after each round, so when we try to recover some delayed processors, they may be delayed for more than one rounds, and some other processors is waiting for its former data. So it should

send though the traditional butterfly algorithm path to ensure no

Algorithm2 (ith processor)

```
1:  round = 0
2:  datagot[0] = 1
3:  datagot[1...K-1] = 0
4:  while round < K do
5:    while datagot[round] == 1 do
6:      tgt ← i xor (1 << round++) // ODE
7:      if round ∈Redundant Exchange Round Set then
8:        tgt ← i xor 1 // Redundant Data Exchange
9:      end if
10:   end while
11:   while data_arrived do
12:     if ODE then
13:       datagot[src_round] = 1
14:     else
15:       while round < src_round do
16:         tgt ← i xor (1 << round++) // RDE
17:       end while
18:     end if
19:   end while
20: end while
```

binary tree for the final result is deleted.

Normally, processors need to wait for the data from the corresponding processor after sending, so the redundant data exchange can be done in the gap between the original sending event and receiving event normally which will not slow the performance down significantly. While once it works, the improved performance is very significant. **Fig.4** shows these properties of redundant data exchange. The redundant data arrives earlier than the original data, so the performance of P2 is improved, but the performance of the sender P1 is not reduced.

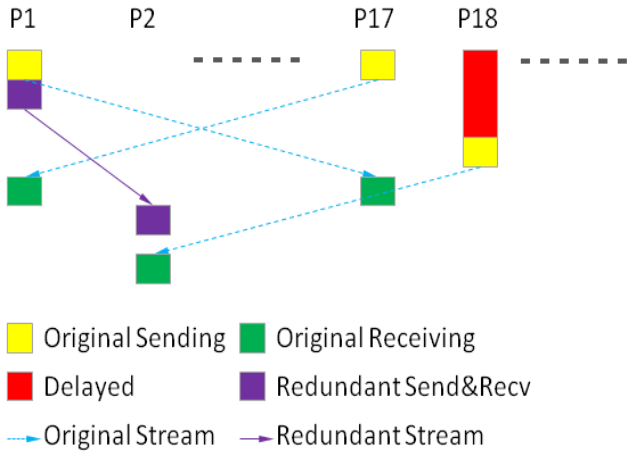


Fig.4 Properties of redundant data exchange

3. The Model

In this section we introduce a general model base on LogGOPS model to simulate the collective operation, a noise model to simulate the effect of noise, and a common computation model of supercomputer system.

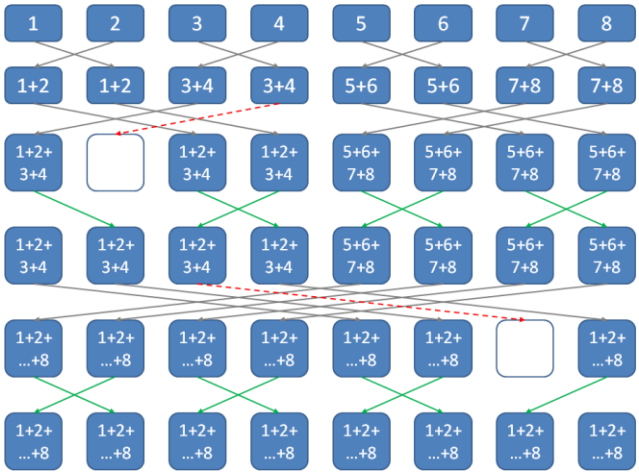


Fig.3 Macroscopic example of the proposed method

3.1 LogGOPS Model

LogGOPS model [15] is a member of LogP model family, in our simulation the size of data is very small, so LogGOPS model is more accurate. **Table.1** shows all the seven parameters and their interpretations. In our simulation, g is set to 0 due to the targets of any two continuous sending are different, so g overlap with target changing; O is also set to 0 because the data size we use is 8 bits and it is too small that o can overlap it and by this idea S is set to larger than 8 bits. Here are the other parameters that we use: L=1e-6s, G=1e-9s, o=1e-9s. **Fig.5** shows the scenario of a send receive event. Notice that in this paper P stands for period, so we set N to represent Quantity of processes.

Table.1 Parameters of LogGOPS model

L	Maximum network latency between every two endpoints
o	CPU overhead, o_s and o_r for send and receive respectively
g	Inter-message gap between two messages (1/g = message-rate)
G	Gap per byte (1/G = bandwidth)
O	CPU overhead for send and receive per byte
P	Quantity of processes
S	Maximum data size of one communication

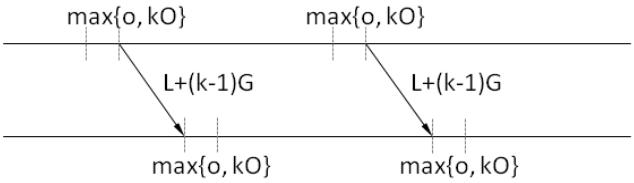


Fig.5 Scenario of a send-receive event

3.2 Noise Model

There are many kinds of noise, interrupts, daemons, page faults, cache missing and so on. But for OS of supercomputer, most of daemons are unnecessary, and page faults and cache missing can be controlled by application. So we mainly simulate timer interrupt in our model, which occurs approximately periodically [7], so let us assume that this kind of noise on all processors

occurs exactly periodically, with period (P) and duration (D). It should be noticed that, although all processors are homogeneous, which means they have same noise period and duration, time of occurrence of noises on different processors are not the same.

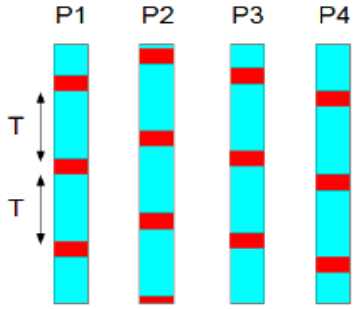


Fig.6 Noise model

3.3 Computing-Communicating Cycle Model

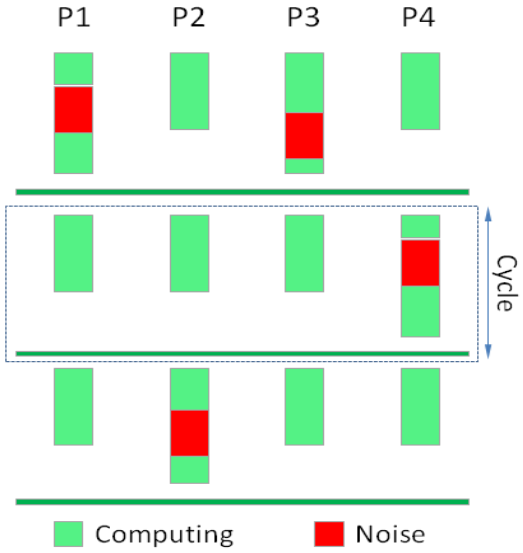


Fig.7 Cycle Model

We assume that the application balances the workload perfectly in each computing phase and controls the page faults and cache missing well, in which case applications can obtain the best performance. Let T represents the entire time consuming of the application, W represents the workload in one cycle, t_{cm} represents the time consuming of each round in butterfly algorithm. In ideal noiseless case, it is easy to know that $T = \sum_{i=0}^{n}(W + t_{cm}K)$. While in noisy case, let T_{cp} represents the distribution of time consumption of each processor in computing phase. Therefore

$$T_{cp} = \begin{cases} W + (W/P + 1) \times D & \text{with probability } (W\%P)/P \\ W + (W/P) \times D & \text{with probability } (P - W\%P)/P \end{cases}$$

(here / represents quotient and % represents remainder.)

It means that some of processors will affect by noise one more time than the others.

Once a processor ends its computing phase, it enters the communicating phase, in which it executes collective operation. After communication, it starts a new computing phase of the next computing-communicating cycle.

4. Evaluation

In the previous Section, we introduced the model of our simulation, but we still need some assumption for other parameters: the memory bandwidth is 10 GB/s; the network is full-bisection with bandwidth which is 1 GB/s and latency which is 1 microsecond; the size of data is small and unchanged after the binary operation; and the distances between any two processors are the same. Besides, we also assume $N = 2^K$ to simplify the simulation. After that, we evaluate our model by changing the value of D, P and W.

To get high accuracy, we use a time minimum heap as an event engine and initialize it by registering a start up event at time of zero for each processor, and keep popping and processing the event which has the smallest timestamp, if new event created then insert it into the time minimum heap, till the heap has no node.

We evaluate three cases in our experiment, i.e. the noiseless case, conventional case and proposed case. Noiseless case means there is no noise in the communicating phase, but in the computing phase, noises still exist. So the noiseless case is the optimal case that the communicating phase can be optimized to. The no-optimization case is original algorithm with noise in communicating phase and optimization case is the proposed algorithm.

Fig.8 presents the results of the simulation of 10 computing-communication cycles, the results shows that with the scale of computing system increasing, the performance decreasing strongly, but the recovered performance is also significant. At the same time, the time cost is relatively small when the effect of noise is insignificant.

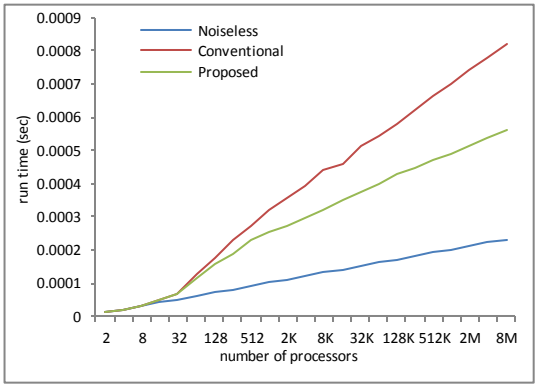
Fig.9 presents the overhead caused by noise of 8M processors, it shows that the overhead increases almost linearly when the number of cycles increases, and the recovery ratio is nearly 0.4. Because we assume all processors start at the same time so the time cost of 1st cycle is a little larger than the following cycles, but after the 1st cycle the slope of the time cost line tend to be constant.

5. Conclusion and Future Work

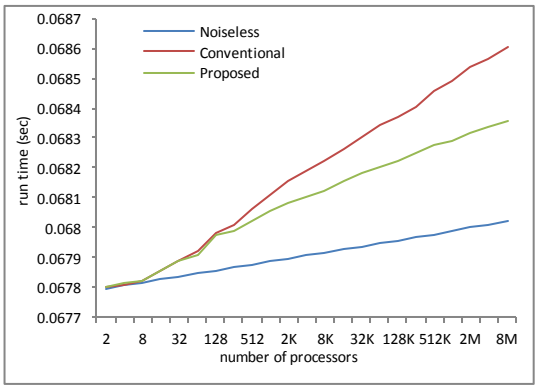
In this paper, we analyzed the butterfly algorithm which is widely used in all-reduce operations, and proposed a new method which is adding some redundant data exchange between original data exchange in butterfly algorithm to decrease the shortcoming of the original algorithm, and the recovery ratio is nearly 0.4 in our simulation.

However, the Redundant Exchange Round Set we use is the all rounds except the 1st round, which means it sends redundant data after every round except the 1st one, so if we can send it dynamically it may work better. Besides, the data size in our algorithm is small and constant, so if it increases after each round then the situation will be totally different.

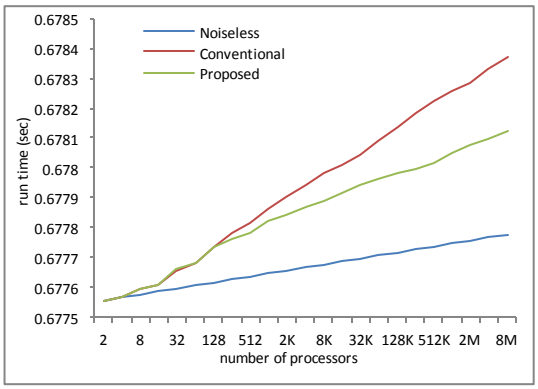
In section 3, we present some models to simulate the above algorithms. LogGOPS model is widely used in communication simulation in supercomputer system, the noise model we use is a periodical noise model with different starting point, and the



(a) Period=0.01s Duration=0.0001s Workload=0s

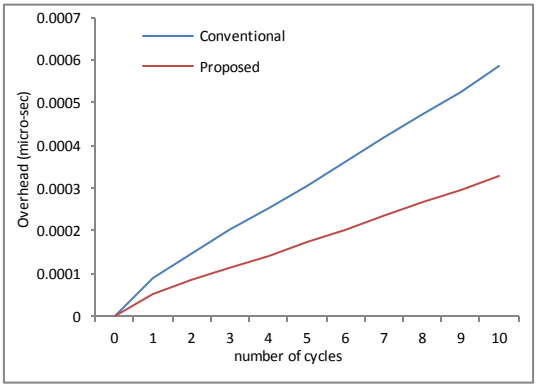


(b) Period=0.01s Duration=0.0001s Workload=0.00666667s

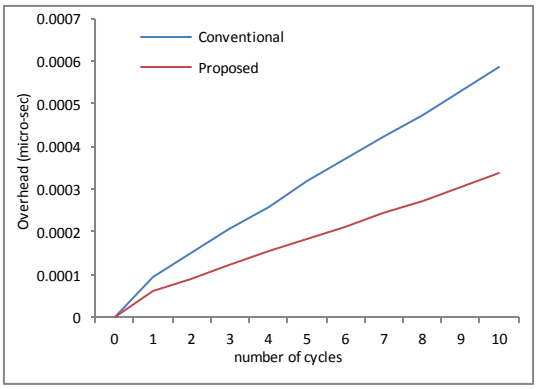


(c) Period=0.1s Duration=0.0001s Workload=0.066667s

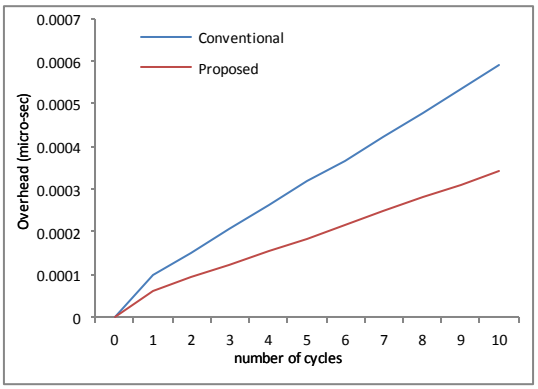
Fig.8 Run time of 10 cycles



(a) Period=0.01s Duration=0.0001s Workload=0s



(b) Period=0.01s Duration=0.0001s Workload=0.00666667s



(c) Period=0.1s Duration=0.0001s Workload=0.066667s

Fig.9 Overhead caused by noise of 8M processors

entire computation model is computing-communicating cycle model, which is simple but representative. At last, we need to implement the proposed method on real machine.

Acknowledgments Thanks to Xiaochen Tian, Cheng Luo, Takashi Hamada, Yoshinori Tamada, and other members in Suda Lab. It's hard to move this research forward without your help. This work is partially supported by JST CREST project "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Computing Systems", and MEXT Grand-in-Aid Project "Materials Design through Computics; Complex Correlation and Non-equilibrium Dynamics".

Reference

[1] Petrini, F., Kerbyson, D. J. and Pakin, S.: The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, ACM/IEEE Conference on Supercomputing (SC'03), Phoenix, Arizona, USA, (2003).

[2] Jones, T., Tuel, W., Breneer, L., Fier, J., Caffrey, P., Dawson, S., Neely, R., Blackmore, R., Maskell, B., Tomlinson, P. and Roberts, M.: Improving the Scalability of Parallel Jobs by Adding Parallel Awareness to the Operating System, ACM/IEEE conference on Supercomputing (SC'03), New York, NY, USA, (2003).

[3] Gioiosa, R., Petrini, F., Davis, K. and Lebaillif-Delamare, F.: Analysis of System Overhead on Parallel Computers, the 4th IEEE

International Symposium on Signal Processing and Information Technology (ISSPIT 2004), Rome, Italy, pp. 387-390 (2004).
[4] “Top500 Supercomputer Site”, http://www.top500.org/
[5] Jone, L. Brenner, B. and Fier, J. M.: Impacts of Operating Systems on the Scalability of Parallel Applications, Tech. Rep. UCRL-MI-202629, Lawrence Livermore National Laboratory, (2003).
[6] Akkan, H., Lang, M. and Liebrock, L. M.: Stepping Towards Noiseless Linux Envirnment, the 2nd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS’12), New York, NY, USA, (2012).
[7] Beckman, P., Iskra, K., Yoshii, K., Coghlan, S. and Nataraj, A.: Benchmarking the Effects of Operating System Interference on Extreme-Scale Parallel Machines, Cluster Computing, pp. 3-16 (2008).
[8] Tsafrir, D., Etsion, Y., Feitelson, D. G. and Kirkpatrick, S.: System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications. the 19th annual international conference on Supercomputing, pp. 303-312(2005).
[9] Kale, V. and Gropp, W.: Load Balancing for Regular Meshes on SMPs with MPI, the 17th European MPI users' group meeting conference, pp. 229-238 (2010).
[10] Hack, J., Rosinski, J., Williamson, D., Boville, B. and Truesdale, J.: Computational Design of the NCAR Community Climate Model, Parallel Computing, pp. 1545–1569 (1995).
[11] Thakur, R. and Gropp, W.: Improving the Performance of Collective Operations in MPICH, the 10th European PVM/MPI User’s Group Meeting, Venice, Italy, pp. 257-267 (2003).
[12] Beckman, P., Iskra, K., Yoshii, K. and Coghlan, S.: The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale, 2006 IEEE International Conference, Barcelona, Spain, pp. 1-12 (2006).
[13] Frachtenberg, E., Feitelson, D. G., Petrini, F. and Fernandez, J.: Flexible coscheduling: mitigating load imbalance and improving utilization of heterogeneous resources, Parallel and Distributed Processing Symposium, (2003).
[14] Agarwal, S., Yoo, A.B., Choi, G.S., Das, C.R. and Nagar, S.: Co-ordinated coscheduling in time-sharing clusters through a generic framework, IEEE International Conference on Cluster Computing, HK, pp. 84-91 (2003).
[15] Hoefler, T., Schneider, T. and Lumsdaine, A.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Washington, DC, USA, pp.1-11 (2010).
[16] 松本英樹, 須田礼仁, ジッタの影響を緩和する集団通信アルゴリズム, 情報処理学会研究報告, Vol. 2012-HPC-137, No.19, 第137 回 HPC 研究会, (2012).
[17] 松本英樹, 須田礼仁, バタフライの中間に冗長なデータ交換を行いジッタの影響を緩和する集団通信アルゴリズム, 2013 年ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS 2013), ポスター発表.(2013).