Energy-Performance Modeling of Speculative Checkpointing for Exascale Systems

Muhammad ALFIAN AMRIZAL††, Atsuya UNO†††, Nonmembers, Yukinori SATO††††, Hiroyuki TAKIZAWA††††, a) and Hiroaki KOBAYASHI†††††, Members

SUMMARY Coordinated checkpointing is a widely-used checkpoint/restart protocol for fault-tolerance in large-scale HPC systems. However, this protocol will involve massive amounts of I/O concentration, resulting in considerably high checkpoint overhead and high energy consumption. This paper focuses on speculative checkpointing, a CPR mechanism that allows for temporal distribution of checkpoings to avoid I/O concentration. We propose execution time and energy models for speculative checkpointing, and investigate energy-performance characteristics when speculative checkpointing is adopted in exascale systems. Using these models, we study the benefit of speculative checkpointing over coordinated checkpointing under various realistic scenarios for exascale HPC systems. We show that, compared to coordinated checkpointing, speculative checkpointing can achieve up to a 11% energy reduction at the cost of a relatively-small increase in the execution time. In addition, a significant energy-performance trade-off is expected when the system scale exceeds 1.2 million nodes.

key words: checkpoint/restart, coordinated checkpointing, speculative checkpointing, exascale, performance model, execution time, energy consumption

1. Introduction

High-performance computing (HPC) systems are now rapidly heading toward exascale for enabling finer-grained scientific simulations. However, the growth in size and complexity of these exascale HPC systems will also bring new challenges. Two main challenges are reliability and energy efficiency.

Future exascale HPC systems will be hit by errors/ faults much more frequently than petascale systems [1]. The MTBF (mean time between failures) of future systems is projected to shrink to tens of minutes [2]. Thus, fault-tolerance has become more important than ever for future exascale HPC systems.

The de-facto standard fault-tolerance technique in HPC systems is checkpoint/restart (CPR). CPR periodically writes the state of all the parallel processes of an HPC application to checkpoint files, which obviously requires extra storage space, power consumption, and time. The linking of space, power and time suggests that there will be some interplays between energy and performance.

Although many protocols exist for CPR, a coordinated checkpointing protocol [3] is usually chosen due to its relative simplicity in implementation [4]. Several optimization techniques have been proposed to increase the effectiveness of coordinated checkpointing [5–7]. In particular, we are interested in one optimization technique called speculative checkpointing [6], a technique that allows for temporal distribution of checkpoings to avoid I/O concentration and hence potentially reduce the checkpoint overhead of coordinated checkpointing.

In this paper, we create energy-performance models to investigate the execution time and the energy consumption characteristics of speculative checkpointing. Our evaluation results show that speculative checkpointing can achieve a considerably large reduction in execution time and energy consumption for exascale systems. In addition, significant energy-performance trade-offs can be observed in speculative checkpointing under certain scenarios of next-generation HPC systems.

The contributions of this paper are as follows:

• We develop mathematical models for computing the execution time and the energy consumption of an application when speculative checkpointing is adopted.

• We derive the optimal checkpoint period that optimize execution time and the optimal checkpoint period that optimize energy consumption.

• We perform energy consumption measurements on the K computer, which is the 7th world’s fastest supercomputer system at the time this paper is submitted [8], to obtain several important system-level parameters.

• We analyze the behaviors of speculative checkpointing and discuss its energy-performance trade-off by exploring the model with a set of expected parameters of future exascale HPC systems.

The rest of this paper is organized as follows. Section 2 briefly introduces the concept of speculative checkpointing. Section 3 describes speculative checkpointing models and their optimizations in terms of time and energy. Section 4 shows evaluation results and discusses the energy-performance trade-off. Section 5 reviews some related work. Finally, the conclusion of this paper is stated in Sect. 6.
2. Speculative Checkpointing

Over the past decades, CPR has been used to periodically save the state of a long-running HPC application that runs from few hours to few months. The application is checkpointed to a stable storage in order to avoid having to restart from the beginning in the case of a failure. Coordinated checkpointing is a traditional CPR protocol that has been widely-used in HPC systems. In Coordinated checkpointing, the application needs to be suspended before the checkpoint data is written to a shared stable storage such as a parallel file system (PFS). As the scale of the HPC system grows, the memory usage of the application also grows, and hence the amount of data needed to be checkpointed usually increases proportionally. Simultaneously writing this huge amount of checkpoint data will degrade PFS performance due to contention[9]–[11], leading to a longer checkpointing time of the application and a large energy consumption of the system. To make matters worse, the power dissipated to execute I/O transfer, such as checkpointing, is predicted to dramatically increase in exascale systems[12], making coordinated checkpointing less feasible for future HPC systems.

One popular optimization technique to reduce a huge amount of checkpoint data in coordinated checkpointing is by performing incremental checkpointing[5]. In incremental checkpointing, only the updated data are checkpointed whenever a checkpoint is taken. However, current incremental methods cannot achieve a significant decrease in checkpoint file size due to their reliance on a page protection mechanism[3]. Speculative checkpointing[6] has been proposed for an iterative class of HPC applications (loop-centric scientific programs) to further reduce I/O contention during coordinated checkpointing. The idea of speculative checkpointing is that, during the interval of two consecutive coordinated checkpointings, the user or the system will predict whether a write to a particular data/memory page will be the last write prior to the next coordinated checkpointing timing and thus this memory page can be speculatively checkpointed earlier, overlapping it with application execution.

The effectiveness of speculative checkpointing substantially depends on how accurately we can predict the last write for a particular memory page. Several tools/algorithms have been developed for this purpose. Agarwal et al. proposed AIC: a tool that can dynamically predict changes in memory pages with a high accuracy based on past memory access patterns[13]. Nicolae et al. proposed an algorithm for predicting the order of memory page modifications and used the prediction results to save memory pages in an asynchronous fashion, i.e., overlapping memory page writes with computation[14]. Their experiments show more than 40% overhead reduction during coordinated checkpointing timing. In addition, last writes can also be predicted if we assume the application can be profiled in advance. For example, Exana[15] is one of the profiling tools that can record every memory access of an application. Therefore, the user or the system can build a last write prediction function by using such a profiling tool, and hence last writes are predictable with a certain accuracy.

The last write prediction can be either correct or incorrect. Figure 1 shows the case when the predictions are correct, hence successfully spreading out the checkpointing I/O. When a prediction to a particular memory page is incorrect, i.e., the memory page is written again after it had been speculatively checkpointed earlier, such a miss-prediction must be detected and the memory page must be re-checkpointed during coordinated checkpointing. Altogether, with a good last write predictor, we may achieve a large reduction in the checkpoint overhead.

3. Execution Time and Energy Models

In this section, we propose mathematical models of speculative checkpointing. First, we introduce all the model parameters. We divide the parameters into two categories: checkpointing model parameters (resilience-related parameters) and energy model parameters (power-related parameters). Using these parameters, we show how to compute the execution time and energy consumption along with the optimal periods that minimize them.

3.1 Checkpointing Model Parameters

We first consider a parallel application executed on a cluster with a shared stable checkpoint storage. The application performs coordinated checkpointing at a fixed interval after some amount of computation has been performed. This corresponds to a full application run partitioned into execution periods of duration \( T_C \). Every execution period consists of a checkpoint phase of duration \( C \) and a computation phase of duration \( (T_C - C) \) as shown in Fig. 2 (a).

Next, we consider the case where coordinated checkpointing is combined with speculative checkpointing as in Fig. 2 (b). We assume that, between any two consecutive coordinated checkpointings, the application actively performs speculative checkpointing on each memory page subjected to checkpoint based on its predicted last write timing. For simplicity, we assume that the speculative checkpointing is evenly distributed throughout the computation phase. Let \( f_p \)
be a ratio of memory pages whose last writes are predicted incorrectly (false prediction). During coordinated checkpointing timing, all we have to do is to re-checkpoint these miss-predicted memory pages. Hence, the duration of the checkpoint phase reduces to \( f_p C \), where \( 0 \leq f_p \leq 1 \). We also call \( f_p \) a speed-up factor of coordinated checkpointing.

Agarwal et al. [13] observed that, for most typical HPC applications, changing the length of an execution period will not have a significant impact on a reduction in the time of each checkpoint phase (speed-up factor \( f_p \)). This is because such applications are iterative in nature and hence a sufficiently good predictor, such as the one found in [14], can be easily developed. Based on these facts, we assume that \( f_p \) remains constant throughout the application run.

Unlike coordinated checkpointing, speculative checkpointing is performed asynchronously, i.e., the checkpoint data are saved to the stable storage on the fly, overlapped with the computation phase. Such an asynchronous event may cause a background jitter that interferes the application run and degrades its performance [16], especially when the application is run on a large-scale cluster [17], [18]. Considering this fact, we create a model where the application is in the condition of either normal run (no jitter) or a slow-down run (due to jitter). To model this situation, we introduce a slow-down factor \( \phi \), where \( \phi \geq 1 \). When computation and checkpoint are overlapped, the execution normally taking \( t \) seconds will run slower and require \( \phi t \) seconds. In Fig. 2 (b), this slow-down run is represented by the red line, while the normal run is represented by the black line. Due to these slow-down runs, the computation phase of the application becomes longer. We call the increased time of computation phase slow-down overhead. Since we assumed that the speculative checkpointing is performed on all memory pages subjected to checkpoint, the checkpoint time with the total length of \( C \) must be overlapped with the computation phase. Hence, the slow-down overhead is \( (\phi - 1)C \). Altogether, the speeded-up checkpoint phase and the slowed-down computation phase will alter the duration of the execution period. Such an execution period with speculative checkpointing is denoted by \( T_S \).

Next, we have to consider failures. First, we assume that the application runs on a system whose MTBF (Mean Time Between Failures) is \( M \). Note that, when we assume that failure intervals follow exponential distribution, if the system consists of \( N \) identical compute nodes, and MTBF of each node is \( M \), then \( M = \frac{M}{N} \). When a failure occurs, the application can only be restarted from the last coordinated checkpointing. This is because the correctness of the checkpoint file can only be guaranteed at coordinated checkpointing as last write miss-prediction may occur during speculative checkpointing. Let \( R \) be a restart time, i.e., the time to read this last stored checkpoint file.

### Table 1: Summary of model parameters

| notation | description |
|----------|-------------|
| \( T_S \) | Execution period with speculative checkpointing |
| \( C \) | Checkpoint time of coordinated checkpointing |
| \( f_p \) | Speed-up factor of coordinated checkpointing |
| \( \phi \) | Slow-down factor due to jitter |
| \( R \) | Restart time |
| \( M \) | Mean time between failures |
| \( M_t \) | Mean time between failures of individual compute node |
| \( N \) | Number of compute nodes |
| \( P_{\text{sta}} \) | Static power |
| \( P_{\text{CPU}} \) | Power to perform computation using CPU |
| \( P_{\text{I/O}} \) | Power to perform checkpoint or restart |
| \( E_{\text{tot}} \) | Total energy consumption |
| \( T_{\text{tot}} \) | Total execution time (with checkpointing and failures) |
| \( T_{\text{ori}} \) | Original execution time (no checkpointing and no failure) |
| \( T_{\text{CPU}} \) | Total CPU usage time |
| \( T_{\text{I/O}} \) | Total I/O system usage time |
| \( T_{\text{opt-Time}} \) | Execution period for minimal execution time |
| \( T_{\text{opt-Energy}} \) | Execution period for minimal energy consumption |

### 3.2 Energy Model Parameters

Throughout its execution, the application will perform these three operations: computing, checkpointing (either coordinated or speculative), or restarting. Each of these operations consumes a different amount of power. To compute the total energy consumption, we need to consider the power consumption of these operations. To this end, we define:

- **\( P_{\text{sta}} \):** this is a static power consumption when the system is turned on.
- **\( P_{\text{CPU}} \):** this is a power consumption when the system performs computation.
- **\( P_{\text{I/O}} \):** this is a power consumption when the system performs any type of I/O transfer operation such as checkpointing or restarting from a failure.

The basic model parameters under the time and power consideration are summarized in Table 1.

### 3.3 A Model of Execution Time

We denote the original execution time of a parallel application by \( T_{\text{ori}} \) and we assume that, in general, an application user can estimate with a certain high accuracy his/her own application original execution time based on experimental analysis [19]. Note that \( T_{\text{ori}} \) does not include any other fault-tolerance-related costs such as a checkpoint overhead and time lost due to failures. Our objective is to compute the expected total execution time \( T_{\text{tot}} \) (accounting both for...
checkpointing and for failures) and to find the value of $T_S$ that minimizes $T_{tot}$.

The total execution time $T_{tot}$ of an application depends on the number of coordinated checkpoints and the execution period $T_S$. There are two overheads for every $T_S$, a slowdown overhead and a coordinated checkpoint overhead. The total of these two overheads is $(\phi + 1 + f_p)C$. Hence, the number of coordinated checkpoints is 

$$T_{no-fail} = \frac{T_{tot}}{T_S - (\phi + 1 + f_p)C}.$$  

Calculating the fault-free execution time $T_{no-fail}$. The derivation of $T_{no-fail}$ is simple. $T_{no-fail}$ is the product of the number of coordinated checkpoints and the execution period $T_S$. Hence, the time lost due to failures can be expressed as follows,

$$T_{fault} = \frac{T_{tot}}{M} \left( R + \frac{T_S}{2} \right).$$

Finally, by substituting Eqs. (2) and (3) into Eq. (1), we can express the total execution time:

$$T_{tot} = \frac{T_{ori}}{(T_S - (\phi + 1 + f_p)C) \left( 1 - \frac{R + TS/2}{M} \right)}.$$  

This equation is minimal when

$$T_S = \sqrt{2(M + R) (\phi + 1 + f_p)C}.$$  

We can call this optimal period $T_{opt-Time}$. When $\phi = 1$, i.e., no slowdown due to background jitters, we obtain a period that is close to Young’s [20] and Daly’s [21] formulas.

3.4 Model of Energy Consumption

Suppose that the total energy consumption of an application is $E_{tot}$. We divide $E_{tot}$ into the following three parts:

$$E_{tot} = E_{sta} + E_{CPU} + E_{I/O}.$$  

Here $E_{sta}$, $E_{CPU}$, and $E_{I/O}$ represent static energy consumption, energy consumption of the CPU, and energy consumption of the I/O system, respectively. In order to compute $E_{sta}$, $E_{CPU}$, and $E_{I/O}$, we must first identify how much time is spent for every operation, i.e., pure-computation; speculative checkpointing; coordinated checkpointing; and restarting. These times are then multiplied accordingly with their power parameters introduced in Sect. 3.2.

**Static energy consumption $E_{sta}$.** The application execution consumes $P_{sta}$ throughout the entire application run. Hence, the corresponding energy cost for $E_{sta}$ is $T_{tot} P_{sta}$.

**Energy consumption of the CPU $E_{CPU}$.** Let $T_{CPU}$ be the total CPU usage time. When the CPU is used, a power overhead of $P_{CPU}$ is induced. $T_{CPU}$ consists of time for coordinated checkpointing, speculative checkpointing, and restarting from failures.

The number of coordinated checkpointing that are performed in a fault-free execution is the same as the number of execution periods, $\frac{T_{tot}}{T_{ori}}$. The I/O time overhead taken by coordinated checkpointing is therefore 

$$\frac{T_{tot} f_p C}{(T_S - (\phi + 1 + f_p)C).}$$

For each failure, the system needs to be restarted. Let assume that the restart process requires R seconds. Then, there will be additional I/O overheads that may differ depending in which phase the failure occurs. With a probability $\frac{T_C - f_p C}{T_S}$, a failure occurs during the computation phase, and the additional I/O overhead is, on average, $\frac{\phi C}{2}$. With probability $\frac{T ثن}{T_S}$, the failure occurs during the checkpoint phase, and the additional I/O overhead is, on average, $\phi C + \frac{f_p C}{2}$. These additional overheads show the average time of checkpointing I/O that must be re-executed. Altogether, we obtain

$$T_{I/O} = \frac{T_{ori} f_p C}{T_S - (\phi + 1 + f_p)C} + \frac{T_{tot}}{M} \left( R + \frac{T_S - f_p C}{T_S} \left( \frac{\phi C}{2} + \frac{f_p C}{T_S} \left( \phi C + \frac{f_p C}{2} \right) \right) \right).$$

Hence, the corresponding energy cost for $E_{I/O}$ is $T_{I/O} P_{I/O}$.

The final expression of the total energy consumption can be obtained by substituting all the above equations into Eq. (6). Hence, we obtain

$$E_{tot} = T_{tot} P_{sta} + \left( T_{ori} + \frac{T_{tot}}{M} \left( \frac{T_S}{2} \right) \right) P_{CPU}.$$
Before we describe how to minimize the energy consumption, it is important to understand that due to the checkpoint-computation overlapping in speculative checkpointing, $T_{tot} \neq T_{CPU} + T_{I/O}$. Hence, minimizing execution time is not equivalent to minimizing energy consumption.

We introduce two new notations to ease the derivation of the optimal period $T_S$ that minimizes $E_{tot}$. The two notations are $\gamma$ and $\delta$ such that $P_{CPU} = \gamma P_{sta}$ and $P_{I/O} = \delta P_{sta}$. Also, let $\alpha = (\phi - 1 + f_p)C$, $\beta = (1 - \frac{R}{M})$, $T'_{tot} = \frac{\delta E_{tot}}{\delta T_S}$, and $E_{tot} = \frac{\delta E_{tot}}{\delta T_S}$. Taking the first derivative from Eq. (4), we obtain:

$$\frac{T'_{tot}}{T_{ori}} = \frac{-\alpha \beta + \frac{T_S^2}{2}}{(T_S - \alpha)^2 (\beta - \frac{T_S}{2M})^2}. \hspace{1cm} (10)$$

Similarly, taking the first derivative from Eq. (9), we obtain:

$$\frac{E'_{tot}}{P_{sta}} = \frac{T_{tot}}{M} \left( \frac{\gamma \phi C}{2} + \frac{\delta}{2T_S} \right) + \frac{\delta (\phi + f_p) f_p C^2}{2T_S} + \frac{\delta}{M} \left( \frac{\delta (\phi + f_p) f_p C^2}{T_S}^2 \right) - \frac{\delta \tau_{ori} f_p C}{(T_S - \alpha)^2}. \hspace{1cm} (11)$$

Then, by using

$$K = \frac{(T_S - \alpha)^2 (\beta - \frac{T_S}{2M})^2}{P_{sta} T_{ori}},$$

we have

$$KE'_{tot} = T_S^2 \left( \frac{\delta R}{2M^2} + \frac{\beta \gamma}{2M} + \frac{\alpha \gamma + \delta (\phi - f_p) C}{4M^2} + \frac{1}{2M} \right) + TS \left( \frac{\delta (\phi + f_p) f_p C^2}{M} + \frac{\delta (\phi + f_p) f_p C^2}{2M^2} \right) - \frac{\alpha \beta (\delta \phi C + 2 \delta R + 2M)}{2M} - \beta^2 \delta f_p C - \left( \frac{\beta}{2M} + \frac{\alpha}{4M^2} \right) \delta (\phi + f_p) f_p C^2. \hspace{1cm} (12)$$

By substituting $E'_{tot} = 0$ into Eq. (12), we will obtain a quadratic polynomial equation in the form of

$$AT_S^2 + BT_S + C = 0,$$

where $A$, $B$, and $C$ are the corresponding coefficients of the polynomial terms in Eq. (12). The value of $T_S$ that minimizes $E_{tot}$ is the positive root of this quadratic polynomial equation. We call this optimal period $T_{opt}$-Energy. In the next section, we use these optimized periods and other parameters from Table 1 to analyze the behavior of execution time and energy consumption of speculative checkpointing.

4. Evaluation

The following evaluation consists of three parts. First, we perform energy consumption measurement on a current large-scale HPC system to obtain some important system-level parameters for our energy model. Then, we use these measurement results to verify the accuracy of our model using a simulation-based approach. Finally, we use our model to explore the energy-performance behavior of speculative checkpointing under various conditions in exascale systems.

4.1 Energy Measurement of a Large-Scale HPC System

To obtain some important system-level parameters, such as $P_{sta}$, $P_{CPU}$, and $P_{I/O}$, we perform energy consumption measurement on the K computer. The system itself consists of 82,944 SPARC64 VIIIIfx oct-core processors running at the clock speed of 2.0 GHz with 1.3 PB of memory. We measure the system’s rack energy consumption for one hour under three different kinds of activities: when it is idle, when it runs HPL (High Performance Linpack) benchmark [22], and when it performs I/O transfer (checkpointing).

The energy consumption measurement results are shown in Table 2. When computation is performed, we observed that the rack energy consumption increases around 4.6 KWh from its idle state energy consumption. Since each rack of K computer consists of 96 compute nodes, hence approximately 48 W of power is dissipated from every compute node during computation. Next, we observed that the rack energy consumptions during idle state and checkpointing are roughly the same, i.e., around 10.2 KWh or approximately 106 W per compute node. This is because each rack of K computer is equipped with I/O nodes that need to be constantly on and hence consumes energy even if the application does not execute I/O transfers. In other words, in the case of K computer, no additional power is required to perform checkpointing. Altogether, we obtained $P_{sta} = 106 W$, $P_{CPU} = 48 W$, and $P_{I/O} = 0 W$.

4.2 Model Verification

To verify the accuracy of our mathematical model, we use an event-driven simulator similar to [23]. While our models are governed by more strict assumptions, the event-driven simulator is closer to a real-case scenario. For example, in the simulator, failures can occur at any moment throughout the entire application run, even during a restart process from

| Table 2 K computer’s rack energy consumptions |
|---------------------------------------------|
| type of activity   | energy consumption |
|-------------------|--------------------|
| Idle              | 10.2 KWh           |
| Computation       | 48.0 KWh           |
| Checkpointing     | 10.2 KWh           |
a failure. In addition, the simulator considers a down-time, which is a time span between the occurrence of failure that stops the system from running, until the system can actually start reading the checkpoint file from the storage.

Figure 3 shows the verification scheme. The simulator takes two inputs: application and failure events. The application is described in terms of its original execution time $T_{ori}$, coordinated checkpointing time $C$, restart time $R$, speed-up factor $f_p$, and slow-down factor $\phi$. The failure events are created using a failure generator that randomly selects time stamps for failure occurrence.

We set the original execution time of the application to one day (24 hours). The application performs coordinated checkpointing once every 200 minutes and performs speculative checkpointing five times in between two coordinated checkpoints. The coordinated checkpointing time and recovery time were set to 10 minutes. The accuracy of the last write predictor for speculative checkpointing was set to 50% ($f_p = 0.5$) and the slow-down factor due to jitter was set to 1.

Before the application is executed, the failure generator randomly selects three different time stamps for the failure events. Upon a failure, the application rolls back to the most recent coordinated checkpoint. When the execution is completed, the simulator returns a measured value. The measured value is the sum of times for pure-computation, speculative checkpointing (computation and background checkpointing), coordinated checkpointing, and restarting. The total energy consumptions is then measured by multiplying these time values with the corresponding power consumption of K Computer from Sect. 4.1.

We compare our model’s prediction of execution time and energy consumption with the simulation results in Table 3. The prediction error in Table 3 indicates how accurate our model is in predicting the last writes of memory pages. The smaller the value of $f_p$, the shorter the checkpoint phase becomes. The $y$-axes in Figs. 4(a) and 4(b) correspond to the maximum reduction in execution time and the maximum reduction in energy consumption, respectively. These maximum reduction values in execution time and energy consumption are obtained by initiating the models with $T_{opt}$, $E_{opt}$, $T_{opt}$-Time, and $T_{opt}$-Energy, respectively. The positive values on the $y$-axis

### Table 3: Comparison of model prediction and simulator measurement

| Predicted Value (Our Models) | Measured Value (Simulator) | Prediction Error |
|-----------------------------|----------------------------|-----------------|
| $T_{tot}$                   | 1832 min                   | 1889 min       | 3.02%           |
| $E_{tot}$                   | 4579 KWh                   | 4729 KWh       | 3.17%           |

4.3 Model Exploration Under Various Speed-Up Factors $f_p$

We now use the models to explore speculative checkpointing in a more general context. For this purpose, we choose realistic values for all of the model parameters from current projections for exascale platforms [24]-[26]. These include all energy model parameters ($P_{sta}$, $P_{cpu}$, and $P_{I/O}$), all checkpointing model parameters ($C$, $f_p$, $\phi$, and $R$), and system’s MTBF $M$.

Due to the extremely high power requirements, DoE has limited the power budget of exascale systems to 20 MW. With 1 million compute nodes, this represents an average peak power of 20 W per node. In an idle state, generally, compute nodes still draw at least 60% of their peak power [29]. Hence, it is reasonable to assume that 60% of this peak power is used for operating the platform, i.e., $P_{sta} = 12$ W, and the other 40% is for computing, i.e., $P_{cpu} = 8$ W. Since power requirement for I/O operations, i.e., for reading and writing checkpoint data, is expected to be more and more expensive [12], we set $P_{I/O}$ to an order of magnitude bigger than $P_{cpu}$.

K computer, with $N = 82,944$ processors, is reported to have a processors monthly failure rate of $1.2 \times 10^{-4}$ [30]. This is equivalent to approximately one fault per three days, which leads to a MTBF of individual processor $M_i$ approximately equal to $\frac{365 \times 10^3}{1.2 \times 10^{-4}} \approx 868$ years. With 1 million processors, the MTBF will go down to approximately $M = 400$ min. As future systems will become larger and larger, we vary the number of processors from $N = 1$ million to $N = 10$ million. Hence, the MTBF $M$ varies from $M = 400$ min down to $M = 40$ min.

Oyama et al. observed 26.5% slow-down due to jitter on HPC applications that frequently performing disk I/O [16]. Hence, it is reasonable to set $\phi = 1.25$. As for the checkpoint and restart overheads ($C$, $R$), we set both to 10 min.

Using these parameter settings, first, we compare two kinds of CPR mechanisms, a coordinated checkpointing only mechanism (Coor-Only) and a coordinated checkpointing combined with a speculative checkpointing mechanism (Coor-Spec). More specifically, we discuss which of the two mechanisms leads to a better execution time and/or less energy consumption for exascale systems.

Figure 4 shows the performance and energy improvement of Coor-Spec in comparison with those of Coor-Only under different parameter settings. The $x$-axis corresponds to the speed-up factor $f_p$. Here, $f_p$ indicates how accurate we can predict the last writes of memory pages. The smaller the value of $f_p$, the shorter the checkpoint phase becomes. The $y$-axes in Figs. 4(a) and 4(b) correspond to the maximum reduction in execution time and the maximum reduction in energy consumption, respectively. These maximum reduction values in execution time and in energy consumption are obtained by initiating the models with $T_{opt}$ and $E_{opt}$, respectively. The positive values on the $y$-axis
Figures 4 (a) and 4 (b) indicate that, with a good last write predictor (smaller $f_p$), a considerably large reduction in execution time and energy consumption can be obtained by Coor-Spec. Especially when the value of $f_p = 0$, up to 45% reduction in execution time and 60% reduction in energy consumption are observed under a high-failure rate environment ($M = 40$ min). However, it is practically impossible to obtain the value of $f_p$ as small as 0. In the original work of speculative checkpointing [6], with a perfect last write predictor, the authors observed that the coordinated checkpointing overhead is reduced to approximately 60% ($f_p = 0.6$) of its original overhead on the NAS Parallel Benchmark (NPB) [31]. This is equivalent to up to a 10% reduction in execution time (Fig. 4 (a)) or up to a 15% reduction in energy consumption (Fig. 4 (b)), depending whether we choose to optimize execution time or energy consumption. The threshold for execution time can be observed from Fig. 4 (a). This figure shows that Coor-Spec has a shorter execution time than Coor-Only when $f_p < 0.75$. When $f_p = 0.75$, the benefit obtained from the speed-up of the checkpoint phase is cancelled out by the slow-down of the computation phase due to jitter. Strictly speaking, in terms of execution time, speculative checkpointing can be beneficial as long as $\phi + f_p < 2$. When $\phi + f_p = 2$, the length of the execution period of Coor-Spec will be the same as that of Coor-Only, and hence the reduction in execution time becomes 0% with no relation to the value of $M$.

Unlike the execution time, the threshold for energy consumption is harder to quantify since it depends on the value of $M$. In general, this threshold is slightly bigger than its execution time counterpart, as shown in Fig. 4 (b). When $M$ is small, the reduction in energy consumption becomes 0% when $f_p \approx 0.8$ or $\phi + f_p \approx 2.05$, and for bigger $M$ it becomes 0% when $f_p \approx 0.9$ or $\phi + f_p \approx 2.15$.

Next, we show how this threshold fits in real implementations. To this end, we analyze the experimental results using two real HPC applications presented by Nicolae et al. in [14]. The two applications used in their experiments are CM1 [32] and MILC [33]. 42 compute nodes of Grid’5000 system were used as the experimental testbed. 32 nodes were used to run the applications while the other 10 nodes were configured to act as a PFS. For each application, two experiments were performed. The first one is with the Coor-Only mechanism and the second one is with the Coor-Spec mechanism.

The summary of the experimental results with application CM1 are shown in Fig. 5. The original simulation time of CM1, i.e., the execution time with checkpointing deactivated, was fixed to 180 seconds and a total of three coordinated checkpoints were taken throughout the application’s execution both in Coor-Only and Coor-Spec. In Coor-Spec, a simple heuristic was used to predict the order of memory pages modification: in the first interval of 50 seconds, the memory access pattern was recorded. Then, based on the recorded data, the memory pages are speculatively and asynchronously saved to a storage during the computation phases of the next intervals. The computation phases of Coor-Spec are $\phi$ times longer (on average) than those of Coor-Only due to the background jitter (denoted by the red
Therefore, in the case of CM1, we define the values of a short-period trial test with some checkpoints and investigate long-running HPC applications. For example, users can run whether to use speculative checkpointing or not with their thresholds from our models can guide users on deciding accurate enough to describe the behavior of speculative Coor-Only in MILC (see Fig. 5 in [14]).

The total time of Coor-Only (286.8 seconds). Similar results are also obtained in the case of MILC. For MILC, \(\phi\) and \(f_p\) are 1.016 and 0.641, respectively, which also lead to \(\phi + f_p < 2\). Again, our theoretical conditional threshold matches well with the experimental results. The total time of Coor-Spec is approximately 200 seconds shorter than that of Coor-Only in MILC (see Fig. 5 in [14]).

These results further emphasize that our models are accurate enough to describe the behavior of speculative checkpointing in real HPC applications. The conditional thresholds from our models can guide users on deciding whether to use speculative checkpointing or not with their long-running HPC applications. For example, users can run a short-period trial test with some checkpoints and investigate the values of \(\phi\) and \(f_p\) before fully running their applications.

4.4 Model Exploration Under Various Power Settings

Next, we explore our models to study the energy-performance trade-off of Coor-Spec under various power consumption configurations. A key metric for the model exploration is a ratio of I/O power to computation power, \(r\). We define \(r\) as

\[
r = \frac{P_{\text{sta}} + P_{1/\text{O}}}{P_{\text{sta}} + P_{\text{CPU}}} = \frac{1 + \delta}{1 + \gamma}.
\]

This metric indicates the power gap between I/O and computation. For example, when \(P_{\text{sta}} = 12\) W, \(P_{\text{CPU}} = 8\) W, and \(P_{1/\text{O}} = 80\) W, we would get \(r = 4.6\). The bigger the value of \(r\), the more expensive the checkpointing power cost will be in comparison to the computation power cost.

Finally, we evaluate the impact of the ratio \(r\) on the energy-performance trade-off based on the time-cost, \(T\), and the energy-saving, \(E\), defined as follows,

\[
T = \frac{T_{\text{tot}} - E_{\text{tot}}}{T_{\text{tot}} - T_{\text{opt}} - E_{\text{opt}}} = \frac{T_{\text{tot}} - E_{\text{tot}}}{T_{\text{tot}} - T_{\text{opt}} - E_{\text{opt}}} - 1.
\]

\[
E = \frac{E_{\text{tot}} - E_{\text{opt}}}{E_{\text{tot}} - E_{\text{opt}}} = \frac{E_{\text{tot}} - E_{\text{opt}}}{E_{\text{tot}} - E_{\text{opt}}} - 1.
\]

Here, \(T_{\text{tot}} - T_{\text{opt}}\) and \(E_{\text{tot}} - E_{\text{opt}}\) represent the total execution time and the total energy consumption, respectively, when \(T_{\text{opt}}\) is chosen as the execution period. While \(T_{\text{tot}} - E_{\text{opt}}\) and \(E_{\text{tot}} - E_{\text{opt}}\) represent the total execution time and the total energy consumption when \(E_{\text{opt}}\) is chosen as the execution period. The time-cost indicates the percentage of performance degradation from the optimal execution time, and the energy-saving indicates the percentage of energy saved at the cost of its corresponding performance degradation.

Figure 6 shows the effect of \(r\) on performance and energy in speculative checkpointing under different parameter settings. The x-axis corresponds to the ratio \(r\). The y-axes in Fig. 6(a) and 6(b) correspond to the time-cost and energy-saving, respectively. We increase \(r\) by increasing \(P_{\text{CPU}}\) and keeping \(P_{\text{sta}}\) and \(P_{\text{CPU}}\) constant. We evaluate the time-cost and energy-saving under various values of \(M\).
Figure 6 clearly shows that, when $r$ increases, the energy-saving increases faster than the time-cost. When $r$ is large enough, we can observe a significant trade-off in performance and energy. The energy-saving reaches more than 10% with only an increase in execution time for less than 5%. This indicates that, as the power gap between I/O and computation becomes bigger, choosing $T_{opt-Energy}$ as an execution period may lead to a significant gain in energy at the cost of a relatively small increase in execution time. However, when $r$ is small, no significant trade-off is observed. For example, when $r$ is less than 2, the time-cost and energy-saving are both close to 0%. This is because, although extra I/O power overheads are dissipated by the miss predictions of speculative checkpoints, their values are not big enough to cause a significant change in the total power consumption of the entire execution. Hence, minimizing execution time will also lead to a suboptimal energy consumption, and vice versa. When $M$ is small (60 min), the coordinated checkpointing must be taken frequently so as to make sure that the application can finish its execution, hence limiting the energy-saving to a value that is almost proportional to the time-cost.

Finally, we study the energy-performance trade-off of Coor-Spec when each node is equipped with a local storage, and hence allowing it to have a high I/O bandwidth capability to create a scalable CPR mechanism [7]. We set the durations of $C$ and $R$ to 1 minute, without relation to the number of compute nodes. This is reasonable provided that each compute node is equipped with a fast local storage such as RAM disk. We vary the number of nodes from $N = 100,000$ to $N = 15$ million. The MTBF of 1 million nodes is set to 1 hour, this value is inversely proportional to the number of nodes $N$, where a larger system has a shorter MTBF and vice versa. We set the maximum of $N$ to 15 million nodes because beyond this scale, the system’s MTBF will become too short so that the system will spend most of its time for checkpointing rather than performing useful computation. Such a case is unlikely to happen in practical HPC systems. We set other parameters to be the same as the previous ones.

Figures 7 (a) and 7 (b) show the time-cost and energy-saving, both as a function of $N$. From these two figures, we can see that there exists a maximum energy-saving when the scale of the system exceeds 1 million nodes. The system scale, in which this maximum energy-saving is obtained, is different depending on the value of $r$. When we vary $r$ from 5 to 10, $N$ varies from 2.5 million to 1.25 million nodes. At maximum, an energy-saving of up to 11% can be obtained for a time overhead of only 6%. If we keep increasing $N$ beyond this maximum value, we can observe that the energy-saving and time-cost start to decrease and converge into one value. It will eventually converge to approximately 2% when the system scale is 15 million nodes. This shows that, there is no significant trade-off when the scale of the system becomes too large. At extreme-scale, the high-occurrence of failures forces the system to perform coordinated checkpointing more frequently. This will limit the option range of the checkpoint period, and hence, limiting the energy-performance trade-off to such an insignificant value.

5. Related Work

Many CPR models exist to compute the optimal checkpoint period that minimizes the total execution time [20, 21, 34, 35]. However, only few models focus on minimizing energy consumption and discuss its trade-off with execution time. Diouri et al. [27] proposed an energy estimation framework when several CPR protocols, such as coordinated, uncoordinated, and hierarchical checkpointing protocols, are available. Their framework will select which protocol is the most energy-efficient one under a certain parameter configuration. Meneses et al. [28] developed performance and energy models for global recovery, message logging, and parallel recovery protocols. Their models revealed that parallel recovery is the most energy-efficient one because it significantly reduces the recomputation time. Our work differs from Diouri’s and Meneses’s ones. Our focus is only on the coordinated checkpointing protocol and its optimization technique, i.e., speculative checkpointing, because coordinated checkpointing is the most widely-used CPR protocol in practice.

Balaprakash et al. [36] developed an energy model for a multi-level checkpointing system [7] that less frequently uses PFS as a checkpoint storage by combining different
storage levels, each with its own checkpoint interval, checkpoint overhead and failure rate. Multi-level checkpointing is orthogonal to speculative checkpointing, and they can be used together. For example, speculative checkpointing can be performed during the interval of two consecutive checkpoints of the same checkpoint level. Currently, our model does not assume a multi-level checkpointing scenario. This could be an interesting topic to explore in our future work.

Aupy et al. [37] developed performance and energy models, and derived an optimal formula for energy consumption in coordinated checkpointing. Our work is different from theirs because we not only assume coordinated checkpointing, but also its optimization technique, i.e. speculative checkpointing. The addition of speculative checkpointing will further complicate the problem. For example, our model shows that it is not always beneficial to add speculative checkpointing to the scenario if we do not have a good enough predictor to predict the memory access pattern of the running application. Furthermore, our model is the more general one compare to theirs because we can also build a coordinated checkpointing-only case from our model by setting \( \phi = f_p = 0 \).

6. Conclusions

In this paper, we have provided a detailed analysis to compute the optimal checkpointing period, when the coordinated checkpointing is combined with speculative checkpointing, a checkpointing strategy where the file I/O transfer is overlapped with computations and distributed throughout application execution. We have considered two different optimization targets: the first one is to minimize the total execution time and the second one is to minimize the total energy consumption. Since the power overhead of computation and I/O are different, we obtain different optimal periods for each optimization target.

We have explored our models with real parameters of current HPC systems and expected parameters of future exascale systems. From the models exploration, we clarified the conditional thresholds, in which speculative checkpointing can be beneficial compared to merely performing coordinated checkpointing. We found out that these thresholds are different, depending whether we choose to optimize execution time or to optimize energy consumption. For execution time optimization, the conditional threshold only depends on the last write predictor accuracy and slow-down overhead due to jitter, while for energy consumption optimization, it also depends on system’s MTBF. Hence, in general, HPC users have to consider these three important factors when deciding whether or not to use speculative checkpointing.

Also, we have studied the impact of the ratio of I/O power to computation power on the interplays between energy consumption and execution time. With current systems’ parameters, we show that speculative checkpointing can save more than 10% of energy with an increase of less than 5% in execution time. The maximum energy savings are expected when the number of nodes in a system is from 1.25 million to 2.5 million (up to 11% energy savings).

Our future work includes analyzing energy and performance characteristics of other fault tolerance protocols, such as replication and message logging, to examine under what scenario these protocols could be beneficial in comparison to the traditional coordinated checkpointing.

Acknowledgements

This research is partially supported by JST CREST An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems and Grant-in-Aid for Challenging Exploratory Research #26540049. The authors would like to thank Prof. Kobayashi and Prof. Egawa of Tohoku University for their meaning-full discussion.

References

[1] S. Amarasinghe, D. Campbell, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, and K. Hill, “Exascale Software Study: Software Challenges in Extreme Scale Systems,” DARPA IPTO, Air Force Research Labs, Tech. Rep, pp.1–153, 2009.
[2] B. Schroeder and G.A. Gibson, “Understanding Failures in Petascale Computers,” Journal of Physics: Conference Series, vol.78, no.1, pp.12–22, 2007.
[3] E.N.M. Elnohazy, L. Alvisi, Y.M. Wang, and D.B. Johnson, “A Survey of Rollbackrecovery Protocols in Message-passing Systems,” ACM Comput. Surv., vol.34, no.3, pp.375–408, 2002.
[4] J. Hursey, “Coordinated Checkpoint/Restart Process Fault Tolerance for MPI Applications on HPC Systems,” PhD thesis, Indiana University, 2010.
[5] J.C. Sancho, F. Pertini, G. Johnson, J. Fernandez, and E. Frachtenberg, “On the Feasibility of Incremental Checkpointing for Scientific Computing,” Proceedings of IPDPS 2014, pp.58–67, 2004.
[6] I. Yamagata, S. Matsuoka, H. Jitsumoto, and H. Nakada, “Speculative Checkpointing: Exploiting Temporal Affinity of Memory Operation,” HPC Asia 2009, pp.390–396, 2009.
[7] A. Moody, G. Bronevetsky, K. Mohror, and B.R. de Supinski, “Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System,” Proceedings of SC10, pp.1–11, 2010.
[8] TOP500.org, “Top 500 Supercomputer Sites,” available on the WWW, 2016. http://www.top500.org/.
[9] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, “Parallel File System Testing for the Lunatic Fringe: The Care and Feeding of Restless I/O Power Users,” Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, pp.3–17, 2005.
[10] T.H. Cormen and C. Leiserson, “Introduction to Algorithms,” The MIT Press, 2001.
[11] J. Shalf, “Using IOR to Analyze the I/O Performance for HPC Platforms,” Proceedings of the 49th Cray User Group (CUG) Conference 2007, pp.1–15, 2007.
[12] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale Computing Technology Challenges,” VECPAR 2010, LNCS, vol.6449, pp.1–25, 2011.
[14] B. Nicolae and F. Capello, “AI-Ckpt: Leveraging Memory Access Patterns for Adaptive Asynchronous Incremental Checkpointing,” Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing, pp.155–166, 2013.

[15] Y. Matsubara and Y. Sato, “Online Memory Access Pattern Analysis on an Application Profiling Tool,” Proceedings of the 2nd International Symposium on Computing and Networking, pp.602–604, 2014.

[16] Y. Oyama, S. Ishiguro, J. Murakami, S. Sasaki, R. Matsumiya, and O. Tatebe, “Reduction of Operating System Jitter Caused by Page Reclaim,” Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers, Article no.9, 2014.

[17] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan, “Operating System Issues for Petascale Systems,” ACM SIGOPS Operating Systems Review, vol.40, no.2, p.29, 2006.

[18] F. Petrini, D.J. Kerbyson, and S. Pakin, “The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8192 Processors of ASCI Q,” ACM Supercomputing, pp.1–17, 2003.

[19] S. Di, M.S. Bouguerra, L. Bautista-Gomez, and F. Capello, “Optimization of multi-level checkpoint model for large scale HPC applications,” Proceedings of IPDPS 2014, pp.1181–1190, 2014.

[20] J.W. Young, “A First Order Approximation to The Optimum Checkpoint Interval,” Communications of the ACM, vol.17, no.9, pp.530–531, 1974.

[21] J.T. Daly, “A Higher Order Estimate of The Optimum Checkpoint Interval for Restart Dumps,” Future Generation Computer Systems, vol.22, no.3, pp.303–312, 2006.

[22] P. Luszczek, J. Dongarra, D. Koester, B. Rabenseifner, J. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, “Introduction to the HPC Challenge Benchmark Suite,” available on the WWW, 2015. http://icl.cs.utk.edu/hpcc/pubs

[23] Z. Zheng, L. Yu, and Z. Lan, “Reliability-Aware Speedup Models for Parallel Applications with Coordinated Checkpointing/Restart,” IEEE Trans. Comput., vol.64, no.5, pp.1402–1415, 2015.

[24] J. Dongarra, P. Beckman, P. Aerts, F. Capello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero, “The International Exascale Software Project: A Call to Cooperative Action by the Global High-performance Community,” International Journal of High Performance Computing Applications, vol.23, no.4, pp.303–322, 2009.

[25] V. Sarkar, et al., “Exascale Software Study: Software Challenges in Extreme Scale Systems,” 2009, White paper available at: ExascaleComputingStudyReports/ECCS%20report%20101909.pdf

[26] US Department of Energy, “Exascale Computing Initiative Update 2012,” available at: http://science.energy.gov/~/media/ascr/ascc/pdf/meetings/aug12/2012-ECI-ASCAC-v4.pdf.

[27] M. el Mechdi Diouri, O. Gluck, L. Lefèvre, and F. Capello, “ECOFIT: A framework to estimate energy consumption of fault tolerance protocols for HPC applications,” 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid13), pp.522–529, 2013.

[28] E. Menezes, O. Sarood, and L.V. Kal’e, “Energy profile of rollback-recovery strategies in high performance computing,” Parallel Computing, pp.536–547, 2014.

[29] X. Fan, W.D. Weber, and L.A. Barroso, “Power Provisioning for a Warehouse-sized Computer,” Proceedings of the 34th Annual International Symposium on Computer Architecture, pp.13–23, 2007.

[30] F. Shoji, S. Matsui, M. Okamoto, F. Sueyasu, T. Tsukamoto, A. Uno, and K. Yamamoto, “Long Term Failure Analysis of 10 Petascale Supercomputer,” HPC in Asia Poster, ISC, 2015.

[31] D.H. Bailey, J.T. Barton, T.A. Lasinski, and H.D. Simon, “The NAS Parallel Benchmarks,” Technical Report RNR-91-002 Revision 2, NASA Ames Research Laboratory, pp.1–71, 1991.

[32] G.H. Bryan and R. Rotunno, “The Maximum Intensity of Tropical Cyclones in Axisymmetric Numerical Model Simulations,” Journal of the American Meteorological Society, vol.137, pp.1770–1789, 2009.

[33] Nersc 6 Procurement Benchmark, available on the WWW. http://www.nersc.gov

[34] A. Duda, “The Effects of Checkpointing on Program Execution Time,” Information Processing Letters, vol.16, no.5, pp.221–229, 1983.

[35] J.S. Plank and M.G. Thomason, “Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems,” Journal of Parallel Distributed Computing, vol.61, no.11, pp.1570–1590, 2001.

[36] P. Balaprakash, L.A.B. Gomez, M.-S. Bouguerra, S.M. Wild, F. Capello, and P.D. Hovland, “Analysis of the Tradeoffs Between Energy and Run Time for Multilevel Checkpointing,” Performance Modeling, Benchmarking, and Simulation of High Performance Computer Systems, pp.249–263, 2014.

[37] G. Aupy, A. Benoit, T. Herault, Y. Robert, and J. Dongarra, “Optimal Checkpointing Period: Time vs. Energy,” CoRR, abs/1310.8456, 2013.

Muhammad Alfian Amrizal is currently a postdoctoral researcher at Cyberscience Center, Tohoku University. His research interests include dependability for HPC systems, novel fault tolerance techniques for exascale computing, performance modelling and optimization. He received the B.E. Degree in Mechanical Engineering, and the M.S. and Ph.D. Degrees in Information Sciences from Tohoku University in 2012, 2014 and 2017, respectively.

Atsuya Uno is currently a team head of System Operation and Development Team, Operations and Computer Technologies Division at RIKEN Advanced Institute for Computational Science. His research interests include system software of high-performance computing systems. He received his B.E. and M.E. degrees in Information Engineering from the University of Tsukuba in 1995 and 1997, respectively. He received his Ph.D. degree in Engineering, also from the University of Tsukuba, in 2000.

Yukinori Sato is a specially appointed associate professor (lecturer) at Tokyo Institute of Technology in Japan. His research interests lie in the broad area of computer system and architecture research from high performance computing to reconfigurable computing with particular emphasis on their software and tools. He received the Ph.D. degree in information Sciences from Tohoku University. From 2007 to 2015, he was an Assistant Professor at Japan Advanced Institute of Technology. He is a member of the IEEE CS, the ACM, the IEICE and the IPSJ.
Hiroyuki Takizawa is currently a professor of Cyberscience Center, Tohoku University. His research interests include high-performance computing systems and their applications. He received the B.E. Degree in Mechanical Engineering, and the M.S. and Ph.D. Degrees in Information Sciences from Tohoku University in 1995, 1997 and 1999, respectively. He is a member of the IEEE CS, the ACM SIGHPC, and the IPSJ.

Hiroaki Kobayashi is currently Professor of the Graduate School of Information Sciences, Tohoku University. His research interests include high-performance computer architectures, supercomputer systems, and their applications. He received the B.E. Degree in Communication Engineering, and the M.E. and D.E. Degrees in Information Engineering from Tohoku University. He is a senior member of IEEE CS, and a member of ACM and IPSJ. He is also an associate member of Science Council of Japan.