Computing and Informatics, Vol. 35, 2016, 369–390

EFFECTIVE SCHEDULING OF GRID RESOURCES USING FAILURE PREDICTION

Woochul Kang, Jibum Kim*

Embedded Systems Engineering Department
Computer Science and Engineering Department
Incheon National University
12-1 Songdo-dong, Yeonsu-gu
Incheon, South Korea
e-mail: {wchkang, jibumkim}@incheon.ac.kr

Abstract. In large-scale grid environments, accurate failure prediction is critical to achieve effective resource allocation while assuring specified QoS levels, such as reliability. Traditional methods, such as statistical estimation techniques, can be considered to predict the reliability of resources. However, naïve statistical methods often ignore critical characteristic behavior of the resources. In particular, periodic behaviors of grid resources are not captured well by statistical methods. In this paper, we present an alternative mechanism for failure prediction. In our approach, the periodic pattern of resource failures are determined and actively exploited for resource allocation with better QoS guarantees. The proposed scheme is evaluated under a realistic simulation environment of computational grids. The availability of computing resources are simulated according to real trace that was collected from our large-scale monitoring experiment on campus computers. Our evaluation results show that the proposed approach enables significantly higher resource scheduling effectiveness under a variety of workloads compared to baseline approaches.

Keywords: Grid computing, resource scheduling, failure prediction, reliability, job execution service

* corresponding author

1 INTRODUCTION

Computational grids [13] consist of diverse computing resources ranging from cheap desktop machines to highly available and high performance servers and clusters. Examples include Condor [35], Entropia [8], Genesis II [1], EGI (European Grid Initiative) [10], and Globus [13]. They are motivated by the needs of large-scale, computation-intensive scientific applications that cannot be provided by resources within a single administrative domain. Today, thanks to such computational grids, researchers and scientists from a variety of disciplines can tap into an enormous amount of distributed computing resources of more than a petaflop [14]. Computational grids have enabled researchers to work on complex scientific problems of a scale larger than ever.

In this work, we assume a job execution environment, such as Condor [35] and Genesis II [26], which deals with the execution of idempotent jobs using various shared resources ranging from desktop PCs to super computers. In such environments, ensuring QoS (Quality-of-Service) properties such as job completion time or availability often requires predicting whether a particular resource such as a host will be functioning over some time interval [4, 27, 28, 32]. For example, if we wish to execute a job J, which takes 10 CPU hours, at time T and guarantee that it will complete at some time $T + 10$ *hours* on a host H with probability P, then we need to determine the probability that the host will continue to function over that time interval. If we cannot find a host that satisfies the requirements P then we might need to replicate the job to two or more hosts so that we can run multiple copies of J and know that with probability P at least one replica of J will complete successfully. The challenging problem in this situation is how we can precisely estimate the probabilities of diverse computing resources. More formally, let Δt be a time interval with a start time and a stop time, and let $P_i(\Delta t)$ be the probability that host H_i will be available over the interval Δt. How do we compute the P_i's? One obvious solution is to exploit statistical models using historical data of *up/down* states of the machines.

However, the problem, as discussed in Section 2, is that statistical models manifest very poor prediction performance if a significant number of periodic events present. For example, a machine becomes periodically unavailable if the machine is automatically rebooted on some schedule, or when the machine is turned off every night, or when weekly backups take the machine off-line. Indeed, in shared computational grid environments, if we model a host as being unavailable or "down" when a user at the keyboard is actively using the machine, one can observe patterns of periodic availability. Our previous study [15, 17] with 729 machines spanning several departments of an institution shows that a large number of machines actually have such regular patterns in their availability.

To address the problems of traditional statistic modeling techniques in failure prediction, we have developed a failure prediction model, called a *filtered failure prediction model (FFP)* [17]. The basic idea behind FFP is to determine the regularity of resource availability and pro-actively exploit the information for failure

prediction. For instance, the information on regular reboots might be exploited for checkpointing of long-running jobs [25]. In FFP, we first determine periodic events from the historical monitoring traces, note them, filter them out, and then pass the remaining events onto a traditional statistical method. Then, to compute $P_i(\Delta t)$ we first check against periodic failures, and if not affected by a periodic failure, we use the statistical method to determine the probability of success.

In our previous work [17], we presented analytic evaluation of FFP by analyzing the autocorrelation and self-similarity of resource availability signals. In this paper, our focus is to show the effectiveness of FFP under computational grid environments. FFP is being implemented as an extension of OGSA-BES (Open Grid Service Architecture Basic Execution Service) [12] in the Genesis-II computational grid platform [26]. Before deployment in the active grid, we have conducted a number of simulations using traces from actual computers. Our experiment results show that FFP significantly outperforms statistical techniques which ignore periodic pattern of failures. This results allow us to provide more accurate Quality-of-Service (QoS) bounds with significantly less resource consumption.

The remainder of this paper is as follows. Section 2 presents our monitoring environment and summarizes FFP. In Section 3, we discuss the simulation environment of computational grids. In Section 4, we present our evaluation results that quantify the benefits of FFP via trace-driven simulation. Related work is presented in Section 5 and Section 6 concludes the paper.

2 FAILURE PREDICTION

In this section, we demonstrate that ordinary statistical methods perform poorly in the presence of periodic events. We also present the *Filtered Failure Prediction* technique, in which failures of a resource are categorized into periodic and non-periodic failures, rendering more effective predictions.

2.1 Resource Monitoring Environment

To understand the behavior of a large collection of computing resources, a monitoring daemon is installed in 729 classroom and laboratory PCs at the University of Virginia. The monitoring daemon at each machine gathers statistics every 5 minutes, including Up/Down heartbeat signals, CPU utilization, and memory usage. The snapshots are reported to a central monitoring server. The study comprises PCs operated by two administrative organizations: 668 PCs by the Department of Information Technology & Communication (ITC) and 61 PCs by the Computer Science Department. The details of the monitoring environment can be found in [15].

During the continuous observation for 3 months, many machines reported regular reboots for 'software rejuvenation' [16, 39]. The idea behind software rejuvenation is to restart software periodically to prevent accumulating errors. This software rejuvenation through periodic reboots is a typical technique that is performed by

many institutions managing a large collection of computing resources. In Figure 1, the number of available machines during the 3 months is shown. We can see the regularity of the resource availability. This regular pattern of the resource availability provides the motivation of this work.

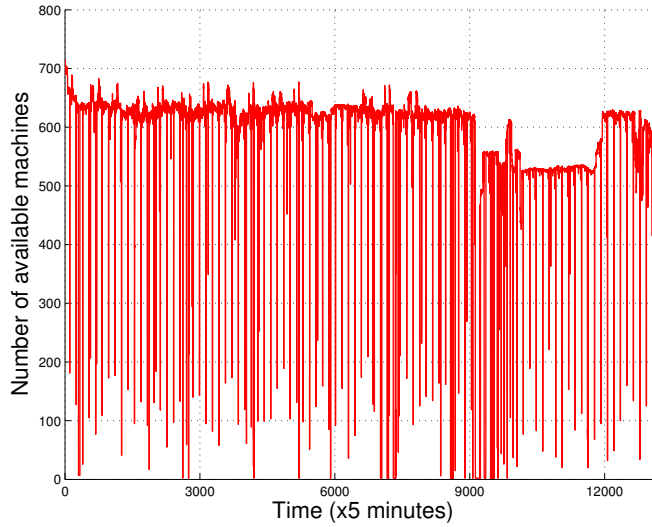

Figure 1. The number of available PCs for 3 months [15]

Throughout the remainder of the paper, we use the traces from this monitoring study for both analysis and evaluation since we believe that the University's computing environment is representative and typical for many other institutions.

2.2 Statistical Modeling of Grid Resources

To predict the reliability of computing resources, one typical approach is to use statistical methods such as fitting the collected data to statistical models. For instance, *exponential distribution* is commonly used to model the pattern of resource failures. *Weibull distribution* might be considered for more exact estimation [37]. For a resource, which does not manifest periodic behaviors, such statistical models work reasonably well as far as we choose a right model. However, in the presence of periodic patterns, the behavior of resources cannot be easily modeled by such well-known statistical models.

Figure 2 illustrates an exponential distribution and a Weibull distribution that were fitted using the *maximum likelihood technique*. In the figure, the empirical distribution shows a sudden surge at 1 440 minutes. This discontinuity at 1 440 minutes occurs due to the daily 'rejuvenation' of the machine. With the presence of such periodic failure patterns, the empirical distribution obtained from a machine cannot

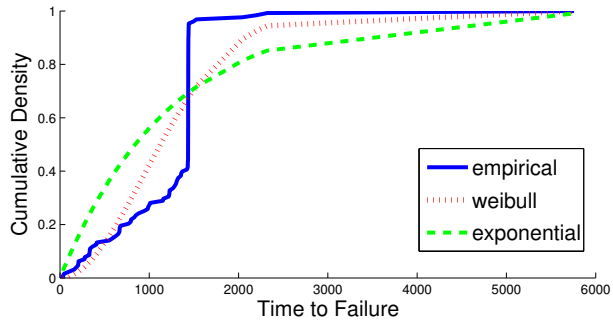

Figure 2. The distribution of a machine's time-to-failure modeled using the Weibull and the exponential distribution

be easily approximated by any statistical distribution. As a result, if a predictor is not aware of these kinds of periodic behavior, the prediction results might be inaccurate, resulting in poor resource allocation decisions for the grids. For instance, the empirical distribution in Figure 2 shows that the resource has almost zero probability of surviving longer than 24 hours. However, the prediction results from both the Weibull and the exponential distribution are totally misleading; the distributions show that the resource has about 30% probability of surviving after 24 hours.

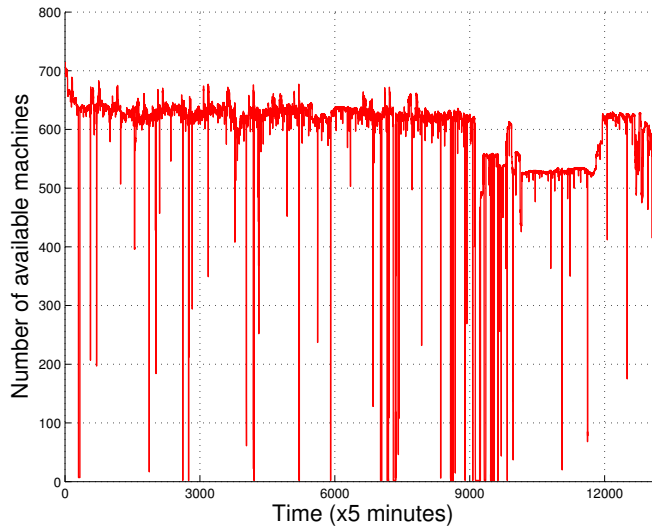

Figure 3. Number of available machines for 3 months after filtering out periodic events

Figure 3 shows the number of available machines after these periodic failures are filtered out using the filtering algorithm proposed in our previous work [17]. Fig-

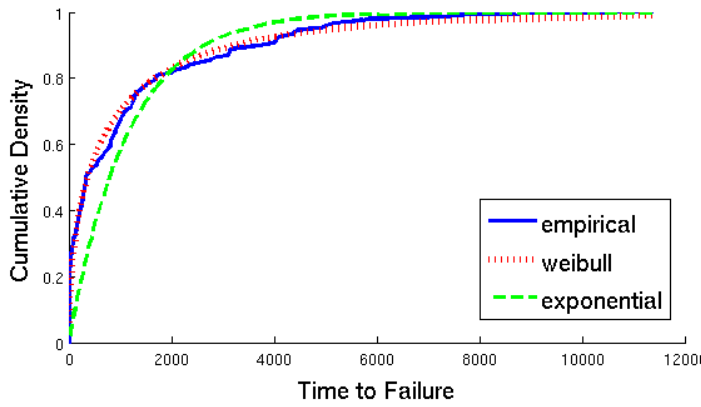

Figure 4. Time-to-failure distribution of a machine after filtering out periodic failures and its fitting to Weibull and exponential distributions

ure 4 illustrates the corresponding empirical distribution and the fitted Weibull and exponential distributions. In the empirical distribution, the previous discontinuity, observed in Figure 2, has disappeared. Figure 4 also shows that the empirical data can be more closely modeled using typical statistical distributions after the removing periodic events. In Table 1, the accuracy of the statistical models is quantified using *root mean square errors* (RMSE) between the empirical data and the fitted statistical models. The RMSE analysis shows that the accuracy of statical models improves significantly after filtering out periodic events. For example, the RMSE of Weibull model decreases from 0.161 to 0.032 when periodic events are filtered out. This result implies that we can achieve more accurate failure predictions once periodic failures are filtered out.

	Exp	Weibull
Without filtering of periodic events	0.232	0.161
After filtering of periodic events	0.141	0.032

Table 1. Root mean square errors (RMSE)

2.3 Filtered Failure Prediction Model

To address the problems of traditional statistic modeling techniques in failure predictions, we proposed a failure prediction model, called the *filtered failure prediction model (FFP)* [17]. In FFP, a sequence of up/down signal from the monitored resource is fed into the filter that detects and separates periodic failure events from the original signal. This detection step can be performed using an algorithm that scales linearly with respect to the number of events [24]. The periodicity detec-

tion algorithm should be able to handle impreciseness of time information, which may be resulted from lack of clock synchronization, rounding, and network delays. Hence, the algorithm has several tunable parameters to control the accuracy and reliability of the algorithm. For example, the time tolerance of period length is a tunable parameter. The algorithm detects periodic events if the number of their occurrences is more than the threshold value. The threshold value is adjusted according to the chi-squared test with a given confidence level, which is also tunable. Readers are referred to [17, 24] for more detailed discussion of periodicity detection algorithms.

After the filtering of periodic failures out, a failure predictor $P_{filtered}(\Delta t)$ is obtained from the updated up/down signal. For the failure predictor $P_{filtered}(\Delta t)$, we can consider any well-known probabilistic models such as Weibull and exponential distributions. If the sampled resource up/down signal has periodic component, it is detected and used to predict future periodic failures. The reliability acquired from $P_{filtered}(\Delta t)$ is only valid if Δt does not span the future periodic failure points. Since our objective is to predict the reliability, Δt includes both a start time and duration of a job. $P(\Delta t)$ is the actual predicted reliability over duration Δt and it can be summarized as Equation (1).

$$P(\Delta t) = \alpha \times P_{filtered}(\Delta t),$$
$$\text{where } \alpha = \begin{cases} 1 & \text{if } \Delta t \text{ does not spans periodic} \\ & \text{failure points,} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$
and P_{filter} is the reliability model obtained after filtering out periodic failures.

As an example, let us consider a resource that reboots every 24 hours and its last periodic failure occurred at time t. Assume that the resource's P_{filter} is 0.9 for 10 hours. At time $t + 5$ and $t + 15$, the resource is requested to run a job that requires 10 hours CPU time with more than 0.8 reliability. In this scenario, the first request can be accepted because the resource satisfies the reliability requirement (e.g., $0.9 > 0.8$). However, under FFP, the second request cannot be accepted because the job's execution spans the next periodic failure point, which is $t + 24$. Hence, the actual predicted reliability for the second request becomes $0\ (= 0 * 0.9)$, not 0.9. Further, we might consider accepting the job with the expected reliability of 0.9 and make a checkpointing of the job before the regular failure [25].

3 COMPUTATIONAL GRID ENVIRONMENT

In this section, we first describe the Genesis II computational grid [26], which is the basis of this work. We also discuss the simulation environment of the Genesis II computational grid, where jobs are replicated to achieve the specified QoS levels.

3.1 Genesis II Computational Grid

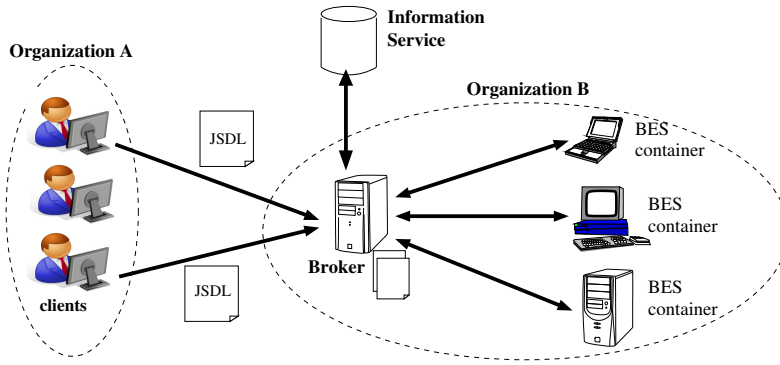

Figure 5. Genesis II computational grid environment

The Genesis II computational grid is an implementation of OGSA (Open Grid Service Architecture) from OGF (Open Grid Environment). The implementation includes several standards such as WS-Addressing and JSDL (Job Submission Description Language) [2]. Figure 5 shows an example of the Genesis II computational grid, which has two virtual organizations. Organization A and B, respectively, is a pure resource consumer and a provider. If a client in the organization A wants to use the resources in the organization B, he/she submits a job request to the *broker* using a JSDL (Job Submission Description Language) document. JSDL is an XML-formatted document that describes the properties of the submitted job. In the Genesis II platform, JSDL standard is extended to include QoS requirements, such as reliability of the resources. A client might use a portal with web interfaces to build a job description [11, 38]. Figure 6 shows an example of JSDL extended to include QoS specification for the target resource. It specifies that the target resource is supposed to have 95% reliability for a job running over 150 minutes on a canonical machine[1]. The *ExecutionTime* in JSDL specifies the estimated execution time of a job, which is normalized to a canonical machine. Due to the heterogeneity of grid resources, it is extremely difficult, if not impossible, to accurately estimate the execution time of a job, in general. Hence, our grid platform targets a set of high-throughput applications that renders such estimation. For example, parameter sweep applications, such as GridBlast [20], tend to be highly parallel, require minimal inter-node communications, and entail multiple executions of the same executable against different data. For such applications, the completion time of a job can be estimated.

After the broker receives a request, it selects one or a group of resource containers, and manages the execution of the job. If available resources are not enough

[1] Current JSDL v1.0 [2] does not support QoS specification yet.

```
<jsdl:JobDefinition id="myComputingJob">
<jsdl:JobDescription>
    <jsdl:Application>
        <jsdl:Name>GridBLAST</jsdl:Name>
    </jsdl:Application>
    <jsdl:DataStaging>
        <jsdl:FileName>input_file_115.dat</jsdl:FileName>
    </jsdl:DataStaging>
    <jsdl:Resources>
        ...
        <jsdl:QoS>
            <jsdl:ExecutionTime>150M</jsdl:ExecutionTime>
            <jsdl:Deadline>240M</jsdl:Deadline>
            <jsdl:Reliability>95</jsdl:Reliability>
        </jsdl:QoS>
    </jsdl:Resources>
...
</jsdl:JobDefinition>
```

Figure 6. An example of JSDL for submitting a job

to execute the job, the broker enqueues the request until more resources become available. Once a job instance is created in a BES (Basic Execution Service) container [12], the container will also help with staging in input data, performing the execution of the job, and staging out the results. Note that similar architectures have been proposed in cloud systems to facilitate resource and service management [6, 29].

The *information service* monitors the availability of the resources periodically and provides information required for the broker to make a proper decision on resource allocations. Information services have been investigated as one of core components in grids [9]. Information services provide services such as service discovery, resource monitoring, and resource characterization. To prevent potential overloads at the information service, the monitoring interval of each resource should be properly determined[2]. In this study, the information service simply provides a list of live service providers that are willing to accept a new job. However, they may support query interfaces to find resources satisfying clients' complex constraints. Support of such complex queries might result in the complexity and overheads around the information services. To address this problem, distributed resource allocation strategies have been adopted in a few distributed system, e.g. [7], which requires no central information service. Although a distributed system might have better scalability, it might become inefficient due to the limited information at the local systems. We leave further analysis of the two approaches as future work.

[2] In this work, each resource container reports its status every 5 minutes.

3.2 Job Replication Strategies

Since Genesis II targets less reliable desktop machines as its computational resources, the reliability requirement of 0.9 is too high to meet, especially when the runtime length of a job is longer than 10 hours. In case of having insufficient resources to meet the reliability requirement alone, we apply two replicated execution polices to meet the client's QoS requirement, replication in time and in space; we call them *replication-in-time* and *replication-in-space*, respectively. These replication strategies were first introduced in our previous work [18].

In the *replication-in-space* strategy, a job is replicated onto a set of selected resources. The number of selected resources for the replication is determined so that the *aggregate reliability* of selected resources is no bigger than the requirement. *Aggregate reliability* is the probability that *at least one of the selected resources survive without a failure*. For example, if a job is replicated onto two resources with the reliability of 0.7 and 0.8, respectively, the aggregate reliability is 0.94 $(= 1 - (1 - 0.7) \times (1 - 0.8))$. This indicates that one of the replica might complete successfully with 0.94 probability.

Algorithm 1: Replication-in-space with FFP

Input: Job Specification J with $(j, r, \delta t)$, where j is job description, r is required reliability, δt is expected execution time

1 Get a list of candidate resources S from the information service;
2 Each resource s in S has a tuple (r_s, p_s), where r_s is reliability, t_s is next periodic failure time;
3 $S_f = \{\}$;
4 **while** $\left(1 - \prod^{S_f}(1 - r_s) < r\right)$ **do**
5 s $\leftarrow$ pick s from S;
6 **if** *J.δt dos not span s.*t_s **then**
7 $S_f = S_f \cup (s)$;
8 **end**
9 remove s from S;
10 **end**
11 **if** $(1 - \prod(1 - r_s)) < r_s$ **then**
12 dispatch replicated jobs to service sites in S_f;
13 **else**
14 wait until more resources are available;
15 **end**

In Algorithm 1, the resource allocation algorithm with replication-in-space strategy and FFP is shown. The first step is to get a list of candidate resources from the information service (Line 1-2). A set of resources is selected from the candidates to achieve the target reliability using the replication-in-space strategy. The aggregate reliability, which is defined as $1 - \prod^{S_f}(1 - r_s)$, is updated as a new resource is selected from the candidates (Line 4). In Line 6, a resource is checked if its next periodic

failure point overlaps with the job's execution range. Once the aggregate reliability becomes greater than the required reliability, the job is dispatched to all selected resources concurrently (Line 11-12). If all replicated executions fail, the job failure is notified to the client and marked as failure.

In the *replication-in-time* strategy, resources are selected in a similar manner. However, unlike *replication-in-space* strategy, a job is dispatched onto only one resource at a time. If the resource fails before the completion of the job, then another resource is chosen for retries.

3.3 Simulation Environment and Settings

In a grid environment, it is hard to evaluate an algorithm by performing actual experiments in a repeatable and controllable manner since resources are distributed across multiple organizations with their own policy [5]. To overcome this problem and show the effectiveness of FFP, we use the Java-based discrete-event grid simulation toolkit, which was introduced in our previous work [18]. The simulator uses the same trace introduced in Section 2 to simulate each resource's behavior. Readers are referred to [18] for more detailed discussion on the simulator.

Parameter	Value
Model training period	1 month
Test running period	1 month
Total number of machines	729
Avg. length of short jobs (T_{SJ})	3 ± 0.5 hours
Avg. dispatch rate of short jobs	1 job/10 ± 2 minutes
Avg. length of long jobs (T_{LJ})	$6, 9, 12, 15, 18, 21 \pm 0.5$ hours
Avg. dispatch rate of long jobs	1 job/20 ± 5 minutes

Table 2. Parameters and settings of the simulation

Table 2 shows details of the simulation. The arrival of jobs from clients to the broker is synthesized to follow bimodal distribution. The bimodal distribution is composed of two normal distributions; one mode is for short runtime jobs with the average of 3 hours, and the other is for long runtime jobs with varying averages. Since we are more interested in the behavior of long runtime jobs, we change the mean length of longer runtime jobs, T_{LJ}, from 6 hours to 21 hours while the mean length of short jobs fixed to 3 hours. We expect that as the execution time of jobs get closer to the periodic failure interval, 24 hours in our case, the job will more likely to fail before its completion.

4 EVALUATION

In this section, we demonstrate the effectiveness of the proposed FFP in the simulation environment.

4.1 Baselines

We evaluated 3 algorithms, shown in Table 3, to verify the effectiveness of FFP for job scheduling.

Name	Job assignment strategy
Random	A job is assigned to any randomly chosen live machines without considering reliability.
FFP	Periodic failures are detected and the reliability prediction is made from a filtered series of events.
NFFP	Reliability is predicted without filtering.

Table 3. Resource scheduling approaches

For both $NFFP$ and FFP, a job is submitted to a resource only if the predicted reliability is no less than the required reliability from clients. In case of FFP, a candidate machine is first checked if it is going to have periodic failure during the job's expected run, and if not affected by a periodic failure, the reliability requirement is compared to the predicted reliability. In contrast, NFFP does not consider periodic resource failures when resources are selected. Further, NFFP's statistical models are built without filtering out periodic failures. For replication-in-time strategy, *Random* scheduling is used as a base case, in which any available live and idle machine is assigned for job execution without comparison of the required and predicted reliability. For all scheduling approaches, we assume that the time delay for job scheduling, including service delay to access information service, is negligible compared to the long execution time of jobs.

4.2 Results

For both *replication-in-time* and *replication-in-space*, we evaluate the resource usages and job success rates. The job success rate indicates how faithfully a client's reliability requirement is satisfied. The reliability requirements from clients are set to 0.90 in our simulation. Therefore job success rate should be close to 0.90. Achieving that required reliability with less resource usage is desired.

4.2.1 Varying Job Execution Time in Replication-in-Time

In this experiment, we observe job success rates and the resource usage in the replication-in-time mode, while the mean execution time of long jobs, T_{LJ}, is varied from 6 to 21 hours. Figures 7 and 8 show the average number of retries and its achieved job success rates, respectively. Both NFFP and FFP achieve the requested reliability from clients quite closely. However, the number of retries to achieve that success rate is quite different. In NFFP, the number of retries is almost same as Random, which is the worst case. For example, NFFP needs 3.3 retries to achieve 0.68 job success rate while FFP needs 1.3 retries to achieve 0.71 job success rate.

This shows that the average reliability of resources is much lower, as Figure 7 shows, when the periodic failures are not filtered out. As a result, the higher amount of resources consumption is required to meet the reliability requirement. However, that higher resource consumption does not make significant improvement in job success rate, as shown in Figure 8. For example, FFP achieves 0.88 job success rate using less than half computing resources than NFFP when T_{LJ} is 21 hours.

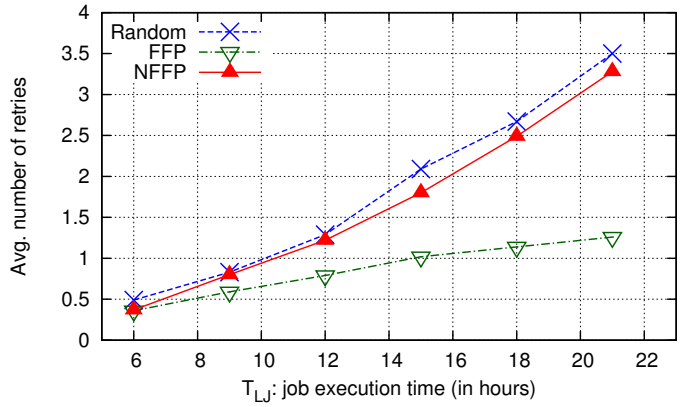

Figure 7. Average number of retries

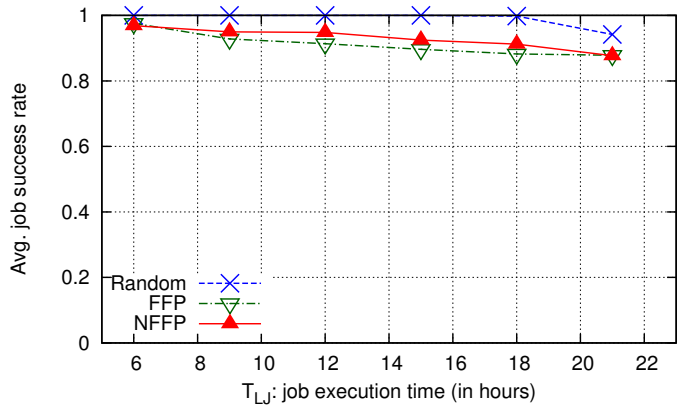

Figure 8. Average job success rate

This high performance gap between NFFP and FFP demonstrates that resources are more effectively used by considering periodic failures. This is more evident when the average turnaround time and the changes of the waiting queue at the broker are

observed. Figure 9 shows the average turnaround time. In the figure, *Ideal* shows the turnaround time when no resource failure is occured during the job execution. Interestingly enough, the gap between NFFP and FFP is not as big as the number of retries. For FFP, instead of doing retries and consuming resources which is going to fail due to periodic failure, the jobs are kept in the waiting queue when the run of job execution are predicted to span the next periodic failure time. The depth of the queue, as shown in Figure 10, increases until the periodic failure points, which occurs every 24 hours, and sharply decreases after them. The interval of this pattern conforms to the interval of periodic failures. In contrast, the depth of NFFP's queue, as shown in Figure 11, does not manifest this kind of pattern, indicating that it does not exploit the periodic behavior of resources for intelligent job scheduling.

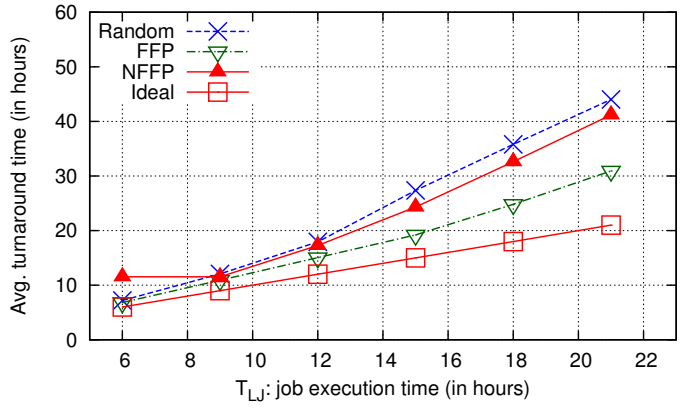

Figure 9. Average turnaround time

4.2.2 Varying Job Execution Time in Replication-in-Space

In this experiment, we perform the same experiment when the replication-in-space strategy is deployed. Figures 12 and 13 show the average degree of spatial redundancy of jobs and its achieved job success rate, respectively, while the job execution time of long jobs, T_{LJ}, is varied from 6 to 21 hours. As shown in Figure 12, the number of concurrent job executions required for NFFP increases sharply as T_{LJ} is getting closer to the periodic failure interval, 24 hours. For example, for 21 hours jobs, NFFP requires over 2 times of resources to achieve the similar level of reliability guarantees of FFP. Surprisingly, in contrast to the replicated execution in time, replication-in-space showed very low job success rate, around 0.7. This seems odd at first. After analyzing logs, we found that the failures of resources are highly correlated. Because our resource availability trace has been obtained from resources of a single institution, the availabilities of resources are often affected at the same

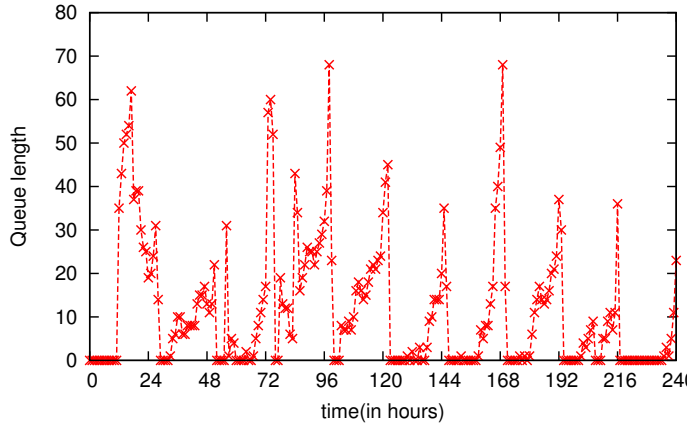

Figure 10. Changes of the queue in FFP for 10 days

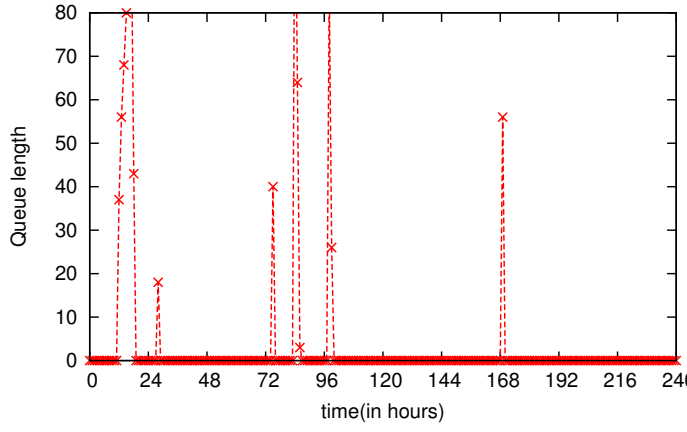

Figure 11. Changes of the queue in NFFP for 10 days

time by power failures, network partitioning, non-regular administrative maintenance, and etc. In the presence of these frequent correlated failures, the redundant execution of job is less effective. We leave the investigation on this correlated failure as our future work.

5 RELATED WORK

There has been much research on analyzing events and building a statical model for machine/resource behavior in order to make a future event prediction. Some researchers have been focusing on analyzing error and failure event logs [21, 22, 32, 31].

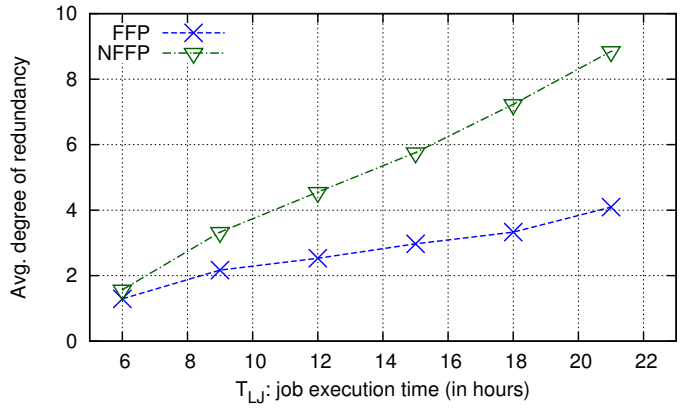

Figure 12. Average redundancy

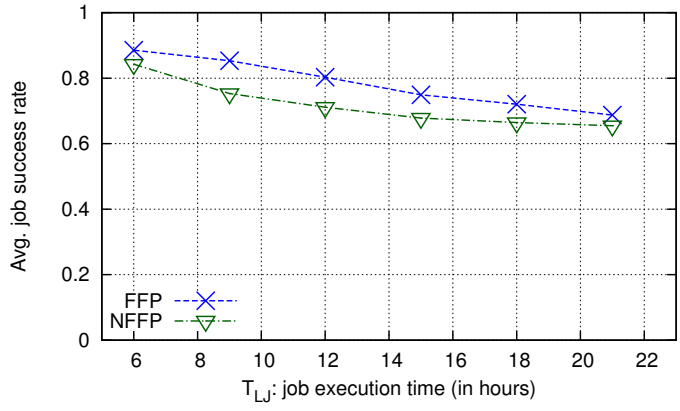

Figure 13. Average job success rate

Work of Lin et al. [22] is similar to our work in terms of that they see the failures and errors are due to multiple sources, not a single source. Lin et al. separate and categorize errors into intermittent and transient errors and provide a set of rules for fault/failure prediction. They show that the time between errors from each separate source follows a Weibull distribution and the combined errors cannot be modeled by any single well-known distributions. However, their approach uses information obtained in controlled environment. In grids which typically span multiple autonomous administrative domains, we have very limited control on resources and the available information about resource error and failure. Therefore, their approach cannot be used for our research as it is. Our approach does not rely on detailed information

from a resource. We assume the only information we can obtain is heartbeat signals to check the liveness of a resource.

Saadatfar et al. [31] used machine learning techniques to mine Grid Workload Archive (GWA) data in order to derive patterns and rules that affects failures of jobs. This study demonstrated that several parameters such as CPU-intensity, queue load, and execution hour of day, etc. affect failure rates closely. The proposed failure predictor assumes that a predictor has such detailed real-time information, which is rarely available in shared grid environments.

The preprocessing of the original data set of events to simplify analysis has been done before, e.g., [37] and [42]. However, the preprocessing have been mostly preformed to remove redundant information such as successive repeated errors. In our study, the original series of events are preprocessed to separate periodic events from non-periodic events.

The emergence of grids as new distributed computing platform has fostered many studies on resource availability/reliability prediction in grids such as [4, 28, 42, 3] and [30, 31, 33, 34]. Brevik et al. [4] compare parametric and non-parametric approaches for machine availability prediction. Their result shows that non-parametric approach is better in most situations in estimating the lower bound of a given quantile, especially when the sample size is small. Non-parametric approach is very good when the distribution is not easy to fit by parametric methods as our case. However, the problem of non-parametric approaches is that they are appropriate for testing assertions rather than estimation of effects. From a practical viewpoint, a job scheduler needs to quantify the reliability of the resources at some specific time to test and compare the fitness of the resources. Our work shows that parametric approaches might have problem in the presence of periodic failures but the distribution can be fitted using parametric methods if periodic events are filtered out. Despite the large body of work on error/failure prediction, the absence of any research on periodic resource unavailability as a separate source of failure and modeling of them to make a prediction in job scheduling is the motivation of our study in this paper.

Recently, cloud computing [41], such as Amazon EC2, has become a popular on-demand computing platform. Reliability/availability issues in clouds have been studied in [36, 40, 23, 19], where various models were proposed to characterize the failures in clouds. For clouds, unlike grids, reactive approaches such as checkpointing and restarts have been more actively investigated since the migration of tasks/platforms can be efficiently supported in the the virtualized computing environment of clouds [43].

6 CONCLUSIONS

Accurate prediction of resource reliability is critical to provide completion time, availability, and other quality of service (QoS) guarantees. We analyzed the real traces from the large scale monitoring experiments, and demonstrated that accurate prediction using traditional statistical methods is difficult in the presence of periodic

events. We have developed the failure prediction scheme, called FFP, that separates non-time-homogeneous regular failures from time-homogeneous non-regular failure events. To show the effectiveness of the proposed scheme, we have developed a computational grid simulator, in which the entities and their availability are modeled based on a large scale monitoring experiment on real computing resources. Through the extensive simulation, we demonstrated that FFP outperforms other techniques that ignore periodicity for failure prediction. The improvements are particularly pronounced as the time interval approaches the periodic failure interval. Furthermore, our results show that ignoring periodicity in the presence of actual periodic failures might lead to ineffective resource allocation, as bad as doing no prediction.

Acknowledgements

This work was supported by the Incheon National University Research Grant in 2014.

REFERENCES

[1] Genesis II project homepage. `http://genesis2.virginia.edu/wiki/`, 2014.

[2] Anjomshoaa, A.—Brisard, F.—Dresher, M.—Fellows, D.—Ly, A.—McGough, S.—Pulsipher, D.—Savva, A.: Job Submission Description Language (JSDL) Specification. Version 1.0. 2005.

[3] Khoo, B. T. B.—Veeravalli, B.: Pro-Active Failure Handling Mechanisms for Scheduling in Grid Computing Environments. Journal of Parallel and Distributed Computing, Vol. 70, 2010, No. 3, pp. 189–200.

[4] Brevik, J.—Nurmi, D.—Wolski, R.: Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems. In IEEE CCGrid, 2004, pp. 190–199.

[5] Buyya, R.—Murshed, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience, Vol. 14, 2002, No. 13, pp. 1175–1220.

[6] Buyya, R.—Yeo, C. S.—Venugopal, S.—Broberg, J.—Brandic, I.: Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5^{th} Utility. Future Generation Computer Systems, Vol. 25, 2009, No. 6, pp. 599–616.

[7] Chard, K.—Bubendorfer, K.: High Performance Resource Allocation Strategies for Computational Economies. IEEE Transactions on Parallel and Distributed Systems, Vol. 24, 2013, No. 1, pp. 72–84.

[8] Chien, A.—Calder, B.—Elbert, S.—Bhatia, K.: Entropia: Architecture and Performance of an Enterprise Desktop Grid System. Journal of Parallel and Distributed Computing, Vol. 63, 2003, No. 5, pp. 597–610.

[9] Czajkowski, K.—Fitzgerald, S.—Foster, I.—Kesselman, C.: Grid Information Services for Distributed Resource Sharing. Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC '01), 2001, pp. 181–194.

[10] David, M.—Borges, G.—Gomes, J.—Pina, J.—Campos, I.—Fernandez, E.—Lopez, Á.—Oriviz, P.—Cacheiro, J.—Fernandez, C.—Simon, A.: Software Provision Process for EGI. Computing and Informatics, Vol. 31, 2012, No. 1, pp. 135–148.

[11] Dziubecki, P.—Grabowski, P.—Krysinski, M.—Kuczynski, T.—Kurowski, K.—Piontek T.—Szejnfeld D.: New Science Gateways for Advanced Computing Simulations and Visualization Using Vine Toolkit in PL-Grid. Computing and Informatics, Vol. 32, 2013, No. 5, pp. 1100–1115.

[12] Foster, I.—Grimshaw, A.—Lane, P.—Lee, W.—Morgan, M.—Newhouse, S.—Pickles, S.—Pulsipher, D.—Smith, C.—Theimer, M.: OGSA Basic Execution Service, Version 1.0. Technical report, Open Grid Forum, 2007.

[13] Foster, I.—Kesselman, C. (Eds.): The Grid2, Second Edition: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2004.

[14] Geddes, N.: The Large Hadron Collider and Grid Computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 370, 2012, No. 1961, pp. 965–977.

[15] Huang, H. H.—Karpovich, J. F.—Grimshaw, A. S.: Analyzing the Feasibility of Building a New Mass Storage System on Distributed Resources. Concurrency and Computation: Practice and Experience, Vol. 20, 2008, No. 10, pp. 1131–1150.

[16] Huang, Y.—Kintala, C.—Kolettis, N.—Fulton, N. D.: Software Rejuvenation: Analysis, Module and Applications. Fault-Tolerant Computing, International Symposium, 1995.

[17] Kang, W.—Grimshaw, A.: Failure Prediction in Computational Grids. 40th Annual Simulation Symposium, March 2007, pp. 275–282.

[18] Kang, W.—Huang, H. H.—Grimshaw, A. S.: Achieving High Job Execution Reliability Using Underutilized Resources in a Computational Economy. Future Generation Computer Systems, Vol. 29, 2013, No. 3, pp. 763–775.

[19] Khazaei, H.—Misic, J.—Misic, V. B.—Mohammadi, N. B.: Availability Analysis of Cloud Computing Centers. IEEE Global Communications Conference (GLOBECOM), 2012, pp. 1957–1962.

[20] Krishan, A.: GridBLAST: A Globus-Based High-Throughput Implementation of BLAST in a Grid Computing Framework. Concurrency and Computation: Practice and Experience, Vol. 17, 2005, No. 13, pp. 1607–1623.

[21] Lee, I.—Iyer, R. K.—Tang, D.: Error/Failure Analysis Using Event Logs from Fault Tolerant Systems. Proceedings of 21st International Symposium on Fault-Tolerant Computing, 1991.

[22] Lin, T.-T. Y.—Siewiorek, D. P.: Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis. IEEE Transactions on Reliability, Vol. 39, 1990, No. 4, pp. 419–432.

[23] Longo, F.—Ghosh, R.—Naik, V. K.—Trivedi, K. S.: A Scalable Availability Model for Infrastructure-as-a-Service Cloud. 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), 2011, pp. 335–346.

[24] Ma, S.—Hellerstein, J. L.: Mining Partially Periodic Event Patterns with Unknown Periods. Proceedings of the 17th International Conference on Data Engineering, 2001, pp. 205–214.

[25] Medeiros, B.—Sobral, J.: AspectGrid: Aspect-Oriented Fault-Tolerance in Grid Platforms. Computing and Informatics, Vol. 31, 2012, No. 1, pp. 89–101.

[26] Morgan, M. M.—Grimshaw, A. S.: Genesis II – Standards Based Grid Computing. Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), 2007, pp. 611–618.

[27] Muppala, J. K.—Ciardo, G.—Trivedi, K. S.: Stochastic Reward Nets for Reliability Prediction. Communications in Reliability, Maintainability and Serviceability, Vol. 1, 1994, No. 2, pp. 9–20.

[28] Ren, X.—Lee, S.—Eigenmann, R.—Bagchi, S.: Resource Failure Prediction in Fine-Grained Cycle Sharing System. 2006 15th IEEE International Conference on High Performance Distributed Computing (HPDC), 2006, pp. 93–104.

[29] Rochwerger, B.—Breitgand, D.—Levy, E.—Galis, A.—Nagin, K.—Llorente, I. M.—Montero, R.—Wolfsthal, Y.—Elmroth, E.—Cáceres, J. et al.: The Reservoir Model and Architecture for Open Federated Cloud Computing. IBM Journal of Research and Development, Vol. 53, 2009, No. 4, pp. 535–545.

[30] Rood, B.—Lewis, M. J.: Grid Resource Availability Prediction-Based Scheduling and Task Replication. Journal of Grid Computing, Vol. 7, 2009, No. 4, pp. 479–500.

[31] Saadatfar, H.—Fadishei, H.—Deldari, H.: Predicting Job Failures in AuverGrid Based on Workload Log Analysis. New Generation Computing, Vol. 30, 2012, No. 1, pp. 73–94.

[32] Sahoo, R. K.—Oliner, A. J.—Rish, I.—Gupta, M.—Moreira, J. E.—Ma, S.—Vilalta, R.—Sivasubramaniam, A.: Critical Event Prediction for Proactive Management in Large-Scale Computer Clusters. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03), ACM Press, 2003, pp. 426–435.

[33] Salinas, S. A.—Garino, C. G.—Zunino, A.: An Architecture for Resource Behavior Prediction to Improve Scheduling Systems Performance on Enterprise Desktop Grids. Advances in New Technologies, Interactive Interfaces and Communicability, Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7547, 2012, pp. 186–196.

[34] Sonmez, O.—Yigitbasi, N.—Iosup, A.—Epema, D.: Trace-Based Evaluation of Job Runtime and Queue Wait Time Predictions in Grids. Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, ACM, 2009.

[35] Thain, D.—Tannenbaum, T.—Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17, 2005, No. 2-4, pp. 323–356.

[36] Thanakornworakij, T.—Nassar, R. F.—Leangsuksun, C.—Păun, M.: A Reliability Model for Cloud Computing for High Performance Computing Applications. Proceedings of the 18th International Conference on Parallel Processing Workshops (Euro-Par '12), Springer-Verlag, 2013, pp. 474–483.

[37] Trivedi, K. S.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications. 2nd ed., John Wiley and Sons, 2001.

[38] Tugores, A.—Colet, P.: Web Interface for Generic Grid Jobs, Web4Grid. Computing and Informatics, Vol. 31, 2012, No. 1, pp. 173–187.

[39] Vaidyanathan, K.—Harper, R. E.—Hunter, S. W.—Trivedi, K. S.: Analysis and Implementation of Software Rejuvenation in Cluster Systems. ACM SIGMETRICS Performance Evaluation Review, Vol. 29, 2001, No. 1, pp. 62–71.

[40] Vishwanath, K. V.—Nagappan, N.: Characterizing Cloud Computing Hardware Reliability. Proceedings of the 1st ACM Symposium on Cloud Computing, ACM, 2010, pp. 193–204.

[41] Voorsluys, W.—Broberg, J.—Buyya, R.: Introduction to Cloud Computing. In: Buyya, R., Broberg, J., Goscinski, A. (Eds.): Cloud Computing: Principles and Paradigms. John Wiley & Sons, 2011, pp. 1–41.

[42] Wolski, R.—Spring, N. T.—Hayes, J.: Predicting the CPU Availability of Time-Shared Unix Systems on the Computational Grid. Cluster Computing, Vol. 3, 2000, No. 4, pp. 293–301.

[43] Yi, S.—Kondo, D.—Andrzejak, A.: Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010, pp. 236–243.

Woochul Kang received his Ph.D. degree in computer science from the University of Virginia in 2009. He was a senior researcher at Electronics and Telecommunications Research Institute (South Korea, 2000–2004, 2009–2012), and a post-doc at the University of Illinois at Urbana-Champaign (USA, 2012–2013). He joined Incheon National University as Assistant Professor in 2013. His current research interests include cyber-physical systems, embedded databases, distributed middleware, feedback control of computing systems, and safe integration of medical devices.

Jibum Kim received his B.Sc. and M.Sc. degrees in electrical engineering from Yonsei University, Seoul, South Korea, in 2003 and 2005, respectively, and his Ph.D. degree in computer science and engineering from the Pennsylvania State University in 2012. He was a postdoctoral fellow at the Los Alamos National Laboratory (Group T-5) in 2013. He is currently Assistant Professor in the Department of Computer Science and Engineering at Incheon National University, South Korea. His research interests include embedded computing systems, computational science, and high performance computing.