Fault-Aware Runtime Strategies for High-Performance Computing

Yawei Li, *Student Member*, *IEEE*, Zhiling Lan, *Member*, *IEEE Computer Society*, Prashasta Gujrati, *Member*, *IEEE Computer Society*, and Xian-He Sun, *Senior Member*, *IEEE Computer Society*

Abstract—As the scale of parallel systems continues to grow, fault management of these systems is becoming a critical challenge. While existing research mainly focuses on developing or improving fault tolerance techniques, a number of key issues remain open. In this paper, we propose runtime strategies for spare node allocation and job rescheduling in response to failure prediction. These strategies, together with failure prediction and fault tolerance techniques, construct a runtime system called Fault-Aware Runtime System (FARS). In particular, we propose a 0-1 knapsack model and demonstrate its flexibility and effectiveness for reallocating running jobs to avoid failures. Experiments, by means of synthetic data and real traces from production systems, show that FARS has the potential to significantly improve system productivity (i.e., performance and reliability).

Index Terms—High-performance computing, runtime strategies, fault tolerance, performance, reliability, 0-1 knapsack.

1 Introduction

By 2011, it is anticipated that researchers will be able to access a rich mix of systems with some capable of delivering sustained performance in excess of one petaflop/s. Production systems with hundreds of thousands of processors are being designed and deployed. Such a scale, combined with the ever-growing system complexity, is introducing a key challenge on fault management for high-performance computing (HPC). Despite great efforts on designing ultrareliable components, the increase in system size and complexity has outpaced the improvement of component reliability. Recent studies have pointed out that the mean-time-between-failure (MTBF) of teraflops and soon-to-be-deployed petaflop machines are only on the order of 10-100 hours [30]. This situation is only likely to deteriorate in the near future, thereby threatening the promising productivity of large-scale systems [26].

The conventional method for fault tolerance is checkpointing, which periodically saves a snapshot of the system to a stable storage and uses it for recovery in case of failure. Yet, it does not prevent failure, and work loss is inevitable due to the rollback process. An increasing interest in HPC is to explore proactive techniques like process migration to avoid failures by leveraging the research on failure prediction. For example, object migration is proposed for AMPI-based applications to avoid hardware failures [5]. The experiment with a Sweep3D application has shown that object migration may only take less than 2 seconds. In [21], live migration is explored on Xen virtual machines, and the experiments with scientific applications have shown that migration overhead is as low as 30 seconds. In our own previous study [17], we have demonstrated that timely process migrations can greatly improve application performance—application execution times—by up to 43 percent.

While process migration itself has been studied extensively, a number of issues remain open in the design of fault-aware runtime systems. Key issues include how to allocate resources to accommodate proactive actions and how to coordinate multiple jobs for an efficient use of the resources in case of resource contention. Further, there is a lack of systematic study of runtime fault management by taking into account various factors including system workload, failure characteristics, and prediction accuracy. As an example, a commonly asked question is, "Given that prediction misses and false alarms are common in practice, how much gain can a fault-aware runtime system provide?"

This study aims at filling the gap between failure prediction and fault tolerance techniques by designing runtime strategies for *spare node allocation* and *job rescheduling* (i.e., reallocate running jobs to avoid failures). These strategies, together with failure prediction and fault tolerance techniques, construct a runtime system called FARS (Fault-Aware Runtime System) for HPC.

The first runtime strategy is for spare node allocation. To enable running jobs to avoid anticipated failures, spare nodes are needed. As jobs in the queues also compete for computing resources, a desirable runtime system should make a balanced allocation of resources between failure prevention and regular job scheduling. While static allocation by reserving a fixed number of nodes in prior is simple, it does not adapt to the runtime dynamics inherent in production environments. We propose a nonintrusive allocation strategy that dynamically allocates spare nodes for failure prevention.

The second runtime strategy is for job rescheduling. Given the existence of failure correlations in large-scale systems, simultaneous failures on multiple nodes are possible. Selection of jobs for rescheduling becomes crucial

• *The authors are with the Department of Computer Science, Illinois Institute of Technology, 10 W. 31st Street, Chicago, IL 60616. E-mail: {liyawei, lan, gujrpra, sun}@iit.edu.*

Manuscript received 30 Oct. 2007; revised 12 May 2008; accepted 1 July 2008; published online 8 July 2008.
Recommended for acceptance by C.-Z. Xu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-10-0399.
Digital Object Identifier no. 10.1109/TPDS.2008.128.

when spare nodes are not sufficient to accommodate migration requests originating from jobs. Previous research generally assumed the availability of sufficient spare nodes. But unfortunately, system resources are limited. Job rescheduling in case of resource contention can significantly impact the overall system productivity, given that jobs have quite different characteristics. In this study, we present a 0-1 knapsack model to address the job rescheduling problem, and then demonstrate its flexibility and effectiveness by presenting three job rescheduling strategies.

We evaluate FARS under a wide range of system settings by using both synthetic data and real traces from production systems. Considering that the often-used performance metrics like system utilization rate are mainly designed to measure system performance without giving much attention to failure, we choose a set of six metrics (including both performance and reliability metrics) and a 6D Kiviat graph for a comprehensive assessment of FARS. We also examine the sensitivity of FARS to system load, node MTBF, and prediction accuracy.

Preliminary results show that FARS can substantially reduce job response time, job slowdown caused by failure, job failure rate, and work loss, as well as slightly improve system utilization and job throughput. For a moderately loaded system, FARS is capable of improving its productivity (i.e., performance and reliability) by over 30 percent as against the case without it. Our experiments demonstrate the effectiveness of FARS as long as the failure predictor can capture 25 percent of failure events with a false alarm rate lower than 75 percent.

FARS complements existing research on checkpointing and job scheduling by reallocating running jobs to avoid failures in response to failure prediction. It can be easily integrated with job schedulers and checkpointing tools to jointly address the fault management challenge in HPC. The proposed 0-1 knapsack model provides a flexible method to address the job rescheduling problem, from which new rescheduling strategies can be easily derived.

The remainder of this paper is organized as follows: Section 2 gives an overview of FARS. A dynamic strategy for spare node allocation is described in Section 3. Section 4 formalizes the rescheduling problem as a 0-1 knapsack model and presents three rescheduling strategies. Section 5 describes our evaluation methodology, followed by experimental results in Section 6. Section 7 briefly discusses related work. Finally, Section 8 summarizes this paper and presents future work.

2 Problem Description

Consider a system with N compute nodes. User jobs are submitted to the system through a batch scheduler. For example, first-come, first-serve (FCFS) scheduling is commonly used by batch schedulers in HPC [20]. A job may be a sequential application or a parallel application. A job request is generally described by a three-parameter tuple $\{a_i, t_i, n_i\}$, where a_i is job arrival time, t_i is job execution time, and n_i is job size in terms of number of compute nodes.

The job scheduler is responsible for allocating *inactive jobs* (i.e., jobs in the queues) to compute nodes. Once a job is allocated, it is termed as an *active job.* FARS is responsible

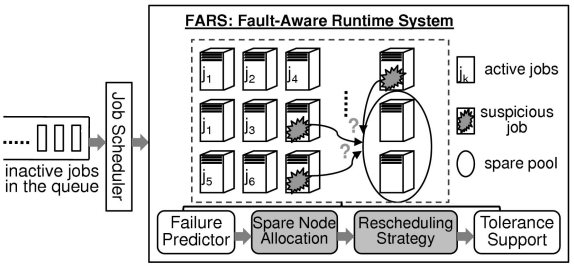


Fig. 1. Overview of FARS. User jobs are submitted through the job scheduler, while FARS is responsible for managing active jobs (i.e., running jobs) in the presence of failure. The dark shaded boxes indicate the major contributions of this study: spare node allocation (Section 3) and job rescheduling (Section 4).

for fault management of active jobs in response to failure prediction. Here, a *failure* is defined as an unexpected event in system hardware or software that stops a running application immediately. In case of parallel applications like tighly-coupled MPI applications [4], a single node failure usually aborts the entire application. Failed nodes are excluded from the pool of compute nodes until the problem is repaired. Active jobs are supposed to be checkpointed by application-initiated or system-initiated checkpointing tools [3], [11], [37]. Fig. 1 gives an overview of FARS.

FARS may be triggered in two ways: 1) predefined, where FARS is invoked at predefined points set by system administrators and 2) event-triggered, where FARS is alerted by the failure predictor when a worrisome event occurs. This paper assumes predefined mechanism for the convenience of study.

FARS periodically consults the failure predictor for the status of each compute node during the next interval. Regardless of prediction techniques, prediction result can be either *categorical* where the predictor forecasts whether a failure event will occur or not, or *numerical* where the predictor estimates failure probability. Numerical results can be easily converted to categorical results via threshold based splitting; hence in this paper, we uniformly describe *failure prediction* as a process that periodically estimates whether a node will fail during the next interval. Such a prediction mechanism is generally measured by two metrics: *precision* and *recall* as described in Table 1.

Upon each invocation, FARS identifies the set of nodes that are likely to fail in the next interval based on failure prediction. Suppose that N_s out of N nodes are predicted to be failure-prone (denoted as *suspicious nodes*), and $\{j_i^s | 1 \le i \le J_s\}$ is the set of active jobs residing on these suspicious nodes (denoted as *suspicious jobs*).

The objective of FARS is to dynamically reallocate suspicious jobs so as to minimize failure impact on system productivity. Toward this end, runtime strategies are developed for allocating spare nodes and reallocating suspicious jobs. Based on these runtime strategies, process migration support can be applied to transfer application processes away from failure-prone nodes to healthy spare nodes.

Before presenting our strategies, we present our nomenclature in Table 1.

TABLE 1
Nomenclature

Symbol	Description
$precision$, $recall$	Prediction accuracy, defined as $\frac{T_p}{T_p+F_p}$ and $\frac{T_p}{T_p+F_n}$ respectively where T_p is no. of true positives, F_p is no. of false positives, and F_n is no. of false negatives
N, N_s	Number of nodes (suspicious nodes) in the system
S	Number of spare nodes
J, J_s	Number of active jobs (suspicious jobs)
t_i	failure-free execution time of job i
n_i, n_i^s	Number of nodes (suspicious nodes) allocated to job i
f_i	Failure probability of job j_i
t_{last}^i	The most recent time where the job i can be safely started from
I_F	FARS interval
O_{ckp}	Job checkpoint overhead
O_r	Job restart cost
O_F	FARS overhead
O_q	Job re-queuing time

3 Spare Node Allocation

Spare nodes for failure prevention can be allocated either *statically* or *dynamically*. Static allocation, which reserves a fixed set of nodes, is commonly used in existing studies due to its simplicity [29]. Static allocation is simple. However, it is difficult, if not impossible, to determine an optimal reservation at a prior time. An excessive allocation can lead to low system productivity due to less resources for regular job scheduling, while a conservative allocation can undermine the effectiveness of FARS because of insufficient spare nodes for process migration. Further, system load tends to change dynamically during operation, and static allocation does not adapt to these changes.

We propose a dynamic allocation strategy, which is based on a key observation in HPC.

3.1 Observation

After examining a number of job logs that are shared on the public domain [13] or are collected from production systems, we have observed that *idle nodes are common in production systems, even in the systems under high load*.

In Table 2, we list the statistics of idle nodes from 10 production systems. This list contains a variety of systems with different scales, utilization rates, and architectures. Production systems typically consist of a collection of compute nodes and some service and I/O nodes. The data shown in the table only lists compute nodes.

While these systems may exhibit different patterns in terms of idle node distributions, they share a common characteristic, that is, idle nodes are often available. In fact, on all the systems we have examined, the probability that at least 2 percent of the system resources are idle at any instant of time is high (more than 70 percent), and in some systems, the probability is even as high as 90 percent. We believe that the table clearly delivers the message that idle nodes are common in production systems. Indeed, this observation is confirmed by system administrators and is also mentioned in [43].

3.2 Dynamic Allocation Strategy

Based on the above observation, we propose *a nonintrusive, dynamic allocation strategy for FARS*. Here, the "dynamic" means that spare nodes are determined at runtime, and the "nonintrusive" indicates that FARS does not violate any reservation made by the job scheduler. The detailed steps are as follows:

- Upon invocation, FARS first harvests the available idle nodes into a candidate pool.
- Next, it excludes failure-prone nodes from the candidate pool according to failure prediction. The rationale is to avoid the situation wherein an application process is transferred to a failure-prone node.
- Finally, FARS excludes a number of nodes from the candidate pool to ensure job reservations made by the batch scheduler for some queued jobs [20]. The resulting pool is denoted as *spare pool*, and will be used for runtime failure prevention.

Fig. 2 illustrates how our dynamic allocation strategy works with FCFS/EASY backfilling scheduling [20]. Under FCFS/EASY, jobs are served in FCFS order, and subsequent jobs continuously jump over the first queued job as long as they do not violate the reservation of the first queued job. FCFS/EASY backfilling is widely used by many batch schedulers, and it has been estimated that 90 percent to 95 percent of batch schedulers use this default configuration [39]. As shown in the figure, to guarantee the reservation of

TABLE 2
Statistics of Idle Nodes in Production Systems

System	Period	CPUs	Jobs	Util	Prob
SDSC SP2	24 mo	128	59,725	83.5%	73.0%
NASA iPSC	3 mo	128	18,239	46.7%	79.2%
OSC Cluster	22 mo	178	36,097	43.1%	79.2%
LLNL T3D	4 mo	256	21,323	61.6%	96.9%
SDSC Paragon	24 mo	400	86,105	71.0%	99.3%
CTC SP2	11 mo	430	77,222	66.2%	93.0%
TACC Lonestar	6 mo	1024	25,000	94.0%	95.8%
LANL CM5	24 mo	1024	122,060	75.2%	98.0%
SDSC Blue	32 mo	1152	243,314	76.2%	94.0%
LANL O2K	5 mo	2048	121,989	64.0%	99.7%

The last column gives the probability that at least 2 percent of system nodes are idle at any time instant.

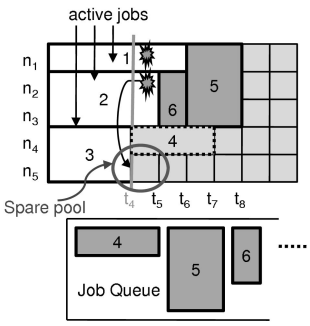


Fig. 2. Dynamic node allocation. An example by using FARS with FCFS/EASY is shown, in which spare nodes are not sufficient. At time t_4, there are two suspicious jobs (j_1 and j_2) and two idle nodes (n_4 and n_5). To guarantee the reservation of the first queued job j_4 at t_4 required by FCFS/EASY, FARS only puts n_5 into the spare pool.

the first queued job (job 4 at time t_4), FARS only puts one idle node into the spare pool. After this, FARS will apply the rescheduling method presented in Section 4 to select a job (which is j_2 in this case) for rescheduling, and j_1 will fail if the failure prediction is correct.

This dynamic allocation strategy is not limited to FCFS/EASY and can be applied to other scheduling policies. For example, in case of a greedy scheduling that does not guarantee any job reservation, FARS aggressively acquires all idle nodes and puts them into the spare pool; in case of a more conservative scheduling such as conservative backfilling [20], FARS excludes the nodes to guarantee the reservation of all the queued jobs and only puts the remaining idle nodes into the spare pool.

4 Job Rescheduling

After acquiring the spare pool, the next step is to provide a strategy for rescheduling suspicious jobs. In case that the number of spare nodes is sufficient, all the suspicious jobs will be reallocated. Otherwise, contention occurs among the suspicious jobs for the spare nodes. A desirable rescheduling strategy should weigh the benefit of reallocating different jobs, with the goal of minimizing failure impact on system productivity.

In this section, we first describe how to formalize the problem into a 0-1 knapsack model and then present three rescheduling strategies derived from this model.

4.1 0-1 Knapsack Model

Suppose there are S nodes in the spare pool, J_s suspicious jobs $\{j_i^s | 1 \le i \le J_s\}$, with each job j_i residing on n_i^s suspicious nodes. Hence, the rescheduling problem can be formalized as follows:

Problem 1. *To select a set of application processes from the suspicious jobs, with the objective of minimizing failure impact on system productivity.*

For a parallel job (e.g., a MPI job), failure of even a single process usually aborts the entire job. Thus, for a suspicious job, migrating some of its suspicious processes does not eliminate the possibility of failure. An effective rescheduling strategy should be more *job oriented*, meaning that all suspicious processes belonging to the same job should be migrated together if possible. As a result, Problem 1 can be transformed into the following problem:

Problem 2. *To select a subset of $\{j_i^s | 1 \le i \le J_s\}$ such that their rescheduling requires no more than S spare nodes, with the objective of minimizing failure impact on system productivity.*

For each suspicious job j_i, we associate it with a gain v_i and a weight w_i. Here, v_i represents productivity gain by rescheduling the job, which will be elaborated in the next section. And w_i denotes its rescheduling cost, which is the number of spare nodes needed for rescheduling the job. We can further transform Problem 2 into a standard 0-1 knapsack model:

Problem 3. *To determine a binary vector $X = \{x_i | 1 \le i \le J_s\}$ such that*

$$\begin{aligned} &\text{maximize} \sum_{1 \le i \le J_s} x_i \cdot v_i, \quad x_i = 0 \text{ or } 1 \\ &\text{subject to} \sum_{1 \le i \le J_s} x_i \cdot n_i^s \le S. \end{aligned} \tag{1}$$

The solution X determines the jobs for rescheduling. If x_i is 1, meaning that j_i is selected for reallocation by transferring some of its application processes so as to avoid job failure.

4.2 Three Rescheduling Strategies

Depending on the primary objective of fault management, a variety of rescheduling strategies can be derived from the aforementioned 0-1 knapsack model by properly setting v_i.

In practice, failure impact can be observed from different aspects. When a failure occurs, the affected job fails and rolls back to its initial state or the most recent checkpoint, thereby causing a loss of computing cycles. In the field of HPC, such a loss is generally measured by *service units* which are defined as the aggregated processing time. Users are typically concerned about failure probability of their jobs. This can be quantified by *job failure rate*, which is defined as the ratio between the number of failed jobs and the total number of jobs submitted. Furthermore, given that different jobs have different characteristics, it is often important to determine the average slowdown caused by failure on user jobs. *Failure slowdown* (FSD), defined as the ratio of the time delay caused by failure to failure-free job execution time, can be used to serve the purpose.

In this paper, we propose three rescheduling strategies, each focusing on reducing one specific failure impact as discussed above.

1. *Service Unit Loss Driven (SUL-D).* It aims at minimizing the loss of service units (defined as the product of the number of compute nodes and the amount of time wasted due to failure). Not knowing the exact failure time, we assume that failures are uniformly distributed in the next interval. Hence, for a suspicious job j_i, its rescheduling gain can be estimated as

$$v_i = f_i \cdot n_i \cdot \left(t + \frac{I_F}{2} - t_{last}^i - O_F \right), \tag{2}$$

where $f_i = 1 - (1 - precision)^{n_i^s}$.

Here, t is the current time, I_F is FARS interval, n_i is job size, n_i^s is number of suspicious nodes, f_i is job failure probability, and t_{last}^i is the most recent time where the job can be safely started from, e.g., the last checkpoint or the job start time. As we can see, $(t + \frac{I_F}{2} - t_{last}^i)$ is the amount of time saved by rescheduling j_i. O_F is the overhead of reallocating the job, which can be obtained by tracking operational costs at runtime. Depending on the specific implementation of process migration, O_F may differ. For example, in case of a stop-and-restart design that checkpoints the application and then restarts it on a new set of resources [18], O_F can be approximated by $(O_{ckp} + O_r)$; in case of live migration support, it can be estimated by O_r. Both O_{ckp} and O_r can be tracked in practice [17].

2. *Job Failure Rate Driven (JFR-D).* It aims at rescheduling as many suspicious jobs as possible. This strategy intends to minimize job interrupts caused by failure so as to improve user satisfaction of system service. Hence, for a suspicious job j_i, its rescheduling gain v_i is

$$v_i = f_i \cdot 1. \tag{3}$$

3. *Failure Slowdown Driven (FSD-D).* The objective of this strategy is to minimize the slowdown caused by failure. For a suspicious job j_i, failure impact includes job re-queuing cost O_q, its restart cost O_r and the recomputation time on the lost work. The recomputation time can be estimated in the same way as in the SUL-D strategy; O_q can be determined based on historical data, such as by using LAST or MEAN predictive method [22]. Thus, the gain of rescheduling job i can be estimated as

$$v_i = f_i \cdot \left(t + \frac{I_F}{2} - t_{last}^i + O_q + O_r - O_F\right)/t_i, \tag{4}$$

where t_i is failure-free job execution time.

4.3 Dynamic Programming

After setting the gain value in (1), the 0-1 knapsack model can be solved in pseudopolynomial time by using dynamic programming method [6]. To avoid redundant computation, we use the tabular approach by defining a 2D table G, where $G[k, w]$ denotes the maximum gain that can be achieved by rescheduling suspicious jobs $\{j_i^s | 1 \le i \le k\}$ with no more than w spare nodes, where $1 \le k \le J_s$ and $1 \le w \le N_s$. $G[k, w]$ has the following recursive feature:

$$= \begin{cases} 0 & kw = 0, \\ G[k-1, w] & n_k^s > w, \\ max\big(G[k-1, w], v_k + G\big[k-1, w - n_k^s\big]\big) & n_k^s \le w. \end{cases} \tag{5}$$

The solution $G[J_s, S]$ and the corresponding binary vector X determine the selection of suspicious jobs for rescheduling. The computation complexity of (5) is $O(J_s \cdot S)$.

4.4 Residual Issue

After the aforementioned job-oriented selection, it is possible that there are some spare nodes and suspicious jobs left, where the spare nodes are not sufficient to accommodate any suspicious job. We call this *a residual issue.* To address the issue, we adopt a best-effort method to select one more suspicious job for rescheduling [16]. Suppose there are R spare nodes left after the job-oriented selection, FARS calculates rescheduling gain for each of the remaining suspicious jobs and selects the job with the maximal gain value. The calculation of v_i is the same as shown in (2), (3), and (4), except that $f_i = 1 - (1 - precision)^{(n_i^s - R)}$.

5 Evaluation Methodology

Our experiments were based on event-driven simulations by means of synthetic data and real traces collected from production systems. An event driven simulator was developed to emulate a HPC system using FCFS/EASY scheduling [16]. We compared FARS-enhanced FCFS/EASY as against the plain FCFS/EASY. In the rest of this paper, we simply use the term SUL-D, JFR-D, FSD-D, and FCFS to denote three rescheduling strategies and the plain FCFS/EASY. This section describes our evaluation methodology, and the results will be presented in the next section.

5.1 Simulator

The simulator was driven by three classes of events: 1) *job events* including job arrivals and terminations; 2) *failure events* including failure arrivals and repairs; and 3) *fault tolerant events* including job checkpointing and rescheduling events. Upon a job arrival, the simulator was informed of job submission time, job size, and its estimated runtime. It started the job or placed it in the queue based on FCFS/EASY. Upon a job termination, it removed the job and scheduled other queued jobs based on FCFS/EASY. Upon a node failure, the simulator suspended the node and the job running on it for failure repair. After failure repair, the simulator resumed the job that was suspended by the failure and the time delay was added into job completion time. Each job was checkpointed periodically, and the checkpoint frequency for each job was set based on the widely used formula [42]. Upon checkpointing events, checkpoint overhead was added into job completion time. In case of FARS rescheduling, FARS overhead was added into the corresponding job completion time.

The behavior of a failure predictor was emulated and its prediction accuracy was controlled by two metrics:

1. *Recall.* If there exists a failure on a node in the next interval, the predictor reports a failure on the node with the probability of $recall$.
2. *Precision.* Suppose the predictor has totally reported x failures for the intervals with actual failures. According to the definition of $precision$, for intervals without an actual failure, the predictor randomly selects $\frac{x \times (1 - precision)}{precision}$ intervals and gives a false alarm on each of them.

5.2 Synthetic Data and System Traces

Both synthetic data and real machine traces were used for the purpose of comprehensive evaluation. Synthetic data was used to extensively study the sensitivity of FARS to a variety of system parameters, whereas machine traces were

critical for assessing the practical effectiveness of FARS in real computing environments.

5.2.1 Synthetic Data

Our synthetic data was generated to emulate a 512-node cluster:

- *Job events.* The job arrivals, lengths, and sizes were based on exponential distributions, where the means were set to 1,000.0 seconds, 1,500.0 seconds, and 10 CPUs, respectively. The system utilization rate was set to 70 percent, to reflect a moderate system load.
- *Failure events.* Exponential and Weibull distributions are two commonly used models for failure arrivals [12], [32]. Hence, we generated two sets of failure events with *Exponential distribution* and *Weibull distribution*, respectively. For the Weibull distribution, to reflect the commonly observed "bathtub" behavior [35], we used a composite Weibull distribution with three subpopulations where the shape parameters were set to $\beta = 0.5$, $\beta = 1.0$, and $\beta = 1.5$, respectively. They were used to simulate the burn-in, normal, and worn-out phases of the system [23]. To study the sensitivity of FARS to MTBF, we tuned the mean of Exponential distribution λ^{-1} and the scale parameter of Weibull distribution η. The failure repair process was based on an exponential distribution at a mean of mean-time-to-repair (MTTR).

5.2.2 System Traces

System traces were collected from 512-node production systems:

- *Job events.* A six-month job log was collected from the Lonestar system at Texas Advanced Computing Center (TACC). The cluster contained 512 Dell PowerEdge 1,750 compute nodes, 13 Dell PowerEdge 2,650 I/O server nodes, and 2 Dell PowerEdge 2,650 login/management nodes. The job log only contained workload information from the compute nodes. As shown in Table 2, the system utilization rate was 94 percent. The average job running time was 3,171.0 seconds, the job arrival rate was 0.0044, and the average job size was 14 CPUs.
- *Failure events.* Due to the unavailability of a corresponding failure log from the TACC Lonestar, we used a failure log from a comparable Linux cluster at NCSA [19]. The machine had 520 two-way SMP 1-GHz Pentium-III nodes (1,040 CPUs), 512 of which were compute nodes (2-Gbyte memory), and the rest were storage nodes and interactive access nodes (1.5-Gbyte memory). The MTBF was 0.79 hour for the system and was 14.16 days per node. The MTTR was about 1.73 hours.

5.3 Evaluation Metrics

Three *performance metrics* and three *reliability metrics* were used for evaluation:

1. *Average response time (Resp).* Let J be the total number of jobs, c_i be the completion time of job j_i, and a_i be job arrival time. The average response time of the system is defined by
$$\left[\sum_{1\le i\le J}(c_i - a_i)\right] \Big/ J.$$
2. *Utilization rate (Util).* Let T be the total elapsed time for J jobs, N be the number of nodes in the system, s_i be the start time of job i, and n_i be the size of job i. System utilization rate is defined as
$$\left[\sum_{1\le i\le J}(c_i - s_i)\cdot n_i\right]/(N\cdot T).$$
3. *Throughput (Thru).* It is defined as the average number of completed jobs in a unit of time.
4. *Service unit loss (SUL).* Defined as the total amount of wasted service units (i.e., product of wall clock hours and number of nodes) caused by failure. This metric directly indicates the amount of computing cycles lost due to failures—an important metric to both system managers and users.
5. *JFR.* Defined as the ratio between the number of failed jobs and the total number of jobs submitted. It reflects percentage of jobs that are interrupted by failures, an important indicator of system's quality of service.
6. *FSD.* Defined as the ratio of time delay caused by failure to failure-free job execution time, average over the total number of jobs. To mitigate the impact of small jobs, a threshold of 10.0 seconds was applied in the calculation of FSD. Different from the widely used scheduling metric *bounded slowdown*, this metric provides a direct indication of failure impact on job completion time.

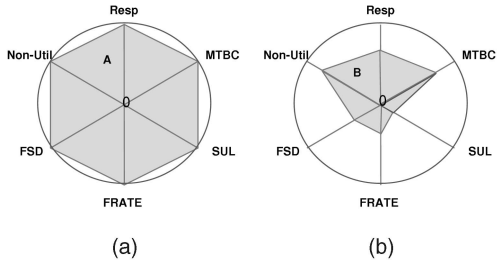


Fig. 3. Exemplar of 6D Kiviat graph. The range of each metric is from zero to the largest value observed in the experiments. The relative gain of method B over A is defined as $\frac{K(A)-K(B)}{K(A)}$, where $K(\cdot)$ denotes the K-value of a method. The K-value of a method is the shaded area. The smaller the K-value is, the better the performance a method has.

In addition, a 6D Kiviat graph was employed to provide a composite view of these metrics (see Fig. 3) [46]. The graph consists of six dimensions, each representing one of the aforementioned metrics emanating from a central point. Note that *nonutilization rate* (defined as $(1 - Util)$) and *mean-time-between-completion (MTBC)* (defined as $(1/Thru)$) are used in the graph. The range of each metric is from zero to the largest value observed in the experiments. As shown in the figure, the composite view of six metrics is the shaded area. The smaller the area is, the better the performance is.

TABLE 3
Baseline Configuration

System load (i.e., utilization)	0.7
Node MTBF	14 days
Node MTTR	45 minutes
Job restart cost O_r	3 minutes
Job checkpoint overhead O_{ckp}	3 minutes
FARS overhead O_F	6 minutes
FARS interval I_F	30 minutes
Prediction *precision*	0.7
Prediction *recall*	0.7
Number of jobs	21048
Duration of jobs	249.79 days

To compare two methods A and B, we calculate the relative gain of B over A as $\frac{K(A)-K(B)}{K(A)}$, where $K(\cdot)$ denotes the K-value of a method. A K-value of a method is defined as the shaded area of its associated Kiviat graph. Obviously, the smaller the K-value is, the better the performance a method has.

6 Experimental Results

6.1 Results on Synthetic Data

A series of simulations were conducted to analyze the impact of system load, node MTBF, and prediction accuracy on FARS. The baseline configuration is summarized in Table 3. These parameters and their corresponding ranges were chosen according to the results reported in [23], [41], [43], and our experiments [17].

6.1.1 Baseline Results

Baseline results are presented in Fig. 4 (Exponential Failure Distribution) and Fig. 5 (Weibull Failure Distribution). FARS-enhanced methods outperform FCFS in terms of both performance and reliability metrics, with the improvement on reliability metrics being more substantial. Let us first take a look at reliability metrics. As we can see, the use of FARS can reduce SUL by more than 2,000 CPU hours and the number of failed jobs is reduced from 600+ to 400 under both failure distributions. We also observe that each rescheduling strategy achieves the best improvement with regard to its target metric. As an example, SUL-D is able to minimize SUL. This result implicitly validates the calculation of v_i in (2), (3), and (4).

With regard to performance metrics, a noticeable improvement is observed on response time. For example, under Exponential distribution, the average response time is reduced from 19,400+ seconds by using FCFS to around 18,000 seconds by using either of the rescheduling strategies. It indicates that the improvement on reliability by using FARS can lead to an increase in scheduling performance. In terms of utilization rate and throughput, the improvement is relatively trivial. This is because utilization rate and throughput are mainly determined by job arrivals and the scheduling policy. This observation also indicates the necessity of using other metrics, in addition to performance metrics, to measure system productivity in the presence of failure.

It is hard to tell which rescheduling strategy is better by simply comparing the results in Figs. 4 and 5. To provide a holistic comparison of different rescheduling strategies, we calculated their relative gains over FCFS by using Kiviat graph as shown in Fig. 3. All the strategies are able to provide more than 30 percent gain over FCFS. We also observe that the gain achieved by FSD-D is relatively lower than the other two. We believe this stems from the fact that the estimation of v_i in (4) is not precise. Getting an accurate estimation of O_q is difficult as it is influenced by many dynamic factors such as failure repair time, job queue status, and resource availability. This observation implies that more sophisticated methods such as the one presented in [22] may be applied to improve prediction of job queuing time.

In summary, the results indicate that the use of FARS can greatly improve system productivity in the presence of failure, with the relative improvement of over 30 percent as compared to the case without using it. The selection of rescheduling strategy depends on the primary objective of fault management. In general, if the objective is to improve

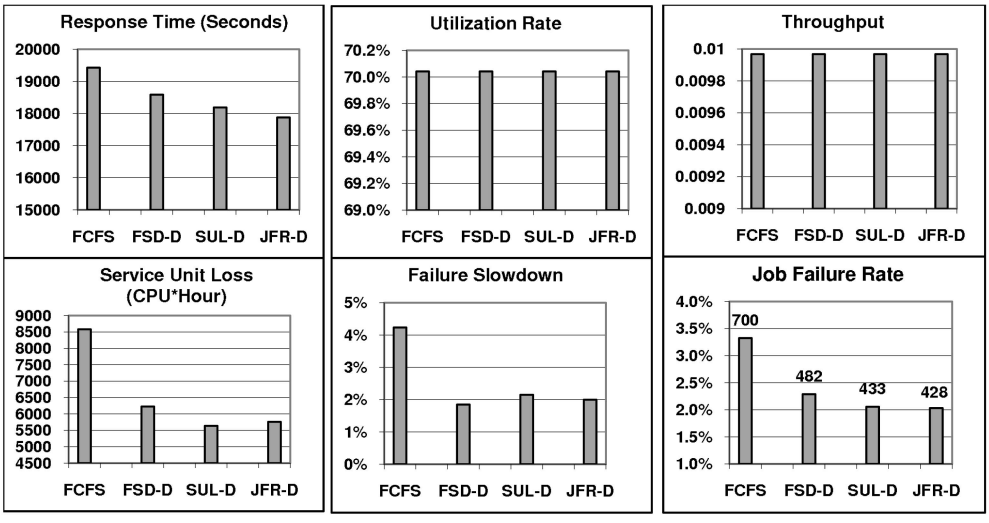


Fig. 4. Synthetic data under Exponential failure distribution. The data label in the plot of JFR indicates the actual number of failed jobs. The composite gain over FCFS is 34.02 percent (FSD-D), 36.35 percent (SUL-D), and 37.34 percent (JFR-D).

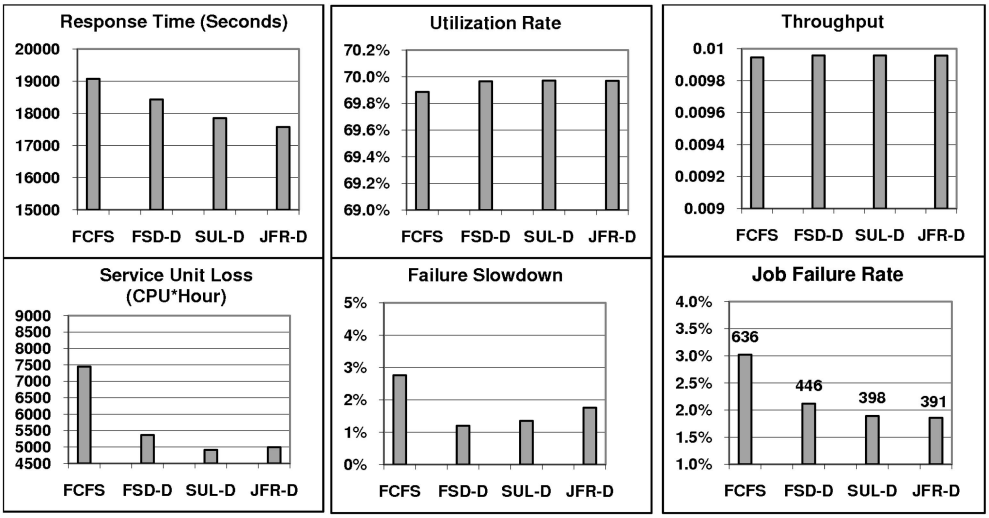


Fig. 5. Synthetic data under Weibull bathtub failure distribution. The data label in the plot of JFR indicates the actual number of failed jobs. The composite gain over FCFS, based on the Kiviat graph, is 33.73 percent (FSD-D), 36.62 percent (SUL-D), and 33.84 percent (JFR-D).

the overall productivity, then both SUL-D and JFR-D are good candidates.

6.1.2 Sensitivity to System Load

In this set of simulations, we varied failure-free system utilization rate from 0.1 to 0.95 by adjusting job service times t_i. The purpose is to assess the impact of system load on FARS.

The raw data achieved by using FCFS are listed in Tables 4 and 5. Fig. 6 presents relative improvements by using FARS. First, let us look at Figs. 6a and 6b. There are six curves in each plot, representing the relative gains of six metrics over FCFS as system load changes. It shows that the performance of FARS drops as system load increases. This is reasonable because a higher system load means a smaller sized spare pool, thereby degrading the effectiveness of FARS. The plots also show that for each metric, its trends under both failure distributions are similar, although the absolute value under Weibull distribution is lower than that under Exponential distribution. Hence, failure distribution does not have a significant impact on the performance of FARS. A major reason is that FARS is mainly influenced by failure prediction, instead of by long-term failure characteristics.

Different metrics exhibit different trends, according to the figure. The curves of utilization and throughput stay close to the x-axis, meaning the relative gain on these metric is close to 0. As stated earlier, this is due to the fact that both metrics are mainly determined by job arrivals and the scheduling policy. The curve of response time is heading up when the load increases to 0.9. After that, it starts to drop. When the load is beyond a certain point, meaning that the system is about saturating, spare nodes become scarce, thereby limiting the capability of FARS.

More apparent changes are observed on SUL and JFR, when system load increases from 0.1 to 0.95. Recall that FARS adopts a dynamic strategy for spare pool allocation. A higher load leads to fewer spare nodes, thereby limiting the effect of FARS for failure avoidance. The same trend is observed on FSD, except that it drops more sharply when system load increases beyond 0.8. We attribute this to the instability of FSD: when the load is high, more dynamics is introduced into the system, thereby making it hard to accurately estimate the rescheduling gain v_i, particularly O_q.

Fig. 6c presents the overall gains of different rescheduling strategies over FCFS under Exponential and Weibull failure distributions. As system load increases from

TABLE 4
Raw Results by Using Plain FCFS/EASY under Different System Loads

Load	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
Resp	1392(E) 1403(W)	2599(E) 2619(W)	4169(E) 4212(W)	6187(E) 6252(W)	8867(E) 9157(W)	12470(E) 12819(W)	19429 (E) 19070(W)	41142(E) 39816(W)	178782(E) 143803(W)	751660(E) 708973(W)
Util	11.270(E) 11.270(W)	20.045(E) 20.037(W)	29.999(E) 29.987(W)	40.453(E) 40.436(W)	50.880(E) 50.861(W)	60.547(E) 60.494(W)	70.043(E) 69.887(W)	79.903(E) 79.923(W)	87.005(E) 87.688(W)	89.470(E) 90.075(W)
Thru	.01003(E) .01003(W)	.01002(E) .01002(W)	.01001(E) .01001(W)	.01000(E) .01000(W)	.00999(E) .00998(W)	.00998(E) .00997(W)	.00997(E) .00995(W)	.00991(E) .00992(W)	.00952(E) .00959(W)	.00807(E) .00812(W)
SUL	133(E) 168(W)	302(E) 342(W)	565(E) 498(W)	811(E) 724(W)	1041(E) 920(W)	1292(E) 1101(W)	1467(E) 1274(W)	1651(E) 1440(W)	1808(E) 1548(W)	2224(E) 1971(W)
FSD	.00881(E) .01305(W)	.01571(E) .02224(W)	.03641(E) .02206(W)	.04543(E) .02603(W)	.04650(E) .04361(W)	.03078(E) .01879(W)	.04235(E) .02761(W)	.02843(E) .02033(W)	.02583(E) .03230(W)	.05761(E) .01835(W)
JFR	.0039 (E) .0050 (W)	.0091 (E) .0092 (W)	.0140 (E) .0136 (W)	.0187 (E) .0184 (W)	.0248 (E) .0238 (W)	.0289 (E) .0265 (W)	.0332 (E) .0302 (W)	.0399 (E) .0333 (W)	.0419 (E) .0379 (W)	.0529 (E) .0458 (W)

There are two values in each cell: the upper one is from Exponential distribution (E) and the bottom one is from Weibull distribution (W).

TABLE 5
Raw Results by Using Plain FCFS/EASY under Different Node MTBFs

Node MTBF	448 days	224 days	112 Days	56 days	28 days	14 days	7 days	3.5 days	1.75 days
Resp	15532(E) 15516(W)	15618(E) 15702(W)	15913(E) 15836(W)	16036(E) 16311(W)	16983(E) 17191(W)	19429(E) 19070(W)	25522(E) 25216(W)	59453(E) 55196(W)	798254(E) 459605(W)
Util	70.043(E) 70.017(W)	70.043(E) 70.043(W)	70.043(E) 70.020(W)	70.042(E) 69.966(W)	70.043(E) 70.013(W)	70.043(E) 69.887(W)	69.750(E) 69.562(W)	69.154(E) 69.185(W)	54.893(E) 58.810(W)
Thru	.00997(E) .00996(W)	.00997(E) .00997(W)	.00997(E) .00997(W)	.00997(E) .00996(W)	.00997(E) .00996(W)	.00997(E) .00995(W)	.00993(E) .00990(W)	.00984(E) .00985(W)	.00781(E) .00837(W)
SUL	23(E) 65(W)	58(E) 115(W)	184(E) 284(W)	406(E) 331(W)	629(E) 775(W)	1467(E) 1274(W)	2838(E) 2774(W)	5677(E) 4893(W)	11979(E) 10017(W)
FSD	.00082(E) .00069(W)	.00131(E) .00282(W)	.00233(E) .00382(W)	.01005(E) .00756(W)	.00926(E) .01757(W)	.04235(E) .02761(W)	.05369(E) .04262(W)	.09220(E) .12889(W)	.33466(E) .21687(W)
JFR	.0009 (E) .0017 (W)	.0020 (E) .0033 (W)	.0043 (E) .0069 (W)	.0103 (E) .0076 (W)	.0153 (E) .0178 (W)	.0332 (E) .0302 (W)	.0676 (E) .0631 (W)	.1392 (E) .1188 (W)	.3011 (E) .2539 (W)

There are two values in each cell: the upper one is from Exponential distribution (E) and the bottom one is from Weibull distribution (W).

0.1 to 0.7, the overall gain achieved by FARS smoothly decreases to 30 percent. When system load increases beyond 0.70, the gain decreases to 15 percent. In both plots, we observe that the performance of FSD-D is slightly lower than those achieved by SUL-D and JFR-D. This result is consistent with the results obtained in Figs. 4 and 5.

6.1.3 Sensitivity to Node MTBF

We also studied FARS sensitivity to failure rate by tuning node MTBFs based on the baseline value with the ratio linearly changed from 1/32 to 8. In other words, node MTBF was varied from 448 days to 1.75 days.

Similar to Fig. 6, we first plot the relative gains of individual metrics in Figs. 7a and 7b, and then present the overall gains of different FARS strategies in Fig. 7c.

The curves of utilization rate and system throughput are close to zero, meaning there is no significant change on these metrics. In terms of job response time, the curve first heads up from 0 percent to around 40 percent as node MTBF decreases from 448 to 3.5 days. This is because a

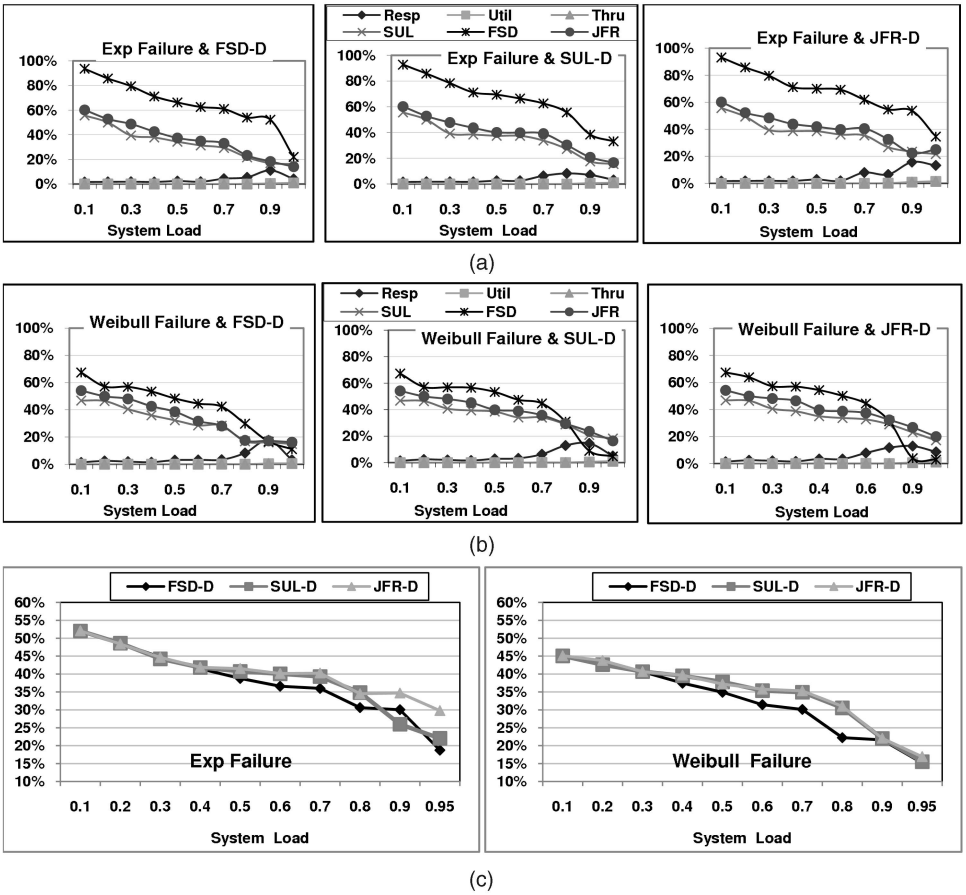


Fig. 6. Sensitivity to system load. (a) and (b) plot the relative gain of each metric over FCFS, under Exponential Distribution and Weibull Distribution, respectively. (c) presents the overall gains of different rescheduling strategies, based on Kiviat graph. The performance of FARS drops as system load increases. FARS always outperforms FCFS by over 15 percent, even when the system has high load.

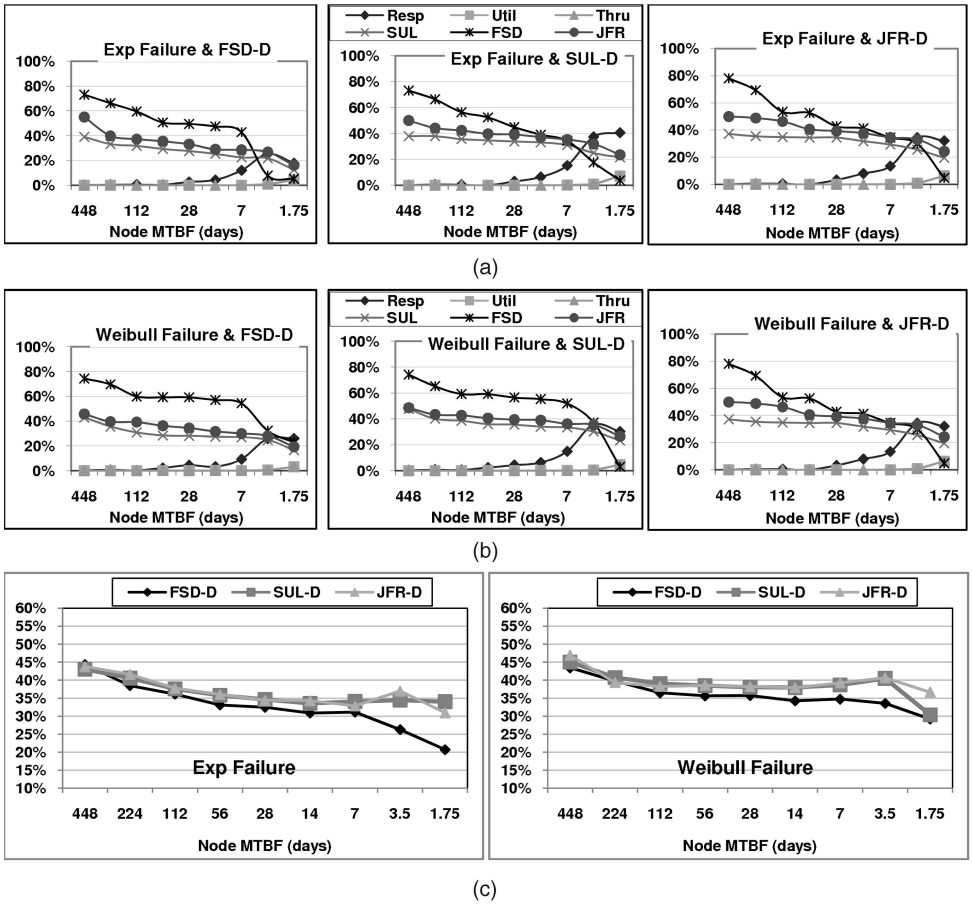


Fig. 7. Sensitivity to Node MTBF. (a) and (b) plot the relative gain of each metric over FCFS under Exponential and Weibull failures, respectively. (c) presents the overall gains of different rescheduling strategies, where the overall gain is calculated based on Kiviat graph. The results show that in general, the benefit brought by FARS drops as node MTBF decreases.

lower value of MTBF means higher failure rate, thereby resulting in more opportunities for FARS to avoid failures, and consequently, reducing job response time. When node MTBF drops below 3.5 days, the curve starts to drop. We believe this is caused by the insufficiency of spare nodes. When the system becomes extremely unreliable, suspicious nodes significantly outnumber the available spare nodes, thereby degrading the performance of FARS.

As shown in the figure, the curves for reliability metrics generally head down as node MTBF is getting smaller. The curves of SUL and JFR gradually drop from 50 percent to 20 percent, whereas the FSD curve (FSD) drops more quickly. A major reason for the fast drop of FSD is due to the instability of FSD. As discussed earlier, getting an accurate estimation of O_q is difficult, especially when failure interrupts become more frequent.

Fig. 7c shows the composite gains of different rescheduling strategies over FCFS. As we can see, the performance achieved by FARS drops as node MTBF decreases. For the systems whose node MTBFs are larger than 14 days, FARS can provide more than 30 percent performance gain; when system nodes become unreliable with a low MTBF value (e.g., lower than 14 days), FARS still outperforms FCFS/EASY by more than 20 percent. We also notice that when node MTBF drops below 7 days (extremely unreliable), the gain achieved by FARS decreases dramatically. This is caused by insufficient spare nodes due to high failure rate. We also observe that the performance of FSD-D is slightly lower than those achieved by SUL-D and JFR-D, which is consistent with the results shown in the previous figure.

The above study also justifies that promising gain may be achieved in production systems by using FARS. According to the failure data repository [32], node MTBFs of real systems vary from 120 days to a couple of years. By a simple projection based on the gain curves in Fig. 7, for these systems, FARS can provide more than 35 percent gain.

6.1.4 Sensitivity to Prediction Accuracy

Obviously, the performance of FARS depends on prediction accuracy. In this set of simulations, we simulated different levels of prediction accuracies and quantified the amount of gain achieved by FARS under different prediction *precision* and *recall* rates.

In Fig. 8, we show the distribution of composite gain achieved by SUL-D as against FCFS, where *precision* and *recall* range between 0.1 and 1.0. We have also analyzed the sensitivities of JFR-D and FSD-D to different prediction accuracies. Their distributions (not shown) are similar to the results shown in Fig. 8.

The figure clearly shows that the more accurate a prediction mechanism is, the higher the gain SUL-D can provide. For example, under both failure distributions, the best performance is achieved when *precision* and *recall* are

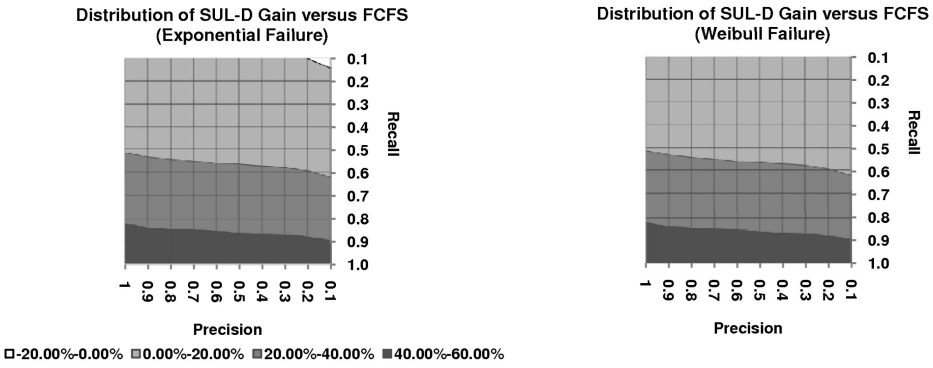


Fig. 8. Distribution of SUL-D gain versus FCFS. Similar distributions are observed for JFR-D and FSD-D (not shown).

1.0 (perfect prediction) and the worst case occurs when both are set to 0.1 (meaning that 90 percent of the predicted failures are false alarms and 90 percent of the failures are not captured by the failure predictor). With a perfect prediction, the optimal gain achieved by SUL-D is more than 50 percent. We also notice that as long as both *precision* and *recall* are higher than 0.2, SUL-D always outperforms the plain FCFS. In other words, although false alarms may cause unnecessary job rescheduling, the benefit brought by FARS often overcomes its negative impact, under the condition that the failure predictor can capture 20 percent of failures with the false alarm rate lower than 80 percent.

6.2 Results on System Trace

Fig. 9 presents the results obtained with the system traces by using the same baseline configuration as listed in Table 3, except that the failure events and the job events are from real system traces.

Consistent with the results on synthetic data, FARS outperforms FCFS, especially in terms of reliability metrics. Each rescheduling policy is capable of minimizing its target metric. As an example, SUL-D gives the best result in terms of SUL, whereas JFR-D is good at minimizing JFR. All these rescheduling strategies are capable of providing over 35 percent composite gain, as compared to FCFS.

We also examined the impact of failure prediction by tuning prediction accuracies. Fig. 10 shows the distribution of the composite gain achieved by SUL-D as against FCFS, where *precision* and *recall* vary from 0.1 to 1.0. Similar trends are observed by using JFR-D and FSD-D, so we only present the results by using SUL-D here.

The maximum gain 53 percent is achieved under a perfect prediction where *precision* and *recall* are 1.0. The negative gain (−4 percent) is observed only when both parameters are as low as 0.1. Although the trends are similar to those obtained with synthetic data shown in Fig. 8, we notice that the performance of SUL-D drops fast as prediction precision decreases. This is caused by the higher load in the job log, which reaches 84 percent even under a failure-free computing environment. A lower *precision* means more false alarms, which consequently demands more spare nodes. This situation is exacerbated when the available spare resources are limited under a high system load.

6.3 Result Summary

In summary, our experiments with synthetic data and real system traces have shown the following:

- FARS can effectively improve system productivity as long as failure prediction is capable of predicting

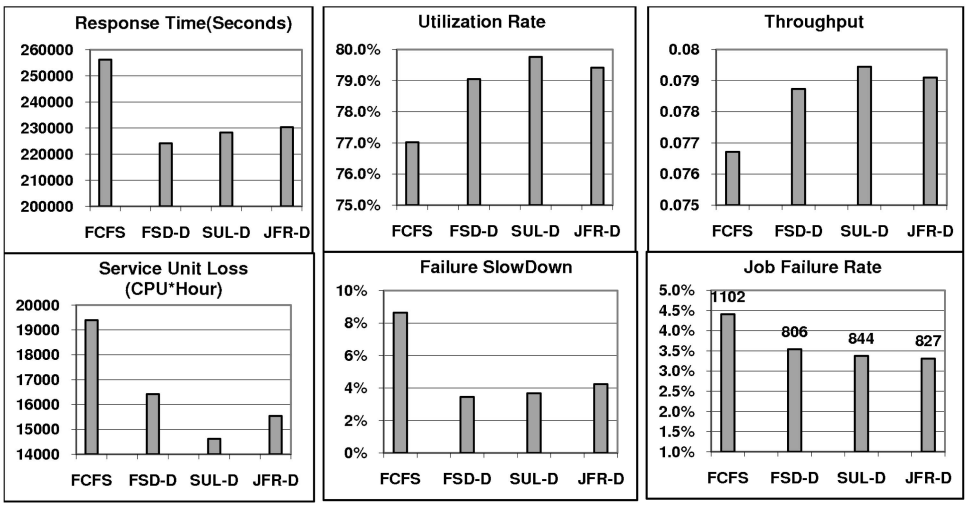


Fig. 9. Results on system traces. The data label in the plot of JFR indicates the actual number of failed job. The relative gain over FCFS, based on the Kiviat graph, is 35.45 percent (FSD-D), 38.47 percent (SUL-D), and 35.21 percent (JFR-D).

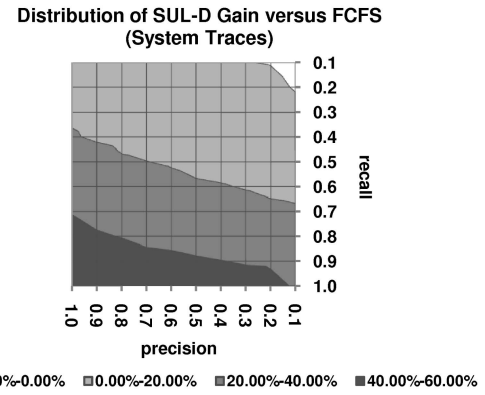


Fig. 10. Distribution of SUL-D gain versus FCFS on system traces.

25 percent of failure events with a false alarm rate lower than 75 percent (see Figs. 8 and 10).

- System load implicitly determines number of spare nodes, thereby impacting the performance of FARS. In the systems under a moderate load (e.g., $\leq$ 0.7), the gain achieved by FARS is always above 30 percent. Even when the system load is as high as 0.95, the gain is still above 15 percent (see Fig. 6).
- For systems with node MTBF ranging between several weeks to a couple of years, FARS is capable of providing more than 30 percent gain in terms of system productivity (see Fig. 7).

7 Related Work

Considerable research has been conducted on fault management for HPC. They generally fall into two categories, one on fault-aware scheduling and the other on runtime fault tolerance techniques.

Fault-aware scheduling focuses on making an appropriate mapping of jobs or tasks to compute resources by taking system reliability into consideration. The objective is to optimize performance metrics, such as job response time or job slowdown [1]. In [10], Hariri and Raghavendra proposed two reliability-aware task allocation algorithms to optimize the probability of successful task completions. Shatz et al. presented a task-graph-based performance model for maximizing a reliability cost function, and developed several scheduling algorithms based on this model [34]. Dogan and Ozguner investigated two reliability-aware cost functions to enable scheduling of precedence-constrained tasks in heterogeneous environments [7]. Srinivasan and Jha introduced the safety concept for reliable task allocation in distributed systems [38]. There are also several studies on utilizing redundant resources for task scheduling. Kartik and Murthy proposed a branch-and-bound algorithm to maximize the reliability of distributed systems by using two-level redundancy [14]. Recently, an increasing attention has been paid to fault-aware scheduling in the field of HPC. In [43], Zhang et al. suggested utilizing temporal and spatial correlations among failure events for better scheduling. Oliner et al. presented a fault-aware job-scheduling algorithm for Blue Gene/L systems by exploiting node failure probabilities [24]. In [15], a fault-aware scheduling was presented for the HA-OSCAR framework.

Fault-aware scheduling mainly focuses on providing an optimal mapping of *inactive jobs* (i.e., jobs in the queues) onto available resources based on *long-term failure models*, such as observed failure characteristics or distributions. Different from fault-aware scheduling, this study emphasizes on dynamically adjusting the placement of *active jobs* (i.e., running jobs) to avoid imminent failures discovered by *short-term failure predictors*. Here, "short-term" means the time is on the order of several minutes to an hour. There are several active projects on exploiting data mining and pattern recognition technologies for the development of short-term failure predictors [31], [45]. For example, Fu and Xu have designed and implemented a framework called hPREFECTS for failure prediction in networked computing systems [47]; in our own studies [44], [45], [48], we have investigated online failure prediction for large-scale systems by applying ensemble learning and automated data reduction techniques.

Fault-aware scheduling and FARS complement each other, where fault-aware scheduling prevents inactive jobs from the failures that are well captured in the long-term failure models and FARS enables active jobs to avoid imminent failures that may not follow any long-term pattern but can be discovered via runtime diagnosis.

Checkpointing and process migration are two prevailing fault tolerance techniques. Checkpointing centers upon reducing recovery cost by periodically saving an intermediate snapshot of the system to a stable storage. A detailed description and comparison of different checkpointing techniques can be found in [9]. A number of checkpointing libraries and tools have been developed for HPC, and examples include libckpt [27], BLCR [11], open MPI [4], MPICH-V [3], and the Cornell Checkpoint (pre)Compiler (C3) [33]. In addition, a number of optimization techniques have been developed to reduce its cost and overhead [25], [28], [42]. Oliner et al. proposed to dynamically skip unnecessary checkpoints via failure prediction [23]. In essence, checkpointing is reactive, meaning that it only deals with failures after their occurrences. *In contrast to these studies on checkpointing*, the proposed FARS emphasizes the use of proactive action (i.e., reallocating running jobs) to avoid failures.

Unlike checkpointing, process migration takes preventive actions—transferring application processes away from failure-prone nodes—before failures. Intensive research has been done on process migration. Process migration can be performed at the kernel level or the user level. Kernel-level migration requires a modification of the operating system, whereas user-level methods allow migration without changing the operating system kernel. A detailed survey regarding migration can be found in [36]. There are several active projects on providing process migration support for sequential and parallel applications. For instance, Condor allows user-level process migration by first checkpointing the application and then restarting it on a new set of resources [18]. The PCL protocol used in the MPICH-V package applies a similar research effort on developing live migration support for MPI applications. Du and Sun proposed distributed migration protocols to support live migration [8]. In the AMPI project, a proactive migration

scheme was proposed to move objects to reliable nodes based on fault prediction provided by hardware sensors [5]. Nagarajan et al. discussed the use of Xen virtual machine technology to facilitate transparent process migration [21].

The majority of research focuses on the development and optimization of runtime techniques, yet there is a lack of systematic study on fault-aware runtime management by taking account of various factors. *This study bridges the gap by presenting runtime strategies for spare node allocation and job rescheduling.* These strategies coordinate jobs and computing resources in response to failure prediction. To the best of our knowledge, we are among the first to comprehensively and systematically study FARS for HPC.

The term of *job rescheduling* has been used in Grid computing. For example, in the GrADS project, an application-level job migration and processor swapping approach was presented to reschedule a Grid application when a better resource is found [2]. Fundamentally different from these studies, our work utilizes job rescheduling to improve system resilience to failures. The issues such as spare node allocation and job selection for rescheduling are not addressed in Grid computing.

8 Conclusions

Although much work remains to make FARS fully operational, our results have shown the importance and potential of exploiting runtime failure prediction to improve system productivity. In particular, we have presented runtime strategies for coordinating jobs and computing resources in response to failure prediction. Our extensive experiments have indicated that FARS is capable of improving system productivity as long as the failure predictor can capture 25 percent of failure events with a false alarm rate lower than 75 percent. With the advance in failure prediction, we believe FARS will become more effective. The proposed 0-1 knapsack model gives a general and flexible method for job rescheduling, from which we can derive a variety of rescheduling strategies.

Our study has some limitations that remain as our future work. First, we are in the process of collecting more workloads and failure events from production systems to further evaluate the effectiveness of FARS. Second, we plan to integrate FARS with fault-aware scheduling work such as [24]. We expect that this combination can further improve system productivity. Lastly, exploiting failure patterns for failure prediction is an on-going project in our group [44], [45], [48]. Integrating FARS with the work on failure prediction is part of our future work. Our ultimate goal is to implement FARS, along with failure prediction work, in job scheduling systems for better fault management of HPC.

Acknowledgments

The authors would like to thank Warren Smith at TACC for providing the Lonestar job log. This work was supported in part by the US National Science Foundation Grants CNS-0720549, CCF-0702737, and a TeraGrid Compute Allocation. Some preliminary results of this work were presented in [16]. *Zhiling Lan is the corresponding author.*

References

[1] S. Albers and G. Schmidt, "Scheduling with Unexpected Machine Breakdowns," *Discrete Applied Math.,* vol. 110, nos. 2-3, pp. 85-99, 2001.

[2] F. Berman, H. Casanova et al., "New Grid Scheduling and Rescheduling Methods in the GrADS Project," *Int'l J. Parallel Programming,* vol. 33, no. 2-3, pp. 209-229, 2005.

[3] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello, "MPICH-V: A Multiprotocol Automatic Fault Tolerant MPI," *Int'l J. High Performance Computing and Applications,* vol. 20, no. 3, pp. 319-333, 2006.

[4] E. Gabriel, G. Fagg et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," *Proc. 11th European PVM/MPI Users' Group Meeting (Euro PVM/MPI '04),* Sept. 2004.

[5] S. Chakravorty, C. Mendes, and L. Kale, "Proactive Fault Tolerance in MPI Applications via Task Migration," *Proc. Int'l Conf. High Performance Computing (HiPC '06),* p. 485, 2006.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. The MIT Press and McGraw-Hill Book, 2001.

[7] A. Dogan and F. Ozguner, "Reliable Matching and Scheduling of Precedence-Constrained Tasks in Heterogeneous Distributed Computing," *Proc. Int'l Conf. Parallel Processing (ICPP '00),* pp. 307-314, 2000.

[8] C. Du and X. Sun, "MPI-Mitten: Enabling Migration Technology in MPI," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGRID '06),* pp. 11-18, 2006.

[9] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A Survey of Rollback Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys,* vol. 34, no. 3, pp. 375-408, 2002.

[10] S. Hariri and C. Raghavendra, "Distributed Functions Allocation for Reliability and Delay Optimization," *Proc. ACM Fall Joint Computer Conf. (FJCC '86),* pp. 344-352, 1986.

[11] P. Hargrove and J. Duell, "Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters," *Proc. Scientific Discovery through Advanced Computing (SciDAC),* 2006.

[12] R. Jain, *The Art of Computer Systems, Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, 1991.

[13] *Parallel Workloads Archive,* http://www.cs.huji.ac.il/labs/parallel/workload/, 2008.

[14] S. Kartik and C. Murthy, "Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems," *IEEE Trans. Computer Systems,* vol. 46, pp. 719-724, 1997.

[15] K. Limaye, C. Leangsuksun, and A. Tikotekar, "Fault Tolerance Enabled HPC Scheduling with HA-OSCAR Framework," *Proc. High Availability and Performance Workshop (HAPCW),* 2005.

[16] Y. Li, P. Gujrati, Z. Lan, and X. Sun, "Fault-Driven Re-Scheduling for Improving System-Level Fault Resilience," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2007.

[17] Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High Performance Computing," *IEEE Trans. Computers,* vol. 57, no. 12, pp. 1647-1660, 2008.

[18] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," Technical Report 1346, Univ. of Wisconsin-Madison Computer Science, 1997.

[19] C. Lu, "Scalable Diskless Checkpointing for Large Parallel Systems," PhD dissertation, Univ. of Illinois at Urbana-Champaign, 2005.

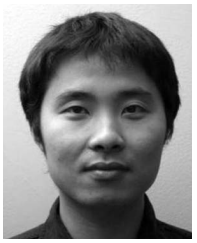
[20] A. Mu'alem and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel and Distributed System,* vol. 12, no. 6, pp. 529-543, 2001.

[21] A. Nagarajan, F. Mueller, C. Engelmann, and S. Scott, "Proactive Fault Tolerance for HPC with Xen Virtualization," *Proc. Int'l Conf. Supercomputing (ICS '07),* pp. 23-32, 2007.

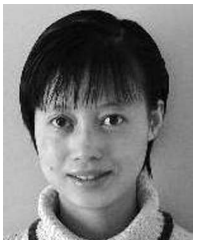
[22] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy, "Evaluation of a Workflow Scheduler Using Integrated Performance Modeling and Batch Queue Wait Time Prediction," *Proc. ACM/IEEE Conf. Supercomputing (SC),* 2006.

[23] A.J. Oliner, L. Rudolph, and R.K. Sahoo, "Cooperative Checkpointing a Robust Approach to Large-Scale Systems Reliability," *Proc. Int'l Conf. Supercomputing (ICS '06),* pp. 14-23, 2006.

[24] A. Oliner, R. Sahoo, J. Moreira, and M. Gupta, "Fault-Aware Job Scheduling for BlueGene/L Systems," *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS '04),* p. 64, 2004.
[25] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Min-max Checkpoint Placement under Incomplete Failure Information," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04),* p. 721, 2004.
[26] F. Petrini, "Scaling to Thousands of Processors with Buffered Coscheduling," *Proc. Scaling to New Height Workshop,* 2002.
[27] J. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under Unix," *Proc. Usenix,* 1995.
[28] J. Plank, K. Li, and M. Puening, "Diskless Checkpointing," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 10, pp. 972-986, 1998.
[29] J. Plank and M. Thomason, "Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 11, pp. 1570-1590, 2001.
[30] D. Reed, C. Lu, and C. Mendes, "Big Systems and Big Reliability Challenges," *Proc. Parallel Computing (ParCo '03),* pp. 729-736, 2003.
[31] R. Sahoo, A. Oliner et al., "Critical Event Prediction for Proactive Management in Large-Scale Computer Clusters," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDDM '03),* pp. 426-435, 2003.
[32] B. Schroeder and G. Gibson, "A Large Scale Study of Failures in High-Performance-Computing Systems," *Proc. Int'l Symp. Dependable Systems and Networks (DSN),* 2006.
[33] M. Schulz, G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill, "Implementation and Evaluation of a Scalable Application Level Checkpoint-Recovery Scheme for MPI Programs," *Proc. ACM/IEEE Conf. Supercomputing (SC '04),* p. 38, 2004.
[34] S. Shatz, J. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Trans. Computers,* vol. 41, no. 9, pp. 1156-1168, 1992.
[35] R. Smith and D. Dietrich, "The Bathtub Curve: An Alternative Explanation," *Proc. Ann. Reliability and Maintainability Symp. (RAMS '94),* pp. 241-247, 1994.
[36] J. Smith, "A Survey of Process Migration Mechanisms," *Operating Systems Rev.,* vol. 22, no. 3, pp. 102-106, 1988.
[37] J. Squyres and A. Lumsdaine, "A Component Architecture for LAM/MPI," *Proc. 10th European PVM/MPI Users' Group Meeting (Euro PVM/MPI),* 2003.
[38] S. Srinivasan and N. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 3, 1999.
[39] D. Tsafrir, Y. Etsion, and D. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 6, 2007.
[40] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains," *Proc. IEEE Int'l Conf. Data Mining (ICDM),* 2002.
[41] L. Wang, K. Pattabiraman, L. Votta, A.C. Vick, Z. Wood, and R. Kalbarczyk, "Modeling Coordinated Checkpointing for Large-Scale Supercomputers," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '05),* pp. 812-821, 2005.
[42] J. Young, "A First Order Approximation to the Optimal Checkpoint Interval," *ACM Comm.,* vol. 17, no. 9, pp. 530-531, 1974.
[43] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo, "Performance Implications of Failures in Large-Scale Cluster Scheduling," *Proc. Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '04),* pp. 233-252, 2004.
[44] Z. Zheng, Y. Li, and Z. Lan, "Anomaly Localization in Large-Scale Clusters," *Proc. IEEE Int'l Conf. Cluster Computing (Cluster),* 2007.
[45] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2007.
[46] M. Morris, "Kiviat Graphs: Conventions and Figures of Merit," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 3, no. 3, 1974.
[47] S. Fu and C.Z. Xu, "Exploring Event Correlation for Failure Prediction in Coalitions of Clusters," *Proc. ACM/IEEE Conf. Supercomputing (SC),* 2007.
[48] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B. Park, "Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2008.



Yawei Li received the BS and MS degrees from the University of Electronic Science and Technology of China in 1999 and 2002, respectively. He is currently a PhD student of computer science in the Department of Computer Science, Illinois Institute of Technology, Chicago. He specializes in parallel and distributed computing, and scalable software systems. His current research interests include adaptive fault management in large-scale computer systems, checkpointing optimization, and load balancing in Grid environment. He is a student member of the IEEE.



Zhiling Lan received the BS degree in mathematics from Beijing Normal University, the MS degree in applied mathematics from the Chinese Academy of Sciences, and the PhD degree in computer engineering from Northwestern University in 2002. She is currently an assistant professor in the Department of Computer Science, Illinois Institute of Technology, Chicago. Her main research interests include fault-tolerant computing, dynamic load balancing, and performance analysis and modeling. She is a member of the IEEE Computer Society.

Prashasta Gujrati received the BTech degree from Banaras Hindu University. He is currently a master's student in the Department of Computer Science, Illinois Institute of Technology, Chicago. His current research interests include fault-tolerance in parallel/distributed systems, data mining, and pattern recognition. He is a member of the IEEE Computer Society.

Xian-He Sun is a professor of computer science in the Department of Computer Science, Illinois Institute of Technology (IIT), Chicago, a guest faculty member at the Argonne National Laboratory and the Fermi National Accelerator Laboratory, and the director of the Scalable Computing Software (SCS) laboratory at IIT. Before joining IIT, he was a postdoctoral researcher at the Ames National Laboratory, a staff scientist at ICASE, NASA Langley Research Center, an ASEE fellow at the Naval Research Laboratory, and an associate professor at Louisiana State University-Baton Rouge. He is a senior member of the IEEE Computer Society.