In the Proc. of International Conference on Parallel Processing (ICPP'07)

Fault-Driven Re-Scheduling For Improving System-level Fault Resilience

Yawei Li[*], Prashasta Gujrati[*], Zhiling Lan[*], Xian-he Sun[*#]

*Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
{liyawei,gujrpra,lan,sun}@iit.edu

#Computing Division
Fermi National Accelerator Laboratory
Batavia, IL 60510-0500

Abstract

The productivity of HPC system is determined not only by their performance, but also by their reliability. The conventional method to limit the impact of failures is checkpointing. However, existing research shows that such a reactive fault tolerance approach can only improve system productivity marginally. Leveraging the recent progress made in the field of failure prediction, we propose fault-driven rescheduling (FARS) to improve system resilience to failures, and investigate the feasibility and effectiveness of utilizing failure prediction to dynamically adjust the placement of active jobs (e.g. running jobs) in response to failure prediction. In particular, a rescheduling algorithm is designed to enable effective job adjustment by evaluating performance impact of potential failures and rescheduling on user jobs. The proposed FARS complements existing research on fault-aware scheduling by allowing user jobs to avoid imminent failures at runtime. We evaluate FARS by using actual workloads and failure events collected from production HPC systems. Our preliminary results show the potential of FARS on improving system resilience to failures.

1. Introduction

As the scale of high performance computing (HPC) keeps increasing, reliability is becoming more of an issue due to the decreasing system mean time before failure (MTBF) value. Failures can lead to an intolerable loss of system performance, in addition to substantial maintenance cost. To limit failure impact on system productivity, checkpointing has been widely used, in which system snapshots are periodically saved and stored. Nevertheless, existing research has shown that checkpointing can cause severe performance degradation if used too frequently. Moreover, such a reactive approach suffers from non-trivial recovery cost and operational cost [19,24]. Hence, a new fault tolerant approach is needed to improve system resilience to failures in HPC.

Recently, much progress has been made in failure prediction by using various hardware or software based technologies. Given that systems usually present certain symptoms before the occurrence of a failure, short-term prediction emphasizes on learning and discovering pre-failure symptoms for failure forecasting in the near future (e.g. in the order of minutes) [8]. Typical examples include the warnings produced by hardware sensors [1,12,16] regarding potential hardware problems or by software-based predictive methods using data mining and machine learning techniques [2,10,29].Considerable research has been conducted on fault-aware scheduling [4,22,24,28,30]. This research mainly focus on intelligent job allocation based on global failure distribution functions such as exponential, Weibull, or other long-term probabilities, rather than utilizing short-term fault prediction at runtime.

Leveraging the research on short-term fault prediction, in this paper we *design and study a fault-driven rescheduling mechanism denoted as FARS*. It aims at enhancing system resilience to failures by preemptively transferring running processes from failure-prone nodes to healthy ones. In particular, this paper intends to answer two questions. The first question is, "*Given the limited number of spare nodes available, which processes should be migrated in case of multiple simultaneous failures*?" Process migration or rescheduling introduces operational overhead, prediction is imperfect, and jobs usually have different computational characteristics. Therefore, the movement of processes belonging to different jobs can result in different system performance. By considering these factors, we develop performance models to quantitatively evaluate the impact of failures and rescheduling on user jobs. Based on the models, we then design a heuristic rescheduling scheme to reschedule active jobs in response to failure prediction. The next key question is, "*How much performance*

gain can we achieve by using fault-driven rescheduling?" To gain realistic insight into the effectiveness of FARS, we conduct extensive trace-based simulation by using actual workloads and failure events collected from production HPC systems. Preliminary results indicate that FARS can improve system productivity by more than 10% in average, even with modest prediction accuracy and a conservative reservation of spare nodes.

A key feature that distinguishes FARS from existing fault-aware scheduling work is that FARS focuses on dynamically adjusting the placement of *active jobs* (i.e. the jobs that are already scheduled and running) in the presence of imminent failures. In other words, FARS complements fault-aware scheduling by allowing active jobs to avoid anticipated failures at runtime. Different from our previous work on application-level fault management that aims at reducing the completion time of a given application[15], the proposed FARS is at system-level, meaning that the primary objective is to improve system productivity such as system utilization rate, average job response time, job failure rate. Specifically, the paper makes the following major contributions:

- Design a fault-driven rescheduling (FARS) scheme to improve system productivity in the presence of failures by leveraging the research on short-term fault prediction.
- Develop an event-based simulator which incorporates the proposed rescheduler into the widely used batch scheduler by considering a variety of system recovery strategies.
- Demonstrate the potential of FARS to improve system resilience to failures by using actual workloads and failure events from HPC systems.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 classifies job recovery model in HPC. Section 4 presents the proposed fault-driven rescheduling scheme. Section 5 presents our evaluation methodology and experiment results. Finally, Section 6 discusses our future work.

2. Related Work

Failure Prediction. Modern hardware devices are deployed with various features (e.g. hardware sensors) that can monitor the degradation of an attribute over time for early failure detection [1, 12, 16]. On the other side, software-based fault prediction is generally approached from two different angles: model-based or data mining based [2,8,10,27,29]. A model-based approach derives an analytical or probabilistic model of the system. A warning is triggered when a deviation from the model is detected. Data mining, in combination with intelligent systems, focuses on learning and classifying known faults without constructing a model ahead of time. According to the research literature, modern predictors can detect 60% to 80% of failures, with the false alarm rate of less than 35%. By leveraging the research on failure prediction, we focus on design fault-driven rescheduling to avoid anticipated failures in this paper.

Process Migration. Intensive research has been done on process migration [17]. Process migration can be performed at either the kernel-level or the user-level. Kernel-level migration requires the modification of operating systems, while user-level migration enables process migration without changing the operating system. For instance, Condor allows user-level process migration by using checkpointing [18]. Existing works on process migration for parallel applications (e.g. MPI applications) are mainly based on the stop-and-restart model, where all the application processes stop and then restart on a new set of resources (e.g. swapping the failure-imminent nodes with healthy ones). This is mandated due to the static nature of MPI communicators. Instead of applying the stop-and-restart mode, the HPCM (High Performance Computing Mobility) system allows live migration of parallel applications [5]. The proposed FARS can utilize HPCM for the support of rescheduling processes.

Fault-aware Scheduling. There are a number of research efforts on fault-aware scheduling or resource management [4]. For example, Shatz et al. propose a task graph based performance model to maximize a reliability cost function, and develop several heuristic algorithms based on this model [28]. The authors in [22] develop a failure-aware job scheduling algorithm for the Blue Gene/L system. It exploits node failure probabilities for intelligent job allocation on Blue Gene/L. Xiao and Hong propose a reliability-cost oriented dynamic scheduling heuristic for real-time jobs running in heterogeneous clusters [26]. In HA-OSCAR [13], a cost-effective recovery policy is proposed for head nodes in Linux clusters. This work is based on hot-standby replications by using failure prediction. Ching-chih et al. propose a fault tolerant algorithm to deal with deadline failures by invoking an alternative execution of the job when deadline violations are upcoming [9]. Different from these works that focus on an optimal initial job placement or resource replication, our research emphasizes on dynamically adjusting the placement of *active jobs* (i.e. running jobs) in response to failure prediction. The proposed FARS can be easily integrated with existing

fault-aware schedulers to further improve system resilience to failures on large-scale clusters.

There are a few research projects studying job rescheduling on parallel and distributed computing systems. For example, the GrADS project [3] proposes an application-level job migration and processor swapping approach are presented to reschedule a Grid application when a better resource is found. To our best knowledge, there is no prior work on exploiting failure prediction in job rescheduling to improve the system resilience to failures.

3. Failure Recovery Model

According to the job schedulers that are widely used in HPC [13,14,20,29], we categorize existing system-level failure recovery policies into three categories:

RETRY: The system attempts to re-run the failed job on the original set of resources. In this policy, the failed job suffers from the failure downtime.

RESUME: The system attempts to resume the job immediately if alternative resources are available. Otherwise, the failed jobs are put into a special resume queue, waiting for computing resources. In this policy, the failed job is restarted as early as possible, but its recovery may cause delay of other jobs in the regular job queue.

RESUBMIT: The system automatically resubmits the aborted job at the end of the regular job queue. In this policy, the failed job is regarded as a newly arriving job and will experience the queue waiting time before its restart.

4. FARS: FAult-driven ReScheduling

4.1 Failure Prediction Model

We assume that failures follow the fail-stop model and that a fault prediction mechanism is available in the system to forecast potential failures. Failure prediction is out of the scope of this paper, and interested readers can refer to the literature listed in Section 2 for more details. Predictive techniques generally fall into two categories according to the nature of predictive results:

- Categorical strategies, where a categorical value is predicted to determine whether a failure event will occur in the near future or not.
- Numerical strategies, where a numerical value is predicted to evaluate the probability of failures in a given time window.

Given that numerical-value results can be mapped into categorical values by discretization, we uniformly describe a failure prediction as a process that forecasts whether or not a node will experience failures in a given time window. Such a predictor is generally measured by two accuracy metrics: the false-positive rate and the false-negative rate. Here, the false-positive rate f_p reflects the precision of the failure predictor, which is defined as $f_p = \frac{false\ positives}{false\ positives + true\ positives}$, The false-negative rate f_n reflects the sensitivity of the predictor, which is defined as $f_n = \frac{false\ negatives}{false\ negatives + true\ positives}$.

In FARS, we denote the node with positive prediction as *a suspected node*, the process and the job running on the node as *suspected process* and *suspected job*.

4.2 Main Idea

FARS quantitatively evaluates failure implications on system performance and then re-allocates a set of suspected jobs in response to failure prediction. It divides the nodes in a system into two groups: *working nodes* for regular job scheduling and *spare nodes* for rescheduling. The use of spare nodes is to guarantee the availability of resources required by preemptive migration in the presence of failures. Currently, we statically allocate spare nodes according to the historic node usage of the system. Our study shows that even for highly loaded systems, the allocation of spare nodes in FARS is feasible in practice (see Section 5).

FARS works with a regular job scheduler in a cooperative way. A user job is submitted to the job scheduler, such as the first-come first-server (FCFS) batch scheduler using backfilling [21], and starts the execution on the allocated working node. FARS is triggered at pre-defined points (denoted as *decision points*) inserted by the system. Upon each invocation, based on failure prediction, FARS first identifies suspected working nodes and available spare nodes. It then launches checkpointing or rescheduling depending on failure prediction: (1) for the processes residing on failure-prone working nodes, FARS transfers a selected set of suspected processes based on the algorithm discussed in the next subsection; (2) for other processes, FARS conducts checkpointing. After rescheduling, the suspected nodes become spare nodes and stand by, while the swapped spare nodes serve as working nodes. If any failure occurs during job rescheduling, we regard the affected job as being failed.

4.3 Rescheduling Algorithm

Since multiple failures may occur simultaneously, it is possible that at a decision point, multiple user processes from different jobs will be affected by failures. As the number of available spare nodes is limited, contentions exist among the suspected jobs for the spare nodes. To determine which suspected processes should be migrated, an iterative greedy algorithm is adopted, with the objective to maximize the rescheduling gain based on performance impact models. In the following, we first describe the performance model and then present our rescheduling algorithm.

According to the failure model discussed in Section 3, the implication of failures on a specific job can be divided into four parts: (1) *workloss*, the amount of volatile work lost due to the failure; (2) *restart cost*, the time required to restart the job during recovery; (3) *failure downtime*, the time used to repair the failed node; and (4) *requeue cost*, the queue waiting time of the job that is caused by the failure.

Our performance models estimate the rescheduling gain *G(J,K)* of migrating the suspected processes belonging to suspected job *J* using *K* healthy spare nodes as follows:

Step 1: Calculate the failure probability of job J as $F_J = \begin{cases} 1 - f_p^{(S^J - K)} & if \ \ S^J > K \\ 0 & if \ \ S^J \le K \end{cases}$ where S^J denotes the number of suspected processes of job J and f_p is the false positive rate of failure prediction.

Step 2: Calculate failure impact on job J if F_J is greater than 0:

- Calculate the aggregated workloss of the job as $C_{workloss} = (|J| \times \min(I, remaining\ work\ of\ J))$, where $|J|$ is the size of the job and I is the decision interval. Without the precise knowledge of failure occurrence time, we pessimistically assume that the job will fail right before the next decision point.
- Calculate the job restart cost as $C_{restart}(J)$, which can be estimated according to the latest job (re)start cost.
- Calculate the downtime cost as $C_{downtime} = \left(\sum_{i=1}^{S^J - K} DT(N_i^J) \right) \Big/ \left(S^J - K \right)$ where N_i^J denotes the node where the *i*th suspected process of job *J* resides and $DT(N_i^J)$ is the estimated node downtime. This formula calculates the average downtime over all the remaining suspect processes that cannot be migrated. It is applied to the RETRY policy and the RESUME policy when alternative resource is not available.
- Calculate the requeue cost C_{queue}, which is the queue waiting time. It can be estimated by historical data. This cost is applied to the RESUBMIT policy.
- Calculate the expected performance loss as $E(J,K) = F_J \times (C_{workloss} + C_{restart} + C_{downtime} + C_{queue})$

Step 3: Calculate the rescheduling overhead O_J as $C_{restart}$ since we regard a job rescheduling as a proactive restart immediately after a checkpointing.

Step 4: Calculate the rescheduling gain of migrating K suspected processes of job J: $G(J,K) = E(J,0) - E(J,K) - O_J$, which is the performance difference with and without job rescheduling.

Based on the above performance impact models, we develop a rescheduling algorithm (see Figure 1). Here, *K* denotes the number of total available spare nodes, M_{job} denotes the set of suspected jobs. The rationale of this iterative algorithm is to identify a suspected process i^* of a job *J** for each available spare node such that the rescheduling of the process i^* can lead to a maximum performance gain in an iterative style. A max-heap data structure is used for efficient maximum-selection operation in line 2 in figure 1. The computational complexity of the algorithm is $O(K \cdot Log(\sum_{j \in M_{job}} S^J > 0))$.

FARS (*K*, M_{job})

1. For each suspected job *J*, calculate the *G(J,1)*,which is the rescheduling gain if we assign one spare node to job *J*
2. Store the J and its rescheduling gain G(*J*,*1*) in a max-heap based priority-queue.
3. **while** (*K*>0 and $\sum_{j \in M_{job}} S^J$)
4. Select the job J^* having the maximum gain $G(J^*,1)$, assign one spare node to *J**
5. Inside the job J^*, select the suspected process *i** running on the node $N_{i^*}^{J^*}$, which has the maximum downtime $DT(N_{i^*}^{J^*})$
6. Remove process i^* from the suspected processes list of job J^*, then update gain $G(J^*,1)$.
7. $S^{J^*} = S^{J^*} - 1;\ K = K - 1;$
8. **end while**

Figure 1. The FARS rescheduling algorithm

5. Performance Evaluation

5.1. Methodology

A discrete event-based simulator is developed to evaluate FARS. It integrates the proposed FARS with the widely used FCFS/EASY Backfilling scheduler [21]. We compare FCFS/EASY Backfilling with FARS rescheduling with against FCFS/EASY Backfilling with periodic checkpointing. We simply use the term FCFS and FARS to denote these two strategies for the rest of the paper.

In order to gain practical insight, the simulator uses the job log from a 128-node IBM SP2 machine at SDSC and a failure log collected from a production system at NCSA [19]. The job log contains 59,725 parallel jobs. In the failure log, the system-wide MTBF is 0.79 hour and node-wide MTBF is 14.2 days.

Six performance metrics are used for evaluation: (1) *response time Resp*; (2) *utilization rate Util*; (3) *throughput Thru*; (4) *workloss*, the service unit loss due to failures; (5) *job failure rate Frate*, the ratio between the number of failed jobs and the total number of jobs; 6) *migration cost*, the cost introduced by process migration (i.e. rescheduling) per job.

5.2. Results Under Default Setting

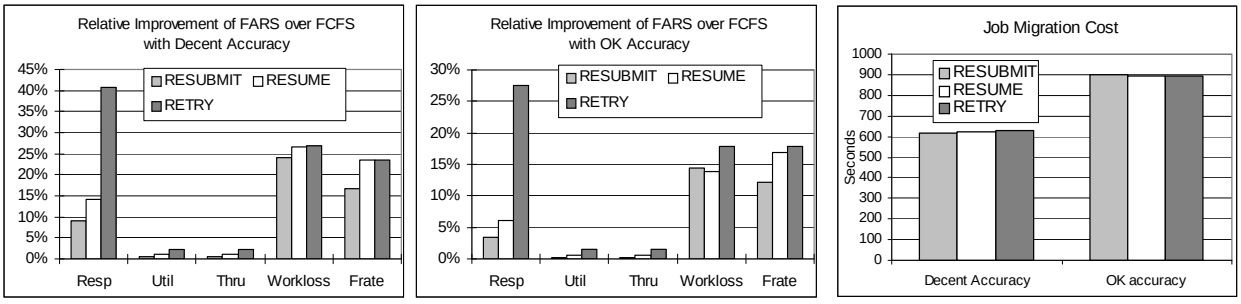
The default runtime parameters are listed in Table 1. These parameters and their corresponding ranges are selected based on the results reported in [15,19,22,30].

Table 1. Default parameter setting

Reschedule Interval	3 hours
Checkpointing Cost	3 mins
Rescheduling overhead	6 mins
Decent Prediction(f_p,f_n)	(0.3,0.3)
OK Prediction (f_p,f_n)	(0.5,0.5)
Number of Spare Nodes	2

We select two prediction settings: (1) *decent prediction* to represent the prediction with a high accuracy; and (2) *OK prediction* to represent the prediction with a low accuracy. Conservatively, we only reserve two nodes out of 128 nodes (less than 2%) as spare nodes.

The relative performance improvement of FARS over FCFS is presented in Figure 2. The relative improvement in terms of *response time* is 8%, 14.7% and 41% under three failure recovery policies respectively by using FARS with decent prediction; with OK prediction, the relative improvements are 3.5%, 6.7% and 27% respectively. FARS provides relatively smaller improvement (ranging from 0.5% to 2.3%) in terms of *utilization rate* and *throughput*. This is due to the fact that both metrics are mainly determined by the system usage, rather than by fault tolerance strategies. Figure 2 also indicates that FARS provides significant improvement in terms of reliability-related metrics (i.e. *workloss and job failure rate*). With decent prediction, 24%-26% improvement in workloss and 16%-24% improvement in job failure rate are observed for three recovery policies. With OK prediction, 14%-28% improvement in workloss and 12%-18% improvement in job failure rate are observed. This demonstrates the effectiveness of FARS in improving system resilience to failures. The overhead is estimated as the job restart cost, including both I/O and communication cost. Figure 2(c) lists the amount of overhead (in seconds) incurred by rescheduling. For decent and OK prediction, the average costs are around 600.0 and 900.0 seconds respectively. Given that the average job response time is around 1.0E+05 seconds, the overhead accounts for less than 1% of the total elapsed time.



(a) With Decent prediction accuracy (b) With OK prediction accuracy (c) Migration costs

Figure 2. Performance comparison of FARS versus FCFS under default setting

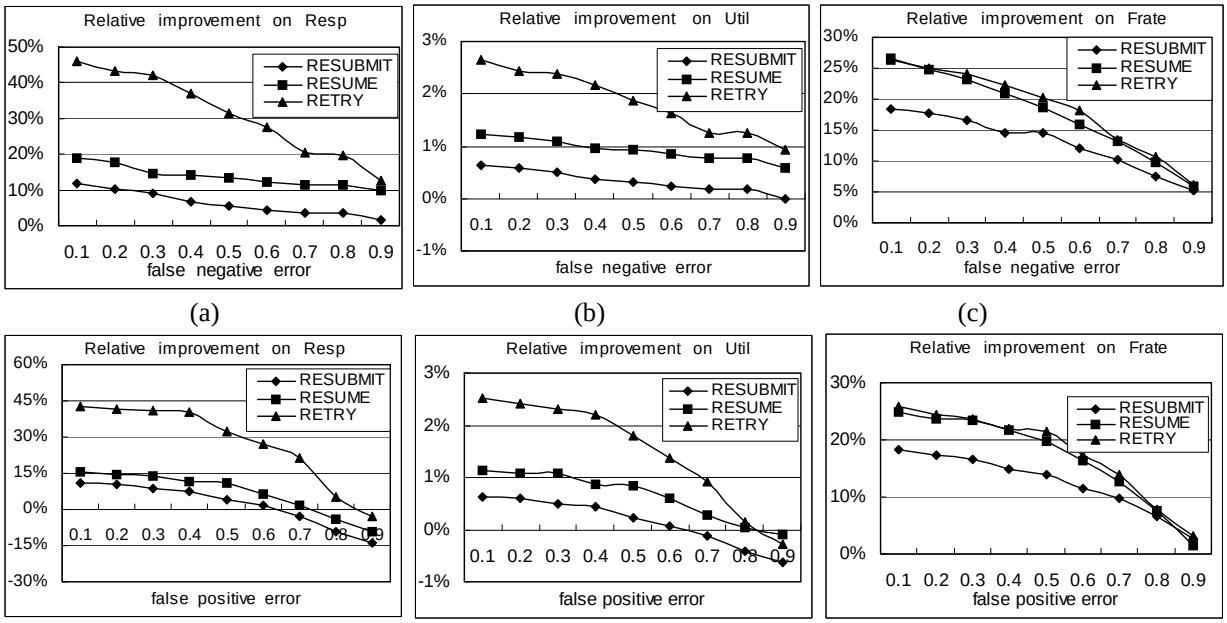


Figure 3. Relative performance gain with varying prediction accuracy

5.3. Impact of Prediction Accuracy

In this set of experiments, we investigate the sensitivity of FARS to predication accuracy. The default parameter setting is used, except that we change the values for false positive and false negative. The performance results are presented in Figure 3: (1) in Figure 3(a)-(c), the false negative f_n changes from 0.1 to 0.9 with a fixed f_p (=0.3), and (2) in Figure 3(d)-(f), the false negative f_p changes from 0.1 to 0.9 with a fixed f_n (=0.3). Due to the space limitation, we only present three performance metrics, namely average response time, system utilization rate and job failure rate.

From Figure 3(a)-(c), we can see that FARS maintains its performance gain over the regular FCFS, even with modest prediction accuracy in terms of false negative errors. With increasing values of f_n, the relative performance gains gradually approach zero. When f_n increases, it is getting unlikely for the failure predictor to detect failures. As a consequence, the effectiveness of FARS is decreasing. From Figure 3(d)-(f), we notice that the performance gain achieved by FARS drops sharply with increasing false positive rate. In other words, the performance impact caused by f_p is more pronounced than that caused by f_n. For example, with RESUME policy, the relative gain of average response time decreases from 18% to 1% when f_p increases from 0.1 to 0.65 and then becomes negative when f_p increases beyond 0.65. Similar trends are observed with other performance metrics as well. A high value of f_p means that the failure predictor produces many false alarms, thereby causing negative performance gain of FARS. There are two major reasons for this. First, false alarms may cause contention for the limited amount of spare nodes, thereby reducing the chance for FARS to find backup resources for those true failure predictions. Second, due to the high false alarm rate, unnecessary rescheduling may be incurred, which can cause non-trivial overhead to system performance. Nevertheless, from these figures, we can still state that as long as the false positive rate f_p is smaller than 0.65, FARS always outperforms the regular FCFS.

5.4. Impact of Spare Nodes

In this set of experiments, we change the number of spare nodes from 1 to 8 in the default parameter setting. Figure 4(a) – (c) shows the curves of five performance metrics under different system recovery policies. The overall performance trend shows small variations. For each system recovery policy, we observe that an optimal number of spare nodes exist. For example, it is *1* for RETRY policy, *2* for RESUME policy and *5* for RESUBMIT policy. Although the optimal value generally depends on the system configuration and usage, the experiments suggest that we can always conservatively reserve a modest number of spare nodes (e.g. less than 5% of the overall computing resources) for fault management to improve system productivity. For instance, even with only one

spare node (less then 1% out of 128 nodes), the performance gain for *response time* achieved by FARS is around 42% for RETRY policy, 14.7% for RESUME policy, and 9% for RESUBMIT policy.

To investigate the feasibility of spare node reservation, we have studied the distribution of idle nodes on a number of production systems [7]. The probability mass functions (PMF) of idle nodes are plotted in Figure 5. Here, we present the results of two systems: the 128-node SDSC-SP2 system with 84% utilization rate and the 400-node SDSC-Paragon system with 71% utilization rate. They are selected to represent heavily and moderately loaded systems. The results indicate that most production systems generally have a couple of unused nodes during the operation. For example, the chances that there are more than 2% of idle nodes are 73% and 99% on SDSC-SP2 and SDSC-Paragon respectively. We have found similar patterns on other production systems. This observation indicates that it is feasible to reserve a few nodes as spare nodes for fault-driven rescheduling used in FARS.

6. Summary

In this paper, we have presented a fault-driven rescheduling (FARS) for improving system resilience to failures, and investigated the effectiveness of utilizing failure prediction to dynamically adjust the placement of active jobs. In particular, we have designed a rescheduling algorithm based on quantitative performance models. Extensive trace-based simulations with actual workload and failure events were conducted to study the performance of the proposed FARS under a wide range of prediction accuracies and system parameters. Our preliminary results indicate that FARS is valuable and has the potential to significantly improve system resilience to failures.

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 large-scale systems to further evaluate the effectiveness of the proposed FARS. Second, we plan to integrate FARS with existing scheduling system[21, 23]. We expect that this combination can further improve the system performance. Lastly it is also interesting to investigate other possible rescheduling solutions and evaluating them with various batch schedulers.

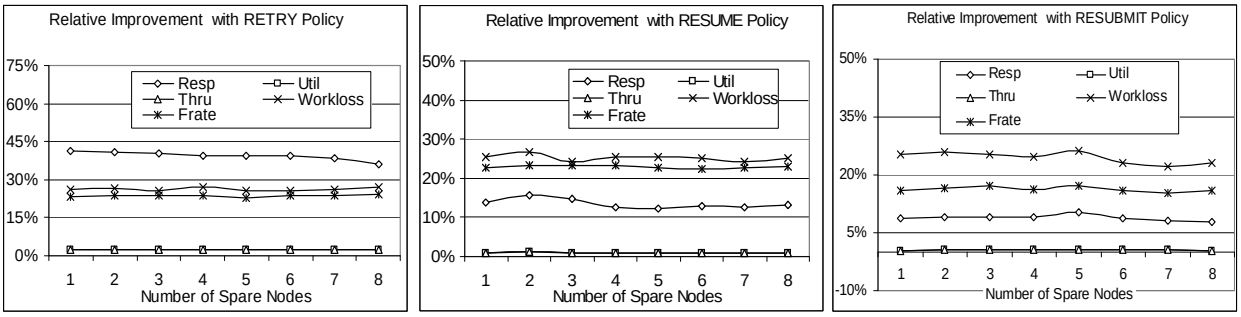


Figure 4.Relative performance gain of FARS against FCFS with varying number of spare nodes

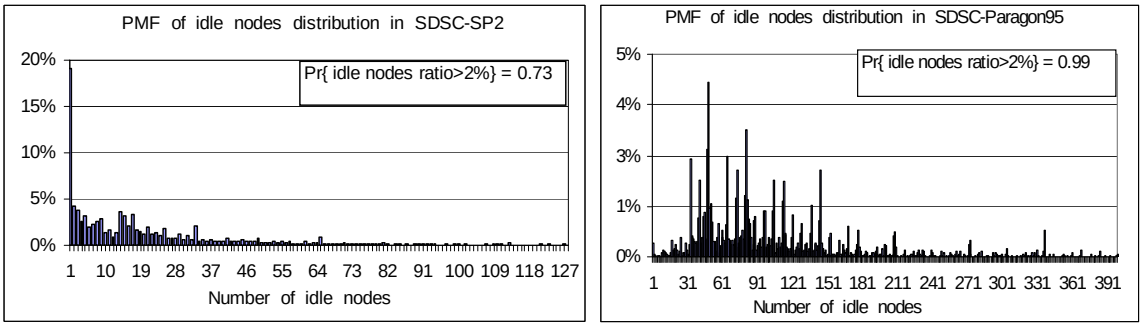


Figure 5.PMF functions of idle nodes distribution on two clusters

References

[1] B. Allen, "Monitoring Hard Disk with SMART", *Linux Journal*, January, 2004.

[2] J. Brevik, D. Nurmi, and R. Wolski, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems", *Proc. of IEEE CCGrid*, IEEE Computer Society, Chicago,IL, 2004, pp. 190-199.

[3] F. Berman, H. Casanova, et al., "New Grid Scheduling and Rescheduling Methods in the GrADS Project", *Intl. Journal of Parallel Programming*, 2005, pp. 209-229

[4] A. Dogan,F. Ozguner, "Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing,"In *Proc. of the ICPP*, IEEE Computer Society, Toronto, Canada, 2000, pp. 307

[5] Cong Du and Xian-He Sun, "MPI-Mitten: Enabling Migration Technology in MPI", in *Proc. of CCGRID*, IEEE Computer Society, Singapore, 2006, pp. 11-18

[6] Elmootazbellah N. Elnozahy and James S. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery", *IEEE Transactions on Dependable and Secure Computing*, Volume 1, No 2, 2004, pp. 97-108.

[7] D.Feitelson. Parallel Workloads Archive http://cs.huji.ac.il/labs/parallel/workload/index.html

[8] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "Exploring Meta-learning to Improve Failure Prediction in Supercomputing Clusters", in *Proc. of ICPP07*, 2007

[9] C.-C. Han, K.G. Shin, J. Wu, "A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults," *IEEE Trans. Computers*, Vol.52, No.3 pp.362- 372, 2003

[10]] G. Hoffmann, F. Salfner, M. Malek, "Advanced Failure Prediction in Complex Software Systems", in *Proc. of SRDS*, 2004

[11] K. C. Gross, R. M. Singer, S. W. Wegerich, J. P. Herzog, R. VanAlstine, and F. Bockhorst, "Application Of A Model-Based Fault Detection System To Nuclear Plant Signals", in Proc. of ISAP,Seoul,Korea, 1997, pp. 66-70

[12] Health Application Programming Interface, http://www.renci.org

[13] C. Leangsuksun et al, "A Failure Predictive and Policy-Based High Availability Strategy for Linux High Performance Computing Cluster", in *Proc. of LCI International Conference on Linux Clusters: The HPC Revolution 2004*, Austin, TX, 2004

[14]IBM LoadLeveler for AIX 5L, available at http: //publib.boulder.ibm.com

[15] Yawei Li, Zhiling Lan, "Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing", in *Proc. of IEEE CCGrid'06*, Singapore,2006,pp. 531-538

[16] Hardware monitoring by lm sensors, available at http: //secure.netroedge.com/-lm78/info.html.

[17] R. Lawrence, "A Survey of Process Migration Mechanisms",http://www.cs.uiowa.edu/~rlawrenc/research/Papers/proc_mig.pdf

[18] M. Lizkow, T. Tannenbaum, et al., "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", University of Wisconsin-Madison Computer Science Technical Report #1346, 1997.

[19] Charng-Da Lu, "Scalable Diskless Checkpointing for Large Parallel Systems", Ph.D. thesis, University of Illinois at Urbana-Champaign, 2005

[20] Moab Workload Manager, available at http://www.clusterresources.com

[21] A. Mu'alem and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling", in *IEEE Trans. Parallel and Distributed Systems*, Vol. 12(6), 2001,pp. 529-543

[22] A. Oliner, Ramendra K. Sahoo, José E. Moreira, Manish Gupta, Anand Sivasubramaniam, "Fault-Aware Job Scheduling for BlueGene/L Systems", in *Proc. of IPDPS*, 2004,

[23]Parallel Workloads Archive, available at http://www.cs.huji.ac.il/labs/parallel/workload/

[24] Petrini, F.; Davis, K.; Sancho, J.C," System-level fault-tolerance in large-scale parallel machines with buffered coscheduling", in *Proc. of IPDPS*, 2004, pp. 209

[25] S. Srinivasan, and N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," in *IEEE Trans. Parallel and Distributed Systems*, Vol 10(3), 1999, pp. 238-251

[26] X. Qin and H. Jiang, "A Dynamic and Reliability-driven Scheduling Algorithmfor Parallel Real-time Jobs on Heterogeneous Clusters," in *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, 2005, pp. 885-900.

[27] Ramendra K. Sahoo, A. Oliner, et al., "Critical Event Prediction for Proactive Management in Large-scale Computer Clusters", in *Proc. of KDD*, Washington DC, USA,2003,pp. 426-435

[28] S. Shatz, J. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems", in *IEEE Trans. on Computers*, Vol 41(9), 1992,pp. 1156 - 1168

[29] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains", in *Proc. of IEEE ICDM*, 2002, pp.474-481

[30] Y. Zhang et al., "Performance Implications of Failures in Large-Scale Cluster Scheduling", *Proc. of 10th Workshop on JSSPP, held in conjunction with SIGMETRICS* , New York, USA, 2004.