Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing

Yawei Li and Zhiling Lan
Department of Computer Science
Illinois Institute of Technology, Chicago, IL 60616
{liyawei, lan}@iit.edu

Abstract

As the scale of cluster computing grows, it is becoming hard for long-running applications to complete without facing failures on large-scale clusters. To address this issue, checkpointing/restart is widely used to provide the basic fault-tolerant functionality, yet it suffers from high overhead and its reactive characteriristic. In this work, we propose FT-Pro, an adaptive fault management mechanism that optimally chooses migration, checkpointing or no action to reduce the application execution time in the presence of failures based on the failure prediction. A cost-based evaluation model is presented for dynamic decision at run-time. Using the actual failure log from a production cluster at NCSA, we demonstrate that even with modest failure prediction accuracy, FT-Pro outperforms the traditional checkpointing/restart strategy by 13%-30% in terms of reducing the application execution time despite failures, which is a significant performance improvement for long-running applications.

1. Introduction

To meet the increasing computational demand in science and engineering, computing capacity of clusters has been increased dramatically in the past decades and various systems with hundreds to thousands of processors have been deployed. According to [3], one third of the TOP500 systems have more than 500 processors. Increasing number of processors improves the overall performance; however, it also increases the probability of hardware and software failures of the system. Furthermore, the present trends in denser integration of semiconductor circuits, lower power consumption, and utilization of COTS components also lead to increasing occurrence of system failures [20]. In the meantime, many scientific applications are designed to run for days, weeks, or longer until completion. Therefore, it becomes difficult or impossible for long-running simulations to complete without facing failures on large-scale clusters, and this is becoming a major performance impediment for cluster computing due to the work loss.

Checkpoint/recovery (CPR) is a commonly used technique for fault management in cluster computing, and a wealth of literatures is available covering a wide range of issues related to checkpointing/recovery [6][7][11][14][16][17][25]. Among them, how to reduce the cost associated with checkpointing has been studied extensively. Much work has been done on selecting the optimal checkpointing intervals or reducing the overhead per checkpointing action so as to reduce the checkpointing overhead. In fact, a major issue with CPR is that it only deals with failures after their occurrence through the rollback approach. When one of the parallel processes experiences a failure, other processes have to restart from the most recently saved state prior to the failure. Thus, significant performance loss can be incurred due to the non-trivial failure downtime and recovery cost.

In the past decades, much progress has been made in the field of failure prediction. For instance, modern hardware devices are designed with various features (e.g. hardware sensors) that can monitor the degradation of an attribute over time for early failure detection [1][2][5], and a number of statistical and machine learning based prediction techniques have been presented with up to a considerable accuracy [8][9][18][19][20][22][23][24]. Proactive fault tolerance techniques based on failure prediction have been adopted to achieve high availability for safety-critical applications [4][15], but not well studied for cluster computing [5].

This paper fills this gap by investigating and evaluating FT-Pro, an adaptive fault management approach that exploits failure prediction. Here, the "adaptive" means that the proposed scheme dynamically chooses a preventive action, e.g. proactive process migration or reactive checkpointing, during the execution of parallel applications. A cost-based evaluation algorithm is proposed to adaptively select different preventive actions based on the accuracy of failure prediction. Further, a skip-window strategy is employed to prevent severe impact brought by the false-negative error of failure prediction. In FT-Pro, the primary goal is to reduce unnecessary checkpoints and avoid failures so

as to minimize the completion times of parallel applications.

We study the impact of failure prediction on the performance gain achieved by FT-Pro as compared to the traditional CPR. More specifically, we compare the performance of FT-Pro against periodic checkpointing under a wide range of prediction accuracies, problem sizes, and system parameters. Experiments show that FT-Pro outperforms periodic checkpointing by about 13% to 30% with the false-negative error and false-positive error ranging from 0.1 to 0.9 respectively. The paper makes the following contributions:

- Instead of solely relying on checkpointing for fault tolerance, the proposed FT-Pro utilizes a cost-based evaluation algorithm to innovatively integrates proactive process migration with reactive checkpointing;
- Unlike most fault-tolerant research works that require the knowledge of global failure distribution [6][7][14][25], the proposed FT-Pro makes use of local failure event prediction at each node;
- Instead of optimizing the system availability, FT-Pro targets on parallel applications running on large-scale clusters with the objective of reducing the execution times despite failures;
- We demonstrate that the proposed adaptive fault management scheme can be effective even with modest prediction accuracy.

The rest of the paper is organized as below. In Section 2, we review the related research works. Next, in Section 3, the main idea of FT-Pro is presented and the cost-based evaluation model is elaborated. We show the experiments comparing FT-Pro against the traditional CPR under various conditions in Section 4, and conclude the paper in Section 5.

2. Relate Work

Failure Event Prediction. In recent years, literature indicates significant progresses in the area of failure event prediction that estimates the occurrence of failures in a short time ahead. The time weaver [23][24] adopts genetic algorithm to identify the predictive temporal failure pattern from time-series data and shows a capability to detect 63% telecommunication failure events. Instead of depending on equal spaced time-series data, in [19][20][22] the researchers propose a rule-based data mining method to predict failure events based on the frequent event set before the failure event with an accuracy up to 70%. Hoffiman et al. [8] employ Markov chain and UBF function for failure forecasting. Their results show that they can achieve 82%-92% accuracy by using these methods in predicting rare failure events.

Prediction-based Fault Management. More and more works have been conducted to utilize the results obtained in failure prediction. In OSCAR, the failure event prediction is used to schedule deliberate job checkpointing and migration in the Linux Beowulf cluster [9]. In [4], the authors present a scheme that first predicts the resource exhaustion failure and proactively conducts software rejuvenation. In [21], the failure event prediction is cooperated with software rejuvenation to improve the performability of the stateful distributed systems. In [13], a fault-tolerant job scheduling algorithm is proposed for BlueGene/L system. It considers the failure state of each node in the next a few seconds and allocates jobs to reduce the total failure loss across the system. Oliner et al. [12] propose a coordinative checkpointing strategy that optimistically skips a checkpoint when no failure prediction exists in the near future to maximize the system throughput. The MEAD system [16] predicts failure occurrences in distributed CORBA environment and switches the processing to the backup server by redirecting the client requests. AMPI [5] provides a checkpointing and migration infrastructure so as to allow AMPI objects to migrate away from a node in case of an imminent failure.

3. FT-Pro Description

3.1 Nomenclatures

First we illustrate a set of nomenclatures that are frequently used in the rest of the paper (see Table 1).

3.2 Main Ideals

In the design of FT-Pro, we make several assumptions. First, we assume that an event-based failure prediction mechanism is available on each node to forecast a failure event before the failure occurrence. Second, no failure occurs during the checkpointing, recovery, or process migration. Lastly, failure events are independent. These assumptions are widely used in both the literatures and realistic systems [4][12][13][19][20][22][23][24].

The performance of an event-based failure predictor is generally measured by two errors: the *false-positive* and *false-negative* error, which depends on the prediction window. Here, the false-positive error *fp* reflects the precision of the failure predictor, which is defined as $fp = \frac{false\ positive}{false\ positive + true\ positive}$. A smaller *fp* indicates that the predictor reports fewer false alarms. The false-negative error *fn* reflects the accuracy of the predictor, which is defined as $fn = \frac{false\ negative}{false\ negative + true\ positive}$. A

smaller *fn* indicates that the predictor can recognize more failures.

Table 1 Nomenclature

P	Number of nodes allocated to the application
T	The fault-free running time of the application
I	The fault-tolerance interval requested by users
L_{ckp}	The index of the last checkpointing
S	Number of spare nodes
fp	False positive error of the failure predictor
fn	False negative error of the failure predictor
p_f	The failure probability of the application during the next decision interval
f	Failure type
N_f	Number of predicted failure in the next interval
C_f	The failure downtime for a specific failure type f
C_d	The estimated failure downtime of the application in the next interval
C_{cp}	Cost per checkpoint
C_{pm}	Cost per process migration
LW	Length of skip-window
E_{fp}	Expected application execution time under FT-Pro
E_{cp}	Expected application execution time under periodic checkpointing

FT-Pro adopts a cooperative approach, that is, the application programmer or user can insert fault tolerance requests denoted as *decision points* in the application and FT-Pro makes run-time decision on what type of preventive action should be taken at each decision point. More specifically, at each decision point, the predictor at each node is consulted to provide a failure forecasting between the current and the next decision points. Based on this information and the application status, FT-Pro estimates the costs for different preventive actions and invokes an appropriate action that introduces the minimal cost. The cost models will be presented in the next subsection. Currently, FT-Pro supports three different preventive actions:

- *Process Migration*: all the processes first conduct a coordinated checkpointing; and the processes running on suspicious nodes are migrated to other healthy nodes.
- *Checkpointing*: all the processes are stopped to conduct a coordinated checkpointing.
- *No action*: the application continues its execution without being interrupted.

Figure 1 illustrates the main ideal used in FT-Pro. Suppose a parallel application composed of P processes. At each decision point (denoted as DP), FT-Pro adaptively determines which action should be taken. For example, at the decision point DP1 and DP3, the checkpointing action is taken; at DP2 and DP5, "no action" is taken to skip unnecessary overhead; and at DP4, to avoid a predicted failure, process P_n is migrated to the spare node P_s.

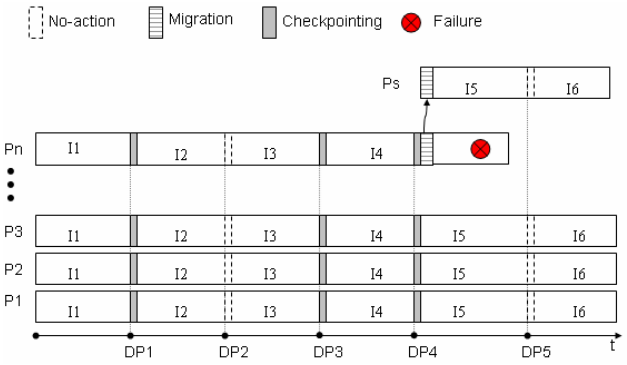

Figure 1. The FT-Pro Adaptive Fault Management

3.3 Cost-Based Evaluation Models

The primary objective in high performance cluster computing is to reduce the execution time. Hence, FT-Pro uses the execution time as the primary performance metric for the adaptive decision making. Without assuming failure distributions as done in most related work, FT-Pro adopts a greedy-style algorithm to make runtime decisions. At each decision point, FT-Pro selects the action which results in the minimum E_{next}, the expected execution time for the application from the current decision point to the next one. Through such a greedy strategy, FT-Pro aims to reduce the overall execution time of the application despites failures. In other words, the problem statement of the adaptive decision making used in FT-Pro is given as below:

> *Suppose the parallel application is executing on P processors with a fault-free execution time of T and S spare nodes are available, at each decision point, FT-Pro adaptively choose an action(e.g. process migration, checkpointing, or no action) with the objective to minimize* E_{next}.

Here, the number of spare nodes determines the maximal number of processes that can be migrated at a specific decision point simultaneously. The value of S can be designated by the user or through the negotiation with the scheduler. In case that the number of predict-to-failure nodes exceeds that of spare nodes, FT-Pro migrates the processes with more severe failures, i.e. whose predicted failures have longer downtime.

To evaluate E_{next}, FT-Pro first estimates the failure probability of the application in the next interval based on the failure prediction of each node. Suppose N_f out of P nodes have predicted failures and each has a failure type f_i ($1<i<N_f$). Thus p_f, the failure probability of the application in the next interval, can be calculated as follows:

- *Process migration*: suppose this action is selected and S (out of N_f) processes are migrated to the spare nodes. Thus We have :

$$P_f = \begin{cases} 1 - \prod_{i=1}^{N_f - S} fp \;\; if \;\; N_f > S \\ 0 \;\; if \;\; N_f \le S \end{cases} \quad (1)$$

- *Checkpointing*: suppose this action is selected. The probability of no failure for a node in the next interval is fp according to the definition of false-positive error. Therefore, we have:

$$p_f = \begin{cases} 1 - \prod_{i=1}^{N_f} fp \;\; if \;\; N_f > 0 \\ 0 \;\; if \;\; N_f = 0 \end{cases} \quad (2) .$$

- *No action*: suppose this action is selected. The failure probability of the application is the same as the case when checkpointing is chosen.

$$p_f = \begin{cases} 1 - \prod_{i=1}^{N_f} fp \;\; if \;\; N_f > 0 \\ 0 \;\; if \;\; N_f = 0 \end{cases} \quad (3)$$

FT-Pro then evaluates C_d, the failure downtime of the application in the next interval, which is calculated as the average of N_f possible failures:

- *Process migration*:

$$C_d = \begin{cases} \frac{1}{N_f - S} * \sum_{i=1}^{N_f} C_{ft(i)} \;\; if \;\; S < N_f \\ 0 \;\; if \;\; S \ge N_f \end{cases} \quad (4)$$

- *Checkpointing*: all the nodes with a predicted failure contribute to the possible downtime. Thus we have:

$$C_d = \frac{1}{N_f} * \sum_{i=1}^{N_f} C_{f(i)} \quad (5)$$

- *No action*:, the failure downtime is the same as the case of checkpointing:

$$C_d = \frac{1}{N_f} * \sum_{i=1}^{N_f} C_{f(i)} \quad (6)$$

Now we can move to calculate E_{next} that is used to make a decision at the current decision point. Due to the uncertainty of the exact failure time, we consider the pessimistic case in which the actual failure occurs immediate before the next decision point. Suppose that the index of the current decision point is L, then based on the failure probability p_f and the corresponding failure E_{next} is calculated as follows:

- *Process migration*: (1) The failure occurs in the next interval with a probability p_f. In this case, (C_{cp}+C_{pm}) is the cost of migration operation, C_d is the cost of the failure downtime, and 2I is spent on the execution and the rollback during this interval. (2) There is no failure in the next interval with a probability of (1- p_f). In this case, the application runs successfully to the next decision point with the execution cost of I and (C_{cp}+C_{pm}) for migration operation. By the law of total expectation we have:

$$E_{next} = (2I + C_d + C_{cp} + C_{pm}) * p_f + (I + C_{cp} + C_{pm}) * (1 - p_f) \quad (7)$$

- *Checkpointing:* (1) The failure occurs in the next interval with a probability p_f. In this case each process takes C_{cp} time on the checkpointing, C_d time to experience failure downtime, and 2I time for execution from the current decision point to the next one and then rollback. (2) There is no failure in the next interval with a probability of (1- p_f). In this case, each process takes C_{cp} time on the checkpointing and I time for the execution from the current decision point to the next one. Thus we have:

$$E_{next} = (2I + C_d + C_{cp}) * p_f + (I + C_{cp}) * (1 - p_f) \quad (8)$$

- *No action*: (1) The failure occurs in the next interval with a probability p_f. In this case, all the processes first spend I time for the execution and then roll back from the next decision point to the last checkpoint, L_{ckp}. (2) There is no failure in the next interval with a probability of (1- p_f). In this case, application smoothly proceeds to the next decision point. Thus we have

$$E_{next} = \left[(L - L_{ckp} + 2) * I + C_d \right] * p_f + I * (1 - p_f) \quad (9)$$

Due to the randomness of failures, currently it is impossible for a predictor to detect all the failures in advance. In general, the ratio of unpredicted failures is quantified by *fn*. In FT-Pro, if we solely depend on the failure prediction, the work loss caused by the unexpected failures could be significant when a number of "no action" decisions are taken continuously before an unpredicted failure. As shown in Figure 2, solely based on E_{next}, FT-Pro selects "no action" decisions for the consecutive decision intervals from I_{k1} to I_{kj} because no failure is alarmed during these intervals. If a false-negative prediction occurs in the interval of (I_{kj}+1), the application would end up with a significant penalty and roll a long way back to where the last checkpointing is taken. To address this issue, we employ a *skip-window* mechanism such that FT-Pro enforces a checkpointing when the number of the consecutive "no-action" intervals increases to the window size. The main purpose is to limit the loss of any false-negative failure to a tolerable range,

i.e. the length of the skip-window. Currently FT-Pro heuristically calculates the skip-window size LW as: $LW = (MTTF / I) / fn$. The rationale here is to enforce a checkpointing approximately before an un-predicted failure. MTTF is the mean time to failure value of the system, so the term (MTTF/I) denotes the average number of intervals between two failures. Hence the above formula indicates the average number of intervals between un-predicted failures considering the false negative error.

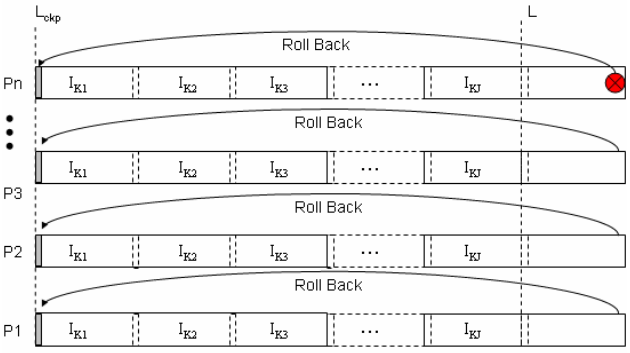

Figure 2. Adverse impact of false-negative error

In summary, the pseudo-code of the adaptive fault management mechanism is shown in Figure 3.

On each decision point L:

Step 1: Synchronize all the process.

Step 2: Inquire the predictor about the failure prediction in the next interval, obtain N_f failure alarms.

Step 3: Use formula (1) to (6) to calculate the failure probability p_f and corresponding failure downtime C_d for checkpointing, no-action and process migration actions respectively.

Step 4: Use formula (7) to (9) to calculate the E_{next}

Step 5: Choose an action with the minimum E_{next}:

On checkpointing choice, conduct a coordinated checkpointing and update the L_{ckp}: $L_{ckp} = L$

On no-action, if $L - L_{ckp} + 1 > LW$ then conduct a checkpointing, and update L_{ckp}: $L_{ckp} = L$. Otherwise do nothing

On migration, conduct a checkpointing first and update L_{ckp}: $L_{ckp}=L$, identify S nodes that have the most severe failures and migrate the processes away from these nodes.

Figure 3. Adaptive decision making algorithm

4. Experiments

In this section, we present the performance results comparing FT-Pro against the traditional periodic checkpointing strategy. We use the mean execution time as our evaluation metric, which is calculated as the average of multiple runs. For the convenience of comparison, we also calculate the relative execution time reduction of FT-Pro over periodic checkpointing:

$$relative\ reduction = \frac{E_{cp} - E_{fp}}{E_{cp}}$$

4.1 Experiment Environments and Methodology

Instead of replying on any special failure distribution, we use a real failure log of the eight-month period from the production supercomputer Platinum at NCSA [10]. Platinum consists of 520 IA-32 nodes. For every failure entry on each node, there are three associated properties: failure time, failure type and downtime. MTTF of the entire system is 0.79 hour and it is 14.2 day per node. There are three types of failures: software error, hardware error and scheduled maintenance. Table 2 summaries the failures in the system.

Table 2 Failure summary of Platinum

Failure type	Distribution percentage	Downtime (hour)
Software error	83%	0.7
Hardware error	1%	100.7
Maintenance	16%	1.2

In our experiments, the failure simulator scans the failure log of each node in the time order and simulates a failure when a real failure entry is encountered. For an application running on P nodes, if one of the P nodes encounters a failure during the execution, the application is stopped for the downtime associated with the failure. Thus the timing and distribution of failures in our experiments reflect the real failure behavior of the Platinum system.

The user application is a general MPI program, and the cost of checkpointing and migration is simulated by controllable cost. To model the behavior of failure predictor, the entire 8-month on each node is divided into multiple intervals by decision points. When inquired about the failure prediction in the next interval, the failure predictor scans the failure log of each working node and provides a failure prediction based on the false-negative error *fn* and the false-positive error *fp* as follows:

(1) *To simulate fn*: for each time interval, if a failure entry exists, the predictor reports a failure of its type with the probability of (1-*fn)*

(2) *To simulate fp*: suppose the predictor has totally reported *x* failures so far for those intervals with actual failure events. For all the other intervals without an actual failure occurrence, the predictor randomly chooses (x**fp/(1-fp))* intervals and predicts a false failure alarm for each of them. The corresponding failure type of each false-alarmed failure is randomly determined with the distribution percentage shown in Table 2.

4.2 Execution Times

First, we demonstrate the performance gain achieved by the proposed FT-Pro as compared to the traditional periodic checkpointing under various conditions. Based on real system scenario [10][11], we use the time unit and set the parameters as follows: C_{cp} = 0.05 and C_{pm} = 0.05. The default interval of the decision point, i.e. user requested checkpointing is set according to the well-known equation $I = \sqrt{2C_{cp} * MTTF}$ [25]. The accuracy of the predictor is controlled by: fp=0.2 and fn=0.3. Here, we conservatively choose S = 1, allowing only one process migration at each decision point.

Figure 4 shows the execution times on two processors in which the fault-free application execution time T is ranging from 100 to 5000 time units. Figure 5 illustrates the relative improvement of FT-Pro against the periodic checkpointing mechanism. As seen from these figures, FT-Pro always maintains a steady performance gain (between 13.6% and 14.3%) for different values of T, which is a significant improvement for parallel applications, especially those long-running applications.

We also evaluate FT-Pro against the periodic checkpointing under various numbers of processors. Figure 6 presents the relative improvement of FT-Pro with the number of processor ranging from 4 to 128. The figure shows that FT-Pro has a good scalability: as the number of processors P increases and the fault-free application execution time T increases, the relative reduction steadily increases to a convergent point, nearly 20%. We also notice that in the case that T= 500 and P=64 or 128, the relative reduction has an abrupt increase. This is due to the fact that according to the failure log, several nodes suffer a long-time downtime for four consecutive days. When P increases, the probability that these nods are allocated to the application is increased and the consequent failure downtime makes the execution time under traditional checkpointing much longer as compared to its relative shorter T.

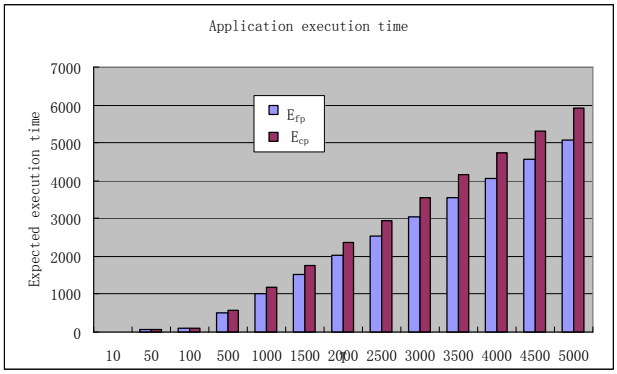

Figure 4. Application expected running time under FT-Pro and Checkpointing (P=2)

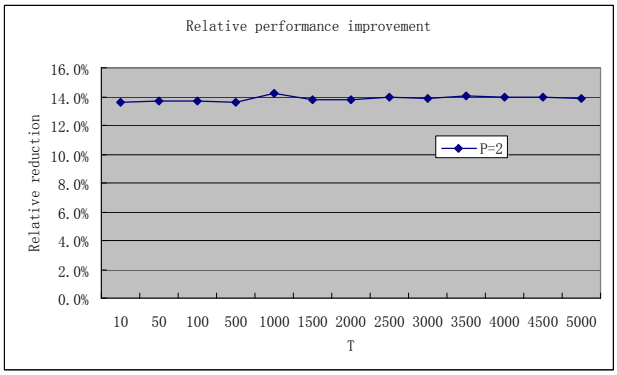

Figure 5. Relative performance improvement for P=2

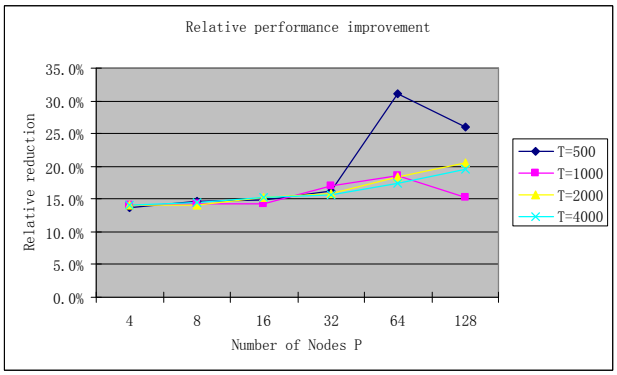

Figure 6. Relative performance improvement on different number of processors

4.3 Impact of Prediction Accuracy

In this set of experiments, we study the sensitivity of FT-Pro to the prediction accuracy. In Figure 7, we examine the performance of FT-Pro with different T values on 128 nodes where the *fp* value varies from 0.1 to 0.9 with a fixed *fn* = 0.3. It is observed that the performance variance incurred by the changes of *fp* is trivial. As *fp* increases to 0.9, the relative improvement of FT-Pro is always around 20%. This is because that the value of *fp* only increases the possibility of checkpointing and migration decision. Unnecessary checkpointing and migrations incurred by false-positive errors have lower overhead compared to the benefit of FT-Pro.

Under the same setting, we evaluate the impact of *fn* on performance of FT-Pro with a fixed *fp* = 0.2. The result is illustrated in Figure 8. As shown in the figure, the relative reduction for each T decreases for about 3%. This degradation is more apparent than that caused by *fp*. The increasing value of *fn* indicates an increasing miss rate of predicted failures. Hence, the result shows that in general the work loss caused by unpredicted failures is larger than the overhead introduced by unnecessary checkpointing or migration actions. In the mean time, the impact of the

skip-window is investigated as shown in Figure 10. We notice that without skip-window, for each T, the performance of FT-Pro degrades quickly as the *fn* grows.

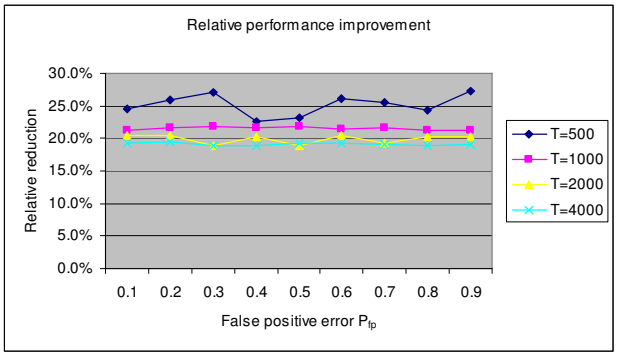

Figure 7. The impact of false-positive error

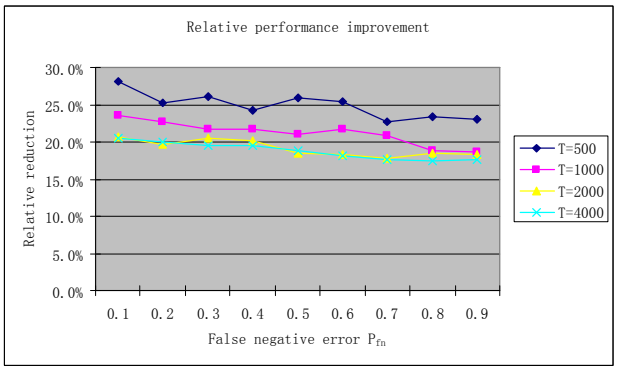

Figure 8. The impact of false-negative error

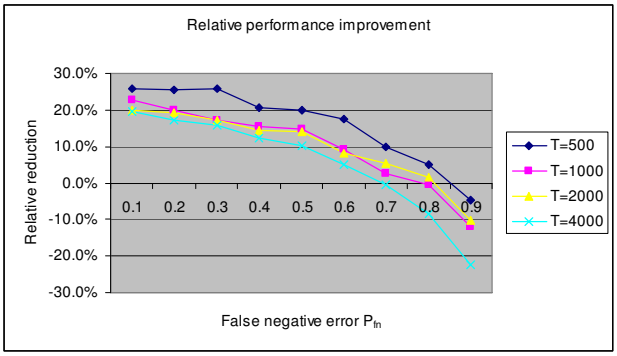

Figure 9 The impact of false-negative error without skip-window

4.3 Impact of Preventive Actions

In this set of experiments, we evaluate the sensitivity of FT-Pro to the costs of preventive actions. Using the same setting as in the previous experiment in figure 4.1, we vary the number of spare nodes, the cost ratio of C_{cp} / C_{pm} and the size of decision intervals respectively in Figure 12-14. Figure 12 shows that even if there is only one spare node available, FT-Pro can still outperform merely checkpointing. The main reason is that failure events are essentially rare and the probability of two simultaneous failures is small. The results in Figure 13 indicates that FT-Pro can maintain a stable performance gain when the ratio of C_{cp} / C_{pm} varying from 1 to 32. As described in Section 3, FT-Pro adaptively chooses a preventive action with the lowest expected execution time so as to avoid frequent migration when the migration cost is high. In Figure 14 and 15, we change the decision intervals, i.e. user's fault tolerance request interval, from I/8 to 8*I, where I is the default interval described in 4.2. Figure 14 shows that as the interval increases, the relative reduction of FT-Pro decreases. When the interval is small, the periodic checkpointing strategy introduces significant overhead by frequently interrupting the application, whereas FT-Pro can assist the application to ignore those unnecessary checkpointing requests. This again illustrates the advantage of FT-Pro: if the user unwisely posts too frequent checkpointing requests, FT-Pro can mitigate this adverse effect. Figure 15 shows the expected execution times of the same experiment, indicating when the interval becomes larger, and the execution times by using these strategies tend to increase. When the interval increases, FT-Pro has fewer chances to make adaptive decision so that the possibility of encountering failures increases.

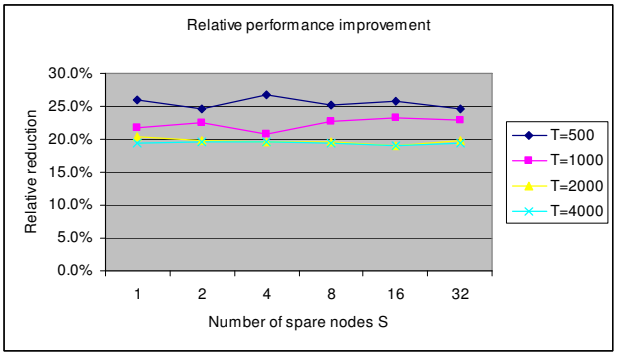

Figure 12. The Impact of number of spare nodes

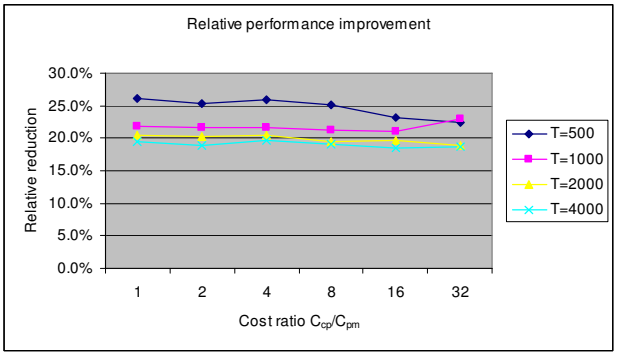

Figure 13. The impact of cost ratio

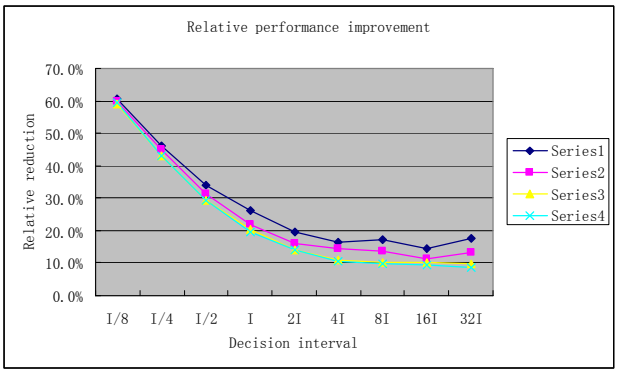

Figure 14. The impact of decision interval on relative reduction

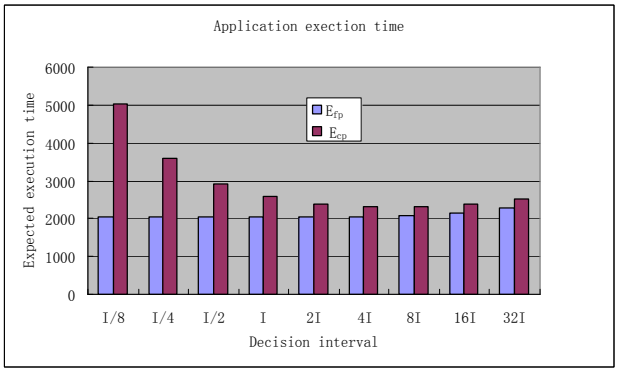

Figure 15. The impact of decision interval on application execution time

6 Conclusions

In the paper, we proposed an innovative fault management mechanism which exploits the failure prediction to adaptively conduct checkpointing and migration for cluster computing. Through extensive experiments with the real failure log from a production supercomputer at NCSA, we demonstrated that FT-Pro can significantly reduce application execution time compared to the traditional checkpointing/restart strategy with even modest prediction accuracy.

Our future work includes integrating the proposed adaptive fault management mechanism with various failure prediction algorithms and providing an automatic fault tolerant tool for cluster computing.

Acknowledgement

The authors would like to thank Xianhe Sun at Illinois Institute of Technology for numerous comments and suggestions that contributed to this work. We also would like to thank Charng-da Lu at University of Illinois at Urbana-Champaign for the Platinum failure log.

References

[1] Health Application Programming Interface (HAPI) . Available at http://www.renci.org/software/hapi/

[2] Hardware monitoring by lm sensors. Available at http://secure.netroedge.com/lm78/info.html.

[3] A. Bouteiller et al., "Coordinated Checkpoint versus Message Logging for Fault Tolerance MPI", Proc. of IEEE Cluster03, 2003

[4] V. Castelli et al., "Proactive Management of Software Aging", IBM Journal of Research and Development, 45(2), 2001

[5] S. Chakravorty, C. Mendes, and L. Kale, "Proactive Fault Tolerance in Large Systems", Proc. of HPCRI workshop, 2005

[6] E.G. Coffman and E.N. Gilbert, "Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance", IEEE Trans. Reliability, vol. 39, no. 1, pp. 9-18, Apr. 1990.

[7] A. Duda, "The Effects of Checkpointing on Program Execution Time", Information Processing Letters, vol. 16, no. 5, pp. 221-229,June 1983.

[8] Hoffmann GA, Salfner F., Malek M. Advanced Failure Prediction in Complex Software Systems,. SRDS 2004

[9] C. Leangsuksun, T. Liu, S. L. Scott T. Rao, andRichard Libby. A failure predictive and policy-basedhigh availability strategy for Linux high performancecomputing cluster. Proceedings of 5th LCIInternational Conference on Linux Clusters, 2004

[10] Charng-Da Lu, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2005

[11] A. Oliner, Ramendra K. Sahoo, José E. Moreira, Meeta S. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems", IPDPS 2005

[12] A. J. Oliner, L. Rudolph, Ramendra K. Sahoo, José E. Moreira, Manish Gupta: Probabilistic QoS Guarantees for Supercomputing Systems. DSN 2005

[13] A. Oliner, Ramendra K. Sahoo, José E. Moreira, Manish Gupta, Anand Sivasubramaniam: Fault-Aware Job Scheduling for BlueGene/L Systems. IPDPS 2004

[14] Tatsuya Ozaki, Tadashi Dohi, Hiroyuki Okamura, Naoto Kaio, "Min-Max Checkpoint Placement under Incomplete Failure Information", DSN 2004: 721-730

[15] Soila M. Pertet, Priya Narasimhan, "Proactive Recovery in Distributed CORBA Applications", DSN 2004: 357-366

[16] Hiroyuki Okamura, Yuki Nishimura, Tadashi Dohi: A Dynamic Checkpointing Scheme Based on Reinforcement Learning. PRDC 2004: 151-158

[17] J. S. Plank and K. Li, "Ickp: a consistent checkpointer for multicomputers", IEEE Parallel & Distributed Technology, 2(2):62--67, Summer 1994.

[18] D. Tang, R. Iyer, and S. Subramani, "Failure Analysis and Modeling of a VAXcluster System", Proc. of Intl. Symp. Fault Tolerance Computing, 1990

[19] R. Vilalta and S. Ma, Predicting Rare Events in Temporal Domains Using Associative Classification Rules, Technical Report,IBM Research, T. J. Watson Research Center, Yorktown Heights, NY (2002).

[20] Ramendra K. Sahoo, A. Oliner, Irina Rish, Manish Gupta, José E. Moreira, Sheng Ma, Ricardo Vilalta, Anand Sivasubramaniam: Critical event prediction for proactive management in large-scale computer clusters. KDD 2003: 426-435

[21] Ann T. Tai, Kam S. Tso, William H. Sanders, Savio N. Chau: A Performability-Oriented Software Rejuvenation Framework for Distributed Applications. DSN 2005: 570-579.

[22] Ricardo Vilalta, Sheng Ma: Predicting Rare Events In Temporal Domains. ICDM 2002: 474-481

[23] Gary M. Weiss, Haym Hirsh: Learning to Predict Rare Events in Event Sequences. KDD 1998: 359-363

[24] Gary M. Weiss，Haym Hirsh，Learning to Predict Rare Events in Categorical Time-Series Data，AAAI Workshop，1998

[25] John W. Young, "A First Order Approximation to the Optimal Checkpoint Interval", Comm. ACM 17(9): 530-531(1974)