Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

# FTRP: a new fault tolerance framework using process replication and prefetching for high-performance computing*

Wei HU‡[1,2], Guang-ming LIU[1,2], Yan-huang JIANG[1]

[1]*College of Computer, National University of Defense Technology, Changsha 410073, China*

[2]*National Supercomputer Center in Tianjin, Tianjin 300457, China*

E-mail: huwei@nscc-tj.gov.cn; liugm@nscc-tj.gov.cn; yhjiang@nudt.edu.cn

Received Aug. 3, 2016; Revision accepted Mar. 3, 2017; Crosschecked Oct. 9, 2018

**Abstract:** As the scale of supercomputers rapidly grows, the reliability problem dominates the system availability. Existing fault tolerance mechanisms, such as periodic checkpointing and process redundancy, cannot effectively fix this problem. To address this issue, we present a new fault tolerance framework using process replication and prefetching (FTRP), combining the benefits of proactive and reactive mechanisms. FTRP incorporates a novel cost model and a new proactive fault tolerance mechanism to improve the application execution efficiency. The novel cost model, called the 'work-most' (WM) model, makes runtime decisions to adaptively choose an action from a set of fault tolerance mechanisms based on failure prediction results and application status. Similar to program locality, we observe the failure locality phenomenon in supercomputers for the first time. In the new proactive fault tolerance mechanism, process replication with process prefetching is proposed based on the failure locality, significantly avoiding losses caused by the failures regardless of whether they have been predicted. Simulations with real failure traces demonstrate that the FTRP framework outperforms existing fault tolerance mechanisms with up to 10% improvement in application efficiency for common failure prediction accuracy, and is effective for petascale systems and beyond.

**Key words:** High-performance computing; Proactive fault tolerance; Failure locality; Process replication; Process prefetching

https://doi.org/10.1631/FITEE.1601450 **CLC number:** TP338.6

## 1 Introduction

Reliability is one of the major challenges as the scale of the supercomputer grows up from petascale to exascale. Recent work indicates that failures occur more frequently than commonly believed. The failure rate per year of servers reaches 2%–4% (Schroeder et al., 2009), the failure rate of disk drives reaches 1%–5% (Pinheiro et al., 2007), and the failure rate of dynamic random access memory (DRAM) reaches 2% (Schroeder et al., 2009). Current petascale systems reportedly have a mean time between failures (MTBF) of less than 10 h to a few days, while future exascale systems are anticipated to have an MTBF of less than one hour (Bhatele et al., 2011).

Large scientific applications usually employ many processors for days or weeks to model complicated problems. The larger the application scale or the longer the running time, the more failures the applications will encounter. These typical applications commonly use a message passing interface (MPI) to span the processors (Vetter and Mueller, 2003). In this situation, the failure of a single process directly

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61272141, 61120106005, and 61303068) and the National High-Tech R&D Program of China (No. 2012AA01A301)

ORCID: Wei HU, http://orcid.org/0000-0002-8839-7748

aborts the entire applications.

To avoid suffering from failures, many fault tolerance mechanisms have been presented in the high-performance computing (HPC) area (Kalaiselvi and Rajaraman, 2000; Elnozahy et al., 2002; Roman, 2002; Lan and Li, 2008; George and Vadhiyar, 2012, 2015; Bouguerra et al., 2013; Egwutuoha et al., 2013). The most popular and widely used is the checkpoint/restart mechanism. This periodic checkpoint mechanism induces a lot of overheads, because the checkpoint data of all the processes must be stored in permanent storage, i.e., the parallel file system (Pinheiro et al., 2007). Although much work has been presented for checkpointing optimization, such as incremental checkpointing (Sancho et al., 2004) and diskless checkpointing (Plank et al., 1998; Lu, 2005), the checkpoint/restart mechanism is still a reactive mechanism and is inherently inefficient.

Recently, proactive fault tolerance has gradually gained importance because failure prediction techniques have progressed. Gujrati et al. (2007) presented a failure predictor, whose precision (proportion of the number of correct predictions to the total number of predicted failures) exceeds 80% and whose recall (proportion of the number of correct predictions to the total number of true failures) is about 65%. Gainaru et al. (2012b) presented another predictor with the precision exceeding 90% and the recall being about 50%. Such a prediction performance provides enough opportunities for proactive fault tolerance. Much work also provided proactive fault tolerance based on failure prediction (Lan and Li, 2008; George and Vadhiyar, 2012, 2015; Bouguerra et al., 2013). However, proactive fault tolerance cannot avoid re-executing the application alone if failures are not perfectly predicted. Therefore, these mechanisms are often combined with periodic checkpointing. How to effectively choose a countermeasure in both proactive and reactive mechanisms according to the fault tolerance status of an application is a crucial and complicated problem, because a number of factors need to be considered, involving overheads of the actions, the accuracy of the prediction, and the real effectiveness of actions. Raising the efficiency and flexibility of the whole fault tolerance framework, which previous work lacks, is the motivation of our work.

In this study, we develop a new fault tolerance framework using process replication and prefetching (FTRP). FTRP provides a pattern for complementarily combining proactive and reactive fault tolerance mechanisms. It can efficiently use the benefits of both proactive and reactive fault tolerance mechanisms and avoid their shortcomings. This framework effectively improves the application performance in the presence of failures for large parallel systems. The main contributions of this study include the following:

1. We present the FTRP framework combining the benefits of proactive and reactive fault tolerance mechanisms to form a whole fault tolerance system. FTRP provides a work-most (WM) cost model to evaluate the system status against failures in real time and adopts the appropriate fault tolerance choice at runtime to minimize the failure impact. FTRP is flexible enough to decrease the impact of failures on the applications while maximizing its efficiency.

2. Based on the observation and analysis of real parallel systems, to the best of our knowledge, we are the first group to find failure locality in supercomputers. We divide failure locality into temporal and spatial localities and analyze their characteristics separately.

3. Based on failure locality, we present a new proactive fault tolerance mechanism called 'process replication with process prefetching' (PRP2). This mechanism can provide the redundancy for failure-prone processes to the greatest extent, because it not only replicates the processes according to the results of the failure predictor, but also prefetches the processes based on failure locality. This new method can significantly reduce losses caused by failures.

## 2 Related work

For fault tolerance in supercomputing, checkpointing is a conservative but frequently used method which has been studied for several decades. Kalaiselvi and Rajaraman (2000), Elnozahy et al. (2002), Roman (2002), and Egwutuoha et al. (2013) have detailed these different checkpointing techniques. In the field of HPC, many checkpointing implementations have been developed, such as the libckpt tool for Unix (Plank et al., 1995), Berkeley Lab Checkpoint/Restart (BLCR) (Hargrove and Duell, 2006), and scalable checkpoint/restart (SCR)

(Moody et al., 2010). A lot of work has also been done to optimize the checkpoint mechanisms (Young, 1974; Babaoglu and Joy, 1981; Plank et al., 1998; Sancho et al., 2004; Lu, 2005; Daly, 2006; Mohror et al., 2012). However, the checkpointing mechanisms now face the challenges of the larger computation scale and the higher failure rates.

There has been much progress on failure prediction (Salfner et al., 2010). For both model-based (Hamerly and Elkan, 2001; Hellerstein et al., 2001) and data-driven (Vilalta and Ma, 2002; Sahoo et al., 2003; Liang et al., 2006; Gujrati et al., 2007) methods, these work provides predictors of certain accuracy offering the possibility to solve problems from a proactive perspective.

Research on proactive fault tolerance has attracted more interest. Lan and Li (2008) developed an adaptive fault management approach called 'FT-Pro' using proactive migration and reactive checkpointing. Bouguerra et al. (2013) considered the combination of failure prediction, proactive checkpointing, and periodic checkpointing to improve the computing efficiency. George and Vadhiyar (2012, 2015) employed checkpointing, live migration, and rescheduling based on failure prediction to improve the performance of malleable applications, and then developed a mechanism for proactive fault tolerance combining partial replication and periodic checkpointing. This work is similar to the focus of ours; however, our work is significantly different. Locality in the application program has been all known and used in cache and memory management (Denning, 2005; Weinberg et al., 2005; Zhong et al., 2009; Gupta et al., 2013). Nevertheless, the concept of failure locality has not been presented in HPC, and we are among the first ones to define failure locality in HPC. We propose the FTRP framework by introducing a new cost model which enables appropriate fault tolerance action choices to be resiliently made, and a fault tolerance mechanism using failure locality to minimize work losses caused by the failures. FTRP focuses on improving the system efficiency and resilience under failures and achieves good results.

## 3 Failure locality

Before discussing the FTRP framework, we detail the failure locality phenomenon, which was observed in the process of analyzing the failure logs. We illustrate the failure locality phenomenon taking the failure traces PNNL08 (Brown and Smith, 2008) and THNSCC (Hu et al., 2015) as examples. PNNL08 failure traces were collected from the Hewlett-Packard (HP) supercomputers called 'MPP2'. This system was operated by the Environmental and Molecular Science Laboratory (EMSL) in Richland, Washington (USA), and consisted of 940 compute nodes marked as 0–939. THNSCC failure traces were collected from the TH-1A supercomputer in the National Supercomputer Center in Tianjin (NSCC-TJ) in 2015. TH-1A has a total of 7168 compute nodes, and the failure traces include the data of 5632 compute nodes marked as 0–5631.

The locality of references in the memory system is familiar to us, and the locality phenomenon also exists in the occurrence of failures in supercomputers. Fig. 1 shows the failure events distributing with time, one point representing one failure event. It can be clearly seen that if one compute node fails, the same node is likely to fail in the near future (Fig. 1) . This phenomenon is shown in rectangles in Fig. 1, and we call it 'failure temporal locality'. Similarly, if one compute node fails, some neighbors of the node are likely to fail in the near future. This phenomenon is shown as ellipses in Fig. 1, and we call it 'failure spatial locality'. Fig. 1 shows an intuitionistic presentation of failure locality, showing both failure temporal locality and failure spatial locality. Figs. 2 and 3 detail the number of failure localities and their influencing factors. To further observe and analyze failure locality in supercomputers, we give the following definitions:

**Definition 1** (Failure temporal locality) The nodes that failed in the immediate past have a high probability of failure in the immediate future.

**Definition 2** (Failure spatial locality) The nodes that physically located near a node that failed in the immediate past have a high probability of failure in the immediate future.

Failure temporal and spatial localities reveal the failure occurrence tendency when parallel applications run on supercomputers. Figs. 2 and 3 show the statistics of failure locality in different real failure traces. We use recurrence distance to measure the failure temporal locality, which is similar to reuse distance related to memory temporal locality. The recurrence distance is defined as the number of node failures between two consecutive failures of the same

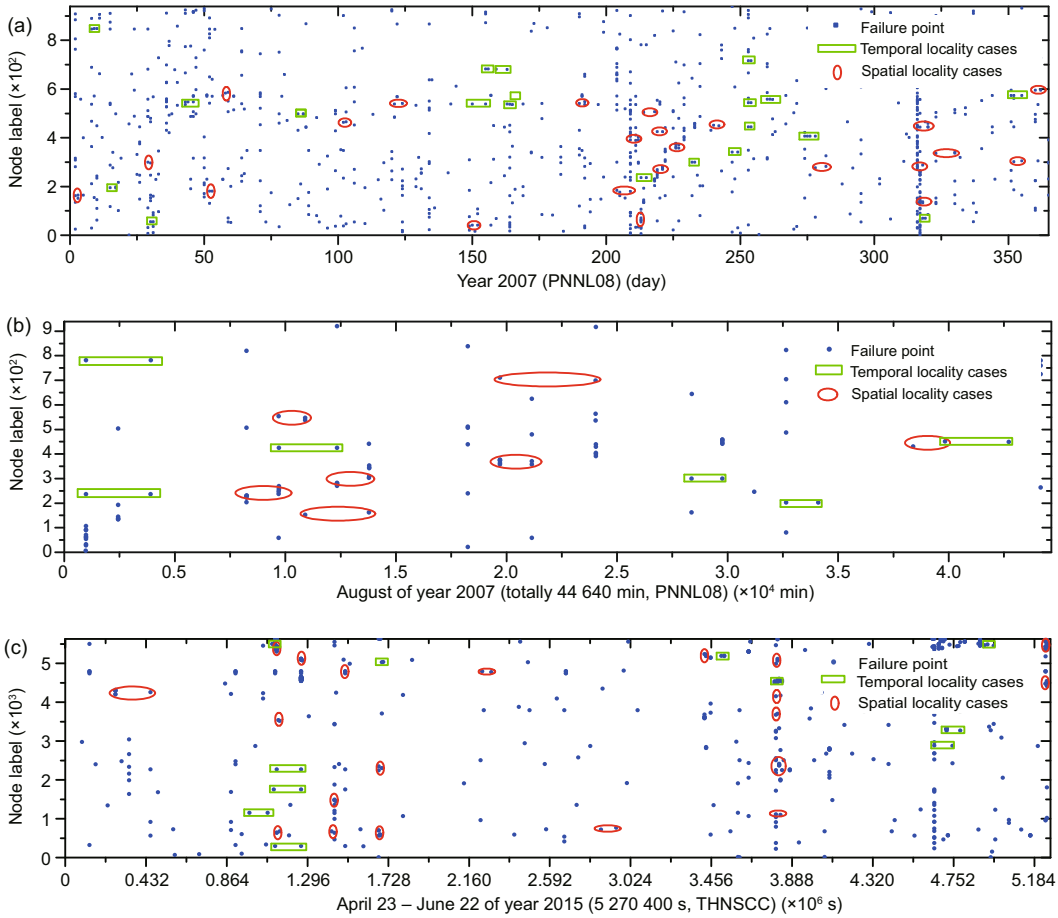


**Fig. 1 Failure events distributing over time: (a) PNNL08 failure traces in 2007; (b) PNNL08 failure traces in August 2007; (c) THNSCC failure traces for 61 days in 2015**

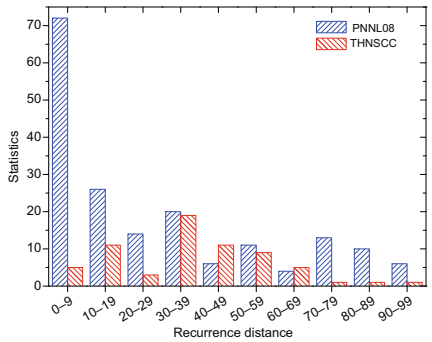


**Fig. 2 Temporal locality statistics under different recurrence distances**

Recurrence distance is defined as the number of node failures between two consecutive failures of the same node

node. For example, if node $a$ fails, followed by the failures of nodes $b$ and $c$, then node $a$ fails again, and thus the recurrence distance of node failure $a$ is three. Fig. 2 shows the statistics of failure nodes under different recurrence distances. It can be seen that there are more temporal localities within shorter recurrence distances, and the failure temporal locality is widespread.

When we measure the failure spatial locality, we use a look back window (LBW) considering the previous $N$ node failures as the spatial locality search scope and stride length to quantify the relative physical distance between the failure nodes. For example, if node $a$ fails, followed by the failures of nodes $b$ and $c$, then node $a-1$ fails; therefore, the LBW from node failures $a-1$ to $a$ is three, and the stride length from node failures $a-1$ to $a$ is one. Fig. 3 shows the statistics of failure spatial locality of the two systems under different LBWs and stride length. From both PNNL08 and THNSCC failure traces, the spatial locality is increasing with the growth of the LBW and stride length. The larger LBW and stride length can acquire higher spatial locality, but more search over-

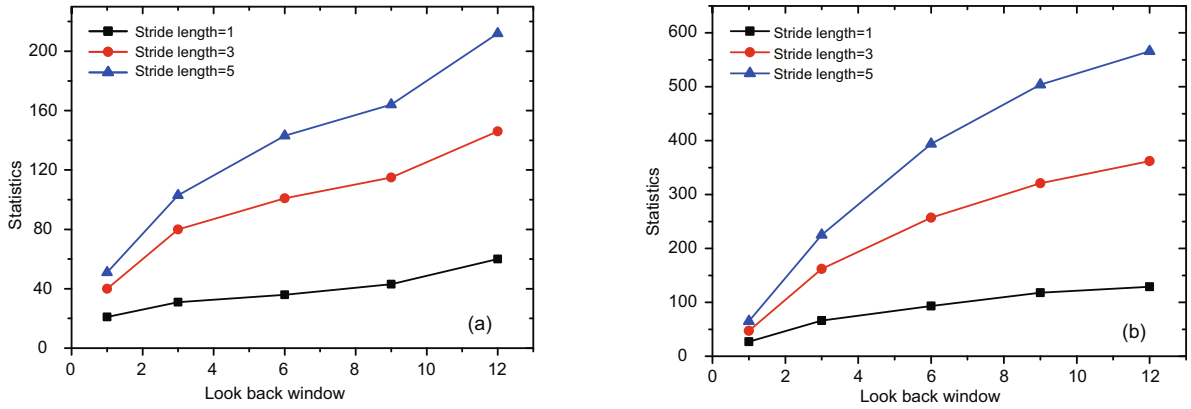


**Fig. 3 Spatial locality statistics in different look back windows (LBWs) and stride lengths: (a) PNNL08 traces; (b) THNSCC traces**

LBWs are used to consider the previous $N$ node failures as the spatial locality search scope, and stride length is used to quantify the relative physical distance between the failure nodes

heads including time and space. The stride length reflects the correlation between the node failures, and this correlation is related to not only the topology of network interconnection but also the failure types.

Failure locality provides a guidance for process replication and redundancy. In Section 6, the new proactive fault tolerance mechanism, PRP2, will be built on failure locality.

## 4 FTRP framework

A parallel application on a supercomputer consists of a number of processes running on different compute nodes. In this study, we focus on fail-stop failures (Dwork et al., 1988; Mohammed et al., 2012). When a failure occurs, there is an immediate termination of the running application. This fail-stop model is commonly used in fault tolerance mechanisms.

Assume that there is a failure predictor that can predict the node failures in a system for a given time window (Lan et al., 2010; Gainaru et al., 2012a). In this study, the predictor is a process to periodically estimate the future status of the health of all nodes and give warnings for failure-prone nodes in the next time interval.

There are two metrics to measure this prediction mechanism: precision and recall. Obviously, the higher the values of precision and recall, the better the predictor. If the predictor predicts that one node will fail, this node is considered failure-prone in the next time interval; otherwise, the node is considered healthy.

Fig. 4 shows an overview of FTRP, which manages the different fault tolerance mechanisms to minimize the application running time. FTRP segments the running time of a parallel application into time intervals according to the user requirements or system settings. At the end of each segment, there is a decision point (DP). The aim of FTRP is to adaptively choose the best strategy on the DPs to minimize the impact of failures and reduce the running time of the whole application. We divide all the compute nodes used for an application into two pools: application pool and replica pool. The real work of the application is running in the application pool, whereas the replica processes in the replica pool. At DPs, according to the results of the failure predictor and the execution status of the applications, the WM cost model can evaluate the different fault tolerance actions and give the appropriate action suggestion for the next time interval. FTRP chooses the fault tolerance action and further promotes the application execution to the next DP until the completion of the applications.

On each DP, we provide four fault tolerance actions as follows:

1. SKIP is where no action is taken because of no failure risk.

2. PRP2 is denoted as the process replication with a process prefetching mechanism, where the processes on failure-prone nodes are selected for allocation of replica processes on healthy nodes in the replica pool. Meanwhile, the processes on the failure

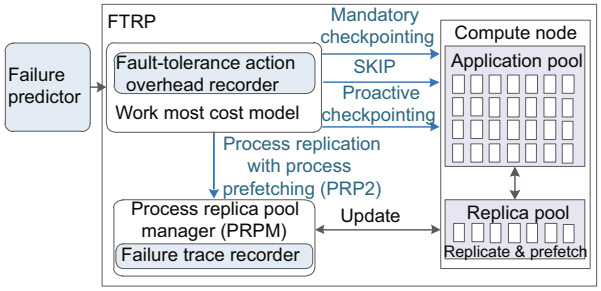


**Fig. 4 FTRP framework**

locality nodes are selected to allocate replica processes for prefetching. According to the locality phenomenon, two kinds of nodes are more prone to fail in the near future, called 'failure locality nodes': one is the node that just failed, and the other is the node adjacent to the failure node. There is more possibility that these failure locality nodes will fail, and the processes on them should have prefetched replica processes.

3. Proactive checkpointing (PCKP) is where the application takes a proactive checkpoint. This is used for cases that the PRP2 mechanism cannot effectively manage, such as a burst of multiple failures in one interval, and temporary shortages of the nodes that can be used for PRP2 in the replica pool. We use coordinated checkpointing, and these processes synchronize to perform checkpointing.

4. Mandatory checkpointing (MCKP) is where the application takes a mandatory checkpoint. This is used to make up losses caused by the failures that have not been predicted. Because the recall of a failure predictor usually cannot reach 100%, the performance losses may be significant when a number of SKIP or PRP2 actions have been continuously taken before an unpredicted failure. We use MCKP, which is also coordinated checkpointing, to avoid this situation.

The running flow of FTRP is illustrated in Fig. 5. Suppose that there are a total of $n$ nodes allocated to the applications, named $\mathrm{CN}_0$ to $\mathrm{CN}_{n-1}$. In the initialization procedure, all nodes are divided into application pool and replica pool nodes, as Fig. 5 shows; nodes $\mathrm{CN}_0$ to $\mathrm{CN}_{m-1}$ comprise the application pool, and nodes $\mathrm{CN}_m$ to $\mathrm{CN}_{n-1}$ comprise the replica pool ($m$=1, 2, ..., $n$). There are a total of $m$ processes ($P_0$, $P_1$, ..., $P_{m-1}$) running on each node in the application pool, and the other ($n-m$) processes in the replica pool are the replica processes prefetched from the application pool. During the runtime, when the applications run for every $I$ time, there will be a DP. On every DP, FTRP triggers the WM cost model to determine the appropriate fault tolerance action according to the current program running status and prediction results. Then FTRP invokes the corresponding action, and the cost model records the runtime overhead of the fault tolerance actions. If the application fails during the normal running or the execution process of fault tolerance actions, it rolls back to the most recent checkpoint and is re-executed. FTRP uses a doubly linked list to manage the replica pool and a text file to store the mapping between compute nodes and replica nodes. This is a global data structure which is accessible by all the processors in the shared file system.

Assume that the application and FTRP run as Fig. 5 shows, and we use these examples to illustrate the available benefits of a parallel application with FTRP. The WM cost model is used in the decision-making process for choosing an appropriate fault tolerance action on every DP in real time. In the following examples we briefly describe the reasons for choosing fault tolerance actions, and will discuss them in detail in Section 5.

1. If the failure predictor does not predict any failure in the near future, the application-related system may have a low risk of failure. In this case, FTRP avoids the fault tolerance actions by taking a SKIP action. This significantly decreases the fault tolerance overheads, such as checkpointing, when the failure probability is slight. The SKIP action is taken at $\mathrm{DP}_2$ and $\mathrm{DP}_4$ (Fig. 5).

2. At $\mathrm{DP}_3$ where the predictor predicts a failure on node $\mathrm{CN}_2$ (which turns out to be a correct prediction, e.g., a true positive), FTRP takes a PRP2 action: it replicates process $P_2$ from node $\mathrm{CN}_2$ to node $\mathrm{CN}_{m+1}$, and prefetches process $P_3$ from node $\mathrm{CN}_3$ to node $\mathrm{CN}_{m+2}$. This action efficiently avoids work losses due to the failure of node $\mathrm{CN}_2$.

3. At $\mathrm{DP}_5$ where the predictor fails to warn about the upcoming failure on process $P_3$ (which turns out to be a missed failure, e.g., a false positive), FTRP takes a SKIP action. Generally, the application loses the work done from the last checkpoint and needs to restart, but process $P_3$ has been prefetched at $\mathrm{DP}_3$. The replica process $P_3'$ on node $\mathrm{CN}_{m+2}$ avoids the work losses.

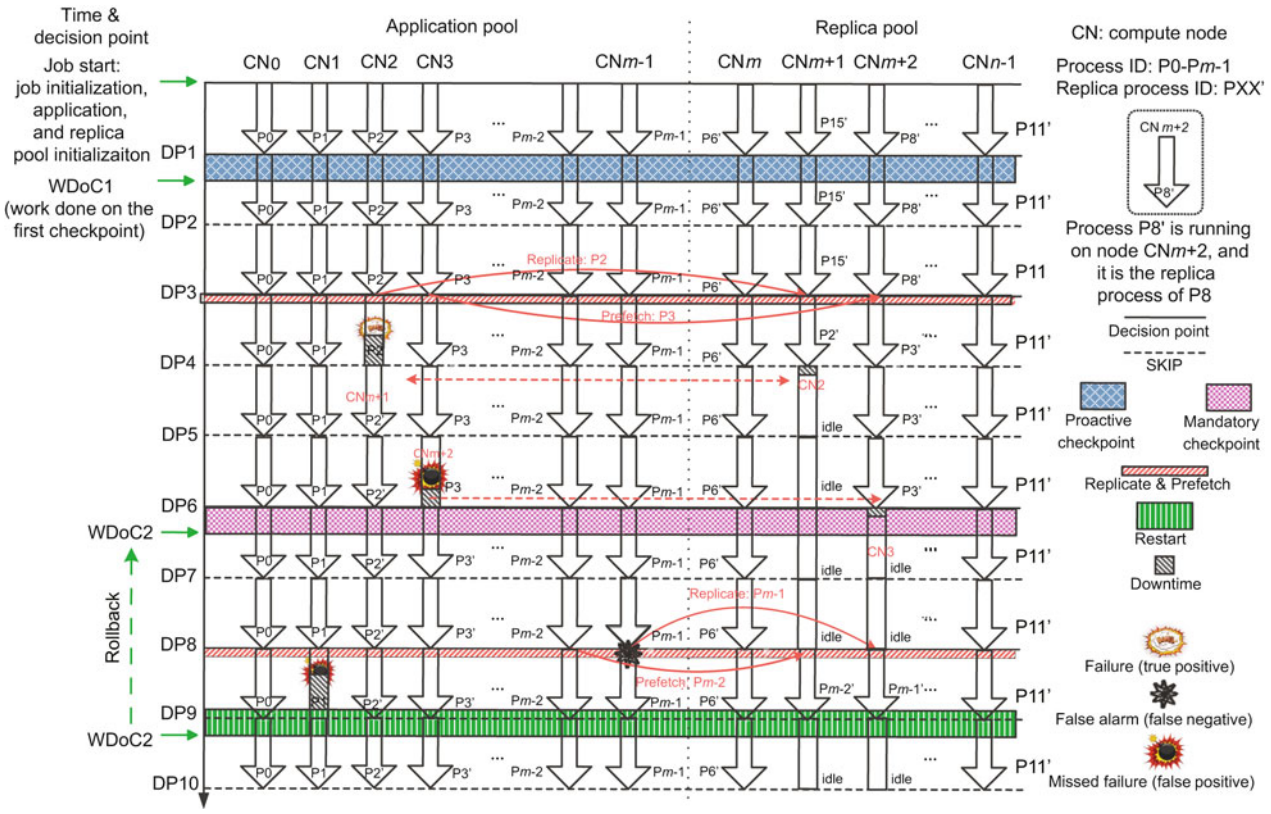


**Fig. 5 Examples of the FTRP running process**

4. At $DP_6$, considering that the work losses would be significant if an unforeseeable failure occurs in the next interval, FTRP decides to take an MCKP action, although no failure warning issues at this point. At $DP_8$, there is a false alarm/negative from the failure predictor, and FTRP takes a PRP2 action but does not protect process $P_1$ due to the false failure prediction. When process $P_1$ fails in the interval between $DP_8$ and $DP_9$, the application has to restart, but the work that needs to be re-executed is significantly reduced by the MCKP action.

Although FTRP may introduce additional overheads in the cases of false alarms/negatives, such as at $DP_8$, the benefits from PRP2 and SKIP are dominant, and this will be discussed in Section 7.

## 5 Work-most cost model

The WM cost model is used to choose the appropriate fault tolerance action at each DP. Fig. 4 shows that this model considers the latest program running status and failure prediction results to quickly determine the appropriate action to decrease the impact of failure occurrences and fault tolerance overheads.

The WM cost model can automatically record and update the parameters related to the program running status, including failure prediction accuracy, checkpoint overhead, process replication overhead, and replica pool status. This decision-making process is a trade-off. Adopting a fault tolerance action can efficiently reduce the risk of failures, but the action itself will create a overhead that cannot be ignored. Meanwhile, if there is not enough fault tolerance work against failures, the program may suffer from many losses in case of failures.

Assume that the expected time for real work in the next time interval is $U_{\text{next}}$. Different actions have different fault tolerance overheads and benefits. Therefore, the aim of the model is to complete more real work in the next time interval and avoid the failure risk. Because the results of the predictor are efficient only for the next time window, the WM model should take account of the timeliness of the prediction results and select the action to maximize

$U_{\text{next}}$.

After listing a set of symbols that are frequently used in this study in Table 1, we detail the WM cost model. Based on the precision and recall of the failure predictor, we estimate $U_{\text{next}}$ of all the actions and give the action choice that has the maximum $U_{\text{next}}$ value.

1. SKIP: Assume that the failure occurs immediately before the next DP. If one or more failures occur in the next interval, the useful time of the applications will lose $C_{\text{r}}$ time for recovery and $J$ time for redoing the real work that has been done since the most recent checkpoint. If there is no failure, the applications use the whole time interval $I$ to do the real work. Thus, we have

$$\begin{aligned} U_{\text{next}} &= I \cdot (1-P_{\text{fail}}) + (-C_{\text{r}} - J) \cdot P_{\text{fail}}, \\ P_{\text{fail}} &= 1-(1-\text{Precision})^{N_{\text{fail}}}, \end{aligned} \tag{1}$$

where $N_{\text{fail}}$ denotes the number of compute nodes that are predicted to be failure-prone in the next interval.

2. PRP2: The process replication action is combined with process prefetching in parallel. In this action, the applications spend $C_{\text{rep}}$ for process parallel replication and prefetching, and the prefetching operations do not create any new overhead. Due to the possibility of multiple failures in one interval, if nodes in the replica pool are not enough to accommodate all the replication requests, FTRP implements a best-effort strategy to replicate as many processes as possible, and process replication has a higher priority than process prefetching. Thus, we can calculate $U_{\text{next}}$ as

$$\begin{aligned} U_{\text{next}} &= (I-C_{\text{rep}}) \cdot (1-P_{\text{fail}}) + (-C_{\text{r}} - J) \cdot P_{\text{fail}}, \\ P_{\text{fail}} &= \begin{cases} 1-(1-\text{Precision})^{N_{\text{fail}}-N_{\text{s}}}, & N_{\text{fail}} > N_{\text{s}}, \\ 0, & N_{\text{fail}} \le N_{\text{s}}, \end{cases} \end{aligned} \tag{2}$$

where $N_{\text{s}}$ denotes the number of compute nodes in the replica pool that are failure-free and can be allocated for new processes in the next interval.

3. PCKP: This proactive checkpoint is based on the precision of the failure predictor. The applications spend $C_{\text{chk}}$ to complete checkpointing and update the record of the most recent checkpoint. If one or more failures occur in the next interval, the useful time of the application will lose only $C_{\text{r}}$ time for recovery. If there is no failure, the application uses $C_{\text{chk}}$ to complete checkpointing and the remaining time to do the real work. Thus, we have

$$\begin{aligned} U_{\text{next}} &= (I-C_{\text{chk}}) \cdot (1-P_{\text{fail}}) + (-C_{\text{r}}) \cdot P_{\text{fail}}, \\ P_{\text{fail}} &= 1-(1-\text{Precision})^{N_{\text{fail}}}. \end{aligned} \tag{3}$$

4. MCKP: The above three fault tolerance actions are all based on the precision of the failure predictor. Given the possibility of unpredicted failures, the performance losses could be significant when a number of SKIP or PRP2 actions have been continuously taken before an unpredicted failure. We use

**Table 1 List of main symbols, their meanings, and occurrences**

| Symbol | Meaning | Occurrence |
|---|---|---|
| Precision | Proportion of the number of correct predictions to the total number of predictions made | Section 4 |
| Recall | Proportion of the number of correct predictions to the total number of failures | Section 4 |
| $U_{\text{next}}$ | Expected time for real work in the next time interval | Section 5 |
| $J$ | Time for redoing the real work that has been done since the most recent checkpoint | Section 5 |
| $N_{\text{fail}}$ | Number of compute nodes predicted to be failure-prone in the next interval | Section 5 |
| $I$ | Fault tolerance time interval (time between two decision points) | Section 5 |
| $C_{\text{r}}$ | Recovery overhead | Section 5 |
| $C_{\text{rep}}$ | PRP2 overhead | Section 5 |
| $C_{\text{chk}}$ | Checkpointing overhead | Section 5 |
| $N_{\text{s}}$ | Number of compute nodes in the replica pool that are failure-free and can be allocated for new processes in the next interval | Section 5 |
| $W_{\text{u}}$ | Real work that has been done since the last checkpoint | Section 5 |
| $U_{\text{thr}}$ | Threshold that can be set to trigger the MCKP action by comparison with $W_{\text{u}}$ | Section 5 |
| $T_{\text{ckp}}$ | Optimal checkpoint interval for periodic checkpointing | Section 5 |
| $E$ | Performance metric called application efficiency, $E = T_{\text{IDL}}/T_{\text{R}}$ | Section 7 |
| $T_{\text{R}}$ | Application completion time in the real running environment with fault tolerance mechanisms in the presence of failures | Section 7 |
| $T_{\text{IDL}}$ | Application completion time when it is in the ideal failure-free environment | Section 7 |
| $N_{\text{actual}}$ | Number of actual failures reported by the failure predictor | Section 7 |

MCKP to avoid this situation. The trigger condition for MCKP can be expressed as

$$W_{\mathrm{u}} \geq U_{\mathrm{thr}}, \quad U_{\mathrm{thr}} = \frac{T_{\mathrm{ckp}}}{1 - \mathrm{Recall}}, \tag{4}$$

where $W_{\mathrm{u}}$ is the real work that the application has done since the last checkpoint. Fig. 6 shows that the WM cost model restarts the $W_{\mathrm{u}}$ time at the end of each checkpoint, and that it updates $W_{\mathrm{u}}$ on every DP to calculate the real work that the application has done since the last checkpoint. If $I$=1800 s, $C_{\mathrm{chk}}$= 600 s, and $C_{\mathrm{rep}}$=120 s, at $\mathrm{DP}_{m+3}$, we can obtain $W_{\mathrm{u}}$=4680 s (Fig. 6).

$U_{\mathrm{thr}}$ is a threshold that can be set to trigger the MCKP action by comparison with $W_{\mathrm{u}}$. Considering the recall of the failure prediction and the optimal checkpoint interval for pseriodic checkpointing $T_{\mathrm{ckp}}$, $U_{\mathrm{thr}}$ is set to $T_{\mathrm{ckp}}/(1 - \mathrm{Recall})$. This is based on an intuitive estimation that the optimal checkpoint interval is $T_{\mathrm{ckp}}/(1 - \mathrm{Recall})$ for periodic checkpointing with the help of an active fault tolerance. On DPs, the WM cost model updates and checks $W_{\mathrm{u}}$. If $W_{\mathrm{u}}$=$U_{\mathrm{thr}}$, the MCKP action will be triggered. Otherwise, the model evaluates the above three actions and makes the choice. This method can effectively avoid the impact of unpredicted failures. Clearly, if the recall is equal to 0, it means that the FTRP may become periodic checkpointing with extra overheads of PRP2 actions.

## 6 Process replication with prefetching

In this section, we will detail a new proactive fault tolerance action and process replication with prefetching. Process replication methods are mature, such as rMPI (Ferreira et al., 2011), RedMPI (Elliott et al., 2012), and PAREP-MPI (George and Vadhiyar, 2015). Thus, our work focuses on the combination of process replication and prefetching.

FTRP completes the PRP2 action in the following ways:

1. Application initialization: When the resource manager allocates the compute nodes to the application, the process replica pool manager (PRPM) reserves a certain number of nodes as the replica pool, and the other nodes are allocated for real work as the application pool. In this stage, PRPM chooses some nodes to comprise the replica pool. According to the failure prediction results, the failure-prone nodes that may fail in the first time interval are put into the replica pool, and then some other nodes are randomly selected to make up the remainder of the replica pool. Then PRPM prefetches the processes running on the suspicious nodes to the replica pool in the reverse chronological order according to the node failure logs. The suspicious nodes are the nodes recorded by the failure log recorder and their adjacent nodes, and therefore these nodes have the failure locality.

2. FTRP calls PRP2 while the application runs: When FTRP calls a PRP2 action at the DPs, PRP2 completes process replication and prefetching in parallel; this method can effectively hide the prefetching overhead. PRPM manages nodes in the replica pool in a first in first out (FIFO) manner. When adopting the PRP2 action, the nodes allocated by PRPM give a priority to process replication, and use the rest of the nodes for prefetching (nodes that are down are ignored). PRPM uses a doubly linked list to manage the node order for process replication and prefetching. The node at the head of the doubly linked list is used to assume a new process for replication or prefetching, and then it is inserted at the end of the doubly linked list. Before the PRP2 action, the node positions in this linked list are adjusted in accordance with the following principles: (1) Nodes that have just returned from failures in the replica pool are inserted at the end of the link list, because reducing the process replication chances to these kinds of nodes (which have failure temporal localities) can reduce the failure probabilities for the replica processes; (2) If the predictor provides a false alarm/negative and

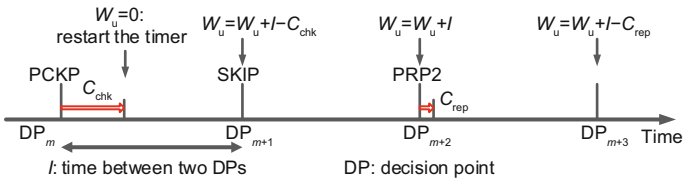


**Fig. 6 An example showing the meaning and calculation of $W_{\mathrm{u}}$**

the PRP2 action has been taken, these nodes with useless pocesses are inserted at the head of the linked list and have a priority to obtain replica processes at subsequent PRP2 actions. When the PRP2 action is completed, the changes in the doubly linked list are updated to the text file of the compute-replica map.

3. Replica pool management: We use an efficient data structure for the compute-replica map, which is similar to the one that George and Vadhiyar (2015) used. This compute-replica map file is a text file on a shared file system ensuring its consistency across nodes. It consists of $n$ lines, each of which is a map between the compute node number and the corresponding replica node number, and $n$ represents the total number of compute nodes . If a compute node does not have a replica node, its replica node number will be $-1$. At the DPs, this map file needs to be updated according to the doubly linked list only when the PRP2 action is adopted. The overhead used by the processes to check the updates of the map file can be neglected for the following reasons: if the file is not modified, it needs only to check the file modification time; otherwise, each process needs only to read a line from the file. With the acceleration of read cache policies in modern operating systems, this read overhead can be neglected.

The process replicas on the nodes of the replica pool can effectively avoid the damage of node failures in the application pool. Two copies of a process simultaneously run; if a process stops performing its desired function, the corresponding replica process can take over its computation position. Process replication based on the failure predictor can cover a certain number of, but not all, failure processes in the next time interval. Meanwhile, process prefetching can compensate for the predictor's weakness to a certain degree to back up more failure processes. This will efficiently reduce the time losses required to recover from the most recent checkpoint and redo the real work. The different LBWs and stride lengths can also influence the efficiency of process prefetching.

## 7 Experiments

We evaluated the FTRP framework using trace-based simulations. The performance metric called 'application efficiency' ($E$) was used to compare the effects of different mechanisms. Thus, the application efficiency can be expressed as

$$E = T_{\mathrm{IDL}}/T_{\mathrm{R}}, \tag{5}$$

where $T_{\mathrm{R}}$ denotes the application completion time in the real running environment with fault tolerance mechanisms in the presence of failures, and $T_{\mathrm{IDL}}$ denotes the application completion time in the ideal failure-free environment.

The simulator developed takes the failure traces, accuracy metrics of the failure predictor, and other application execution parameters as inputs. It is a trace-driven simulator and uses real failure traces from production supercomputers. The failure traces are PNNL08 and THNSCC traces, which have been detailed in Section 3. This simulator uses the WM cost model to evaluate and choose appropriate fault tolerance actions according to failure events, prediction accuracy, and application execution status at a constant interval $I$. These actions include SKIP, PRP2, PCKP, and MCKP. Their advantages and overheads have been detailed in Section 5. The outputs provided by the simulator are application completion time such as $T_{\mathrm{FTRP}}$ using FTRP and related running information.

We evaluated our framework against FT-Pro and periodic checkpointing developed by our simulator. We have extended FT-Pro (Lan and Li, 2008) to support a live migration that does not involve checkpointing, and modified FT-Pro. Thus, it takes a precautionary checkpoint since the last checkpoint reaches a threshold. This can fit the live migration strategy and be comparable to FTRP. For periodic checkpointing, we use a higher-order checkpoint/restart model (Daly, 2006) to calculate the optimal checkpoint interval, which is also used in MCKP. This model takes the platform MTBF and the overheads of checkpointing as inputs.

The simulator-related parameters are summarized in Table 2. These configurations and the corresponding ranges are acquired from real experiments, failure traces, and research (Lan and Li, 2008). They are in line with the values given for the 2011 cost scenario (Cappello et al., 2010). The interval $I$ between DPs is set as 30 min. This selection is rational, and it is based on the results of previous efforts reporting the best accuracy metrics for a time window between 15 min and 1 h depending on the systems (Gujrati et al., 2007). FTRP divides the total compute nodes into two mutually exclusive pools—

**Table 2 Base settings of the experiments**

| Trace | $N_{\text{nodes}}$ | MTBF (s) | $T_{\text{IDL}}$(h) | $I$ (s) | $N_{\text{s}}$ | Prec | Rec | $C_{\text{chk}}$ (s) | $C_{\text{r}}$ (s) | $C_{\text{rep}}$ (s) | $N_{\text{stride}}$ | $T_{\text{down}}$ (s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PNNL08 | 940 | 48 817 | 480 | 1800 | 5 | 0.7 | 0.7 | 300 | 300 | 120 | 1 | 1200 |
| THNSCC | 5632 | 6427 | 336 | 1800 | 20 | 0.7 | 0.7 | 600 | 600 | 120 | 1 | 1200 |

Prec: precision; Rec: recall

the application pool and the replica pool. In our experiments, nodes in the replica pool are about 0.5% of the total nodes. We show the experimental results on the node number changes of the replica pool in Section 7.4. The experimental results are influenced by the accuracy of the failure predictor, and we detail the impact of prediction accuracy in Section 7.1. Thus, in our experiments, unless otherwise specified, both precision and recall of the failure predictors are set as 0.7.

For FTRP, we counted and analyzed the effect of the number of failure hits on the application efficiency. A process $P_i$ running on a failure node will normally incur the whole application crash, but this will not happen if there is a corresponding replica process $P_i'$ in the replica pool which was prefetched. When process $P_i$ fails, $P_i'$ replaces it and the application survives; this is called 'failure hit' in the replica pool. It is important to note that the processes that have been replicated by the replication action do not count. The failure hit includes only the processes on the failure nodes that have not been predicted but have been prefetched.

Fig. 7 shows the running process of PRP2. The trace-based simulator scans the failure traces in time order and simulates prediction results based on the setting of the recall and precision parameters. The simulator randomly reports a certain number of actual failures according to the recall. Due to the definition of precision, suppose that the predictor has reported a total of $N_{\text{actual}}$ actual failures. It randomly gives $N_{\text{actual}} \cdot (1/\text{Precision} - 1)$ false alarms in the traces.

Because different failure traces have different characteristics including failure locality, the experimental results for both PNNL08 and THNSCC traces are given. To avoid the impact of randomness in the experiments, we randomly chose the start time of experiments using a random number generator, and took the average results by running each experiment 60 times as the final results. Thereafter, both replica nodes for FTRP and spare nodes for FT-Pro are referred to as redundant nodes.

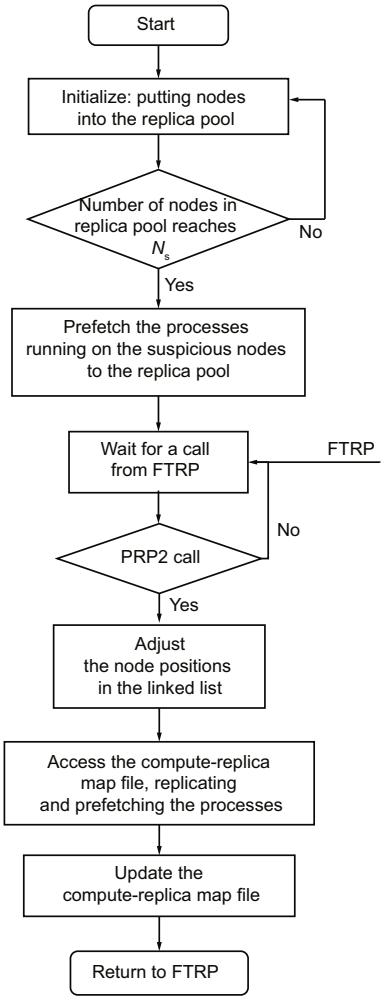


**Fig. 7 Running process of PRP2**

## 7.1 Impact of prediction accuracy

It is clear that the failure prediction accuracy influences the application efficiency of FTRP. The more accurate a prediction mechanism is, the better the application efficiency that FTRP can achieve. In this subsection, we detail the results of several experiments simulating different levels of prediction accuracies.

Figs. 8 and 9 show the application efficiency obtained by FTRP for two traces, where precision

and recall range from 0.1 to 0.9. For PNNL08 traces, the computation scale is set to 940 nodes including five redundant nodes; for THNSCC traces, the computation scale is set to 5632 nodes including 20 redundant nodes.

The experimental results of two failure traces have some differences due to the distinct failure features, but they show many similarities. In Figs. 8 and 9, the results clearly show that the more accurate a predictor is, the higher the application efficiency of FTRP is. For PNNL08 traces in Fig. 8, as long as the recall and precision are more than 0.1, FTRP can achieve a better efficiency than periodic checkpointing. The best efficiency in experiments for PNNL08 is 95.94% when precision=recall=0.9. For THNSCC traces in Fig. 9, as long as the recall is more than 0.3 and the precision is more than 0.1, FTRP can achieve a better efficiency than periodic checkpointing. The best efficiency in experiments for THNSCC is 80.19% when precision=recall=0.9. With the advantages of low replication overheads and supports of process prefetching, FTRP can achieve a high efficiency at low precision and recall levels.

It can be found that FTRP is more sensitive to recall than to precision. This phenomenon exists in both traces and is more noticeable in PNNL08 traces. A high efficiency can be achieved for a high recall even with a low precision, but if the recall is low, the efficiency is poor even for a high precision, as shown in Figs. 8 and 9. The above observation provides an interesting hint for future work on failure predictors. Recall is the focus of the predictor, more important than precision for failure prediction. This finding is consistent with those in Lan and Li (2008)

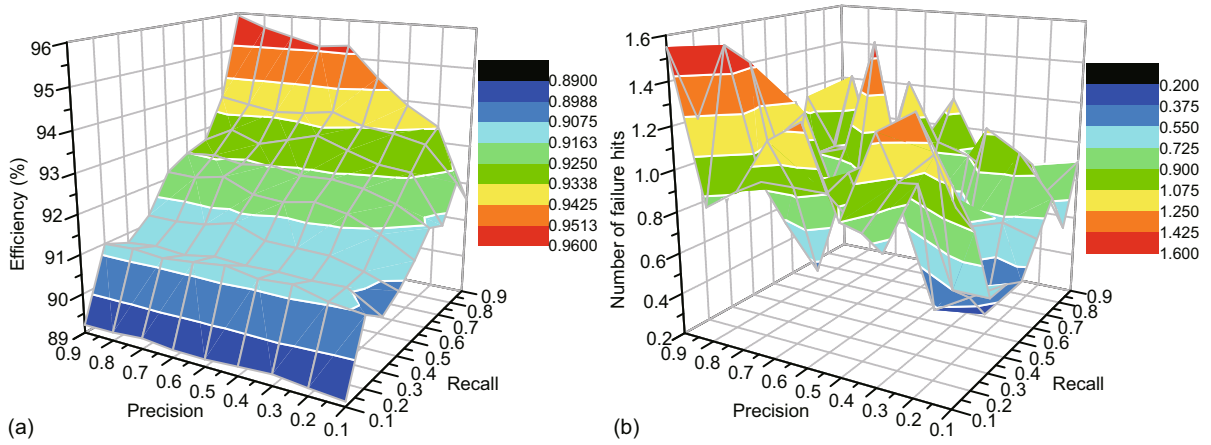


**Fig. 8 Impact of prediction accuracy using PNNL08 traces: (a) application efficiency under different precision and recall; (b) corresponding number of failure hits in the replica pool**

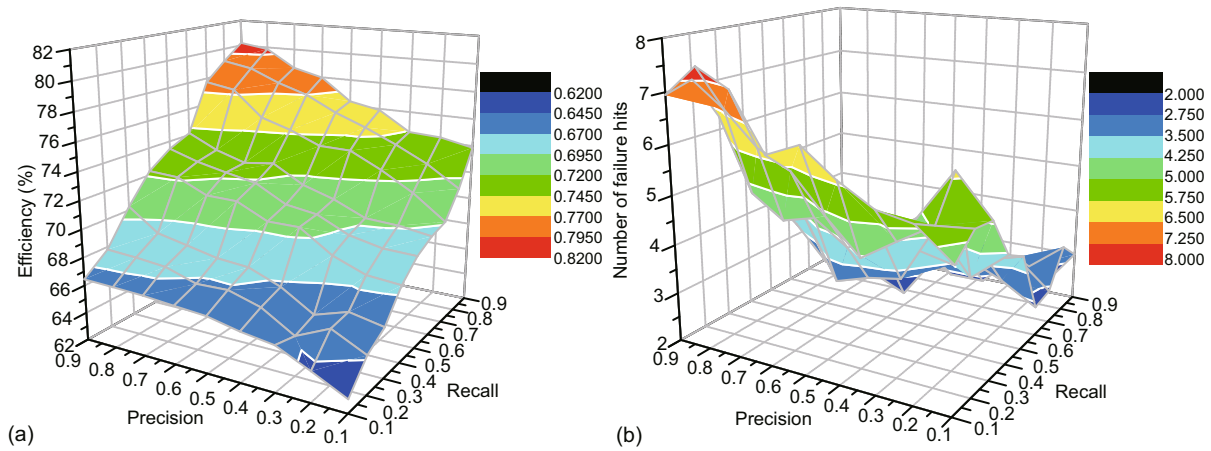


**Fig. 9 Impact of prediction accuracy using THNSCC traces: (a) application efficiency under different precision and recall; (b) corresponding number of failure hits in the replica pool**

and George and Vadhiyar (2015).

In this study, the process prefetching relieves the sensitivity between application efficiency and prediction accuracy including precision and recall. Figs. 8b and 9b show that the failure hits in the replica pool have their own features related to recall and precision. The number of failure hits increases when the recall decreases and precision increases. Obviously, when recall reduces, it means that the number of predicted failures out of the total failures reduces. There are more chances for the prefetched processes in the replica pool to replace the failure process and make the applications survive. When precision decreases, the WM cost model will invoke more PRP2 actions. This will lead to excessive chances for the replica pool to update the prefetched processes, which may reduce the failure locality in the replica pool and also the number of failure hits. As a result, with the support of process prefetching and failure hits in the replica pool, the efficiency can be improved when recall is less than 1.

### 7.2 Impact of replication overhead

In this subsection, we investigate the impact of replication overhead on the application efficiency of FTRP. Fig. 10 illustrates the application efficiency obtained by FTRP for two traces, where the replication overhead ranges from 60 s, 120 s, 240 s, and 480 s to 960 s. The other settings are the same as those given in Table 2. Obviously, a more efficient replication mechanism can serve to obtain a better application efficiency. For PNNL08 traces, even when the replication overhead is 3.2 times the checkpointing overhead, FTRP still achieves 4% more efficiency than periodic checkpointing. For THNSCC traces, when the replication overhead is 0.8 times the checkpointing overhead, FTRP can achieve only 0.7% more efficiency than periodic checkpointing. When the replication overhead is 1.6 times the checkpointing overhead, FTRP efficiency falls behind periodic checkpointing. For different traces, we find that reducing the replication overhead is more helpful in achieving a high application efficiency.

With an increasing replication overhead, the PRP2 action can be disabled more often by the node failure occurings during the execution of the PRP2 action. This situation may result in more work losses. Nonetheless, Fig. 10 shows that the number of failure hits increases with the help of avoiding or compensating for the losses caused by the invalidation of PRP2 actions.

### 7.3 Impact of stride length

From these results, we can obviously see that both process replication and process prefetching can effectively use the redundant nodes in the replica pool to improve the application efficiency of FTRP. We discuss the relationship between the number of failure hits and failure locality parameters. The number of nodes in the replica pool will be used to test the impact of the LBWs on the number of failure hits in Section 7.4. Therefore, we have completed a set of experiments to test the impact of stride length on the number of failure hits.

Stride length is the relative physical distance between the failure nodes. Process prefetching in

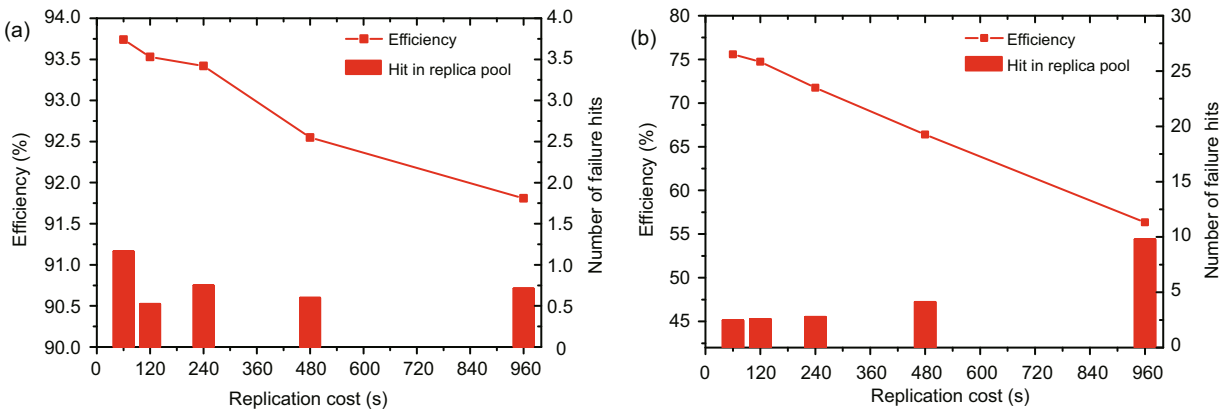


**Fig. 10 Impact of replication overhead when the replication overhead ranges from 60 s, 120 s, 240 s, and 480 s to 960 s: (a) PNNL08 traces; (b) THNSCC traces**

the replica pool uses stride length as the node prefetching range. For PNNL08 traces, because there are few redundant nodes, we conducted experiments using only a stride length of 1–2. For THNSCC traces, we used a stride length of 1–3. Fig. 11 shows that when the stride length increases, the application efficiency of FTRP does not improve in most cases, because for failure locality, the stride length should not be considered the adjacent location in the physical space, but the adjacent location in the network topology. This requires the guidance of fault propagation theory. When a supercomputer system becomes larger, the number of failures increases. In this situation, a single linked list cannot effectively preserve the failure locality, and the prefetched processes that can be hit in the future are easily pushed out. Therefore, it is better to establish multiple linked lists to manage the replica nodes and prefetched processes according to the region of the network topology. How to mine the locality of node failures, considering some affecting factors such as network topology and failure types, is the subject of our ongoing research.

### 7.4 Application efficiency comparison

In this set of simulations, we compared the application efficiency among different fault tolerance mechanisms including FTRP, FT-Pro, and periodic checkpointing. The FTRP was allocated different node counts in the replica pool, and FT-Pro was given a corresponding number of spare nodes.

Fig. 12 shows the efficiency results achieved by FTRP over FT-Pro and periodic checkpointing, where the number of redundant nodes $N_s$ ranges from 1 to 30 for PNNL08 traces (Fig. 12a) and from 1 to 300 for THNSCC traces (Fig. 12b). In Figs. 12a and 12b, it can be seen that the efficiency of FTRP and FT-Pro initially keeps up with the increase of the number of redundant nodes, and then starts

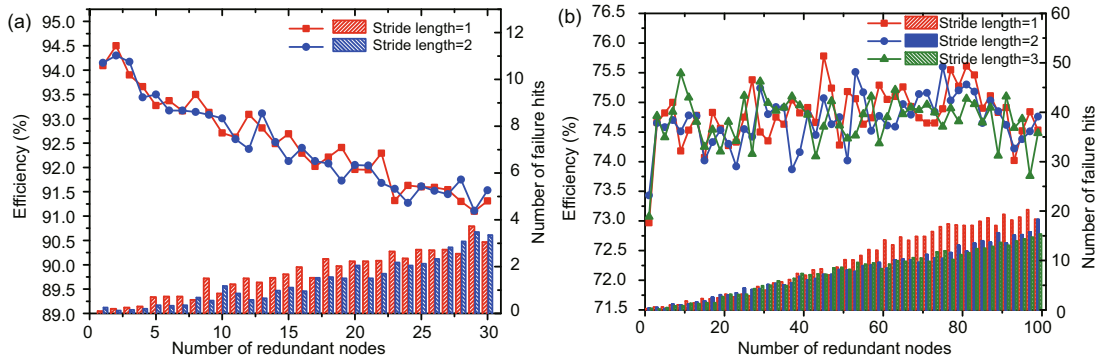


**Fig. 11 Impact of stride length**

Due to the map sheet limit, compared with (a), (b) illustrates only the results of odd redundant node numbers

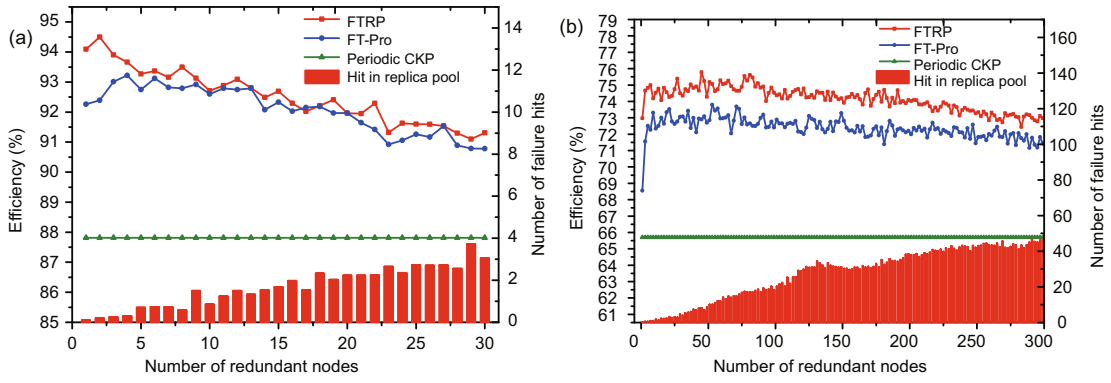


**Fig. 12 Application efficiency comparison among three fault tolerance mechanisms including FTRP, FT-Pro, and periodic checkpointing: (a) PNNL08 traces; (b) THNSCC traces**

declining slowly. Allocating more redundant nodes does not always increase the overall efficiency, because the number of nodes used for real work decreases beyond a certain point and the benefits from the fault tolerance of redundant nodes cannot cover the real work losses. The efficiency of periodic checkpointing is a straight line because it is an average result and has nothing to do with the redundant nodes. When more redundant nodes are allocated, FTRP acquires more failure hits in the replica pool due to process prefetching. FTRP can save application execution time because it uses the WM cost model to choose the appropriate fault tolerance mechanism. Also, PRP2 including the process prefetching mechanism can reduce work losses, which the above mechanisms cannot do.

The columns in Fig. 12 represent the numbers of failure hits in the replica pool. They monotonically grow with the increasing number of redundant nodes, because more redundant nodes can use more failure localities. The number of failure hits rapidly grows with the increase of the number of redundant nodes when the number of redundant nodes is small, and the trend subsequently slows down with a continuous growth of the number of redundant nodes. The growth of the replica pool provides chances for failure hits due to the failure locality, but the failure locality is limited and cannot keep up with the scaling of the replica pool, and thus excessive redundant nodes will lead to reduced efficiency. Fig. 12 reveals the relationship between the efficiency and the scale of the replica pool. We usually choose 0.5%–1% of the total nodes as the redundant nodes. According to these results, in other experiments, we set the scale of the replica pool to 0.5% of the total nodes and sometimes round for convenience.

In Fig. 12, by allocating about 0.5% of the total nodes as redundant nodes, in terms of application efficiency, FTRP outperforms periodic checkpointing by 6.8% and FT-Pro by 2.1% in PNNL08 traces. In THNSCC traces, the FTRP mechanism outperforms periodic checkpointing by 10.4% and FT-Pro by 3.1%.

### 7.5 Application efficiency on different computation scales

Parallel applications have rapidly scaled with supercomputer development. In this set of experiments, we evaluated the application efficiency of FTRP on different computation scales in the THNSCC traces. We tuned the number of compute nodes from 704 to 5632 (the maximum node number in THNSCC traces is 5632) with the number of redundant nodes in the replica pool from 4 to 28 (0.5% of the total nodes).

The redundant nodes are still a small proportion of the total compute nodes. Assume that the replication overhead $C_{\mathrm{rep}}$ is 2 min for this set of experiments. With the scaling of the application size, the checkpointing overhead $C_{\mathrm{chk}}$ and recovery overhead $C_{\mathrm{r}}$ change correspondingly. These settings are complicated because they are closely related to the application and the system. Considering the checkpointing principle and the parallel file system (PFS) performance features, assume that $C_{\mathrm{chk}}$ meets the power function $C_{\mathrm{chk}} = a \cdot N_{\mathrm{nodes}}^{b} + c$ ($a$, $b$, and $c$ are constants). The parameters are detailed in Table 3. When the application scales, the data to the checkpoint increase while the aggregating I/O performance may initially increase to a peak and then decline (Alam et al., 2007; Fahey et al., 2008), and thus the checkpointing overhead $C_{\mathrm{chk}}$ may initially increase slowly and then increase sharply.

**Table 3 Settings of experiments on different computation scales**

| $N_{\mathrm{nodes}}$ | $N_{\mathrm{s}}$ | $C_{\mathrm{chk}}$(s) | $C_{\mathrm{r}}$(s) | $C_{\mathrm{rep}}$(s) | Precision | Recall |
|---|---|---|---|---|---|---|
| 704 | 4 | 240 | 240 | 120 | 0.7 | 0.7 |
| 1408 | 7 | 253 | 253 | | | |
| 2112 | 11 | 275 | 275 | | | |
| 2816 | 14 | 308 | 308 | | | |
| 3520 | 18 | 360 | 360 | | | |
| 4224 | 21 | 430 | 430 | | | |
| 4928 | 25 | 510 | 510 | | | |
| 5632 | 28 | 600 | 600 | | | |

Fig. 13 shows the application efficiency of FTRP against the other two mechanisms. It can be seen that FTRP significantly outperforms the other two mechanisms at all application scales. The application efficiency of FTRP is 5%–13% higher than that of periodic checkpointing and 2%–5% higher than that of FT-Pro. There are two interesting features: (1) The application efficiency of all the three mechanisms declines when the computation scale increases. This is rational because more failures occur when more nodes are used. Fig. 13 shows that the application efficiency of FTRP

declines more slowly than those of the other two mechanisms, which means FTRP has more potential to deal with larger computation scales. (2) The number of failure hits in the replica pool increases with the increase of the application scale. Maintaining an appropriate number of nodes to replicate and prefetch the process is an effective method for improving the application efficiency of fault tolerance. To raise the number of failure hits, simple and effective replica pool management is the focus of follow-up study.

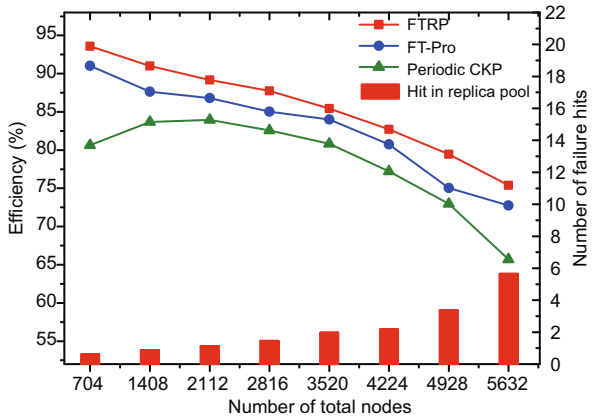


**Fig. 13 Application efficiency on different computation scales**

## 8 Conclusions

In this paper, we have presented a new fault tolerance framework (FTRP) combining the benefits of proactive and reactive mechanisms. The WM cost model has been provided to give the appropriate fault tolerance choice at runtime. Failure locality has been defined and analyzed in supercomputers for the first time, and a new proactive fault tolerance mechanism PRP2 has been presented using process replication and process prefetching based on failure locality. Simulations with real failure traces have shown that our framework outperforms existing fault tolerance mechanisms significantly: (1) FTRP requires a less accurate failure prediction with the help of PRP2 and failure locality (FTRP outperforms periodic checkpointing when the precision is more than 0.1 and the recall is more than 0.1 or 0.3 for different systems). The process prefetching method using failure locality effectively improves the flexibility of the framework. (2) An allocation of redundant nodes (0.5%–1%) is efficient for FTRP to reach the above performances. (3) FTRP has the potential to achieve a high performance for larger supercomputer systems.

In the future, more work needs to be done: (1) We will further investigate the principles of failure locality and propagation; (2) We plan to develop a new management mechanism for the replica pool to fully use the potential of failure locality when the system becomes larger; (3) We will combine FTRP with our ongoing failure prediction research, and evaluate the whole fault tolerance system on large production systems.

## References

Alam SR, Kuehn JA, Barrett RF, et al., 2007. Cray XT4: an early evaluation for petascale scientific simulation. Proc ACM/IEEE Conf on Supercomputing, p.1-12. https://doi.org/10.1145/1362622.1362675

Babaoglu O, Joy W, 1981. Converting a swap-based system to do paging in an architecture lacking page-referenced bits. Proc 8th ACM Symp on Operating Systems Principles, p.78-86. https://doi.org/10.1145/800216.806595

Bhatele A, Jetley P, Gahvari H, et al., 2011. Architectural constraints to attain 1 exaflop/s for three scientific application classes. Proc IEEE Int Parallel & Distributed Processing Symp, p.80-91, https://doi.org/10.1109/IPDPS.2011.18

Bouguerra MS, Gainaru A, Gomez LB, et al., 2013. Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing. IEEE 27th Int Symp on Parallel Distributed Processing, p.501-512. https://doi.org/10.1109/IPDPS.2013.74

Brown D, Smith G, 2008. MPP2 Syslog Data (2006-2008). Technical Report, PNNL-SA-61371.

Cappello F, Casanova H, Robert Y, 2010. Checkpointing vs. migration for post-petascale supercomputers. Proc 39th Int Conf on Parallel Processing, p.168-177. https://doi.org/10.1109/ICPP.2010.26

Daly JT, 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. *Fut Gener Comput Syst*, 22(3):303-312. https://doi.org/10.1016/j.future.2004.11.016

Denning PJ, 2005. The locality principle. *Commun ACM*, 48(7):19-24. https://doi.org/10.1145/1070838.1070856

Dwork C, Lynch N, Stockmeyer L, 1988. Consensus in the presence of partial synchrony. *J ACM*, 35(2):288-323. https://doi.org/10.1145/42282.42283

Egwutuoha IP, Levy D, Selic B, et al., 2013. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *J Supercomput*, 65(3):1302-1326. https://doi.org/10.1007/s11227-013-0884-0

Elliott J, Kharbas K, Fiala D, et al., 2012. Combining partial redundancy and checkpointing for HPC. IEEE 32nd Int Conf on Distributed Computing Systems, p.615-626. https://doi.org/10.1109/ICDCS.2012.56

Elnozahy ENM, Alvisi L, Wang YM, et al., 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput Surv*, 34(3):375-408. https://doi.org/10.1145/568522.568525

Fahey M, Larkin J, Adams J, 2008. I/O performance on a massively parallel Cray XT3/XT4. IEEE Int Symp on Parallel and Distributed Processing, p.1-12. https://doi.org/10.1109/IPDPS.2008.4536270

Ferreira K, Stearley J, Laros JH III, et al., 2011. Evaluating the viability of process replication reliability for exascale systems. Proc Int Conf for High Performance Computing, Networking, Storage and Analysis, Article 44. https://doi.org/10.1145/2063384.2063443

Gainaru A, Cappello F, Kramer W, 2012a. Taming of the shrew: modeling the normal and faulty behaviour of large-scale HPC systems. IEEE 26th Int Symp on Parallel Distributed Processing, p.1168-1179. https://doi.org/10.1109/IPDPS.2012.107

Gainaru A, Cappello F, Snir M, et al., 2012b. Fault prediction under the microscope: a closer look into HPC systems. Proc Int Conf on High Performance Computing, Networking, Storage and Analysis, Article 77. https://doi.org/10.1109/SC.2012.57

George C, Vadhiyar S, 2012. ADFT: an adaptive framework for fault tolerance on large scale systems using application malleability. *Proc Comput Sci*, 9:166-175.

George C, Vadhiyar S, 2015. Fault tolerance on large scale systems using adaptive process replication. *IEEE Trans Comput*, 64(8):2213-2225. https://doi.org/10.1109/TC.2014.2360536

Gujrati P, Li Y, Lan Z, et al., 2007. A meta-learning failure predictor for blue gene/l systems. Proc Int Conf on Parallel Processing, p.1-8. https://doi.org/10.1109/ICPP.2007.9

Gupta S, Xiang P, Yang Y, et al., 2013. Locality principle revisited: a probability-based quantitative approach. *J Parall Distrib Comput*, 73(7):1011-1027. https://doi.org/10.1016/j.jpdc.2013.01.010

Hamerly G, Elkan C, 2001. Bayesian approaches to failure prediction for disk drives. Proc 18th Int Conf on Machine Learning, p.202-209.

Hargrove PH, Duell JC, 2006. Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters. *J Phys Conf Ser*, 46(1):494. https://doi.org/10.1088/1742-6596/46/1/067

Hellerstein JL, Zhang F, Shahabuddin P, 2001. A statistical approach to predictive detection. *Comput Netw*, 35(1):77-95. https://doi.org/10.1016/S1389-1286(00)00151-1

Hu W, Jiang Y, Liu G, et al., 2015. DDC: Distributed Data Collection Framework for Failure Prediction in Tianhe Supercomputers. Springer International Publishing, p.18-32. https://doi.org/10.1007/978-3-319-23216-4_2

Kalaiselvi S, Rajaraman V, 2000. A survey of checkpointing algorithms for parallel and distributed computers. *Sadhana*, 25(5):489-510. https://doi.org/10.1007/BF02703630

Lan Z, Li Y, 2008. Adaptive fault management of parallel applications for high-performance computing. *IEEE Trans Comput*, 57(12):1647-1660. https://doi.org/10.1109/TC.2008.90

Lan Z, Gu J, Zheng Z, et al., 2010. A study of dynamic meta-learning for failure prediction in large-scale systems. *J Parall Distrib Comput*, 70(6):630-643. https://doi.org/10.1016/j.jpdc.2010.03.003

Liang Y, Zhang Y, Jette M, et al., 2006. Bluegene/l failure analysis and prediction models. Int Conf on Dependable Systems and Networks, p.425-434. https://doi.org/10.1109/DSN.2006.18

Lu CD, 2005. Scalable Diskless Checkpointing for Large Parallel Systems. PhD Thesis, Champaign, IL, USA.

Mohammed A, Kavuri R, Upadhyaya N, 2012. Fault tolerance: case study. Proc 2nd Int Conf on Computational Science, Engineering and Information Technology, p.138-144. https://doi.org/10.1145/2393216.2393240

Mohror K, Moody A, de Supinski BR, 2012. Asynchronous checkpoint migration with MRNet in the Scalable Checkpoint/Restart Library. IEEE/IFIP Int Conf on Dependable Systems and Networks Workshops, p.1-6. https://doi.org/10.1109/DSNW.2012.6264668

Moody A, Bronevetsky G, Mohror K, et al., 2010. Design, modeling, and evaluation of a scalable multi-level checkpointing system. Proc ACM/IEEE Int Conf for High Performance Computing, Networking, Storage and Analysis, p.1-11. https://doi.org/10.1109/SC.2010.18

Pinheiro E, Weber WD, Barroso LA, 2007. Failure trends in a large disk drive population. Proc 5th USENIX Conf on File and Storage Technologies, p.2.

Plank JS, Beck M, Kingsley G, et al., 1995. Libckpt: transparent checkpointing under Unix. Proc USENIX Technical Conf Proc, p.18.

Plank JS, Li K, Puening MA, 1998. Diskless checkpointing. *IEEE Trans Parall Distrib Syst*, 9(10):972-986. https://doi.org/10.1109/71.730527

Roman E, 2002. A survey of checkpoint/restart implementations. Technical Report LBNL-54942, Lawrence Berkeley National Laboratory.

Sahoo RK, Oliner AJ, Rish I, et al., 2003. Critical event prediction for proactive management in large-scale computer clusters. Proc 9th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining, p.426-435. https://doi.org/10.1145/956750.956799

Salfner F, Lenk M, Malek M, 2010. A survey of online failure prediction methods. *ACM Comput Surv*, 42(3):10.1-10.42. https://doi.org/10.1145/1670679.1670680

Sancho JC, Petrini F, Johnson G, et al., 2004. On the feasibility of incremental checkpointing for scientific computing. Proc 18th Int Symp on Parallel and Distributed Processing Symp, p.58. https://doi.org/10.1109/IPDPS.2004.1302982

Schroeder B, Pinheiro E, Weber WD, 2009. DRAM errors in the wild: a large-scale field study. Proc 11th Int Joint Conf on Measurement and Modeling of Computer Systems, p.193-204. https://doi.org/10.1145/2492101.1555372

Vetter JS, Mueller F, 2003. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J Parall Distrib Comput*, 63(9):853-865. https://doi.org/10.1016/S0743-7315(03)00104-7

Vilalta R, Ma S, 2002. Predicting rare events in temporal domains. Proc IEEE Int Conf on Data Mining, p.474-481. https://doi.org/10.1109/ICDM.2002.1183991

Weinberg J, McCracken MO, Strohmaier E, et al., 2005. Quantifying locality in the memory access patterns of HPC applications. Proc ACM/IEEE Conf on Supercomputing, p.50. https://doi.org/10.1109/SC.2005.59

Young JW, 1974. A first order approximation to the optimum checkpoint interval. *Commun ACM*, 17(9):530-531. https://doi.org/10.1145/361147.361115

Zhong Y, Shen X, Ding C, 2009. Program locality analysis using reuse distance. *ACM Trans Program Lang Syst*, 31(6), Article 20. https://doi.org/10.1145/1552309.1552310