Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 166 – 175

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

# ADFT: An Adaptive Framework for Fault Tolerance on Large Scale Systems using Application Malleability

Cijo George[a,1], Sathish S. Vadhiyar[a]

[a] *Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India*

**Abstract**

Exascale systems of the future are predicted to have mean time between failures (MTBF) of less than one hour. Malleable applications, where the number of processors on which the applications execute can be changed during executions, can make use of their malleability to better tolerate high failure rates. We present ADFT, an adaptive fault tolerance framework for long running malleable applications to maximize application performance in the presence of failures. ADFT framework includes cost models for evaluating the benefits of various fault tolerance actions including checkpointing, live-migration and rescheduling, and runtime decisions for dynamically selecting the fault tolerance actions at different points of application execution to maximize performance. Simulations with real and synthetic failure traces show that our approach outperforms existing fault tolerance mechanisms for malleable applications yielding up to 23% improvement in application performance, and is effective even for petascale systems and beyond.

*Keywords:* Fault Tolerance, HPC, Malleable Parallel Applications, Large Scale Systems, Rescheduling.

## 1. Introduction

With the development of high performance systems with massive number of processors [1] and long running scalable scientific applications that can use large number of processors for executions [2, 3], the mean time between failures (MTBF) of the processors used for a single application execution has tremendously decreased [4]. Current petascale systems are reported to have MTBFs of less than 10 hours [5, 6], and future exascale systems are anticipated to have MTBFs of less than an hour [6]. Hence, it is highly imperative to develop efficient fault tolerance strategies to sustain executions of long-running real scientific applications on future large and very large scale systems.

Most of the traditional fault tolerance techniques including checkpointing [7, 8], and live process migration [9] resume the application on the same number of processors after failures. Choosing a good static number of processors for execution is difficult in large scale systems like peta and exa scale systems where the number of processors available at a given point of time widely varies throughout application execution due to the very low MTBFs on these systems. Some checkpointing systems support the development of parallel applications that can change the number of processors during execution [10, 11]. For example, this is achieved in SRS [10] by instrumenting the application with SRS calls for specifying data for checkpointing, along with the distribution to processors. We refer to these

*Email addresses:* cijo@ssl.serc.iisc.in (Cijo George), vss@serc.iisc.in (Sathish S. Vadhiyar)
[1] Corresponding author

1877-0509 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.procs.2012.04.018

applications as *malleable* applications, and the action of changing the number of processors of a malleable application during execution as *rescheduling*.

With the development of these different fault tolerance strategies, the selection of a strategy for application execution has to be carefully made to maximize the application performance in the presence of failures. Depending on failure predictions and the cost of different strategies, a runtime system may have to dynamically select the most cost-effective fault tolerance strategy at a given instance of application execution. In this work, we have developed AdFT, an adaptive fault tolerance framework for long running malleable applications to maximize application performance in the presence of failures. We have developed cost models that consider different factors like accuracy of failure predictions and application scalability, for evaluating the benefits of various fault tolerance actions. Our adaptive framework uses the cost models to make runtime decisions for dynamically selecting fault tolerance actions at different points of application execution to maximize performance.

The primary focus of our work is to evaluate the benefits of malleability for real scientific applications on very large scale systems and to develop an effective strategy for fault tolerance in future systems. While AdFT makes use of malleability for better fault tolerance and performance for malleable applications, it can also be used for non-malleable applications for adaptive fault tolerance.

Using simulations, we evaluate AdFT in terms of work done per unit time by the application in the presence of failures. Our results show that AdFT involving malleability outperforms the popular periodic checkpointing approach by at least 21%, and also yields up to 23% higher amount of work than a dynamic strategy, called *FT-Pro* [12], that does not involve malleability. Our results also show that our adaptive strategy yields high performance even for petascale systems and beyond, and that application malleability will be highly essential for future exascale systems.

In Section 2, we present related efforts in fault tolerance strategies. Section 3 gives the overall methodology of AdFT framework. In Section 4, we describe in detail the cost model for evaluating the benefits of various fault tolerance strategies. Section 5 explains the fault tolerance simulator used for evaluations. In Section 6, we describe our evaluation methodology, experiments with real and synthetic traces and applications on large scale systems, and give salient observations. Section 7 gives a summary of our work and presents scope for future work.

## 2. Related Work

Most of the fault tolerance mechanisms are based on checkpointing [7, 8]. Recently, there has been increasing interest in live process migration [9] due to its lower overhead in transferring the process images when compared to the high cost of checkpointing. To help a runtime system to use these fault tolerance mechanisms, failure predictors have been developed to predict failures based on event logs of systems, using data mining techniques [13, 14].

Cappello et al. [15] have also analyzed fault tolerance for post petascale systems. The work compares proactive migration with proactive checkpointing based on analytical performance models. The analysis is based on the assumption of having a perfect failure predictor with 100% accuracy. While their work gives overall statistics using the assumption, our work performs actual simulations of application progress in realistic scenarios with prediction errors.

*FT-Pro:* The work by Lan and Li [12] has also developed an adaptive fault management framework similar to the focus of our work. Their *FT-Pro* framework provides fault tolerance for applications by performing proactive migration or checkpointing based on a cost model. However, their work confines to non-malleable applications that execute on a fixed number of processors throughout application execution. The cost model of *FT-Pro* is not capable of taking advantage of malleability of applications to provide better fault tolerance. Considering malleability involves the following significant challenges to developing a cost model and framework.

- Malleable applications can recover instantly from a failure by changing the number of processors. Hence there can be multiple failure-rollback cycles in the same time interval. *FT-Pro* assumes a single application failure in an interval, which makes it unsuitable for malleable applications.
- For malleable applications, the time required to complete a given amount of work depends on the number of processors used. *FT-Pro* assumes this time to be a constant for a given amount of work.
- Since malleable applications execute on different number of processors during execution, the application scalability on varying number of processors has to be considered in the cost model. *FT-Pro* does not support the use of application scalability.

AdFT uses an entirely new cost model that addresses all the above challenges. Moreover, *FT-Pro* is evaluated for short running applications using stochastic modeling for a maximum of 192 nodes and trace based simulations for a maximum of 64 nodes. Our evaluation of AdFT is much more comprehensive, using real traces from LANL for 512 and 1024 nodes and also using synthetic traces for very large scale systems up to exascale. We use synthetic scalability curves as well as the scalability curves of real long running applications for the simulations.

## 3. AdFT Framework

We assume the presence of a *failure predictor* [13, 14] that can periodically estimate expected node failures in the system in the near future. *Precision* ($P$) of such a predictor is defined as the ratio of the number of correct predictions to the total number of predictions made and *Recall* ($R$) is defined as the ratio of the number of correct predictions to the total number of failures. The higher the values of $P$ and $R$, the better the predictor.

The overall working of our framework is illustrated in Figure 1. AdFT takes runtime fault tolerance actions at decision making points, denoted as *adaptation points* ($AP$), such that the application performs constant amount of work, $W$, between two consecutive $AP$s. Following are the possible actions that can be taken.

- *SKIP*, where no action is taken.
- *CHECKPOINT*, where the application takes a proactive checkpoint. We assume coordinated checkpointing, where the processes synchronize to perform checkpointing.
- *MIGRATE*, where the processes on failure-prone nodes are migrated to healthy nodes. We assume that live-migration method [9], which does not involve checkpointing, is used for the purpose.
- *RESCHEDULE*, where the application is rescheduled to a different set of nodes, which does not include any failure prone node (*proactive rescheduling*). This action can be taken only if the application is malleable.

## 4. A Cost Model for Application Execution between Adaptation Points

AdFT uses a cost model that takes into account failure prediction *accuracy metrics* (precision and recall), operation costs of the fault tolerance actions, number of available nodes and *application scalability* data to select the best fault tolerance action at each $AP$. Application scalability is expressed in the form of work done per unit time on various number of processors. AdFT uses this data to compute the following two variables in our cost model.

- $N(n)$: The number of nodes $p \leq n$, corresponding to maximum work done by the application per unit time.
- $T(w, n)$: The time taken by the application to perform $w$ units of work on $n$ nodes. This is obtained by dividing $w$ with the work done per unit time for $n$ nodes obtained from the application scalability data.

At each $AP$, the failure predictor forecasts the expected node failures for the next time interval $I$, where $I$ is the estimated time to complete $W$ amount of work using the current working set of nodes, $N_w$, in a failure-free environment (given by $I = T(W, N_w)$). AdFT uses the cost model to compute $E_{next}$, the expected time to complete the next $W$ amount of work and thus reach the next $AP$, for each possible fault tolerance action. The action with minimum value of $E_{next}$ is selected.

### 4.1. Illustration: Cost Model for 3 Nodes

We assume that malleable applications can be recovered instantly from node failures by rescheduling to a different set of nodes (*reactive rescheduling*). Suppose at $AP_i$, the predictor predicts that nodes $A$, $B$ and $C$ are prone to failures in the next time interval $I$. The worst case in which these nodes can fail is as follows.

- Node $A$ fails when the application is about to reach $AP_{i+1}$.
- Node $B$ fails after the application recovers from failure of node $A$ and is about to reach $AP_{i+1}$ again.

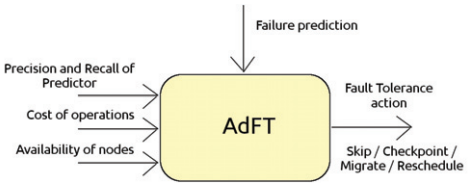



Figure 1: AdFT Framework

| Scenario | Probability |
|---|---|
| All three of the nodes A, B and C fail | $P^3$ |
| Any two of A, B and C nodes fail | ${}^3C_2 * P^2 * (1-P)$ |
| Any one of A, B and C nodes fail | ${}^3C_1 * P^1 * (1-P)^2$ |
| None of the nodes fails | $(1-P)^3$ |

Table 1: Failure Scenarios (Example)

- Node $C$ fails after the application recovers from failure of node $B$ and is about to reach $AP_{i+1}$ again.

If a *SKIP* action was taken at $AP_i$, the total time taken to reach $AP_{i+1}$ in this example can be expressed as follows.

$$E_{next} = T(W, N_w) + [T_{resch} + T_{recover} + T((W_{lost} + W), N(N_w - 1 + N_s))] + [T_{resch} + T_{recover} + T((W_{lost} + W), N(N_w - 2 + N_s))] + [T_{resch} + T_{recover} + T((W_{lost} + W), N(N_w - 3 + N_s))] \quad (1)$$

The application takes $T(W, N_w)$ time to perform $W$ amount of work on the current working set of $N_w$ number of nodes, to reach $AP_{i+1}$. At this point one of the 3 nodes fails and the application spends $T_{resch}$ time for reactive rescheduling and $T_{recover}$ time for recovering the application on the new set of $(N_w - 1 + N_s)$ number of nodes. The new set of nodes is obtained by excluding the node that has failed and including $N_s$ number of spare nodes. The application then spends $T((W_{lost} + W), N(N_w - 1 + N_s))$ time to reach $AP_{i+1}$ using the new set of nodes. Here, $W_{lost}$ is the work done between the last checkpoint and $AP_i$, calculated as $(AP_{current} - AP_{ckp}) * W$, where $AP_{current}$ is the index of the current $AP$ and $AP_{ckp}$ is the index of the latest $AP$ where a checkpoint was taken. When the application almost reaches $AP_{i+1}$, the second node fails and similar costs are involved to reach $AP_{i+1}$ again. Then the third node fails and the process repeats. Hence the total cost is as given in Equation (1), which can be simplified and expressed as follows.

$$E_{next} = T(W, N_w) + 3 * (T_{resch} + T_{recover}) + \sum_{j=1}^{3} T((W_{lost} + W), N(N_w - j + N_s)) \quad (2)$$

Equation (2) gives the time taken to reach $AP_{i+1}$ if a SKIP decision was taken at $AP_i$ and all 3 nodes predicted to fail actually fails and in the worst possible way, which is just one possible scenario. Table 1 shows all the possible scenarios and their corresponding probabilities if 3 nodes are predicted to fail. Note that the probability that a given node which is predicted to fail will actually fail is equal to precision, $P$, of the predictor.

In general, the probability that $i$ nodes out of the 3 nodes which are predicted to fail will actually fail is given by ${}^3C_i * P^i * (1-P)^{3-i}$. Now the estimated cost of the *SKIP* decision in the given example considering all scenarios can be expressed as given below.

$$E_{next} = \sum_{i=1}^{3} {}^3C_i * P^i * (1-P)^{3-i} * [T(W, N_w) + i * (T_{resch} + T_{recover}) + \sum_{j=1}^{i} T((W_{lost} + W), N(N_w - j + N_s))] + (1-P)^3 * [T(W, N_w)] \quad (3)$$

Similarly, the cost model is developed for different actions for $N_f$ number of predicted node failures.

### 4.2. A General Cost Model

At each $AP$, AdFT computes the estimated cost of each of the possible actions using the cost model given below and takes the action that has the least estimated value of $E_{next}$.

- *SKIP*: Depending on the number of nodes that fail, the application may have to perform rollback recovery several times. If none of the nodes fail, no extra costs are incurred and the time taken to reach the next $AP$ will be the time taken to complete $W$ amount of work using $N_w$ nodes. Hence, $E_{next}$ is computed as follows.

$$E_{next} = \sum_{i=1}^{N_f} {}^{N_f}C_i * P^i * (1-P)^{N_f-i} * [T(W, N_w) + i * (T_{resch} + T_{recover}) + \sum_{j=1}^{i} T((W_{lost} + W), N(N_w - j + N_s))] + (1-P)^{N_f} * [T(W, N_w)] \quad (4)$$

  Equation (4) is a simple adaptation of Equation (3) for $N_f$ number of nodes. $\sum_{i=1}^{N_f} {}^{N_f}C_i * P^i * (1-P)^{N_f-i}$ is the probability that the application will fail and $(1-P)^{N_f}$ is the probability that the application will not fail.

- *CHECKPOINT*: The application spends some time for checkpointing at the beginning of the next interval. $E_{next}$ is computed as follows.

$$E_{next} = \sum_{i=1}^{N_f} {}^{N_f}C_i * P^i * (1-P)^{N_f-i} * [T_{ckp} + T(W, N_w) + i * (T_{resch} + T_{recover}) + \sum_{j=1}^{i} T(W, N(N_w - j + N_s))] + (1-P)^{N_f} * [T_{ckp} + T(W, N_w)] \quad (5)$$

  If the application fails, the cost involved will be the sum of the time for checkpointing, $T_{ckp}$, the time to reach the next adaptation point, $T(W, N_w)$, the cost of rescheduling ($T_{resch}$) and recovery ($T_{recover}$) for each of the node failures and the time taken to redo the work to reach the next $AP$ for each of the node failures, $T(W, N(N_w - j + N_s))$. If the application does not fail, the cost will be the sum of $T_{ckp}$ and the time to reach the next $AP$, $T(W, N_w)$.

- *MIGRATE*: In this case, *live-migration* is performed at the beginning of the next interval. There are two possible scenarios.

  1. if $N_f <= N_s$, i.e. the number of nodes predicted to fail is less than the number of spare nodes, all failure prone nodes can be migrated to healthy spare nodes and hence failure probability will be *ZERO*.
  2. if $N_f > N_s$, only $N_s$ number of failure prone nodes can be migrated to healthy nodes. Hence there is still a failure probability involving $N_f - N_s$ nodes.

  The above two conditions are taken care of by defining a variable, $N_{fm}$, such that if $N_f <= N_s$, $N_{fm} = 0$, else $N_{fm} = N_f - N_s$. $E_{next}$ is then computed as follows.

$$E_{next} = \sum_{i=1}^{N_{fm}} {}^{N_{fm}}C_i * P^i * (1-P)^{N_{fm}-i} * [T_{mig} + T(W, N_w) + i * (T_{resch} + T_{recover}) + \sum_{j=1}^{i} T((W_{lost} + W), N(N_w - j + N_s))] + (1-P)^{N_f} * [T_{mig} + T(W, N_w)] \quad (6)$$

  If the application fails, the cost involved will be the sum of the time for migration, $T_{mig}$, the time to reach the next adaptation point, $T(W, N_w)$, the cost of rescheduling ($T_{resch}$) and recovery ($T_{recover}$) for each of the node

failures and the time taken to redo the work to reach the next $AP$ for each of the node failures, $T((W_{lost} + W), N(N_w - j + N_s))$. The latter time includes $W_{lost}$ since no checkpoint is taken for live-migration at the current $AP$. If the application does not fail, the cost will be the sum of $T_{mig}$ and the time to reach the next $AP$.

- *RESCHEDULE*: Here, the probability of application failure is *ZERO* since the application is rescheduled, avoiding all failure prone nodes. Hence, the cost involves only the overhead for rescheduling and the time taken to complete $W$ amount of work using the new set of nodes. $E_{next}$ is computed as follows.

$$E_{next} = T_{ckp} + T_{resch} + T_{recover} + T(W, N(N_w - N_f + N_s)) \quad (7)$$

The above cost model relies on precision, $P$ of the predictor. But, for a predictor with a recall, $R$, of less than 1, there can also be failures which are not predicted. To tolerate such unforeseen failures, a *precautionary checkpoint* is taken when the time since last checkpoint reaches a threshold. For a given $R$ value, the time interval between such unpredicted failures can be estimated as $\frac{MTBF}{1-R}$. This value is taken as the threshold for precautionary checkpointing.

## 5. Failure Simulator

For evaluating ADFT, we have developed a robust discrete-event failure simulator that can simulate application execution in the presence of failures. It takes as input, node failure-recovery trace of a system, accuracy metrics of the failure predictor, type of fault tolerance to be adopted, application scalability data and other data including cost of each of the fault tolerance operations and estimated MTBF of the system. It simulates application execution based on the given inputs and considering application malleability. The simulator outputs the work done per unit time by the application at the end of the simulation, along with other details of the application behavior in the presence of failures.

In the absence of real traces, the trace generation component of the simulator can generate synthetic traces with failure times of different distributions including Weibull and Exponential distributions and repair times of Log-normal distribution for the simulation. The failure prediction component in the simulator can take expected predictor accuracy metrics as input and simulate the behavior of a failure predictor that estimates at regular intervals of time, the list of nodes that might fail in the next interval, with the given accuracy metrics.

## 6. Experiments and Results

ADFT is evaluated against *FT-Pro* and periodic checkpointing with checkpointing interval that gives maximum performace, based on simulations of application execution using our failure simulator. Node failure-recovery trace of the system considered, application scalability data and accuracy metrics of the failure predictor are given as input to the simulator. For fair comparison, we have extended *FT-Pro* [12] to consider the scalability of applications to decide what number of nodes out of the available nodes should be used for execution for best performance. *FT-Pro* takes a precautionary checkpoint when the number of consecutive *SKIP* decisions reaches a threshold. This is based on the assumption that *SKIP* decision is the only one that does not involve checkpointing. Since we consider *live* migration that does not involve checkpointing, we have also modified *FT-Pro* such that it takes a precautionary checkpoint whenever the time since last checkpoint reaches a threshold. This makes the precautionary checkpointing strategy of *FT-Pro* similar to that of ADFT.

*FT-Pro* requires allocation of a constant number of spare nodes so that the application can be executed on the constant remaining number of nodes in the system till completion. In our analysis, we have found that the optimal spare node allocation for maximum performance can vary based on various factors including number of simultaneous node failures, scalability curve of the application etc. For the purpose of our evaluation, we allocate the number of spare nodes equal to the average of the number of failed nodes (or nodes that were down) at any point of time in the failure history before the time when the simulated application starts execution. This is to make sure that there are enough spare nodes to exercise the option of process migration while avoiding high spare node allocation to reduce the amount of idling in the system.

For our experiments, we set $W$, which is the constant work to be completed between each $AP$, as the work done by the application in 30 minutes in a failure free environment. This is based on the results in previous efforts on failure

Table 2: AdFT vs other methods (Linear Scalability, P=R=0.7)

| $N_{nodes}$ | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|
| 512 (LANL) | 481.96 | 23.22 | 156.38 |
| 1024 (LANL) | 642.33 | 8.70 | 77.57 |
| 16384 (Synthetic) | 10038.62 | 15.16 | 87.27 |

Table 3: Fault Tolerance Actions taken by AdFT (Linear Scalability, P=R=0.7)

| $N_{nodes}$ | $MTBF(hrs)$ | $N_{skip}$ | $N_{ckp}$ | $N_{mig}$ | $N_{p.resch}$ | $N_{pre.ckp}$ | $N_{r.resch}$ |
|---|---|---|---|---|---|---|---|
| 512 (LANL) | 23.95 | 0 | 0 | 16 | 3 | 5 | 4 |
| 1024 (LANL) | 5.31 | 0 | 0 | 101 | 24 | 17 | 24 |
| 16384 (Synthetic) | 10.86 | 0 | 0 | 64 | 7 | 7 | 11 |

predictors [13] that report the best accuracy metrics for a time window between 15 minutes to 1 hour depending on the system. MTBF of the system, which is used for precautionary checkpointing is taken as the observed MTBF from the trace history. We also assume the following costs for the various fault tolerance actions: $T_{ckp}$ : 5 minutes, $T_{mig}$ : 0.33 minutes, $T_{resch}$ : 3 minutes, $T_{recover}$ : 5 minutes. These are in accordance with the values given for the 2011 cost scenario in [15].

Real traces from LANL [16] corresponding to system 20 which is a 512-node system and system 18, which is a 1024-node system are used for simulations of small and medium scale systems. For simulations of very large scale systems for which real failure traces are not available, we generate synthetic traces using our simulator for different number of nodes based on the observation in [17] that the times to failure of nodes in a system follows a Weibull probability distribution and the times to recover follow a Log-normal probability distribution.

Simulations are done for a period of 30 days and evaluation is based on the work done by the application in unit time. For the LANL traces, a random year is chosen from the trace of a system for simulation. Application execution is simulated for the last month of the one year trace. Observed MTBF of the system and the number of spare nodes to be allotted for *FT-Pro* is obtained using the trace history of the previous eleven months. A similar strategy is also adopted for synthetic traces, where a trace is generated for one year and simulations are performed for the last month of the year.

### *6.1.* AdFT *Performance on Small and Medium Scale Systems*

Simulations on 512 and 1024 nodes are done using real traces from LANL. A synthetic trace is generated for 16384 nodes with an MTBF of approximately 10 hours. We assume a synthetic application with linear and perfect scalability, such that the work done per unit time by the application on $N$ nodes in a failure free environment is $N$ units. Precision, $P$ and recall, $R$ of the predictor are assumed to be 0.7.

Table 2 shows the work done per second by AdFT ($WD/s_{(AdFT)}$) and the percentage gain over *FT-Pro* and periodic checkpointing ($\%G_{FTPro}$ and $\%G_{per.ckp}$, respectively). The result shows that AdFT gives 8-23% improvement over *FT-Pro* and more than 87% improvement over periodic checkpointing. Table 3 shows the number of various fault tolerance actions taken by AdFT during the application execution period. In the table, $N_{p.resch}$, $N_{r.resch}$ and $N_{pre.ckp}$ correspond to proactive rescheduling, reactive rescheduling and precautionary checkpointing respectively.

We find that most of the fault tolerance actions are migrations since unlike other actions, live-migration does not involve checkpointing, and incurs much lesser cost (0.33 minutes in our experiments) than the others. A significant percentage of fault tolerance actions are related to rescheduling ($N_{p.resch} + N_{r.resch}$). It is also observed that the number of rescheduling decisions increases as MTBF decreases, which is due to the increased number of proactive rescheduling to avoid failures. The number of reactive rescheduling decisions also increase due to the increase in the number of unanticipated failures. The results show that rescheduling plays an important role in AdFT in adapting to a large scale failure environment with high failure dynamics or low MTBFs.

### *6.2. Spares vs Failures*

We also found that rescheduling also contributes to increase in application performance in an indirect way. It helps in adaptively maintaining sufficient number of spare nodes most of the time during application execution. Figure 2

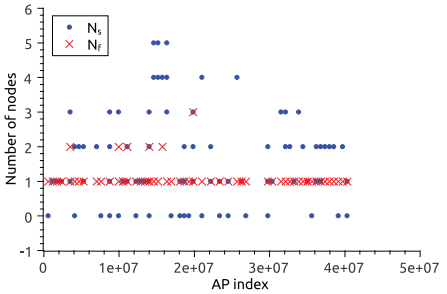



Figure 2: $N_s$ vs $N_f$ for AdFT

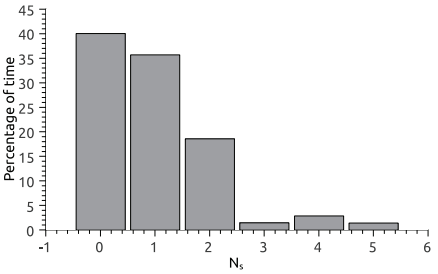



Figure 3: Analysis of number of idle nodes for AdFT during the entire simulation period (LANL-1024)

| $P$ | $R$ | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|---|
| 1.0 | 1.0 | 1017.82 | 0.31 | 181.376 |
| 0.8 | 1.0 | 1017.55 | 0.32 | 181.301 |
| 0.6 | 1.0 | 1016.62 | 0.35 | 181.044 |
| 0.4 | 1.0 | 1015.69 | 0.31 | 180.787 |
| 1.0 | 0.8 | 664.04 | 0.56 | 83.5734 |
| 0.8 | 0.8 | 673.17 | 6.19 | 86.0974 |
| 0.6 | 0.8 | 595.35 | 17.31 | 64.5841 |
| 0.4 | 0.8 | 586.65 | 4.94 | 62.179 |

| $P$ | $R$ | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|---|
| 1.0 | 0.6 | 639.00 | 5.20 | 76.6511 |
| 0.8 | 0.6 | 648.26 | 3.24 | 79.211 |
| 0.6 | 0.6 | 652.80 | 4.27 | 88.5108 |
| 0.4 | 0.6 | 651.55 | 6.60 | 80.1205 |
| 1.0 | 0.4 | 701.71 | 1.21 | 96.3675 |
| 0.8 | 0.4 | 680.60 | 0.11 | 88.1514 |
| 0.6 | 0.4 | 681.91 | 2.06 | 88.5135 |
| 0.4 | 0.4 | 633.51 | 1.56 | 77.8647 |

Table 4: Varying Precision and Recall for AdFT

shows the variations in the number of spare nodes and predicted failures with AdFT for 1024 nodes (LANL) at each *AP* where failures are predicted. It can be seen that in majority of the cases, the number of spare nodes in the system is greater than or equal to the number of predicted failures. This helps increase the number of low-cost migration decisions to avoid failures, hence improving performance significantly.

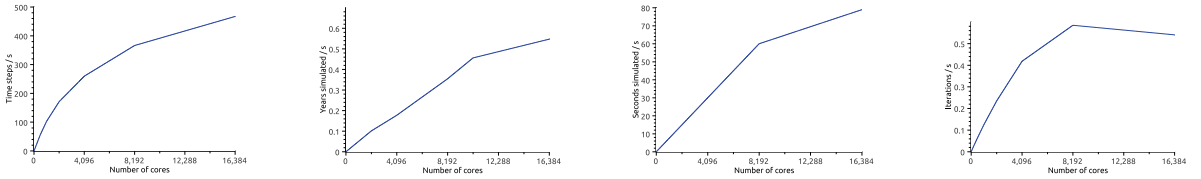
### *6.3. Resource Utilization*

Rescheduling also helps in utilizing spare nodes in the system that become available after recovering from a failure, for application execution. Whenever proactive or reactive rescheduling is performed, AdFT tries to accommodate the healthy spare nodes in the system.

Figure 3 shows the percentages of time of application execution observed for different idle node numbers for 1024 nodes (LANL). We can observe that for up to 40% of the time there are no idle nodes and for about 94% of the time the number of idle nodes is less than or equal to 2. A similar analysis on *FT-Pro* showed that 99.99% of the time, the number of idle nodes in the system is 3, which is the allotted number of spare nodes. This shows the effectiveness of AdFT in dynamically adapting to failures while keeping the number of idle nodes to a minimum.

### *6.4. Accuracy of Failure Predictions*

Table 4 shows the performance of AdFT for different precision ($P$) and recall ($R$) values of the predictor for 1024 nodes (LANL). It shows that AdFT outperforms periodic checkpointing by a huge margin and also outperforms *FT-Pro* for the given $P$ and $R$ values. However, we have observed that *FT-Pro* performs better by a small margin for $R$ values below 0.2. This is due to the large number of unforeseen failures, which results in large number of reactive rescheduling, incurring huge cost, resulting in *FT-Pro* that does not perform rescheduling giving slightly better performance. It is observed that AdFT gives the most improvement over *FT-Pro* for $R$ values between 0.6 and 0.9. Since failure predictors today have an $R$ value of more than 0.6, it can be concluded that AdFT performs better than *FT-Pro* for all practical purposes.




(a) NAMD (b) CCSM (c) WRF (d) ChaNGa

Figure 4: Scalability of different Applications

Table 5: ADFT vs other methods for different Applications

| *Application* | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|
| NAMD | 286.32 | 15.15 | 87.28 |
| CCSM | 0.34 | 15.13 | 86.37 |
| WRF | 48.42 | 15.26 | 87.47 |
| ChaNGa | 0.461 | -0.21 | 21.22 |

Table 6: ADFT vs other methods for a Petascale system

| $N_{nodes}$ | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|
| $2^{17}$ | 802.96 | 11.21 | 145.79 |

Table 7: ADFT vs other methods for an Exascale system

| $N_{nodes}$ | $WD/s_{(AdFT)}$ | $\%G_{FTPro}$ | $\%G_{per.ckp}$ |
|---|---|---|---|
| $2^{23}$ | 3065.11 | 12.5 | 21 |

### 6.5. Real Applications

Simulations were also done using scalability data of four real life applications as observed in real large scale systems. The simulations correspond to application execution on 16384 nodes using a synthetic failure trace. The application scalability curves used are that of NAMD [18], CCSM [19], WRF [20] and ChaNGa [21]. Figure 4 shows the scalability curves of the applications as observed in BlueGene/L and their corresponding units of work done.

As shown in Table 5, ADFT performs much better than periodic checkpointing for all the applications and gives about 15% better performance than *FT-Pro* for NAMD, CCSM and WRF. But, *FT-Pro* shows slightly better performance in case of ChaNGa. This can be attributed to the modification done to *FT-Pro* that allows it to start the application on the number of nodes which gives maximum performance. It can be observed from the scalability curve of ChaNGa in Figure 4 that the application performance decreases after 8192 nodes. Hence, the modified *FT-Pro* technique will start the application only on 8192 nodes, leaving a large number of spare nodes, allowing it to perform low-cost live migrations at all *AP*s, resulting in performance improvement over ADFT by a small margin.

### 6.6. Petascale and Exascale Systems

Simulation of NAMD is also done for a hypothetical petascale system with $2^{17}$ nodes and a hypothetical exascale system with $2^{23}$ nodes. For the petascale system, each node is assumed to have approximately 7.6 GFlops/s peak performance. For the exascale system, we assume that each node is quad-core, so that the total number of processors is $2^{25}$. Each processor is assumed to have a peak performance of approximately 29.8 GFlops/s, which is approximated assuming that the approximate ratio of the average processor peak performance of an exascale system to that of a petascale system will be approximately equal to a similar ratio between a petascale system and a terascale system.

The scalability curve for NAMD was obtained from a study on BlueGene/L system [18] with each node having approximately 2.7 GFlops/s peak performance. Approximate scalability curve for the petascale system was generated based on the assumption that the work done by a node in the hypothetical petascale system will be approximately equal to the work done by 3 nodes of BlueGene/L. Scalability curve for the exascale system is generated in a similar way. We generated synthetic traces with MTBF of approximately 4 hours for petascale system as reported in [22, 6] and 35 minutes for exascale system, as reported in [6].

Results in Tables 6 and 7 show that ADFT outperforms periodic checkpointing by about 145% and about 21% for petascale and exascale systems, respectively. It also outperforms *FT-Pro* by about 11% and about 12.5% for petascale and exascale systems, respectively. It is observed that in exascale system, ADFT performs significantly more number of migrations than *FT-Pro* (406 against 300), which again shows its effectiveness in dynamically maintaining enough spare nodes to maximize low-cost migrations. Also, the number of rescheduling increases drastically from 51 to 364 when moving from petascale to exascale, showing that application malleability plays an important role in the

performance of AdFT and that with increasing size of the systems and decreasing MTBF, application malleability and rescheduling can play a very important role in developing better fault tolerance strategies.

## 7. Conclusions and Future Work

In this work, we have developed AdFT, an adaptive framework that makes runtime decisions on fault tolerance techniques at different points of application execution. Our framework considers application malleability and exploits the benefits of rescheduling for fault tolerance in such applications. Evaluations based on simulations showed that our strategy involving malleability outperforms the popular but static periodic checkpointing approach by at least 21%, and also yields up to 23% higher amount of work than the dynamic *FT-Pro* strategy that does not involve malleability. Our results also show that our adaptive strategy yields high performance even for petascale systems and beyond. We also showed that application malleability will have to be considered strongly for future exascale systems.

In future, we plan to develop a fault management software suite that will consist of the fault management framework discussed in this paper, tools for performing various fault tolerance actions, and techniques that give failure predictions. We also plan to enhance our failure simulator to study alternate fault tolerance options for future systems.

## References


[1] Top 500 Supercomputing Sites, `http://www.top500.org/`.

[2] L. Oliker, A. Canning, J. Carter, C. lancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, T. Goodale, Scientific Application Performance on Candidate PetaScale Platforms, in: IPDPS '07: Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1–12.

[3] A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W. D. Gropp, L. V. Kale, Architectural Constraints to Attain 1 Exaflop/s for Three Scientific Application Classes, in: IPDPS '11: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, 2011.

[4] F. Petrini, K. Davis, J. Sancho, System-Level Fault-Tolerance in Large-Scale Parallel Machines with Buffered Coscheduling, in: IPDPS '04: Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium, 2004, pp. 209–.

[5] N. R. Adiga, et al., An Overview of the BlueGene/L Supercomputer, in: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, 2002.

[6] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, Exascale Computing Study: Technology Challenges in Achieving Exascale Systems, (P. Kogge, Editor and Study Lead) (2008).

[7] J. S. Plank, An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance, technical Report, University of Tennessee Knoxville, TN, USA (1997).

[8] J. Ansel, K. Arya, G. Cooperman, DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop, in: IPDPS '09: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, 2009.

[9] C. Wang, F. Mueller, C. Engelmann, S. L. Scott, Proactive process-level live migration in HPC environments, in: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008.

[10] S. Vadhiyar, J. Dongarra, SRS - A Framework for Developing Malleable and Migratable Parallel Applications for Distributed Systems, Parallel Processing Letters 13 (2) (2003) 291–312.

[11] G. Zheng, L. Shi, L. V. Kale, FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI, in: CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing, 2004, pp. 93–103.

[12] Z. Lan, Y. Li, Adaptive Fault Management of Parallel Applications for High-Performance Computing, IEEE Transactions on Computers 57 (12).

[13] P. Gujrati, Y. Li, Z. Lan, R. Thakur, J. White, A Meta-Learning Failure Predictor for Blue Gene/L Systems, in: ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing, 2007.

[14] N. Nakka, A. Agrawal, A. Choudhary, Predicting Node Failure in High Performance Computing Systems from Failure and Usage Logs, in: IPDPS '11: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, 2011.

[15] F. Cappello, H. Casanova, Y. Robert, Checkpointing vs. Migration for Post-Petascale Supercomputers, in: ICPP '10 Proceedings of the 2010 39th International Conference on Parallel Processing, 2010.

[16] Failure Trace Archive, `http://fta.inria.fr/apache2-default/pmwiki/index.php?n=Main.DataSets/`.

[17] B. Schroeder, G. Gibson, A Large-scale Study of Failures in High-Performance Computing Systems, in: Proceedings of the International Conference on Dependable Systems and Networks (DSN2006), 2006.

[18] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, L. V. Kale, Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms, in: IPDPS '08: Proceedings of the 22rd IEEE International Parallel and Distributed Processing Symposium, 2008.

[19] J. M. Dennis, R. Jacob, M. Vertenstein, T. Craig, R. Loy, Toward an Ultra-High Resolution Community Climate System Model for the Bluegene Platform, Journal of Physics: Conference Series 78.

[20] J. Michalakes, J. Hacker, R. Loft, M. O. McCracken, A. Snavely, N. J. Wright, T. Spelce, R. Walkup, B. Gorda, WRF Nature Run, in: Supercomputing '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 2007.

[21] F. Gioachin, P. Jetley, C. L. Mendes, L. V. Kale, T. Quinn, Towards Petascale Cosmological Simulations with ChaNGa, technical Report, Parallel Programming Laboratory, University of Illinois (2007).

[22] F. Cappello, Resilience: One of the main challenges for Exascale Computing, INRIA Illinois Joint-Laboratory on Petascale computing.