Adaptive Fault Management of Parallel Applications for High Performance Computing

Zhiling Lan, *Member, IEEE*, and Yawei Li, *Student Member, IEEE*

***Abstract*—As the scale of high performance computing (HPC) continues to grow, application fault resilience becomes crucial. In this paper, we present FT-Pro, an adaptive fault management approach that combines proactive migration with reactive checkpointing. It aims to enable parallel applications to avoid anticipated failures via preventive migration, and in the case of unforeseeable failures, to minimize their impact through selective checkpointing. An adaptation manager is designed to make runtime decisions in response to failure prediction. Extensive experiments, by means of stochastic modeling and case studies with real applications, indicate that FT-Pro outperforms periodic checkpointing, in terms of reducing application completion times and improving resource utilization, by up to 43%.**

***Index Terms*—Adaptive fault management, Parallel applications, High performance computing, Large-scale systems.**

I. Introduction

IN the field of high performance computing (HPC), the insatiable demand for more computational power in science and engineering has driven the development of ever-growing supercomputers. Production systems with hundreds of thousands of processors, ranging from tightly-coupled proprietary clusters to loosely-coupled commodity-based clusters, are being designed and deployed [1]. For systems of this scale, *reliability* becomes a critical concern as the system-wide mean time between failures (MTBF) decreases dramatically with the increasing count of components. Studies have shown that MTBFs for teraflop- and petaflop-scale systems are only on the order of 10-100 hours, even for systems based on ultra-reliable components [2], [3]. Meanwhile, to accurately model realistic problems, parallel applications are designed to span across a substantial number of processors for days or weeks until completion. Unfortunately, the current state of parallel processing is such that the failure of a single process usually aborts the entire application. As a consequence, large applications find it difficult to make any forward progress because of failures. This situation is likely to deteriorate as systems get bigger while applications become larger.

Checkpointing is the conventional method for fault tolerance. It is reactive by periodically saving a snapshot of the application and using it for restarting the execution in case of failures [4], [5]. When one of the application processes experiences a failure, all the processes, including non-faulty processes, have to roll back to the previously saved state prior to the failure. Thus, significant performance loss can be incurred due to the work loss and failure recovery. Unlike checkpointing, the newly emerged *proactive approach* (e.g. process migration) takes preventive actions before failures, thereby preventing failure experiencing and avoiding rollbacks [6], [7]. Nevertheless, it requires accurate fault prediction, which is hardly achievable in practice. Hence, proactive approach alone is unlikely sufficient to provide a reliable solution for fault management in HPC.

In this paper, we present *FT-Pro, an adaptive approach* for fault management of parallel applications by combining the merits of proactive migration and reactive checkpointing. Proactive actions enable applications to avoid anticipated faults if possible, and reactive actions intend to minimize the impact of unforeseeable failures. The goal is to reduce application completion time in the presence of failures. While checkpointing and process migration have been studied extensively, the key challenge facing the design of FT-Pro is *how to effectively select an appropriate action at runtime*. Towards this end, an adaptation manager is designed to choose a best-fit action from opportunistic skip, reactive checkpointing, and preemptive migration by considering a number of factors.

We demonstrate that FT-Pro can enhance fault resilience of parallel applications and consequently improve their performance, by means of stochastic modeling and case studies with parallel applications. Our results indicate that FT-Pro outperforms periodic checkpointing, in terms of reducing application completion time and improving resource utilization, by up to 43%. A modest allocation of spare nodes (less than 5%) is usually sufficient for FT-Pro to achieve the above gain. Additionally, the overhead caused by FT-Pro is less than 3%.

FT-Pro is built on the belief that technological innovation combined with advanced data analysis makes it possible to predict failures with a certain degree of accuracy. Recent studies with actual failure traces have shown that with a proper system monitoring facility, critical events can be predicted with an accuracy of up to 80% [8]–[12]. A distinguishing feature of FT-Pro is that it does not require perfect failure prediction to be effective. As we will show, FT-Pro outperforms periodic checkpointing as long as failure prediction is capable of capturing 30% of failures, which is feasible by using existing predictive technologies.

FT-Pro is intended to bridge the gap between failure prediction and fault handling techniques by effectively exploring failure prediction for better fault management. It complements the research on checkpointing and process migration by providing adaptive strategies for runtime coordination of these techniques. The proposed FT-Pro can be integrated with state-

Manuscript received April 25, 2007; revised August 14, 2007; accepted December 4, 2007.

Zhiling Lan is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. Email: lan@iit.edu

Yawei Li is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. Email: liyawei@iit.edu

of-the-art failure predictors and existing fault tolerance tools [9], [10], [13]–[20] to provide an end-to-end system for adaptive fault management of parallel applications.

The remainder of this paper is organized as follows. Section II briefly discusses the related work. Section III gives an overview of FT-Pro, followed by a detailed description of its adaptation manager in Section IV. Section V describes our stochastic modeling and simulation results. Section VI presents case studies with a number of parallel applications. Finally, Section VII summarizes the paper and points out future directions.

II. Related Work

Checkpointing, in various forms, has been studied extensively over the past decades. A detailed description and comparison of different checkpointing techniques can be found in [4]. In the field of HPC, a number of checkpointing libraries and tools have been developed, and examples include libckpt [16], BLCR [17], open MPI [18], MPICH-V [13], and the C^3 (Cornell Checkpoint (pre)Compiler) [15]. Checkpointing optimization is generally approached by selecting an optimal intervals [21]–[23] or reducing the overhead per operation such as copy-on-write [24], incremental checkpointing [25], diskless checkpointing [26], [27], double in-memory technique [28], etc. Generally speaking, checkpointing is a conservative method. It requires increasing number of checkpoints to deal with higher failure rates as the computing scale increases.

Much progress has been made in failure analysis and prediction. On the hardware side, modern computer systems are designed with various features (e.g. hardware sensors) that can monitor the degradation of an attribute over time for early detection of hardware errors [29]–[32]. On the software side, a variety of predictive techniques have been developed to infer implicit and useful fault patterns from historical data for failure prediction. They can be broadly classified as *model-based* or *data-driven*. A model-based approach derives an analytical or probabilistic model of the system and then triggers a warning when a deviation from the model is detected [33]–[37]. Data mining, in combination with intelligent systems, focuses on learning and classifying fault patterns without assuming a priori model ahead of time [8]–[11]. In addition, leading HPC vendors have started to integrate hardware and software components in their systems for fault analysis, such as the Cluster Monitoring and Control System (CMCS) service in IBM Blue Gene systems and Cray RAS and Management System (CRMS) in Cray XT series systems [38], [39].

Leveraging the research on failure prediction, there are growing interests in utilizing failure prediction for proactive fault management. For example, the HA-OSCAR project provides high-availability for head nodes in Beowulf clusters by using a failover strategy [40]. There are several research efforts on failure-aware scheduling [41], [42]. Process or object migration is a widely used proactive technique [7]. Most migration methods adopt the *stop-and-restart* model for the migration of parallel applications, in which the application takes a checkpoint and then restarts on a new set of resources - after swapping the failure-prone nodes with healthy ones [43]. There are several active research projects on providing *live migration* support for MPI applications [19], [20], [44]. While proactive approach is cost efficient, it requires accurate failure prediction. In practice, prediction misses and false alarms are common. Prediction misses can lead to significant performance damage, whereas false alarms can introduce intolerable overhead. Hence, solely relying on proactive approach is not sufficient for HPC.

Recognizing the limitations of reactive and proactive approaches, FT-Pro aims at getting the best of both worlds by intelligently coordinating process migration with checkpointing. Similar to cooperative checkpointing [45], FT-Pro ignores unnecessary fault tolerance requests when failure impact is trivial. Further, it enables an application to avoid imminent failures through preventive migration. The adaptation between process migration and selective checkpointing is built upon a quantitative modeling of application performance.

The idea of using adaptation for fault management is not new. It has been used in the fields such as mission-critical spacecrafts and storage systems [46], [47]. Nevertheless, to the best of our knowledge, we are among the first to exploit adaptive fault management for high performance computing. Different from the above research that mainly focuses on efficiently utilizing duplicated components for high availability, this work centers upon reducing application completion time by dynamically choosing between proactive and reactive actions.

III. Overview of FT-Pro

We define a *failure* as any event in hardware or software that results in an immediate termination of a running application. To be effective, FT-Pro requires the presence of a failure predictor. Predictive techniques mentioned in Section II, as well as our own previous work [11], [48], [49], can be used to provide such an engine. Failure prediction can be either *categorical* where the predictor forecasts whether a failure event will occur or not in the near future, or *numerical* where the probability of failures is provided for a given time window. Numerical results can be easily translated into categorical results by applying threshold based splitting; hence in this paper, we uniformly describe *failure prediction* as a process that periodically estimates whether a node will experience failures in a given time window (e.g. a few minutes to an hour). Such a prediction mechanism is generally measured by two accuracy metrics: *precision* and *recall*. Precision is defined as the proportion of correct predictions to all the predictions made (i.e. $\frac{T_p}{T_p + F_p}$), and recall is the proportion of correct predictions to the number of failures (i.e. $\frac{T_p}{T_p + F_n}$). Here, T_p is number of correct predictions (i.e. *true positives*), and F_p is number of false alarms (i.e. *false positives*), and F_n is number of missed failures (i.e. *false negatives*). Obviously, a good prediction engine should achieve a high value (closer to 1.0) for both metrics.

A user can set fault tolerance requests denoted as *adaptation points* for the application execution, and FT-Pro makes runtime decision upon these points to determine which action should be taken [50]. For example, a user may set adaptation points to where the application completes a segment of useful

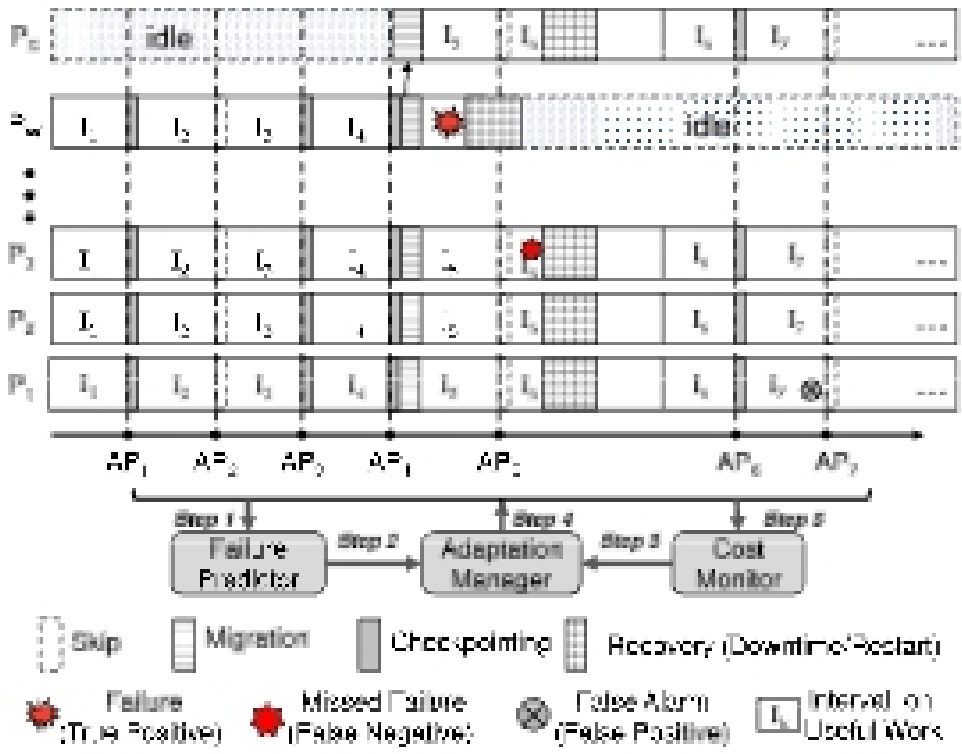

Fig. 1. The Main Idea and Steps of FT-Pro.

work, i.e. the computation that is not redone due to a failure [51]. Three actions are currently considered in FT-Pro:

- *SKIP*, where the fault tolerant request is ignored. This action is taken to remove unnecessary actions when failure impact is trivial.
- *CHECKPOINT*, where the application takes a checkpoint. This action is to reduce application work loss caused by unforeseeable failures.
- *MIGRATION*, where the processes on suspicious nodes (i.e. the nodes predicted to be failure-prone in the near future) are transferred to healthy nodes. This action is to avoid an imminent failure. Currently, we assume that process migration is conducted by taking a coordinated checkpoint followed by a stop-and-restart migration [43].

The main idea of FT-Pro is illustrated in Figure 1, where the useful work is segmented into intervals denoted by I_k. Suppose the application runs on $\{P_1, P_2, \ldots, Pw\}$ and one spare node denoted as P_S is allocated for proactive actions. Spare nodes can be either reserved at the application submission or allocated through the resource manager during execution. Upon each adaptation point AP_i, FT-Pro first consults *the failure predictor* to get the status of each computation node. It then triggers *the adaptation manager* (discussed in Section IV) to determine a best-fit action in response to failure prediction, followed by invoking the corresponding action on the application. Here, *the cost monitor component* keeps track of runtime overhead different fault tolerance actions. If the application fails during checkpointing or migration, it rolls back to the most recent checkpoint. Let's take a look at a few examples:

- FT-Pro always grants the first fault tolerance request at AP_1 by taking a checkpoint.
- At AP_2 where the failure predictor does not anticipate any failure in the near future, given that failure impact during the next interval is trivial, FT-Pro ignores the request by taking a SKIP action. Similarly, FT-Pro takes a SKIP action at AP_7.
- At AP_3, considering that the work loss would be significant if an unforeseeable failure occurred in the next interval, FT-Pro decides to take a coordinated checkpoint although no failure warning is issued at this point.
- At AP_4 where the predictor forecasts a failure on P_W (which turns out to be a true positive), FT-Pro transfers the process from P_W to the spare node P_S. The application is first checkpointed, followed by a process migration. Once repaired, P_W becomes a spare node.
- At AP_5 where the predictor fails to warn the upcoming failure on P_3 (e.g. a false negative), FT-Pro takes a SKIP action. The application, therefore, loses the work done between AP_5 and the failure occurrence, suffers from failure recovery, rolls back to the last checkpoint completed at AP_4, recomputes the work due to the failure, and proceeds to the next adaptation point AP_6.
- In case of false alarms, such as at AP_6 where the predictor erroneously gives a warning (e.g. a false positive), FT-Pro takes a checkpoint.

Fig. 2. Diagram of Adaptation Algorithm

IV. Adaptation Manager

The adaptation manager is responsible for determining the most suitable action upon each adaptation point. Designing an efficient manager is challenging. First, it must consider a range of factors that may impact application performance. These include not only the available spare nodes, but also costs and benefits of different fault tolerance actions. Second, given that a failure predictor is subjected to false negatives and false positives, it must take account of both errors during its decision making process. Lastly, it must make a timely decision without causing noticeable overhead on application performance.

By considering the above requirements, we develop an adaptation manager, which is illustrated in Figure 2. It takes account of three sets of parameters for decision making, namely prediction accuracy, operation costs of different actions, and the number of available resources for proactive actions. Before presenting our detailed algorithm, we first list a set of nomenclatures that will be frequently used in the rest of the paper (see Table I).

Upon each adaptation point AP_i, if the failure predictor anticipates any failure on a computation node, the manager takes account of prediction *precision*. Specifically, it estimates E_{next} - the expected time for the application to reach the next adaptation point AP_{i+1} - and *selects the action that minimizes* E_{next}. Suppose the current interval index is $I_{current}$. Due to the uncertainty of the exact failure time, a conservative view is adopted by assuming that the failure will occur imme-