Enhancing Application Robustness through Adaptive Fault Tolerance[*]

Zhiling Lan, Yawei Li, Ziming Zheng, and Prashasta Gujrati
Illinois Institute of Technology, Department of Computer Science
{lan,liyawei,zzheng11,gujrpra}@iit.edu

Abstract

As the scale of high performance computing (HPC) continues to grow, application fault resilience becomes crucial. To address this problem, we are working on the design of an adaptive fault tolerance system for HPC applications. 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. Both prior and ongoing work are summarized in this paper.

1. Introduction

Over the past decades, the insatiable demand for more computational power in science and engineering has driven the development of ever-growing supercomputers. High performance computing (HPC) systems with hundreds to thousands of processors, ranging from tightly-coupled proprietary systems to loosely-coupled commodity-based clusters, are being designed and deployed [1]. For systems of such scales, *reliability* becomes a critical concern as the system-wide MTBF (mean-time-between-failure) 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 the systems based on ultra-reliable components [2,3].

To accurately model realistic problems, parallel applications are designed to span across a substantial number of processors for days or weeks until completion. Among various parallel paradigms, MPI (Message passing interface) has become the de facto standard for parallel processing in HPC. Unfortunately, the current state is such that the failure of a single process usually aborts the entire MPI application. As a consequence, large applications, in particular tightly-coupled parallel applications, find it difficult to make any forward progress because of failures. This situation is likely to deteriorate as systems get bigger and parallel applications become larger.

Checkpointing is the conventional method for fault tolerance. In essence, 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. Moreover, with the growing gap between processor speed and data access speed, frequent checkpointing can further increase the disparity between sustained performance and peak performance in HPC. Unlike checkpointing, the newly emerged proactive approach (e.g. process migration) takes preventive actions before failures, thereby preventing failure experiencing and avoiding rollbacks [6,7,8]. Nevertheless, it requires accurate fault prediction, which is hardly achievable in practice. As a result, the proactive approach alone is unlikely sufficient to provide a reliable solution for fault management in HPC.

In this report, we present our on-going research project on addressing the above problem by exploring *an adaptive approach*. Proactive techniques are dynamically integrated with reactive methods at runtime, where proactive actions enable applications to avoid anticipated faults if possible and reactive actions intend to minimize the impact of unforeseeable failures. By intelligently coordinating proactive and reactive actions, the proposed system aims at improving fault resilience of parallel applications. Figure 1 shows the proposed adaptive fault management system, along with its major components. Specifically, the contributions of the project include:

1. *Fault forecasting* to improve fault prediction and localization in large-scale systems by investigating ensemble learning and pattern recognition techniques [9,10,23].

[*] This work is supported in part by NSF CNS-0720549.

2. *Adaptation manager* to dynamically select a best-fit fault tolerance action in response to fault forecasting [11,12].
3. *Integrated runtime support* to enable cost-effective integration and enhancement of fault tolerance techniques according to adaptation manager [13,14].

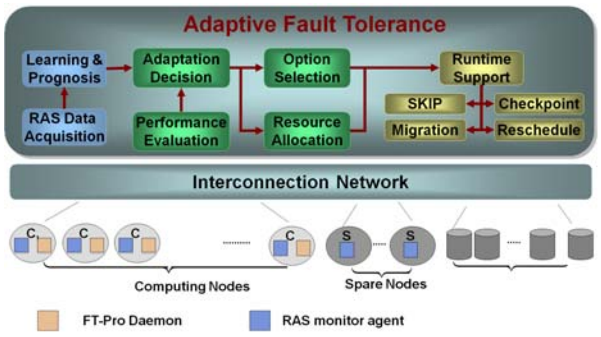

Figure 1. Overview of Adaptive Fault Tolerance

2. Fault Forecasting

Despite extensive research on various fault tolerance techniques, little work has been done on fault forecasting, especially in large-scale systems such as those used in HPC. The goal of *fault forecasting* is to automatically analyze the data collected by various monitoring tools and then feed the learned information to fault tolerance tools for fast failure recovery and/or timely failure avoidance [9,15-17]. While it is clearly desirable for efficient fault management, it is also challenging. Due to the massive quantities of information collected from a large number of components, fault patterns and root causes are often buried like needles in a haystack.

To address the above challenge, we have adopted two strategies. One is to integrate multiple data sources (including RAS logs, performance data, sensor reading, etc.) and the other is to combine various predictive methods (including statistical learning, data mining, pattern recognition, etc.). By combining and cross-correlating data from different sources, we aim at capturing a variety of fault patterns and interactions in the system, even when these patterns are dynamically changing with time. By combining and coordinating multiple predictive methods, we intend to boost prediction coverage and accuracy. Our current work has focused on answering two questions. The first is *the "when" question*, meaning to predict when failures are likely to occur. The second is *the "where" question*, meaning to identify which part of the system that failures are likely to occur.

To answer the "when" question, we have designed *an ensemble learning based prediction mechanism* (Figure 2). Given that raw data collected by different monitoring tools are often in different formats and contain repeated or redundant entries, during the *event preprocessing* phase, raw data is cleaned and categorized. This step is intended to get a unique and standardized set of events. During the *base prediction* phase, a number of predictive methods are applied on the preprocessed data to identify fault patterns and characteristics. For example, statistical based methods are applied to capture probabilistic characteristics among failure events, and association rules are used to discover causal correlation between different events. During the *meta-learning prediction* phase, ensemble learning techniques are explored to boost prediction accuracy by combining the strengths of different predictive methods [19].

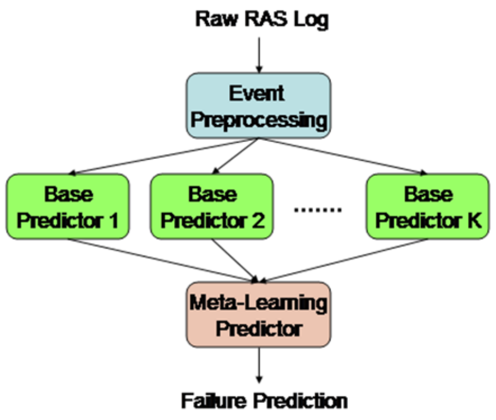

Figure 2. Ensemble Learning Based Prediction

Due to the lack of information provided in the RAS log, our ensemble learning based prediction can only detect when the system behaves abnormally, without identifying the root cause (i.e. the components that cause the problem). To answer the "where" question, we have developed *a PCA (principal component analysis) based localization method* (Figure 3). PCA is a well-known pattern recognition technique, which has been successfully used in many fields for reducing data dimensionality [18]. Our localization method also consists of three steps: (1) *feature collection* to assemble a feature space for the system (generally has a high dimensionality), e.g. the matrix X in the figure; (2) *feature extraction* to obtain the most significant features out of the original feature space for efficient data analysis by applying the principal component analysis (PCA) algorithm, e.g. the matrix Y in the figure; and (3) *outlier detection* to quickly identify the nodes that are "far away" from the majority by using the cell-based detection algorithm [20].

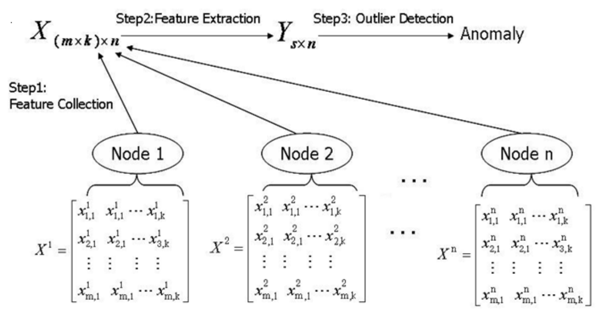

Figure 3. PCA based Anomaly Localization

We have assessed our methods by means of trace studies and fault injection. Our case study with 15-month RAS logs from SDSC and ANL has shown that the ensemble learning based prediction mechanism is able to capture more than 65% of failures, with the false alarm rate less than 35%. In other words, both *Precision* and *Recall* are higher than 65%. Here, *Precision* is defined as T_p / (T_p + F_p), and *Recall* is defined as T_p / (T_p + F_n). 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 incorrect non-failure predictions (i.e. *false negatives*). A good prediction engine should achieve a high value (closer to 1.0) for both metrics. Under a variety of fault modes, the proposed localization method can discover every fault injected, except in one case where *Recall* is lower than 1.0. Besides, *Precision* is always above 0.8, meaning that less than 20% of identified root causes are false alarms.

3. Adaptation Manager

Fault forecasting is only part of the story, and the next major challenge facing the design of adaptive fault tolerance is *how to select an appropriate action at runtime*. There are several requirements in the design of such an adaptation manager. 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.

Toward this end, we have designed an adaptation manager called *FT-Pro* as shown in Figure 4. It dynamically selects a best-fit action from three options. (1) *SKI*P, where the fault tolerant request is ignored. This action is taken to remove unnecessary actions, when failure impact is trivial. (2) *CHECKPOINT*, where the application takes a checkpoint. This action is to reduce application work loss that might be caused by unforeseeable failures. (3) *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 upcoming failure.

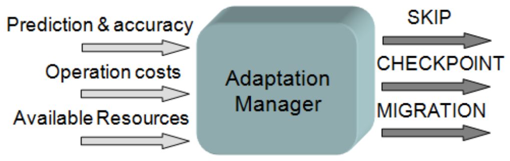

Figure 4. Adaptation Manager

The adaptation manager performs runtime selection based on quantitative performance modeling of applications. More specifically, it estimates the expected application execution time E_{next} during the next interval and selects the action that minimizes E_{next}:

(1) MIGRATION:

$$E_{next} = (2I + C_r + C_{pm}) \cdot f_{appl} + (I + C_{pm}) \cdot (1 - f_{appl})$$

$$where\ \ f_{appl} = \begin{cases} 1-(1-precision)^{N_W^f - N_S^h} & N_W^f > N_S^h \\ 0 & N_W^f \le N_S^h \end{cases}$$

(2) CHECKPOINT:

$$E_{next} = (2I + C_r + C_{ckp}) \cdot f_{appl} + (I + C_{ckp}) \cdot (1 - f_{appl})$$

$$where\ \ f_{appl} = 1-(1-precision)^{N_W^f}$$

(3) SKIP:

$$E_{next} = \left[C_r + (2 + l_{current} - l_{last}) \cdot I\right] * f_{appl} + I * (1 - f_{appl})$$

$$where\ \ f_{appl} = 1-(1-precision)$$

Where I is the adaptation interval, C_{ckp} is the checkpointing overhead, C_{pm} is the migration overhead, C_r is the mean recovery cost, f_{appl} denotes the failure probability of the application during the next interval, $l_{current}$ and l_{last} are the index of the current or the last checkpoint location, N_W^f and N_S^h represent the number of failed computation nodes and healthy spare nodes allocated to the application.

The manager also takes account of prediction *recall*. Given the possibility of unpredictable failures, the performance loss could be significant when a number of SKIP actions have been taken continuously before an unpredicted failure. Hence, when the number of consecutive SKIP actions reaches a threshold (i.e. $MTBF/(I \cdot (1-recall))$), rather than blindly relying on the prediction, the manager enforces a checkpoint.

4. Integrated Runtime Support

We have implemented FT-Pro in the open-source checkpointing package MPICH-VCL 0.76 [21]. Note that FT-Pro is independent of the underlying checkpointing tool, and can be easily implemented with other checkpointing tools. Figure 5 illustrates our implementation: (1) FT-Pro *daemons* that are collocated with application processes on computation nodes, (2) *the dispatcher* that is responsible for managing computing resources, (3) *the adaptation manager* which is in charge of runtime decision making, and (4) *the CKP server* to perform coordinated checkpoints. Process migration may be done through live migration [8,22] or based on the simple stop-and-restart approach.

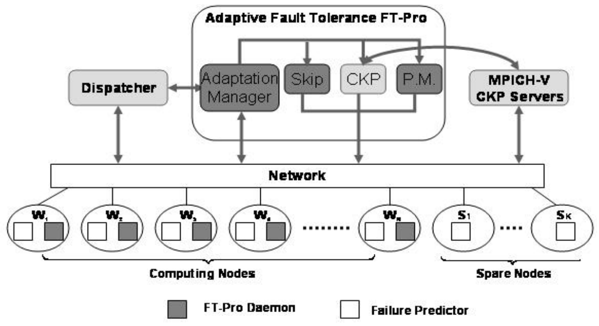

Figure 5. Implementation with MPICH-V

We have evaluated the prototype system with a number of real applications (cosmology application ENZO, molecule dynamics application GROMACS, and the parallel benchmark NPB) under a wide range of settings [24-26]. Preliminary study has indicated that it outperforms periodic checkpointing in terms of reducing application completion times and improving resource utilization, by up to 43% (Figure 6). It does better than periodic checkpointing, even when failure prediction has a 70% false positive and false negative rate (Figure 7). Additionally, the overhead caused by runtime adaptation is less than 3% [12].

5. Summary

This paper presents our on-going project on the design and development of adaptive fault tolerance for HPC applications. We have conducted initial implementation and experimental evaluation of the prototype system under a wide range of computing settings. Our study shows very promising results.

Our on-going efforts include the investigation of advanced failure diagnosis and prognosis techniques, the development of a better integration and coordination support, and extensive evaluation with real applications on production systems.

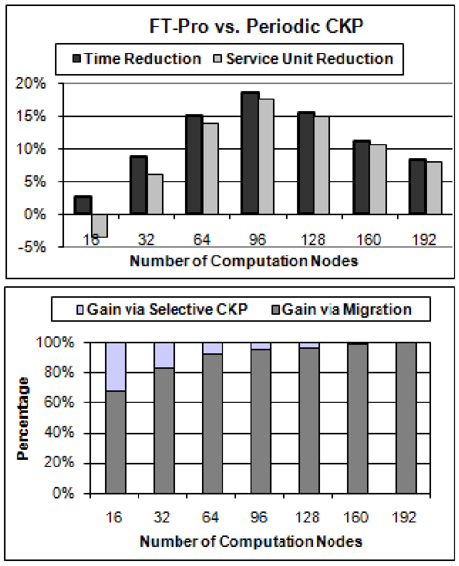

Figure 6. Impact of Computation Scales, where the number of spare nodes is set to 1.

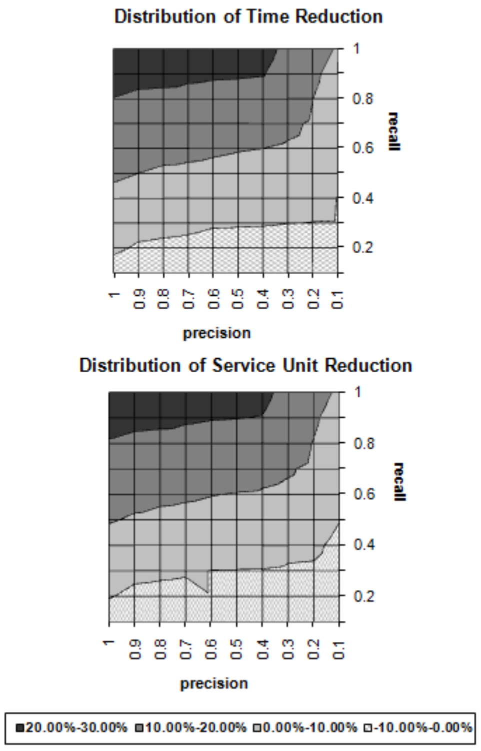

Figure 7. Impact of prediction accuracies.

References

[1] The top500 supercomputer site. [Online]. Available: http://www.top500.org

[2] D. Reed, C. Lu, and C. Mendes, “Big systems and big reliability challenges,” in *Proc. of Parallel Computing*, Germany, 2003.

[3] B. Schroeder and G. Gibson, “A large scale study of failures in high performance-computing systems,” in *Proc. of DSN ’06*, 2006.

[4] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, “A survey of rollback recovery protocols in message-passing systems,” *ACM Computing Surveys*, vol. 34(3), 2002.

[5] E. Elnozahy and J. Plank, “Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1(2), 2004.

[6] V. Castelli, R. Harper, P. Heldelberger, S. Hunter, K. Trivedi, K. Vaidyanathan, and W. Zeggert, “Proactive management of software aging,” *IBM Journal of Research and Development*, vol. 45(2), 2001.

[7] S. Chakravorty, C. Mendes, and L. Kale, “Proactive fault tolerance in large systems,” in *Proc. of HPCRI Workshop*, 2005.

[8] A. Nagarajan and F. Mueller and C. Engelmann and S. Scott, “Proactive Fault Tolerance for HPC with Xen Virtualization”, in *Proc. of the International Conference on Supercomputing*, 2007.

[9] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, “A meta-learning failure predictor for blue gene/l systems,” in *Proc. of International Conference on Parallel Processing*, 2007.

[10] Z. Lan, Y. Li, P. Gujrati, Z. Zheng, R. Thakur, and J. White, “A fault diagnosis and prognosis service for TeraGrid clusters”, *Proc. of The 2nd TeraGrid Conference*, WI, 2007.

[11] Y. Li and Z. Lan, “Exploit failure prediction for adaptive fault tolerance in cluster computing”, *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid06),* Singapore, 2006.

[12] Z. Lan and Y. Li, “Adaptive fault management of parallel applications for high performance computing”, to appear in *IEEE Trans. on Computers*, 2008.

[13] Y. Li and Z. Lan, “Using adaptive fault tolerance to improve application robustness on the Teragrid”, *Proc. of The 2nd TeraGrid Conference*, WI, 2007.

[14] Y. Li and Z. Lan, “A fast recovery mechanism for checkpointing in networked environments”, *SCS Tech Report*, Illinois Institute of Technology, 2007.

[15] R. Sahoo, A. Oliner, I. Rish, M. Gupta, J. Moreira, and S. Ma, “Critical event prediction for proactive management in large-scale computer clusters,” in *Proc. of SIGKDD’03*, 2003.

[16] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, “Blue Gene/L failure analysis and prediction models,” in *Proc. of DSN’06*, 2006.

[17] A. Oliner and J. Stearley, “What supercomputers say: A study of five system logs,” in *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, 2007.

[18] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley Interscience, New York, NY, 2001. 2nd edition.

[19] R. Polikar, “Ensemble based systems in decision making”, *IEEE Circuits and Systems Magazine,* vol. 6(3), 2006.

[20] Edwin M. Knorr, Raymond T. Ng, Vladimir Tucakov, “Distance-based outliers: algorithms and applications”, *The VLDB Journal*,(2000) 8: 237-253.

[21] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello, “Mpich-v: A multiprotocol automatic fault tolerant mpi,” *International Journal of High Performance Computing and Applications*, 2005.

[22] C. Du and X. Sun, “Mpi-mitten: Enabling migration technology in mpi,” in *Proc. of CCGrid’06*, 2006.

[23] Z. Zheng, Y. Li and Z. Lan, “Anomaly Localization in Large-scale Clusters”, *Proc. of IEEE Cluster’07*, 2007.

[24] G. Bryan, T. Abel, and M. Norman, “Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: Resolving primordial star formulation,” in *Proc. of SC’01*, 2001.

[25] H. Berendsen, D. V. der Spoel, and R. van Drunen, “Gromacs: A message-passing parallel molecular dynamics implementation,” *Comp. Phys. Comm.*, vol. 91:43-56, 1995.

[26] Nasa nas parallel benchmarks. [Online]. Available: http://www.nas.nasa.gov/Resources/Software/npb.html