Proactive Fault Tolerance Using Preemptive Migration*

C. Engelmann, G. R. Vallée, T. Naughton, and S. L. Scott
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{engelmannc|valleegr|naughtont|scottsl}@ornl.gov
http://www.fastos.org/ras

Abstract

Proactive fault tolerance (FT) in high-performance computing is a concept that prevents compute node failures from impacting running parallel applications by preemptively migrating application parts away from nodes that are about to fail. This paper provides a foundation for proactive FT by defining its architecture and classifying implementation options. This paper further relates prior work to the presented architecture and classification, and discusses the challenges ahead for needed supporting technologies.

1. Introduction

In order to address anticipated high failure rates, resiliency characteristics have become an urgent priority for next-generation high-performance computing (HPC) systems [8]. Recent trends in HPC system architecture have clearly indicated future increases in performance will be achieved through corresponding increases in system scale. As HPC systems scale beyond 100,000 processor cores, they need to be able to deal with frequent failures in such a manner that their capability is not severely degraded.

The notion of *proactive fault tolerance (FT)* emerged in recent years. It is a concept that prevents compute node failures from impacting running parallel applications by *preemptively migrating* parts of an application (task, process, or virtual machine) away from nodes that are about to fail. Pre-fault indicators, such as a significant increase in heat, can be used to avoid an imminent failure through anticipation and reconfiguration. As computation is migrated away, application failures are avoided and application mean-time to failure (AMTTF) is extended beyond system mean-time to failure (SMTTF). Since avoiding a failure through preemptive migration is significantly more efficient than recovery from failure via traditional *reactive FT* mechanisms, such as *checkpoint/restart*, HPC system utilization becomes more efficient.

Recent efforts in proactive FT primarily targeted two aspects, *failure prediction* [4, 9] and *process-* or *virtual-machine-level migration* [13, 6]. Other initial work focused on a *proactive FT framework* [12], which combines both to perform prediction triggered migration. However, evaluation and comparison of individual solutions is very difficult at this early research stage due to missing realistic architectural models for the deployment of proactive FT technology in extreme-scale HPC systems. Looking at published concepts and developed prototypes, it has become clear that there is a need for a unified definition of a proactive FT architecture, a classification of implementation options, and a discussion of the technology challenges ahead.

This paper offers a foundation for proactive FT in HPC by defining its architecture and classifying implementation options. It further relates prior work to the presented architecture and classification, and discusses the challenges ahead. This paper concludes with a short summary and a brief description of future work.

2. Architecture

Proactive FT keeps an application alive by avoiding experiencing failures through preventative measures. In contrast, *reactive FT* keeps an application running through recovery from experienced failures. More precisely, proactive techniques anticipate, while reactive mechanisms respond. Although reactive solutions perform preventative measures, they do not avoid failures, instead, applications experience failures and re-

*This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

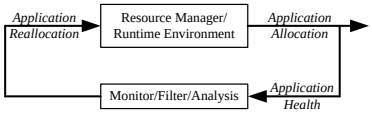

Figure 1. Feedback-loop control of proactive FT using preemptive migration

cover. In contrast, proactive techniques perform *anticipatory reconfiguration* and possibly *graceful degradation* before a failure occurs, such that it can be tolerated without requiring recovery.

Proactive FT using preemptive migration relies on a feedback-loop control mechanism (Figure 1), where the health of each application running on a HPC system is monitored and preventative action is taken to avoid imminent application failure by reallocating running application parts from unhealthy to healthy compute nodes. The feedback loop is formed by continuous application health monitoring, reallocation of application parts in the presence of threads, and reflection of application allocation in application health.

Application health monitoring encompasses hardware and software monitoring, such as observing fan speeds, processor temperature, and processor utilization. Software health monitoring may even include watching application progress, such as I/O patterns, similar to performance monitoring. Monitoring data may be processed by filters and an online reliability analysis to identify trends, correlations, patterns, imminent failure indications, and possible future threads. In addition to the aspect of reliability, the feedback-loop control mechanism may also consider performance parameters, such that application and system health is evaluated in terms of *performability*.

Reallocation evicts application parts from one or more nodes and *excludes them from further use*. As the pool of nodes is reduced, migration is either performed to currently unused nodes, reserved spares, or already allocated ones (oversubscription). Excluded nodes are added to the pool after manual inspection by the system administrator, *e.g.*, in case of a fan fault, and/or an automated testing procedure, *e.g.*, in case of unusually low processor utilization. Reallocating parts of an application typically involves the system resource manager and the compute node runtime environment.

Proactive FT using preemptive migration via a feedback-loop control mechanism is a real-time problem. Application reallocation must be finished before the expected failure occurs, otherwise the expected failure is experienced by applications. The quality of service provided by a feedback-loop control mechanism is measured by the types of failures it covers and the accuracy (timeliness) migration is executed.

Since not all failures can be anticipated, *e.g.*, double bit-flips in ECC memory are random soft errors, proactive FT using preemptive migration is *not* capable of covering all types of failures. A combination of proactive and reactive FT technology provides efficient coverage for predictable and unpredictable failures.

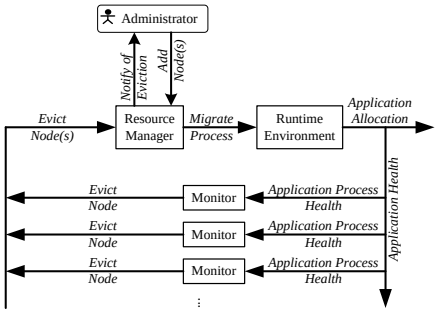

Figure 2. Feedback-loop control of Type 1

3. Classification

Depending on the monitoring capabilities on the compute nodes and the processing of monitoring data within the loop feedback, different control loop properties are displayed. Based on the design of the loop feedback, four distinct types of proactive FT using preemptive migration can be derived.

3.1. Type 1

Type 1 (Figure 2) is the most basic form. A Monitor residing on each compute node is constantly observing system and application health parameters, such as fan speeds, processor temperature, and processor utilization. Upon detecting alerts or exceeding preset limits, such as a fan fault, dangerous processor temperature, or low processor utilization, the Monitor notifies the Resource Manager to evict all application parts from its compute node. The Resource Manager reallocates the application and notifies the Runtime Environment to migrate application parts to other nodes.

This type of alert-driven proactive FT provides coverage for the most basic failures, such as a fan fault that would later lead to overheating or an unusual high processor temperature that would later result in an automated shutdown. Since this type is unable to evaluate the history and context in which an alert is triggered or a limit is exceeded, it is very prone to false positives, *i.e.*, triggering a migration when not needed, and false negatives, *i.e.*, not triggering a migration when needed. Furthermore, due to the reliance on threshold-based triggers for sensors, such as temperature, this variant is also prone to miss the time window in which a migration needs to be performed to successfully avoid failure.

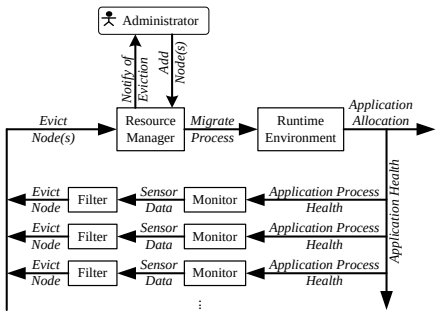

Figure 3. Feedback-loop control of Type 2

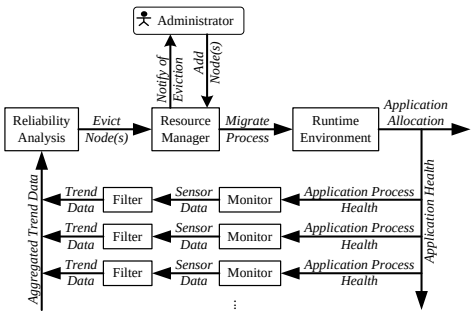

Figure 4. Feedback-loop control of Type 3

Additionally, due to a missing application- and system-wide health status view, migrations may interfere with each other if triggered in rapid succession.

Any type of proactive FT has three core implementation requirements: (1) the Resource Manager must support live removal of nodes, (2) the Resource Manager must be able to notify the Runtime Environment, and (3) the Runtime Environment must support migration. If these technologies exist, implementing Type 1 is simple as only node-local monitoring and Resource Manager notification are additionally required.

The quality of service provided by any type of proactive FT depends on failure type coverage, prediction accuracy, and reaction time. Apart from the coveage of only the most basic failure types, the quality of service of Type 1 is dominated by the already discussed threshold accuracy of the Monitor and the time it takes to notify the Resource Manager, to notify the Runtime Environment, and to migrate. While in Type 1 inaccurate thresholds can quickly result in false positives and false negatives, *all* proactive FT types suffer from the issue that the reaction time required may be longer than a triggered alert or predicted failure precedes an application failure. For example, a fan fault may very quickly result in overheating, giving the feedback-loop control not enough time to react.

3.2. Type 2

Type 2 (Figure 3) is an enhanced form of basic proactive FT. Similar to Type 1, a Monitor residing on each node is constantly observing health parameters. However, instead of just notifying the Resource Manager upon detecting alerts or exceeding preset limits, Type 2 utilizes a Filter on each node to process raw sensor data and alerts from the Monitor to notify the Resource Manager based on a more thorough analysis of current trends, imminent failures, and possible future threads. As in Type 1, the Resource Manager reallocates the application and notifies the Runtime Environment to migrate application parts.

This type extends the alert-driven Type 1 by offering a better coverage for basic failures. Using the Filter, sensor data is processed with short-term historical context, such as the steepness of a temperature increase. This results in a much clearer identification of imminent failures and possible future threads through trend analysis, such that much less false positives and false negatives occur and the time window for migration is not missed. However, there is no correlation of sensor data from multiple nodes to identify related true positives, such as an overall increase in processor heat in a particular section caused by a system or room cooling issue. As in Type 1, migrations may interfere with each other if triggered in rapid succession, due to a missing global health status view.

The implementation requirements for Type 2 are the same as for Type 1 with the exception of the additional node-local Filter, which can be as simple as a trend analyzer combined with a steepness trigger or as complex as a machine learning system.

In comparison to Type 1, the quality of service provided by Type 2 is better as the Filter performs node-local failure prediction. Its accuracy is key to the quality of service of the feedback-loop control. To avoid the problem of triggering a migration not enough ahead of time, the reaction time of the feedback-loop control may be incorporated in the migration trigger decision of the Filter. In this case, a migration is triggered when a failure is predicted with a certain accuracy/probability for the future safe reaction time interval.

3.3. Type 3

Type 3 (Figure 4) is an advanced form of proactive FT. As in Types 1 and 2, a Monitor residing on each node is observing system and application health. As in Type 2, a Filter on each node processes raw sensor data and alerts from the Monitor to extract trends, imminent failure indications, and possible future threads. In contrast to Type 1 and 2, the aggregated data from all Filters is processed by a Reliability Analysis, which performs an application- *and* system-wide identification of correlations, such as an overall increase in processor heat or

a number of nodes idling, and applies policies for preemptive migration actions, such as to trigger migration based on failure propability. The Reliability Analysis notifies the Resource Manager to evict application parts from nodes. The Resource Manager reallocates the application and notifies the Runtime Environment to migrate application parts.

Since the Reliability Analysis is able to obtain a global health status view, this type enables the correlation of sensor data from multiple nodes to identify related true positives and to avoid migration targets that indicate possible future threads. In contrast to Type 1 and 2, system and application health information is not only analyzed, proactive FT policy is also implemented by either excluding nodes as targets or by directly selecting targets via a reliability-aware allocation policy. However, since only short-term historical context is used, failure and reliability patterns associated to applications, like idling processes during a specific application phase, are not detected.

Implementing Type 3 requires, in addition to Type 2, a Reliability Analysis, which models the reliability of the system and of each running application. This can be as simple as a trend analyzer combined with a steepness and deviation trigger or as complex as a machine learning system. Furthermore, coordination of sensor data processing by the node-local Filter and of Filter output processing by the Reliability Analysis is needed to avoid disruption in the feedback-loop control by providing not enough, too much, or the wrong information to the Reliability Analysis. Additionally, a data aggregation mechanism is required as supporting technology to collect Filter output from nodes.

Since Type 3 covers a wider range of true positives and true negatives due to the combined accuracy of the node-local Filter and the global Reliability Analysis, its quality of service is significantly better than Type 1 or 2. Similar to Type 2, the reaction time of the feedback-loop control may be incorporated in the migration trigger decision of the Reliability Analysis.

3.4. Type 4

Type 4 (Figure 5) is an enhanced form of advanced proactive FT. In addition to Type 3, the Reliability Analysis utilizes a History Database for recording prior system and application reliability patterns and for matching them against the currently experienced pattern in order to optimize the feedback control loop using machine learning techniques. The Reliability Analysis may also utilize the History Database to conduct offline investigations of system and application reliability patterns, when longer-term records and more extensive computation is required. The implementation of Type 4 requires a History Database in addition to Type 3, while the Reliability Analysis is a machine learning system. The quality of service provided by this type is even higher as failure prediction accuracy improves with long-term memory about failure patterns.

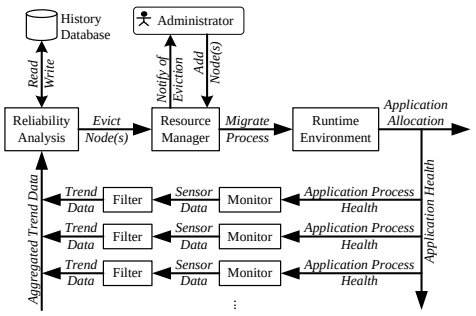

Figure 5. Feedback-loop control of Type 4

4. Previous Work

Prior work targeted *system monitoring*, *system log* and *reliability analyses*, *transparent migration mechanisms*, *proactive/reactive FT trade-off models*, and preliminary *proactive FT frameworks*.

4.1. System Monitoring

A recent standardization effort across server vendors resulted in the Intelligent Platform Management Interface (IPMI) for system management and health monitoring. OpenIPMI (http://openipmi.sourceforge.net) is a software effort to allow full access to IPMI information. OpenIPMI has already been used in Type 1 feedback-loop control solutions [6, 13]. It can be used with any type of feedback-loop control.

Ganglia [5] is a scalable distributed monitoring system, where each node monitors itself and disseminates data to all other nodes. It has already been used in Type 1 feedback-loop control solutions [6, 13].

OVIS 2 [1] is a hierarchical monitoring and analysis tool that collects system health information directly from nodes or from other monitoring solutions, such as Ganglia, for processing using statistical methods for graphical presentation. OVIS 2 provides Type 3 and 4 online Reliability Analysis as well as Type 4 offline Reliability Analysis using a database. It has not yet been used in proactive FT feedback control loops.

MRNet [7] is a software overlay network that provides efficient multicast and reduction communications. It can efficiently compute averages, sums, and more complex aggregations and analyses. MRNet has not yet been used for proactive FT, but its scalable design offers

data reduction and aggregation capabilities that may be used to offload Type 3 and 4 Reliability Analysis to a fan-in tree of compute nodes.

A few other solutions exist, such as the reliability, availability and serviceability (RAS) systems deployed by HPC vendors. They are similar in design and none of them have been used for proactive FT.

4.2. System Log and Reliability Analysis

Analyzing failure patterns using statistical methods has been a recent effort in HPC. Most work relies on log data from the USENIX Computer Failure Data Repository (CFDR) at http://cfdr.usenix.org.

Recent accomplishments include a failure prediction framework [4], which explores correlations and forecasts the time-between-failure of future instances. Results show the system achieves more than 76% accuracy in offline prediction and more than 70% accuracy in online prediction using failure logs from Los Alamos National Laboratory. While the online prediction may be used for Type 3 feedback-loop control, the offline prediction requires the recording capability of Type 4.

Another effort [9] focused on syslogs that are found on nearly all computing systems. The developed classification scheme for syslog messages was able to localize 50% of faults with 75% precision, corresponding to an excellent false positive rate of 0.05%, using three weeks of syslogs from the 512-node "Spirit" Linux cluster at Sandia National Laboratory. This solution entirely focused on a centralized offline analysis (Type 4), *i.e.*, fault localization using syslog data.

Many other efforts in system log and reliability analysis as well as in failure prediction exist. They either focus on node-local or system-wide identification of root causes, failure modes, trends, correlations, patterns, imminent failure indications, and possible future threads. Most solutions can be used for proactive FT.

4.3. Transparent Migration Mechanisms

Recent work focused on proactive FT supported by the virtual machine migration mechanism of the Xen hypervisor [6] and by a newly developed migration mechanism for the BLCR checkpoint/restart solution [13]. Both prototypes utilize a hardware monitoring mechanism to trigger migration (Type 1). While the developed prototypes operate in a Type 1 setting, their transparent migration mechanisms can be used in any type of feedback-loop control.

Another past effort targeted transparent MPI task migration using the Charm++ middleware and its Adaptive MPI (AMPI) [2]. This work primarily focused on the migration aspect and did not provide the feedback-loop control needed for proactive FT. This solution can be used in any type of feedback-loop control.

MPI-Mitten [3] is a similar effort that provides a library between MPI and applications for transparent migration support. Ongoing work in a Fault awareness Enabled Computing Environment (FENCE) [10] uses MPI-Mitten for migration-based reliability-aware scheduling and proactive FT in a Type 4 configuration with SMTTF and AMTTF approximation within the Reliability Analysis.

Many other transparent migration mechanisms have been researched and developed, however, none of them have been used until now for preemptive migration of parallel applications.

4.4. Proactive + Reactive Fault Tolerance

Recent work in this area utilized simulations to evaluate different trade-off models when combining preemptive migration with checkpoint/restart [11]. Using failure logs from Lawrence Livermore National Laboratory, the impact of failure prediction accuracy was evaluated and put in context with restart counts and checkpointing frequency. The results suggest that this *holistic FT* approach provides the best fault resilience with the highest system utilization.

4.5. Proactive Fault Tolerance Frameworks

As previously mentioned, two prototypes for Type 1 feedback-loop control have been developed [6, 13] in the past. Application reallocation was performed using a load balancer. Another Type 1 prototype [12] investigated coordination, protocols, and interfaces between individual system components.

5. Challenges Ahead

Specific needs for future work in proactive FT using preemptive migration can be identified based on the individual components of the different feedback-loop control types and on required supporting technology.

Future work in health monitoring needs to focus on identifying deteriorating applications and operating system conditions. Failure detection and pre-fault indication coverage needs to include application failures, such as programming errors or resource exhaustion. The HPC performance tools community has already made significant advances in this area. Performability monitoring can provide extended coverage.

Similarly, reliability analysis needs to factor in performance parameters as the goal is to improve time to

solution. More extensive work in performability analysis for HPC is needed.

Scalable data aggregation and processing is key for timeliness in the feedback control loop. Work in MRNet-style in-flight monitoring data aggregation, filtering and analysis is needed to provide for scalable real-time characteristics.

Lastly, there is a need for standardized metrics and interfaces. Currently, each solution uses its own metrics for measuring and evaluating health and interfaces between components. This makes comparison and integration unnecessary difficult.

6. Conclusion

We provided a foundation for proactive FT in HPC by defining its architecture and classifying implementation options. The presented architecture relies on a feedback-loop control mechanism, where system and application health is monitored and preventative action is taken to avoid imminent application failure by reallocating running application parts from unhealthy to healthy compute nodes. The feedback loop is formed by continuous application health monitoring, reallocation of application parts in the presence of threads, and reflection of application allocation in application health. We identified four distinct types of proactive FT using preemptive migration based on the monitoring capabilities on the compute nodes and the processing of monitoring data within the loop feedback.

We further related prior work to the presented architecture and classification. Specifically, we described a selection of system monitoring solutions, system log and reliability analyses, transparent migration mechanisms, proactive/reactive FT trade-off models, and preliminary proactive FT frameworks. We also briefly discussed the challenges ahead for proactive FT in HPC.

Our planned research and development efforts primarily focus on the discussed challenges, specifically on standardized metrics/interfaces and on scalable data aggregation and processing. Additionally, our ongoing work also looks at failure injection mechanisms.

References

[1] J. M. Brandt, B. J. Debusschere, A. C. Gentile, J. R. Mayo, P. P. Pébay, D. Thompson, and M. H. Wong. OVIS-2: A robust distributed architecture for scalable RAS. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS): Workshop on System Management Techniques, Processes, and Services (SMTPS)*, 2008.

[2] S. Chakravorty, C. L. Mendes, and L. V. Kalé. Proactive fault tolerance in MPI applications via task migration. In *Lecture Notes in Computer Science: Proceedings of the International Conference on High Performance Computing (HiPC)*, volume 4297, pages 485–496, 2006.

[3] C. Du and X.-H. Sun. MPI-Mitten: Enabling migration technology in MPI. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 11–18, 2006.

[4] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2007.

[5] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[6] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *Proceedings of the ACM International Conference on Supercomputing (ICS)*, pages 23–32, 2007.

[7] P. C. Roth, D. C. Arnold, and B. P. Miller. MRNet: A software-based multicast/reduction network for scalable tools. In *Proceedings of the ACM/IEEE International Conference on High Performance Computing and Networking (SC)*, 2003.

[8] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. In *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference*, volume 78, pages 2022–2032, 2007.

[9] J. Stearley and A. J. Oliner. Bad words: Finding faults in Spirit‘s syslogs. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid): Workshop on Resiliency in High Performance Computing (Resilience)*, 2008.

[10] X.-H. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng. Towards a fault-aware computing environment. In *Proceedings of the High Availability and Performance Workshop (HAPCW), in conjunction with the High-Performance Computer Science Week (HPCSW)*, 2008.

[11] A. Tikotekar, G. Vallée, T. Naughton, S. L. Scott, and C. Leangsuksun. Evaluation of fault-tolerant policies using simulation. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster)*, 2007.

[12] G. R. Vallée, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. B. Leangsuksun, T. Naughton, and S. L. Scott. A framework for proactive fault tolerance. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pages 659–664, 2007.

[13] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive process-level live migration in HPC environments. In *Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2008. To appear.