Introspective Fault Tolerance for Exascale Systems*

Rinku Gupta, Kamil Iskra, Kazutomo Yoshii, Pavan Balaji, Pete Beckman

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439, USA
{rgupta, iskra, kazutomo, balaji, beckman}@mcs.anl.gov

1 Motivation

Faults and errors are an unavoidable aspect of high performance computing systems. Emerging exascale systems will contain billions of hardware components and complex software stacks. In addition, higher fabrication density and power challenges will further compound fault detection, management and recovery. Efficient fault tolerance and resiliency frameworks are thus of immense importance in the path to the exascale era [7].

Faults and errors are broadly categorized as *hard* errors (where a hardware component has failed and requires human intervention to be corrected), *soft/transient* errors (where a fault occurred, but has been corrected by the hardware or low-level system software), and *silent/undetectable* errors (where an error occurred but the hardware and low-level system software could not detect the error) [4]. While there has been a tremendous amount of work for tolerating each of these types of faults, these mechanisms fundamentally lack the ability to share information. For example, the behavior of most operating systems when a hard fault occurs is to kill the application or the entire node. Similarly, the behavior when a soft error occurs is to automatically correct the error, but not inform the user about it. Furthermore, there is currently no way for the user to hint to the operating system about the application's ability to work around hard or soft errors.

2 Introspective Fault Tolerance

While resiliency is achievable in various forms throughout the software stack, a comprehensive fault infrastructure would need mechanisms for lower-level hardware and operating system to interface with upper layers libraries and applications for fault detection, fault information exchange, error handling negotiations as well as recovery in a standardized way. Currently approaches, where most faults are detected at system level through return error codes and OS signals (containing limited information), are insufficient and of little use when available to upper layers libraries and applications [7]. Upper-level software typically cannot make educated decisions for fault recovery based on this information and typically resort to checkpoint/restart/migration or fall-back to program abortion.

This problem of information exchange exists in the reverse direction as well. Considering the power and fault challenges exposed by exascale systems, it is understandable that high tradeoffs will exist between power, resiliency and performance on a system. Increased resiliency may come at the cost of power and/or performance. Similarly, low power may come at the cost of deceased resiliency.

The goal of introspective fault tolerance is to bridge this gap, by providing a two-way communication mechanism between the application and the operating system to exchange information on fault occurrences as well as tuning power/performance/resilience tradeoffs based on application characteristics. To achieve a goal for comprehensive resiliency, it is mandatory to focus on operating system-level methodologies, frameworks and interfaces with the following open questions: 1) What faults/system-changes highly impact upper-level software/applications and how to improve this fault detection at OS- or system-level. 2) how do we bridge the gap for information exchange between the operating system and upper level software, 3) what techniques would allow upper-level software to use this information exchange for either adapting and recovering from faults pro-actively or negotiating tradeoffs between performance, resiliency and power.

Specifically, the operating system would need to provide hooks for users to access and control power, resiliency and performance-impacting hardware. One example of such a behavior can be with respect to memory allocations. For instance, if an application can tolerate memory bit flips for certain parts of its memory, it can ask the OS to turn off ECC checks for those memory regions (e.g., one memory DIMM) and potentially improve power and performance. The form of interface that we envision is a model where memory allocation is conservatively most resilient, but the application can relax this requirement through annotations as shown in Figure 1.

Another example of such introspection demonstrated in

*This work was supported by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

```
1   void *buf1 = malloc(1000); /* memory allocation conservative by default; if a hard error occurs, abort! */
2
3   foo(buf1); /* function foo works on reliable memory */
4   os_check_ecc_errors(&errors);  /* introspect soft ecc errors */
5    /* if there are too many correctable ECC errors.  perhaps better
6    to use a different memory region or a different node */
7
8   void *buf2 = malloc_with_error_returns(1000); /* if a hard error occurs, return an error */
9
10  bar(buf2); /* function bar works on reliable memory, but can tolerate errors */
11  os_check_hard_memory_errors(&errors);  /* introspect hard memory errors (e.g., double bit flips) */
12  iif (errors) {
13     /* recompute data computed in bar */
14  }
15
16  void *buf3 = unreliable_malloc(1000); /* allow OS to turn off reliability checks */
17
18  foobar(buf3); /* function foobar handles memory errors */
```

Figure 1: Memory Management Example

Figure 1 is the ability of the application to query the OS for soft errors and decide whether to continue execution or migrate to a different memory region/node.

Such introspection capabilities allow applications and upper-level software to be closely involved in the performance, power, and resilience management of the overall system, thus guiding the OS into making more educated choices. Such capabilities are already important today, and will be critical in the exascale era.

3 Related Work

Research in the field of resiliency in high performance computing has been vast and varied. Following is some related work from the perspective of this paper.

Several studies have been conducted to understand the types and nature of fault on large supercomputers [3, 8, 13, 17, 20]. Attention has been directed to fault communication and exchange as well, which includes systems logs and log analysis, log compression, analysis and prediction using system logs as well as root causing failures by mining logs [5, 13, 15, 23]. Using environmental monitoring information for system-wide event and fault monitoring has also been vastly studied; along with tools and software to handle and predict faults [6, 11, 12, 16, 21, 22].

There exists several APIs for exchanging information between network peers. However, limited research has targeted interfaces and API focussing on faults, fault monitoring, adaptive fault and recovery handling and fault information exchange at the operating system level [1, 14]. Modern systems and operating systems come with certain interfaces, tools and software that allow upper-level software access to portions of the hardware [2,9,12], but these interfaces are not generic for fault and cannot be used by all levels of the software stack. Gupta et al. [10, 18, 19] developed coordination and fault exchange framework for exchanging fault information among various software on the system. This work serves as the most relevant prior work for our paper.

4 Summary

Challenges addressed: A unified comprehensive fault and resiliency introspection framework for exascale operating systems.

Maturity: Applications and upper level software are playing an increasing important role in system-wide fault tolerance and resiliency. There is already a large body of literature that deals with improving system and application fault tolerance when the fault information is available. There are also ongoing DOE funded activities that deal with managing the performance/power/resilience tradeoffs in applications. Thus, the time is ripe for designing an comprehensive introspection framework that allows these activities to blend with the design and future directions of exascale operating systems.

Uniqueness: Exascale systems are expected to have stricter power and resilience constraints than anything we have seen before. Thus, a model where an OS or hardware architecture can independently manage the performance/power/resilience tradeoffs is already strained and would be irrelevant in the exascale era.

Novelty: Exascale systems will allow tradeoffs between power, performance and faults. This tradeoffs are transparent to applications currently. Our approach allows the operating system to help applications work more closely with system hardware for its optimal needs.

Applicability: We expect the framework and interfaces to be used by researchers working with upper level software such as math libraries, programming libraries and applications.

Effort: The effort would require upper level software folks (i.e programming models, applications scientists) to work closely with operating system researchers and possibly vendors. Estimated effort would be around four-to-five people over the span of three-to-five years.

References

[1] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, Dec. 2003.

[2] S. Atchley, D. Dillow, G. Shipman, P. Geoffray, J. M. Squyres, G. Bosilca, and R. Minnich. The common communication interface (cci). In *Proceedings of the 2011 IEEE 19th Annual Symposium on High Performance Interconnects*, HOTI '11, pages 51–60, Washington, DC, USA, 2011. IEEE Computer Society.

[3] F. Cappello. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *Int. J. High Perform. Comput. Appl.*, 23(3):212–226, Aug. 2009.

[4] F. Cappello, A. Geist, B. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience, Technical Report of the INRIA-Illinois Joint Laboratory on PetaScale Computing, 200.

[5] E. Chuah, S.-H. Kuo, P. Hiew, W.-C. Tjhi, G. K. K. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne. Diagnosing the root-causes of failures from cluster log files. In *HiPC'10*, pages 1–10, 2010.

[6] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Trans. Softw. Eng.*, 30(12):859–872, Dec. 2004.

[7] DOE/NSA/ASCR. Report from the Inter-Agency Workshop on HPC Resilience at Extreme Scale, 2012.

[8] E N M Elnozahy and R Bianchini and T ElâĂŘGhazawi and F Godfrey A Fox and K McKinley and A Hoisie and R Melhem and J S Plank and P Ranganathan, and J Simons. System Resilience at Extreme Scale.

[9] S. Eyerman and L. Eeckhout. Fine-grained dvfs using on-chip regulators. *ACM Trans. Archit. Code Optim.*, 8(1):1:1–1:24, Feb. 2011.

[10] R. Gupta, P. Beckman, B.-H. Park, E. Lusk, P. Hargrove, A. Geist, D. Panda, A. Lumsdaine, and J. Dongarra. CIFTS: A Coordinated Infrastructure for Fault-Tolerant Systems. *International Conference on Parallel Processing (ICPP)*, pages 237–245, 2009.

[11] E. Imamagic and D. Dobrenic. Grid infrastructure monitoring system based on nagios. In *Proceedings of the 2007 workshop on Grid monitoring*, GMW '07, pages 23–28, New York, NY, USA, 2007. ACM.

[12] C. Leangsuksun, T. Rao, A. Tikotekar, S. L. Scott, R. Libby, J. S. Vetter, Y.-C. Fang, and H. Ong. Ipmi-based efficient notification framework for large scale cluster computing. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 23–, Washington, DC, USA, 2006. IEEE Computer Society.

[13] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 583–588, Washington, DC, USA, 2007. IEEE Computer Society.

[14] X. Liu, P. Shenoy, and M. Corner. Chameleon: application level power management with performance isolation. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 839–848, New York, NY, USA, 2005. ACM.

[15] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. Fast entropy based alert detection in super computer logs. In *Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, DSNW '10, pages 52–58, Washington, DC, USA, 2010. IEEE Computer Society.

[16] P. J. Mucci, D. Ahlin, J. Danielsson, P. Ekman, and L. Malinowski. Perfminer: cluster-wide collection, storage and presentation of application level hardware performance data. In *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, Euro-Par'05, pages 124–133, Berlin, Heidelberg, 2005. Springer-Verlag.

[17] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, pages 575–584, Washington, DC, USA, 2007. IEEE Computer Society.

[18] B. H. Park, T. J. Naughton, P. Agarwal, D. E. Bernholdt, A. Geist, and J. L. Tippens. Realization of user level fault tolerant policy management through a holistic approach for fault correlation. In *Proceedings of the 2011 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '11, pages 17–24, Washington, DC, USA, 2011. IEEE Computer Society.

[19] R. Rajachandrasekar, X. Besseron, and D. L. Panda. Monitoring and Predicting Hardware Failures inHPC Clusters with FTB-IPMI. In *SMTPS 2012*, 2012.

[20] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Trans. Dependable Secur. Comput.*, 7(4):337–351, Oct. 2010.

[21] M. J. Sottile and R. G. Minnich. Supermon: A high-speed cluster monitoring system. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '02, pages 39–, Washington, DC, USA, 2002. IEEE Computer Society.

[22] J. Sun, E. Hoke, J. D. Strunk, G. R. Ganger, and C. Faloutsos. Intelligent system monitoring on large clusters. In *Proceedings of the 3rd workshop on Data management for sensor networks: in conjunction with VLDB 2006*, DMSN '06, pages 47–52, New York, NY, USA, 2006. ACM.

[23] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 117–132, New York, NY, USA, 2009. ACM.