D6.1 Preliminary Report on State of the Art Software-Based Fault Tolerance Techniques
Version 1.0

Document Information

Contract Number	610402
Project Website	www.montblanc-project.eu
Contractual Deadline	PM12
Dissemination Level	PU
Nature	R
Author	Frank Cappello (INRIA)
Contributors	Ferad Zyulkyarov (BSC), Osman Unsal (BSC)
Reviewer	Gilles Sassatelli, Abdoulaye Gamatie (CNRS)
Keywords	resiliency, fault tolerance, HPC, reliability, exascale computing, survey

Notices:
The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 610402.

Change Log

Version	Description of Change
V1.0	Initial Draft released to the European Commission

Table of Contents

Executive Summary

This deliverable is a preliminary report on state of the art software-based resiliency techniques for high performance computing (HPC). The document overviews the past resiliency challenges and the proposed solutions to address them. It reviews what the future resiliency challenges would be in exascale computing and tries to project research directions to tackle these problems.

Resilience is a major roadblock for HPC executions on future exascale systems. These systems will typically gather millions of CPU cores running up to a billion threads. Projections from current large systems and technology evolution predict errors will happen in exascale systems many times per day. These errors will propagate and generate various kinds of malfunctions, from simple process crashes to result corruptions.

The state of the art research made extraordinary technical progress in many domains related to exascale resilience. Several technical options, initially considered to be inapplicable to or unrealistic in the HPC context, have demonstrated surprising successes. Despite this progress, the exascale resilience problem is not solved, and the community is still facing the difficult challenge of ensuring that exascale applications complete and generate correct results while running on unstable systems. Many workshops, studies, and reports have improved the definition of the resilience problem and provided refined recommendations. Some projections made during the previous decades and some priorities established from these projections need to be revised.

Part of this document was published in: Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, Marc Snir, "Toward Exascale Resiliency: 2014 Update", Supercomputing Frontiers and Innovations, Vol 1, No 1, 2014 - http://superfri.org/superfri/article/view/14

The findings described in this document has been mentioned at insideHPC (http://insidehpc.com/2014/07/new-paper-toward-exascale-resilience-2014-update/) and received large publicity.

1 Introduction

As we get closer to exascale computing, along with power and performance, reliability becomes the third main concern for future high performance computing (HPC). Probability of **faults**, that cause **errors** and consequently result in **failures** is orders of magnitude higher for supercomputers compared to a standalone desktop PC. Faults can be transient or permanent. Transient faults are a temporary change in the logic values of the latches or hardware logic structures. Transient faults may happen because of high-energy particles or external factors such as high operational temperatures. Permanent faults such as stuck-bit faults are permanent malfunction of hardware components. Permanent faults may happen due to ware out, fabrication defect or a physical damage. Without effective and efficient resiliency techniques, applications that run for many hours on thousands of nodes would not be able to complete successfully or produce an incorrect result.

Supercomputers which are assembled by thousands of nodes and millions of hardware components would be effectively inoperable if **fault tolerance** techniques were not successfully employed. In principle, fault tolerance is achieved by detecting errors, notifying about the detected errors and recovering or compensating the detected errors. Various fault tolerance techniques exist both in hardware and software. Software-based fault tolerance techniques are of particular interest for the MontBlanc 2013-2016 project because the MontBlac prototype supercomputer is to be assembled of commodity of the shelf (COTS) components which does not have built-in error-correcting codes (ECC). Without ECC, supercomputers like MontBlanc are expected to experience fault frequently and thus rendering the entire system inoperable or the results it non-trustable.

This document makes a survey of state of the art in software-based fault tolerance techniques for HPC.

2 Software-Based Fault Tolerance

This section discusses the progress in software-based techniques for fault tolerance including handling fail-stop errors by checkpointing, forward recovery, replication, failure prediction, and silent data corruption mitigation.

2.1 Checkpointing

To tolerate fail-stop errors (node, OS or process crash, network disconnections, etc.), all current production-quality technologies rely on the classic rollback recovery approach using checkpoint restart (application-level or system-level). Solutions focus on applications using MPI and Charm++. The most popular approach is application-level checkpointing, where the programmer defines the state that needs to be stored in order to guarantee a correct recovery in case of failure. The programmer adds some specific functions in the application to save essential state and restore from this state in case of failure. One of the drawbacks of checkpointing in general, and of application-level checkpointing in particular, is the nonoptimality of checkpoint intervals. Another drawback is the burden placed on the I/O infrastructure, since the checkpoint I/O may actually interfere with communication and I/O of other applications. The impact of suboptimal checkpoint intervals is investigated in [45].

System-level checkpoint can also be used in production, with technologies such as BLCR [41]. Some recent research in this domain focuses on the integration of incremental checkpointing in BLCR. In the past few years, research teams have developed mechanisms to checkpoint accelerators [55, 57].

The norm in 2009 was to store the application state on remote storage, generally a parallel file system, through I/O nodes. Checkpoint time was significant (often 15-30 minutes), because of the limited bandwidth of the parallel file system. When checkpoint time is close to the MTBF, the system spends all its time checkpointing and restarting, with little forward progress. Since the MTBF may be an hour or less on exascale platforms, new techniques are needed in order to reduce checkpoint time.

One way of achieving such a reduction is to reduce the checkpoint size. Techniques such as memory exclusion, data compression, and compiler analysis to detect dead variables have been proposed. More recently some researchers have explored hybrid checkpointing [63], data aggregation [44], incremental checkpointing in the context of OpenMP [11], and data deduplication with the hope that processes involved in parallel HPC executions have enough similarities in memory [67] to reduce the size of the data sets necessary to be saved. However, what we mentioned five years ago is still valid: Programmers are in the best position to know the critical data of their application but they cannot use adaptive data protection other than by doing it manually. Annotations about ways to protect or check key data, computations, or communications are still a relevant direction.

Another way of reducing checkpoint time is to reduce the usage of disks for checkpoint storage. In-memory checkpointing has been demonstrated to be fast and scalable [18]. In addition, multilevel checkpointing technologies such as FTI [2] and SCR [52] are increasingly popular. Multilevel checkpointing involves combining several storage technologies (multiple levels) to store the checkpoint, each level offering specific overhead and reliability trade-offs. The checkpoint is first stored in the highest performance storage technology, which generally is the local memory or a local SSD. This level supports process failure but not node failure. The second level stores the checkpoint in remote memory of remote SSD. This second level supports a single-node failure. The third level corresponds to the encoding the checkpoints in blocks and in distributing the blocks in multiple nodes. This approach supports multinode failures. Generally, the last level of checkpointing is still the parallel file system. This last level supports catastrophic failures such as full system outage. Charm++ provides similar multilevel checkpointing mechanisms [16]. The checkpoint period can be defined in different ways. Checkpoints also can be moved between levels in various ways, for example, by using a dedicated thread [2] or agents running on additional nodes [61]). A new semi-blocking checkpoint protocol leverages multiple levels of checkpoint to decrease checkpoint time [54]. A recent result computes the optimal combination of checkpoint levels and the optimal checkpoint intervals for all levels given the checkpoint size, the performance of each level, and the failure distributions for each level [56]. Other recent research concerns the understanding of the energy footprint of multilevel checkpointing and the exploration of trade-offs between energy optimization and performance overhead. The progress in multilevel checkpointing is outstanding, and some initiatives are looking at the definition of a standard interface.

By offering fast checkpointing at the first level, multilevel checkpointing also offers the opportunity to increase the checkpoint frequency and checkpoint at a smaller granularity. For example instead of checkpointing at the outermost loop of an iterative method, multilevel checkpointing could be used in some inner loops of the iterative method. Reducing the granularity of checkpointing is also the objective of task-based fault tolerance. The principle is to consider the application code as a graph of tasks and checkpoint the input parameter of each task on local storage with a notion of transaction: either a task is completed successfully and its results are committed in memory, or the task is restarted from its checkpointed input parameters. This approach is particularly relevant for future nonvolatile memory on computing nodes. The nonvolatile memory would store the input parameters checkpoints and the committed results. The OmpSs environment offers a first prototype of this approach [62]. One of the key questions is how to ensure a consistent restart in case of a node failure. Solutions need to consider how to store the execution graph and the local memory content in order to tolerate a diversity of failure scenarios.

The past five years have also seen excellent advances in checkpointing protocols. Classic checkpointing protocols are used to capture the state of distributed executions so that, after a failure, the distributed executions (or a part of them) restart and produce a correct execution [24]. In the HPC domain, classic checkpointing protocols are rarely used in production environments because most of the applications use application-level checkpointing that implicitly coordinates the capture of the distributed state and guarantees the correctness of the execution after restart. Without additional support, however, application-level checkpointing imposes global restart even if a single process fails. Message-logging fault-tolerant protocols offer a solution to avoid global restart and to restart only the failed processes. By logging the messages during the execution and replaying messages during recovery, they allow the local state reconstruction of the failed processes (partial restart). The main limitation is the necessity to log all messages of the execution. Several novel fault tolerance protocols overcome this limitation by reducing significantly the number of messages to log. They fall into the class of hierarchical fault tolerance protocols, forming clusters of processes and using coordinated checkpointing inside clusters and message logging between clusters [39, 51, 58]. Such protocols need to manage causal dependencies between processes in order to ensure correct recovery. Multiple approaches have been proposed for reducing the overhead of storing causal dependency information [8, 12].

Partial restart opens opportunities for accelerated recovery. Charm++ and AMPI accelerate recovery by redistributing recovering processes on nodes that are not restarting [13]. Message logging can by itself accelerate recovery because messages needed by recovering processes are immediately available and messages to be sent to nonrestarting processes are just canceled [58]. These two aspects reduce the communication time during recovery. A recent advance in hierarchical fault tolerance protocols is the formulation and solving of the optimal checkpoint interval in this context [6]. Partial restart also opens the opportunity to schedule other jobs (than the recovering one) on resources running nonrestarting processes during the recovery of the failed processes. While this principle does not improve the execution performance for the job affected by the failure, it significantly improved the throughput of the platform that execute more jobs in a given amount of time compared with restarting all the processes of a job when a failure occurs [9]. This new result demonstrates another benefit of message-logging protocols that was not known before.

2.2 Forward Recovery

In some cases, the application can handle the error and execute some actions to terminate cleanly (checkpoint on failure [4]) or follow some specific recovery procedure without relying on classic rollback recovery. Such applications use some form of algorithmic fault tolerance. The application needs to be notified of the error and runs forward recovery steps that may involve access to past or remote data to correct (sometimes partially) or compensate the error and its effect, depending on the latency of the error detection.

A prerequisite for rollforward recovery is that some application processes and the runtime environment stay alive. While standard MPI does not preclude an application continuing after a fault [38], the MPI standard does not provide any specification of the behavior of an MPI application after a fault. For that purpose, researchers have developed several resilient MPI designs and implementations. The FT-MPI (fault-tolerant MPI) library [30, 43] was a first implementation of that approach. As the latest development, ULFM [5] allows the application to get notifications of errors and to use specific functions to reorganize the execution for forward recovery. Standardization of resilient MPI is complex; and despite several attempts, the MPI Forum has not reached a consensus on the principles of a resilient MPI. The GVR [25] system developed at the University of Chicago also provides mechanisms for application-specified forward error recovery. GVR design features two key concepts: (1) a global view of distributed arrays to processes and (2) versioning of these distributed arrays for resilience. APIs allow navigation of multiple versions for flexible recovery. Several applications have been tested with GVR.

2.3 Replication

Understanding of replication has progressed significantly in the past five years. Several teams have developed MPI and Charm++ prototypes offering process-level replication. Process-level replication of parallel executions is more reliable than replicated parallel executions under the assumption that the replication scheme itself is reliable. Several challenges need to be solved in order to make replication an attractive approach for resilience in HPC. First, the overhead of replication on the application execution time should be negligible. Second, replication needs to guarantee equivalent state of process replicas[1], which is not trivial because some MPI operations are not deterministic. A third, more complex challenge is the reduction of the resource and energy cost of replication. By default, replication needs twice the number of resources compared with nonreplicated execution.

One major replication overhead comes from the management of extra messages required for replication. Without specific optimization, when a replicated process sends a message to another replicated process, four communications of that message take place. rMPI addresses this problem by reducing the number of communications between replicas [31]. rMPI and MR-MPI [26] orchestrate non-MPI deterministic operations between process replicas to ensure equiva-lence of internal state. While rMPI and MR-MPI focus on process replication to address fail-stop errors, RedMPI [33] leverages process replication for detection of silent data corruptions (SDCs). The principle of RedMPI is to compare on the receiver side messages sent by replicated senders. If the message contents differ, a silent data corruption is suspected. RedMPI offers an optimization to avoid sending all messages needed in a pure replication

[1] Internal state of process replicas do not need to be identical, but external interactions of each process replica should be consistent with a correct execution of the parallel application.

scheme and to avoid comparing the full content of long messages. For each MPI message, replicated senders compute locally a hash of the message content, and only one of the replicated senders actually sends the message and the hash code to replicated receivers. Other replicas of the senders send only the hash code. Receivers then compare locally the hash code received from the replicated senders.

Reducing the resource overhead of process replication is a major challenge. One study [32], however, shows that replication can be more efficient than rollback recovery in some extreme situations where the MTBF of the system is extremely low and the time to checkpoint and restart is high. While recent progress in multilevel checkpointing make these situations unlikely, the results in [32] demonstrate that high rollback recovery overheads can lead to huge resource waste, up to the point where replication becomes a more efficient alternative. MR-MPI explores another way of reducing replication resource overhead by offering partial replication, where only a fraction of the processes are replicated. This approach is relevant when the platform presents some asymmetry in reliability (some resources being more fragile than others) and when this asymmetry can be monitored. Partial replication should be complemented by checkpoint restart to tolerate failures of non-replicated processes [22]. Some libraries, for example, the ACR library [53] for MPI and Charm++ programs, cover both hard failures and SDCs from replication.

2.4 Failure Prediction

One domain that has made exceptional progress in the past five years is failure prediction. Before 2009, considerable research focused on how to avoid failures and their effects if failures could be predicted. Researchers explored the design and benefits of actions such as proactive migration of checkpointing [27, 48, 64]. The prediction of failures itself, however, was still an open issue. Recent results from the University of Illinois at Urbana-Champaign [34, 35, 36] and the Illinois Institute of Technology [66, 37] clearly demonstrate the feasibility of error prediction for different systems: the Blue Waters CRAY system based on AMD processors and NVIDIA GPUs and the Blue Gene system based on IBM proprietary components. Failure prediction techniques have progressed by combining data mining with signal analysis and methods to spot outliers.

Some failures can be predicted with more than 60% accuracy[2] on the Blue Waters system. Further research is needed to extend the results to other subsystems, such as the file system. We are still far from the accuracy needed to switch from pure reactive fault tolerance to truly proactive fault tolerance.

Other advances concern the combination of application-level checkpointing and failure prediction [7]. An important question is how to run the failure predictor on large infrastructures. One approach is to run a failure predictor in each node of a system in order to avoid the scalability limitation of global failure prediction. This approach faces two difficulties: (1) local failure prediction will impose an overhead on the application running on the node, and (2) local failure prediction is less accurate than global failure prediction because the failure predictors have only a local view. These two difficulties are explained in [7]. Another important question is how to compute the optimal interval of preventive checkpoints when a proportion of the failures

[2] Accuracy here is defined as the prediction recall: the number of correctly predicted failures divided by the number of actual failures.

are predicted [7]. In particular, many formulations of the optimal checkpoint interval problem consider that failures follow an exponential distribution of interarrival times. This approximation may be acceptable if we consider all failures in the system. Is it acceptable for failure that are not predicted correctly and for which preventive checkpointing is needed?

2.5 Energy Consumption

A new topic emerged in the community few years ago: energy consumption of fault tolerance mechanisms. The first paper on the topic we are aware of [21] presents a study of the energy footprint of fault tolerance protocols. The important conclusion of the survey is that, at least for clusters, little difference exists between checkpointing protocols. The study also shows that the difference in energy depends mainly on the difference in execution time and only slightly on the difference of power consumption of the operation performed by the different protocols. The reason is that the power consumptions of computing, checkpointing, and message logging are close in clusters.

Some teams [50] developed models for expected run time and energy consumption for global recovery, message logging, and parallel recovery protocols. These models show in an exascale scenario that parallel recovery outperforms coordinated checkpointing protocols since parallel recovery reduces the rework time. Other researchers [1] developed performance and energy models and formulated an optimal checkpointing period considering energy consumption as the objective to optimize.

Another research issue is the energy optimization of checkpoint/restart on local nonvolatile memory [59]. The intersection of resilience and energy consumption is also explored in a recent study [60].

2.6 Mitigating Silent Data Corruptions

One of the main challenges that HPC resilience faces at extreme scale and in particular in exascale systems and beyond is the mitigation of silent data corruptions (SDCs). As mentioned in previous sections, the risk of silent data corruptions is increasing. Several studies have explored the impact of SDCs in execution results [10, 23, 47]. These studies show that a majority of SDCs leads to noticeable impacts such as crashes and hangs but that only a small fraction of them actually corrupt the results. Nevertheless, the likelihood of generating wrong results because of SDC is significant enough to warrant study of mitigation techniques.

An excellent survey of error detection techniques is presented in [46]. A classic way to detect a large proportion of SDCs (but not all) is replicating executions and comparing results. Following this approach, RedMPI [33] offers replication at the MPI process level, and replication at the thread level is studied in [65] by leveraging hardware transactional memory. The first issue with SDC detection by replication is the overhead in resources. A second issue is that, in principle, comparison of results supposes that execution generates identical results, which means obtaining bit-to-bit deterministic results from executions using same input parameters. Applications may not have this property because of nondeterministic operations performed during the execution. In general the trend toward more asynchrony and more load balancing plays against deterministic executions. Then the detection from replication becomes the problem of evaluating the similarity of results generated from replicated executions. Quantification of this similarity is an extremely hard problem because it assumes an understanding of how results diverge as a result of undeterministic operations, which itself depends on the thorough understanding of roundoff error propagation. Consequently, in SDC detection explore solutions that require less resources and potentially relax the precision of detection.

A recent direction could be called approximate replication. The principle of approximate replication is to run the normal computation along with an approximate computation that generates an approximate result. The comparison is then performed between the result and the approximate result. The approximate calculation gives upper and lower bounds within which the result of the normal calculation should be. Results outside the bounds are suspect; and corrective actions, such as further verification or re-execution, may be triggered. Approximate replication is a generic approach. It could be performed at the numerical method [5] level. It also could be used at the hardware level by comparing floating-point results of a normal operator with the ones of an approximate operator [19, 20, 49].

Another important issue related to SDC detection is the choice of methodology for evaluating and comparing algorithms. The standard process is to inject faults and obtain statistics on coverage and recovery overheads. Simulating hardware at the physical level is not feasible, and simulating it at a register transfer level is onerous. Most researchers inject faults in higher-level simulators [17]. But it is difficult to validate such fault injectors and know how the fault patterns they exhibit are related to faults exhibited by real hardware. A recent study compares different injection methods and their accuracy for the SPECint benchmark [14]. However, the accuracy of injection in HPC applications is still an open problem.

2.7 Integrative Approaches

The community has expressed in several reports the need for integrative approaches considering all layers from the hardware to the application. At least five projects explore this notion in different ways. One recent study demonstrates the benefit of cross-layer management of faults [42]. The containment domains approach [15] proposes a model of fault management based on a hierarchical design. Domains at a given level are supposed to detect and contain faults using techniques available at that level. If faults cannot be handled at that level, then the fault management becomes the responsibility of the next level in the hierarchy. Containment approaches operate between domains of the same level and between levels. Such approaches are applicable, in principle, at the hardware level and at all other software levels.

A different approach is proposed by the BEACON pub/sub mechanism in the Argo project [28], the GIB infrastructure in the Hobbes project [29], and the open resilience framework of the GVR project [25]. These mechanisms extend in different ways the concept of communication between software layers, originally proposed in the CIFTS project [40]. BEACON, GIB, and CIFTS/FTB are backplanes implementing exchanges of notifications and commands about errors and failures between software running on a system. Response managers should complement backplane infrastructures by lessening error and failure events and implementing mitigation plans.

3 Summary

This deliverable highlighted the problem of reliability for the future exascale systems and the need for further research in the area of resiliency. The reliability is widely believed to be one of the major roadblocks in exascale computing. Future systems which are mainly to be assembled from commodity components that lack sophisticated error checking and correction implemented in hardware. Unless complementary resiliency solutions are provided in software, such large HPC system would be effectively inutile because they would fail in every few minutes because of a fault.

This document surveyed the most recent progress in research of resiliency HPC. It identified the possibilities that some of the techniques can be reused in exascale and others that may simply not scale need to be revised. The next steps of the resiliency work package in MontBlanc 2013-2016 would be to drive further study the existing resiliency techniques within the context of the prototype as well as propose new resiliency techniques.

Among the many fault tolerance techniques and research directions, exploring fault tolerance within task-based parallel programming models such as OmpSs [62] seems to deliver promising results for large scale parallelism. Tasks naturally delimit the scope of the program that has to be protected and thus the boundaries where the checkpoints should be taken. In addition, checkpointing tasks is a lightweight process with a negligible overheads because it requires taking a snapshot of task's inputs only. Early experimentations with OmpSs and Nanos show promising results that we would also evaluate on the MontBlanc prototype.

Moreover, task checkpoint and recovery may have even larger potential when combined with the emerging non-volatile main memories. The task inputs and outputs can be persisted in form of a graph matching the execution order of the tasks. Then the execution can be rolled back and forth arbitrary number of tasks and thus provide not only reliability for a single task but also reliability across multiple tasks.

Acronyms and Abbreviations

- Each term should be bulleted with a definition

References

[1] Guillaume Aupy, Anne Benoit, Thomas Herault, Yves Robert, and Jack Dongarra. Optimal checkpointing period: Time vs. energy. CoRR, abs/1310.8456, 2013.

[2] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. FTI: high performance fault tolerance interface for hybrid systems. In Proc. 2011 Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC11). ACM, 2011.

[3] Austin R Benson, Sven Schmit, and Robert Schreiber. Silent error detection in numerical time-stepping schemes. International Journal of High Performance Computing Applications, April, 2014.

[4] W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. Extending the scope of the checkpoint-on-failure protocol for forward recovery in standard MPI, concurrency and computation: Practice and experience, special issue: Euro-par 2012. July 2013.

[5] Wesley Bland. User level failure mitigation in MPI. In Euro-Par 2012: Parallel Processing Workshops, pages 499-504. Springer, 2013.

[6] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni. Unified model for assessing checkpointing protocols at extreme-scale, concurrency and computation: Practice and experience. November 2013.

[7] Mohamed-Slim Bouguerra, Ana Gainaru, Leonardo Arturo Bautista-Gómez, Franck Cappello, Satoshi Matsuoka, and Naoya Maruyama. Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing. In Proceedings of IEEE IPDPS, pages 501-512, 2013.

[8] A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. Correlated set coordination in fault tolerant message logging protocols, concurrency and computation: Practice and experience. Vol. 25, No. 4:pp. 572-585, 2013.

[9] Aurelien Bouteiller, Franck Cappello, Jack Dongarra, Amina Guermouche, Thomas Hrault, and Yves Robert. Multi-criteria checkpointing strategies: Response-time versus resource utilization. In Felix Wolf, Bernd Mohr, and Dieter Mey, editors, Euro-Par 2013 Parallel Processing, volume 8097 of Lecture Notes in Computer Science, pages 420 - 431. Springer Berlin Heidelberg, 2013.

[10] Greg Bronevetsky and Bronis de Supinski. Soft error vulnerability of iterative linear algebra methods. In Proceedings of the 22nd annual international conference on Supercomputing, pages 155-164. ACM, 2008.

[11] Greg Bronevetsky, Daniel J. Marques, Keshav K. Pingali, Radu Rugina, and Sally A. McKee. Compiler-enhanced incremental checkpointing for OpenMP applications. In

Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '08, pages 275-276, New York, NY, USA, 2008. ACM.

[12] Franck Cappello, Amina Guermouche, and Marc Snir. On communication determinism in parallel HPC applications. In Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on, pages 1-8. IEEE, 2010.

[13] Sayantan Chakravorty and L. V. Kale. A fault tolerance protocol with fast fault recovery. In Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium. IEEE Press, 2007.

[14] Hyungmin Cho, Shahrzad Mirkhani, Chen-Yong Cher, Jacob A Abraham, and Subhasish Mitra. Quantitative evaluation of soft error injection techniques for robust system design. In Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, pages 1-10. IEEE, 2013.

[15] Jinsuk Chung, Ikhwan Lee, Michael Sullivan, Jee Ho Ryoo, Dong Wan Kim, Doe Hyun Yoon, Larry Kaplan, and Mattan Erez. Containment domains: A scalable, efficient, and flexible resilience scheme for exascale systems. In the Proceedings of SC12, November 2012.

[16] Gengbin Zheng, Chao Huang, and Laxmikant V. Kale. Performance evaluation of automatic checkpoint-based fault tolerance for aMPI and Charm++. SIGOPS Oper. Syst. Rev., 40(2):90-99, April 2006.

[17] Nathan DeBardeleben, Sean Blanchard, Qiang Guan, Ziming Zhang, and Song Fu. Experimental framework for injecting logic errors in a virtual machine to profile applications for soft error resilience. In Proceedings of the 2011 International Conference on Parallel Processing - Volume 2, Euro-Par'11, pages 282-291, Berlin, Heidelberg, 2012. Springer-Verlag.

[18] Gengbin Zheng, Xiang Ni, and L. V. Kale. A Scalable Double In-memory Checkpoint and Restart Scheme towards Exascale, in Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS). Boston, USA, June 2012.

[19] Peter D. Dben, Jaume Joven, Avinash Lingamneni, Hugh McNamara, Giovanni De Michel, Krishna V. Palem, and T. N. Palmer. Low-cost concurrent error detection for floating point unit (FPU) controllers. Philosophical Transactions of the Royal Society A, 20130276, 372(2018), 2014.

[20] Patrick J Eibl, Andrew D Cook, and Daniel J Sorin. Reduced precision checking for a floating point adder. In Defect and Fault Tolerance in VLSI Systems, 2009. DFT'09. 24th EEE International Symposium on, pages 145-152. IEEE, 2009.

[21] Mohammed el Mehdi Diouri, Olivier Gluck, Laurent Lefevre, and Franck Cappello. Energy considerations in checkpointing and fault tolerance protocols. In Proceedings of FTXS workshop, IEEE/IFIP DSN'12, pages 1-6, 2012.

[22] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In Proceedings of the IEEE 32nd

International Conference on Distributed Computing Systems ICDCS, pages 615-626, June 2012.

[23] James Elliott, Mark Hoemme, and Frank Mueller. Evaluating the impact of SDC on the GMRES iterative solver4. In Proceedings of International Parallel and Distributed Processing Symposium, IPDPS'14, 2014.

[24] Elmootazbellah Nabil Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B Johnson. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys (CSUR), 34(3):375-408, 2002.

[25] Ziming Zheng, Andrew A. Chien, and Keita Teranishi. Fault tolerance in an inner-outer solver: a GVR-enabled case study. In Proceedings of VECPAR 2014, Lecture Notes in Computer Science, 2014.

[26] Christian Engelmann and Swen Bohm. Redundant execution of HPC applications with MR-MPI. In Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011, pages 31-38, 2011.

[27] Christian Engelmann, Geoffroy R. Vallee, Thomas Naughton, and Stephen L. Scott. Proactive fault tolerance using preemptive migration. In Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009, pages 252-257, February 18-20, 2009.

[28] Pete Beckman et al. Argo: An exascale operating system. In http://www.mcs.anl.gov/project/argo-exascale-operating-system.

[29] Ron Brightwell et al. Hobbes - an operating system for extreme-scale systems. In http://xstack.sandia.gov/hobbes/.

[30] Graham E. Fagg and Jack Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, pages 346-353, London, UK, UK, 2000. Springer-Verlag.

[31] Kurt Ferreira, Rolf Riesen, Ron Oldfield, Jon Stearley, James Laros, Kevin Pedretti, and Ron Brightwell. rMPI: increasing fault resiliency in a message-passing environment. Technical Report SAND2011-2488, Sandia National Laboratories, Albuquerque, NM, 2011.

[32] Kurt B Ferreira, Rolf Riesen, Patrick Bridges, Dorian Arnold, Jon Stearley, H Laros III James, Ron Oldfield, Kevin Pedretti, and Ron Brightwell. Evaluating the viability of process replication reliability for exascale systems. In ACM/IEEE Conference on Supercomputing (SC11), Nov 2011.

[33] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pages 78:1-78:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[34] Ana Gainaru, Franck Cappello, and William Kramer. Taming of the shrew: modelling the normal and faulty behaviour of large-scale HPC systems. In Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS), pages 1168-1179. IEEE, 2012.

[35] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. Fault prediction under the microscope: A closer look into HPC systems. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE/ACM SC'12, page 77. IEEE Computer Society Press, 2012.

[36] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. Failure prediction for HPC systems and applications current situation and open issues. International Journal of High Performance Computing Applications, 27(3):273{282, 2013.

[37] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter Beckman. A practical failure prediction with location and lead time for Blue Gene/P. In Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on, pages 15-22. IEEE, 2010.

[38] William D. Gropp and Ewing Lusk. Fault tolerance in MPI programs. International Journal of High Performance Computer Applications, 18(3):363-372, 2004.

[39] Amina Guermouche, Thomas Ropars, Marc Snir, and Franck Cappello. Hydee: Failure containment without event logging for large scale send-deterministic MPI applications. In Proceedings of IEEE IPDPS, pages 1216-1227, 2012.

[40] Rinku Gupta, Pete Beckman, B-H Park, Ewing Lusk, Paul Hargrove, Al Geist, Dhabaleswar K Panda, Andrew Lumsdaine, and Jack Dongarra. CIFTS: A coordinated infrastructure for fault-tolerant systems. In Parallel Processing, 2009. ICPP'09. International Conference on, pages 237-245. IEEE, 2009.

[41] Paul H Hargrove and Jason C Duell. Berkeley lab checkpoint/restart (blcr) for Linux clusters. In Journal of Physics: Conference Series, volume 46, page 494. IOP Publishing, 2006.

[42] Chen-Han Ho, Marc de Kruijf, Karthikeyan Sankaralingam, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing. In Proceedings of ICPP, pages 510-519, 2012.

[43] Joshua Hursey, Richard L. Graham, Greg Bronevetsky, Darius Buntinas, Howard Pritchard, and David G. Solt. Run-through stabilization: An MPI proposal for process fault tolerance. In Proceedings of EuroMPI, pages 329-332, 2011.

[44] Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. de Supinski, and Rudolf Eigenmann. Mcrengine: A scalable checkpointing system using data-aware aggregation and compression. Scientific Programming, 21(3-4):149{163, 2013.

[45] William M. Jones, John T. Daly, and Nathan DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In Proceedings of

the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, pages 276-279, New York, NY, USA, 2010. ACM.

[46] Ikhwan Lee, Michael Sullivan, Evgeni Krimer, DongWan Kim, Mehmet Basoglu, Doe Hyun Yoon, Larry Kaplan, and Mattan Erez. Survey of error and fault detection mechanisms v2. Technical Report TR-LPH-2012-001, LPH Group, Department of Electrical and Computer Engineering, The University of Texas at Austin, December 2012.

[47] Dong Li, Jeffrey S Vetter, and Weikuan Yu. Classifying soft error vulnerabilities in extremescale scientific applications using a binary instrumentation tool. In SC12: ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, Salt Lake City, 11/2012 2012.

[48] Antonina Litvinova, Christian Engelmann, and Stephen L. Scott. A proactive fault tolerance framework for high-performance computing. In Proceedings of the 9th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2010, 2010.

[49] M. Maniatakos, P. Kudva, B.M. Fleischer, and Y. Makris. Low-cost concurrent error detection for floating-point unit (FPU) controllers. Computers, IEEE Transactions on, 62(7):1376-1388, July 2013.

[50] E. Meneses, O. Sarood, and L.V. Kale. Energy profile of rollback-recovery strategies in high performance computing. Parallel Computing, 2014.

[51] Esteban Meneses, Laxmikant V. Kal e, and Greg Bronevetsky. Dynamic load balance for optimized message logging in fault tolerant HPC applications. In Proceedings of IEEE Cluster, pages 281-289, 2011.

[52] A. Moody, G. Bronevetsky, K. Mohror, and B.R. de Supinski. Design, modelling, and evaluation of a scalable multi-level checkpointing system. In Proceedings of the 2010 International Conference on High Performance Computing, Networking, Storage and Analysis (SC10), pages 1-11, 2010.

[53] Xiang Ni, Esteban Meneses, Nikhil Jain, and Laxmikant V. Kale. ACR: Automatic checkpoint/restart for soft and hard error protection. In ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '13. IEEE Computer Society, November 2013.

[54] Xiang Ni, Esteban Meneses, and Laxmikant V. Kale. Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm. In Proceedings of IEEE Cluster'12, Beijing, China, September 2012.

[55] Akira Nukada, Hiroyuki Takizawa, and Satoshi Matsuoka. Nvcr: A transparent checkpoint restart library for Nvidia Cuda. In IPDPS Workshops, pages 104-113, 2011.

[56] Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications. Optimization of multi-level checkpoint model for large scale HPC applications. In Proceedings of IEEE IPDPS 2014, 2014.

[57] A. Rezaei, G. Coviello, CH. Li, S. Chakradhar, and F Mueller. Snapify: Capturing snapshots of offload applications on Xeon Phi manycore processors. In Proceedings of High-Performance Parallel and Distributed Computing, HPDC'14, 2014.

[58] Thomas Ropars, Tatiana V. Martsinkevich, Amina Guermouche, Andre Schiper, and Franck Cappello. Spbc: leveraging the characteristics of MPI HPC applications for scalable checkpointing. In Proceedings of IEEE/ACM SC, page 8, 2013.

[59] Takafumi Saito, Kento Sato, Hitoshi Sato, and Satoshi Matsuoka. Energy-aware I/O optimization for checkpoint and restart on a NAND flash memory system. In In Proceedings of the Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS), pages 41{48, 2013.

[60] Osman Sarood, Esteban Meneses, and L. V. Kale. A cool way of improving the reliability of HPC machines. In Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE/ACM SC'13, Denver, CO, USA, November 2013.

[61] Kento Sato, Naoya Maruyama, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R. de Supinski, and Satoshi Matsuoka. Design and modelling of a non-blocking checkpointing system. In Proceedings of IEEE/ACM SC'12, page 19, 2012.

[62] Omer Subasi, Javier Arias, Jesus Labarta, Osman Unsal, Adrian Cristal, and Barcelona Supercomputing Center. Leveraging a task-based asynchronous data flow substrate for efficient and scalable resiliency, research report of Polytechnic University of Catalonia – computer architecture department. num: Upc-dac-rr-cap-2013-12. 2014.

[63] Chao Wang, F. Mueller, C. Engelmann, and S.L. Scott. Hybrid checkpointing for MPI jobs in HPC environments. In Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on, pages 524-533, Dec 2010.

[64] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive process-level live migration and back migration in HPC environments. J. Parallel Distrib. Comput., 72(2):254-267, February 2012.

[65] Gulay Yalcin, Osman Sabri Unsal, and Adrian Cristal. Fault tolerance for multi-threaded applications by leveraging hardware transactional memory. In Proceedings of the ACM International Conference on Computing Frontiers, page 4. ACM, 2013.

[66] Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan, Rinku Gupta, Narayan Desai, Susan Coghlan, and Daniel Buettner. Co-analysis of RAS log and job log on Blue Gene/P. In Parallel & Distributed Processing Symposium (IPDPS), pages 840-85, 2011.

[67] Susmit Biswas, Bronis R. de Supinski, Martin Schulz, Diana Franklin, Timothy Sherwood, and Frederic T. Chong. Exploiting data similarity to reduce memory footprints. In Proceedings of IEEE IPDPS, pages 152-163, 2011.