SAND2009-8430C

Using Cloud Constructs and Predictive Analysis to Enable Pre-Failure Process Migration in HPC Systems

James Brandt*, Frank Chen*, Vincent De Sapio*, Ann Gentile°, Jackson Mayo*,
Philippe Pébay*, Diana Roe°, David Thompson*, and Matthew Wong°
Sandia National Laboratories
MS *9159 / °9152
P.O. Box 969, Livermore, CA 94551 U.S.A.
ovis@sandia.gov

Abstract

Accurate failure prediction in conjunction with efficient process migration facilities can enable failure avoidance in large-scale high performance computing (HPC) platforms. In this work we demonstrate a proof of concept system that incorporates our probabilistic failure prediction system with virtualization mechanisms and techniques to provide a whole system approach to failure avoidance. This work utilizes a failure scenario based on a real-world HPC case study.

1 Introduction

Infrastructure As A Service (IAAS) relies heavily on virtualization technologies and redundancy to provide reliable customized compute environments tailored to a specific customer needs. While these environments may be well suited to software development work, web hosting, and even a host of embarrasingly parallel types of calculations that have traditionally been run on high performance compute platforms, the current interconnects and individual resource reliability in these environments are such that they don't lend themselves to the kind of tightly coupled MPI applications that typify todays large scale scientific applications currently being run on HPC platforms that are specifically designed for their requirements (high reliability, fast processors, high bandwidth low latency interconnects, etc.).

High performance compute platforms, though they are tailored as described above to the requirements of large scale scientific applications, are becoming increasingly less efficient due to the large number of components. The problem is that no matter how reliable the individual components may be, the reliability of a pool of such resources taken as an aggregate can become arbitrarily low as the number in the pool becomes large. This decrease in reliability drives an increase in checkpoint frequency and efficiency goes down correspondingly as time spent in checkpointing doesn't contribute to the solution.

It is well understood that further scaling of platforms (beyond peta-scale) will require mechanisms to deal with the short mean time to failures expected over such large pools of resources [4]. While there are efforts under way to build more fault tolerant applications [19] there is significant investment in current MPI based applications which will still be required to run at significant scale on platforms for the forseeable future and hence be affected by these issues. To date most application related fault tolerance work has been in the form of more efficient checkpoint/restart schemes which has in some cases significantly increased application throughput [9]. The work presented here is based on the premise that if a failure can be predicted with enough lead time, the affected resources could be removed and replaced with good ones and the application would hence not require speculative checkpointing but could rather checkpoint when flagged to do so by the predicting system . In this work we explore the use of Cloud constructs (IAAS and virtualization) in conjunction with resource failure prediction to facilitate migration of tradtional MPI based processes from failing to healthy resources without checkpoint/restart. Though checkpoint/restart could still be used, a reliable failure prediction and migration combination should be able to significantly reduce the need for and wasted time due to checkpoint/restart.

The reason for utilizing these constructs is that the virtual machine provides a nice process state container that can be appropriately placed within the pool of real resources transparently to the application. While there is still overhead associated with the use of virtualized resources, it has

been greatly reduced through hardware support and there is promise of further reduction in the future. With sufficiently reliable failure prediction mechanisms driving migration, the overhead of virtualization has only to be less than that of the checkpointing mechanisms for this to be viable as a failure avoidance mechanism.

In this work we demonstrate use of a proof of concept system for migrating running MPI processes from a node for which failure is predicted to a known good resource including methodologies, constraints, and scaling concerns. Our failure scenario is based on previous work in which we identified precursor behavior for one of our production system's main failure modes (OOM) [3]. In particular we utilize automated detection of such behavior to trigger migration of all MPI processes off a resource identified as likely to fail in a way that is transparent to the application.

This paper is divided as follows: This section gave an introduction, background, and motivation for this work, Section 2 discusses related work, Section 3 discusses our approaches and methodologies, Section 4 covers experimental configuration and outcomes, and Sections 5 contains summary and conclusions.

2 Related Work

In this section we cover in sections work related to the specific components of our system and then similar system approaches.

2.1 Failure Prediction

While there have been many papers about how reliable failure pedictors could enhance the resilience of large scale high performance compute (HPC) platforms, most of this work has been in the area of trying to model failure trends for use in checkpoint frequency calculations (e.g., [4, 7, 12]). Schroeder has done significant work on looking at disk failure trends and predictors [13] as well as those for DRAM [14]. There are also several bodies of work on using system log files (e.g., [11, 15]) to find predictors though these have been more successful in helping to identify

2.2 Checkpoint Restart Strategies

Though the work in this area may not be directly applicable to any of the system components discussed in this paper this is certainly complementary work and until such time as accurate and effective prediction strategies are discovered this will probably continue to be the main fault tolerance mechanism for some time. There are numerous methods (e.g. [9, 5]) for more efficient checkpointing using system memory to store the checkpoint state. Though this is fine for some applications it requires the running application to have a substantially smaller footprint as checkpoint state can require significant memory resources. Oliner et. al. [10] take an interesting and related approach as they allow the system to work in cooperation with the application to take checkpoints at convenient places in the execution and dependent on the systems view of the likelihood of failure. This methodology can benefit from good resource health metrics but in the absence of these degrades to checkpointing at places of convenience for the application. This is similar to our system in which the failure prediction system advises the application that it should migrate and prepares a resource and container for the migration but lets the application initiate the actual migration.

2.3 Virtualization Overhead and Migration Strategies in HPC Applications

There are a host of papers on this subject (e.g., [17, 18, 16]) as transparent targeted process migration is very appealing given the alternative of saving all state and performing application restarts. The overhead numbers seem to range from 60% [6] to actually achieving a speedup [8]. Concensus seems to be, however, that virtualization technology and the process mobility it provides is or will be viable for HPC applications in the near future. The component that seems to be lacking in most, if not all, of this work is the accurate and effective prediction component.

2.4 Complete System

Nagarajan et. al. [8] take a systems approach to this and even incorporate a "proactive fault tolerance daemon" which does health monitoring, load balancing and decision making for virtual machine (VM) migration. The piece that is missing in this work is viable health metrics beyond the known thresholds which suffer from the problem of typically having to be set so high that by the time they are crossed it is too late to react and if they are set low enough to allow reaction time there can be significant numbers of false positives which can result in unnecessary moves and associated overhead. The significant differences between this work and our own is our probabilistic approach to health monitoring and our system for notifying a MPI process of the desirability for migration but then allowing it to request a migration when it reaches an MPI_barrier. This requires re-linking of the application code with our wrapper code. The Charm++ project [19] is a programming language approach that supports object migration for both load balancing and fault tolernace.

3 Approaches and Methodologies

There has been work, as described above, on many aspects of this problem from failure prediction, to migration. Our work in this area demonstrates a full system approach which integrates these pieces together. To do this we needed to add some functionality to the resource manager to incorporate knowledge of both physical and virtual resources and which applications are running where. We also built a Controller to interact with the failure prediction system (OVIS [1, 2]), the Resource Manager, the virtual resource controller (not shown here), and the applications in order to orchestrate migration of KVMs based on failure prediction. Additionally we wrote an MPI_Barrier wrapper to interact with the Controller in order to initiate migration at an appropriate time for the MPI applications so as not to lose in flight messages.

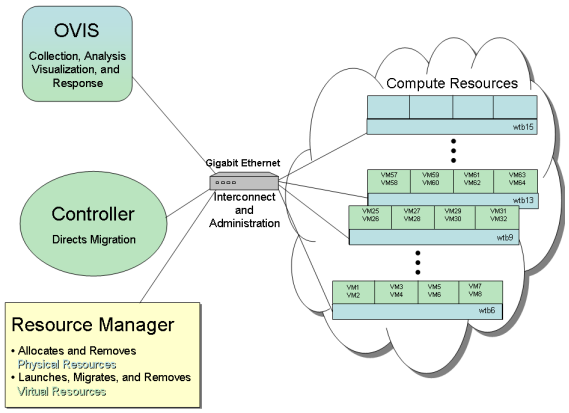

Figure 1. Components in a system to enable pre-failure process migration.

Our proof of concept (POC) system 1 is comprised of a testbed of compute nodes identical to those used in our production systems, a custom piece of code for launching, live migration, and removing Kernel Virtual Machines (KVMs) within our testbed cluster, our OVIS data collection, analysis, visualization, and response tool [1], and a Controller for orchestrating the migration of KVMs and their processes from one location to another on the basis of input from the OVIS monitoring system. In this section we discuss in detail each of these blocks's functionality and limitations.

Our testbed (referred to herafter as wtb) consists of ten diskless quad CPU 2.2Ghz AMD Istanbul (6 core) nodes. The interconnects are 4x DDR Infiniband (IB) and Gigabit Ethernet. In this proof of concept deployment we utilize the Gigabit Ethernet interconnect as we don't currently have the virtio drivers working for the IB network. This would be a limitation with respect to the performance of the MPI code but since this particular work is focused on failure avoidance through migration and not performance (which we will address in the future) it is inconsequential. On all hosts we run a linux 2.6.27.21 distribution.

With respect to the virtualized environment we are useing KVM-86, qemu-0.10.5, libvirt-0.7.0. We boot the KVMs using a OneSis image (currently same as the host image) that resides on an nfs server. When launching or migrating the KVMs, numactl is used to bind the processes associated with a KVM to a specific core and the memory region associated with that core's CPU for both performance and stability reasons. The code that manages KVM launch, migrate, and remove also takes care of virtual interface setup and teardown on the hosts as well as maintaining current state on all KVMs. This is currently driven by the resource manager in conjunction with the controller shown in figure 1.

OVIS, our data collection, analysis, visualization, and response system monitors the hardware related metrics of the testbed. This system, originally built as a research tool for large scale computer system data collection and analysis, is able to scalably collect metric data, such as memory statistics, voltages, resource utilizations, and perform various types of analyses on this data with the end result being characteristics models against which particular data can be compared for the determination of how well it matches in order to drive inference about component health and failure prediction. In this case there are two criteria, both based on the active memory metric, used by OVIS to determine if an anomalous and potentially fatal condition exists. First it determines if an anomalous condition exists using a learned and stored distribution model of active host memory while the MPI application of interest is running on wtb over the hosts involved in the run. Each time an active memory value is collected on a host it is compared with the distribution to determine a relative probability for the value. When a value is determined to be anomalous using this methodology and a user defined threshold of relative probability, OVIS then must determine if a dangerous threshold (based on measurements taken before previous OOM related failures) has been reached and if so contacts the Controller (described below) with a notification of impending failure for that particular host. Additionally OVIS is informed (currently a manual operation) of the change in the resource pool membership being monitored on behalf of the application as soon as the migration to a new resource has been completed. In this case the reason for using two criteria is that an application might be a memory hog in which case migrating processes which are using all memory on their current machines to new ones will do no good. Additionally just being an outlier in this metric doesn't necessarily imperil the application run and we only wish to incur the expense of migration if there is real danger of failure.

Finally the function of the Controller is to effect a migration of all processes from a failing resource to a healthy one. In order to do this it must first be notified, in this case by OVIS, of the impending failure of a particular component or aggregation of components. Since in the failure mode under consideration here the failure happens on a compute node granularity this is the granularity of notification. Once the Controller has been notified by OVIS of an impending resource failure it initiates migration of the affected MPI processes. In order to preclude loss of packets in flight this is done in several steps: 1a) Notification is posted by the Controller to a known location in the controllers file system. 1b) The Controller contacts the Resource Manager to acquire a new resource to host the affected KVMs. 2) An MPI_Barrier wrapper directs each rank, upon passing a barrier for the nth time (n settable by the user and set to 1 for the purpose of this work), to check for notification of the need for a migration of any associated MPI process. If no migration is required the process continues to its barrier. If a migration is required and the reading rank is unaffected it continues to its barrier. If a migration is required and the reading rank is affected it contacts the Controller to notify it that it is ready to be migrated (this is a blocking call and upon return the reading rank continues to its barrier). 3) The controller initiates migration of the KVM hosting each rank that has notified it. 4) Upon successful migration of all affected KVMs the Controller returns notification of completion (at which time the affected ranks continues to its barrier) 5) Upon successful completion of migration of all KVMs from the affected resource the Controller notifies the Resource Manager to remove the resource from the "free" pool for inspection and repair.

We demonstrate the POC system interaction to handle a failure scenario in section 4.2

3.1 Limitations and Issues

Scalability: In this deployment every MPI_Barrier call results in the rank0 process setting a semaphore, grabbing a file, releasing a semaphore and doing two MPI_Bcast calls in order for each rank to determine if a migration is required. This can be made lower impact by increasing the ratio of barriers to checks (application dependent) as well as by having the Controller push out the flag information to the local ram file system of the KVM on which rank0 resides. This infomation would have to be sent to the Controller perhaps by an MPI_Init wrapper. Using semaphores around flag writes and reads generates extra overhead but is requiered in our POC system to guarantee data corruption doesn't occur due to reading/writing conflicts which we experienced before implementing the locks. It is possible that loss of in flight barrier messages could occur during migration on large scale deployments due to propagation delays and barrier messages getting misrouted to a migrating KVM during the final phase of migration.

Other: The KVM images used in this deployment take 90 seconds to boot. Because we build the virtual infrastructure for an application at the time it is to be launched, each application must wait an initial 90 seconds, independent of the number of resources required or the time it will be run, to start. For small, short lived applications this is too much overhead and can be mitigated by maintaining a pool of virtual resources that can be allocated to such application runs in the same way as real resources are currently. The setup time for a container to migrate an already booted KVM to is sub-second and the migration process on order of seconds so maintaining such a pool and relocating as appropriate to an application would significantly reduce startup overhead.

4 Experimental Configuration and Demonstration

In this section we discuss the system functional components and their functionalities and interactions.

4.1 System Components and Configuration

Our system is comprised of five main components, four of which are illustrated in Figure 1. The high level detail of each is described below:

OVIS: This is a data collection, analysis, visualization, and response system that is used to collect metric data from the hardware and can also collect from the KVMs. In this case since we are looking specifically at active memory this is the only thing analyzed. At each collection time OVIS collects the Active Memory metric of each host. This is compared with a memory threshold (this was chosen to be 75% of the total memory) which is below that used to invoke the host's OOM killer application (90% of the total memory). At the same time the metric is compared to this threshold, the active memory distribution over all host resources involved in the application running on the threshold is learned. If the threshold is reached and the host is not a high statistical outlier (user defined at 2 std. dev. here), it is assumed that the application is just using a lot of memory and things continue. If the threshold is reached and the host that reaches it is a statistical outlier with respect to the job group it is assumed that there are memory problems on the host and the KVMs on that host are live migrated to a free host.

Resource Manager: This is a prototype resource management system that maintains a "free" resource pool, maintains a mapping between (job, virtual resource allocation, physical resources hosting the virtual resources), maintains a "defunct" resource pool, performs initial launch

of virtual resources onto physical resources on behalf of an application, starts an mpdring on the virtual resources, and launches an application on the resources.

Virtual Resource Manager (part of Resource Manager): Performs the per-KVM launch, migrate, and tear-down on the specified resources, sets up and tears down the network environment, and maintains a state directory for each virtual resource. This is used by the Resource Manager to launch or tear down a pool of KVMs and by the Controller to perform KVM live migration.

Controller: Listens to OVIS for indication of impending resource failure. Contacts the Resource Manager to determine affected KVMs. Writes updated flag files on a per job id basis, using a semaphore, that will be read by Rank0 at a barrier and shared with all processes in its job group to inform of the need to migrate. Update the Resource Manager on state of resources after a migrate.

Compute Resources: These are comprised of both the hardware and virtual resources. In this particular work we are using all virtual resources but these could be managed and allocated as virtual or real depending on the nature of the application and expected run time as discussed previously.

4.2 Demonstration

To demonstrate the POC system, we emulate a precursor symptom to major failure mode which occurs on one of our production systems. In the actual scenario, reported in [3] there is a non-application related (application isn't using too much memory) OOM condition of active memory. A discoverable precursor symptom is the active memory of a node being abnormally high with respect to others in the pool running that application. We emulate this by running a background process on a selected host that ties up a significant portion of the host memory (50% in this case). Our MPI application also includes a process for consuming memory so that we eventually meet both criteria given in section 3 for migration.

A sequence of screenshots during the POC system discovery and response interactions in shown in Figure 2. An MPI-based application was initially assigned to 2 cores (small blocks) per cpu on each of 4 cpus (white blocks) on the lower 8 nodes of the rack (upper 2 nodes are idle). In our POC system, OVIS discovers the signature of failure via statistical analysis (top figure) and reports the potentially failing resource to the Controller. The Controller then coordinates with the Resource Manager to determine new resources to which the processes on the failing resource can be migrated. The Controller then instigates the migration (middle fig). After migration the application continutes using the new resources (bottom fig). In the bottom fig, the remaining process on the 2nd from bottom node is the one

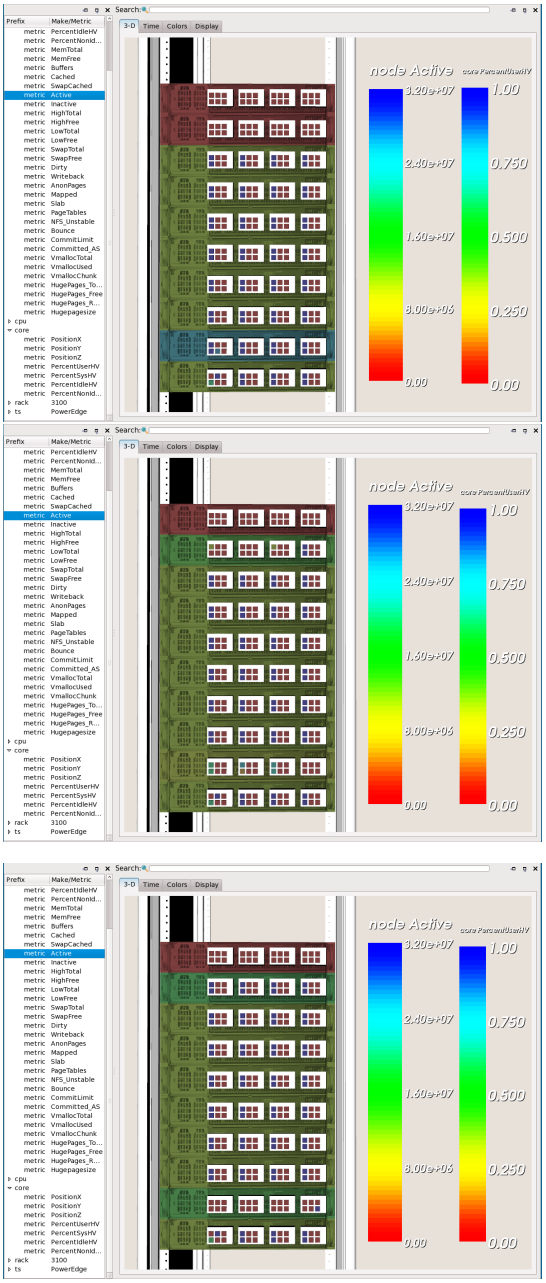

Figure 2. OVIS display showing Active Memory values on the nodes and CPU Utilization (scaled) on the cores. (top fig) OVIS discovery of abnormal memory utilization on a node (2nd from bottom) relative to the job group triggers OVIS to send a message to the Controller of impending node failure which in turn (middle fig) instigates migration of the endangered resources to a new node(2nd from top). During live migration, both the original and the new cores are in use. Migration is complete (bottom fig).

used to simulate the failure condition.

5 Conclusions

We have demonstrated the utility of using virtualization and migration in conjunction with failure prediction to facilitate failure avoidance by applications of degrading resources. This proof of concept work used a real life failure scenario found in production systems in order to illustrate such utility.

References

[1] OVIS. `http://ovis.ca.sandia.gov`.

[2] J. Brandt, A. Gentile, B. Debusschere, J. Mayo, P. Pebay, D. Thompson, and M. Wong. OVIS 2: A robust distributed architecture for scalable RAS. In *Proc. 22nd IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Miami, FL, Apr. 2008.

[3] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. Methodologies for advance warning of compute cluster problems via statistical analysis: A case study. In *Proc. 18th ACM Int. Symp. on High Performance Dist. Comp. (2009 Workshop on Resiliency in High-Performance Computing)*, 2009.

[4] J. T. Daly. Performance challenges for extreme scale computing. `http://www.pdsi-scidac.org/publications/`, Oct. 2007.

[5] C. Engelmann and A. Geist. A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. In *CLADE '03: Proceedings of the 1st International Workshop on Challenges of Large Applications in Distributed +Environments*, page 47, Washington, DC, USA, 2003. IEEE Computer Society.

[6] M. Fenn, M. A. Murphy, and S. Goasguen. A study of a kvm-based cluster for grid computing. In *ACM-SE 47: Proceedings of the 47th Annual Southeast Regional Conference*, pages 1–6, New York, NY, USA, 2009. ACM.

[7] Gottumukkala, Liu, Leangsuksun, Nassar, and Scott. Reliability analysis in HPC clusters. In *Proc. High Availability and Performance Computing Workshop 2006*, Sante Fe, NM, Oct. 2006.

[8] Nagarajan, Mueller, Engelmann, and Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proc. ACM International Conference on Supercomputing*, 2007.

[9] Oldfield, Arunagiri, T. Seelam, V. Riesen, and Roth. Modeling the impact of checkpoints on next-generation systems. In *24th IEEE Conference on Mass Storage Systems and Technologies*, 2007.

[10] A. J. Oliner, L. Rudolph, and R. K. Sahoo. Cooperative checkpointing: a robust approach to large-scale systems reliability. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 14–23, New York, NY, USA, 2006. ACM.

[11] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.

[12] B. Schroeder and G. Gibson. Understanding failures in petascale computers. In *J. Phys.: Conf. Ser. 78 012022*. IOP Publishing, 2007.

[13] B. Schroeder and G. A. Gibson. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, page 1, Berkeley, CA, USA, 2007. USENIX Association.

[14] B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer +systems*, pages 193–204, New York, NY, USA, 2009. ACM.

[15] Stearley and Oliner. Bad words: Finding faults in Spirit's syslogs. In *Proc. 8th IEEE Symposium on Cluster Computing and the Grid (2008 Workshop on Resiliency in High-Performance Computing)*, Lyon, France, May 2008.

[16] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, S. L. Scott, and A. M. Filippi. Effects of virtualization on a scientific application running a hyperspectral radiative transfer code on virtual machines. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, 2008.

[17] C. Wang, F. Mueller, C. Engelmann, and S. Scott. "a job pause service under lam/mpi+blcr for transparent fault tolerance. In *Proc. of the IEEE International Parallel and Distributed Porcessing Symposium, 2007*, 2007.

[18] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive process-level live migration in hpc environments. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[19] G. Zheng, L. Shi, and L. Kale. Ftc-charm++: an in-memory checkpoint-based fault tolerant runtime for charm++ and mpi. *IEEE International Conference on Cluster Computing*, 2004.