Toward efficient failure detection and recovery in HPC

Sunil Rani[1], Chokchai Leangsuksun[1], Anand Tikotekar[1], Vishal Rampure[1]
Stephen L. Scott[2], Richard Libby[3]
[1]Computer Science Department, Louisiana Tech University
Ruston, LA 71272, USA
[2]Computer Science and Mathematics Division, Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
[3] HPC Technical Marketing Engineer Digital Enterprise Group, Intel Corporation

[1]{ssr011, box, aat007,vdr003}@latech.edu, [2]scottsl@ornl.gov,[3]richard.m.libby@intel.com

Abstract

Application outages due to node failures are common problems in high performance computing. Reliability becomes a major issue, especially for long running jobs, as the number of compute nodes increase. Support for head node failure exists in projects like HA-OSCAR [8] and SLURM [9]. However, fault tolerance for compute node outages has not been effectively tackled. In this paper, we present a transparent fault tolerant approach and proof-of-concept solution that provides the ability to detect and self-recover parallel runtime environment in cases of compute node failure. Our solution aims to address common fault tolerance issues in the large scale system, especially due to permanent component failure in a parallel MPI [5] environment that requires a recovery with an exact runtime configuration. We proposed a lightweight heartbeat protocol (BHB) that addresses the scalability issues in system monitoring and failure detection. We also introduce a cloning technique for our recovery framework that essentially recreates an identical environment from spare nodes and thereby minimizes and controls the length of mean time to repair (MTTR) of any node outage. Our preliminary experiments show that our protocol can reduce overhead over the common approach by an order-of-magnitude.

1. Introduction

There are two central issues in large scale high performance computing (HPC) today. The first issue is concerned with the reliability for jobs that require large number of nodes and are long running. Thus, the reliability is affected due to the scalability and possible compute node failures. Mean time to repair (MTTR) for permanent failures can be significant. Parallel MPI jobs normally stall in the case of any node failures. If one of the compute node crashes the whole MPI application crashes. This happens since a typical MPI runtime system represents a static view of cooperating machines. Jobs thus need to be restarted from scratch on another set of compute nodes that has exactly the same configuration. One of the most common ways to avoid this is to checkpoint and then to restart the job from the last checkpoint state. There is a caveat however. Most of the checkpoint/restart Mechanisms such as BLCR [14] require the exact run time configuration. Moreover, the parallel run time environment such as LAM/MPI [10] has its peculiarities such as job session suffix and its checkpoint files and thus bolstering the exact run time configuration condition. This complicates the issue of restarting the job transparently from the last checkpoint. The issue is further exacerbated when required number of nodes is not available in the event of any failure. The MTTR can vary greatly with the type and nature of the failure, thereby making it difficult to restart the failed job quickly. The failed job recovery time thus becomes a function of the MTTR of the failed nodes.

The other central issue is scalability for some important services such as failure detection and system monitoring. Current system monitoring/failure detection techniques such as ganglia [1], clumon [2] tend to be heavyweight and consequently become difficult to scale. A lightweight mechanism for failure detection is thus desirable for tackling the scalability issue in large clusters.

[1] Research supported by the Department of Energy Grant no: DE-FG02-05ER25659.
[2] Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

We discuss about the related research in section 2. Section 3 presents algorithm for our fault tolerant framework; section 4 entails experiment and analysis study as well as comparisons followed by conclusion in section 5.

2. Related research

Within the project scope, our main objectives are to provide an efficient failure detection mechanism and secondly to deal with failed compute nodes running MPI environment that requires transparent checkpoint/restart mechanism.

Our first goal demands a lightweight approach for failure detection, especially in a very large scale system. Although there are quite a few existing tools for system monitoring, they may not be practical and applicable as efficient and scalable failure detection for a significant size system. Ganglia is a popular system/performance monitoring tool for clusters and grids. Clumon is a cluster monitoring system which employs SGI Performance Copilot (PCP) module [6]. Clumon consists of node monitoring daemon PCP, server daemon clumond, relational database and web portal. Supermon [11] and big brother [21] are other similar cluster monitoring systems. These tools gather metrics from nodes which normally contain performance-related and system health data. For example, Ganglia send a packet size of 36 bytes per node at 60 seconds time interval. The network traffic can quickly become overwhelming as the number of compute nodes increase. Even though, these tools acceptably perform for comprehensive system monitoring purposes, they are rather heavyweight for failure detection purposes.

In a HPC cluster environment, a head node acts as a gateway to the system. An outage at the head node renders a single point of failure (SPOF). There are some solutions that tackle the head node failure, but only very few that address the compute node failure problem, especially for MPI applications. Moreover, in downtime sensitive or mission critical environment, it is of paramount importance that the solution must be automatic and transparent from the resource manager's point of view.

Transparent head-node failure recovery exists in projects like HA-OSCAR and SLURM. HA-OSCAR is an Open source project that provides high availability and eliminates the single point of failure in a cluster environment. Lawrence Livermore National Laboratory SLURM was developed with scalability in mind coupled with fault tolerance and simplistic management. Although, SLURM has an Active/Standby server configuration for fault tolerance, it does not dictate a redundancy at the head-node like HA-OSCAR. Thus, it results in SOPF when the head-node outage occurs. Furthermore, SLURM doesn't have support for compute node failures and address a fault tolerance to running MPI applications. Linux-HA [12] is a tool for building Highly Available clusters. However, Linux-HA only provides a heartbeat and failover mechanism for a flat-structure cluster and does not provide any support for addressing compute node failures, especially in MPI environments. Commercial software like Déjà vu [20] exist that aims to solve the self restart job fault tolerance problem but not open source.

Numerous checkpoint/restart mechanisms like Epckpt [13], BLCR [14], Libckpt [15], and CoCheck [16] exist for a process-level solution. LAM/MPI [10] provides coordinated checkpoint/restart mechanisms based on BLCR. However, these mechanisms can not deal with transparent restart of the jobs in the event of any node failures. Furthermore, both BLCR and LAM/MPI requires exact run time configuration to restart the job and HA mechanism outside BLCR must exist to create it. Fault tolerant enabled MPI such as FT-MPI [17] target issues such as process failure on a compute node by re-spawning/restarting them. However, the FT-MPI requires the application developer to include a special construct similar to C++ try-catch block when a failure event occurs. LA MPI only handles network failures and suffers from the same problems as in FT-MPI. Open MPI [18] is an attempt to bring together forte of all other MPI implementations. The fault tolerance approach has not been properly carved out as yet. Kerrighed [19] uses process migration in the case of failure of any process on the compute node. A process migration solution is a fine grained approach and therefore is not concerned with providing an integrated solution with the resource manager.

Our goal is to develop a framework that enables and automates the recovery of MPI applications and does not require any or minimum modifications under the run time system. Our mechanism aims to help these checkpoint/restart mechanisms to transparently restart the jobs in addition to addressing lightweight failure detection in the large scale system. Our framework is designed to be easily plugged-in within a HPC runtime system. In next sections, we described the proposed solution, as well as our experiment and integration with HA - OSCAR.

3. Proposed solution

Our main objective is to provide a low-overhead support for node failure detection in a large scale system running MPI application that supports third party checkpoint/restart mechanism. Our proposed solution has two main phases:

1. To detect the failed node(s) efficiently
2. To recover from any failure and present an environment for a full recovery such as job fault tolerance

Let set C represent monitored nodes in the cluster = $\{C_1, C_2 \ldots, C_n\}$, Let set R representing corresponding node response set from the broadcast request = $\{R_1, R_2 \ldots, R_n\}$, Let set Fail represent the nodes that have apparently failed but need confirmation = $\{Fail_1, Fail_2 \ldots Fail_n\}$

1. Detection Phase
   - Select a tunable interval for the Broadcast heartbeat (BHB) protocol
   - Broadcast BHB to all the nodes in C
   - Retrieve the response in R
   - A = C – R
   - If A = ϕ
     - Sleep for the tunable interval
     - Goto step 1
   - End If
   - If A $\neq$ ϕ
     - Conf = A
   - End If
   - Apply point-to-point confirmation protocol to the nodes in Conf.
   - Goto step 1
2. Recovery infrastructure creation phase
   - Select a spare node from the spare nodes database
   - If the network configuration file for the spare node is found,
     - Clone the spare node with the ip.
   - If the network configuration file for the spare node is NOT found,
     - Create new conf file and transfer to the spare node
     - Configure the spare node to clone the failed one
     - Log the failed IP for analysis
     - Allocate the repaired node to the spare node's database.

Figure 1 algorithm for our framework

The broadcast heartbeat (BHB) mechanism

Our algorithm is divided into two distinct phases. Firstly, in the detection phase, we employ our lightweight protocol that relies on the low-overhead heartbeat broadcast method to send heartbeats to all the nodes in the cluster as shown in Figure 2(a). The heartbeat rate can be set using a tunable interval. A point to point confirmation protocol as shown in Figure 3(b) is applied to ensure that there are no false alarms. The strategy of using point to point (P2P) protocol after the broadcast is critical to our goal of lightweight failure detection mechanism by performing P2P confirmation only for a confirmation. This two-phase solution saves considerable overhead as detailed in the results section.

Secondly, the recovery phase as shown in Figure 4 is invoked after detecting the failure of any nodes. This phase is responsible for recreating the recovery infrastructure and spawning the exact runtime configuration on the spare nodes. We simplified the task of this phase by pre-provisioning the cloned spare nodes. The spare nodes assume the identity of the failed nodes. We conduct our experiment and compare our approach vs. existing systems in later section.

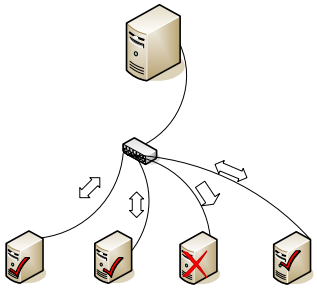

Figure 2(a) BHB Protocol

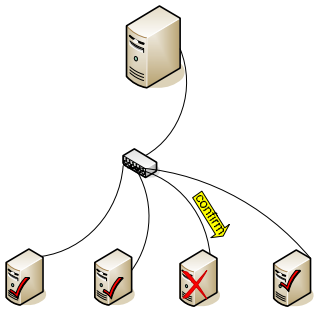

Figure 3(b) Confirmation Protocol

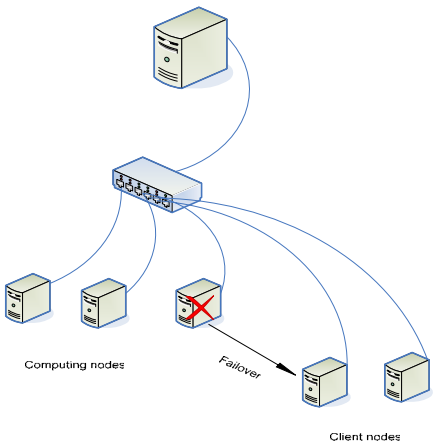

Figure 4: Recovery Phase Protocol

4. Results and analysis

Our experiment focuses on overhead study among various monitoring mechanisms. The following results discuss our findings on the failure detection mechanism. Figure 5, Figure 6, and Figure 7 show the comparison for CPU, memory and bandwidth usage, generated by experimental study on Ganglia, Clumon, BHB protocol and typical heartbeat mechanism.

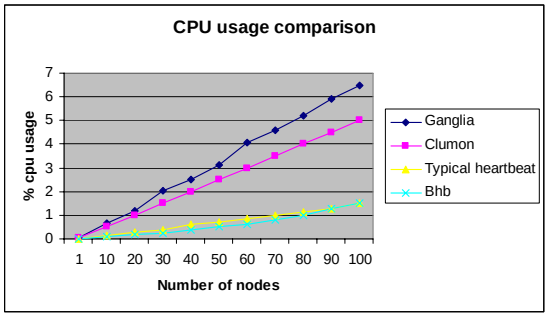

Figure 5: Ganglia, Clumon, BHB and Typical heartbeat CPU Load Comparison

The resource usage was, measured for an experimental cluster size of 1 to 8 and the rest was synthesized from a performance model. For example, the CPU usage was 0.05% for clumon and 0.0675% for ganglia. However after another node was added, the CPU usage increased by another 0.026% for clumon and 0.03 for ganglia. When scaled for about hundred nodes, the CPU usage for ganglia is about 1.5% more than that used by clumon.

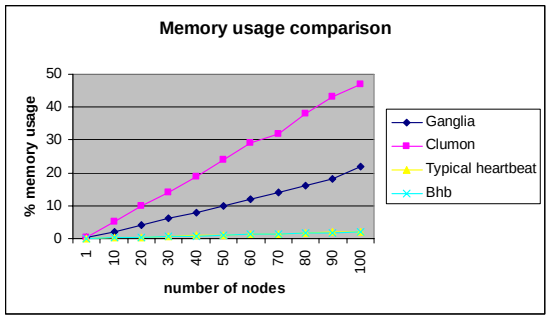

Figure 6: Ganglia, Clumon, BHB and Typical heartbeat Memory usage Comparison

Similar to CPU usage, memory usage was also measured for a cluster size of 1 to 8. The memory usage for clumon and ganglia are similar for nodes less than 6, but there is a difference of 26-30% when graphed for hundred nodes. BHB and typical heartbeat almost go hand in hand and have little CPU usage and memory usage as compared to the heavyweight application like ganglia and clumon.

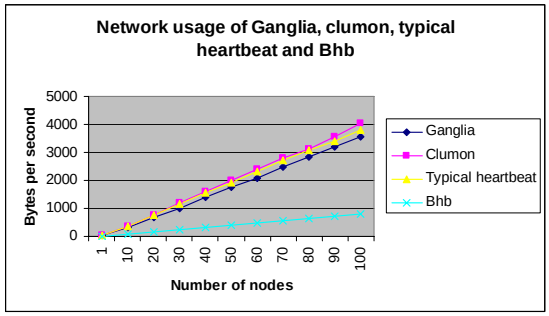

Figure 7: Ganglia, Clumon, Typical heartbeat and BHB Bandwidth usage Comparison

The bandwidth usage for ganglia is 36 bytes/sec/ node, typical heartbeat is 40 bytes/sec/node and that of BHB is 8 bytes/sec/node. The above experiments were conducted for a cluster of size 8 and then the scalability was synthesized based on the actual experimental data.

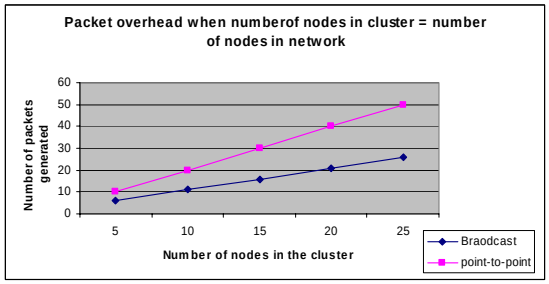

Figure 8: Packet overhead

From Figure 8 the number of packets in a case of point-to-point = 2*N whereas, the number of packets generated in case of broadcast is P + 1. The advantage of broadcast is amplified when P

= N and rapidly deteriorates when P becomes more than N. The advantage is nullified when N = (P + 1) / 2

5. Conclusion

In this paper, we have presented solutions with two main goals, efficient scalable failure detection and transparent runtime recovery. Firstly, we have demonstrated that our failure detection mechanism has considerably less overhead than most existing monitoring mechanisms. Our recovery infrastructure satisfies our main objective of fulfilling the need for a fast recovery by recreating exact runtime configuration. We also believe that our solution can be adapted to augment other MPI runtime systems, equipped with a third party checkpoint/restart mechanism.

Last, we have presented an approach that creates the necessary environment for efficient failure detection and recovery. We also have demonstrated the ability of our transparent fault tolerant approach in both departments. Our results clearly show that our detection mechanism has a very low overhead as compared to existing monitoring systems. We also took a step towards separating functionalities of metrics collection and failure detection.

6. References:

1. "The ganglia distributed monitoring system: Design, Implementation, and Experience" M.L.Massie, B.N.Chun, D.E.Culler
2. Clumon cluster monitoring system, http://clumon.ncsa.uiuc.edu
3. M.J. Brim, T.G. Mattson, and S.L.Scott, "OSCAR: Open Source Cluster Application Resources", Ottawa Linux Symposium, Ottawa, Canada, 2001.
4. "Highly Reliable Linux HPC Clusters: Self-awareness Approach" C.Leangsuksun, T.Liu, Y.Liu, S.L.Scott, R.Libby, I.Haddad, In the International Symposium on Parallel and Distributed Processing and Applications, Hong Kong.
5. A Message Passing Interface Standard, http://www.mpi-forum.org/docs/mpi1-report.pdf
6. Performance co-pilot website, http://oss.sgi.com/projects/pcp/
7. "Failure Prediction in Hardware Systems" D.Turnbull, N.Alldrin.
8. Availability Prediction and Modeling of High Availability OSCAR Cluster, IEEE Cluster 2003, Hong Kong, December 2003.
9. SLURM: Simple Linux Utility for Resource Management. http://www.llnl.gov/linux/slurm/, June 2005.
10. S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. "The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpoint." The 2003 Los Alamos Computer Science Institute Symposium, Santa Fe, NM. October 2003.
11. Matthew J. Sottile Ronald G. Minnich, Supermon: *A high-speed cluster monitoring system,* IEEE International Conference on Cluster Computing (CLUSTER'02)
12. http://www.linux-ha.org/
13. http://www.research.rutgers.edu/~edpin/epckpt/
14. J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart, 2002.
15. http://www.cs.utk.edu/~plank/plank/www/libckpt.html
16. G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In Proceedings of the 10th International ParallelProcessing Symposium, Honolulu, HI, 1996.
17. G. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic, and J. Dongarra. Process fault-tolerance: Semantics, design and applications for high performance computing. International Journal for High Performance Applications and Supercomputing, 2004.
18. E. Garbriel, G. Fagg, G. Bosilica, T. Angskun, J. J. D. J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine,R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In Proceedings, 11th European PVM/MPI Users' Group Meeting, 2004.
19. Christine Morin, Renaud Lottiaux, Geoffroy Valle, Pascal Gallard, David Margery, Jean-Yves Berthou, and Isaac Scherson. Kerrighed and data parallelism: Cluster computing on single system image operating systems. In Proceedings of Cluster 2004. IEEE, September 2004.
20. http://www.csrl.cs.vt.edu/dejavu.html
21. http://www.quest.com/bigbrother/