A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud

Ifeanyi P. Egwutuoha¹, Shiping Chen², David Levy¹, Bran Selic¹, Rafael Calvo¹

¹School of Electrical & Information Engineering, The University of Sydney, Australia
²Information Engineering Laboratory, CSIRO ICT Centre, Australia

{ifeanyi.egwutuoha, david.levy, bran.selic, rafael.calvo}@sydney.edu.au; shiping.chen@csiro.au

Abstract—Cloud computing offers new computing paradigms, capacity, and flexibility to high performance computing (HPC) applications with provisioning of a large number of Virtual Machines (VMs) for computation-intensive applications using the Hardware as a Service (HaaS) model. Due, however, to the large number of VMs and electronic components in HPC systems in the cloud, any fault during the execution would result in re-running the application, which will cost time, money and energy. In this paper we present a proactive fault tolerance (FT) approach to HPC systems in the cloud to reduce the wall clock execution time in the presence of faults. We develop a generic FT algorithm for HPC systems in the cloud. Our algorithm does not rely on a spare node prior to prediction of a failure. We analyze the dollar cost of provisioning spare nodes to assess the value of our approach. Our experimental results obtained from a real cloud environment show that the wall clock execution time of the computation-intensive applications in cloud can be reduced by as much as 30%. The frequency of checkpointing of computation-intensive applications can be reduced to 50% with our fault tolerance approach for HPC in the cloud, compared to current FT approaches.

Keywords: HPC, cloud computing, HaaS, Proactive Fault tolerance, computation-intensive application

I. INTRODUCTION

Cloud service providers such as Amazon [1] and baremetalcloud [2] offer computing resources (processors, storage, software etc.) as services. These services can be classified as Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructures as a Service (IaaS) and Hardware as a Service (HaaS) [3], [4]. HaaS allows users to access the hardware’s “whole computing power”. The users determine the operating system on the back end, as well as the number of virtual machines (VMs) running on the hardware. Research communities can easily lease HaaS for computation-intensive and/or data-intensive applications and configure HPC systems according to their needs. Consequently, computation-intensive applications that were traditionally run on HPC systems can now be executed in the cloud. These resources can be relinquished when not in use.

Fault tolerance is, however, one of the major challenges that cloud services for HPC applications face. A large set of failure data released by CFDR [5], comprising the failure statistics of 22 HPC systems, and the recent studies conducted by Schroeder and Gibson [6], show that hardware (processors, hard disk drive, integrated circuit sockets, and memory) causes more than 50% of the failures on HPC systems. As the number of processors and virtual instances increase, with associated increases in communication links and integrated circuits sockets, the likelihood of failure rises. It has been predicted that a system with 100,000 processors will experience a processor failure every few minutes [7]. A failure occurs when a hardware component is broken and needs replacement or a node/processor is halted or forced to reboot; or software has failed to complete its run. In this case, an application utilizing the failed component will fail. In addition, HPC applications deployed in cloud environments run on VMs, which are more likely to fail due to resource sharing and contention. Therefore, fault tolerance (FT) technology is particularly important for HPC applications running in cloud environments, because FT can avoid restarting, reducing thereby operational costs and energy consumption.

A reactive FT technique is commonly used for computation-intensive applications in classical grid computing through checkpoints and restart. However, it usually increases the wall clock execution time of HPC applications. Reactive FT techniques allow computation-intensive applications (which may take hours or days to complete) to log their intermediate results and states at checkpoints during their execution. Once a failure occurs, the application can be restarted from the checkpoint prior to the point of failure, rather than from the beginning. The frequency at which a component or application fails is an important measure in FT. It has been predicted that in peta-scale computing the Mean Time To Interrupt (MTTI) is short; i.e., an application running on a peta-scale system will be interrupted by failure more often, with the MTTI decreasing as the reciprocal of the number of nodes [8], [6].

In our work, we focus on Message Passing Interface (MPI) [9] applications. MPI is a parallel programming standard in which tasks executing in parallel on different processors/VMs can exchange data via messaging. It provides two modes of operation – running or failed. MPI applications, such as GROMACS (GRoningen Machine for Chemical Simulations) [23] and molecular modeling applications will greatly benefit from HPC systems in the cloud because of its scalability and the availability of...
reliable implementations as shown in [10].

This paper presents an implementation of our FT framework proposed in [11]. We use a proactive technique to provide FT. Our implementation allows a computation-intensive application with MPI implementations running in a cloud to complete its execution with reduced overhead in the presence of faults. In the next section we discuss proactive fault tolerance for HPC systems in clouds. In Section III we present the proactive FT algorithm and analysis, while Section IV presents some experimental results. Section V discusses related work. Finally, some conclusions are presented in Section VI.

II. PROACTIVE FAULT TOLERANCE FOR HPC SYSTEMS IN CLOUDS

The cloud provisions pools of computing resources as services via the Internet using a pay-as-you-go price model, that eliminates initial costly capital investments in hardware and infrastructure. Research and academic communities can leverage the benefit of the cloud price model for their computation-intensive applications that traditionally run in HPC environments. Cloud services fall into four major categories, as shown in Figure 1 [12]: Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Hardware as a Service (HaaS). HaaS providers lease out the “bare bone” hardware like computers, data servers, and storage, while cloud users are responsible for configuring and maintaining the services. Users may choose HaaS when they want full control over the server, the operating system and the software stack, as well as the number of VMs. Performance and other types of evaluation can be more easily made when there is complete control of the application that executes in the HPC environment. Cloud service providers do not commonly provide fault tolerance at this level.

Proactive FT uses an avoidance mechanism to tolerate faults. It achieves this by relying on the system log and health monitoring facilities. The system log (e.g., Reliability, Availability, and Serviceability (RAS)), and health monitoring provide information about the hardware/software state [13]. Health monitoring of hardware has recently attracted attention in fault tolerance communities because sensors are installed on modern hardware to monitor, for example, the processor temperature and fan speeds. This information is used to predict future failures.

Our proactive FT for HPC system in the cloud requires four types of modules: (1) Node monitoring module with an lm-sensor, (2) a failure predictor, (3) a proactive fault tolerance policy module and (4) the controller module. These are explained in the following sections:

A. Node monitoring with lm-sensors

Modern processors are equipped with sensors that can be used to monitor CPU temperature, fan speeds, memories and other parameters [12], [14]. We use the Lm-sensors package that provides tools, libraries, and drivers for monitoring these parameters. The libsensors library is used to access the values of the monitored parameters. It provides user-space support for the hardware monitoring drivers and console tools that report sensor readings. Lm-sensors allows easy setting of sensor limits [15]. We selected lm-sensors because most HPC systems run Linux, and lm-sensors uses Linux OS kernel drivers. We used lm-sensors to develop an FTdaemon that can be easily deployed on an HPC system in the cloud. Our methods, however, may easily be generalized to other OS platforms.

To centrally monitor the health of all the nodes in an HPC system with over 100,000 processors would impose heavy overhead on the network as well as on the HPC system. Therefore, we have designed our system to reduce the monitoring overhead, by having each node monitor its hardware by periodically reading its parameters. In our prototype, the FTdaemon running on each computing node collects lm-sensors information (e.g., processor temperature) every 600 milliseconds (the user can also set this interval to a higher value). An alarm is triggered whenever the monitored parameters exceed the maximum set values. The alarm prompts the reading of the sensors’ values and computation to determine if failure is likely to occur.

B. Failure predictor

The FTdaemon runs on each node in the user space. It uses rule-based prediction techniques to predict failure, based on the history of past failures in the system log and the maximum operating values obtained from the manufacturer's datasheet. The future failure situation is determined by periodically reading the sensors values. The current parameter values are compared against the set maximum operating conditions. For example, we assigned weights of -1, 0, and 1 to normal, maximum, and critical values respectively. The result obtained by comparing current sensors values with maximum set thresholds is used to determine if a failure is likely to occur soon.
The overhead of the failure predictor on the computation-intensive application was considered by recording the time to complete execution of the application while running FTDaemon, as well as when FTDaemon is turned off but checkpoints are still being taken.

C. Proactive fault tolerance policy

The goal of the proactive FT policy is to reduce the impact of failure on the execution of a computation-intensive application. We defined and implemented three policies: 1) lease an additional node from the service provider, 2) relinquish the unhealthy node and 3) notify the administrator to take action. When failure is predicted, the FTDaemon can proceed either to lease an additional node or to inform the administrator. The default policy is to lease an additional node and to log the details of the newly leased node with the head host. The head host maintains a database of all nodes. The functionality of the head host is transferred to newly leased node in the event of head host being predicted to fail. The “relinquish the unhealthy node policy” is executed after migration of VMs from the unhealthy to the newly leased node.

D. The controller module

The controller module implements the policies listed above. A controller module is installed on all nodes. This allows immediate action by the node that is about to fail. The FTDaemon invokes this controller module when a failure is predicted. The controller module contacts the service provider and provides the service provider with the credentials (e.g., user name and password) that are required in the leasing process. After leasing the additional node, it carries out live migration of the VMs from the unhealthy node to the newly leased node. It also logs the details of the additional node with the head host. On completion of migration of the VMs, the controller module also installs the FTDaemon on the newly leased node. Figure 2 shows the architecture of our system.

Figure 2: System Architecture.
III. PROACTIVE FAULT TOLERANCE ALGORITHM AND ANALYSIS

In this section, we describe our algorithm, and provide a quantitative mathematical analysis of its properties. The current sensors information is used to determine the state of monitored parameters. The algorithm predicts future failure, and takes action to reduce the impact of failure on the application. Finally, it also relinquishes the unhealthy node and installs an FTDaemon on the newly leased node. The algorithm is given as follows:

\[
\text{// FTDaemon running on all computing nodes } C_i (i = \{0, 1, \ldots, n\}); \\
\text{// Monitored parameters: } V = \{\text{temperature, fan speed, voltages}\}; \\
\text{// Variables } \Rightarrow \text{ operating conditions } V_a; \text{ weight } (-1, 0, 1); \\
\text{ // where: } -1 = \text{ operating at normal values of all parameters; } V; \\
\text{ // } 0 = \text{ operating at max values of one or more parameters} \\
\text{ // } 1 = \text{ operating at critical value; } CV = 1; \\
\text{// For compute node } CV; \\
\text{ FTDaemon:} \\
\text{ begin} \\
\text{ record the hostname of all guest VMs active on node } C_i; \\
\text{ set critical values of } \{CV\}; \\
\text{ read & compute:} \\
\text{ while TRUE do;} \\
\text{ read parameters } CV; \\
\text{ compute for } CV; \\
\text{ if } (CV = 1) \text{ then;} \\
\text{ break; } // \text{ exit loop} \\
\text{ elseif } (CV = 0) \text{ then;} \\
\text{ record the max } V; \\
\text{ delay; } \\
\text{ else} \\
\text{ check if alarm trigger is received; } \\
\text{ end while; } \\
\text{ controller module:} \\
\text{ begin} \\
\text{ lease additional node;} \\
\text{ live migration of } <\text{VM1}, \ldots, \text{VMn}>; \\
\text{ install FTDaemon on newly leased node;} \\
\text{ send details of newly leased node to head host;} \\
\text{ relinquish the unhealthy node; } \\
\text{ end}
\]

A. Quantitative Analysis

Case 1:

We first analyze the total dollar cost of the proactive FT algorithm used in [14], when a spare node is provisioned ahead of prediction of failure. The total dollar cost model for providing FT to computation-intensive application in the cloud with this model is:

\[
\text{Total cost} = C_{cn} + C_{sm}
\]

where:

\[
\begin{align*}
C_{cn} &= \sum_{i=1}^{n} C_i \text{ is the cost spent in leasing } n \text{ computation nodes.} \\
C_{sm} &= \sum_{i=1}^{m} C_i \text{ is the cost spent in leasing } m \text{ spare nodes.}
\end{align*}
\]

With this model the cost of running computation-intensive applications in cloud will be relatively high, due to the cost of the spare nodes. The cost implication of this model is mathematically shown in case 2.

Case 2:

A configuration is established for which the compute node \(C_{i}V_w = -1\) (as described above). In this state, there is no need to keep a spare node. From observations and records, HPC systems operate in this region most of the time, except when failure is about to occur (when a node enters its critical state (i.e., \(C_{i}V_w = 1\)). Only in this state does the controller model lease an additional node from the service provider as well as relinquish the unhealthy one. Using the above equation (1), the operating cost of the spare node is close to zero, because the unhealthy node is relinquished immediately after migration of the VMs from the unhealthy to the newly leased one. Our experimental results show that that provision of a node and live migration of four VMs takes about 20 seconds on our test system

\[
\text{Cost of spare node } = \sum_{i=1}^{m} C_i \approx 0
\]

Therefore:

\[
\text{Total dollar cost with FTDaemon } = \sum_{i=1}^{n} C_i
\]

There is a significant dollar saving with our model as can be seen from comparing Equations (2) and (1).

IV. EVALUATION

We have experimented with the characteristics of our FT design in a real cloud environment. We leased four servers from a HaaS cloud service provider [2]. Each compute node/server had the following configuration:

Dual core processor (2 x 3.5GH)
4GB memory
PC3200 3.5” SCSI 1000rpm; and
100GB network drive using iSCSI SAN [16]
Xen hypervisor [17] runs on each node. Xen hypervisor is an open source, industrial standard virtualization technology. The Linux operating system runs on top of the Xen hypervisor. We installed a para-virtualised guest OS on the nodes. A para-virtualized OS uses a modified kernel, and reduces the size of the image. Each host node is configured to host 1 to 4 VMs. Each VM is configured to have one processor, 250MB memory and 5GB hard drive. With the four compute nodes we leased, we formed a cluster of 16 nodes for testing of our algorithm.

We conducted three sets of experiments with 2, 4, 8, and 16 nodes per cluster. We ran a real HPC application, the High Performance Linpack benchmark (HPL) [18] in an OpenMPI environment.

We executed the HPL application with four different problem sizes of 2000, 4000, 6000 and 8000 on 2, 4, 8, and 16 nodes respectively. The wall clock execution time of each the problem size was recorded without checkpoint, with checkpoint, and with the FTDaemon (our proposed solution). For the tests with checkpoints, the number of checkpoints used is shown in Table 1, which also shows the number of live migration associated with each problem size. This helps to determine the effect of checkpointing on computation-intensive applications running in a cloud.

<table>
<thead>
<tr>
<th>HPL Problem Sizes</th>
<th>Number of Nodes</th>
<th>Number of checkpoints</th>
<th>Number of live migrations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4000</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6000</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8000</td>
<td>16</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1: HPL with different problem sizes and nodes

In the prototype implementation, we monitored the real-time CPU temperature as a system reliability metric with the FTDaemon running on each node. High temperature variations on the nodes affects system reliability, degrades performance, and causes failure of CPUs and circuits [12]. We simulated high temperatures on the CPU with the running HPL. We also recorded the time to lease and provision a newly leased node for migration of the VMs. The time to lease and provision a node is about 18 seconds. The average migration time obtained from our experiment was 0.315 seconds. The performance results are shown in Figure 3.

The result shows that the proposed proactive fault tolerance approach to High Performance Computing (HPC) in the Cloud significantly reduces the wall clock execution time of computation-intensive applications running in the cloud. We observed that our algorithm significantly improved application resiliency at a reduced cost compared to more common reactive approaches.

Figure 3: Performance of HPL benchmarking without checkpointing, with checkpointing, and with FTDaemon.

V. RELATED WORK

Fault tolerance techniques for HPC applications with MPI implementation can be classified into two major groups: (a) reactive FT techniques and (b) proactive FT techniques. A reactive FT technique tends to minimize the impact of failure on the computation-intensive applications in the presence of failure of one or more computational nodes. A good example of reactive FT is checkpoint and restart. Checkpoint and restart allows computation-intensive problems that may take long time to execute in HPC systems to be restarted from the point of failure in the event of errors or failures. Checkpoint and restart techniques have received a considerable attention in the past [19], [20], [21]. Their works tend to reduce the overhead caused by checkpoint and restart FT techniques to computation-intensive applications. However, recent publications [8], [6] show that with steadily increasing numbers of components in today’s HPC systems, applications running on HPC systems may not be able to achieve meaningful progress with the basic checkpoint and restart approach.

Proactive FT mitigates the effect of failure, during the lifetime of a computation-intensive application by taking proactive measures. It uses failure prediction techniques to predict future failures. The commonly used failure prediction techniques include analysis of the RAS log, and monitoring the hardware parameters such as processor temperature, fan speeds and voltages. In the pioneering work of Nagarajan, et al [14] on Proactive FT for HPC with Xen virtualization, processes are migrated from unhealthy nodes to spare nodes. However, this requires spare nodes to be always available. This technique may not be efficient in cloud computing...
because the spare nodes will be billed, and the cost of running the application in the cloud will be higher.

Our work differs from previous works in that our FT algorithm provides FT to HPC in the cloud at the hardware level at reduced cost, while running in user space (under users’ control). It does not rely on the existence of pre-configured spare nodes. It is a FT solution particularly suited to users that lease HaaS.

VI. CONCLUSION

In this paper, we have presented the design and implementation of a Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud. We analyzed the dollar cost of holding spare nodes ahead of prediction of failure. We showed that our solution does not rely on the provision of spare nodes ahead of the prediction of failure. We presented experimental results carried out in a real cloud environment. The experimental results clearly show that the proposed proactive FT approach to HPC systems in the cloud can significantly improve the execution time of computation-intensive applications running in a cloud. The dollar cost for running such applications can be reduced by as much as 30%, and the frequency of checkpointing the applications can be reduced by up to 50% with our FTDaemon. Thus, our approach can help reduce energy consumption by reducing the wall execution time of computation-intensive HPC applications.

ACKNOWLEDGMENT

The authors would like to acknowledge sponsors of FutureGrid [22]. FutureGrid cloud resources were provided free for this project.

REFERENCES

[16] [Online]. http://www.open-iscsi.org/index.html#docs