Energy Efficient Fault Tolerance for High Performance Computing (HPC) in the Cloud

Ifeanyi P. Egwutuoha*, Shiping Chen†, David Levy*, Bran Selic* and Rafael Calvo*
*School of Electrical & Information Engineering, The University of Sydney, Australia
Email: {ifeanyi.egwutuoha, david.levy, bran.selic, rafael.calvo}@sydney.edu.au
†Information Engineering Laboratory, CSIRO ICT Centre, Australia
Email: shiping.chen@csiro.au

***Abstract*—With cloud computing, a large number of Virtual Machines (VMs) can be provisioned to form high performance computing (HPC) to run computation-intensive applications using the Hardware as a Service (HaaS) model. Fault Tolerance (FT) for HPC in the cloud is increasingly a challenging issue, because any fault during the execution would result in re-running the application, which will cost time, money and energy. There has been a significant increase in energy consumption of HPC systems in cloud as a result of rerunning application and fault tolerance (e.g., redundant computing). In this paper we present energy efficient fault tolerance for HPC in the cloud. We develop a generic FT algorithm for HPC systems in the cloud. Our algorithm uses proactive process-level migration approach, however it does not rely on a spare node or redundant computing prior to prediction of a failure. Our experimental results obtained from a real cloud execution environment show that the energy utilization for HPC in the cloud while providing fault tolerance can be reduced by as much as 30%.**

***Keywords*-HPC, cloud computing, HaaS, proactive fault tolerance, computation-intensive applications, process-level migrations.**

I. Introduction

Cloud computing [1] is a revolutionary computing paradigm for storing data and running applications, including computation-intensive applications. It promises numerous benefits, which includes, wide spread of servers across the different location for disaster recovery and no upfront investments. Cloud computing also reduces development time, staff (e.g., administrators), hardware which results to significant cost saving. It is expected that more computation-intensive applications will be run in HPC system in the cloud.

There are two major key players in cloud computing: cloud providers and cloud users. Cloud providers (e.g., Baremetalcloud [2] and Amazon [3]) provide the hardware, VMs, etc as services that can be subscribed to and consumed on a pay-as-you-go basis without contracts. Cloud users are organizations or individuals (e.g., scientists) who subscribe to a selection of cloud services. It allows them either to operate their IT at reduced costs in the cloud or to create products (or perform experiments) that would be difficult without the possibilities inherent in the cloud. Figure 1 illustrates and shows the level of involvement of the key cloud players on HaaS architecture.

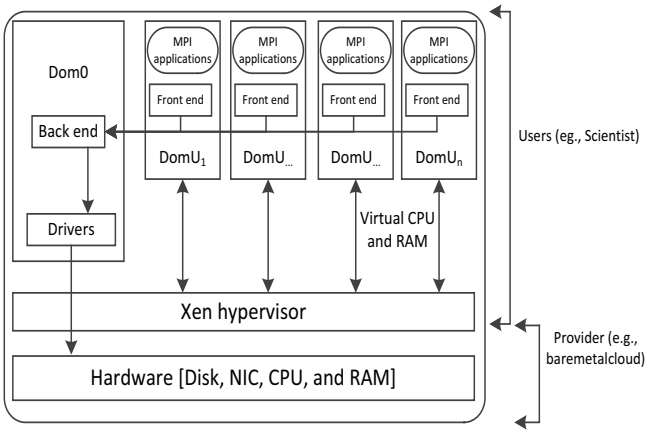

Figure 1. HaaS architecture

Based on the services provided by the cloud service provider, clouds fall into four competing categories [1], [4], [25] which are: *Hardware as a Service (SaaS), Infrastructure as a Service (IaS), Platform as a Service (PaaS), and Application as a Service (AaaS).*

This work focuses on *Hardware as a Service (HaaS)*, therefore, we briefly describe HaaS. Cloud provider basically rent out bare-bone hardware (e.g., server/host and data). An example of cloud provider that offers HaaS is Baremetalcloud [2]. The cloud users connect to this service via the Internet, install and configure (e.g., VMs) the server they leased. Cloud users choose HaaS, because it gives them full control on the server, operating system, and software stack, as well as the number of VMs they execute on it. 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

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.69

cloud. However, Cloud service providers do not provide FT to users at this level at (HaaS).

Our work focuses on providing energy-efficient fault tolerance for HPC systems in the cloud when HaaS is leased. This work is inspired by [5] and it is based on the algorithm we proposed in [6], [26]. In this work we use a proactive technique to provide FT through process-level migrations. Our implementation allows computation-intensive applications with MPI implementations running in a cloud to complete its execution with reduced energy consumption in the presence of faults. Our approach does not require an availability of spare nodes ahead of failure prediction [5] or redundant [24] computing.

In the next Section we present an analysis of failures in hpc systems. In Section III we present proactive fault tolerance for HPC systems in clouds, while Section IV presents design and implementation. Proactive fault tolerance algorithm and evaluation are presented in Section V and Section VI respectively. Section VII discusses related work. Finally, some conclusions and future work are presented in Section VIII.

II. Analysis of failures in HPC systems in the clouds

Fault tolerance is, however, one of the major challenges that cloud services for HPC applications face and HPC system in the cloud is not an exemption. They are three primary failures that occur in HPC in the cloud: hardware failure, VM failures and computation-intensive application failure. We first analyzed the failures, which occurred in HPC systems. We use large set of failure data, the computer failure data repository (CFDR) released by usenix [7], comprising the failure statistics of 22 HPC systems (18 clusters, 3 SMP and 1 NUMA). The data repository contains failure data of the HPC systems recorded for a period of 9 years at Los Alamos National Laboratory (LANL) production systems. To analyze the failure data, we selected 7 HPC systems from the failure data repository. The system selected consists of 5 clusters with highest number of totals CPUs and/or compute nodes, 1 Symmetric MultiProcessing (SMP) system with highest number of CPUs, and the only Non-Uniform Memory Access (NUMA) system on the data repository. Figure 2 shows the analysis of failure rate of HPC systems with their system IDs. We can obtain from Figure 2 that more than 60% of the recorded failures for HPC systems are hardware failures.

The recent studies [8], [9], [23] and data sets available [10], show that hardware (processors, hard disk drive, integrated circuit sockets, and memory) causes more than 50% of the failures on HPC systems. Their works also revealed that:

1) Failure rate is almost proportional to the number of CPUs (failure increases with the number of nodes and/or processors). As the number of processors and virtual instances increase, with associated increases in communication links and integrated circuits sockets, the likelihood of failure rises.
2) The intensity of the workload affect the failure rate, hence, failure rate increases with increase in the intensity of workload, and
3) There exist a time varying correlation with failure rate [11]. For example, HPC system that is seasonally used by students to run course assignment with submission deadlines.

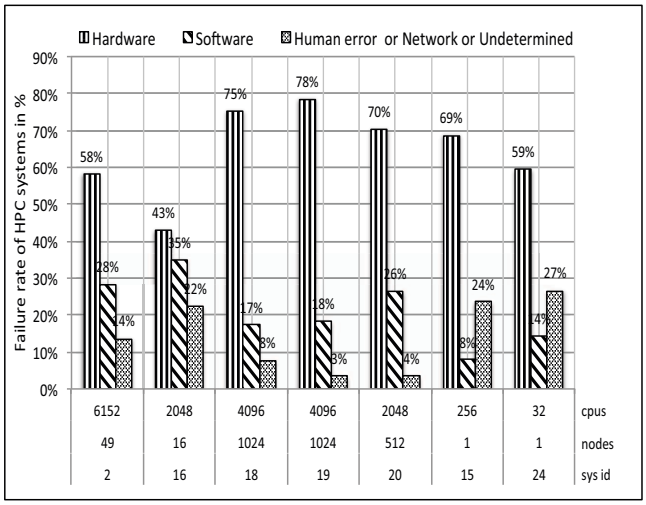

Figure 2. Failure rate of HPC systems with different CPUs and nodes

It has been projected that a system with 100,000 processors will experience a processor failure every few minutes [12]. 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 FT can avoid restarting, reducing thereby operational costs and energy consumption.

III. Proactive Fault Tolerance for HPC Systems in the Clouds

A reactive FT technique is commonly used for computation-intensive applications in classical grid computing through checkpoints and restart. This approach does not actually consider the current state of the system; hence it does not rely on the dynamic runtime of the system in order to predict future failure. However, it usually increases the wall clock execution time of HPC applications thereby increases the energy consumption. Reactive FT techniques allow computation-intensive applications (which may take

hours or weeks 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], [12].

Proactive FT uses an avoidance mechanism to tolerate faults. It accesses the monitored parameter to determine if it may impose risk on the system that it may not deliver its promised services. It achieves this by relying on the system log and health monitoring facilities to predict future failure. When future failure is predicted, actions are taken to mitigate the effects of the failure. 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. Proactive FT will play an active role in fault tolerance. The accuracy of prediction algorithms has been discussed on [14]. We used rule-based failure prediction technique to predict future failure.

In our work, we focus on Message Passing Interface (MPI) [15] 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) 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 [16].

IV. Design and Implementations

Our Energy efficient fault tolerance for High Performance Computing (HPC) in the Cloud requires four types of modules:

1) Node monitoring module with an lm-sensor
2) A rule-based failure predictor
3) Migration policy module and
4) The controller module.

The primary goal of this design and implementation is to provide a FT for HPC in the cloud with energy efficient capability. We have considered how to achieve this without significant impact to performance of the HPC system. Figure 3 shows the fundamental architecture of our proposed solution. Xen hypervisor [17] is virtualization software, which is installed on each of the host/server. This allows multiples paravirtualized OSs to be installed on each host. The $Domain_0$ (the host OSs) provides the infrastructure for the four modules monitoring, prediction, control, and migration policies. They run the management console and have special privileges to access the hardware. The backend and FTDaemon communicate to the hardware through the drivers. The $DomU_0, \ldots, DomU_n$ (unprivileged domains) are the guest VMs(compute nodes). The guest VMs is configured to form a cluster. The guest VMs executes the computation-intensive applications. In the following section, we explain the FT modules in details.

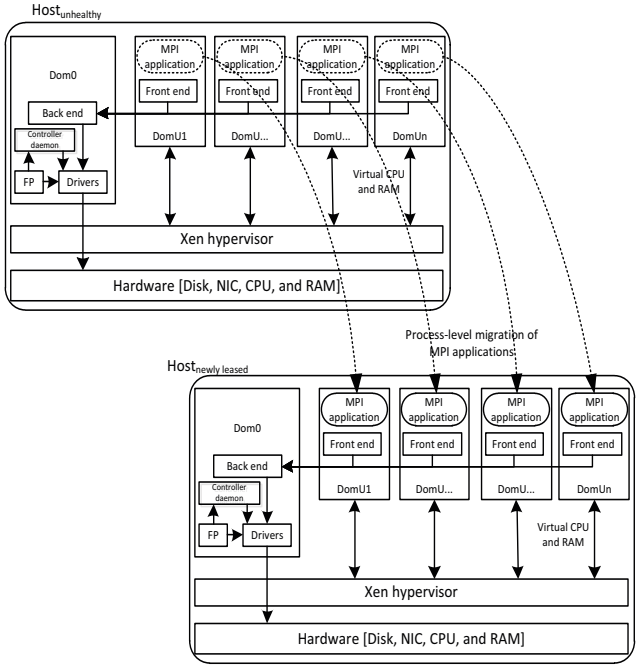

Figure 3. Fundamental architecture of our proposed solution

A. Node monitoring with lm-sensors

Node monitoring with lm-sensors Lm-sensors is an open-source software tool for monitoring the health of modern computers. Modern processors are built with sensors that can be used to monitor CPU temperature, fan speeds, memories and other parameters [18]. 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 [19]. 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). However, FTDaemon uses more computer resources if the frequency of the collection of the monitored parameters is high. The information can be published to head host if the user activates this feature. 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. Rule-based 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 data sheet. The future failure situation is determined by periodically reading the sensors values and CPU utilization. Figure 4 illustrates the predictor model. The predictor has a vector input of the four parameters: temperature T, voltage V, fan speed F, and CPU utilization C. We assigned weight values to the read parameter values in order to compute the systems operating state. The current state are compared against the set maximum operating state to determine if the current operating state is failure prone. For example, we assigned weights of 1, 2, and 3 to normal, warning, and critical values respectively.

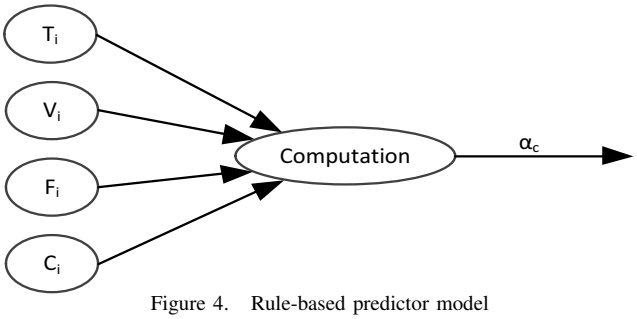

Figure 4. Rule-based predictor model

The weights assigned are used to perform the calculation using the following equation, which we have derived.

$$\alpha_c = \prod_{i=1}^{n} T_i.F_i.V_i.C_i + K_c \quad (1)$$

where;

$$\alpha_c = \begin{cases} \textit{critical state} \text{ for} & |\alpha_c| > 24 \\ \textit{warning state} \text{ for} & |\alpha_c| = 24 \\ \textit{normal state} \text{ for} & |\alpha_c| < 24 \end{cases} \quad (2)$$

The result α_c is obtained by computation of current sensors values and CPU utilization. We associate the input valuables with weight as already state, therefore $T_i \in$ [1, 2, 3], $F_i \in$ [1, 2, 3], $V_i \in$ [1, 2, 3] and $C_i \in$ [1, 2, 3]. For example, after reading the sensor's values and CPU utilisation, if the weight of T_i = 3, F_i = 3, V_i = 2 and C_i = 2. Hence, the product of $T_i.F_i.V_i.C_i$ = α_c = 3x3x2x2 = 36. Therefore, α_c is critical and immediate action is required. We generalize the result α_c in (1). The result α_c is used to compare with the maximum set thresholds to determine if a failure is likely to occur with the present state. Equation (2) is the threshold *24* which we used based on the system log and manufacturer's data sheet. We introduced the constant K_c which can be used to improve the accuracy of prediction. In our implementation, we set K_c = 0. The migration policy is activated when the system is operating at critical state. The overhead due to 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. We observed that there is no significant impact on the system when FTDaemon is running.

C. Migration policy

The goal of the proactive FT policy is to reduce the impact of failure on the execution of a computation-intensive application while computation-intensive complete execution with minimum energy utilization. We defined and implemented four polices:

1) Lease an additional node,
2) Relinquish the unhealthy node
3) Publish the critical state to the head host
4) Inform the system administrator.

When future 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 publish the details of the newly leased node to the head host. The head host maintains a database of all hosts and compute nodes (VMs). 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 host policy is executed after process-level migration of the computation-intensive applications from the unhealthy to the newly leased node.

D. The controller module

The controller module implements the policies listed above. This is the module for leasing, migrating processes and relinquishing of the host. A controller module is installed on all the hosts; we call this approach a decentralized technique. The controller module has the facility to lease and execute the process-level migration. The unhealthy host initiates the migration when it received message from its FTDaemon to do the migration. The details of the compute nodes to migrate their process to is also provided by the FTDaemon. We prefer this approach (decentralized technique) to the head host (central control) approach because all the information necessary to make decision resides on each host. Moreover, head host approach will utilize more bandwidth compare to decentralized approach. This allows immediate action by the host that is failure prone. The FTDaemon invokes this controller module when a failure is predicted. The controller module algorithm shown is shown in algorithm 1. The protocol of the approach is summarized below:

1) It determines the number of the VMs on the unhealthy host by querying the information centre on host.
2) It also obtains the process ID that are running on the unhealthy VM from the information centre on the host.
3) It proceeds to lease an additional host from the service provider by providing all the details required (e.g., user name and password)
4) Starts VMs on the newly leased host based on the number of the VMs running on the unhealthy host
5) Perform process-level migration of the computation-intensive applications.
6) It relinquish the unhealthy node and
7) Publish details of the newly leased host and the VMs running on it to the head host.

Algorithm 1 Controller module

```
1: Controller module <-- service provider
2: if (lease = ipaddress) then
3:    Start VM(n) <-- start   # start n number of VMs
4:                            # on the newly leased host
5:    if (VM(n) = VM(number of nodes on unhealthy host)) then
6:       end if
7:    end if
8:    for all nodes in unheathy host  do
9: U_node_ip <--  N_node_ip  # pair VMs on unhealthy host and
10:                          # VMs on newly leased host
11: select_node <-- process
12: while TRUE do
13: migrate --> N_node_ip
14: else
15:    error
16: end while
17: Unhealthy host <-- relinquish the node
18: Newly leased host --> headhost
```

V. Proactive Fault Tolerance Algorithm and Analysis

In this section, we describe our algorithm, and provide a quantitative analysis of its properties. The current sensors information is used to determine the state of system. The algorithm predicts future failure, and takes action to reduce the impact of failure on the applications. The actions include leasing of additional node from service provider, process-level migration of the applications, and publishing of the warning state to the head host. Finally, it also relinquishes the unhealthy node and installs an FTDaemon on the newly leased host. The algorithm is given as algorithm 2

A. Energy consumption analysis

Model 1:

We first analyze the total energy utilization of the proactive FT algorithm used in [5], [24], when a spare host is provisioned ahead of prediction of failure or redundant computing technique is used to provide FT to HPC systems. Energy *E*, is equal to the product of power *P* and time *T*. *E = P x T*. The total energy utilization for providing FT to computation-intensive application in the cloud with this model is expressed with the following equation:

$$\textit{Total energy critical state } T_e = E_h + E_s \tag{3}$$

where;

$$T_e = \sum_{i=1}^{n} E_{hi} + \sum_{j=1}^{k} E_{sj} \tag{4}$$

- E_{hi} is the energy utilization of n computation hosts
- E_{sj} is the energy utilization of k spare nodes.

Analyzing this model, the energy utilization in running computation-intensive applications in cloud with fault tolerance provided will be relatively high, due to the energy utilization of the spare nodes or redundant computing. Based on the server details obtained from the HaaS service provider, we assumed that each host/server consumes 150Watt. We assume that at least one spare node is available base on the [5, 24]. Table 1 shows the sampled analysis and energy utilization for each model during the execution of HPL application.

Model 2:

A configuration is established for which the host is not operating at critical state (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 host enters its critical state). 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 energy utilization of the spare node is close to

Algorithm 2 FTDaemon

```
1:  # FTDaemon running on all host Hi (i = {0, 1,..., n});
2:  # Monitored parameters: α = {temperature, fan speed, voltages and
3:  # CPU utilization};
4:  # Variables => current operating state {αw}: weight (1, 2, 3);
5:  # where: 1 = operating at normal value of the parameter
6:  #          2 = operating at max value of parameters
7:  #          3 = operating at critical value
8:  # For host Hi;
9:  FTDaemon:
10:    begin
11:       record the ipaddress of all guest VMs active on host Hi
12:    read & compute:
13:       while TRUE do;
14:          read parameters α
15:          assign weight to α
16:          compute for αc;
17:       if (αw < 24) then;
18:          break;    # exit loop
19:       elseif (αw = 24) then;
20:          record the max α;
21:          delay;
22:       else
23:       check if alarm trigger is received;
24:       end while;
25:    controller module:
26:       begin
27:          lease additional node;
28:          process-level migration
29:          install FTDaemon on newly leased node;
30:          publish details of newly leased node to head host;
31:          relinquish the unhealthy node;
32:       end
33:    end
```

zero, because the unhealthy host is relinquished immediately after the process-level migration of the application from the unhealthy to the newly leased one. Our experimental results show that that provision of a host and process migration takes few seconds on our test system.

$$\textit{Energy utilization of spare host, } E_{sj} = \sum_{j=1}^{k} E_{sj} \approx 0 \quad (5)$$

Therefore;

Total energy utilization of computation with FTDaemon

$$T_e = \sum_{i=1}^{n} E_{hi} \quad (6)$$

VI. Evaluation

We have experimented with the characteristics of our FT design in a real cloud environment. We leased five servers from a HaaS cloud service provider [2]. Each compute server/host 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. We installed Xen hypervisor [17] runs on each host. Xen hypervisor is an open source, industrial standard virtualization technology. Each host is configured to host 1 to 7 VMs. Each VM is configured to have one processor, 250MB memory and 5GB hard drive. With the five host we leased, we formed a cluster of 2, 4, 8, 16 and 32 nodes for testing of our algorithm. We installed OpenMPI [27] on each VM. We used OpenMPI process-level migration tool available in OpenMPI implementation.

We conducted two sets of experiments with the nodes cluster to determine the CPU utilization of our FTDaemon; and the functionality of our algorithm. We used the power rating obtained from the service provider to calculate the energy utilization based on our solution and compare it with the solution proposed in [5] and redundant computing [24]. The energy rating from the manufacture data was used because we could not have physical access to the HaaS provided measure the utilization with wattmeter. We ran a real HPC application, the High Performance Linpack benchmark (HPL) [20] in an OpenMPI environment. We executed the HPL application with five different problem sizes of 2000, 4000, 6000, 8000 and 10000 on 2, 4, 8, 16 and 32 nodes respectively. Table 1 summaries the relationship between the problem sizes and the VMs numbers.

Table I
HPL with different problem sizes and nodes

Problem size	VM	Host	Spare host (sh)	Energy utilization with sh (kJ)	Energy utilization with FTDaemon (kJ)
2000	2	1	1	111	55.5
4000	4	2	1	185.85	123.9
6000	8	3	1	379.8	284.85
8000	16	4	1	918	734.4
10000	32	5	1	1926.9	1605.75

The wall clock execution time of each the problem size was recorded while we execute process-level migration with spare nodes and with the FTDaemon (our proposed solution). The result obtained from the experiments is shown in figure 5. This helps to determine the energy utilization of cluster when a spare node is positioned ahead of the prediction.

To determine the CPU utilization of FTDaemon on each host, we profile the CPU utilization. We observed that there is no significant impact on the system due to the FTDaemon.

In the prototype implementation, we monitored the real-time CPU usage; CPU temperature and CPU utilization as system reliability metrics with the FTDaemon running on each host. The variation of CPU temperature and CPU utilization (work load) affects system reliability, degrades performance, and causes failure of CPUs and circuits [18], [9]. We simulated high temperatures on the CPU and high CPU utilization 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 host with our pre-configured OS from Baremetalcloud [2] is about 18 seconds, this time may varies if the provider allows their unleased HaaS to be on hibernation state. We assume that the time after failure prediction is enough to lease a host and to perform process migration. The performance results are shown in Figure 5.

From the experimental results, we can observed that there is a signification reduction on the energy utilization as shown in figure 5. We also observed that our algorithm significantly improved application FT at a reduced energy utilization compared to more commonly used approaches. The energy utilization HPC system in the cloud can be significantly reduced by 30% with our algorithm.

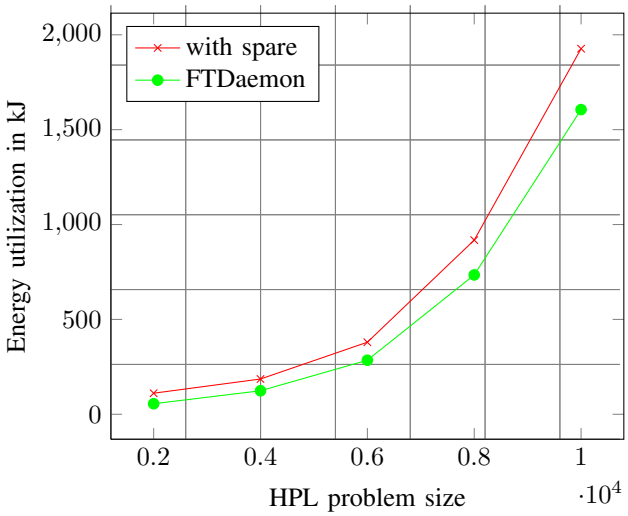

Figure 5: Energy utilization of FT with spare node compared with FTDaemon.

VII. 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 [21], [22], [23]. Their works tend to reduce the overhead caused by checkpoint and restart FT techniques to computation-intensive applications. However, recent publications [12], [9] show that with steadily increasing numbers of components in todays HPC systems, applications running on HPC systems may not be able to achieve meaningful progress with the basic checkpoint and restart approach.

Redundant computing has recently been proposed [24] to reduce wall-clock time of computation-intensive application running in HPC systems in the presence of failure. In redundant computing, compute nodes are replicated twice. The replication of the compute nodes increases the energy utilization and and overhead while running computation-intensive application in HPC systems in the cloud. Redundant computing is however not energy efficient FT.

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. They have been significant improvement to accuracy and time lapses before the actual failure occurred is enough to do process-level migration.

They are recent works on proactive FT, which uses live process-level migration and VM migration techniques [6], [5]. VM migration techniques has been shown to have more overhead compare to process-level migration. Most of the present FT works were not designed with Energy efficient in mind nor designed for HPC system in cloud computing. In the pioneering work of Wang *et al* [5] on Proactive FT for HPC with Xen virtualization, processes are migrated from unhealthy nodes to spare hosts. However, this requires spare nodes to be always available. The work also proposed that the process can be migrated to the less loaded node. Generally, migrating processes to less loaded node has higher probability of making the less load node overload. The work was also not designed for clouding computing or with energy efficient in mind because the cost of keeping spare host will be high if the techniques is adapted for HPC system in cloud computing.

Our work differs from previous works in that our FT algorithm provides FT to HPC in the cloud at the hardware level when HaaS is leased. It does not rely on the existence of spare nodes as proposed in [5]. Our algorithm provides energy efficient FT for HPC system in the cloud, while running in user space (under users control). Our work is designed to run on open mpi and MPI environments. It is a FT solution particularly suited to users that lease HaaS. Furthermore, we also analyzed the failure rate of HPC systems and showed that hardware is the major contributor of failure in HPC systems.

VIII. Conclusion and Future Work

In this paper, we have presented the design and implementation of an Energy efficient fault tolerance for High Performance Computing (HPC) in the Cloud. We analyzed the large failure data repository CFDR provided by usenix. We showed that hardware failure is commonly experienced in HPC system especially when the number of compute node runs in thousands. We analyzed the energy utilization

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 or on redundant computing. 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 reduce the energy consumption of computation-intensive applications running in a cloud. Our solution compliments checkpoint/restart solution. 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 30% because spare host is not provisioned.

In the future work, we will determine the accuracy of the prediction model and also the mean time to failure (MTTF) after failure prediction.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz,A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia. ''Above the Clouds: A Berkeley View of Cloud computing,'' University of California at Berkley, 2009.

[2] Baremetalcloud. (2013) http://baremetalcloud.com/index.php/en/

[3] Amazon. (2012). http://aws.amazon.com/ec2/

[4] B. P. Rimal, E. Choi, and I. Lumb, ''A Taxonomy and Survey of Cloud Computing Systems,'' in Fifth International Joint Conference on INC, IMS and IDC, 2009, pp. 4451.

[5] 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, Tampa, FL, 2008.

[6] I. P. Egwutuoha, S. Chen; D. Levy, Bran Selic and R. Calvo, ''A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud,'' in The 2nd International Conference on Cloud and Green Computing, Xiangtan, Hunan, China, 2012, pp. 268 - 273.

[7] CFDR. (2012). http://cfdr.usenix.org

[8] B. Schroeder, and G.A. Gibson, ''Understanding failures in petascale computers,'' in Journal of Physics: Conference Series, 78:012022, 2007

[9] B. Schroeder, and G. Gibson, ''A large-scale study of failures in high performance computing systems,'' IEEE Transactions On Dependable and Secure Computing, vol. 7, no. 4, pp. 337-351, Oct 2010

[10] Failure trace archive. (2013). http://fta.inria.fr

[11] N. Yigitbasi et al, ''Analysis and Modeling of Time-Correlated Failures in Large-Scale Distributed Systems,'' in International Conference on Grid Computing, 2010, pp. 65-72.

[12] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir., ''Toward Exascale Resilience,'' International Journal of High Performance Computing Applications, vol. 23, no. 4, pp. 374388, 2009.

[13] A. Oliner and J. Stearley, ''What supercomputers say: A study of five system logs,'' in In DSN 07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Washington, DC, USA, 2007, pp. 575584.

[14] F. Salfner, M. Lenk, and M. Malek, ''A survey of online failure prediction methods,'' ACM Comput. Surv., vol. 42, no. 3, pp. 1 - 42, 2010.

[15] MPI Forum, ''MPI: A message-passing interface standard,'' 1994.

[16] Constantinos Evangelinos, and Chris N. Hill, ''Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2,'' 2008.

[17] Xen hypervisor. http://www.xen.org/products/xenhyp.html

[18] A. Kumar, L. Shang, L. Peh, and N. Jha., ''System-Level Dynamic Thermal Management for High-Performance Microprocessors,'' vol. 27, no. 1, pp. 96108, 2008.

[19] Lm-sensors, (2013), http://lm-sensors.org/wiki/Documentation

[20] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary. (2008, Sept) HPL, http://www.netlib.org/benchmark/hpl/

[21] E.N.M. Elnozahy, L. Alvisi, Y.M. Wang, and D.B. Johnson, ''A Survey of Rollback-Recovery Protocols in Message-Passing Systems,'' vol. 34, no. 3, 2002.

[22] J. T. Daly, ''A higher order estimate of the optimum checkpoint interval for restart dumps,'' Future Generation Computer Systems, vol. 22, pp. 303312, 2006.

[23] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, ''A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems,'' The Journal of Supercomputing (2013): 1-25, 2013. http://checkpointing.org/

[24] R. Riesen, K. Ferreira, J. Stearley, ''See applications run and throughput jump: The case for redundant computing in HPC.'' In: 1st International Workshop on Fault-Tolerance for HPC at Extreme Scale, FTXS 2010 (2010)

[25] I. P. Egwutuoha, D. Schragl, R. Calvo, ''A Brief Review of Cloud Computing, Challenges and Potential Solutions,'' Parallel & Cloud Computing (PCC), Vol.2, No.1, 2013, pp. 7-14.

[26] I. P. Egwutuoha, S. Chen, D. Levy, & B. Selic, ''A Fault Tolerance Framework for High Performance Computing in Cloud'', In Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on (pp. 709-710). IEEE.

[27] OpenMPI, http://www.open-mpi.org/