2009 International Conference on Availability, Reliability and Security

Proactive Resource Management for Failure Resilient High Performance Computing Clusters

Song Fu
Department of Computer Science
New Mexico Institute of Mining & Tech., NM USA
song@cs.nmt.edu

Cheng-Zhong Xu
Department of Electrical & Computer Engineering
Wayne State University, MI USA
czxu@eng.wayne.edu

Abstract

Virtual machine (VM) technology provides an additional layer of abstraction for resource management in high-performance computing (HPC) systems. In large-scale computing clusters, component failures become norms instead of exceptions, caused by the ever-increasing system complexity. VM construction and reconfiguration is a potent tool for efficient online system maintenance and failure resilience. In this paper, we study how VM-based HPC clusters benefits from failure prediction in resource management for dependable computing. We consider both the reliability and performance status of compute nodes in making selection decisions. We define a capacity-reliability *metric to combine the effects of both factors, and propose the* Best-fit *algorithm to find the best qualified nodes on which to instantiate VMs to run user jobs. We have conducted experiments using failure traces from the Los Alamos National Laboratory (LANL) HPC clusters. The results show the enhancement of system dependability by using our proposed strategy with practically achievable accuracy of failure prediction. With the* Best-fit *strategies, the job completion rate is increased by 10.5% compared with that achieved in the current LANL HPC cluster. The task completion rate reaches 82.5% with improved utilization of relatively unreliable nodes.*

1 Introduction

Recently, virtual machine (VM) technologies are experiencing resurgence in both industry and academia. They provide desirable features to meet demanding requirements of computing resources in modern computing systems. Performance isolation, ease of management, checkpointing, migration, OS customization, and security are among the many features that make VM a powerful and popular technique for high performance computing (HPC) [7, 10, 11].

Most of existing works on virtual machine based HPC focus on infrastructure design [7, 11, 3] and reducing virtualization overhead [20, 15, 4]. However, few has investigated the reliability issue, such as avoiding, coping and recovering from failures, which is one of the hardest problems in HPC systems. The coming petaflop systems will require the simultaneous use and control of hundreds of thousands or even millions of processing, storage, and networking components. With this large number of components involved, component failures will be frequent, making it increasingly difficult for applications to make forward progress.

Results from a recent work [9] show that the existing reliability of large HPC clusters is constrained by a mean time between failures (MTBF) in the range of 6.5 - 40 hours, depending on the maturity of the installation. Failure occurrence as well as its impact on system performance and operation costs are becoming an increasingly important concern to system designers and administrators [27, 13, 30]. The success of petascale computing will depend on the ability to provide dependability at scale.

Current techniques to tolerate faults focus on reactive schemes where fault recovery commonly relies on checkpointing/restart mechanisms. However, checkpointing a job in a large-scale HPC system could incur significant overhead. The LANL study [19] estimated the checkpointing overhead based on current techniques to run a 100 hour job (without failure) by an additional 151 hours in petaflop systems. As a result, frequent periodic checkpointing often proves counter-effective.

As the scale and complexity of HPC systems continue to grow, research on system dependability has recently shifted onto failure prediction and proactive autonomic management technologies [17, 5, 16, 13, 8, 6, 22]. It is desired to construct an adaptive computing infrastructure, which adjusts its configuration dynamically at runtime according to

978-0-7695-3564-7/09 $25.00 © 2009 IEEE
DOI 10.1109/ARES.2009.13

components' availability. A most important feature provided by modern VM techniques is their ability of reconfiguration through VM migration [4, 26]. It allows system administrators to move a virtual machine instance to another physical node without interrupting any hosted services on the migrating instance. It is an extremely powerful management tool to provide efficient proactive failure management, online system maintenance and load balancing in high-performance computing clusters [21, 4].

In this paper, we investigate the reliability issue in resource management of VM-based HPC clusters. We propose proactive VM construction algorithms by leveraging failure prediction results for dependable computing. We consider both the reliability and performance status of compute nodes in making selection decisions. We define *capacity-reliability* metrics to combine the effects of both factors in node selection, and propose the *Best-fit* algorithm to find the best qualified node on which to instantiate a virtual machine for parallel jobs. We evaluate the performance of our proposed algorithm by using failure and workload traces from production systems, the Los Alamos National Laboratory (LANL) HPC clusters. Experimental results show that the proposed strategy enhances the system dependability significantly. By using the *Best-fit* strategy, the job completion rate is increased by 10.5% compared with that achieved in the current LANL HPC cluster. The task completion rate reaches 82.5% with improved utilization of relatively unreliable nodes.

The rest of this paper is organized as follows: Section 2 describes the distributed virtual machine framework. Section 3 discusses failure resilient system configuration. We present the proposed proactive resource management strategies in Section 4. The experimental results are included in Section 5. Section 6 discusses the related works. Conclusions and remarks on future works are presented in Section 7.

2 Distributed Virtual Machine Framework

A distributed virtual machine (DVM) consists of a set of virtual machines running on top of multiple physical compute nodes. Each compute node hosts multiple virtual machines running user jobs. These virtual machines multiplex resources of the underlying physical node. The *virtual machine monitor* (VMM) is a thin layer that manages hardware resources and exports a uniform interface to upper guest OSes. Existing VMM products include Xen [2] and VMware's ESX Server [28], and more.

A virtual machine (VM) comprises a guest OS and runtime support services for user jobs running on top of them. A VM encapsulates execution states of services and running jobs. It is the unit of instantiation in DVM construction.

VMs on a compute node are managed locally by a *DVM daemon*, which is also responsible for communication with the *LoadManager*, *FailurePredictor* and *Coordinator*. The DVM daemon runs in a privileged domain, *Domain0*. It collects performance data of local VMs, and then sends them to the *LoadManager*, which analyzes workload distribution and issues VM workload reports. Based on the performance data and history occurrences of failure events, the *FailurePredictor* computes the statistics of failures and forecasts when the next failure is going to occur in the future. These VM workload reports and failure predictions are sent to the *Coordinator*, which is responsible for making DVM reconfiguration decisions. It also receives job execution requests from users, and selects compute nodes to construct a DVM. In large-scale HPC systems, compute nodes are coupled into clusters, which may further be interconnected into grids. In such large systems, there are multiple Coordinators, FailurePredictors and LoadManagers. Each of them manages a portion of the entire system. They form a hierarchical structure to manage resource and VMs in node, cluster and system wide.

A user job is composed of multiple parallel tasks, which are executed on different VMs. Multiple VMs running tasks of different jobs may reside on a compute node in a multiprogramming environment. An abortion of a task makes the corresponding job uncompleted. However, VM migration provides a potent approach to tolerating node failures. Distributed virtual machines are created, reconfigured, and released at runtime.

The execution status of a virtual machine is monitored. Node failures and overload are handled at runtime by means of VM migrations. Physical compute nodes are treated as a single, unified pool. Service pre-installation is not needed. The distributed virtual machine infrastructure is resilient to dynamics of its computing environment. In the following sections, we will describe the compute node selection strategies by which we choose qualified nodes to instantiate virtual machines in DVM construction.

3 Failure Resilient Distributed Virtual Machine Configuration

After a user job is submitted with its computation requirement to a HPC system, the system *Coordinator* evaluates qualifications of active compute nodes. It selects a set of them for that job, initiates the creation of VMs on them, and then dispatches the job tasks for execution. In making selection decisions, the Coordinator needs to look into the performance status of each active node, such as processor and memory capacity and utilization, and its workload level. Moreover, the reliability status of a node also needs to be considered, because this factor affects the extent to which the performance of a node will be actually achieved and how long its performance can sustain.

When we refer to a "failure" in the following discussion, we mean any anomaly caused by hardware or software defect, incorrect design, unstable environment or operator mistakes that makes services or compute nodes unavailable.

To derive selection strategies in DVM construction, we formulate the selection problem as follows. Let $(a_1, a_2, \ldots, a_k)$ denote k tasks that job A requests for execution in a HPC system with N compute nodes. Let

$$\begin{aligned} \mathrm{w}_i(t) &= \textit{remaining workload of task } a_i \textit{ at time } t \\ \mathrm{d}_i &= \textit{deadline of completing } a_i \\ \mathrm{c}_j(t) &= \textit{available capacity of node } j \textit{ at time } t \\ \delta_j(t) &= \textit{time to next failure forecast by the FailurePredictor for node } j \textit{ at time } t \end{aligned}$$

If we concern only with the finish time of the entire job, we can set d_i equally to the desired deadline. Otherwise, each d_i can have a different value.

The Coordinator selects k compute nodes to execute the k tasks of job A. To estimate reliability of a node, the FailurePredictor forecasts when the next failure will occur in that node [5]. Next, we take both performance and reliability status of compute nodes into account in making DVM construction decision. We use t_0 to denote the time when the *Coordinator* dispatches an execution request from a user job A. The Coordinator first examines a list of available compute nodes, and then selects a group from them according to certain criteria. Without loss of generality, we assume $(n_1, n_2, \ldots, n_k)$ is the group of nodes selected for job A. The *FailurePredictor* forecasts when next failures will occur on them, *i.e.* the time-between-failures $(\delta_1(t_0), \delta_2(t_0), \ldots, \delta_k(t_0))$ [1].

To select a compute node, say n_j, to execute a job task a_i, the Coordinator considers the reliability status of node n_j. The deadline of task a_i can be either before the predicted occurrence time of n_j's next failure or after, depending on their values. For the first case, where $d_i \leq t_0 + \delta_j(t_0)$, task a_i can complete its execution on node n_j if this node has enough capacity, that is the available capacity of node n_j is ample enough to complete the computation of task a_i before this node comes to a failure. This is expressed as

$$\int_{t_0}^{d_i} c_j(t)\, dt \geq w_i(t_0), \tag{3.1}$$

where the available capacity of this node $c_j(\cdot)$ changes with time.

For the second case, a failure is predicted to occur on node n_j before the deadline of task a_i. That is $d_i > t_0 + \delta_j(t_0)$. It is observed that the average time between failure of a compute node are larger than four months in production systems [27, 23]. In addition, the average execution time of user jobs in production HPC systems much less than this large time-between-failure. For example, among the 74,463 finished jobs between Nov. 20 2003 and Oct. 23 2005 in a LANL HPC cluster, 90% of them completed execution within 9 hours and less than 1% of them lasted for more than a week [1]. As a result, the probability that a job task will experience two node failures in its execution is very small. This is verified by our observation in many production HPC systems. For instance, among the 494 jobs affected by failures in the LANL cluster, 37 (7.5%) of them experienced more than one failure during their execution. Within the 615 tasks of those jobs, only 4 (0.6%) of them suffered from failures twice [1]. Therefore, it is sufficient for us to assume that there is at most one failure that might occur in the lifetime of a job task.

Based on the preceding observations and considerations, a virtual machine migration will happen in the second case. Execution states of task a_i along with its underlying supporting systems will be moved to another compute node and a_i will resume execution there. Assume node n_k is selected to accommodate this migrated VM for task a_i. Then, nodes n_j and n_k can be selected to run a_i, if they have enough capacity to complete the task's workload before deadline, *i.e.*:

$$\int_{t_0}^{t_0+\delta_j} c_j(t)\, dt + \int_{t_0+\delta_j+\tau}^{d_i} c_k(t)\, dt \geq w_i(t_0), \tag{3.2}$$

where τ is the overhead of VM migration caused by a failure which is predicted to occur on node n_j. Its value depends on the size of working set generated by a job task [4] and the size of supporting systems including the underlying guest OS. By stop and copy migrations, the migration overhead is 12-14 seconds with a 700 MB guest OS, as measured in our experiments. Compared with 8.28 hours of average job execution time [1], this migration overhead is negligible. To provide a practical estimation of the migration overhead τ, we use the measurement of overhead in a previous VM migration to set the value of τ in a future migration for the same type of guest OS.

In (3.1) and (3.2), the capacity of a node, $c_\cdot(t)$, changes with time. It is very difficult, if not impossible, to have an accurate prediction of the available capacity of a compute node in a dynamic environment. To make problem tractable, we use the measurement of available capacity of a node at the time of making selection decision to approximate its capacity in the makespan of a job task. As a result, the preceding two equations can be re-written as

$$c_j(t_0) \cdot (d_i - t_0) \geq w_i(t_0), \text{ if } d_i \leq t_0 + \delta_j(t_0) \tag{3.3}$$

and

$$c_j(t_0)\cdot\delta_j + c_k(t_0+\delta_j+\tau)\cdot(d_i - t_0 - \delta_j - \tau) \geq w_i(t_0), \text{ if } d_i > t_0 + \delta_j(t_0). \tag{3.4}$$

[1] Originally, δ_i refers to the predicted interval between the time (t_l) when the last failure occurs and that (t_f) of the next one on node n_i. Since we know the difference between t_0 and t_l, we refer $\delta_i(t_0)$ to the interval between t_f and t_0.

We introduce a Bernoulli random variable $o_{i,j}$ as $o_{i,j} = 1$, if $d_i > t_0 + \delta_j$. Otherwise, $o_{i,j} = 0$. It indicates whether a failure is predicted to occur on node n_j during the execution of task a_i, or not. With $o_{i,j}$, (3.3) and (3.4) are combined into

$$(1 - o_{i,j}) \cdot c_j(t_0) \cdot (d_i - t_0) + o_{i,j}[c_j(t_0) \cdot \delta_j + c_k(t_0 + \delta_j + \tau) \cdot (d_i - t_0 - \delta_j - \tau)] \geq w_i(t_0). \quad (3.5)$$

The value of $o_{i,j}$ is determined by comparing the predicted occurrence time of the next failure in node n_j and the deadline of task a_i.

4 Distributed Virtual Machine Construction Strategies

A user job with parallel tasks runs on multiple compute nodes. The tasks may experience node failures in their lifetime. Many of the failures, such as those caused by hardware faults or system software faults, are isolated. That is, they only compromise single nodes. For this type of failures, we can migrate the affected tasks to other available nodes so that the corresponding job can continue execution instead of getting aborted. There are also failures that are correlated with each other, such as those caused by software bugs in jobs' programs or by environment turbulences. In those cases, we either stop the jobs to fix their bugs before restarting, or migrate the parallel tasks to other nodes that are not affected by the turbulences.

So far, we have described the criteria in choosing compute nodes to instantiate virtual machines for parallel tasks. We now present strategies to select nodes based on these criteria with different objectives. The goal in constructing these node groups is to maximize the rate of jobs (and tasks) that complete successfully, and the utilization of available resources in terms of performance and reliability.

In the following discussion, we leverage failure prediction techniques in construction and reconfiguration of DVMs in HPC systems. A failure will cause an affected compute node to repair. The parallel tasks running on that node will be migrated to other available nodes for tolerating node failures. The tasks that run on other nodes are not affected. They will remain on those nodes as long as the reliability status and available capacity of the nodes satisfy the tasks' computation requirement.

4.1 Random-Select

The Random-select strategy randomly assigns nodes to a group of size k to instantiate virtual machines and run the k tasks of job A. Each node in the group is assigned a task of A. This is the baseline strategy for our system model as it does not guarantee that the tasks will finish computation before deadline. This algorithm does not use the reliability status of compute nodes in an intelligent way. For a given pool of nodes and a set of parallel tasks of a job, this strategy will form a group with fixed number of nodes, irrespective of $\delta_.(t)$.

As a result, the reliability status of the group is a random variable that takes any value between 0 and 1, depending on which nodes are selected. Since nodes are selected at random, we cannot ensure that job A will be completed within its desired deadline. In (3.5), $o_{i,j}$ can be either 0 or 1 and VM migrations may occur within the makespan of job A.

4.2 First-Fit

In the First-fit strategy, for each task a_i in job A, the available compute nodes are divided into two sets. Nodes in the first set, S_1, have reliability status which ensures there is no failure predicted to occur before the deadline of task a_i, *i.e.* $S_1 = \{n_j | d_i \leq t_0 + \delta_j(t_0)\}$. The rest of nodes fall into the second set. For nodes within set S_1, we sort them according to their available capacity in a decreasing order. Then, we select the first node, say n_j, in the sorted list to create a VM for task a_i. After that, we update the available capacity of node n_j by considering the workload of task a_i. Other tasks can perform the First-fit algorithm in a similar way.

Intuitively, First-fit attempts to form a group of compute nodes that will not fail during the execution of tasks $\{a_i\}$ in a greedy manner. The reliability status of the group is predicted to be close to 1 with high probability. The most powerful node is selected for each task, which ensures that job A will be completed as soon as the system can. In (3.5), $o_{i,j} = 0$ and the capacity of n_j is the largest one that minimizes the finish time of task a_i in

$$c_j(t_0) \cdot (t_{f_i} - t_0) = w_i(t_0),$$

where t_{f_i} is the finish time of task a_i.

4.3 Best-Fit

The Best-fit algorithm attempts to form a group of nodes for tasks of job A, so that utilization of system resources is maximized in a failure-aware manner.

We focus on those compute nodes which have a "good" reliability status. The *Best-Fit* selection algorithm attempts to find the "best" node for each task from a set of nodes that are predicted not to fail before a task completes its execution.

The set of candidate nodes is $S = \{n_j | t_0 + \delta_j(t_0) \geq d_i\}$. To guarantee that the execution of task a_i will catch its deadline, we have $c_j(t_0) \cdot (d_i - t_0) \geq w_i(t_0)$ as expressed in (3.3). The minimum capacity satisfying the above inequality is

$$c_j^*(t_0) = \frac{w_i(t_0)}{d_i - t_0}.$$

Then, we can search the candidate set S for the node n_j whose available capacity is equal to or marginally above the minimum value $c_j^*(t_0)$.

Although the preceding algorithm can find the node with minimum available capacity, it may assign very reliable nodes to short-running tasks and cause other long-running tasks not being able to find suitably reliable nodes. For example, tasks a_1 and a_2 request computation of 10 and 40 hours respectively, and node n_2 is a reliable one that will not have a failure in the next 6 days by prediction. Its available capacity meets the minimum requirement of task a_1 and is selected for this task. However, this may result in that task a_2 cannot find any node that will not fail within the next 40 hours and also has enough capacity.

To tackle this problem, we consider both the reliability status and capacity levels of candidate nodes in searching for the best one. We use an adjustable parameter $\alpha \in [0, 1]$ and define *capacity-reliability* metric as

$$cr = \alpha \frac{c_j(t_0) - \frac{w_i(t_0)}{d_i - t_0}}{\frac{w_i(t_0)}{d_i - t_0}} + (1 - \alpha) \frac{\delta_j(t_0) - d_i(t_0) + t_0}{d_i(t_0) - t_0}. \quad (4.1)$$

Note that α is the weight put on the two factors in making selection decisions. As α increases and approaches 1, cr is becoming more *capacity-biased*. The Best-fit algorithm attempts to find the node with the least available capacity among quantified candidates. If α gets to 0, then cr is *reliability-biased*, and the node that has the least reliability status among quantified candidates will be selected. The value of α can be dynamically adjusted at runtime depending on the status of resource utilization and nodes' reliability.

In the Best-fit algorithm, we first construct the set of quantified nodes $S = \{n_j | c_j(t_0) \geq w_i(t_0)/(d_i - t_0)\ AND\ \delta_j(t_0) + t_0 \geq d_i\}$ for task a_i at time t_0. Among nodes in set S, we select the one that has the least value of *capacity-reliability* cr, according to (4.1). Then, a virtual machine will be instantiated on the selected node to run task a_i.

When a failure is predicted/detected at runtime in the execution of a parallel job, a renewal process will be applied to perform the Best-Fit algorithm and try to find the best node in a pool of available nodes. Then, the virtual machine from the failing node will be migrated to the newly selected best available node and computation proceeds.

5 Performance Evaluation

We implemented the *Random-select*, *First-fit*, and *Best-fit* strategies to evaluate their performance in improving the dependability of HPC clusters. We measured the *job completion rate* as the proportion of jobs finished by their deadlines among all submitted jobs, the *task completion rate* as the proportion of finished parallel tasks among all tasks. The reason for us to measure both the *job completion rate* and the *task completion rate* is that some tasks may complete their execution although their corresponding jobs fail, and this amount of computation used to complete those tasks is valid from the system's perspective, although it is not delivered to the jobs' owners.

5.1 Workload and Failure Models

In our experiments, we used job logs and failure traces from the Los Alamos National Laboratory (LANL) HPC clusters system [1]. The data span 22 high-performance computing systems that have been in production use at LANL between 1996 and 2005. In total, the system includes 4750 nodes with 24101 processors, and experienced 23,739 failures.

To concentrate on analyzing the influence of reliability status on node selection and to reduce distractions from non-essential factors, we focus on a typical cluster in the system: Cluster 20. It has 512 compute nodes. There were 307 failures occurred on its 256 active nodes between September 1, 2004 and August 31, 2005. In total, 112,454 user jobs were submitted to the cluster in that one-year period. Among them, 74,463 (66.2%) were finished, while 30,194 (27.0%) were killed due to failures or errors. Each job requested 1 - 256 nodes to execute its tasks.

5.2 Experimental Results

In the first set of experiments, we simulated the behaviors of compute nodes in the LANL HPC cluster by using its job, performance and failure traces [1]. We then evaluated the performance of our proposed failure-aware DVM construction and reconfiguration strategies. We adopted the failure predictor that we proposed in [5] to forecast the occurrence time of future failures. The failure predictor uses an order-8 neural network prediction algorithm. We trained the predictor by using the failure measurements from September 2003 to August 2004, and predicted failure occurrences from September 2004 to August 2005. The average prediction accuracy achieved by the predictor is 76.5%, and the prediction error due to false negative is 3.7% [5]. To evaluate the sensitivity of those algorithms to the accuracy of failure prediction, we also generated synthetic traces of failure predictions with different accuracy. We used those predicted results to determine the time-between-failure δ_i for each node n_i.

The available capacity of a compute node is approximated by subtracting the normalized workload of all running tasks from the processing capacity of that node. We extract the data of submission time, execution deadline [2],

[2] In the job traces of LANL HPC clusters, execution deadline was spec-

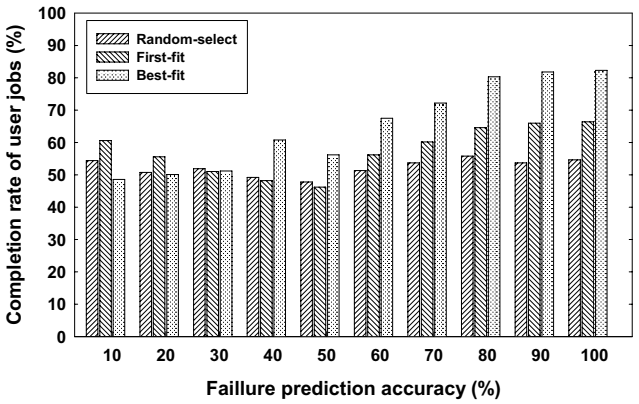

Figure 1. Percentage of successfully completed user jobs.

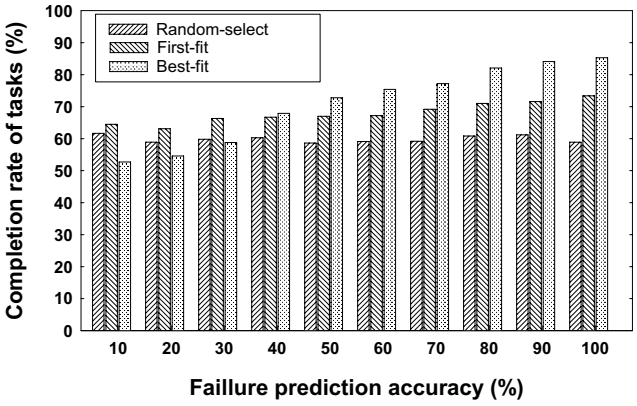

Figure 2. Percentage of completed tasks.

and the number of tasks of each job from the job traces. We use those workload data, failure predictions, node capacity information in our experiments. The VM migration overhead is 12 seconds with a stop and copy migration as we measured in our experiments.

We first evaluate the *job completion rate*, that is the percentage of user jobs that successfully complete before their deadlines. This is the performance of a HPC system that is perceived by users. Each job spawns multiple tasks on a DVM. A job is considered completed only if all of its tasks finish execution before its deadline. Figure 1 shows the job completion rates achieved by the four strategies. We use synthetic traces of failure predictions with different prediction accuracy in applying these strategies. From this figure, we observe that by using the *Best-fit* (α = 0.5), more user jobs successfully complete their execution compared with the other two strategies. The Best-fit strategy is relatively sensitive to the accuracy of failure predictions. It does not perform well with bad predictions. However, as the prediction error is less than 40%, it achieves better *job completion rate*. By using failure predictions generated by our predictor [5] with a 76.5% prediction accuracy, we achieve 80.2% job completion rates by the *Best-fit* strategy. Compared with the 69.7% job completion rate achieved in the current LANL HPC cluster, the performance improvement is significant.

In addition to *job completion rate*, *task completion rate* is another important metric in evaluating the productivity of a HPC system. A node failure or other critical events may cause a task to be aborted, while other tasks of the same job may continue running and finish. The amount of system resources consumed by those tasks is valid from the system's perspective. The completed tasks need to be counted in the productivity of the system. Figure 2 presents the *task completion rates* achieved by using the four DVM construction and reconfiguration strategies. They follow a similar trend of changes to that of the *job completion rate* as the prediction accuracy increases. The *Best-fit* strategy perform more efficiently in utilizing system resources and mitigating the impact of failures on running tasks. It outperforms the *Random-select* and *First-fit* strategies by more than 21.3% and 12.2% respectively, as the accuracy of failure prediction reaches 60%. By using the failure prediction trace with 76.5% of accuracy, we improve the *task completion rate* to 82.5% by using the *Best-fit* strategy.

ified for each job. Thus, we set the deadline of a task equal to that of its associated job.

6 Related Works

Recently, interest in using virtual machines (VM) as the abstraction for cluster computing and for high-performance computing in general has been growing [7, 10, 11]. Virtual machine monitors, such as Xen [2] and VMware [28] have the potential to greatly simplify management from the perspective of resource owners and to provide great flexibility to resource users.

VMs also provide powerful mechanisms for failure resilience [21], as they can be created on compute nodes and a VM can be migrated to other available nodes when a failure is detected. Most of the existing virtual machine monitors provide this reconfiguration mechanism, such as VM live migration in Xen [4]. The performance of VM migrations has been extensively evaluated in [26, 4]. Advanced techniques have been proposed to reduce its overhead. VM migration is still an expensive operation. Therefore, the frequency of migration in a failure-prone computing environment needs to be kept low.

Noticeable progress has been made on failure management research, and failure analysis [27, 14, 24] reveals fail-

ure characteristics in high performance computing systems. Sahoo et al. [22] inspected the event set within a fixed time window before a target event for repeated patterns to predict the failure event of all types. Liang et al. [13] profiled the time-between-failure of different failure types and applied a heuristic approach to detect failures by using a monitoring window of preset size. Mickens and Noble [16] assumed independency of failures among compute nodes and used the per-node uptime data to predict whether a failure might occur on that node in the next time window of fixed size. Fu and Xu [5, 6] exploited the temporal and spatial correlations among failure events to predict the occurrences of failures in HPC systems. Most of these works focused on improving the prediction accuracy, and few of them considered how to leverage predication results for resource management in practice.

Salfner et al. [25] mentioned that proactive failure management has the potential to improve system availability up to an order of magnitude, based on their analysis of prediction accuracy. The FT-Pro project [12] demonstrated a significant performance improvement for long-running applications provided by proactive fault tolerance policies. Heuristic job scheduling and quality of service negotiation algorithms considering the effect of failures were developed for BlueGene/L system, and simulation studies showed that the use of these new algorithms could significantly improve the system performance [18]. Sun et al. [29] modeled the the effect of failures on the application completion time and applied checkpointing technique for fault tolerance.

However, there lacks formal approaches to analyzing the impact of failure events on resource management decisions, and to developing effective resource allocation strategies and algorithms for high-performance computing by leveraging failure prediction techniques.

7 Conclusions

Large-scale high performance computing clusters are susceptible to hardware and software failures, which significantly affect the system performance and management. We propose a proactive distributed virtual machine construction strategy for failure resilient HPC systems. In addition to performance states of candidate nodes, we also consider their reliability status in resource management. We have evaluated the performance of the proposed strategy by using failure traces from production systems. Experimental results show the enhancement of system dependability by using the *Best-fit* strategy with practically achievable accuracy of failure predictions.

In this work, we have analyzed the impact of accuracy of failure predictions on the performance of the proposed strategies. Currently, we only consider the cases in which the predicted occurrence time of future failures precedes the observed time. However, it is also possible that the opposite cases may happen. A VM migration is not suitable in those cases. A checkpointing mechanism is helpful instead. Although failure prediction cannot forecast all possible failures without error, the prediction results are useful in reducing the checkpointing frequency. As a future work, we will investigate the integration of failure prediction and checkpointing to enhance the availability and productivity of HPC systems.

Acknowledgments We would like to thank the anonymous reviewers for their constructive comments and suggestions. This research was supported in part by U.S. NSF grants CCF-0611750, DMS-0624849, CNS-0702488, CRI-0708232, and NASA grant 03-OBPR-01-0049.

References

[1] System availability, failure and usage data sets. Los Alamos National Laboratory (LANL). Available at: http://institutes.lanl.gov/data/fdata/.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of ACM Symp. on Operating Systems Principles (SOSP)*, 2003.

[3] B. Calder, A. A. Chien, J. Wang, and D. Yang. The entropia virtual machine for desktop grids. In *Proc. of ACM/USENIX conf. on Virtual Execution Environments (VEE)*, 2005.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of ACM/USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[5] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proc. of ACM/IEEE Conf. for High Performance Computing, Networking, Storage, and Analysis (SC'07)*, 2007.

[6] S. Fu and C.-Z. Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *Proc. of IEEE Symp. on Reliable Distributed Systems (SRDS)*, 2007.

[7] W. Goscinski and D. Abramson. Motor: A virtual machine for high performance computing. In *Proc. of IEEE Symp. on High Performance Distributed Computing (HPDC)*, 2006.

[8] N. R. Gottumukkala, C. Leangsuksun, N. Taerat, R. Nassar, and S. L. Scott. Reliability-aware resource

allocation in HPC systems. In *Proc. of IEEE Conf. on Cluster Computing (Cluster)*, 2007.

[9] C.-H. Hsu and W.-C. Feng. A power-aware run-time system for high-performance computing. In *Proc. of ACM/IEEE conf. on Supercomputing (SC'05)*, 2005.

[10] W. Huang, J. Liu, B. Abali, and D. K. Panda. A case for high performance computing with virtual machines. In *Proc. of ACM Intl. Conf. on Supercomputing (ICS)*, 2006.

[11] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *Proc. of ACM/IEEE High Performance Computing, Networking and Storage Conf. (SC'04)*, 2004.

[12] Y. Li and Z. Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. In *Proc. of IEEE Intl. Symp. on Cluster Computing and the Grid (CCGRID)*, 2006.

[13] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. K. Sahoo. BlueGene/L failure analysis and prediction models. In *Proc. of Conf. on Dependable Systems and Networks (DSN)*, 2006.

[14] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a BlueGene/L prototype. In *Proc. of Intl. Conf. on Dependable Systems and Networks (DSN)*, 2005.

[15] J. Liu, W. Huang, B. Abali, and D. K. Panda. High performance VMM-bypass I/O in virtual machines. In *Proc. of USENIX Annual Technical Conf. (USENIX)*, 2006.

[16] J. W. Mickens and B. D. Noble. Exploiting availability prediction in distributed systems. In *Proc. of USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2006.

[17] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *Proc. of the ACM Intl. Conf. on Supercomputing (ICS)*, 2007.

[18] A. J. Oliner and J. E. Moreira. Probabilistic QoS guarantees for supercomputing systems. In *Proc. of IEEE Conf. on Dependable Systems and Networks (DSN)*, 2005.

[19] I. Philp. Software failures and the road to a petaflop machine. In *Proc. of Symp. on High Performance Computer Architecture Workshop*, 2005.

[20] H. Raj and K. Schwan. High performance and scalable i/o virtualization via self-virtualized devices. In *Proc. of symp. on High Performance Distributed Computing (HPDC)*, 2007.

[21] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *IEEE Computer*, 38(5):39–47, 2005.

[22] R. K. Sahoo, A. J. Oliner, I. Rish, and et al. Critical event prediction for proactive management in large-scale computer clusters. In *Proc. of ACM Intl. Conf. on Knowledge Discovery and Data Dining (SIGKDD)*, 2003.

[23] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proc. of Conf. on Dependable Systems and Networks (DSN)*, 2004.

[24] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proc. of Intl. Conf. on Dependable Systems and Networks (DSN)*, 2004.

[25] F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Proc. of IEEE Parallel and Distributed Processing Symp. (IPDPS)*, 2006.

[26] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. of USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.

[27] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance-computing systems. In *Proc. of Intl. Conf. on Dependable Systems and Networks (DSN)*, 2006.

[28] VMware Corporation. White paper:VMware ESX server performance and resource management for CPU-intensive workloads. Available at: http://www.vmware.com/pdf/ESX2_CPU_Performance.pdf.

[29] M. Wu, X.-H. Sun, and H. Jin. Performance under failure of high-end computing. In *Proc. of Intl. Conf. for High Performance Computing, Networking, Storage, and Analysis (SC'07)*, 2007.

[30] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo. Performance implications of failures in large-scale cluster scheduling. In *Proc. of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.