Failure-aware resource management for high-availability computing clusters with distributed virtual machines

Song Fu
Department of Computer Science, New Mexico Institute of Mining and Technology, Socorro NM, USA

ABSTRACT

In large-scale networked computing systems, component failures become norms instead of exceptions. Failure-aware resource management is crucial for enhancing system availability and achieving high performance. In this paper, we study how to efficiently utilize system resources for high-availability computing with the support of virtual machine (VM) technology. We design a reconfigurable distributed virtual machine (RDVM) infrastructure for networked computing systems. We propose failure-aware node selection strategies for the construction and reconfiguration of RDVMs. We leverage the proactive failure management techniques in calculating nodes’ reliability states. We consider both the performance and reliability status of compute nodes in making selection decisions. We define a capacity–reliability metric to combine the effects of both factors in node selection, and propose Best-fit algorithms with optimistic and pessimistic selection strategies to find the best qualified nodes on which to instantiate VMs to run user jobs. We have conducted experiments using failure traces from production systems and the NAS Parallel Benchmark programs on a real-world cluster system. The results show the enhancement of system productivity by using the proposed strategies with practically achievable accuracy of failure prediction. With the Best-fit strategies, the job completion rate is increased by 17.6% compared with that achieved in the current LANL HPC cluster. The task completion rate reaches 91.7% with 83.6% utilization of relatively unreliable nodes.

1. Introduction

Networked computing systems continue to grow in scale and in the complexity of their components and interactions. In these systems, component failures become norms instead of exceptions. Results from recent work [23] show that the existing reliability of large high-performance clusters is constrained by a mean time between failures (MTBF) in the range of 6.5–40 h, depending on the maturity of the installation. Another recent study conducted by Los Alamos National Laboratory (LANL) estimates that the MTBF would be 1.25 h on a petascale system, extrapolating from current system performance [34]. Failure occurrences as well as its impact on system performance and operation costs are becoming an increasingly important concern to system designers and administrators [42,27,53]. The success of petascale computing will depend on the ability to provide dependability at scale.

Failure-aware resource management is a crucial technique for understanding emergent, system-wide phenomena and self-managing resource burdens. Based on the analysis of failure and performance data in a system, a resource manager can help determine possible occurrences of fatal events in the near future and help develop more effective resource management solutions for failure resilience and improving system availability.

Current techniques to tolerate faults focus on reactive schemes where fault recovery commonly relies on checkpointing/restart mechanisms [11]. However, checkpointing a job in a large-scale system could incur significant overhead. The LANL study [34] estimated the checkpointing overhead to be an additional 151 h for a 100 h job (without failure) in petascale systems based on current techniques. As a result, frequent periodic checkpointing often proves counter-effective.

As the scale and complexity of networked computing systems continue to grow, research on system dependability has recently shifted onto failure prediction and proactive autonomic management technologies [31,15,27,38,16,49,32,21,45]. It is desired to construct an adaptive computing infrastructure, which adjusts its configuration dynamically at runtime according to components’ availability.

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 networked computing systems [20,24,13]. A most important feature provided by modern VM techniques is their ability of reconfiguration through VM migration [8,41]. This 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 [37,8].

Fig. 1 shows the execution time of the NAS Parallel Benchmarks [1] on 16 compute nodes. The benchmarks contain a set of programs used to evaluate the performance of parallel computer systems. The performance degradation caused by one node failure in each experiment is mitigated by migrating VMs from failing nodes to other idle nodes. The duration of stop and copy migrations ranged between 12 and 14 s with a 700 MB guest VM. However, idle nodes may not always be available as system utilization reaches 80% or more and the number of jobs is far more than that of compute nodes in a system [2]. If the affected VMs are migrated to failure-prone or busy nodes, they will suffer from server performance compromise, as shown in the figure.

In this paper, we study the proactive resource management in the face of component failures in VM-based networked computing systems. We design a reconfigurable distributed virtual machine (RDVM) framework for high system availability and performance. We propose failure-aware resource selection strategies for the construction and reconfiguration of RDVMs. We leverage failure prediction techniques for resource management in a proactive manner. We consider both the performance status and reliability levels of compute nodes in making selection decisions. We define capacity–reliability metrics to combine the effects of both factors in node selection, and propose Best-fit algorithms with optimistic and pessimistic selection strategies to find the best qualified node on which to instantiate a virtual machine for parallel jobs. The algorithms also consider the impact of virtual machine migrations in the case of node failures on system performance, and try to mitigate the effect of migration overhead by reducing the migration frequency. We evaluate the performance of our proposed algorithms by using failure and workload traces from production systems. Experimental results show that the proposed strategy enhances the system productivity significantly. By using the Best-fit strategies, the job completion rate is increased by 17.6% compared with that achieved in the current LANL HPC Cluster. The task completion rate reaches 91.7% with 83.6% utilization of relatively unreliable nodes. In our experiment running the NAS Parallel Benchmark programs on a 16-node cluster with the Pessimistic Best-fit strategy, the average execution time of long-running jobs is increased by less than 12% compared with that without failure, which indicates the practicality of our approach.

The rest of this paper is organized as follows. Section 2 presents the system model of reconfigurable distributed virtual machines and failure-aware configuration. Section 3 describes the proposed failure-aware distributed virtual machine construction and reconfiguration strategies. The performance of these strategies is evaluated in Section 4. Section 5 presents the related work and Section 6 summarizes this paper.

2. Reconfigurable distributed virtual machine infrastructure and failure-aware configuration

Virtual machines provide an additional layer of abstraction for networked computing. Applications running in virtual machines achieve performance comparable to that in host OS directly [36,30,5]. In addition, performance isolation and fast VM instantiation make resource management easier and more efficient. The service availability can be enhanced significantly by means of convenient checkpointing and migration mechanisms. In order to efficiently utilize resources on different compute nodes, multiple virtual machine instances are created for a parallel job. They run cooperatively to complete the job.

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

2.1. Reconfigurable distributed virtual machines

A reconfigurable distributed virtual machine (RDVM) as illustrated in Fig. 2 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 [37].

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 migration in RDVM reconfiguration.

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 the 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 RDVM reconfiguration decisions. It also receives job execution requests from users, and selects compute nodes to construct an RDVM. In large-scale 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. The VM Coordinator, FailurePredictor, and VM LoadManager can reside on a single master node or be distributed on multiple master nodes in a coalition system, such as a compute grid. Within each cluster, a master node hosts a VM Coordinator, a FailurePredictor, and a VM LoadManager for failure and resource management of the cluster. They aggregate the scheduling, failure reports, and workload distribution information in the cluster and send it to the system master node(s). The FailurePredictors and VM LoadManagers submit reports to the VM Coordinator which makes job scheduling decisions. They form a hierarchical structure to manage resource and VMs in the 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 multi-programming 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. If the request of a user job cannot be satisfied immediately, it needs to wait until compute nodes with qualified resources become available. Each job has a completion deadline. The VMs for a job will be collected when the deadline is reached. 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 reconfigurable 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 RDVM initiation and to host migrated VMs in VM migrations.

2.2. Failure-aware RDVM configuration

After a user job is submitted with its computation requirement to a coalition system, the Coordinator evaluates the qualifications of active compute nodes, which may require collaboration of Coordinators at multiple layers of the system. 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 be sustained.

To derive selection strategies in RDVM construction, we formulate the selection problem as follows. Let \( a_1, a_2, \ldots, a_k \) denote the set of jobs that the Coordinator requests for execution in a system with \( N \) compute nodes. Let

- \( w_i(t) \): remaining workload of task \( a_i \) at time \( t \)
- \( d_i \): deadline of completing \( a_i \)
- \( c_j(t) \): available capacity of node \( j \) at time \( t \)
- \( \delta_j(t) \): time to next failure forecast by the FailurePredictor for node \( j \) at time \( t \).

If we concern ourselves only with the finish time of the entire job, we can set each \( d_i \) 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 the reliability of a node, the FailurePredictor forecasts when the next failure will occur in that node [15]. Let \( (p_1, p_2, \ldots, p_k) \) be the probabilities that nodes in a selected group \( G \) will fail within an interval \( \delta \). The reliability status, \( r_G \), of the group \( G \) in the interval \( \delta \) is expressed as

\[
r_G = 1 - p_G = \prod_{i=1}^{k} (1 - p_i).
\] (2.1)

Next, we take both the performance and reliability status of compute nodes into account in making RDVM construction decisions. 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 that \( (n_1, n_2, \ldots, n_k) \) is the group of nodes selected for job \( A \). The FailurePredictor forecasts when the next failures will occur on them, i.e. the time-between-failures \((\delta_1(t_0), \delta_2(t_0), \ldots, \delta_k(t_0))\).

To select a compute node, say \( n_i \), to execute a job task \( a_i \), the Coordinator considers the reliability status of node \( n_i \). The deadline of task \( a_i \) can be either before the predicted occurrence time of \( n_i \)’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 \( n_i \) if this node has enough capacity, that is, the available capacity of node \( n_i \) is ample enough to complete the computation of task \( a_i \) before this node comes to a failure. This is expressed as

\[
\int_{t_0}^{d_i} c_j(t) \, dt \geq w_i(t_0),
\] (2.2)

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

For the second case, a failure is predicted to occur on node \( n_i \) before the deadline of task \( a_i \). That is, \( d_i > t_0 + \delta_j(t_0) \). It is observed

\[\text{...}\]

1 Originally, \( \delta_j \) refers to the predicted interval between the time \( (t_i) \) when the last failure occurs and that \( (t_j) \) of the next one on node \( n_j \). Since we know the difference between \( i_0 \) and \( i_0 \), we refer \( \delta_j(t_0) \) to the interval between \( t_j \) and \( t_0 \).
that the average time between failure of a compute node is larger than four months in production systems [42,39]. In addition, the average execution time of user jobs in production systems is much less than this large time–between-failure. For example, among the 74,463 finished jobs between 20 November 2003 and 23 October 2005 in an LANL HPC cluster, 90% of them completed execution within 9 h and less than 1% of them lasted for more than a week [3]. 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 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, as shown in Fig. 3. Within the 615 tasks of those jobs, only 4 (0.6%) of them suffered from failures twice [3]. 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 \( a \) along with its underlying supporting systems will be moved to another compute node and \( a \) will resume execution there. Assume that node \( n_k \) is selected to accommodate this migrated VM for task \( a \). Then, nodes \( n_i \) and \( n_k \) can be selected to run \( a \), if they have enough capacity to complete the task’s workload before the deadline; i.e.

\[
\int_{t_0}^{t_0 + \delta_i} c_j(t) \, dt + \int_{t_0 + \delta_i}^{t_0 + \delta_j + \tau} c_j(t) \, dt \geq w_j(t_0),
\]

(2.3)

where \( \tau \) is the overhead of VM migration caused by a failure which is predicted to occur on node \( n_i \). Its value depends on the size of the working set generated by a job task [8] and the size of the supporting systems including the underlying guest OS. By stop and copy migrations, the migration overhead is 12–14 s with a 700 MB guest OS, as measured in our experiments. Compared with 8.28 h of average job execution time [3], this migration overhead is negligible. To provide a practical estimation of the migration overhead \( \tau \), we use the measurement of overhead in a previous VS migration to set the value of \( \tau \) in a future migration for the same type of guest OS.

In (2.2) and (2.3), the capacity of a node, \( c(t) \), 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 the problem tractable, we use the measurement of available capacity of a node at the time of making the selection decision to approximate its capacity in the makespan of a job task. As a result, the preceding two equations can be rewritten as

\[
c_j(t_0) \cdot (d_i - t_0 - \delta_i) \geq w_j(t_0), \quad \text{if } d_i \leq t_0 + \delta_j(t_0)
\]

(2.4)

and

\[
c_j(t_0) \cdot \delta_j + c_j(t_0 + \delta_j + \tau) \cdot (d_i - t_0 - \delta_j - \tau) \geq w_j(t_0),
\]

if \( d_i > t_0 + \delta_j(t_0) \).

(2.5)

We introduce a Bernoulli random variable \( o_{ij} \) as

\[
o_{ij} = \begin{cases} 1, & \text{if } d_i > t_0 + \delta_j; \\ 0, & \text{otherwise.} \end{cases}
\]

This indicates whether a failure is predicted to occur on node \( n_j \) during the execution of task \( a_i \), or not. With \( o_{ij} \), (2.4) and (2.5) are combined into

\[
(1 - o_{ij}) \cdot c_j(t_0) \cdot (d_i - t_0) + o_{ij} c_j(t_0) \cdot \delta_j + c_j(t_0 + \delta_j + \tau) \times \left( d_i - t_0 - \delta_j - \tau \right) \geq w_j(t_0).
\]

(2.6)

The value of \( o_{ij} \) is determined by comparing the predicted occurrence time of the next failure in node \( n_j \) and the deadline of task \( a_i \).

In a multi-programming environment, multiple virtual machines may reside on a single compute node to multiplex its resources. These VMs run different job tasks. The available capacity for a VM \( i \), \( c_i(t) \), is a portion of the available capacity of a node \( j \). The quantity of the proportion depends on the resource scheduling strategies adopted by the node and the amount of resources requested by the VM. As jobs run and finish dynamically and VMs migrate onto/away from a compute node, the available resources (capacity) of the node change dynamically at runtime and the proportion of resources allocated to each VM on the node varies as well, although the compute nodes may have a homogeneous configuration.

3. Failure-aware RDVM construction and reconfiguration 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 failure, 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.

There is a trade-off between the resource utilization and the completion rate. By forming groups with the most powerful and reliable nodes, we generally increase the likelihood that a job will complete execution by its deadline, but decrease the utilization of less powerful and reliable nodes, which may in turn decrease the throughput of the entire system. Conversely, assigning less powerful and reliable nodes to tasks can improve the resource utilization, but at the price of increasing the possibility that jobs may experience multiple migrations and fail behind their deadlines. Next, we present four strategies which achieve different extents of trade-off between the two objectives.

In the following discussion, we leverage failure prediction techniques in the construction and reconfiguration of RDVMs in networked computing 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.
3.1. Random-select

The Random-select strategy randomly assigns nodes to a group \( G \) of size \( k \) to instantiate virtual machines and run the \( k \) tasks of job \( A \). Each node in group \( G \) 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 the 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 a fixed number of nodes, irrespective of \( \delta(t) \).

As a result, the reliability status \( r_c \) of group \( G \) in (2.1) is a random variable that takes any value between 0 and 1, depending on which nodes are selected. Since the nodes are selected at random, we cannot ensure that job \( A \) will be completed within its desired deadline. In (2.6), \( o_j \) can be either 0 or 1 and VM migrations may occur within the makespan of job \( A \).

3.2. 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 a reliability status which ensures that there is no failure predicted to occur before the deadline of task \( a_i \), i.e. \( S_1 = \{ n_i \mid d_i < t_0 + \delta_i(t_0) \} \). The rest of the 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 with high probability during the duration of tasks \( \{a_i\} \) in a greedy manner. According to \( (2.1) \), the reliability status of group \( G, r_c \), is predicted to approach 1. The most powerful node is selected for each task, which ensures that job \( A \) will be completed as soon as the system allows. In (2.6), \( o_j \) = 0 and the capacity of \( n_j \) is the largest one that minimizes the finish time of task \( a_i \) in

\[
C_j(t_0) \cdot (t_j - t_0) = w_j(t_0),
\]

where \( t_j \) is the finish time of task \( a_i \).

3.3. Optimistic Best-fit

The Best-fit algorithm attempts to form a group \( G \) of nodes for tasks of job \( A \), so that the utilization of system resources is maximized in a failure-aware manner. Two node selection strategies are described in this section, based on the different levels of reliability status of compute nodes.

We first focus on those compute nodes which have a "good" reliability status. The optimistic selection strategy 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 \( S = \{ n_i \mid t_0 + \delta_i(t_0) \geq d_i \} \). 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 (2.4). The minimum capacity satisfying the above inequality is

\[
C_j^{*} = \frac{w_i(t_0)}{d_i - t_0}.
\]

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 \( C_j^{*} \).

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 h 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 task \( a_2 \) not finding any node that will not fail within the next 40 h 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 the capacity–reliability metric as

\[
c_r = \alpha \frac{w_i(t_0)}{d_i - t_0} + (1 - \alpha) \frac{\delta_i(t_0) - d_i(t_0) + t_0}{d_i(t_0) - t_0}.
\]

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

In the Optimistic Best-fit algorithm, we first construct the set of quantified nodes \( S = \{ n_i \mid C_j(t_0) \geq w_i(t_0)/(d_i - t_0) \} \) AND \( \delta_i(t_0) + t_0 \geq d_i \) 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 \( c_r \), according to (3.1). Then, a virtual machine will be instantiated on the selected node to run task \( a_i \).

3.4. Pessimistic Best-fit

The preceding optimistic selection strategy considers only those compute nodes that are reliable enough to avoid failures in the duration of tasks. However, there are also nodes which are not very reliable but have ample available capacity to run the tasks. The pessimistic selection strategy is derived based on the observation that each task suffers from at most one nodal failure during its execution (see Section 2.2). Therefore, a VM migration is performed to continue task execution on another available node to tolerate a failure. The goal is to utilize the available capacity of those unreliable nodes in RDVM construction and reconfiguration.

For a task \( a_i \) of job \( A \), we choose those less reliable nodes to form a set \( S = \{ n_i \mid \delta_i(t_0) + t_0 < d_i \} \). For a node \( n_i \) in set \( S \), if its available capacity satisfies the inequality in (2.5), i.e. \( C_j(t_0) \cdot (d_i - t_0 - \tau) \geq w_i(t_0) \), then it is also qualified to instantiate a VM and run task \( a_i \). In (2.5), we need to select not only the node \( n_i \) on which a VM will be instantiated, but also another node \( n_k \) to which a VM migration will be made. One way to find the second node, \( n_k \), is to use the available capacity status of nodes at the current time as an estimate of their capacity status at the time when a migration occurs. Thus, we can select node \( n_k \) as the migration destination and reserve its capacity for future use. However, this results in resource waste on \( n_k \) and it is even worse if node \( n_k \) is not available any more when the migration is to occur.

To tackle this problem, we adopt a renewal process approach. To select the first node, \( n_j \), we use the available capacity status at time \( t_0 \) to make a decision. When a VM migration is scheduled at time \( t_1 \), we select the new destination node, \( n_k \), based on the available capacity status at \( t_1 \), treating the selection process as a renewal one. To find \( n_j \) for a task \( a_i \), we use the average available capacity level among the compute nodes that will not fail before \( d_i \) to estimate the reliability status of \( n_k \). That is, for set \( S_k = \{ n_i \mid \delta_i(t_0) + t_0 \geq d_i \} \),
4. Performance evaluation

We implemented the Random-select, First-fit, Optimistic Best-fit, and Pessimistic Best-fit strategies to evaluate their performance in improving the productivity of networked computing systems. 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, and the utilization of relatively unreliable nodes. 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.

4.1. Workload and failure models

In our experiments, we used job logs and failure traces from the Los Alamos National Laboratory (LANL) HPC clusters system [3]. The data span 22 high-performance computing systems that have been in production use at LANL between 1996 and 2005. Most of these systems are large clusters of either non-uniform memory access (NUMA) nodes, or 2-way and 4-way symmetric multi-processing (SMP) nodes. In total, the system includes 4750 nodes with 24,101 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. Fig. 4 shows the distribution of the 307 failures on nodes that occurred between 1 September 2004 and 31 August 2005. They were failures caused by hardware or system software faults that made compute nodes unavailable. 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. The numbers of nodes requested by those jobs and their required processing time are depicted in Fig. 5.

4.2. 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 [3]. We then evaluated the performance of our proposed failure-aware RDVM construction and reconfiguration strategies. We adopted the failure predictor that we proposed in [15] 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 prediction error is defined as err = (ActualTime − PredictedTime)/ActualTime * 100%. In this work, we exploit the reliability estimation generated by failure prediction mechanisms in RDVM construction and reconfiguration. We consider the cases in which the predicted occurrence time of a future failure precedes the actual occurrence time. The average prediction accuracy achieved by the predictor is 76.5%, and the prediction error due to false negatives is 3.7%. To evaluate the sensitivity of those algorithms to the accuracy of failure predictions, we also generated synthetic traces of failure predictions with different accuracy. We used those predicted results to determine the time-between-failure δi for each node ni.

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, and the number of tasks of each job from the job traces. We use those workload data, failure predictions, and node capacity information in our experiments. The VM migration overhead is 12 s 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 system that is perceived by users. Each job

\[
\text{error} = \left( \frac{\text{ActualTime} - \text{PredictedTime}}{\text{ActualTime}} \right) \times 100\%.
\]

In the job traces of LANL HPC clusters, the execution deadline was specified for each job. Thus, we set the deadline of a task equal to that of its associated job.
spawns multiple tasks on an RDVM. A job is considered completed only if all of its tasks finish execution before its deadline. Fig. 6 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 Optimistic Best-fit ($\alpha = 0.5$) and Pessimistic Best-fit, more user jobs successfully complete their execution compared with the other two strategies. Both Best-fit strategies are relatively sensitive to the accuracy of failure predictions. They do not perform well with bad predictions. However, as the prediction error is less than 40%, they achieve better job completion rate. By using failure predictions generated by our predictor [15] with a 76.5% prediction accuracy, we achieve 80.2% and 87.3% job completion rates by the Optimistic Best-fit and Pessimistic Best-fit strategies, respectively. Compared with the 69.7% job completion rate achieved in the current LANL HPC cluster, the performance improvement is significant.

In addition to the job completion rate, the task completion rate is another important metric in evaluating the productivity of a 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. Fig. 7 presents the task completion rates achieved by using the four RDVM construction and reconfiguration strategies. They follow a similar trend of changes to that of the job completion rate as the prediction accuracy increases. The Pessimistic Best-fit and Optimistic Best-fit strategies perform more efficiently in utilizing system resources and mitigating the impact of failures on running tasks. They outperform the Random-select and First-fit strategies by more than (32.7%, 21.3%) and (25.1, 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 91.7% and 82.5% by using the Pessimistic Best-fit and Optimistic Best-fit strategies.

The reliability status of compute nodes changes with time in a system. The four strategies adopt different policies in node selection. An important system metric is the utilization of relatively unreliable nodes, whose reliability status is below the average level in the system. We measured the utilization of those nodes when using different selection strategies. Fig. 8 shows the results. From the figure, we can see that, when the reliability estimates are not accurate, the four strategies behave similarly. As the accuracy of failure prediction increases, the Best-fit strategy tends to dispatch parallel tasks to powerful and reliable compute nodes. As a result, the unreliable nodes are underutilized; only 60.4% of utilization with failure predictions of 76.5% in accuracy. On the other hand, the Pessimistic Best-fit strategy assumes that a node failure might be experienced by a task in its lifetime and attempts to choose those less reliable nodes to run tasks while trying to preserve their finish time before deadlines. It achieves 83.6% utilization of those unreliable nodes with that 76.5%-accuracy failure prediction trace. In comparison, the Optimistic Best-fit strategy obtains a utilization of 71.3%.

4.3. Benchmark performance

We also conducted our performance evaluation on a 16-node cluster system. Each node in the cluster is equipped with two AMD Opteron 2.4 GHz processors (each dual core) and 4 GB of memory interconnected by a 1 Gbps Ethernet switch. The Xen 3.0.3 virtual machine monitor is installed on all the nodes. The nodes run a para-virtualized RedHat AS4 with Linux 2.6.12 kernel as a privileged virtual machine (domain0) on top of the Xen hypervisor. All user domains are configured to run the same version of OS as that of the domain0. They run with a single virtual processor, 512 MB memory and 700 MB disk image.

We evaluated the performance of our RDVM construction and reconfiguration strategies with the NAS Parallel Benchmark programs [1] in this cluster. We used the failure event traces of node 1 to 16 in the LANL HPC Cluster 20 between September 2004 and August 2005. Then, we scaled the time from a one-year period to 10 days, and used the resulting failure event trace in failure injection to our 16-node cluster to simulate the occurrences of failures in production systems. The 16 nodes experienced three failures in 10 days. As to the job arrival sequence, we used the job trace from the LANL Cluster 20 in a similar way to generate a scaled job trace for our experiments. We ran the five NAS benchmark programs (BT, EP, CG, LU and SP) alternately on the 16 nodes. We also scaled the failure prediction trace obtained by our failure
predictor [15] for the LANL Cluster 20. The prediction accuracy is 76.5%. Another pool of four nodes with the same configuration was available to accommodate migrated VMs at runtime. A failed node was added to the pool after a fixed repair time.

We measured the overhead of VM migrations by using the stop and copy migration and live migration techniques. It took 12–14 s to migrate a VM with the stop and copy migration. In contrast, 16–25 s were spent in a live migration on average. Although the service downtime is short in the latter case, its overall migration time is longer, compared with the former approach. Due to its small overhead and the imperfection of failure prediction, stop and copy migrations are more preferable in our failure-aware resource management schemes. Fig. 9 shows the performance of the five benchmark programs. We measured the average execution time in three cases: without failure; by restarting after a failure is predicted; by using the Pessimistic Best-fit strategy with stop and copy migrations. From the figure, we can see that the total execution time increases dramatically by using the restart strategy, which is 1.4–1.8 times that without failure. With the Pessimistic Best-fit strategy, we migrate a VM to a “best” node selected to accommodate the VM and resume task execution. As a result, the total execution time is only 8%–25% more than that without failure. For long-running jobs, such as BT, LU and SP, their execution time increases by less than 12%. Combined with the significant improvement of system productivity, in terms of the job and task completion rates and resource utilization presented in Section 4.2, the Best-fit strategies are efficient in managing system resources in a failure-aware manner. The Optimistic Best-fit strategy is suitable for systems with reliable nodes, while the Pessimistic Best-fit strategy is good for relatively unreliable systems.

5. Related works

Recently, interest in using virtual machines (VMs) as the abstraction for distributed and parallel computing in general has been growing [20,24,13]. Virtual machine monitors, such as Xen [5] and VMware [46], 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 [37], 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 [8] and OS migration in VMware [47]. The performance of VM migrations has been extensively evaluated in [41,8]. 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 in failure management research, and failure analysis [42,28,39,52] reveals failure characteristics in high-performance computing systems. Zhang et al. evaluated the performance implications of failures in large-scale clusters [53]. Sahoo et al. [38] inspected the eventset within a fixed time window before a target event for repeated patterns to predict the failure event of all types. Liang et al. [27] 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 [31] assumed the 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 [15] exploited the temporal and spatial correlations among failure events to predict the occurrence time of next failures in HPC systems. Gokhale and Trivedi [18] forecast the software reliability by using Markov chains to represent software architecture. Most of these works focused on improving the prediction accuracy, and few of them considered how to leverage their prediction results for resource management in practice. Salfner et al. [40] mentioned that proactive failure management has the potential to improve system availability by up to an order of magnitude, based on their analysis of prediction accuracy. The FT-Pro project [29] 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 the BlueGene/L system, and simulation studies showed that the use of these new algorithms could significantly improve the system performance [33]. In [17], we proposed a proactive mechanism which utilizes failure prediction techniques in resource management without considering the effects of VM migrations. Sun et al. [51] modeled the effect of failures on the application completion time and applied a checkpointing technique for fault tolerance. In addition to parallel systems, reliability-aware scheduling in heterogeneous distributed systems has also been studied [10,22,35].

Correia et al. [6] proposed a dependable tuple space for distributed systems to enhance Byzantine fault tolerance. Tuple spaces along with checkpointing and replication mechanisms have been applied to grid scheduling in [12]. The performance of distributed checkpointing protocols has been evaluated by Agbaria and Friedman [4]. They consider the overhead ratio which also takes the recovery time into account in performance evaluation. Défago [9] proposed an accrual failure detector which outputs a real-valued suspicion level based on the system runtime status. Fetzer and Wappler [50] extended the vital code technique in detecting hardware failures. In [44], Chen et al. explored the downtime statistics in making an availability estimation for replica replacement in distributed storage systems. They also investigated the quality of failure detectors and then proposed an optimal predictor based on message behaviors [7]. Jiménez et al. [25] proved that a failure detector class with weak completeness can be implemented only if every process knows the identity of the rest of processes in redundant computing. The influence of processing speed and message delay in redundant computing has been investigated by Raynal et al. [19]. In [26], Junqueira and Marzullo proposed a framework which considers correlated failures for the design of distributed algorithms, and they used replication predicates to express process replication constraints. In [43,48], Suri et al. investigated the heterogeneity of fault occurrences as a combination of faults with different types and proposed a framework to customize fault models.

However, there are few formal approaches to analyzing the impact of failure events on resource management decisions, and to develop effective resource allocation strategies and algorithms to construct and reconfigure virtual machines for networked computing by leveraging failure prediction results.
6. Conclusions

Large-scale networked computing systems are susceptible to software and hardware failures and administrators’ mistakes, which significantly affect the system performance and management. Virtual machines provide a flexible and potent tool for efficient resource management, failure resilience, and online system maintenance. We propose failure-aware distributed virtual machine construction and reconfiguration strategies for networked computing systems. In addition to the performance states of candidate nodes, we also consider their reliability status in selecting nodes for RDVMs. We have evaluated the performance of the proposed strategies by using failure traces from production systems and through experiments in real systems. Experimental results show the enhancement of system productivity by using the Best-fit strategies 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 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 productivity of systems.

Acknowledgments

This research was supported in part by US NSF Grant CNS-0915396 and LANL Grant IAS-1103. The author thanks Dr. Kai Hwang and anonymous reviewers for their constructive comments and suggestions. A preliminary version of this paper was presented in the Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid’09) [14].

References


Song Fu is currently an Assistant Professor in the Department of Computer Science at New Mexico Institute of Mining and Technology. He is the director of the Dependable Computing Systems Lab. He received his Ph.D. degree in Computer Engineering from Wayne State University in 2008, his M.S. degree in Computer Science from Nanjing University, China, in 2002, and his B.S. degree in Computer Science from Nanjing University of Aeronautics and Astronautics, China, in 1999. His research interests lie in distributed, parallel and cloud systems, particularly in system reliability and security, dependable computing, self-managing and reconfigurable systems, virtualization, and power management. His research projects have been sponsored by the US National Science Foundation and Los Alamos National Laboratory. He is a member of IEEE and a member of ACM.


