Decision Support for Virtual Machine Re-Provisioning in Production Environments

Kyrre Begnum and Matthew Disney – Oslo University College, Norway
Æleen Frisch – Exponential Consulting
Ingard Mevåg – Oslo University College

ABSTRACT

Management of virtual machines (VMs) on a large scale remains a significant challenge today. We lack general and vendor-independent quantitative criteria/metrics by which to describe the state of infrastructures for virtualization. This data is essential to both expressing administrative policy goals and to measure ongoing compliance with them in production settings.

In this work, we consider VM management in production environments. We investigate VM applicability to real-world scientific computing problems by comparing application performance in a controlled environment of physical servers with the Manage Large Networks (MLN) tool's implementation of the same scenario on virtual machines. Based on our observations, we propose three new metrics by which to describe and analyze the infrastructure in order to incorporate virtual machine management closer into policy. The metrics have been implemented into the latest release of our MLN tool for virtual machine management.

Introduction

Most approaches to virtualization devote the bulk of the efforts on their features related to deployment and fail-over mechanisms. Indeed, much of the work in this area focuses on these areas exclusively. However, after running a virtual infrastructure in production, one discovers how deployment decisions and virtual machine behavior affect the overall performance in ways which are not completely described nor obviously transparent. One key challenge is avoiding resource conflicts among the virtual machines. This task depends on many local factors, such as the number and behavior of the virtual machines, as well as the number and capacity of physical servers and other infrastructure resources like common storage. We use the term ''virtual(ized) infrastructure'' as a general scenario where more than one physical servers are used to host a range of virtual machines with varying life-span and purposes. We do not intend to refer to a particular virtual machine managment framework or product.

The system administrator needs methods to analyze and describe the site's virtualization infrastructure in a technology independent way. Such analysis would assist in the following tasks:

- Determine the level of redundancy in server capacity for downtime planning.
- Review the level of resource conflicts between virtual machines in order to identify and remove bottlenecks.
- Find the optimal server location for new virtual machines in the infrastructure.
- Identify which virtual machines are apt to demand resources at the same time in order to separate them from each other.

Possessing this and related data will also enable the system administrator to express and implement explicit, quantitative policy rules for the site, enabling her to address a variety of concerns: How should one describe the desired level of redundancy in server capacity and measure its compliance? Could a level of resource conflicts function as an indication of how well the virtual machines are deployed across the site? This is not only valuable for system administrators today, but necessary steps towards autonomic capabilities of future management tools, since self-optimizing system behavior techniques require quantifiable metrics in order to measure success.

MLN (Manage Large Networks) [1] is a virtual machine management tool developed at the University College of Oslo. It supports both Xen and User-Mode Linux. After using MLN in various scenarios and addressing the need to efficiently deploy large scenarios of virtual machines [2, 3, 4], we have arrived at the point where long-time management reveals new challenges, problems which are still unexplored by the community. This article addresses these challenges by proposing three methods by which to analyze a virtualized infrastructure. The *Server redundancy level*, *Resource conflict matrix* and *Location conflict table* are technology independent measures of the state of the infrastructure.

This text is organized as follows: We first demonstrate the viability of using Xen virtual machines and MLN in a production environment. This case study demonstrated to us the challenges of long-time management of virtual machines. Next, our approaches for analysis are presented and discussed in detail. We then discuss the implementation of these in MLN and

discuss our findings to date. Finally, we outline future work and review related work.

VM Viability in Production Scientific Computing Environments

In recent years, there have been many calls for the increased use of virtualization technologies for high performance computing (HPC) and especially scientific modeling and simulations (see, e.g., [5, 6]). The many advantages of virtualization and virtual machines (VMs) usually apply to such specialized computing environments as they do to general purpose systems (e.g., the ability to control resource usage, security barriers, performance and reliability improvements due to isolation of distinct processes, and so on).

The computing environment and typical job characteristics of scientific computing environments, whether located in academic institutions, corporate research and development or government laboratories, share many common traits which distinguish them from more general computing environments:

- Rather than a mix of many jobs of various types and needs, scientific computing typically consists of a few computations of significant duration. The most intensive of scientific computing applications have essentially unbounded need for CPU cycles, physical memory and memory bandwidth, and, in some cases, disk and/or network I/O capacity.
- Research efforts tend to rely on a few software packages related to the field under investigation. Most production software is commercial for which source code is not available (and the expertise required to modify it is not present at most sites).
- Research groups prefer to have their own computing resources, with typically little or no system administrative support from any centralized IT organization.

Such environments face many challenges which virtualization technology, in conjunction with a tool like MLN, can address:

- Deploying idle computing resources, where and when they are available. This can include applying general purpose computers to simulation problems after hours. VMs allow the host operating system environment to remain unaffected by allowing scientific computing usage of the hardware.
- Since scientific computations can execute for hours, days or even weeks, being able to start, pause, restart and migrate such jobs is very beneficial. VMs and MLN provide this capability to any application which employs their resources, often adding this valuable feature for the first time. For example, Gaussian 03, the production code we employed in performance tests, has only limited job restarting capabilities. MLN allows jobs to be paused at any point. In the past, such capabilities were present on in special purpose checkpointing libraries [7, 8].
- Multiple operating system environments – even legacy ones – can be maintained and invoked on the same hardware. This is important in this arena in that distinct software packages typically have limited, and often contradictory, operating system version support (often lagging well behind the current releases). MLN allows distinct OS environments to be set up and cloned easily, ready to be reused as often and for as long as needed.

Despite these very real benefits, however, performance is still the most important consideration in scientific computing, so virtualization will need to come with few associated resource costs in order to be adopted. Previous work has focused on the performance achieved on standard benchmarks [9, 10, 11, 12]. While this data is interesting, and in general encouraging with respect to VM use for HPC, the limitations of benchmarks in modeling actual scientific computing are well known. First, such benchmarks tend to focus on single performance metrics in isolation: CPU speed, memory bandwidth, disk I/O, network message passing, and so on. Secondly, the algorithms employed in the computational portions are among the simplest of those used in real applications (e.g., Linpack). In addition, the results of the more complex, higher level benchmarks (e.g., the SPEC suite, the GCM program used in [10], and the like), the results are reduced to a single metric, megaflops achieved, which has been repeatedly shown to bear only a vaguely proportionate relationship to actual production performance. Finally, and often most importantly, the problem size is far too small to be useful or representative of actual computations and computing requirements.

Accordingly, we chose to conduct some tests with actual production code on modest sized but realistic problems. We used the Gaussian 03 computational chemistry package. Gaussian 03 performs electronic structure calculations, modeling the properties of chemical compounds and reactions. It is widely used by academic and industrial chemists, chemical engineers, biochemists, physicists, and materials scientists throughout the world, addressing one of the key Grand Challenge level problems. Computationally, it is a very demanding and rigorous application whose achieved performance depends on the combination of CPU performance, memory bandwidth and, for some simulations, disk I/O transfer rates.

We ran three jobs of increasing CPU requirements in several ways: under a standard Red Hat Enterprise Linux kernel, directly under a Xen version of that operating system, and in a Xen virtual machine (details, which are probably understandable only to chemists, are given in a separate section below). We

also ran the jobs on a single node and in a parallel mode using three nodes; the parallel computations were performed under the Xen-enabled operating system and in VMs. In this way, we could test the performance effects of both aspects of virtualization: the modified Linux kernel and using a VM itself. The three nodes chosen for the parallel computations were quite different in performance characteristics and available resources. This selection was made to model the performance that might be obtained from drawing together idle systems in an ad hoc manner after hours. However, parallel performance is known to be best when using symmetric nodes; this approach thus represents a worst-case scenario in that the performance of the more powerful systems is reduced to that of the weakest node.

The following table gives the parallel speedups obtained in the two environments (comparisons are with respect to a single node of the same type):

Parallel Speedups (1 vs. 3 Nodes)		
Job	Xen	VM
1	2.1	2.1
2	2.8	2.8
3	1.2	1.3

Table 1: Parallel speedup comparison.

Job 1, the shortest job, obtains reasonable parallel speedups, and Job 2 does quite well (as the maximum speedup is 3.0); both of these jobs have few I/O requirements. This is in contrast to Job 3, chosen because of it substantial I/O requirements. Even in this case, however, parallelization provides some performance benefits. For our purposes, however, the key result in the preceding table is that using virtual machines for the computations produced identical performance to the jobs run directly on the hardware. Using a VM had no adverse affect on parallelization efficiency.

The following table explores the overhead associated with virtualization – and specifically the Xen approach – in more detail:

Virtualization Overhead		
Job	Xen over RHEL	VM over RHEL
1	3.8%	0.6%
2	3.9%	0.8%
3	3.1%	7.9%

Table 2: Virtual overhead comparison.

The columns compare the performance for single processor jobs in the three environments: directly on the hardware with a standard kernel (RHEL), directly on the hardware with a Xen-enabled kernel (Xen) and in a Xen VM. The table indicates that when disk I/O is not a factor, then there is only a quite small performance penalty associated with the Xen operating system and virtually none associated with using a VM, both compared to the vanilla Linux OS. Interestingly, the Xen kernel itself slightly less efficient than the vanilla operating system running in a VM.

When disk I/O is a significant factor, the results are somewhat different. The overhead associated with the Xen-enabled OS remains more-or-less constant, but there is addition associated with running in a VM (at least 4.8% for this job). The total overhead of about 8% is still quite acceptable, but reducing that level would be desirable. Our initial tests focused on making virtualization configuration and use as easy as possible. Thus, the VM was built on a logical volume. In the jobs running directly under the Xen-enabled OS, however, disk I/O associated with the computation was to a logical volume as well, and the LVM undoubtedly imposed some overhead.

Future work will look at minimizing I/O overhead in the VM environment while still retaining VM configuration and build simplicity. However, these results show clearly that performance in an MLN-managed virtual environment is quite acceptable for production Gaussian 03 calculations.

New Challenges and Approaches

Our infrastructure consisted of a total of 11 servers spread over two separate locations. Although the performance levels of Xen were acceptable, we encountered management issues which revealed new challenges:

- Previous work has shown that there is a substantial performance degradation if two virtual machines compete for the same resources on the same server [3]. We encountered difficulties avoiding such conflicts due to lack of overview over the entire infrastructure.
- At one point, one of the servers became unstable and crashed. This taught us a valuable lesson because we ended up with insufficient capacity to host all of our virtual machines until the problem was fixed. We believe that a way to plan ahead for these eventualities would avoid the same situation in the future.
- There was no decision support from MLN to find the optimal placement for a new virtual machine project. We had to manually inspect the other running projects to arrive at a solution.

We believe that these challenges are universal for all system administrators who have to manage sufficiently many servers and virtual machines. Let us first describe a virtualized infrastructure in a more generalized way.

The foundation of a virtualized infrastructure is its physical servers, for which we will use the term *server*, and the virtual machines (*VMs*). We make two basic assumptions:

1. The infrastructure consists of more than one physical server.

2. The infrastructure enables the system administrator to move virtual machines between the physical servers.

What is left is to find the optimal placement of the virtual machines such that the load is evenly distributed and that all the VMs experience a satisfactory level of performance. This process is highly dependent on the organizations context. The resulting *provisioning policy* will act as a local guidebook for decision making or a policy for which an expert system may surveil and tune the infrastructure.

It is essentially the system administrator's duty to re-provision virtual machines for the best performance. This manual process depends on information about the infrastructure and the individual virtual machines. In some cases, knowledge about the virtual machine can help the decision process. Is it planned to be a web-service or a shell server? Can we expect performance peaks at specific times? Over what time period will the virtual machine run? This information may be hidden from the administrator if users can create their own virtual machines independently. What remains is the static hardware description of the virtual machines resource consumption and its performance profile. We will take this perspective in this text, assuming that we have no prior knowledge about virtual machine roles. We will, however assume that the administrator is at liberty to re-provision the virtual machines to different physical servers.

Virtual Machine Resources

A virtual machine can be described in two complementing ways. The first is through its static attributes which are defined in the virtual machines design. Examples are the amount of memory, number of CPUs and the placement of its filesystem locally or on a SAN. These design decisions influence the virtual machines performance and are for the most part static variables. Once the virtual machine is running, we get a series of dynamic variables describing how the virtual machine performs over time. We get the CPU consumption, network traffic, IO operations and process interrupts, to name a few.

It is easy to plan ahead for even distribution among the static resources. Memory usually dominates these decisions – there is only so much to go around. The CPU, on the other hand, can be shared, and on a multi-CPU server, virtual machines may even run along side each other without conflict. Too many virtual machines demanding CPU time at once will throttle the performance of all. But in order to avoid this, we need to have knowledge about which virtual machines are troublemakers in order to isolate them. For small sites with a convenient number of virtual machines, intuition and random observation may be sufficient for a satisfactory end-result. But what when the amount of servers and virtual machines exceeds what is practical for manual analysis? On larger sites, the dynamic resources need to be observed as time-series variables and stored for analysis.

- **Static resources** belong to the *server* and are allocated for each VM at both boot and build time. Examples of static resources are:
  - Disk size
  - Filesystem placement
  - Memory
  - Virtual CPUs
- **Dynamic resources** are consumed by the VM at run-time. They are subject to rapid change. Examples of dynamic resources are:
  - CPU seconds
  - Network traffic
  - IO operations

The Server Redundancy Level

Uptime and service availability are strong arguments for virtualization. But this depends strongly on the infrastructures stability and capability to re-provision virtual machines. Several virtual machines on the same server means increased pressure on the server not to fail. If it would show signs of failure, like messages about a failing disk, the virtual machines must be re-provisioned as quickly as possible in order to take down the sever for maintenance. A server that crashes with virtual machines still on is a disaster which is simply described as the all-the-eggs-in-one-basket effect. There is therefore much effort required by the system administrator in order to know when and which server can be taken down.

Live migration is one of the most attractive feature when it comes to re-provisioning a virtual machine. It enables a virtual machine to move to a different server without loss in downtime. Other methods of migration exist too, like cold migration, where the virtual machine is shut down before it is moved. The latter approach usually assumes that the two servers do not share a common storage for the VMs filesystem and it has to be transported to the new server.

In both cases, however, the assumption is that there is enough capacity on the receiving servers to accommodate more virtual machines. In a multi-server environment it becomes difficult to assess the possibility of, e.g., freeing one server from all its virtual machines. Do we have enough *combined* capacity to remove one physical server? How can we find the answer to that question?

We propose a simple notation for the redundancy of server capacity called the *redundancy level*. It is "*R*/*S*" where *R* is the number of servers that are in use currently and *S* the number of which servers can be shut down and removed. The idea is to calculate the available capacity on all servers and the current usage of all the virtual machines. From this, the *R*/*S* value can be derived. It is usually enough to consider only static resources as capacity limits with most focus on

memory and disk-space if the virtual machines are stored locally.

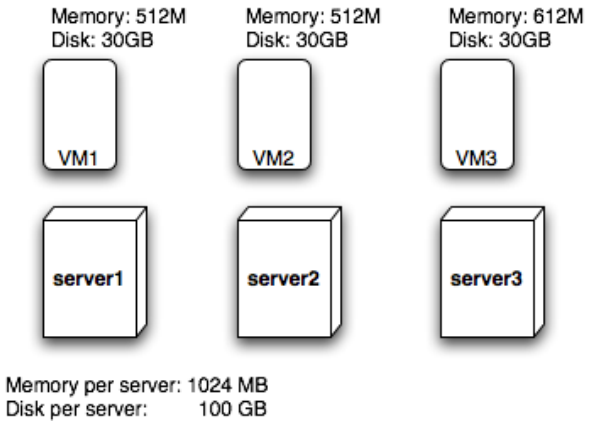

Figure 1: An example showing a server redundancy level of 3/0 because VM3 larger memory setting.

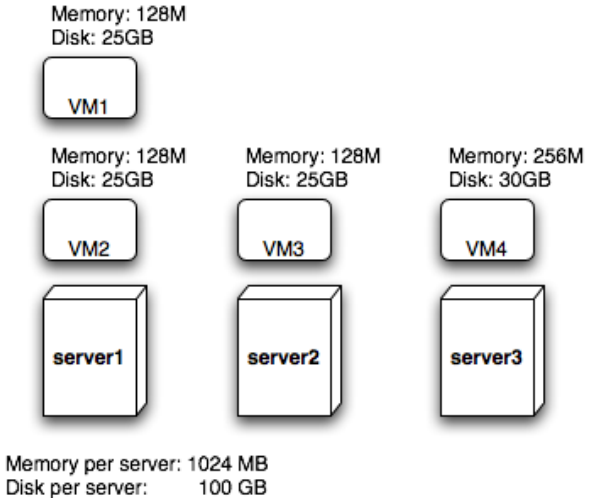

Figure 2: An example showing a server redundancy level of 3/1. With some initial planning, a level of 3/2 could have been achieved.

Example 1

In the following example we have three servers and three virtual machines: each of the three servers has 1024 MB of memory available for virtual machines as well as 100 GB storage space. VM1 and VM2 each use 512 MB of that memory together with 30 GB of storage for their harddisks. VM3 uses 612 MB of memory so that server3 has less free capacity than server2 and server1. The available memory (412 MB) is not enough to accommodate VM1 nor VM2. Server1 and server2 have both 512 MB of available capacity and cannot accommodate VM3. The result is that if server3 should be removed, we will loose VM3. The server redundancy level is 3/0 because we have three servers and no-one of them can be removed.

Clearly, if server1 goes down, then server2 would have enough spare capacity to hold VM1, but the "S" value in the redundancy metric should be as pessimistic as possible and must therefore hold for all servers. For *only* server1 and server2 we have, in fact, 2/1.

Example 2

This example is a little bit more complicated. The servers are the same as before, but now we have more light-weight virtual machines. We see, that the total memory consumption of all four virtual machines is 640 MB, that is less than the capacity of either one of the servers. This means that a single server could accommodate all the virtual machines, which would actually result in the redundancy value of 3/2. But unfortunately, since VM4 uses 30 GB of disk space, the sum of the total disk space consumption would be 105 GB, and that is more than a single server can offer. So the reality is that the redundancy level only is 3/1.

This example shows how some initial planning could improve the systems overall redundancy level. If one server would go down, we would end up with two servers with the combined memory capacity of 2048 MB but only of 640 MB of it actually used. On top of that, the redundancy level would then be 2/0, meaning two underutilized servers where no-one can be spared.

The redundancy level mirrors the infrastructures capability to loose one or more of its servers without loss of virtual machines and is therefore valuable information to a system administrator. It can be considered as a service level policy such as "*The infrastructure is to keep a* 3/1 *redundancy level 90% of the time.*" It can also be useful for capacity planning, calculating how much more virtual machines of a certain type can be deployed before the redundancy level is altered. It will also show how much resources a new server should have in order to keep a certain redundancy level for planned additions of more virtual machines. The notation should not be confused with a division, where 3/1 = 3. It is not meant to be a factor. The redundancy level of 4/2 is obviously also not the same as 2/1.

Resource Conflict Matrix

The previous analysis was from a infrastructure point of view. It did not address potential resource conflicts between the virtual machines. For this task, we propose creating matrixes for every resource type and to use graph theory for an indication of the number of resource conflicts we have. The analysis is simplest for the static variables. A resource conflict matrix is a square diagonal matrix with all the virtual machines. For each virtual machine pair, we put a 1 in their positions if they have a resource conflict for that particular resource type, otherwise a 0.

Let's consider filesystem placement as a static resource (i.e., not current disk usage). Two virtual machines have a conflict if their filesystem is placed

on the same harddrive. This would in principle imply, that two virtual machines placed on the same SAN are in conflict even if they run on different servers. These conflicts do in other words not imply that the performance is going to be poor. It simply states that there is a potential between those two virtual machines that they may influence each other.

The resource conflict matrixes are of size $V \times V$ where V is the number of virtual machines on the infrastructure. One technique to compress the matrix into a single value is to treat the matrix as an adjacency matrix for a bi-directional graph, and to calculate its connectivity using the formula:

$$\frac{C}{\frac{1}{2}V(V-1)}$$

The connectivity of a graph is the number of links divided by the possible number of links. In this case the number of conflicts C divided by the possible maximum number of conflicts. Its highest value, 1, would imply that there is a conflict between all virtual machines for that particular variable. A zero, 0, would mean the opposite. For each variable one can therefore easily get a value describing its current level of conflicts.

A simple example of the analysis is given in Figure 3. The same case would be for the static CPU resource if two virtual machines on the same single-CPU server would be in conflict. Few virtual machines, as used in our examples, may produce obvious matrixes. On large sites, however, the resulting matrix may become too large for manual review.

	VM1	VM2	VM3	VM4
VM1	0	1	0	0
VM2	1	0	0	0
VM3	0	0	0	0
VM4	0	0	0	0

(a) With all servers up. Conflict rate 1/6 = 0.1667 .

	VM1	VM2	VM3	VM4
VM1	0	1	0	0
VM2	1	0	0	0
VM3	0	0	0	1
VM4	0	0	1	0

(b) With server2 down and VM3 re-provisioned to server3. Conflict rate 2/6 = 0.333 .

	VM1	VM2	VM3	VM4
VM1	0	1	1	0
VM2	1	0	1	0
VM3	1	1	0	0
VM4	0	0	0	0

(c) With server2 down and VM3 re-provisioned to server1. Conflict rate 3/6 = 0.5 .

Figure 3: A resource conflict matrix for filesystem placement based on Example 2. As long as all servers are up, we have a low rate of conflicts. If server2 goes down (matrix b and c), the conflicts increase depending on how the virtual machines are re-provisioned. b) is a better solution than c) because it has the lowest conflict rate.

For the dynamic variables, such as CPU or IO usage, we need to compare the actual behavior of the virtual machines. If a time-series profile of each virtual machine existed, it could be compared to see if two virtual machines historically tend to demand the shared resources at the same time. If so, they are in a conflict. The level of correlation or probability of conflict between the two makes for a more fuzzy description than 1 or 0. One can still calculate the average probability rate between all the virtual machines, however it does not carry the same interpretation as the graph connectivity mentioned above, since the result would not represent the average number of conflicts anymore, but the average conflict probability. Other analysis methods could also be applied to the matrix, such as Principle Componant Analysis or centrality, however the interpretation of these results in this case are still being studied.

The resource conflict matrixes are a resource centric description of the infrastructure's state. Connectivity or average conflict levels are ways to compress information into more usable formats. Some level of intuition and knowledge is still needed to interpret its results. It can highlight uneven distribution of resources where there should be none. A matrix is beneficial for a graphical representation and can contain more information than only an average.

We also need a virtual machine-centric perspective of the infrastructure in case we investigate a particular virtual machine or want to make the best provisioning decision for a new one.

Location Conflict Table

The previous section showed how we can get an overall view of the resource conflicts and that re-provisioning of virtual machines influences the result based on what server it is placed on. The example was only for a single resource. How could we get an impression of the conflicts concerning all resources in order to make the best decision? Our solution to this problem is to list the available servers based on the resulting conflicts for all resources if that would be its location. Let us consider the case above where server2 goes down and VM3 needs to be moved. The resulting location conflict table is shown in Table 3.

Location conflict table for VM3

Location	Filesystem Placement	CPU	Total
server1	2	2	4
server3	1	1	2

Table 3: An example location conflict table showing server3 with less conflicts than server1.

For servers with equal performance, one should choose the one with the least resource conflicts. In many cases, some conflicts are unavoidable. Some conflicts may then be given more importance, such as those based on time-series profiles.

The information in this table will potentially change for all virtual machines every time a virtual machine is re-provisioned. This means that if server2 now contained two virtual machines, we would have to make a decision about the first and then the second as if the first has already moved.

Implementation Into MLN

The methods mentioned above are general ways to analyze the state of the infrastructure. The benefit with quantifiable metrics is that their calculation can be automated. The service redundancy level, resource conflict matrixes and location conflict table are implemented into MLN for static variables with preliminary support for dynamic variables also. MLN has enough knowledge about virtual machine location and resources to determine conflicts automatically.

The benefit of this information is currently studied in a master thesis with regard to automated re-provisioning and analysis [13]. MLN is only capable of providing current state values so it cannot compute "what if"-results for planning yet. Work is under way to include analysis features where the user could propose changes to the provisioning or other virtual machine metrics and see the result before they are committed.

Discussion

The server redundancy level only assumes that we will re-provision virtual machines from a given server onto the others. It does not assume that we could re-arrange all the virtual machines optimally on all server for the highest utilization. We have identified some additional challenges in that case which are still under consideration. The most important point is what strategy to use for the re-provisioning process. We have identified a few alternatives:

- **Least Migrations** This would reduce the risk of virtual machines dying in the migration process.
- **Least Memory Copied** Migration time depends on the main memory of the VM to be copied in the background. Less memory to copy would result in faster migration.
- **Minimize resource conflicts** Never-mind the number of migrations, just reduce the number of conflicts to a minimum.
- **Most Important Last** Some VMs are more important than others. Avoid touching the most important.

Which strategy is best depends on the context. In an automated re-provisioning scenario, the user should be able to express to the system which strategy is preferred.

A resource conflict is not equal performance degradation, but it may be a good indicator of where to look for answers if the virtual machine performs poorly. The resource conflict matrixes can be used to look up the current conflicts of a given virtual machine at any time. We consider this helpful support information for system administrators.

Some of the resources and conflicts are diffuse. Consider five single-CPU virtual machines on a four-CPU server. Who is in conflict with who? The strictest interpretation is that all five are in conflict with each other.

The connectivity or average conflict rate may not carry much information at the first calculation. The optimal or lowest possible value is entirely context dependent. It is therefore not suitable to be used as a comparison between two different infrastructures. However it is valuable as a measurement at re-provisioning. It is a way to observe how an addition of a new or existing virtual machine influences the rest.

Conclusions and Future Work

Virtual machines can easily be deployed using MLN and have acceptable performance levels through the Xen virtual machine framework. The ideal scenario would be one virtual machine on each server, but that is not always possible. Finding the best placement of the virtual machines in order to avoid resource conflicts is an open challenge.

Three methods were proposed for infrastructure analysis based on our own experience. Each method is technology independent and can be applied by hand for fair-sized infrastructures. They are also implemented in MLN in order to cope with larger scenarios where manual calculation is impractical.

We recognize that these methods have issues of their own but we see this as steps towards a better understanding of how to manage virtual machines successfully. This research field is only just emerging as more start to use virtual machines on a large scale and encounter the same kind of problems.

Work on MLN will continue in this direction and may function as a way to let the industry test our ideas and provide us with valuable feedback. Users should also be able to define other resources which are included into MLN's existing analysis. This may be easy to accommodate for static variables, however the dynamic ones would also imply that a monitoring framework of that variable should exist.

Related Work

In recent years, many researchers have promoted the use of virtualization and virtual machine technologies in scientific computing and HPC environments. Such arguments appeared initially in the literature of grid computing. For example, Figueiredo and coworkers [6] proposed using virtual machine technology in combination with management middleware to simplify

the task of seamlessly providing distributed computing resources to grid end users. They noted that using virtualization provided several advantages over grids constructed via real systems. Virtualization also offers many advantages to high performance computing; for a succinct summary, see Mergen, et al. [5].

Several groups have performed performance analyses of scientific problems in virtual environments. Figueiredo and coworkers [6] compared the performance of their virtualization-based approach with that of the regular operating system by running a few of the benchmarks from the SPEC suite directly on a Linux system and in a virtual machine running the same operating system under VMWare. Their results indicated that only that minimal overhead was associated with employing virtualization, in the range of about 2-4%. They also considered the time required for virtual machine setup, either via the full startup process (i.e., VM reboot) or by restoring a saved VM. Startup times generally ranged from about 30 seconds to 1 minute, depending of the specifics of the startup method and VM disk file storage location.

Youseff and coworkers [10] compared the performance of Red Hat Enterprise Linux running directly on the hardware as well as a guest operating system under the Xen environment via a series of standard benchmark applications. Their tests included three categories of calculation: micro-benchmarks each focused on a single system resource (CPU, network communication, memory and disk I/O), a series of matrix-based computations designed to test parallel program execution efficiency, and a single scientific simulation taken from the HPC Challenge benchmark suite (the MIT GCM exp2 which models a planetary ocean circulation process). In all cases, including the latter, these researchers found no statistically significant performance degradation from employing virtualization.

Bjerke [14] implemented Xen virtualization for HPC on Itanium systems. He presents some performance data for typical software building processes, again finding little difference between the native system and Xen virtual machine instances.

Finally, Vogels has explored virtualization for HPC in the Windows environment/.NET framework [15]. He ran benchmarks from the SciMark suite, focusing on comparing different virtual environments. In general, however, his results indicate that virtualization is a viable alternative for Java-based high performance computing in this environment as well.

Previous work on checkpointing has focused on general solutions for applications on UNIX systems. For example, Plank and coworker [7] created the libckpt library with the goal of rollback recovery for an executing program on UNIX systems. The library worked by periodically saving the application's current state to a disk file. In the event of a failure (i.e., program crash), the program could be restarted from the most recent save point (checkpoint). The facility was capable of operating in a fully automated mode, without requiring any program modifications, for existing applications; programmers could also add checkpointing-related directives to the code if the source code was available. The former is most directly comparable to virtualization. The researchers tested their approach using several computationally-intensive benchmarks. For these applications, the overhead associated with using libckpt in fully automated mode was considerable, ranging from about 5% to about 15%, with the larger values associated with the larger and most realistic benchmarks.

Wang and coworkers [8] performed similar work at about the same time. Their libckp library for UNIX systems similarly periodically saved the execution state from user applications in a transparent manner. Their work also included a generalization feature which allowed a checkpoint to be reused with different input/data.

VM management has also received substantial research attention. The closest work to MLN is probably the In-VIGO system of Adabala and workers [16]. It is an example of using virtualization for grid computing, specifically by constructing virtual grids on top of real systems via a software middleware layer. In this way, In-VIGO provides grid computing environment in which the actual physical systems and resources are transparent to grid users. It is designed to both simplify using the grid computing resources for end users as well as to simplify the grid management tasks for system administrators.

However, unlike MLN, it is a quite complex system (albeit a powerful one) necessitating a significant learning curve. In addition, many of its features are simply not needed for production scientific computing. The follow-on work, the VMPlants facility [17], provides a facility that is similar to MLN. It provides virtual machine creation, shutdown and cloning. Its operation is client-driven, in keeping with the goals of In-VIGO of simplifying the end user experience, contrast to MLN, which can employ either a push or pull approach.

Finally, Clark and coworkers [9] have studied migrating running virtual machines across physical hosts without the need for even temporary suspension. They found that the challenges of migrating live memory to be challenging, especially for applications with intensive memory write rates. They offer a solution for VM live migration within a group of discrete systems or a cluster with high performance network interconnects and network-based disk storage.

Computation Details

All Gaussian 03 [11] jobs were limited to 256 MB of memory and used the 32-bit version of the

program. The VMs used were allocated 512 MB of memory. The key components of the computer configurations were as follows: 1.8 GHz Xeon with 1 GB main memory (master node for parallel jobs); 2.8 GHz Xeon with 2 GB main memory; 2 GHz AMD-64 with 1 GB main memory. The network interconnect was 100BaseT. Parallel jobs made use of the Linda parallel execution environment [12], as required by the application. Job details: (1) HF/3-21G Opt Freq on Buckminsterfullerene, 540 basis functions; (2) B3LYP/3-21G Force SCF=NoVarAcc on Valinomycin, 882 basis functions; (3) MP2/6-311+G(2d,2p) Opt Freq on Malonaldehyde, 171 basis functions. All jobs were run on otherwise idle systems.

The following table gives the raw results for these jobs. The column headings have the following meanings: RHEL=job run on system running standard kernel; Xen=job run on system running Xen-enabled kernel, but not in a VM; VM=job run in a Xen VM (RHEL guest OS). The single processor jobs were all run on the slowest node, which also served as the master node for the parallel jobs.

Elapsed Time (seconds)

Job	Single Processor			Parallel Execution	
	RHEL	Xen	VM	Xen*3	VM*3
1	346	359	348	169	168
2	727	755	733	268	265
3	420	433	453	366	338

Table 4: Raw results as elapsed time for each test scenario.

Author Biographies

Kyrre Begnum is currently completing his Ph.D. in Network and System Administration at Oslo University College in Norway. Part of his research focuses on managing virtual infrastructures, and he is the author of the Manage Large Networks (MLN) VM administrative tool.

Æleen Frisch has been a system administrator for over 20 years. She currently looks after a pathologically heterogeneous network of UNIX and Windows systems. She is the author of several books, including Essential System Administration (now in its third edition). Æleen was the program committee chair for LISA '03 and is a frequent presenter at USENIX and SAGE events, as well as presenting classes for universities and corporations worldwide.

Ingard Mevåg is a graduate from Oslo University College with a masters degree in Network and System Administration. After part-time work at Game Index Statistics & Analysis AS during the education, he is now employed at ABC Startsiden AS as a system administrator.

Matthew Disney has been working in systems administration since 1998. He has a BS in Computer Science from the University of Tennessee and an MS in Network and System Administration from the University of Oslo. He is currently working as a cyber security administrator at Oak Ridge National Laboratory.

Bibliography

[1] Begnum, K. and J. Sechrest, *The MLN Home Page*, http://mln.sourceforge.net; Last accessed August 28, 2006.

[2] Begnum, K., "Manage Large Networks of Virtual Machines," *Proceedings of the 20th Large Installation System Administration Conference*, USENIX, 2006.

[3] Begnum, K. and M. Disney, "Scalable Deployment and Configuration of High-Performance Virtual Clusters," *CISE/CGCS 2006: 3rd International Conference on Cluster and Grid Computing Systems*, 2006.

[4] Begnum, K., K. Koymans, A. Krap, and J. Sechrest, "Using Virtual Machines for System and Network Administration Education," *Proceedings of SANE conference*, 2004.

[5] Mergen, Mark F., Volkmar Uhlig, Orran Krieger, and Jimi Xenidis, "Virtualization for High-Performance Computing," *SIGOPS Operating Systems Review*, Vol. 40, Num. 2, pp. 8-11, 2006.

[6] Figueiredo, Renato J., Peter A. Dinda, and José A. B. Fortes, "A Case For Grid Computing On Virtual Machines," *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, p. 550, IEEE Computer Society, Washington, DC, 2003.

[7] Plank, James S., Micah Beck, Gerry Kingsley, and Kai Li, "Libckpt: Transparent Checkpointing under Unix," *Proceedings of USENIX Winter 1995 Technical Conference*, pp. 213-224, New Orleans, Louisiana, January, 1995.

[8] Wang, Yi-Min, Yennun Huang, Kiem-Phong Vo, Pe-Yu Chung, and C. Kintala, "Checkpointing and Its Applications," *ftcs*, 00:0022, 1995.

[9] Clark, Christopher, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield, "Live Migration of Virtual Machines," *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 273-286, Boston, MA, May, 2005.

[10] Youseff, Lamia, Rich Wolski, Brent Gorda, and Chandra Krintz, "Paravirtualization for HPC Systems," Technical Report 2006-10, University of California, Santa Barbara and Lawrence Livermore National Lab, 2006.

[11] Frisch, M. J., G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, J. A. Montgomery Jr., T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M.

Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, J. B. Cross, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, S. Dapprich, A. D. Daniels, M. C. Strain, O. Farkas, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, and J. A. Pople. *Gaussian 03, Revision D.02*, Gaussian, Inc., Wallingford, CT, 2004.

[12] Scientific Computing Associates Inc., *The Linda Home Page*, http://www.lindaspaces.com .

[13] Ingard Mevåg, ‘‘Towards Automatic Management and Live Migration of Virtual Machines,’’ Master’s thesis, University of Oslo, Oslo University College, (UiO / HiO), Oslo, Norway, May 2007.

[14] Bjerke, Håvard K. F., ‘‘HPC Virtualization with Xen on Itanium,’’ Master’s thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, July 2005.

[15] Vogels, Werner, ‘‘HPC.NET – are CLI-based Virtual Machines Suitable for High Performance Computing?,’’ *SC ’03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, p. 36, IEEE Computer Society, Washington, DC, 2003.

[16] Adabala, Sumalatha, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu, “From Virtualized Resources to Virtual Computing Grids: The In-VIGO System,” *Future Generation Computing Systtems*, Vol. 21, Num. 6, pp. 896-909, 2005.

[17] Krsul, Ivan, Arijit Ganguly, Jian Zhang, José A. B. Fortes, and Renato J. Figueiredo, ‘‘VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing,’’ *SC ’04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, p. 7, IEEE Computer Society, Washington, DC, 2004.