Paravirtualization for Scientific Computing: Performance Analysis and Prediction

Javier Delgado, Anas Salah Eddin, Malek Adjouadi, S. Masoud Sadjadi
Florida International University
10555 W. Flagler Street
Miami, FL 33174
305.348.4106
{Javier.Delgado, Anas.SalahEddin, Adjouadi, Sadjadi}@fiu.edu

ABSTRACT

Resource virtualization technologies have recently increased in popularity. The emergence of cloud computing, which requires provisioning isolated environments on shared resources, is one reason for this. Virtualization adds flexibility in terms of resource provisioning, but it can impact application performance.

In this work, we analyze the performance of medical image processing and computational fluid dynamics applications when run on virtualized resources. We then apply the observed performance characteristics to a performance prediction model.

We measure the impact of virtualization by performing several benchmarks on virtualized and non-virtualized resources. We evaluate the accuracy of the performance prediction model in this environment. We find that virtualization can slow down some applications by more than 200%, but usually the performance impact is below 15%. The overhead itself is predictable if general application characteristics are known. Execution time in a virtual environment can be predicted to within 13% using a simple mathematical prediction model.

Keywords

Measurement, performance, analysis, Virtualization, Xen, image processing, parallel processing, weather modeling.

1. INTRODUCTION

Resource virtualization has experienced widespread use in several computing domains. Virtualization, in this context, refers to the use of virtual machine (VM) technology to run VMs on top of physical machines (PMs). Currently, this is accomplished using a virtual machine manager (VMM) that deploys *guest* VMs and controls their resource allocations. One reason for the resurgence of virtualization is for its ability to provision resources for *Cloud* computing, which can be a cost effective alternative to in-house clusters [1]. This is particularly the case for domain specialists that are not constantly using compute resources. The benefits of virtualization can be used for traditional data center resource provisioning, for *Clouds*, or even for Grid computing (e.g. [2]).

Virtualization has several benefits for scientific computing. Some of these have been cited in the literature (e.g. [3][4]). For example, users running serial applications can be provisioned isolated computing environments on shared multicore machines. Also, a VM can be suspended and later resumed on either the same or on a different physical machine, which is useful for fault tolerance and load balancing. VMs also make it easier to deploy applications, since virtualization adds a layer of abstraction that eliminates the need to configure the necessary software environment for all physical machines on which an application will be run. Instead, a single VM image is created and deployed on any machine with a compatible VMM. In the context of scientific computing, these special purpose VMs are often referred to as *virtual appliances.* This feature is especially beneficial for scientific applications, which can be difficult to install and/or dependent on legacy libraries, and are executed by domain specialists that are not familiar with the deployment process. For the remainder of this paper, it can be assumed that all VMs are virtual appliances.

The drawback is that virtualization creates overhead, which raises a valid concern for consumers of resource-intensive applications. This overhead is caused by the VMM itself, the additional processing of certain instructions (e.g. memory allocation and I/O) within the VM, as well as the competition created when multiple VMs are running on the same physical machine.

In this work, we explore the use of virtualization for data centers that execute scientific applications as batch jobs, with a specific focus on medical image processing and computational fluid dynamics (CFD) applications. Specifically, we look to answer two questions. First, how is the performance of these applications affected? Second, can a simple prediction model be used to determine execution times of these applications in a virtualized environment? We perform experiments using a production cluster. We attempt to quantify the performance impact under different run time scenarios and with different applications. We then apply a performance prediction model to the results. Performance prediction can be used to provide improved scheduling performance, as has been shown, for example, in [5]. The model we use has worked successfully on non-virtualized platforms [6].

As part of these experiments, we look into the effects of collocating multiple VMs on a single physical machine to observe how individual VMs are affected. Collocation can increase system utilization. It can also be used in job scheduling, for example, in a scenario in which all machines in a data center are occupied running long (duration) jobs, and a short job enters the system. Collocating may delay the long job, but it is standard practice to give short jobs preference since customers expect them to be processed quickly. Scheduling methods such as those employed in [7] enjoy strong scheduling performance in part because they do not set rigid constraints on the number of tasks per physical machine, thus lowering the response time of short jobs. As a practical example, consider two jobs with different quality of service requirements. For example, a medical professional requiring an image analysis has more immediate needs than a developer testing a new algorithm.

We find that for scientific applications that perform little or no communication, the performance impact is small enough that virtualization is a viable option. Applications that require constant access to the resources, on the other hand, will not benefit much, if at all, from virtualization, especially if their applications are communication intensive.

We also find that predicting the execution time of applications run on virtualized systems benefits from predicting the I/O and computation times of the execution separately. By doing this, we could predict execution time to within 13%.

We organize the rest of this paper as follows. In section 2, we describe the experimental setup and the applications used. In section 3, we describe the performance prediction approach taken; in section 4, we describe the experiments and discuss observations; in section 5, we evaluate the prediction model; in section 6, we provide related work. We end with our conclusions.

2. SOFTWARE AND SYSTEM CONFIGURATION

2.1 Infrastructure

We use a 16-node compute cluster for the experiments. Each node in the cluster contains 2 single core Intel Xeon processors with *hyperthreading* technology rated at 3.6GHz; they are based on the *Netburst* CPU architecture. Each node has 2 GB of main memory. The nodes are connected using a 1gigE interconnection. The operating system is the *CentOS* Linux distribution, version 5.3, included in version 5.2 of the *Rocks* cluster distribution [8]. Half of the physical nodes were configured as *compute* nodes, which are used for the BM experiments. The other half were configured as *vm-container* nodes using the *Rocks Xen* roll[1], which deploys selected worker nodes with *Xen*, version 3.0.3 [9]. VM images used for the virtualized experiments contain the same *CentOS* distribution, including the same kernel version. VMs are deployed using *OpenNebula* [10].

For our experiments, we allot 2 cores to each VM and execute up to 2 processes[2] per node. Unless otherwise stated, each VM can use the full processing power of each allotted processor. In the container nodes, the *dom0* VM is allowed to use the virtual processors (i.e. *hyperthreads*). We noticed performance degradation when allowing *Xen* to dynamically change the virtual to physical CPU mappings, so each virtual CPU in a VM is pinned to a specific physical CPU.

All software and guest VM images were installed on the same shared file system, which is hosted on the master node of the cluster. A virtual network was created using *OpenNebula* to link the VMs. Xen's standard network bridging configuration was used.

2.2 Parallel CFD Benchmarks

We employ NASA's Numerical Aerodynamic Simulation Parallel Benchmarks (NPB) as a first step towards understanding the performance impact of *Xen*. The NPB contain three benchmarks that replicate computation and communications patterns of Computational Fluid Dynamics (CFD) and computational aerodynamics applications [11]. Specifically, they provide different kernels for solving Navier-Stokes parallel differential equations on a spatial grid or *mesh* of a given size. In this work, we focus on the lower-and-upper triangular (LU) benchmark. The LU benchmark is iterative and consists of 250 time steps.

There are two versions of the NPB, *original* and *multi-zone* (*MZ*). The computations in the original benchmarks exhibit fine grain parallelism [12], i.e. they perform multiple communications of data for each iteration of the solving stage. As a result, their performance is more sensitive to communication latency. The *MZ* versions take a parallelization approach that mimics different kinds of applications. They solve the same discretization problem, but using multiple meshes (or *zones*). The MZ benchmarks are designed to perform only coarse-grained parallelism at the message passing level[3]. The MZ version is more sensitive to load imbalance than latency, so their performance should be less affected by virtualization. We employ both versions of the benchmarks to see how their performance impacts compare.

The benchmarks come with five data inputs of increasing size, which they refer to as input *classes*. We use classes A, B, and C. Their sizes and dimensions are shown in Table 1.

Table 1. Sizes of the three classes of NPB inputs used

Class	Dimensions	Area
A	128 X 128 X 16	262K
B	304 X 268 X 17	1.075M
C	480 X 320 X 28	4.3M

NPB also contains an *embarrassingly parallel* (EP) benchmark, which we use to observe performance when little communication is involved. EP is designed to test the floating point performance of the system [11]. The only communication it performs is a barrier at the beginning of the program and a collective (*reduce*) operation at the end.

The NPB are well studied and often used to test the performance of HPC systems. The multi-zone versions have been shown to scale well on up to at least 16 processors, and possibly over 1,000, depending on the benchmark and runtime configuration [13]. The purpose of the NPB experiments is to observe the performance that can be expected from certain CFD-like applications when executed in a Xen environment.

2.3 Medical Image Processing

The medical image-processing domain consists of performing complex, computationally intensive algorithms on large sets of images. Virtualization is well suited for these workloads since they perform little or no inter-process communication when processing unrelated data sets.

The image processing applications we focus on will be set on neuroimaging; specifically, we will be focused on brain magnetic resonance image (MRI) processing. All of the algorithms used are implemented in the FMRIB Software Library (FSL) [14]. MRI studies can be dichotomized into structural and functional studies [15]. Structural studies deal with the variability between adjacent brain tissues; we employ a segmentation tool called FMRIB Automated Segmentation Tool (FAST) [16] for these experiments. The algorithm it implements segments the basic tissues of the brain: gray matter, white matter, and cerebral spinal

[1] http://www.rocksclusters.org/roll-documentation/xen/5.3/

[2] By *process*, we mean a process created by an application being tested. For example, an MPI task.

[3] Fine-grained parallelism is performed among threads using OpenMP, but we do not experiment with this in our tests. When multiple cores per node are active, each is assigned an MPI task.

fluid. FAST is an iterative algorithm that depends on the within tissue variability while addressing problems arising from: image noise, head motion artifacts and inhomogeneity in the magnetic field, all of which affect the performance and speed of the algorithm. On the other hand, functional studies deal with the temporal differences in the activation of neurons. We test an exploratory algorithm called Multivariate Exploratory Linear Optimized Decomposition into Independent Components (MELODIC), which consists of a pipeline of algorithms that can be summarized to: motion artifacts correction, image registration [17] to high resolution MRI and to a standard brain image, and finally probabilistic independent component analysis [18].

Data from 4 patients from Miami Children's Hospital were used for the FAST experiments. For MELODIC, 20 patient data sets from the same hospital were used. Each of the latter data sets contain a high resolution MRI and at least one functional MRI (fMRI) dataset, each fMRI dataset consist of 150 volumes in total. The algorithms are applied to each patient separately, so the algorithms themselves do not need to be implemented in parallel; instead, a loosely coupled parallel implementation in which the algorithm is separately applied to each image can be used.

FSL includes built-in mechanisms for automatically spreading jobs across different nodes/processors/cores using the Oracle Grid Engine (*GridEngine)*. *GridEngine* was installed and configured on all BM and VM nodes. Since these experiments involve little or no communication, the VM overhead should be minor.

2.4 Weather Modeling

We perform several weather simulations using the Weather Research and Forecasting (WRF) software model. These benchmarks reveal the performance of virtualized resources executing computationally intensive, highly synchronized (i.e. tightly-coupled) parallel code, where high virtualization overhead is not tolerable.

Due to the long-running and highly synchronized nature of WRF executions, the effect of running small jobs alongside WRF is interesting. This way, the resource isolation functionality provided by VM technology can be leveraged to execute small jobs alongside the large WRF job in order to avoid delaying the short job when there are no idle processors available. Therefore, we perform experiments in which small jobs are allocated alongside WRF jobs of different sizes and see what the performance impact is on each. Another possible benefit of collocation is that it may mitigate the virtualization overhead of communication intensive applications such as WRF, which can leave the CPU idle for relatively long periods of time.

We use two WRF input domains, whose properties are shown in Table 2.

Table 2. Properties of the WRF domains used

Domain Name	Dimensions (grid points)	Resolution (km)	Duration (hours)
Jan00	74x61	30	24
75x4	75x75	15	24

3. PREDICTION

A challenge encountered in data centers is that their job schedulers must know the execution time of all submitted jobs. In production systems such as the *Teragrid*, users are asked to provide this information. If a user under-estimates a job's execution time, it is prematurely killed. If a user over-estimates the time and the system is highly utilized, the scheduler may delay the job in favor of shorter jobs. An alternative is to use automatic, system-generated predictions.

System-generated performance prediction is of additional interest in virtualized data centers, where it is common for multiple virtual machines to run on a physical machine, since users cannot estimate the execution time of a program if they do not know how much of the physical resources will be devoted to their job(s). They may not even know the specifications of the physical resources being offered. For example, in a situation where a small job enters a fully-utilized system, knowing what impact collocating the small job with a bigger job would have on the execution time of both jobs is useful to the job scheduler.

For this work, we apply our existing prediction model [6] to the problem of estimating execution times of applications run on a virtualized cluster. The model is implemented using a regression analysis based predictor, *Aprof*, to predict application execution time using historical run time information. The historical information is obtained using our monitoring program, *Amon*. Essentially, the predictor estimates the contribution of different resource consumption metrics, such as parallelism and CPU allocation, to an exploratory metric, such as execution time. Equation (1) shows the general form of the equation for estimating execution time. In the equation, m is the number of resource consumption parameters, z_i is the i-th parameter and a_i is its contribution coefficient. For inverse relationships, we use the inverse of the resource consumption value. For example, execution time is inversely proportional to the number of nodes, so the inverse number of nodes is given to the model. In [6], we tested the model using 2 clusters; to mimic having additional systems with different processing power, we used a CPU-limiting tool[4] to adjust the processor allocation for the jobs being executed; we obtained prediction accuracy within 10% across the two systems in that study.

$$T_{exec} = \prod_{i=0}^{m-1} a_i z_i \quad (1)$$

Since the model does not necessarily require specific hardware parameters, it is reasonable to expect it will work across different systems. For example, our model, with a few modifications, was able to predict execution times across several larger, more modern systems with under 10% average error [19]. Hence, it is reasonable to expect that the prediction methodology will also work on virtualized systems after the necessary modifications are applied.

4. EXPERIMENTS

In this section we describe the experiments performed. All data reflects the average of 3 executions for each runtime configuration (i.e. platform, number of nodes, and processes per node).

4.1 NPB LU Benchmark

Since the communication portions of the executions are more susceptible to VM overhead, we observe communication and computation times separately. We used the timers built into the original and MZ benchmarks to obtain these values. The original benchmark's timers explicitly measure and report communication time. The MZ benchmarks measure the time it takes to exchange boundary values, which is the only process-level communication performed.

[4] http://cpulimit.sourceforge.net/

Figures 1 and 2 show the overall communication and computation times with up to 8 nodes for the MZ benchmarks, using input classes A and C, respectively. We only show separate times when running 2 processes per node, since the 1 process-per-node executions experienced less than 5% virtualization overhead. The overall overhead for 1 and 2 process per node executions with Class A can be seen in Figure 3. The overhead was most significant when executing with the smallest input, as depicted in Figure 1. With the largest input, whose execution times are depicted in Figure 2, the computation times overshadow the communication overhead. The pattern is similar for the other executions. Computation time increases roughly linearly with the number of nodes for all configurations up until 8 nodes. The 8 node BM Class A executions exhibited longer than expected I/O times, so the overhead was less noticeable. Only the Class A executions had non-linear execution time scaling, due to the communication overhead.

Overall, the figures indicate that the total execution time is always slightly faster for the BM executions, except for the 2 and 4 node, multi-process Class A executions, which suffered significant slowdown when executed in the VM. Since CPU utilization is not high, we suspect that the larger overhead for multi-process executions is due to contention accessing the network interface. The fact that the overhead is more significant when multiple nodes are used suggests that there may also be overhead due to synchronization of computation and communication cycles among the nodes. The fact that the execution time varies so much for separate executions with 2 processes per node supports this point. The lower overhead with 1 process per node is also partially due to the fact that the VMM has an entire processor for itself.

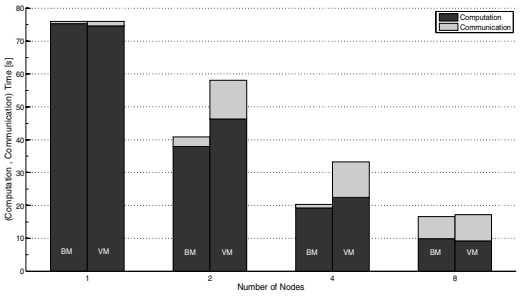

Figure 1. Communication and computation times for LU-MZ Class A when running 2 processes per node

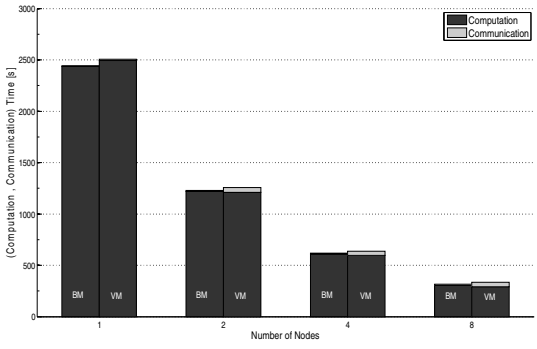

Figure 2. Communication and computation times for LU-MZ Class C when running 2 processes per node

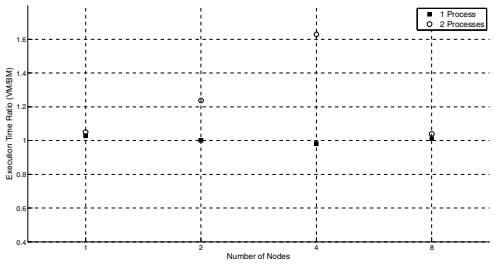

Figure 3. Ratio of execution times (BM vs. VM) for Class A executions

The VM impact is larger for the original LU benchmarks, especially for Class A. This is because they transmit more data than the MZ benchmark. Although we suspected that the round-trip delay in the network due to VM overhead could be the culprit, we did not find this to be the case after several tests using the *ping* command. As with the MZ experiments, only the multi-process executions had a large performance impact, so we only show their results, in Figure 4. A noteworthy observation in this figure is that the linear curve approaches an asymptote for Class A after 4 nodes.

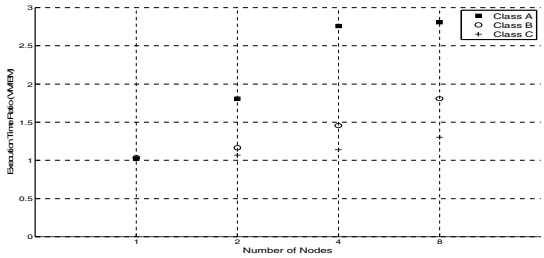

Figure 4. Execution time ratio for the original LU benchmark, running 2 processes per node.

4.2 WRF

As was expected after observing the performance of LU, WRF suffered large virtualization overhead for multi-process executions. Only the original LU benchmark with the smallest input size resulted in higher overhead. Single-process executions of WRF experienced minor overhead. The VM overhead is depicted in Figure 5. A noteworthy characteristic of these executions is that both input domains experienced similar overhead, unlike in LU, for which the larger inputs experienced less overhead. The cumulative CPU and I/O times of all processes for the two WRF domains showed similar distributions, despite the fact that the *75x4* domain took longer to complete.

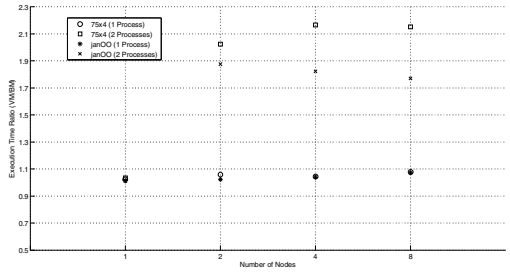

Figure 5. Execution overheads for WRF executions of two domains using 1- (a) and 2- (b) processes per node.

4.3 Image Processing with FSL

4.3.1 Image Segmentation

Surprisingly, the virtualized executions of FAST were 10-15% faster for each of the 4 data sets. Since there is no communication involved, we expected the performance to be about the same. To help explain why the virtualized executions were faster, we profiled the system using *Oprofile* while executing the segmentations on each data set. The execution profiles were similar for all data sets, so we only show it for the first data set, in Figure 6. As can be seen, function *A* has a disparity between the BM and VM executions. This function, which is the *convolve* function, is part of the bias removal which occurs before the segmentation algorithm. We determined that there is a 3-fold slowdown for 1/3 of the calls, specifically, when performing the convolution in the horizontal *k* direction. We found that the BM executions result in 30 cache write misses at the first and second cache levels when iterating, whereas the VM executions never miss the cache.

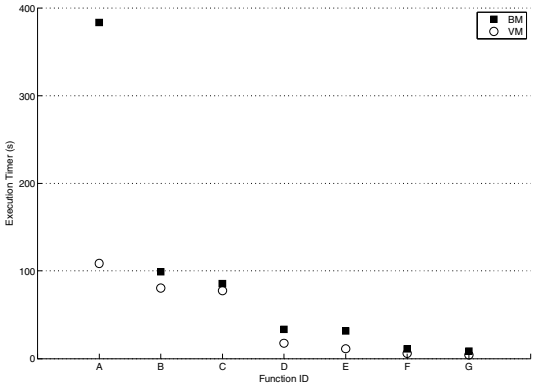

Figure 6. Cumulative execution times for the 7 most time-consuming functions of the segmentation algorithm for one data set.

4.3.2 Image Registration

Comparing the VM and BM performance of the image registration experiments was not as straightforward as with the other applications. The algorithm used by MELODIC varies from execution to execution, even with the same input, due to a random component of the ICA algorithm used by MELODIC [18]. The ICA algorithm does not terminate until it converges, and the number of steps required until it converges depends on the random initial value. We observed anywhere from 63 to 136 steps before converging for identical executions.

While this variance makes it difficult to measure the effect of virtualization, one clear pattern is that the VM executions were consistently slower when simultaneously processing 2 data sets per node. When only one data set at a time was processed on each node, the average overhead was negligible. When all data sets were submitted in batch, the overall VM slowdown was 13%, 10%, and 13% for 1-, 2-, and 4-node executions, respectively. Figure 7 compares the completion times of each data set for the VM and BM experiments for isolated and 4-node executions. The relationship between the VM and BM executions is always the same, with the BM finishing slightly faster.

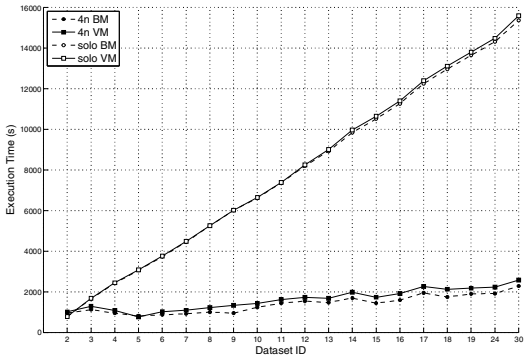

Figure 7. Completion times for the 20 data sets of the image registration job when each is run isolated (*solo*) and when using 4 nodes.

4.4 NPB EP Benchmarks

So far, we have seen the performance impact of very tightly-coupled parallel applications (e.g. WRF and LU) and very loosely-coupled parallel applications (e.g. MELODIC). The EP benchmark falls somewhere in between these extremes. The results for the EP executions suggest that there is no significant overhead. In fact, the VMs sometimes execute the program slightly faster. This suggests that the VM overhead is most sensitive to the frequency of communications. We omit these figures due to space limitations.

4.5 Collocation Effects

Based on the results shown earlier in this section, communication-intensive applications suffer high overhead when run in VMs, particularly when running 2 processes per node. It is of interest to see if the overhead could be mitigated by running serial jobs on collocated VMs. To do so, we ran experiments in which an FSL VM was collocated with one of the WRF worker VMs on the same physical machine. The CPU allocation given to the WRF VM was limited to 50%. The FSL VM was allotted 1 CPU, which was pinned to the second physical processor. The results are shown in Table 3. As the table shows, the 1 process-per-node WRF execution did not slow down significantly due to being limited to 50%, even with the CPU-intensive image registrations constantly executing. The 2 process-per-node execution was slowed down slightly, but it is a worthwhile tradeoff for highly utilized systems.

Table 3. Execution times of WRF when simultaneously executing consecutive image registrations on a collocated worker VM.

Input	#Nodes	Proc. Per node	Alone, 100%	Alone, 50%	Colloc., 50%
75x4	8	1	2721	2722	2727
75x4	8	2	3714	3714	3894

5. PREDICTION MODEL ADJUSTMENTS AND PERFORMANCE

In this section, we discuss the changes made to the prediction model, based on the observations of the experiments. We then evaluate the accuracy of the updated model.

5.1 Model Modifications

As the results in the previous section confirmed, the overhead of running within a Xen environment depends on applications' characteristics, data input size, and runtime configuration. The overhead for computation-intensive portions of the execution is minor and predictable. The overhead for I/O portions is much larger. Guest domains in *Xen* incur overhead when translating memory addresses for direct memory access (DMA) devices, such as the network card, from the privileged domain (*dom0*) to the guest domain (*domU*). This translation is done in software on the version of *Xen* used [20]. Running multiple processes per node exasperates this problem since additional contention occurs due to each process waiting for the translation. This behavior itself is difficult to model. The fact that communication-intensive and computation-intensive chunks of the execution scale differently complicates the performance modeling.

To address this, we modify the prediction methodology. Instead of modeling the wall clock time as a whole, we predict communication and computation time separately. For the computation time, the *user time* (i.e. CPU time spent in user space) collected by *Amon* was used. Communication time is not as simple to obtain using a lightweight monitor such as *Amon*. We use a simple estimator, t_{io} or simply *iotime*, which is the difference between wall clock time and user time, as shown in Equation (2).

$$iotime = t_{io} = t_{wall} - t_{cpu} \quad (2)$$

Before evaluating the revised model's ability to predict execution time, we test the efficacy of the values chosen to separate the CPU and I/O times by comparing them to the values of communication and computation time reported by the timers included with the NPB benchmarks. The computed correlation coefficients for all configurations of the VM executions of LU-MZ were 0.99 (computation) and 0.95 (communication). We consider this a good starting point for the model, hence, we use *user* time as the computation time estimate and *iotime* for the communication.

We measured the synchronous MPI bandwidth of the BM and VM configurations using a simple *ping-pong* test[5] that measures the bandwidth for transfers of different message sizes ranging from 8 Bytes to 1 MByte. The test was run 20 times and the average bandwidth of all runs was taken. The BM node was consistently about 40% faster throughout the range of message sizes evaluated. According to [21], the message sizes for the LU-MZ benchmarks range from approximately 220-350 kB for Class B to 600-950 kB for Class C, for systems with 2-16 processors. Since there is not much variation in the measured bandwidth for this range, the average of the 128, 256, 512, and 1024 kB measurements are used as the network bandwidth metric.

5.2 Prediction Results

The prediction model was evaluated using NPB and WRF. The resource consumption parameters used to estimate the computation times were: inverse number of nodes, inverse number of processes per node, and inverse memory bandwidth. To predict I/O time, the number of nodes, number of processes per node, and inverse of network bandwidth were used. The network bandwidth was adjusted depending on the number of processes per node. Runtime configurations consisted of using 1, 2, 4, and 8 nodes and 1 and 2 processes per node, for a total of 16 data points per experiment. The overall error was calculated using Equation (3), in which *io* is the *iotime* and *u* is the user time.

The actual and predicted computation and communication values for the LU and LU-MZ benchmarks with up to 8 nodes are shown in Figure 8. The predictions were performed separately for each class and for each implementation (i.e. *original* and *MZ*), for a total of 6 sets of experiments. The mean and median prediction errors were 13% and 4%, respectively. The large average error was due to outliers caused by exaggerated communication predictions. This was due to using a linear model even though the communication behavior could not be modeled linearly. The results could be further improved, for example by using a non-linear predictor or by considering 1- and 2- processes per node executions separately. We leave this for future work.

$$error = \frac{|(io_{actual}+u_{actual})-(io_{estimated}+u_{estimated})|}{(io_{actual}+u_{actual})}100 \quad (3)$$

The same experiments were repeated for WRF, using the same run time configurations and the *jan00* and *75x4* domains. The actual and predicted execution times are shown in Figure 9. The mean and median errors in this case were 9% and 6%, respectively. The mean error was more tolerable for WRF by virtue of its longer execution time. The NPB results were skewed due to the higher error of the Class A predictions.

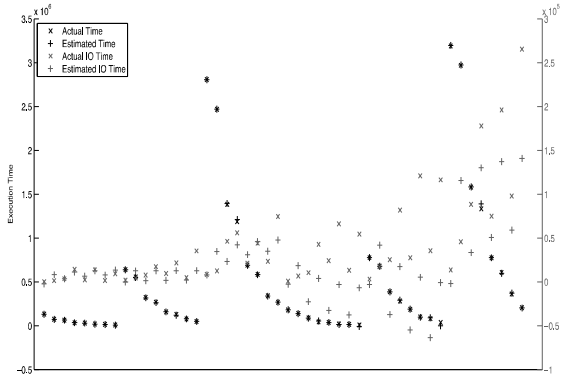

Figure 8. Predicted and actual computation (dark colors) and I/O (light colors) times for LU and LU-MZ.

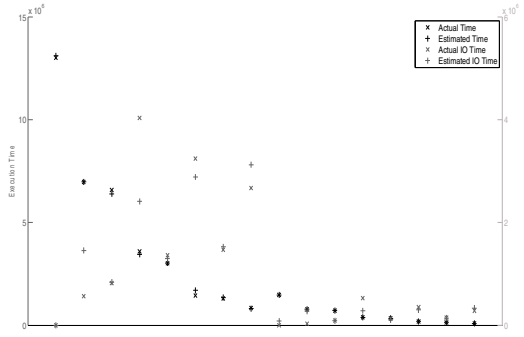

Figure 9. Predicted and actual computation (dark colors) and I/O (light colors) times for WRF

6. RELATED WORK

There have been a number of different works covering the performance of different applications being run on the Xen-

[5] http://www.scl.ameslab.gov/Projects/mpi_introduction/para_ping pong.html

enabled Amazon *Elastic Cloud (*EC2*),* such as data mining [22] and genomics [1]. The authors in [23] performed a thorough performance analysis on the EC2 resources to determine the performance impact of running HPC applications in the cloud. Rather than focusing on the performance of a specific cloud provider, we focus on the impact of the virtualization itself for different kinds of applications. This kind of information is useful for people interested in deploying their own cloud-like infrastructure or who are looking to take advantage of some other benefit of virtualization.

Virtualization impact has been studied as well. In [4], the authors evaluate the performance impact of Xen on the NPB and other benchmarks. They observe similar overhead on the high I/O benchmarks when run on a commodity cluster. They implement a method that achieves near-native performance, but it requires more expensive, *Infiniband* based hardware. They only evaluate Class B of the NPB. In contrast, we focus on just the LU benchmark, but on different input sizes and on both the original and multizone implementations. This way, we can comment on how virtualization overhead varies according to input size as well as how different parallel implementations of an application are affected by virtualization. In [3], the authors perform several micro- and macro-benchmarks on virtualized and non-virtualized systems. The benchmarks they ran included some NPB benchmarks, but only Class C. In addition to the other applications we analyzed, we ran Class A and B of the NPB, which happened to have more overhead, which we believe is an important observation. While their results are indicative of less performance overhead from virtualization, even their bare metal executions have a lower computation to communication time ratio, which can be due to some other issue. One possibility is a larger amount of memory bus contention, since their infrastructure consisted of 4 cores per physical node. The study in [24] also uses the NPB and HPCC benchmarks to evaluate Xen overhead. They use more low-level profiling details such as the number of TLB and cache misses. While their results are interesting, they find that they cannot achieve their goal of characterizing VM performance by application. Furthermore, the data they collect is too in-depth for the performance prediction approach we use.

To summarize, our work enhances existing knowledge about virtualization impact by analyzing a broad range of parallel applications with different communication characteristics, giving insight as to how their virtualization overhead may vary depending on the applications' characteristics, input characteristics, and the execution platform's runtime configuration. Our work attempts to provide additional performance-related details in the context of the applications tested and uses this information to modify an existing performance prediction model. This model will be used for performance-aware scheduling that is part of future work.

Performance prediction, for job scheduling and other applications, has been researched extensively. While performance aware scheduling has been explored in works such as [5], we attempt to look into specific problems that arise when scheduling with VMs rather than on physical machines. A thorough summary of existing performance prediction approaches is beyond the scope of this paper. Essentially, the methods that have been presented can be broken down into two general categories, simulation-based and historical. The latter include statistical methods (e.g. [25][26][27]), machine-learning methods (e.g. [27]), etc. Simulation-based methods (e.g. [28]) rely on detailed knowledge of the execution performance of the hardware and software involved. Methods based on historical runtime information are preferable for job scheduling. For one, due to the complexity of modern processors and the heterogeneity of HPC systems, creating accurate simulation models is quite difficult. Also, the execution behavior of a program varies due to changes in memory state, active processes, etc. Another reason is the need for fast response time (e.g. for making scheduling decisions). Simulation-based methods are more computationally intensive due to the complex modeling involved. Finally, historical methods do not require that the model account for every individual hardware component involved in the applications' execution.

7. CONCLUSION

We have shown the execution time impact of virtualization for scientific applications with different levels of inter-process synchronization requirements. We found that the virtualization overhead is usually below 10%, but can be much higher with communication intensive and/or highly synchronized parallel applications running multiple processes per node. In fact, over 200% overhead was seen for one of the tested applications. On the other hand, it is possible for an application to execute faster in the virtualized environment, as we noticed with the image segmentation software we tested. Parallel applications that have fine-grained communication behavior have more overhead than those that communicate more sparsely. The results for the applications we evaluated corresponded with this; i.e. the most tightly coupled was WRF, followed by LU, LU-MZ, and then EP and the magnitudes of their performance impacts were also in that order, except for very small inputs. We find that the slowdown can be mitigated by collocating high-overhead applications with low-overhead applications. Doing so may also help avoid starvation of short jobs in a batch environment.

We also examine the efficacy of using an existing execution time prediction methodology in the virtualized environment. To account for the disproportionate virtualization overhead of I/O operations, we modified the predictor such that it distinguishes between computation and I/O time. After this modification, the average execution time prediction errors were within 13%.

In the future, we will fine-tune the prediction model's ability to predict I/O times. The model's ability to predict execution times of collocated jobs will then be evaluated. These enhancements will make it more suitable for making resource allocation decisions for scheduling batch jobs. We will also continue to evaluate the virtualized performance of different applications to gain a deeper knowledge of virtualization impact.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Numbers OISE-0730065, CNS-BPC-AE-1042341, CNS-MRI-R2-0959985, HRD-CREST-0833093.

9. REFERENCES

[1] DP. Wall, P. Kudtarkar, VA. Fusaro, R. Pivovarov, P. Patil, and PJ. Tonellato. "Cloud computing for comparative genomics." BMC Bioinformatics. 2010 May 18;11:259

[2] M. Zhao, J. Zhang, and R J. Figueiredo. 2006. "Distributed File System Virtualization Techniques Supporting On-Demand Virtual Machine Environments for Grid Computing." Cluster Computing 9, 1 (January 2006), 45-56. DOI=10.1007/s10586-006-4896-x

[3] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Paravirtualization for HPC Systems," Workshop on XEN in HPC Cluster and Grid Computing Environments

(XHPC), held in conjunction with The International Symposium on Parallel and Distributed Processing and Application (ISPA 2006), December 2006.

[4] W. Huang, J. Liu, B. Abali, and D. K. Panda. 2006. "A case for high performance computing with virtual machines." In Proceedings of the 20th annual international conference on Supercomputing (ICS '06). ACM, New York, NY, USA, 125-134. DOI=10.1145/1183401.1183421

[5] D. Tsafrir, Y. Etsion, and D. G. Feitelson. "Backfilling using system-generated predictions rather than user runtime estimates." In IEEE TPDS, 18:789–803, 2007.

[6] S. M. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, and J. Delgado, H. Duran, and X. Collazo. "A modeling approach for estimating execution time of long-running scientific applications." In International Parallel and Distributed Processing Symposium - IPDPS 2008, April 2008.

[7] M. Stillwell, F. Vivien, H. Casanova. "Dynamic Fractional Resource Scheduling for HPC Workloads." In Proceedings of IPDPS'10, Atlanta, Georgia, April 2010.

[8] P. M. Papadopoulos, M. J. Katz, and G. Bruno. 2001. "NPACI Rocks Clusters: Tools for Easily Deploying and Maintaining Manageable High-Performance Linux Clusters." Proc. of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Yannis Cotronis and Jack Dongarra (Eds.). Springer-Verlag, London, UK, 10-11.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. 2003. "Xen and the art of virtualization." Proc. the nineteenth ACM symposium on Operating systems principles(SOSP '03). ACM, New York, NY, USA, 164-177. DOI=10.1145/945445.945462

[10] Distributed Systems Architecture Research Group: OpenNEbula Project [URL]. Universidad Complutense de Madrid (2009), http://www.opennebula.org/

[11] D. Weeratunga, E. Barscz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, et al. "The NAS Parallel Benchmarks," (March 1994) NAS Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA

[12] R. van der Wijngaart and H. Jin, "NAS Parallel Benchmarks, Multi-Zone Versions," (July 2003) NAS Technical Report NAS-03-010, NASA Ames Research Center, Moffett Field, CA

[13] H. Jin, R. Van der Wijngaart, "Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks," IPDPS, vol. 1, pp.6b, 18th International Parallel and Distributed Processing Symposium (IPDPS 04).

[14] S.M. Smith, M. Jenkinson, M.W. Woolrich, C.F. Beckmann, T.E.J. Behrens, H. Johansen-Berg, P.R. Bannister, M. De Luca, I. Drobnjak, D.E. Flitney, R. Niazy, J. Saunders, J. Vickers, Y. Zhang, N. De Stefano, J.M. Brady, and P.M. Matthews. "Advances in functional and structural MR image analysis and implementation as FSL." NeuroImage. 23(S1), 2004, 208-219.

[15] P. Jezzard, P. M. Matthews, and S. M. Smith. "Functional MRI: An introduction to methods." England: Oxford University Press.

[16] Y. Zhang, M. Brady, and S. Smith. "Segmentation of brain MR images through a hidden Markov random field model and the expectation maximization algorithm." IEEE Trans. Med. Imag., January 2001, 20 (1), 45-57

[17] M. Jenkinson, P. R. Bannister, J. M. Brady, and S. M. Smith.. "Improved optimization for the robust and accurate linear registration and motion correction of brain images." NeuroImage, 2002, 17 (2), 825-841

[18] C.F. Beckmann and S.M. Smith. "Probabilistic independent component analysis for functional magnetic resonance imaging." IEEE Trans. on Medical Imaging, 23(2):137-152, 2004.

[19] J. Delgado, S. M. Sadjadi, H. Duran, M. Bright, and M. Adjouadi. "Performance prediction of weather forecasting software on multicore systems." Parallel and Distributed Processing Symposium (PDSEC), Atlanta, Georgia, April 2010.

[20] M. Ben-Yehuda. "The Xen Hypervisor and its IO Subsystem. "Presentation given at the 2005 IBM Systems and Storage Seminar. http://www.research.ibm.com/haifa/Workshops/systems-and-storage2005/papers/xen-io.pdf

[21] R. Zamani and A. Afsahi. "Communication Characteristics of Message-Passing Scientific and Engineering Applications." IASTED PDCS 2005: 644-649.

[22] P. Noordhuis, M. Heijkoop, and A. Lazovik. "Mining Twitter in the Cloud: A Case Study." In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD '10). IEEE Computer Society, Washington, DC, USA, 107-114. DOI=10.1109/CLOUD.2010.59

[23] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, Th. Fahringer, and D. Epema, "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," Cloud Computing: First International Conference, CloudComp 2009, Munich, Germany, October 19-21, 2009

[24] A. Tikotekar, G. Valle, T. Naughton, H. Ong, C. Engelmann, and S. L. Scott. "An Analysis of HPC Benchmarks in Virtual Machine Environments." In Euro-Par 2008 Workshops - Parallel Processing, 2008.

[25] T. Chen, M. Gunn, B. Simon, L. Carrington, and A. Snavely, "Metrics for ranking the performance of supercomputers," Cyberinfrastructure Technology Watch Journal: Special Issue on High Productivity Computer Systems, vol. 2. Feb. 2007.

[26] W. Smith , I. T. Foster, and V. E. Taylor, "Predicting application run times using historical information," Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1998, pp.122-142.

[27] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz. "A regression-based approach to scalability prediction." In Proceedings of the 22nd annual international conference on Supercomputing, 2008, pp. 368–377.

[28] G. Bernat, A. Colin, and S. M. Petters, "WCET Analysis of Probabilistic Hard Real-Time Systems," pp.279, 23rd IEEE Real-Time Systems Symposium (RTSS'02), 2002.