Experimental Study of Large-scale Computing on Virtualized Resources

Juan C. Martinez, Lixi Wang, Ming Zhao, and S. Masoud Sadjadi
School of Computing and Information Sciences, Florida International University
{jmart054, lwang007, zhaom, sadjadi}@cs.fiu.edu

ABSTRACT
Parallel applications have a pressing need for the utilization of more and more resources to meet users' performance expectations. Unfortunately, these resources are not necessarily available within one single domain. Grid computing provides a solution for scaling out from a single domain; however, it also brings another problem for some applications: resource heterogeneity. Since some applications require having homogeneous resources for their execution, virtualizing the resources is a noble and viable solution.

In this paper, we present two parallel applications, namely WRF and mpiBLAST and report the results of different runs scaling them out from 2 to 128 virtual nodes. Later, we analyze the effects of scaling out based on the application's communication behavior.

Categories and Subject Descriptors
D.1.3 Concurrent Programming: Distributed programming; Parallel programming.

General Terms
Experimentation, Performance

Keywords
Grid Computing, Performance Evaluation, Virtualization

1. INTRODUCTION
There are growing needs for large-scale computing, motivated by the emergence of grand challenge applications in science and engineering, as well as the massive growth of data available for analysis. The increasing adoption of programming paradigms such as the classic MPI and the recently popular Map-Reduce has provided simple yet powerful ways of massively parallel problem solving, generating more interests in large-scale computing as well as the need for systems that can support such computing.

Different types of large-scale distributed computing systems have been developed over the last decade. At one end of the spectrum are volunteer computing systems (e.g., [1][2]), which are an aggregation of a large number of unmanaged resources contributed by individual resource owners. At the other end are *grid computing systems* (e.g., [3]), which are built upon managed resources shared across organizations. Common to these systems are applications that are tightly coupled with their underlying middleware frameworks and they are directly executed on the hosting resources. There are certain limitations to such approaches. From the perspective of application users, existing applications have to be reengineered to use the APIs provided by the middleware in order to enable them on those computing systems so that they can make use of the available resources. In addition, these modified applications have to rely on the mechanisms provided by the host operating systems (OSs) to protect their execution from the other tasks that are sharing the same resources. From the perspective of resource owners, these approaches have only limited control over applications' resource usage and they also have to rely on the available OS mechanisms to protect the security of their resources.

This paper considers a new approach to building large-scale computing systems by virtualizing existing resources using *system virtual machine* (VM) technologies (e.g., VMware [3][4] and Xen [5]) to support flexible resource sharing with strong isolation and convenient application deployment on customized execution environments. VMs are becoming pervasively used, driven by the fast maturation and wide availability of VM products, as well as the rapid growth of computing power of modern computers. Their deployments can be found from enterprise datacenters for resource consolidation to personal computers for multi-OS hosting. In our proposed system, VMs can be dynamically deployed to facilitate the consolidation of applications and co-allocation of the available resources of existing computers both scattered across organizations and owned by individuals. As a result, resource-demanding applications can be distributed and executed along with the VMs in a massively parallel fashion.

In order to investigate the feasibility of building a large-scale virtualized computing system and to identify the potential research challenges, we have developed a large VM-based system consisting of more than 100 VMs hosted on 30+ shared existing physical servers at FIU. Two representative massively parallel applications are tested on this environment and an analysis on how the performance is affected is presented and analyzed accordingly taking into consideration the nature of each of application.

The rest of this paper is organized as follows. Section 2 introduces two representative MPI applications with different communication behavior. Section 3 discusses how we used virtual machines in our work. Section 4 presents the experiments. Section 5 offers a more in-depth discussion. Section 6 examines the related work. Finally, Section 7 concludes the paper.

2. Massively Parallel Applications
Massively parallel applications are typically highly resource-demanding and require large-scale resources to meet the expectation of their users. For this part of our discussion, without losing the generality, we focus on the Weather Research and Forecasting

VTDC'09, June 15, 2009, Barcelona, Spain.

(WRF) code [6] and mpiBLAST as two representative applications for high-performance computing: the former requires frequent communication and there are heavy data dependencies among its worker nodes (called *tightly-coupled communication application*), and the latter requires no communication among the worker nodes with little data dependency between the head and the worker nodes (called *loosely-coupled communication application*).

2.1 WRF

WRF is the latest numerical model developed by the National Center for Atmospheric Research (NCAR) for both operational forecasting and atmospheric research. A measure of its success is that WRF is being rapidly adopted by many meteorological services and researchers worldwide. Also, WRF was shown to significantly improve forecasts of hurricane structure and rainfall during the Florida Hurricanes of 2004 [7].

The impact of hurricanes is so devastating throughout different levels of society that there is a pressing need to provide a range of users with accurate and timely information that can enable effective planning for and response to potential hurricane landfalls. The current version of WRF has been designed to run either on a single machine (with one or multiple processors) or on a cluster of *homogeneous* nodes connected through a high-speed local area network. It has not been designed to scale on resources of heterogeneous nature that may become available during the course of a simulation process. However, the high resource requirements of WRF for fine-resolution forecasting (1km resolution forecast) demand a large number of computing nodes with substantial memory and disk storage. Currently, few organizations (even national agencies) have either the required computational power or bandwidth to produce high resolution forecasts and deliver them to emergency management, businesses, and the public. Therefore, there is a pressing need for large-scale resource enablement of the WRF code so that it can utilize resources available from willing organizations and individuals who want to contribute.

WRF is written in FORTRAN and C and is a typical MPI application. Like any other MPI application, it is very dependent on the architecture of the machine, the OS, and the libraries (including the MPI libraries) against which it is compiled. In addition, WRF is very sensitive to the homogeneity of the computing nodes of the cluster on which it is deployed. To enable WRF to execute on the large-computing resources provided by volunteer and grid computing systems, as discussed before, we need to modify the WRF code to interact with the API of their underlying middleware and compile it for all the possible heterogeneous resources. Unfortunately, even after overcoming this tedious and error prone task, WRF will not run properly on such environments as the heterogeneity in the hardware architectures of potential available resources will skew the result (e.g., some of the resources may be 64 bit while others are 32 bit) and the WRF performance may be degraded significantly too. In this paper, we use resource sharing via virtual machines (VMs) to provide a solution to such problems.

2.2 mpiBLAST

Another typical example from the biology area that has benefited from massively parallel computing is sequence database search. DNA (peptide) and amino-acid (nucleotide) sequences have been used to identify organisms or species. In bioinformatics research, in order to identify a newly discovered sequence, the key approach is to search for similarities between a query sequence and existing sequences against biology databases. The BLAST (Basic Local Alignment Search Tool) is a popular tool providing basic algorithms for sequence database search. Traditional implementations of BLAST such as NCBI-BLAST have been proven to be too slow and as a result failed to catch up with the speed of database growing.

mpiBLAST is an open-source parallel implementation of BLAST based on MPI [14]. One of the characteristics of mpiBLAST is database segmentation. By statically dividing large sequence database into small fragments and distributing those fragments to the nodes in a cluster, mpiBLAST enables simultaneous query over the set of fragments. MpiBLAST wraps the standard NCBI [15] formatting function to format raw sequence database into fragments and put them in a shared storage space. Extra disk I/O, from which traditional BLAST suffers when trying to fit the entire database into memory, is also avoided by the aggregate memory available from all the nodes in the cluster.

3. Using Virtual Machines

3.1 Resource Sharing via Virtual Machines

VM technologies provide a powerful layer of abstraction for resource sharing. The VMs considered in this paper are system-level VMs, which are based on the virtualization of entire physical hosts' resources, including CPU, memory, and I/O devices, presenting virtual resources to the guest operating systems and applications. Although the techniques proposed in this paper can also be applied to some of the other types of virtualization (e.g., OS-extension based VMs [7][8]), those are not the focus of this paper. System VMs include the following two types: full-virtualized VMs and para-virtualized VMs. Full-virtualized VMs (e.g., VMware ESX [4]) present the same hardware interface to guest OSs as the physical machines and thus support unmodified OSs in the VMs. Para-virtualized VMs (e.g., Xen [5]) present a modified hardware interface which is optimized to reduce the overhead of virtualization, but they require the guests OSs to be modified too in order to accommodate these changes.

System virtualization is implemented by the layer of software called virtual machine monitor (VMM, a.k.a. hypervisor). VMM can be either hosted on an existing OS or run directly on top of the hardware. Hosted VMs leverage the native OS to access resources and thus typically incurs more overhead, but they can be conveniently deployed on existing resources and transparently work with their OS installations. Examples include VMware Server on Windows and Linux [3], Parallels Desktop on Mac OS [9]). Non-hosted VMs require existing OSs to be removed so VMM can have direct control of the resources, but they can typically deliver better performance compared to hosted VMs. Examples of non-hosted VM products include Xen [5] and VMware ESX Server [4]. Therefore, non-hosted VMs are gradually gaining dominance in server virtualization environments, whereas hosted VMs are more widely used in systems where VMM needs to coexist with traditional OSs without disrupting the normal operation of those systems.

The emergence of system VMs is driven by the fast maturation and wide deployment of virtualization technologies, as well as the rapid growth of computing power on modern computer systems. On one hand, VM technologies are already efficient and reliable enough to host mission-critical applications, and they are widely available for

the virtualization of various types of systems. Many VM products are also free to use, for example, Xen and kVM are released under GPL free software license, as well as VMware Server and ESXi are free of charge. On the other hand, the ever increasing computing power of today's computers has provided the necessary resources to host VMs. In particular, multi-core and many-core CPUs are quickly emerging on not only high-end systems but also consumer products. VMs are particularly suited to provide space-sharing of resources for such systems. Driven by the above factors, system VMs are becoming increasingly more common within enterprises for server consolidations, as well as by end-users to run different OSs and applications on the same machine.

This paper considers the use of dedicated VMs to host different application instances and allow them to share the underlying physical resources. The goal is similar to the resource sharing provided by conventional multi-user, multi-programming OSs, but the multiplexing of applications to resources is provided at a lower level of the system abstraction. It thereby has the following advantages in supporting resource sharing.

Application and Resource Security: Because system virtualization multiplexes resources below conventional OSs, it can provide strong isolation between an application in a VM and other tasks that share the same host. For a malicious piece of code to comprise the resource, it has to break the protection provided by both the guest OS and VMM layers. It is also well recognized that because a VMM is a much thinner software layer that a typical OS, it is much easier to be implemented in a robust way without hidden security holes. Therefore, VMs can provide stronger security to both the resources and the tasks that are sharing the resources.

Failure and Performance Isolation: In addition to better security protection, VMs also provide strong failure isolation. A catastrophic failure happened inside a VM (e.g., OS crashes or file system corruptions) will not affect the normal executions of the other tasks outside of this VM. In terms of performance isolation for resource sharing applications, research has also shown that hosting applications with independent VMs can provide very good performance isolation [5]. In fact, in the presence of a misbehaving application, system VMs offer much better isolation of interference compared to OS-level resource sharing approaches [10].

Resource Allocation Flexibility: VMs can be used as resource containers to allow flexible resource allocation. Current VM technologies typically allow VMs to be created with desired amount of resources in terms of CPU numbers, memory size, and disk capacity. Server-class VM (e.g., VMware ESX Server, Xen) products also provide fine-grained support for dynamic adjustment of VMs' CPU and memory shares as well as limited support for dynamic allocation of I/O bandwidth. In addition, VMs also allow resource usage to be balanced across physical host boundary by migrating application workloads along with their VMs among the hosts.

Application Mobility: As a VM's CPU, memory, and disk state can be represented as data, it can be easily migrated across hosting resources by transferring its entire state among the hosts. The migration can be done by suspending the VM on the origin host, copying its entire state to the destination host, and resuming its execution on the destination. Modern VM technologies also allow VMs to be migrated while they are continuously executed, across a local area network [11][12]. VM migration allows optimization of application executions by migrating their VMs to resources that would better suit the application requirements.

Execution Environment Customizability: Hosting applications with *dedicated* VMs enables application-specific customization and fine-tuning of execution environments, including OSs and libraries, which are encapsulated within the VMs. In this way, VMs' application-specific customization allows the provision of application-desired execution environments. In contrast, a conventional OS has to support general-purpose usage for a variety of applications and is hence difficult to be customized to suit different needs.

Application Code Portability: VMs enable the seamless deployment of applications on heterogeneous resources. VMs abstract away the heterogeneity of physical resources and provide the basis for creating coherent environments for application executions. For example, a Linux application that requires a certain version of LibC can be easily deployed along with its VM even if the host OS is not Linux or has a different version of the library; similarly, an application binary compiled for a 64-bit system can also be transparently deployed along with its VM on a 32-bit host without any modifications.

3.2 Large-scale Computing on Virtualized Systems

In this paper, we use VMs as new building blocks for large-scale computing systems. For the application developers and users, the proposed system will enable them to conveniently deploy their applications on large numbers of existing resources and conduct massively parallel computing. For the resource owners, this system will also significantly improve the utilization and investment of their resources, while they are less prone to potential security issues.

Hosting large-scale applications on virtualized systems greatly facilitates the deployment of applications and enables them to conveniently leverage the aggregated resources. The enabling process will be as simple as installing it on a single computer. The application user will be given a plain VM (with the basic OS and libraries) to install the application along with the necessary special libraries and tools. Afterwards, the user submits this customized VM and the enabling of the application is completed. The management system will be responsible to create many instances of this VM on the hosting resources to start computing with the desired scale. The VMs will be instantiated on the hosting resources on demand, instead of being statically deployed, in order to support efficient resource multiplexing for dynamic application workloads. A computing session will be started with the instantiations of VMs to host the application's parallel processes, and it will be ended with the termination and cleanup of the instantiated VMs.

Despite all the benefits of resource virtualization for realizing large-scale resources, there are a number of challenging issues that still need to be addressed. For example, because VMM introduces an additional layer of software underneath conventional OSs, VM-based resource sharing generally has more overhead than OS-based resource sharing. Nonetheless, as VM technologies rapidly mature, their efficiency is also quickly improving. Modern VM technologies have demonstrated that their overhead is considerably small, particularly for CPU intensive workloads [3][5]. Being an active research field, substantial work is undergoing in both academia and industry to enhance various aspects of system virtualization. In particular, the emerging hardware CPU extensions (e.g., Intel VT

and AMD-V) are providing important support for CPU, memory, and I/O virtualization, and they have the potential to further improve the efficiency of system VMs.

4. EXPERIMENTAL ANALASIS

4.1 Setup

The setup established for our tests consisted of more than 100 VMs hosted on three set of physical resources distributed across three buildings in two campuses of FIU.

The first set of VMs (*GCB-VM*) is hosted on the compute cluster named GCB. It consists of 8 IBM nodes where each one has an Intel Pentium-4 3GHz CPU with hyper-threading and 1GB RAM, and runs CentOS 4.4 with kernel 2.6.9.

The second set of VMs (*MIND-VM*) is hosted on the compute cluster named MIND. It has 16 Dell PowerEdge 1850 server nodes, where each node has a Dual Intel Xeon 3.6GHz CPUs with hyperthreading and 2 GB RAM, and runs RHEL 4 with kernel 2.6.9.

The third set of VMs (*CS-VM*) is hosted on several physical servers in the Computer Science data center, including five Dell 2950 servers with dual quad-cores and 16 GB RAM per node, one Dell 2950 server with dual quad-cores and 32 GB RAM, four Dell R900 servers with four quad-cores and 128 GB RAM per node, and two Dell 2900 servers with dual quad-cores and 32 GB RAM per node.

For each of the above three set of resources, VMs are started from independent images stored on an NFS server running on one of their physical servers; an additional VM is also used to run the NFS server for each set of the VMs and to provide shared access to the application binaries and input/output data. Parallel applications are executed on the VMs with one parallel process per VM.

CS-VM and GCB-VM are located at different buildings of the FIU main campus, and the network latency between them is 1.625ms. MIND-VM is located at a different campus and its network latency to CS-VM and GCB-VM is 1.219ms and 1.733ms, respectively.

GCB-VM and MIND-VM are virtualized with VMware Server 1.0.8, whereas the CS-VM includes 111 VMs virtualized with Xen 3.0.0 and 16 VMs based on VMware Server 1.0.8. Every VM is configured with one CPU, 1GB RAM, and 4GB of disk. The VMs from CS-VM run paravirtualized UBUNTU Linux with 2.6.18 kernel or native UBUNTU with 2.6.15 kernel. The VMs on GCB and MIND run UBUNTU with 2.6.27 kernel.

4.2 Benchmarks

When running WRF, all the participating processes communicate to each other to exchange messages in a many-to-many communication scheme. For this reason the communication cost is a key factor in the WRF performance. Thus, WRF is considered a tightly coupled communication application. Therefore, the type of network connection from the infrastructure WRF is running becomes crucial and can determine the difference between a good performance or a bad one.

In contrast, the mpiBLAST while is highly data-intensive, at the same time, is an embarrassingly parallel application. Its parallel processes work in a typical master-worker manner. The master is responsible for job scheduling and result collection and the parallel search is done by the workers. Upon startup, the query sequences are first broadcasted to each worker. Workers then send a request to master for assignment of fragment. Fragments are assigned to different workers until one of the workers completes the search on that fragment and returns the result. Thus, one fragment may be assigned to more than one worker. However, the master keeps track of the fragments that a worker has on its local storage. The principle strategy for master to make a decision on assignment is that the worker would be given the fragment already on its local, if not, the fragment that existing on the smallest number of other workers. In this way, the worker could request for what it had to avoid get the same fragment in searching by other workers. The search process completes until all the fragments have been searched by workers.

4.3 WRF Experiments

The first group of experiments considers WRF with GridMPI as the MPI implementation. GridMPI [21] is an implementation of the MPI standard designed for high performance computing in the Grid. It establishes a synthesized computer cluster by binding multiple cluster computers distributed across different domains. Since GridMPI does not incur much overhead compared to MPICH, it was used for WRF executions on both single-domain and cross-domain resources. The GridMPI version considered is 2.1.1.

All runs of WRF were done three times per configuration and at the end, the average of these three was considered. The standard deviation found was approximately 5%. Our experiments were performed in three stages:

- Small-scale Tests: Comparison of WRF execution times from runs on physical/ virtual resources from GCB/ MIND.
- Large-scale Single-domain Tests: Analyze WRF performance on a large number of resources from CS-VM.
- Large-scale Cross-domain Tests: Analyze WRF performance on a large number of resources aggregated across GCB-VM, MIND-VM, and CS-VM.

Small-scale Tests: The first experiment was conducted both on the native physical clusters and on the VM-based clusters from GCB and MIND. Three different configurations were used for performance comparisons between the physical and virtual computing systems. The first one involved the executions on GCB resources only (*GCB-local-physical* vs. *GCB-local-VM*); the second one on MIND resources only (*MIND-local-physical* vs. *MIND-local-VM*), and finally on both GCB and MIND resources (*GCB-MIND-physical* vs. *GCB-MIND-VM*). For all these tests, at most one parallel process was assigned per single CPU core or thread.

From the results on Figure 1, we can see that the overhead of utilizing VMs instead of physical ones is not significant, ranging from 9.2% to 58%. However, because of WRF's communication scheme being tightly-coupled, the overhead of network virtualization is expected to have a considerable effect. In addition, when comparing the results from different number of VMs, it is evident that there is always a considerable speedup as more resources become available although the speedup drops after more than 8 processes are utilized. We believe the reason for this is due to the relatively small input data set size used by WRF in this set of experiments. Finally, when executing WRF across two distributed clusters, its performance drops significantly due to the heavy inter-process communication over the slower network. Nonetheless, the performance on the VMs still closely follows that of the physical ones [20]. These observations confirm that it is feasible to build a

large virtualized computing system and to deliver good performance to parallel applications.

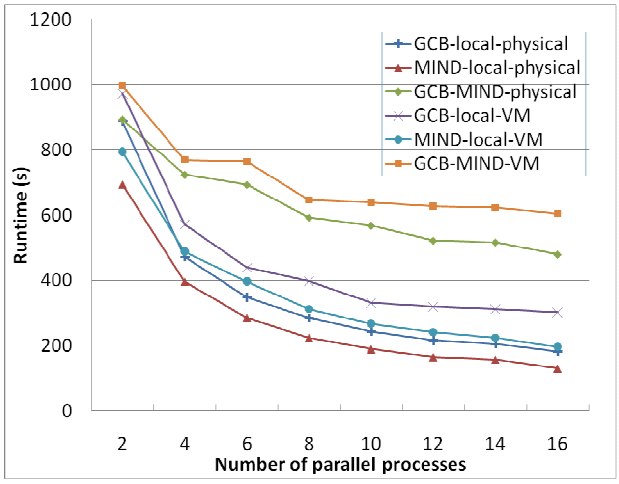

Figure 1- Local vs Distributed execution times for WRF

Large-scale Single-domain Tests: The second experiment involved running WRF on a large number of nodes from the CS-VM. This experiment, as well as the following one, considers a larger data set as the input for WRF as more resources are available for a larger-scale experiment.

Figure 2 shows the runtime of WRF as the number of VMs increases from 2 to 128. It is evident that the performance of WRF improves as more resources become available for its parallel execution (Note: the Y axis is on $\log_2$ scale). However, the speedup drops significantly as the number of VMs reaches beyond 32. The reason for this is that at this point the inter-communication overhead prevails over the computation time at each VM. The overhead from network I/O virtualization further aggravates the performance of WRF when a large number of VMs are involved. Furthermore, even though the data set considered in this experiment is larger than the one used in the previous experiments having an even larger data set may still show an improvement in the speedup when more than 32 VMs are utilized in the parallel execution of WRF.

As a reference, we can see our previous results obtained by Javier Delgado et al. [20] on the MareNostrum supercomputer from the Barcelona Supercomputing Center. The supercomputer consists of 2,560 JS21 blade computing nodes, each with 2 dual-core IBM 64-bit PowerPC 970MP processors running at 2.3 GHz for 10,240 CPUs in total and 20 TB of memory. The results from parallel executions of WRF with the same input data set on MareNostrum show a similar trend for speedup – it stops scaling beyond 32 nodes.

Large-scale Cross-domain Tests: The third experiment executes WRF across all the resources from GCB-VM, MIND-VM, and CS-VM with 8, 32, and 128 VMs, respectively. In this experiment, one VM per CPU core is enforced and thus GCB hosts only 8 VMs. WRF was executed with 32, 64, and 128 VMs respectively distributed in the three set of resources as follows:

- 32 VMs: 8 GCB-VM, 16 MIND-VM, 8 CS-VM
- 64 VMs: 8 GCB-VM, 32 MIND-VM, 24 CS-VM
- 128 VMs: 8 GCB-VM, 32 MIND-VM, 88 CS-VM

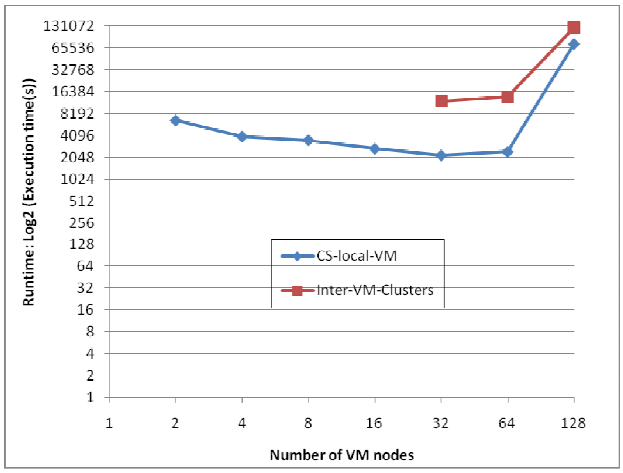

Figure 2 – Runtime of WRF on CS-local-VM vs Inter-VM-Clusters

The results on Figure 2 (see Inter-VM-Clusters) show that the execution times for WRF in this experiment are much larger than those in the previous single-domain experiment. This can be largely attributed to the fact that in this case we are dealing with much higher communication latencies across domains.

We can also observe that as more VMs (cores) are added, the performance degrades since the inter-communication increases and again, similar to the Large-scale Single-domain Tests, the computation time at each VM is not significant compared to the inter-communication overhead**.**

4.4 mpiBLAST Experiments

For the second group of experiments, we deployed mpiBLAST on virtual machine systems. We used MPICH2 for the MPI implementation and mpiBLAST 1.5-pio for the application. Although this version of mpiBLAST has some optimization based on parallel I/O, we didn't utilize this feature because the database to be searched was not on a parallel file system.

Our experiment was benchmarked on the nucleotide sequence database nt, which is frequently used by bioinformatics researchers. The size of the database is 8.4 GB after formatted and the size of the query which consists of 15 sequences is fixed at 10K. The result of the query is a file that is approximately 3MB.

We ran the application with different number of nodes to evaluate the scalability of mpiBLAST hosted on a virtualized infrastructure. Given p nodes for each run, the number of processes was configured as p+1. The nt database was statically partitioned into 127 fragments to have each worker query one fragment when 128 VM nodes were used.

Figure 3 shows the performance measurements of mpiBLAST on the CS-VM cluster. Database fragments were distributed among the workers by the first execution. The total execution time can be decomposed to three major components including the time spent on fragments copying, searching and result merging. For the first run,

the execution time ranged from 1200 to 7600 seconds, dominated by the copying time. We collected the total execution time of the 3 best runs with a different number of VM nodes. It costs over 1 hour for 2 nodes (1 worker and 1 master), whereas less than 30 seconds for 128 nodes (1 master and 127 workers). MpiBLAST achieved a super-linear speedup on the total execution time starting from the 8-node run compared with the base case, which consists of just two nodes. This is because, as more VMs are involved in the parallel computing, the aggregated memory capacity from them grows, enabling more fragments to be fit in the memory with less disk access overhead. However, the performance dropped at the point of 128-node, which was expected to have the best speedup. The reason for this is that the non-searching time, such as the overhead of inter-communication between VM nodes, became dominant over the entire execution. Figure 4 shows the average search time for 8, 32 and 128 nodes. Excluding the non-searching overheard, the performance for node 128 can achieve a linear speedup compared with the base case of running on 8-node.

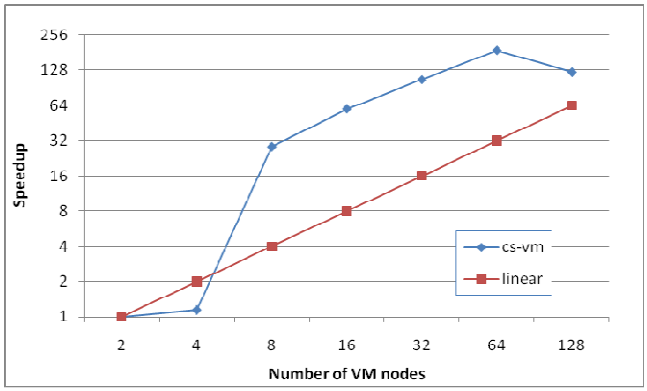

Figure 3 – mpiBLAST on CS-VM

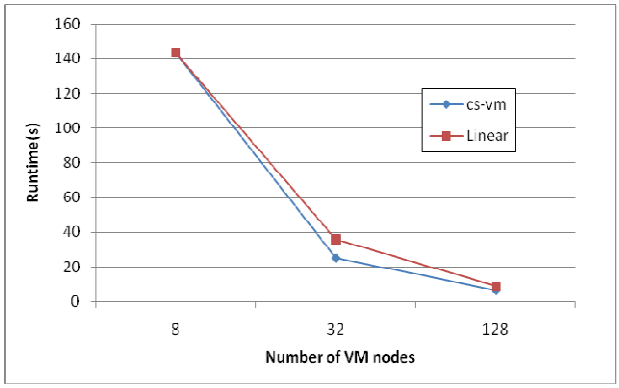

Figure 4 – Average Search Time

5. DISCUSSION

Based on the results from both WRF and mpiBLAST, two representative MPI applications with distinct characteristics, this section offers a further discussion on the feasibility and challenges of VM-based large-scale computing systems. First of all, we could realize that the scalability for VM had the same trend as the one for physical ones. By analyzing WRF, we could see that there is speedup after adding more VMs, even though there is an overhead caused by the use of virtual resources over physical ones. This proves that the overhead from virtualization is acceptable to building a large-scale computing system and it is justifiable as the use of virtualization enables us to more effectively harness the available resources and aggregate them for computing needs.

The results from WRF also show that resources cannot be blindly selected regardless of the parallel application's characteristics. For tightly-coupled communication applications, the cost from communication can be prohibitive as the computing system scales across domains, since in such situation there is an additional cost of going across domains over a much slower network (virtual network). We could see that as we include more than 32 VMs from three different domains the performance did not obtain any speedup, which mainly was caused because of a higher communication delay and the computation time was overwhelmed by the communication time.

As we have observed, WRF's limitation for scaling up is due to its tightly coupled communication scheme; however, when dealing with loosely coupled applications, the scalability on the number of VMs can provide a much more meaningful performance improvement. For example, mpiBLAST as discussed in Section 3.3 presents a speedup even larger than the linear one. The reason for this is because the scaling out on the number of VMs offers a higher level of parallel computation and the communication among them is not significant. Nevertheless, as demonstrated by the execution time of the first mpiBLAST run, for data-intensive applications, the distribution of data set to a large number of processes running on VMs can still be a bottleneck of the system's scalability.

Additionally, it is important to point out that our results for WRF on a VM infrastructure present a similar behavior to the ones obtained when deployed on a physical resource [20].

6. RELATED WORK

Different types of large-scale distributed computing systems have been developed over the last decade, including typical volunteer computing systems [1][2] and grid computing systems [3]. Common to these systems are applications that are tightly coupled with some underlying middleware frameworks and are directly executed on the hosting resources. In comparison, this paper considers building large-scale computing systems based on VM-based resource virtualization, which offers highly flexible, customizable resource sharing with strong security and isolation.

As the deployment of virtualization technologies become pervasive, VM-based computing systems have been considered at different scales, from virtualized data centers [16] to VM-based grid computing [17], VM-based high-performance computing [18], and VM-based cloud computing systems [18]. The key differentiator of this paper is in its focus on large-scale virtualized computing systems for massively parallel applications.

7. CONCLUSIONS AND FUTURE WORK

Across this paper, we have been able to observe how virtualizing resources allowed us to scale out on a large set of resources for two parallel applications which require homogenous resources. Nevertheless, the performance gain varied among both applications, mainly because of their communication requirements.

For tightly coupled communication applications such as WRF, we can see that the cost of the communication was key in the execution time. Therefore, even though, the overhead of using VMs locally did not show much performance loss, when going across domains, the

communication cost became critical. Similarly when scaling out with a large number of VMs, the intercommunication increased and affected the performance dramatically.

On the other hand, for loosely coupled communication applications such as mpiBLAST, we can always see a considerable speedup as here what prevails is the computation gained by an increased parallelism over the communication overhead. Nevertheless, if the dataset is relatively small and the number of VMs utilized is too large, then the parallelism can be overwhelmed by the communication.

Based on the findings of this paper, we will address the research challenges of building large-scale virtualized computing systems from the following three aspects: data management for efficient VM images and application data provisioning; resource management for precise resource control and optimized resource allocation; and job management for scalable application executions. At the same time, a larger input data size will be created for these experiments. Finally, we are planning on taking into consideration communication issues (e.g. network latency) in Javier Delgado's model.

Acknowledgement: This work was supported in part by IBM and the National Science Foundation (grants OISE-0730065, OCI-0636031, HRD-0833093).

REFERENCES

[1] David P. Anderson, et al., "SETI@home: An Experiment in Public-Resource Computing", Communications of the ACM, Vol. 45 No. 11, November 2002, pp. 56-61.

[2] David P. Anderson, "BOINC: A System for Public-Resource Computing and Storage", in Proc. 5th IEEE/ACM International Workshop on Grid Computing, November 2004.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Int. J. High Perform. Comput. Appl., vol. 15, pp. 200-222, August 2001. J. Sugerman, G. Venkitachalam, and B. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", in Proc. of 2001 USENIX Annual Technical Conference, June 2001.

[4] Carl A. Waldspurger, "Memory resource management in VMware ESX server", Proceedings of the 5th symposium on Operating systems design and implementation, December 2002.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. "Xen and the Art of Virtualization", in Proc. of the ACM Symposium on Operating Systems Principles (SOSP), October 2003.

[6] Weather Research & Forecasting Model, http://www.wrf-model.org/

[7] Patrick T. Welsh and Peter Bogenschutz, "Weather Research and Forecast (WRF) Model: Precipitation Prognostics from the WRF Model during Recent Tropical Cyclones", 59th Interdepartmental Hurricane Conference, March 7-11, 2005 Jacksonville, FL. Linux Vserver, http://linux-vserver.org/.

[8] OpenVZ, http://wiki.openvz.org/.

[9] Parallels, http:// http://www.parallels.com/.

[10] J. Matthews, et al., "Quantifying the performance isolation properties of virtualization systems", in Proceedings of the 2007 workshop on Experimental computer science, 2007.

[11] VMware VMotion, http://www.vmware.com/products/vi/vc/vmotion.html.

[12] C. Clark, "Live Migration of Virtual Machines", in Proc. of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005.

[13] C. P. Sapuntzakis, R. Chandra, B. Pfa_, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers", in Proc. the 5th Symposium on Operating Systems Design and Implementation, December 2002.

[14] A. Darling, L. Carey, and W. Feng, "The Design, Implementation, and Evaluation of mpiBLAST", 4th International Conference on Linux Clusters: The HPC Revolution 2003 in conjunction with ClusterWorld Conference & Expo, June 2003.

[15] NCBI, http://www.ncbi.nlm.nih.gov/

[16] VMware Inc., VMware VirtualCenter Users Manual.

[17] R. Figueiredo, P. Dinda, , and J. Fortes, "A case for grid computing on virtual machines", In Proceedings of the 23rd IEEE Conference on Distributed Computing (ICDCS 2003 (May 2003), pp. 550–559.

[18] W. Huang, et al., "A case for high performance computing with virtual machines", Proceedings of the 20th annual international conference on Supercomputing.

[19] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/

[20] Javier Delgado, Marlon Bright, Javier Figueroa, and S. Masoud Sadjadi. Modeling WRF Execution time on MareNostrum. Technical Report FIU-SCIS-2009-02-01, Florida International University, Feb. 2009.

[21] William L. George, John G. Hagedorn, Judith E. Devaney, "IMPI: Making MPI interoperable", Journal of Research of the National Institute of Standards and Technology 2000.