Using Lightweight Virtual Machines to Run High Performance Computing Applications: The Case of the Weather Research and Forecasting Model

Hector A. Duran-Limon, Luis A. Silva-Bañuelos,
Victor H. Tellez-Valdez
Information Systems Department
CUCEA, University of Guadalajara
Zapopan, Jalisco, México
hduran@cucea.udg.mx, adansilva05@hotmail.com,
linux9995@gmail.com

Nikos Parlavantzas
INSA, INRIA, IRISA, UMR 6074
F-35708 Rennes, France
Nikos.Parlavantzas@irisa.fr

Ming Zhao
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
ming@cs.fiu.edu

Abstract— **There are many scientific applications that have high performance computing demands. Such demands are traditionally supported by cluster- or Grid-based systems. Cloud computing, which has experienced a tremendous growth, emerged as an approach to provide on-demand access to computing resources. The cloud computing paradigm offers a number of advantages over other distributed platforms. For example, the access to resources is flexible and cost-effective since it is not necessary to invest a large amount of money on a computing infrastructure nor pay salaries for maintenance functions. Therefore, the possibility of using cloud computing for running high performance computing applications is attractive. However, it has been shown elsewhere that current cloud computing platforms are not suitable for running this kind of applications since the performance offered is very poor. The reason is mainly the overhead from virtualisation which is extensively used by most cloud computing platforms as a means to optimise resource usage. In this paper, we present a lightweight virtualisation approach applied to WRF, a challenging communication-intensive, high performance computing application. Our experimental results show that lightweight virtualisation imposes about 5% overhead and it substantially outperforms traditional heavy-weight virtualisation such as VMware.**

Keywords- virtualisation; lightweight virtual machines; cloud computing; high performance computing

I. INTRODUCTION

High performance computing (HPC) applications expand several scientific fields such as meteorology, astronomy, chemistry, and bioinformatics among others. Examples of these applications include weather forecasting, materials science, quantum chemistry calculations, accelerator modelling, and astrophysical simulations [1]. These kinds of applications have huge demands of computing resources, which are traditionally supported by cluster- or Grid-based systems. Cloud computing, which has experienced a tremendous growth recently, emerged as an approach to provide on-demand access to computing resources. The cloud computing paradigm offers a number of advantages over other distributed platforms. For example, the access to resources is flexible and cost-effective since it is not necessary to invest a large amount of money on computing infrastructure nor pay salaries for maintenance functions. Therefore, the possibility of using cloud computing for running high performance computing applications is attractive. However, it has been shown elsewhere [2-6, 14-17] that current cloud computing platforms are not suitable for running this kind of applications since the performance offered is very poor. The reason is most cloud computing platforms make extensive use of virtualisation as a means to optimise resource usage. Virtualisation faces two core challenges: it does not offer quality of service (QoS) guarantees and has a performance penalty that is unsuitable for certain kinds of applications [9].

In this paper, we present an application-level virtualisation framework, called *Distributed Virtual Task Machine (D-VTM)*, which can significantly increase performance while retaining the benefits of cloud computing in terms of ease of management, cost reductions, resource usage optimisation, etc.

The main focus here is on applying the framework to the Weather Research and Forecasting (WRF) model [10], a challenging communication-intensive, high performance computing application. This application serves as our first attempt to evaluate our approach. We rely on our previous work on lightweight virtualisation in middleware [11], which focused on serial computations (e.g. transcoding a video file to MP4). We have extended this work here whereby lightweight virtualisation is applied to distributed computations (e.g. MPI applications), hence, making it suitable for HPC environments. Our focus is on developing

an open-ended framework for lightweight virtualisation in HPC environments, rather than being tied to any particular virtual machine (VM) implementation. We present a particular implementation of the framework on Linux; however, this implementation is only presented as a means to evaluate the framework. Our experimental results show that our lightweight virtualisation imposes about 5% overhead, hence, making our approach suitable to run WRF in a cloud computing environment. The results also show that D-VTM substantially outperforms traditional heavy-weight virtualisation such as VMware.

The paper is structured as follows. Section II introduces the motivation behind our work. Section III presents some related work. Our virtualisation framework is presented in section IV. The experimental results are shown in section V. Finally, some concluding remarks are given in section VI.

II. Application and Motivation

We approach the problem of mitigating natural disasters, which involve hurricanes, forest fires, and air pollution. For example, a number of hurricanes have had a devastating impact in the U. S. as well as in the north and south part of Mexico. Therefore, a better prediction capacity is required to generate better evacuation plans when this type of disasters happens.

The Weather Research and Forecasting (WRF) model [10] is gaining acceptance worldwide as one of the best models to carry out weather prediction. H-WRF [12] is an extension of WRF to support the prediction of hurricanes. Importantly, WRF can also be used to predict the evolution of forest fires as well as to determine and predict air pollution. However, the numeric model used by WRF demands a large amount of CPU power. Such demand can be increased dramatically if WRF is used to model a big geographical area with a high-resolution level (for example <1 km). Satisfying the high computational demands of WRF requires putting together large numbers of computing resources through infrastructures such as clusters, grids, or clouds.

Currently, Mexico does not possess a user-friendly system whereby Mexican meteorologists have open and remote access to perform on demand runs of WRF. In the best case, some meteorologists have access to a local computer infrastructure where on demand runs can only be carried out with the assistance of highly trained computer engineers. Some other meteorologists do not even have access to enough computing power to run WRF in a reasonable amount of time. Furthermore, to the best of our knowledge there is not a system, worldwide, able to offer on demand runs of WRF in cloud environments.

Within the DiViNE (Disaster mItigation on VIrtualised eNvironmEnts) project [29] (which is part of the LA Grid initiative [13]) we are developing a WRF Web portal able to carry out on demand runs of WRF on cloud computing environments. One of the main research challenges we are addressing involves using lightweight virtualisation as a means to make cloud computing environments suitable for running high performances applications such as WRF.

III. Related Work

Much research work has focused on evaluating the suitability of virtualised environments for executing HPC applications. Jackson et al. [2] and Ekanayake et al. [14] presented comprehensive studies of the performance of parallel applications on the Amazon EC2 cloud platform and a Eucalyptus cloud system, respectively, using a set of typical HPC applications. Related cloud performance studies also considered several specific HPC applications including climate modeling [15], bioinformatics [16], lattice optimisation [17], and astronomy workflows [27]. These results all indicate that cloud execution incurs a significant performance overhead, particularly high for communication-intensive applications. Wang et al. [3] concentrate on the networking performance of Amazon EC2 and reports that the underlying Xen-based virtualisation approach causes significant throughput instability and delay variations. An experimental study of running WRF on large-scale virtualised environments is presented in [6]. While the overhead of using VMs instead of physical machines is relatively low for single-cluster environments, the overhead becomes prohibitive when using multiple clusters.

Addressing the performance limitations of virtualised environments is currently receiving considerable attention. Huang et al. [5] and Raj et al. [18] proposed mechanisms to reduce the overhead of network I/O virtualisation by exploiting the self-virtualisation capability of modern network devices. Amazon has recently introduced a new cloud service, EC2 Cluster Compute instances, specifically targeting HPC applications. Early benchmark results are positive but still indicate performance degradation compared to conventional clusters when executing large-scale parallel MPI applications [4].

An interesting approach to address the performance limitations of virtualised environments is to apply alternative virtualisation approaches. Performance studies using general-purpose benchmarks show that traditional, heavyweight virtualisation approaches, such as Xen [19] and VMware [20], incur a performance penalty from 10-20% compared to native OS environments [21][22]. Even worse, this penalty is higher when there is a huge amount of IO operations. Moreover, heavyweight VMs do not offer response time guarantees [9]. Operating system-level VMs [23], [24] are an alternative to obtain a performance near to the native operating system. However, the main drawback of this approach is that the kernel needs to be recompiled; in our work, we adopt application-level virtualisation instead, which exploits the existing lightweight virtualisation facilities provided by the operating system. Lange et al. [25] proposed a low overhead virtualisation solution for supercomputing by embedding the virtual machine monitor inside of a lightweight kernel. However, this solution relies on the existence the lightweight kernel which supports only limited applications tailored for supercomputing. In contrast,

our lightweight virtual machine framework does not impose such kind of assumptions.

Other related work is exploiting the characteristics of virtualised environments to improve the execution of HPC applications. Henzinger et al. [7] presented a cloud-based job execution environment that uses static scheduling techniques to provide a simple, flexible pricing model for users. Lee et al. [8] proposed a job scheduling scheme that exploits the resource heterogeneity in clouds to simultaneously provide high performance and fairness. Finally, Vecchiola et al. [28] present a cloud platform with flexible scheduling policies, allowing HPC applications to exploit the on-demand elasticity capabilities provided by clouds. Such scheduling approaches and mechanisms are orthogonal to the lightweight virtualisation problem studied in this paper and can be combined with our proposed framework.

IV. The Lightweight Virtual Machine Framework for HPC Environments

The D-VTM framework is based on application-level virtualisation. This kind of virtualisation is built on top of the operating system services. Hence, the same operating system image is used for all virtual machine (VM) instances. As mentioned earlier, the D-VTM framework relies on the VTM framework [11], our earlier work on lightweight virtualisation. Briefly, the VTM framework includes three main elements, namely *resource factories*, *resource managers,* and *abstract resources*. Factories are in charge of creating resources whereas managers are responsible for managing them. Abstract resources explicitly represent system resources. There may be various levels of abstraction in which higher-level resources are constructed on top of lower-level resources. At the lowest-level are the physical resources such as CPU, memory, and network resources. Higher abstraction levels then include the representation of more abstract resources such as virtual memory, team of threads, network connections and more generic type of resources. Top-level abstractions are *virtual task machines (VTMs)*, which encompass lower level resources (e.g. processes and memory) and provide the execution environment of serial computations (e.g. transcoding a video file to MP4).

In this paper, we have extended the concept of VTMs to composite and distributed VTMs. A *D-VTM* provides the execution environment of a distributed computation (e.g. a WRF run) involving multiple processes running in parallel on different nodes. In addition, the framework was also extended with a number of VM management functions, as shown in Fig. 1. The *Job Scheduler* is in charge of receiving job requests accompanied with the desired number of VM instances. The *VTM Scheduler* keeps track of resource usage and determines if the required resources are available. Also, in case of resource contention, the amount of resources given to the D-VTMs can be reallocated by the VTM Scheduler. When the requested resources are available, a resource reservation is performed and the Job Scheduler asks the *VTM Factory* to create instances of D-VTMs. Hence, the life-cycle of D-VTMs (i.e. creation and deletion of D-VTMs) is controlled by the VTM Factory. Finally, the *Job Monitor* is in charge of monitoring task execution on the D-VTMs. In case a task fails (e.g. the task halts or has an abnormal exit), the Job Monitor informs the Job Scheduler about this situation, which in turns ask the VTM Scheduler to stop the execution of the D-VTM associated with the failing task. Afterwards, the VTM Scheduler releases the resources of the D-VTM involved. A notification is sent to the user of such a failure. The Job Monitor also keeps track of resource usage for billing purposes. Further details of each of these modules are presented below.

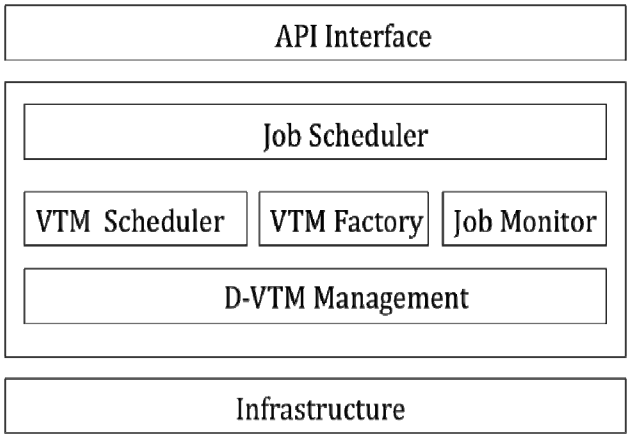

Figure 1. Overall architecture of the D-VTM Framework

A. Distributed VTMs

A D-VTM represents a virtual machine encapsulating a pool of resources allocated for the execution of its associated tasks. The UML model in Fig. 2 illustrates details concerning the relationship between tasks and D-VTMs. C*omposite tasks* are constituted by a number of independent sub-tasks, which can be executed in parallel. Sub-tasks are not further partitioned and are called *primitive tasks*. In addition, primitive tasks are only related to one address space. However, *distributed tasks* involve two or more address spaces. It should be noted that a composite task is not necessarily distributed.

Similarly, VTMs not containing other VTMs as lower-level resources are named *primitive VTMs*. There is a one-to-one relationship between primitive VTMs and primitive tasks. A composite VTM may encompass various local primitive VTMs. In this case, the primitive VTMs represent the virtual cores of a virtual machine (i.e. a composite VTM). Importantly, a distributed VTM is a specialisation of composite VTMs whereby two or more address space are involved. Hence, a distributed VTM includes various local and remote primitive VTMs. Finally, the VTM scheduler and VTM factory are defined on a per-address space basis.

The operation for running a job is the following:

```
run(program_name, dir_input_data,
input_files, dir_output_data, output_files)
```

This operation starts the execution of the given program, which involves multiple parallel processes. The process IDs

(pids) of such processes are evenly attached to the virtual machines contained within the D-VTM.

We used Control Groups (Cgroups) [26] to implement D-VTMs. Cgroups is a generic process-grouping framework, which is part of the Linux kernel since version 2.6.24. The amount of resources used by processes can be controlled per group of processes. A portion of the resources can be allocated to each group; for instance, a portion of the CPU can be assigned to each group. The processes belonging to a group share the group's resources and their resource usage is limited to the resources owned by the group. In our prototype, a VM instance (i.e. a primitive VTM or composite VTM) has allocated 50% of a CPU core. Nevertheless, a different share can be configured. Hence, two VMs are allocated per physical core. Also, primitive VTMs run a single process whereas composite VTMs run one process per virtual core.

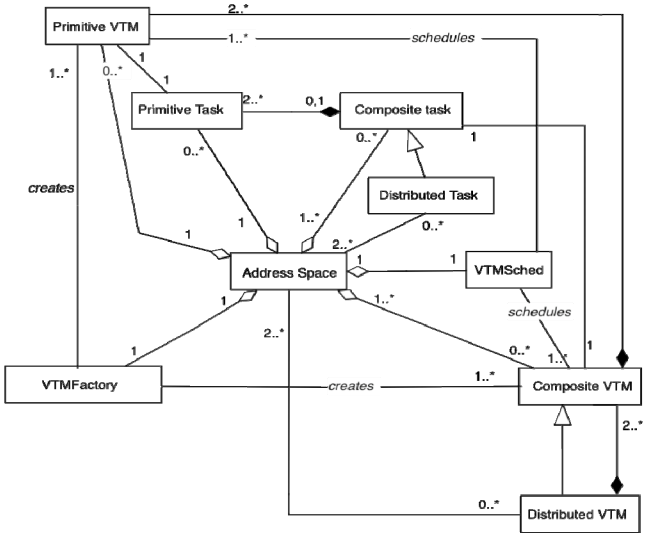

Figure 2. UML Diagram of Relationships between VTMs and Tasks

Virtual cores are implemented as sub-groups, i.e. groups belonging to a group. In this case, the resources used by the sub-groups are constrained by the amount of resources allocated to the group they belong to. Therefore, a D-VTM involves multiple virtual machines running across different nodes. Each virtual machine may be either uni-core or multi-core. Our prototype was implemented in C++ and it currently has control over CPU resources.

B. The Job Scheduler

Job requests are attended by the Job Scheduler. Clients are able to send requests by invoking the `schedule()` operation. A number of parameters are involved in a request, which mainly specify the virtual execution environment and the data related to the program to be executed. The operation for scheduling a job is the following:

```
schedule(no_vms, no_vcores, program_name,
dir_input_data, input_files,
dir_output_data, output_files, job_id)
```

The parameters that specify the virtual execution environment are the former two and define the desired number of virtual machines and the number of virtual cores per virtual machine, respectively. The program related data involves the program name, the directory where the input data is placed and the input file names. In addition, the directory of the output data and output file names are provided. Upon successfully execution of the `schedule()` operation, the job ID is obtained as an output parameter.

When a job request is received, the Job Scheduler executes the script shown below. First, the Job Scheduler asks the VMT Scheduler to perform an admission test, whereby it is determined if the requested virtual machines are available. In case the admission test is successful (line 1), the VTM Scheduler performs a resource reservation (line 2) and the scheduling parameters are obtained, which involve the specific physical cores that will be used as well as the CPU share given to each virtual machine (line 3). This schedule information is used by the VMT Factory to create the requested virtual machines (line 4). As a result, a job ID and a D-VTM are obtained. Afterwards, the D-VTM performs the `run()` operation, which is in charge of starting the execution of the requested program within the created virtual machine environment (line 5).

```
1. if ( vtmSched.admit(no_vms) )
2.     vtmSched.reserve(no_vms)
3.     vtmSched.schedule(no_vms, no_vcores,
schedParam)
4.     vtmFactory.newResource(schedParam,
job_id, d_vtm)
5.     error =d_vtm.run(program_name,
dir_input_data, input_files,
dir_output_data, output_files)
6.     if error
7.           return error
8. else
9.     return error
```

C. The VTM Scheduler

The `admit()` operation examines whether there are enough physical resources to satisfy the demands of the number of virtual machines requested. The operation returns true in a successful case, otherwise returns false. Resource reservation is achieved by the operation `reserve()`, which updates a register of both allocated and unused resources. Conversely, once the execution of a job has finished, the resources (i.e. the VMs) used can be released by the `release()` operation. This operation updates the register of unused resources. The current implementation keeps track of physical cores that are available as well as the information of which physical cores are using each running job.

The `schedule()` operation receives as input the number of virtual machines that will be created and the number of virtual cores for each virtual machine. This operation

produces as output the scheduling parameters that will be used to run the virtual machines in terms of the physical virtual cores assigned to each virtual machine as well as the CPU share for each virtual machine.

Jobs can be finished by the operation `finish()`. This operation can be used, for example, when a job has overcome the maximum allowed execution time. The implementation of this operation basically kills all the processes belonging to the job and invokes the `release()` operation to release the resources used by the job.

D. The VTM Factory

The VTM Factory is responsible for creating D-VTMs. The following operation is used for this purpose.

```
newResource(schedParam, job_id, d_vtm)
```

The `schedParam` argument defines the specific physical cores that will be used as well as the assigned CPU share for the virtual machines. This argument involves a two-dimensional array of integers m_{ij} in which *i* is number of virtual machines that will be created. m_{j0} represents the core number where the virtual machine will be allocated, m_{j1} involves the share value for each virtual machine, and m_{j2} represents the number of virtual cores that the associated virtual machine will include. The number of virtual cores can be either 0 or a multiple of 2. As a result of the operation invocation, a job id and a D-VTM are obtained.

E. The Job Monitor

The Job Monitor is in charge of monitoring the execution of jobs. In case a job finishes its execution, the monitor detects and informs of this situation to the Job Scheduler, which in turns asks the VTM Scheduler to release the resources allocated to the job. The Monitor also detects when a job has overcome the maximum execution time allowed and informs the Job Scheduler of such a situation. In this case, the Job Scheduler asks the VTM Scheduler to stop the execution of the job and release the resources associated with the job. Other operations supported by the Job Monitor are the following.

```
getStatus(job_id, status)
getExecutedTime(job_id, exec_time)
```

The former reports the status of a job, which can be running, suspended, or finished. The latter retrieves the execution time that a job has consumed.

V. Evaluation

We present a performance evaluation of the C++ prototype and compare it with VMware whose performance is similar to Xen's [21][22]. Our experiments are run in a private cloud setup based on the *Nadimit* cluster located at CUCEA, University of Guadalajara. The cluster contains 48 cores, which involves 6 nodes where each node includes two quad-core Xeon 5500 2.0 GH and 12 GB of main memory. The cluster runs Linux Centos 5.5, kernel 2.6.38 and uses Intel MPI 3.2.2, which is based on MPICH2. All the jobs considered in the experimental scenarios involve executing an independent run of WRF 3.2.1. In the case of Job 1, we used a 7.5 Km by 7.5 Km domain decomposition with 20 km resolution, which contains 120 x 70 grid points; and a simulation time of 24 hours. A larger simulation time was used in the rest of the jobs to make sure Job 1 finishes its execution before the other jobs. All runs of Job 1 were performed three times and the average of these runs was considered. We found a standard deviation of about 5%. We are using VMware ESX 4.1. VMware VMs are configured with one CPU, 1 GB RAM, and 7 GB of disk. These VMs run paravirtualised Linux Centos 5.6 with kernel 2.6.18.

All lightweight VMs and VMware VMs run a single process and have a 50% share of a physical core. In addition, each of the VMs belonging to the same job run on a different processor in the case of scenario 1 and scenario 3. Finally, in the context of this work the term 'overhead' is used to mean the ratio whereby a virtualisation approach is slower than the native approach. The overhead is calculated as follows: `overhead = ((100*`e_i`)/`e_j`) - 100`, where e_i is the execution time of a virtualisation approach and e_j is the native execution time.

A. Experimental Results

The first experimental scenario starts running 5 jobs namely, Job 2 to Job 6, in which each job runs 6 lightweight VMs. The execution time of Job 1 is measured for a different number of VMs, as shown in Fig. 3. The same scenario is repeated for the case of using VMware VMs. Finally, this scenario is repeated on the native Linux, i.e. without the use of any virtualisation. In this case, Job 2 to Job 6 run 6 processes, each one on a different processor. Afterwards, Job 1 increases the number of the processes it runs. This is equivalent to running the VMs in terms of the number of processes run. The lightweight VM approach obtains a much better performance than VMware VMs, as shown in Fig. 3. Only about 5% overhead is imposed by the former whereas the latter has an overhead that ranges from 17% to 223%.

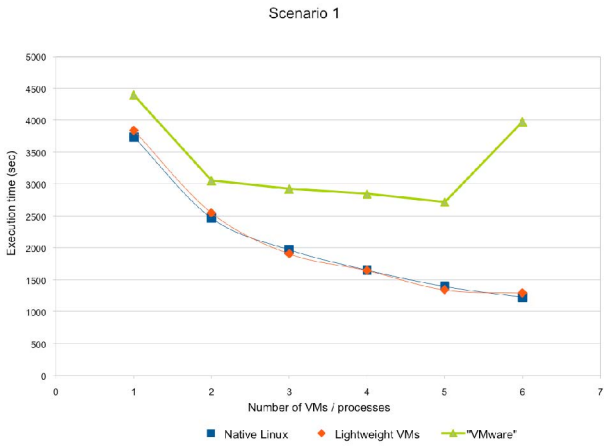

Figure 3. The execution time of Job 1 while Job 2 to Job 6 are executing.

The second experimental scenario is similar to the first scenario. The difference is that the measurements are taken on Job 1 while no other jobs are executing. Also, in the case of VMware, VMs are given 100% of the physical core whereas lightweight VMs receive only 50% of the physical core. In Fig. 4 we can observe that VMware VMs start with a better performance than the lightweight VMs. This result is expected as VMware instances are given more CPU resources. Despite of the fact that the lightweight VMs are in this disadvantageous condition, they outperform the VMware instances when 16 VMs are reached.

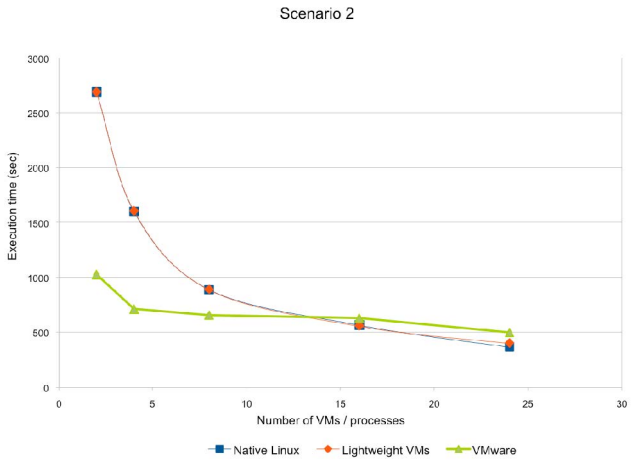

Figure 4. The execution time of Job 1 while no other jobs are executing.

Finally, the third scenario is also similar to the first scenario. The difference is that the VMs of Job 2 to Job 6 have 4 processes as a means to stress the use of CPU, while Job 1 keeps running a single process. The performance of VMware VMs is clearly degraded as the number of VMs increases, as shown in Fig. 5.

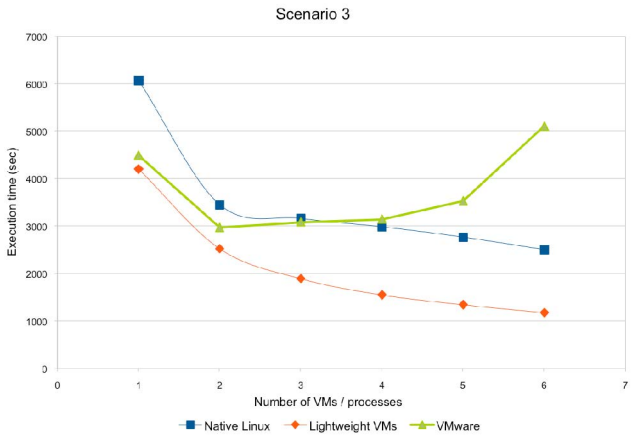

Figure 5. The execution of Job 1 while the other jobs stress the CPU

It should be noted that the lightweight VMs obtain a better performance than the native Linux. The reason is the latter does not have any control on the amount of CPU resources given to Job 1 while the CPU is stressed by other processes. In contrast, the lightweight VMs isolate the amount of resources used by Job 1. Furthermore, the lightweight approach maintains a fairly similar level of resource isolation even in overload conditions. This assertion is corroborated in Fig. 6 where scenario 1 and scenario 3 are compared in the case of lightweight VMs. In contrast, the performance of VMware instances gets worse when the CPU is overloaded, as shown in Fig. 7.

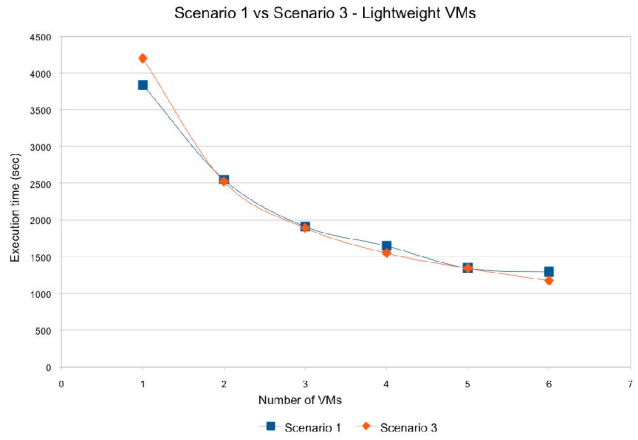

Figure 6. Comparing the performance of lightweight VMs in scenario 1 and scenario 3

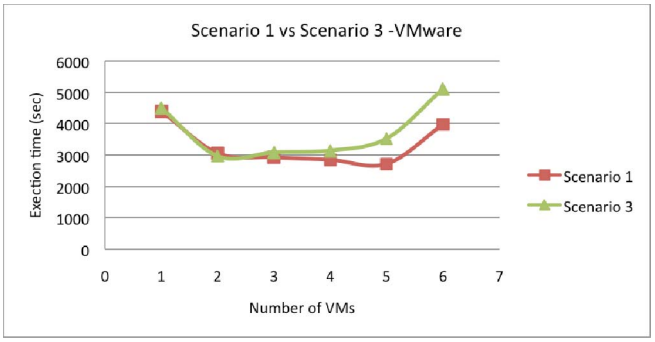

Figure 7. Comparing the performance of VMware VMs in scenario 1 and scenario 3

B. *Discussion*

Lightweight VMs have shown to be a suitable mechanism to run WRF efficiently while retaining the benefits of cloud computing such as resource usage optimisation. The total overhead imposed by the lightweight VM approach is about 5%. In contrast, VMware VMs have a much higher overhead that go up to 223%. It is expected that in larger-scale scenarios the overhead of VMware instances worsens since this is the trend shown in Fig. 7.

The performance of VMware VMs degrades rapidly when several jobs are sharing the same resources whereas the lightweight VM approach maintains a similar performance for different loads. This can be observed when comparing the first and third scenarios in Fig. 6 and Fig 7. The rapid degradation of VMware instances is due to the fact that this kind of virtualisation is negatively impacted by communication-intensive applications such as WRF.

VMware instances behave better when the resources used are not shared by other jobs, as shown in Fig 4. Therefore, the hardware cannot be shared by other applications in order to optimise the performance of VMware instances. However, this contradicts one of the principles of cloud computing in which resource sharing is advocated as a means to optimise resource usage. Even if this contradiction were not fundamental, the performance of the lightweight approach is clearly superior in this scenario, as shown in figure 4.

Although, our lightweight VMs cannot allow different operating systems to run on top of the host operating system, there are certain environments in which having a single operating system running is enough to cover the demands. For instance, various Linux applications can benefit from achieving resource isolation from a VM and still share the same operating system image for working properly.

VI. Concluding Remarks

We have presented the D-VTM framework, which is a lightweight virtualisation approach to make cloud computing suitable for running high performance applications. Our framework is based on application-level virtualisation in which VMs are built on top of the operating system services. At the core of the framework we have D-VTMs, which represents a distributed VM encapsulating a pool of resources and provide the execution environment of distributed computations. A D-VTM involves multiple lightweight VMs running in different nodes in charge of executing a parallel application. In addition, a number of VM management functions are provided. The Job Scheduler attends job requests accompanied with the desired number of VM instances. The VTM Scheduler is able to determine the availability of resources as well as performing resource reservations. Instances of D-VTMs are created by the VTM Factory. Lastly, the Job Monitor is responsible for monitoring the execution of jobs.

The experimental results have shown that our lightweight VM approach involves about 5% overhead. In contrast, VMware VMs are slower and the performance obtained is prohibitive when the hardware is shared with other CPU- and communication-intensive applications. Therefore, our approach is much more efficient than VMware VMs for running WRF while retaining the benefits from cloud computing.

The experimental results presented were focused solely on WRF; future work will look into evaluating our approach with other HPC applications. Also, experiments on a larger cluster are due. Our prototype will be extended to control other kind of resources such as memory and network resources. Finally, we will also consider dynamic D-VTM resource management which allocates resources to lightweight VMs on demand according to application performance and resource utilisation objectives.

Acknowledgment

Hector A. Duran-Limon would like to thank the State Council of Science and Technology of the State of Jalisco (COECYTJAL) (grant 495-2008), IBM (Faculty Award 2008) and the Mexican's Public Education Ministry (grant PROMEP 103.5/08/0992) for supporting this work. Ming Zhao's research is sponsored by National Science Foundation under grant CCF-0938045 and Department of Homeland Security under grant 2010-ST-062-000039. This work is part of the Latin American Grid (LA Grid) initiative [13]. The authors are also thankful to the anonymous reviewers for their useful comments. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] K. Antypas, J. M. Shalf, and H. Wasserman, "NERSC-6 workload analysis and benchmark selection process," LBNL, Tech. Rep., 2008.

[2] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman and N. Wright, "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud Amazon Web Services Cloud", CloudCom 2010.

[3] G. Wang and T. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center", INFOCOM 2010.

[4] Sun, C., Nishimura, H., James, S., Song, K., Muriki, K., and Qin, Y., "HPC Cloud applied to Lattice Optimization", International Particle Accelerator Conference (IPAC´11), San Sebastian, Spain, September 2011

[5] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar 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.

[6] Juan C. Martinez, Lixi Wang, Ming Zhao, and S. Masoud Sadjadi. 2009. "Experimental study of large-scale computing on virtualized resources". In Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing (VTDC '09). ACM, New York, NY, USA, 35-42

[7] Thomas A. Henzinger, Anmol V. Singh, Vasu Singh, Thomas Wies, and Damien Zufferey, "Static Scheduling in Clouds", HotCloud 2011

[8] Lee, G.; Chun, B.; Katz, R.H., "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud", 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2011)

[9] Kroeker, K. L. 2009. The evolution of virtualization. *Commun. ACM* 52, 3 (Mar. 2009), p. 18-20.

[10] WRF, 2011. The Weather Research and Forecasting Model. http://wrf-model.org

[11] H.A. Duran-Limon, M. Siller, G.S. Blair, A. Lopez, and J.F. Lombera-Landa. Using lightweight virtual machines to achieve resource adaptation in middleware. IET Softw. 5, 229 (2011).

[12] HWRF, 2011. The Hurricane Weather Research and Forecast System. http://wwwt.emc.ncep.noaa.gov/?branch=HWRF

[13] LA Grid, Latin American Grid, http://latinamericangrid.org/

[14] J. Ekanayake and G. Fox, “High Performance Parallel Computing with Clouds and Cloud Technologies.” In 1st International Conference on Cloud Computing (CloudComp09), 2009.

[15] Constantinos Evangelinos and Chris N. Hill, “Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon’s EC2,” Cloud Computing and Its Applications 2008.

[16] Jaliya Ekanayake, Thilina Gunarathne, and Judy Qiu, “Cloud Technologies for Bioinformatics Applications,” IEEE Transactions on Parallel and Distributed Systems, Vol. 22, Issue 6, June 2011.

[17] Changchun Sun, Hiroshi Nishimura, Susan James, Kai Song, Krishna Muriki, Yong Qin, “HPC Cloud Applied To Lattice Optimization,” Proceedings of 2011 Particle Accelerator Conference, New York, NY, USA.

[18] H. Raj and K. Schwan, "High Performance and Scalable I/O Virtualization via Self-virtualized Devices", In Proceedings of the 16th International Symposium on High Performance Distributed Computing, Pages 179-188, 2007.

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

[20] VMware. VMware Infrastructure 3 architecture. Technical paper.

[21] XenSource. A Performance Comparison of Commercial Hypervisors, 2007.

[22] VMware. A Performance Comparison of Hypervisors, 2007.

[23] Resource containers: A new facility for resource management in server systems. Gaurav Banga, Peter Druschel and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation. 1999.

[24] Linux-Vserver. http://www.linux-vserver.org

[25] J. Lange, K. Pedretti, P. Dinda, P. Bridges, C. Bae, P. Soltero, A. Merritt, “Minimal Overhead Virtualization of a Large Scale Supercomputer,” Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011), March, 2011.

[26] Control group (cgroup). http://www.mjmwired.net/kernel/Documentation/cgroups/

[27] Hoffa, C., G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman and J. Good, On the Use of Cloud Computing for Scientific Workflows,” SWBES 2008.

[28] Vecchiola, C., Pandey, S., and Buyya, R., “High-Performance Cloud Computing: A View of Scientific Applications”, 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN '09). IEEE Computer Society, Washington, DC, USA, 2009.

[29] DiViNE, Disaster mItigation on VIrtualised eNvironmEnts, http://maestro.cucea.udg.mx/~hduran/project/NaturalDisasterMitigation/v2_NaturalDisasterMitigation_project.html