# Efficiency Assessment of Parallel Workloads on Virtualized Resources

Javier Delgado,
S. Masoud Sadjadi
Florida International University
{javier.delgado,sadjadi}@fiu.edu

Liana Fong, Yanbin Liu,
Norman Bobroff, Seetharami Seelam
IBM T. J. Watson Research Center
{llfong,ygliu,bobroff,sseelam}@us.ibm.com

***Abstract*—In cloud computing, virtual containers on physical resources are provisioned to requesting users. Resource providers may pack as many containers as possible onto each of their physical machines, or may pack specific types and quantities of virtual containers based on user or system QoS objectives. Such elastic provisioning schemes for resource sharing may present major challenges to scientific parallel applications that require task synchronization during execution. Such elastic schemes may also inadvertently lower utilization of computing resources. In this paper, we describe the *elasticity constraint effect* and *ripple effect* that cause a negative impact to application response time and system utilization. We quantify the impact using real workload traces through simulation. Then, we demonstrate that some resource scheduling techniques can be effective in mitigating the impacts. We find that a tradeoff is needed among the elasticity of virtual containers, the complexity of scheduling algorithms, and the response time of applications.**

**General Terms:** scheduling, virtualization, parallel application

## I. Introduction

The current trend in resource provisioning for utility and cloud computing is to provide compute resources as virtual resource containers using virtualization technologies. The most common realization is the use of virtual machine (VM) technology, in which a virtual container manager (VCM) abstracts and controls the physical resources allotted to one or more VMs. The VM abstraction simplifies the deployment of application environments across a wide range of physical systems. The VCM provides fine-grained control of the amount of shared resources to apportion to each VM when they are competing for them. This provides resource providers flexible resource allocation control to satisfy quality of service of requests and to build cost and profit models [1]. This flexibility also allows application execution to be tuned in different ways. For example, execution of a job proceeds faster than expected when its VM borrows idle CPU cycles from collocated VMs. Moreover, parallel applications need not limit the degree of parallelism to the number of physical machines.

The simplicity of application deployment and the flexibility of virtual resource acquisition enabled by the *Cloud* is creating interest among users of CPU-intensive parallel applications [2]. However, this flexibility can present a unique challenge when scheduling and executing them.

Based on its communication patterns, a parallel application is categorized as either tightly-coupled or loosely-coupled. Loosely-coupled applications require infrequent communication among tasks. Tightly-coupled applications must communicate frequently between computation intervals. Thus, their tasks proceed in a "synchronized" manner and the slowest task limits the resource utilization of all other tasks. Because of this, if the tasks of a tightly-coupled parallel application are allotted different amounts of resources, synchronization can lead to inefficient use of some of the resources. Thus, allocating tasks to resources is challenging in virtualized environments, since current provisioning schemes can result in different allotments for different workers. This problem must be addressed in order for both consumers and producers of virtualized environments like the cloud to maximize their return on investment.

Our prior work investigated and proposed new VM-centric metrics of compute capacity that are useful for making job placement decisions in a cloud-like environment [3]. A limitation of that work was that only serial workloads were evaluated. This work extends our prior study to scheduling tightly-coupled parallel scientific workloads on VMs. An early and interesting observation we made was that as more and larger parallel jobs are introduced into the workload trace, the overall system utilization drops from about 100% to 50%. Efforts to increase the load by increasing the simulated job arrival rate or trying more aggressive backfill policies in the scheduling algorithm failed to improve system utilization. Investigation of the source of this scheduling problem leads to the introduction and measurement of 2 phenomena that occur when running multiple parallel applications in virtualized environments in which different application tasks can have different CPU allocations and/or capacity limits. We refer to these as the *elasticity constraint effect* and the *ripple effect* on parallel job execution. *Elasticity constraint effect* refers to the possibility of resources being left under-utilized resulting from the configuration of elasticity constraints on virtual containers. *Ripple effect* is the under-utilization of resources resulting from running applications with parallel task synchronization. Both of these effects negatively impact the utilization of computing resources and application response time.

In this paper, we detail the sources of the elasticity constraint and ripple effects and quantify their impact in

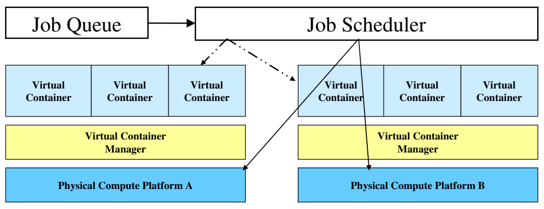

Figure 1. Virtual Container vs Physical Container Scheduling

environments in which highly synchronized parallel jobs are placed in virtual containers that compete for physical CPU capacity. We then experiment with new job scheduling algorithms that address the technicalities of the virtual container provisioning paradigm as well as the elasticity constraint and ripple effects.

We find that executing parallel applications in virtualized environments can result in sub-optimal usage of the underlying physical machines. In fact, we observed up to 47% average under-utilization of the CPU due to ripple effect when using a greedy scheduling algorithm. By applying some simple heuristics that address the elasticity constraint and ripple effects to scheduling algorithms based on the load metrics introduced in [3], we were able to reduce this to 35%, as a result improving median job expansion factor by more than a factor of 3. However, we conclude that more sophisticated algorithms are needed to directly address this issue. Reducing the heterogeneity of VMs reduces the capacity wastage, but hampers the ability to satisfy varying QoS objectives, which is desirable in cloud environments.

## II. System Model and Virtualization Effects

### A. System Model: Resource Sharing in Virtualized Platforms

In a virtualized cluster, a pool of virtual containers is hosted on a set of physical machines. A virtual container manager (VCM) orchestrates resource sharing among these virtual containers. Figure 1 shows two example physical compute platforms each with three virtual containers. The job scheduler can assign jobs directly to the physical platform (solid lines) or to the virtual containers (dashed lines).

The resource sharing policy among virtual containers can be defined by three scheduling or *sharing* parameters: `min`, `max`, and `share`. A set of these is assigned to each VM.

1) `min` - This is the minimum platform capacity that the VM is guaranteed. The capacity cannot be used by collocated VMs even when the VM is inactive.
2) `max` - This is the maximum capacity that a VM can obtain even when there is free capacity.
3) `share` - This apportions free capacity among competing VMs. It is further explained below.

The above sharing model is supported by VMWare [4] and IBM hypervisor [5]. The parameters are best explained with reference to Figure 2. This figure shows two virtual machines ($VM_A$ and $VM_B$) collocated on a physical platform with specific values assigned for `min`, `max`, and `share` for each VM. According to these parameters, $VM_A$ can use at most $\frac{4}{7}$ of the total physical capacity, even when $VM_B$ is not using any; $VM_B$ can use the full physical capacity if $VM_A$ is idle. Since `min` is zero, there is no guaranteed minimum capacity for either of the VMs. The `share` distributes available capacity in relative proportion to each VM, with its `min` and `max` as lower and upper bounds. If all VMs have equal share, they get equal capacity.

The system has a *homogeneous* VM configuration when all VMs have the same `min`, `max`, and `share` values. When these values differ, it has a *heterogeneous* VM configuration. For example, the system shown in Figure 2 is heterogeneous because the two VMs have different values of `max` and `share`. There are numerous reasons for setting these values to be different across different VMs, the primary reason is to provide different quality of service to workloads. In the case of Figure 2, workloads with higher importance could be assigned to $VM_B$ because it has a higher share (4 vs. 3 for $VM_A$). The use of `max` is useful for capping cost expenditures for resource usage.

Now we can introduce two new terms that will be used for job scheduling in virtualized environments: *potential capacity* and *equilibrium capacity*. Potential Capacity (PC) reflects the instantaneous capacity that a VM can obtain, depending on the set of VM sharing parameters discussed above and the current utilization of the collocated VMs. Thus, its value is computed dynamically. Equilibrium Capacity (EC) is the minimum resource capacity guaranteed to a VM when all other collocated VMs are contending for resources. The value of EC is invariant of the resource usage by the collocated VMs and only depends on the sharing parameters. For example, the EC of $VM_A$ and $VM_B$ in Figure 2 are $\frac{3}{7}$ and $\frac{4}{7}$, respectively. The computation of PC and EC is not straightforward with different `min` and `max` values; additional details on these terms are discussed in [3].

Now the job scheduling algorithms can use EC and PC for scheduling jobs on virtual machines. Greedy scheduling algorithms for scheduling jobs on physical machines can simply use the free capacity (FC), i.e., the unused capacity of the physical machine, as the primary metric for job placement. With VMs, the FC of the underlying physical platform is not an ideal metric since it is shared with other VMs. Instead, we use PC and EC, which give us both an

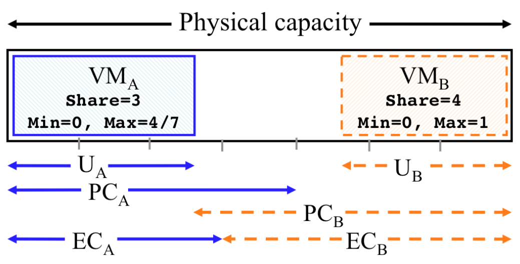

Figure 2. Container CPU resource sharing model

instantaneous upper bound and an invariant lower bound of how much capacity a VM can obtain, as job placement metrics.

### *B. Elasticity Constraint Effect and Ripple Effect in Executing Parallel Jobs on VMs*

Two major problems arise from virtual machine configurations – in terms of the scheduling parameters (`min, max, share`), and in terms of homogeneous and heterogeneous VMs. We call the first problem *elasticity constraint effect* because it occurs due to the constraints set by parameter `max`. We call the second problem *ripple effect*; it is caused when the capacity allocation of one task affects the rest of tasks of the same parallel job.

We illustrate these two issues with a set of examples shown in Figures 3 through 5. These figures show different physical machines ($PM_1, \ldots, PM_4$) each with 2 or 3 virtual machines ($VM_1$, …, $VM_3$). We set `min=0` and `share=1` for all the VMs. The maximum utilization of a VM (`max`) is set to different values to illustrate the issues. Several parallel jobs (A-E), each with a different number of tasks, are placed on these virtual machines (one task per virtual machine).

Consider Figure 3(a) which shows $PM_1$ with a `max` of 0.5 and 0.4 for $VM_1$ and $VM_2$, respectively. Job A executes on $VM_1$ and Job B on $VM_2$. Both of these jobs are compute intensive and each can utilize as much capacity as possible from the underlying physical machine. However, their actual utilization will be limited to 0.5 and 0.4, respectively, because of the `max` (i.e. the elasticity constraint) set on the VMs. Therefore, 0.1 of the platform capacity will not be utilized, even though either job could fully utilize it if not for the `max` parameter. This is one example of system utilization inefficiency due to the elasticity constraint.

Another example of elasticity constraint effect is shown in Figure 3(b). In this case, tasks of jobs C and D are collocated on $PM_3$ and $PM_4$ so each share 0.5 of the utilization. Now, since job C is a parallel job with three tightly-coupled tasks, its utilization on $PM_2$ will be limited to 0.5 as well. Job E on $PM_2$ will have a utilization of 0.4 because of the elasticity constraint of $VM_3$. Again in this case, the total utilization of $PM_2$ will be limited to 0.9.

Obviously, the elasticity constraint effect will be observed only for VMs with `max<1`. Moreover, running highly-synchronized parallel jobs will exacerbate the under-utilization situation.

Another source of under-utilization is what we call *ripple effect*; it is caused by parallel task synchronization. We illustrate the concept of ripple effect by referring to Figure 4 with three homogeneous VMs. In this situation, as the left hand side of the figure shows, the cluster initially has 3 jobs: Job A has three tasks that are placed on one VM from each of $PM_2$, $PM_3$, and $PM_4$, Job B has one task on $PM_1$, and Job C has one task on $PM_3$ and one on $PM_4$. The system utilization in this configuration is $\frac{7}{8}$ because Job A's task on $PM_2$ will only utilize half of the CPU due to task synchronization. On $PM_3$ and $PM_4$, Job A and Job C share the physical machine and thus each can only get $\frac{1}{2}$, so Job A cannot use more than 0.5 on $PM_2$. Next, as shown on the right hand side of the figure, we schedule Job D with three tasks on one VM from each PM. On $PM_3$, each task of Jobs A, C, and D now share the PM and would get a utilization of $\frac{1}{3}$. Then, the utilization of Jobs A, C, and D (on $PM_2$ and $PM_4$ for Job A, $PM_4$ for Job C, and $PM_1$ and $PM_2$ for job D) would also be $\frac{1}{3}$. As a result, the utilization of $PM_4$ drops from 1 to $\frac{2}{3}$ and the utilization of $PM_2$ increases from $\frac{1}{2}$ to $\frac{2}{3}$. Hence, the allocation of job D causes the cluster utilization to be lowered unexpectedly due to the ripple effect.

Another example of ripple effect is presented in Figure 5. The ripple effect is more pronounced in heterogeneous VMs with different shares, as shown in Figure 5(b), than homogeneous VMs with same shares, as shown in Figure 5(a). Due to space limit, we will not describe this in detail.

In summary, elasticity constraint and ripple effects can

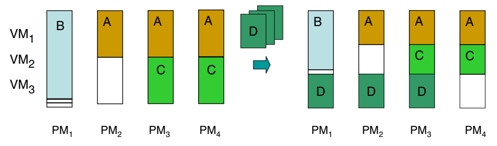

Figure 4. Example of a scenario that causes ripple effect

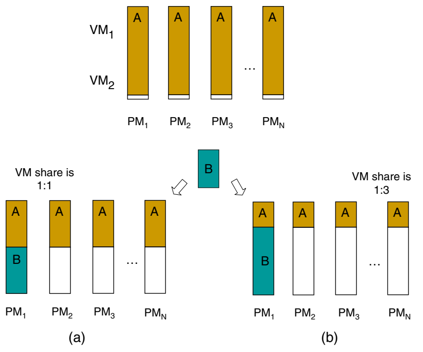

Figure 5. Another scenario that results in ripple effect

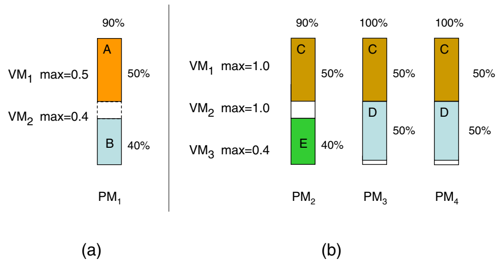

Figure 3. Examples of elasticity constraint effect

impact the efficiency of running parallel applications in virtualized environments. Resource selection algorithms need to consider these effects to mitigate their negative impacts.

## III. Scheduling Algorithms

To evaluate and mitigate the elasticity constraint and ripple effects on parallel workload execution, various methods of job scheduling need to be considered, including both the job ordering policy and the resource selection algorithm. The job ordering policy we use is first come first serve (FCFS) complemented with backfill [6]. Backfill improves system utilization without starving jobs requiring a relatively large number of tasks. In this policy, the job at the front of the waiting queue, for which there are not enough resources to run, will reserve VM(s) at a future time. Other jobs queued behind this job can be assigned to currently idle resources if their execution does not interfere with the job's reservation. In our algorithm, we use EC, which represents the worst-case utilization of a VM, to estimate the runtime of a queued job conservatively when deciding if queued jobs should be run ahead of the job with the reservation.

The resource selection algorithm, which selects the set of VMs that should be assigned to the set of tasks of a job, greatly affects the job's execution time and the efficient utilization of the compute resources. Most algorithms introduced here are at their core a variation of load balancing algorithms. These approaches are based on greedy heuristics such as total free machine capacity in a non-virtualized environment. In a virtualized environment, the PC or EC metrics are shown to work well to maximize the overall system utilization and job response time when serial jobs dominate the workload with homogeneous VMs [3].

However, as the portion of parallel jobs in the workload increases, the elasticity constraint and ripple effects challenge the applicability of classical job scheduling algorithms. Our initial set of experiments confirm this speculation that the effects negatively impact the overall system efficiency, as shown later in Section V. Thus, we devise some resource selection algorithms to mitigate these effects and then evaluate their effectiveness.

The starting point for the algorithms is the straightforward greedy approach using EC or PC as task placement metrics. Here, VMs are ranked according to the value of the placement metric, in descending order, and tasks are assigned accordingly. (Some subtleties are discussed in [3]). Some intuition about the importance and reasoning behind our choices for the resource selection algorithm is exemplified in Figure 6. The underlying system has 7 PMs (A-G), each of which has 4 VMs (1-4), such that $A_1$ represents the first VM on PM A. The figure illustrates the scenario of scheduling a 4-task job when 9 VMs, ranked by either EC or PC, are available. A load-based greedy algorithm selects the 4 VMs with the highest value of EC or PC, i.e. the ones enclosed in the colored rectangle. In contrast, the concept of

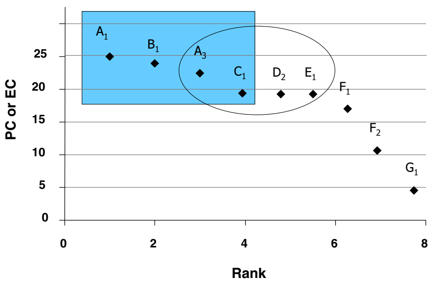

Figure 6. Illustration of ranking scheme for scheduling algorithms

a *semi-greedy* approach is illustrated by selecting the 4 VMs enclosed in the ellipse, where the last two VMs, $D_2$ and $E_1$, have the same rank as $C_1$. The semi-greedy idea comes from the observation that a tightly-coupled parallel job's execution rate is limited to that of its slowest executing task, in this case $C_1$. Thus, both selections result in the same execution rate for the job, but the semi-greedy approach leaves the bigger capacity on $A_1$ and $B_1$ to later jobs.

The resource selection algorithms implemented for the experiments are described below:

- *PC-g*: Select VMs with the highest PC ranking.
- *PC-sg*: Select VMs according to their PC ranking using the semi-greedy algorithm.
- *PCFC-sg*: An extension of PC-sg with special treatment of serial jobs. When scheduling a serial job, first select the VM with the highest PC ranking from the VMs whose PC is not bigger than the free capacity (FC) of the underlying physical machine. If no such VM is found, select the one with the highest PC ranking.
- *EC*: Select VMs with the highest EC ranking.
- *PMVM*: Prioritize PMs with the least number of active VMs; use $share$ as a secondary metric (or *tie-breaker*).

Among the above algorithms, PC-g does not consider the ripple effect. PC-sg reduces the ripple effect by trying to assign parallel tasks to VMs with similar PC. PCFC-sg directly addresses the ripple effect by preventing serial jobs from being assigned to VMs whose PC is bigger than the free capacity (FC) of the PM. This technique prevents serial jobs from creating a ripple effect by being placed on PMs currently running other job(s) and avoids the effect shown in Figure 5. When no VM satisfying this criteria is found, the job is assigned to the highest-PC VM.

PMVM is an algorithm that does not rely on any of our new metrics (EC and PC); it is a simple extension to current resource selection algorithms for physical machines. It is the baseline case for comparison with the EC or PC based algorithms. The above algorithms are not necessarily optimal techniques, but instead illustrate the new challenges faced when scheduling jobs in virtualized environments.

## IV. Experimental Setup

### A. Experimental Platform

We build a trace-driven, event-based simulator using Matlab[1] to measure the performance of scheduling algorithms with workloads from real world traces. The simulator has three components, a *Resource Manager*, a *Job Manager* and a *Scheduler*. The Resource Manager monitors VM states, including each VM's allocated job and its utilization. The states are changed and re-calculated upon job arrival and completion events. It also provides interfaces for other components to inquire about resource states and resource parameters such as PM utilization, EC, PC, and FC. The Job Manager generates job arrival and completion events. The Scheduler schedules jobs on VMs based on resource information from the resource manager upon job events generated by the job manager.

### B. Workloads for Simulation

Experiments are driven by two widely used traces from the Cornell Theory Center (CTC)[2] and the Grid'5000[3]. The traces are run unmodified, using each job's specified number of tasks and run time as the number of VMs required to execute the job when run in isolation (i.e. using the full CPU of the PM). The execution time is assumed to increase proportionally to the allotted share of the CPU given to a task. Our simulations process the first 10,000 jobs of each trace. The Grid'5000 trace has a significantly lower portion of parallel jobs than the CTC trace, which will demonstrate the issues raised by the ripple effect with more parallel jobs.

### C. Simulated Infrastructure

The simulated compute cluster consists of 64 PMs, each with 4 VMs. Each physical machine has a normalized CPU capacity of unity that is shared among the 4 collocated VMs. We assume that each task is compute intensive and can use up to the full capacity of the underlying physical machine. This assumption is somewhat unrealistic, but it is not possible to get fine-grained information about temporal CPU resource utilization data from the trace files. Also, if the VM scheduler fairly distributes CPU cycles and there are multiple VMs on a physical machine, it is not unreasonable to expect that the CPU will be constantly at full utilization. The actual CPU consumption of the VM depends on multiple factors, including not only the collocated VMs' states, but also the states of VMs on other PMs because of the ripple effect introduced by the parallel jobs. Since idle VMs consume negligible CPU resources, we assume that unoccupied VMs consume no CPU cycles.

Except for the study in which we explicitly set a maximum VM capacity less than 1, all VMs have `min`=0 and `max`=1. In experiments where the VMs are described as being homogeneous, each VM has `share`=1. For the heterogeneous case, the `share` assigned to the 4 collocated VMs on each PM are 1, 1, 2, 4. Hence, their EC values are $\frac{1}{8}$, $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{2}$ of the physical CPU, respectively.

[1] http://www.mathworks.com/products/matlab/

[2] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

[3] http://gwa.ewi.tudelft.nl/pmwiki/

## V. Observations

The elasticity constraint and ripple effects are evaluated together with the effectiveness of different scheduling algorithms for parallel workloads. The evaluation metrics include cluster utilization ($U$), efficiency ($\eta$), expansion factor ($XF$), and makespan ($M$). Cluster utilization $U$ is the average CPU utilization over all the PMs in the cluster. Cluster efficiency $\eta$ is the average CPU utilization of occupied PMs in the cluster. An occupied PM is one that is currently executing a task on at least one of its VMs. Efficiency measures the deficit in resource utilization caused by the elasticity constraint and ripple effects. This is because in the absence of these effects (i.e. with all VMs' `max`=1 and no parallel jobs), any VM executing a job uses all available CPU cycles from idle collocated VMs, which results in 100% utilization of the corresponding PM. Then, for example, as parallel jobs are introduced to the workload the ripple effect can preclude them from fully utilizing the idle cycles.

The remaining two metrics address the job performance or user perspective in contrast to the system resource or provider viewpoint. The XF for each job is the "bounded slowdown" of [7] and defined for a virtual environment in Equation 1. Because of resource elasticity during execution, the equation distinguishes a compute component ($XF_c$) that depends on system state from a pure queuing term. The threshold in the denominator limits a potential bias caused by modest delays to very short duration jobs. A threshold of 10 seconds is used here. The median $XF_c$, median XF, and $90^{th}$ percentile values are reported from our experiments.

$$XF = \frac{(queueTime + actualRunTime)}{max(threshold, idealRuntime)} \tag{1}$$

Makespan is the total time to execute a trace of N jobs and reflects a combination of the benefits of scheduling techniques, but restricted by actual job arrival patterns. Results are presented for the Grid'5000 trace, unless otherwise noted.

### A. Inefficiency due to the Elasticity Constraint Effect

The elasticity constraint effect is isolated from ripple effect by filtering the trace to contain only single node jobs. As noted above, so long as the `max` parameter of each VM is 1, the efficiency $\eta$ of the cluster is unity, meaning the CPU is fully utilized on all occupied PMs. This behavior is noted for a heterogeneous cluster using the PC-g algorithm in Figure 7, where the horizontal axis is the occupancy ratio and the vertical axis is the cluster utilization $U$. Note that $\eta$ can also be represented as the ratio between $U$ and the

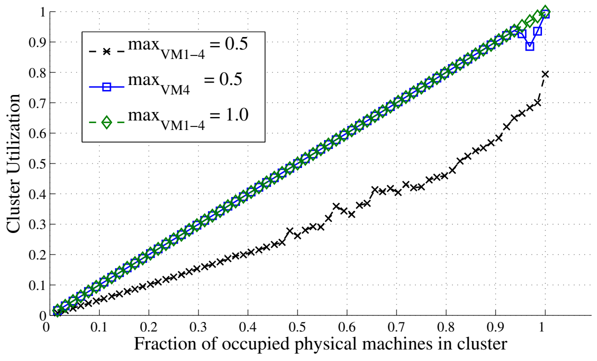

Figure 7. Efficiency $\eta$ on heterogeneous VMs for serial jobs

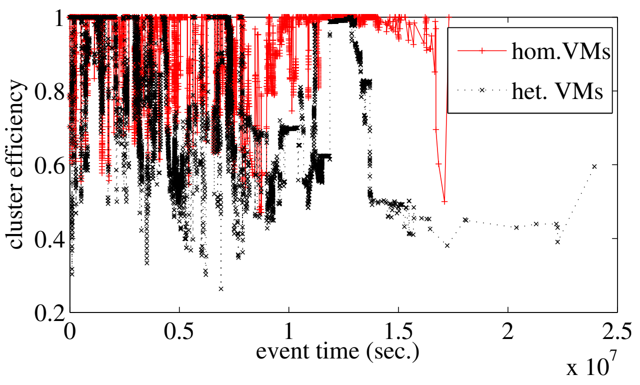

Figure 8. Efficiency $\eta$ for homogeneous (hom.) and heterogeneous (het.) VMs

occupancy rate. The diamond markers (green) show the data with unconstrained VMs, where all VMs have `max=1`. Then, we reduce the max parameter of $VM_4$, which has the largest share (4), on each PM, to 0.5. This change (blue squares) has little effect on efficiency because the PC metric accounts for the decreased `max` ($PC \leq max$) and the greedy approach chooses to place jobs on VMs with smaller share but `max=1`. There is a small decline at high occupancy when there are fewer VMs to choose from. That is, $VM_4$ has a better chance of being chosen, resulting in constrained utilization of the PM. A large drop in $\eta$ occurs when all VMs have `max=0.5` (black 'x'). The PC-g algorithm balances the load across PMs and there are long periods with less running jobs than there are PMs, thus the cluster is not fully utilized. In addition, the mean XF in this case is about 40% higher than in the case of unconstrained VMs.

## B. Inefficiency due to Ripple Effect

The ripple effect is more problematic than the elasticity constraint effect since it cannot be solved by managing VM sharing properties. It is evaluated with the homogeneous and heterogeneous VM sharing configurations described in Section IV-C and without filtering parallel jobs. Figure 8 compares the efficiency as a function of time for heterogeneous (dashed black line) and homogeneous (solid red line) VM configurations. The impact of the ripple effect is apparent in the figure as $\eta < 1$ most of the time. Furthermore, the homogeneous configuration utilizes the resources more efficiently throughout the trace compared to the heterogeneous case. Its makespan is 30-40% lower; its average efficiency is 89%, compared to 65% for the heterogeneous case; its median expansion factor is more than 30% lower. These results lead us to conclude that heterogeneity should be minimized for workloads with parallel jobs for high cluster efficiency, unless more sophistic scheduling algorithms that address the ripple effect are devised. These tests were performed with the PCFC algorithm, which combines all our heuristics for mitigating the ripple effect, as described in Section III.

## C. Algorithm Comparison on Homogeneous VMs

In this section, different scheduling algorithms are compared on their performance for the homogeneous VM configuration.

The observed scheduling performance is shown in Table I for each algorithm. The algorithms perform similarly, except for the EC algorithm. Note that for homogeneous VM configurations, all VMs have the same EC value. Since our EC algorithm does not have a secondary sort criteria, the algorithm assigns the job tasks sequentially to VMs. This characteristic reduces the ripple effect since packing tasks of a job into fewer PMs reduces the chance of uneven task distribution. On the other hand, it increases the XF and reduces overall U when the number of tasks in the cluster is low, as tasks are collocated on PMs when there are free PMs in the cluster. However, when the number of tasks in the cluster is high and few PMs are left free, the benefit of reducing ripple effect may become dominant, so the overall XF and U depend on the workload pattern. As shown in Table I, compared to other algorithms, EC has the best efficiency and 90th percentile expansion factor, but the worst median expansion factor, makespan, and cluster utilization.

Overall, the obtained results suggest that the ripple effect is reduced by packing tasks from the same job on fewer physical machines, but this should only be done when the ratio of tasks to PMs is high, otherwise the cluster can remain underutilized.

Table I
UTILIZATION AND EXPANSION FACTORS FOR 64 PMS AND 4 HOMOGENEOUS VMS/PM

| Algorithm | $U$ | $\eta$ | $XF_c$ | $XF$ | $XF_{90}$ | Mkspan(L) |
|---|---|---|---|---|---|---|
| PC-sg | 0.725 | 0.89 | 4.0 | 6.24 | 795 | 202.92 |
| PCFC-sg | 0.729 | 0.89 | 4.0 | 5.98 | 1006 | 201.78 |
| EC | 0.706 | 0.95 | 4.0 | 11.31 | 414 | 208.28 |
| PMVM | 0.720 | 0.88 | 4.0 | 5.97 | 784 | 204.12 |

## D. Algorithm Comparison with Heterogeneous VMs

The tests of the previous subsection are repeated with the heterogeneous VM configuration.

The scheduling performance metrics for all algorithms are shown in Table II. The table includes the PC-g performance data to contrast the impact of heterogeneity in the fully greedy case compared to the semi-greedy case. The results are mixed, but clearly EC and the PC-sg based algorithms perform more efficiently and their expansion factors are lower than the other methods. With the chosen cluster size, due to the large variation in execution times, the $XF_{90}$ values observed are disproportionately variable as the XF values grow exponentially. However, the median XF values obtained agree with our assumptions.

The efficiency of each algorithm compared by measuring $U$ at each occupancy level is presented in Figure 9. The line at 45 degrees represents optimal efficiency. When less than 50% of the PMs have jobs, all algorithms perform close to optimal. The EC (brown '+'), PC-sg (black 'x') and PCFC-sg (blue circles) are more efficient than PC-g (orange diamonds) and PMVM (green squares). When all PMs are active, PCFC-sg, PC-sg, and EC average roughly the same utilization, with PCFC-sg performing slightly better. PC-g and PMVM perform much worse at this stage due to their tendency to select VMs without consideration of available capacity of the available of VMs, which increases the ripple effect.

Comparing these results to the homogeneous case clearly confirms our previous observation that heterogeneity greatly affects the efficiency and job performance of the cluster. Even with workloads like the Grid'5000, consisting mostly of serial jobs that consume all the compute cycles lost due to the ripple effect, this is the case. Another observation is that the greediest techniques actually make the problem worse. Hence we reiterate that VMs should be as similar as possible if simple scheduling techniques are used.

Table II
UTILIZATION AND EXPANSION FACTORS FOR 64 PMS AND 4 HETEROGENEOUS VMS/PM

| Algorithm | $U$ | $\eta$ | $XF_c$ | $XF$ | $XF_{90}$ | Mkspan(L) |
|---|---|---|---|---|---|---|
| PC-g | 0.490 | 0.53 | 2.00 | 30.7 | 910.84 | 299.80 |
| PC-sg | 0.564 | 0.65 | 2.00 | 8.29 | 887.81 | 260.59 |
| PCFC-sg | 0.529 | 0.61 | 1.66 | 8.00 | 380.31 | 277.87 |
| EC | 0.559 | 0.64 | 1.75 | 8.31 | 407.97 | 262.96 |
| PMVM | 0.508 | 0.56 | 2.00 | 18.1 | 820.57 | 289.66 |

## E. Ripple Effect of CTC Trace

As expected, the ripple effect is more significant for the CTC trace, due to its high percentage of long running and large parallel jobs. Figure 10 shows the temporal efficiency for two scheduling algorithms: PCFC (black 'x' and blue/dark 'o') and PC-sg (light/red '+' and light/green squares) with the homogeneous and *default* heterogeneous

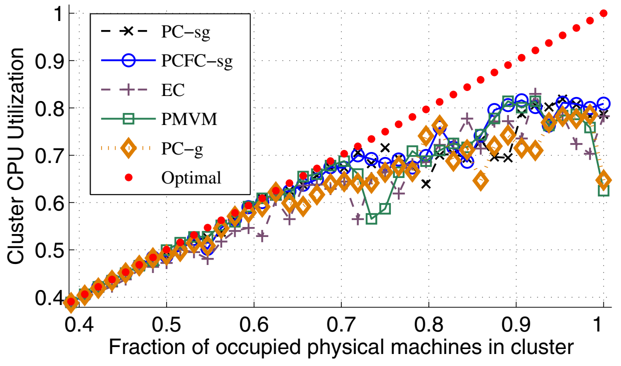

Figure 9. Efficiency $\eta$ for heterogeneous VMs with five scheduling algorithms

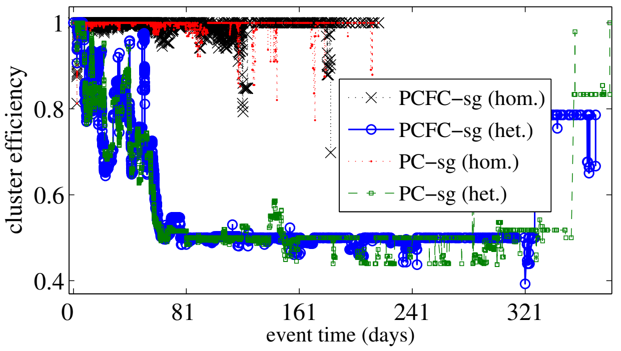

Figure 10. Efficiency $\eta$ of the CTC trace for homogeneous and heterogeneous VMs

VM configurations. From the event time along the X-axis, the makespan is roughly doubled for the heterogeneous VM case. In terms of efficiency, both algorithms are about 98% for all the homogeneous and 56-58% for the heterogeneous configurations. The median expansion factors were nearly twice as large in the heterogeneous VM case, clearly due to the capacity lost to the ripple effect. These results further confirm the negative impact of ripple effect in virtualized environments with heterogeneous configurations.

## VI. Related Work

Virtualization is an essential aspect of cloud computing. However, it comes with important implications for application performance and system utilization.

The virtualization overhead can impact the performance of high performance computing (HPC) applications. The authors of [8] reported the impact of Xen on MPI applications and the authors of [9] quantified the virtualization overhead on HPC applications such as BLAST, GROMACS and HMMer. This paper does not address the virtualization overhead. The focus of the paper is on assessing the effects that negatively impact parallel job execution in virtualized environments and the potential mitigation of the impact with new job scheduling algorithms.

Parallel job scheduling on physical machines is a mature field and is well studied. The development and evaluation of parallel job scheduling algorithms are many and a good summary on issues and approaches is in [10].

There are some studies on the job scheduling algorithms and performance of virtual machines mapped to physical machines [11], [12], [13]. Particularly, the work by Stillwell [14] addressed a set of scheduling issues. Stillwell's work proposed job placement methods based on current CPU utilization and memory occupancy using Multi-Capacity Bin Packing (MCB) algorithms. The job placements are then augmented by periodic preemption and migration to address workload imbalance. Unlike this work, our work uses new load metrics and addresses the scheduling and system utilization issues for mixed workloads of serial and parallel jobs. Our earlier work in [3] defined new load metrics and studied their applicability for serial workloads in homogeneous virtualized environments.

Feitelson looked at fragmentation [15] in gang scheduling that may cause wasted capacity in multiprocessor systems. Similarly, fragmentation occurs when scheduling on virtual machines, but in this case some processor capacity may not be utilized due to sharing physical machines among jobs and elastic resource specifications. Thus, we introduce two novel aspects in virtualization on parallel jobs performance: elasticity constraint and ripple effects.

## VII. Concluding Remarks and Future Work

Based on a generic elastic resource sharing model in virtualized environments, we describe the *elasticity constraint effect* and *ripple effect*, which negatively impact resource utilization and job response time in these environments. We quantify their impact using a simulator and present some new algorithms to mitigate this impact. We argue that simple extensions to traditional job scheduling techniques for physical resources are inadequate for virtualized environments. Hence, a new generation of load metrics and algorithms for job scheduling are required. Otherwise, we find that a tradeoff is needed among the elasticity of virtual containers, the complexity of scheduling algorithms, and the response time of applications.

In this paper, we show experimental results on a limited set of resource sharing parameters and job placement algorithms. For future work, we may investigate additional parameter settings (e.g. `min>0`), more sophisticated scheduling algorithms, and migration of tasks between virtual machines on the same physical machine or across physical machines.

## VIII. Acknowledgments

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

## References

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *HPCC*, 2008, pp. 5–13.

[2] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *ISPAN*, 2009, pp. 4–16.

[3] Y. Liu, N. Bobroff, L. Fong, S. Seelam, and J. Delgado, "New metrics for scheduling jobs on a cluster of virtual machines," in *IDPDP '11: Workshop on System Management Technique, Process and Services*, 2011.

[4] "VMWare: Virtualized Basics." [Online]. Available: http://www.vmware.com/virtualization/virtual-machine.html

[5] "IBM: Advanced POWER Virtualization on IBM System P5." [Online]. Available: http://www.redbooks.ibm.com/abstracts/sg247940.html

[6] D. A. Lifka, "The anl/ibm sp scheduling system," in *JSSPP*, 1995, pp. 295–303.

[7] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1997, pp. 1–34.

[8] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Evaluating the performance impact of xen on mpi and process execution for hpc systems," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. IEEE Computer Society, 2006.

[9] C. Macdonell and P. Lu, "Pragmatics of virtual machines for high-performance computing: A quantitative study of basic overheads," in *Proceeding of the High Perf. Computing and Simulation Conf.*, 2007.

[10] D. G. Feitelson and L. Rudolph, "Theory and practice in parallel job scheduling," in *IPPS '95: Proceedings of the Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1995.

[11] N. Bobroff, A. Kochut, and K. A. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Integrated Network Management*, 2007, pp. 119–128.

[12] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*. New York, NY, USA: ACM, 2008, pp. 175–184.

[13] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster, "Resource leasing and the art of suspending virtual machines," in *HPCC*, 2009, pp. 59–68.

[14] M. Stillwell, F. Vivien, and H. Casanova, "Dynamic fractional resource scheduling for hpc workloads," in *IPDPS*, 2010.

[15] D. G. Feitelson, "Packing schemes for gang scheduling," in *JSSPP*, 1996, pp. 89–110.