

New Metrics for Scheduling Jobs on a Cluster of Virtual Machines

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

Javier Delgado
Florida International University
{javier.delgado}@fiu.edu

Abstract—As the virtualization of resources becomes popular, the scheduling problem of batch jobs on virtual machines requires new approaches. The dynamic and sharing aspects of virtual machines introduce unique challenges and complexity for the scheduling problems of batch jobs. In this paper, we propose a new set of metrics, called potential capacity (PC) and equilibrium capacity (EC), of resources that incorporate these dynamic, elastic, and sharing aspects of co-located virtual machines. We then show that we mesh this set of metrics smoothly into traditional scheduling algorithms. We evaluate the performance in using the metrics in a widely used greedy scheduling algorithm and show that the new scheduler improves job speedup for various configurations when compared to a similar algorithm using traditional physical machine metrics such as available CPU capacity.

I. INTRODUCTION

The job scheduling discipline assigns work (e.g. batch jobs) to compute resources, matching job requirements to the capabilities and capacities of the resources. Traditionally, jobs are assigned to physical compute platforms. The recent trend is to expose the physical compute platform through one or more virtual resource containers, each providing an abstraction of the underlying resources. The most common example of a virtual resource container is the virtual machine (VM), first developed in the late 1960's [1] as a mechanism for timesharing a large system among many clients. The VM provides application isolation, and enhances and simplifies the administrative task of apportioning resources to jobs, when compared to the multiprogramming model where operating system controls are used to provide these functions but without complete process isolation.

The virtual container environment increases the complexity of the scheduling problem by extending the configuration options that control how a physical resource is exposed to the resource management system. Virtual containers, with VM as one example, provide a more complex target for the job scheduler than physical platforms with fixed capacity and a well studied utilization model.

Figure 1 provides an overview of scheduling jobs into multiple physical machines that host a single layer of containers, while multiple layers and heterogeneous resources are possible. In these virtualized computing systems, a container manager presents to each hosted VM an image of the compute resources of the underlying physical machine.

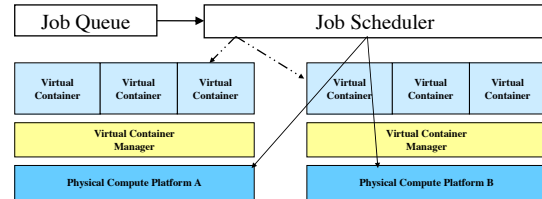


Figure 1. Virtual Container vs Physical Container Scheduling

Resources include processors, CPU cycles, memory, network, and I/O bandwidth which are apportioned to each VM image by the container manager according to VM attributes defined by the system administration and current demand. Unlike physical machines, the capacity or maximum amount of a resource, such as compute resource, that a VM can obtain is typically a dynamic variable. In addition to current demand, actions such as shutdown or start-up of other co-located VMs or migration to or from another physical host also lead to variable capacity.

In the job scheduling discipline, it is common to apply algorithms that rely on an estimate of free resources on the target compute nodes that are physical compute nodes with fixed capacity. In this paper, we consider how to get an accurate estimate of dynamic capacity of a VM particularly for CPU cycles. We introduce the concept of potential and equilibrium capacity, which leads to a derived set of dynamic container capacity metrics that prove to be useful in job scheduling.

In the world of dynamic provisioning and growing interest in cloud computing, the VM is a convenient unit for administrators to provision and timeshare physical compute resources among customers. The VM packages computing power with administrative flexibility (e.g. provisioning from a library of assembled images), suspend, checkpoint, migration, and flexible controls over sharing and managing the limited resources of the underlying physical platform. Therefore, the study of VM scheduling is an important research area. The related work in section IV summarizes several studies in virtualization related topics, but analysis of job scheduling in a virtualized environment is still in its infancy. Our paper addresses the topic of job scheduling in virtualized environment and the contributions are:

- a set of novel capacity metrics for virtual resources managed by virtual container managers
- a methodology for estimating the values of the new capacity metrics
- an illustrative use of the new capacity metrics in job scheduling
- a study of the impact of using dynamic capacity metrics in job scheduling via simulation using two real workload traces

II. NEW METRICS FOR JOB SCHEDULING ON VIRTUAL CONTAINERS

Most traditional batch workload schedulers use a *resource metric* – typically *free processor capacity* – to assign jobs to compute nodes. For a physical machine with a fixed capacity, schedulers estimate the free processor capacity straightforwardly as the difference between the total capacity and the current utilization. Further, these schedulers use a greedy heuristic to assign jobs to nodes. For example, an algorithm may select the nodes with the most free processor capacity to assign a job.

Estimating the *free capacity of a virtual container* is non-trivial because of the elastic, dynamic, and competing nature of the virtual containers co-located on a single physical machine. The free capacity of a virtual container refers to the compute cycles that the underlying virtual container manager would allocate if a new job were to run on the virtual container. This capacity depends on a number of factors, including the load of the virtual container, the utilization of all virtual containers in the parent physical machine, the sharing model of the co-located virtual containers and their parameters. To estimate it, a new set of metrics are proposed that reflect the sharing model of co-located virtual containers on the underlying physical machine. These metrics are used in new scheduling algorithms to optimally assign batch jobs to virtual containers.

A common resource sharing model of virtual containers that permits elastic and dynamic capabilities is described next. In this section, we use the terms virtual and physical containers instead of virtual and physical machines to show the generalization of the model. Then, we will use virtual machine (VM) and physical machine (PM) in later sections for our experiments on a specific platform.

A. Resource sharing model

The resource sharing among virtual containers in a physical container is managed by the container manager or hypervisor. A general model is adopted by container managers to apportion resources to the virtual containers that share a common resource pool with a fixed capacity. That is, each container has parameters that specify the range of platform resource it can take, and how excess capacity is shared among virtual containers under contention. These resource sharing parameters are:

- 1) **min** - container is guaranteed at least **min** capacity when it has a workload to run.
- 2) **max** - total utilization of a container cannot exceed the **max**, even if free capacity is available.
- 3) **share** - competing containers are allocated excess capacity in proportion to their **share**.

This resource sharing model is the most common one and several variations to this model are realized in virtual container controllers including IBM hypervisor [2] and VMWare [3]. This simple model allows us to explain the intricate aspects of batch job scheduling without being constrained by the product specific aspects of these hypervisors.

These parameters are explained with reference to Figure 2. Figure 2(a) shows a virtual container A hosted on a physical platform with a unit capacity (e.g. CPU capacity). The three parameters described above are shown in these figures with sample values. The **min**=0.25 is the guaranteed physical capacity assigned to A. The **max**=0.67 is the maximum physical capacity that container A would be allowed to use for its jobs. When **min**=0 and **max**=1, container A is allowed to exploit the full capacity of the parent container but at the same time it may not get any capacity when there is contention from other containers sharing the parent platform. To avoid this situation, it is common to set **min**> 0. The third sharing metric is described with the help of Figure 2(b) which shows two containers A and B.

Figure 2(b) shows container B with a **min**=0.1 and **max**=1.0. The **share** of A is 4 and the **share** of B is 8. Containers with a higher **share** get higher priority under resource contention. Each container is guaranteed its **min** capacity. The remaining capacity, i.e., the capacity in excess of the guarantees to A and B, are apportioned to each container based on their **share**. In this example, the excess capacity $1 - (0.25 + 0.1) = 0.65$ are apportioned $\frac{4}{12} = \frac{1}{3}$ to A and $\frac{2}{3}$ to B. Accordingly, the total capacity of A consists of its **min** (0.25) and its share from excess capacity $0.65 * \frac{1}{3}$, making it 0.46, while B gets $0.1 + 0.65 * \frac{2}{3} = 0.54$. The apportioned capacities are smaller than their assigned maximum, therefore they do not need to be re-computed as described later.

In the presence of multiple containers with different parameters that host jobs with finite CPU demands, the estimation of the free capacity using the described sharing model becomes more complex. To systematically estimate the capacity, we first introduce a new metric called *equilibrium capacity* (EC).

B. Equilibrium Capacity (EC)

Container Equilibrium Capacity (EC) is the worst case guaranteed capacity that a container would be allocated when every co-located container competes for its fair share. EC is a container property derived from the sharing policies implemented by the container manager. It is independent of dynamic system parameters such as current workload and

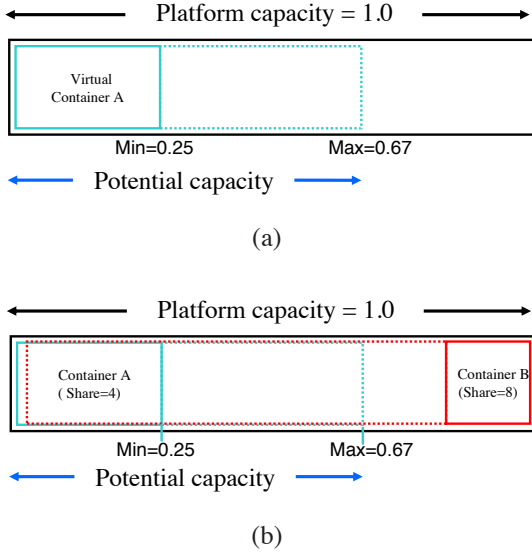


Figure 2. Features of container resource sharing

utilization. A simple expression to compute this metric is ideal but no such expression exists and this section describes the recursive algorithm to compute this for arbitrarily complex situations.

Consider a set of containers VM_1, \dots, VM_m that compete for a fixed capacity $pCap$ of the underlying platform PM . The container equilibrium capacity (EC_i), where $1 \leq i \leq m$ of a container VM_i is the amount of the shared resource that a container can receive when there is full contention from all competing containers.

To explain the concept and illustrate the computation issues of EC, Figure 3 shows the capacity available to two competing containers on a shared platform.

Consider the simple case when competing containers have

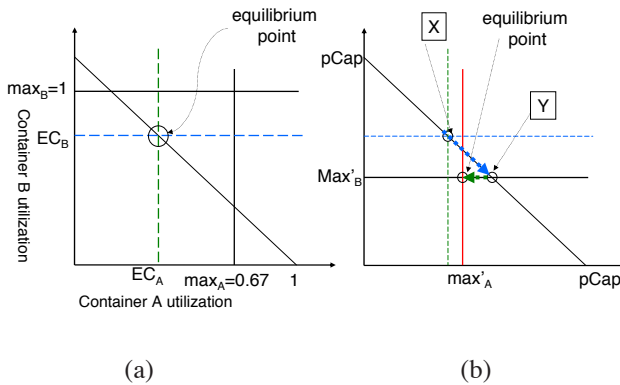


Figure 3. Container Equilibrium Points

$min \geq 0$ and $max = pCap$. The computation of EC is straightforward,

$$EC_i = min_i + \frac{share_i}{\sum_{k=1}^m share_k} pCap', \quad (1)$$

where $pCap' = pCap - \sum_{k=1}^m min_k$ and $1 \leq k \leq m$. That is, each container gets its minimum capacity min_k plus a proportion of the excess capacity $pCap'$, where the proportion is decided by the ratio between its share and the sum of the share of all competing containers.

Applying the above expression to containers A and B in Figure 2 results in $EC_A = 0.46$ and $EC_B = 0.54$. This is the same value derived in the previous section but this expression can be applied to multiple containers. In this case, the EC of both A and B is smaller than that of their corresponding maximum values as shown in Figure 3(a).

This straightforward calculation breaks down when some containers' equilibrium capacity calculated using the above expression is larger than their max parameters. This means that the calculated EC is not a valid estimation of the capacity of those containers. Consider the two-virtual-container example and Figure 3(b). Now assume that the max parameter of B, max_B is lowered so that it is smaller than its equilibrium capacity ($max'_B < EC_B$). Since a container is not allowed to consume more capacity than its max parameter, its equilibrium capacity should be constrained by the max parameter. As result of this new constraint, container A can get more capacity than was estimated above.

More formally, when $EC_i \geq max_i$ for container i , the difference $d = EC_i - max_i$ is the capacity this container cannot use and therefore its $EC_i = max_i$. The difference d should be summed up for all such containers and redistributed among the other containers whose $EC_i < max_i$. When this calculation is repeated to distribute this excess capacity, it may happen that some containers get more capacity than their maximum. As a result the procedure needs to be repeated until no container has a EC bigger than its max value.

Now referring back to Figure 3(b), following equation 1, at first the equilibrium point is the top-left point, X. Because the calculated value $EC_2 > max_2$, we shall set $EC_B = max_B$ and recalculate EC_A with container A sharing the excess resource $1 - max_B$ by itself. Thus, the equilibrium point moves to the point Y on the right. However, we have $EC_A > max_A$ at point Y. Thus, we repeat the procedure and set $EC_A = max_A$. The equilibrium point finalizes at the point in the middle. For this example, there is excessive capacity for the system that can not be used by neither A nor B. The complexity to compute the EC converges to $O(m)$, where m is the number of co-located containers.

The main insight is that EC is a lower limit, which is larger than min , of the amount of the platform resource that is guaranteed to a container. Although EC seems like promising metric to use for scheduling jobs, since it is a

Utilization of A (u_A)	PC of B (PC_B)
$u_A \leq EC_A$	$\max\{pCap - u_A, max_B\}$
$u_A > EC_A$	EC_B

Table I
PC CALCULATION FOR 2 CONTAINER EXAMPLE

pessimistic estimate, scheduling based on it does not result in optimal system utilization. Therefore, we need additional metrics that provide optimistic bounds on the achievable performance.

C. Container Potential Capacity (PC)

The container *potential capacity* (PC) is the maximum amount of the underlying platform resources that a container can receive in a system state. Unlike EC, PC is dependent on the system state, such as the current utilization of the competing containers.

We illustrate the concept of PC through examples. Let two containers A and B share and compete for a fixed size platform resource $pCap$. Let EC_A and EC_B be their EC. Recall that EC defines the amount of resource a container should get with full contention. Let u_A and u_B be their current utilization.

Next, we analyze how much resources container B can receive in the example. Consider the case where A uses less than or equal to its entitled amount. That is, its utilization is less than or equal to its EC, i.e., $u_A \leq EC_A$. In this case, B can grab the remaining capacity $pCap - u_A$ with only the limit of max_B , making it $\max\{pCap - u_A, max_B\}$. In the other case, when A uses more than its entitled amount $u_A > EC_A$, which is possible when B does not have enough workload and has $u_B < EC_B$. Then, if necessary, B can take resource from A and force its utilization to decrease to its equilibrium capacity EC_A . Thus, B can receive its own equilibrium capacity EC_B since it is a full contention situation. By the above discussion, we show the value of PC_B in Table I given the utilization u_A .

In summary, the actual capacity a virtual container is allocated is at least its EC and at most its PC. While EC is a fixed property of the container depending on resource sharing model and sharing parameters, PC also depends on the dynamic system state. As co-located virtual containers become heavier loaded, PC tends towards the EC.

III. APPLICATION TO SCHEDULING PROBLEMS

One way to measure the significance and effectiveness of the new metrics described in the previous section is in their application to traditional batch job scheduling problems, but where the compute nodes are VMs using a shared resource model. Though we now assign jobs to VMs, the scheduling objective is still that we either (a) maximize throughput and/or (b) minimize the response time of jobs relative to

their execution time when they have exclusive access to the resources.

The exclusive model, in which the number of concurrently active jobs in an OS image is unity is widely used, especially for HPC applications. The advent of multi-core processors motivates a return to increased job or task concurrency to improve the utilization of these resources. However, the method by which multiple jobs are executed on physical nodes is likely to be through the shared VM model of Figure 1. In this model exclusivity applies at the VM level, i.e. a VM is either free or hosting a single job.

The batch scheduling studies presented here are relevant to two classes of jobs; serial or single task jobs, and loosely coupled parallel jobs, also called embarrassingly parallel jobs. In the latter, the component tasks have no synchronization interactions and proceed independently of each other. Implementations such as Hadoop often design tasks that proceed independently, pulling work from a central queue until the queue is exhausted. As with serial jobs, the tasks benefit from greedy approaches to scheduling studied here.

In contrast, tasks in many HPC applications have synchronization barriers and proceed in lockstep. This means the job progress is limited to that of the slowest task and assigning job tasks greedily leads to unused cycles on the most powerful nodes. Scheduling algorithms for tightly coupled applications is left to future work.

A. Applying PC and EC Metrics to Scheduling

The two metrics PC and EC are evaluated by using each of them as the primary heuristic of a greedy scheduling algorithm. The corresponding algorithms are referred to as the *EC Algorithm* and *PC Algorithm* for the remainder of the paper. The procedure of assigning a job to a VM using each algorithm is illustrated in Figure 4. Here, the i^{th} VM on a machine A is denoted as A_i . The VMs which are free to accept jobs are sorted in descending order of the EC or PC metric.

The VM selection process for the *EC* algorithm (Figure 4(a)) is straightforward because EC is a fixed property of a VM so that starting or terminating a job on a collocated VMs does not change the EC values. Each arriving job is assigned to the first VM from the sorted EC list using the First Come First Serve (FCFS) policy. Because there is no secondary metric to break ties between VMs with identical EC it is possible for contiguous sequences of VMs on a common PM to have the same rank. This situation leads to collocation of jobs on the same PM which limits the extent to which VMs can borrow unused machine CPU cycles. Intuitively, this situation is only expected at low utilization where the physical machines are unoccupied. In our investigations (e.g. Figure 7) this is only a consideration with cluster utilization less than a few percent.

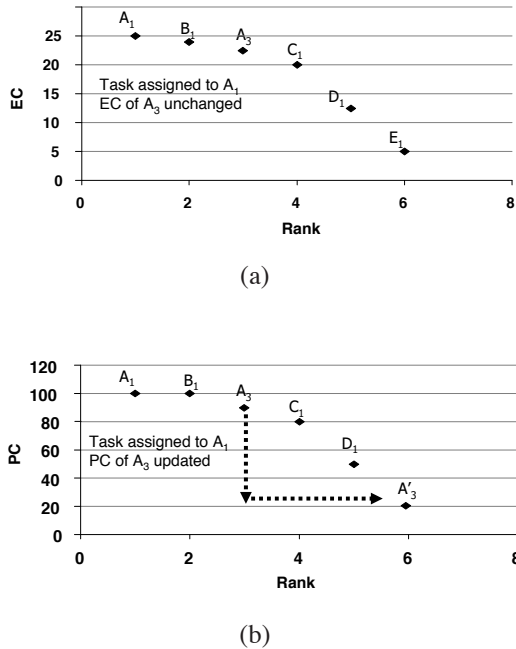


Figure 4. VM ranking order for scheduling algorithms using (a) EC and (b) PC

The PC algorithm of Figure 4(b) is more complex than EC because job start or termination events changes the PC metric of collocated VMs. This is indicated in the figure where assignment of a job to A_1 causes a change in the PC of A_3 . Because of the new assignment, PC of the co-located VM A_3 is reduced. As a result, VMs C_1 and D_1 are better choices than A_3 . When multiple VMs have identical values of PC, the EC metric is used as a secondary sort criteria in addition to PC. This makes sense because EC is a guaranteed lower bound on the CPU allocation a VM receives under contention, while PC is the achievable upper bound with current system state. When two VMs have identical PC the one with the higher EC is expected on average to provide more cycles to the job. The value of $(PC - EC)/EC$ for a VM is loosely interpreted as a measure of the risk of slowdown for currently running jobs when the system is busy. Thus A_1 is ranked first over B_1 even though their PC is equal, because A_1 has a higher EC (as shown in Figure 4(a)).

As an example, suppose three jobs are submitted to a system containing the resources shown in Figure 4(b). The first job to arrive is assigned to A_1 , the second to B_1 . After assigning a job to A_1 , the PC of A_3 is reduced and its new rank is represented as A'_3 as indicated in the figure by the dashed line. Consequently, the third highest ranking VM is now C_1 and so the third job to arrive is placed in C_1 . A similar reordering of the rankings of the VMs occurs on job completion events, which also triggers a re-evaluation of the PC of collocated VMs.

Intuitively, the PC algorithm leverages the advantage of

the elasticity and fair partitioning inherent to VM resources by capturing the dynamic capacity available to each VM. The EC algorithm supports the elasticity and is aware of the partitioning, but since the EC metric is unaware of the instantaneous capacity of the VMs, it is inferior at selecting the VM that will finish the job fastest.

B. Simulation and Evaluation Criteria

The PC and EC algorithms are compared using event driven simulation. Two workload traces are used, the Grid 5000 trace from the Grid Workloads Archive [5] and a second trace from the Cornell Theory Center (CTC) [6]. Trace characterization is available at the sites referenced above; the main difference between the two traces is that CTC jobs have a wider and longer range of run times than Grid'5000 jobs. Both traces include a mix of single node and parallel jobs, but the studies here use only the single node jobs. Where noted, the inter arrival time of jobs is uniformly scaled to increase the range of average cluster utilization over which algorithm performance is studied without modifying the characteristics of the jobs themselves.

The trace data contains job run time, but not the actual CPU demand of the jobs. (i.e. the *run time* reported in the trace tells us the wall clock execution time on the system that originally executed the job.) In a real system the CPU demand is not usually known to the scheduler until the job begins executing and the resource manager reports CPU utilization information. In traditional systems with a single job per physical node this issue is not relevant. We make the conservative assumption that each task is CPU intensive and therefore uses the full potential capacity available to it until it completes¹. We also assume that there is an inverse linear relationship between CPU allocation and execution time.

The data center modeled assumes all VMs have equal share values and the CPU limits are set to $\min=0$, $\max=1$. Each PM contains 4 VMs, thus $EC=0.25$ and $PC=1$ when no jobs are running on collocated VMs. We use a cluster size of 30 PMs to provide an average cluster utilization for the traces of around 50%. Apart from showing which algorithm is better at maximizing available resources, the average cluster utilization determines how much room for improvement there is. For example, if the cluster is fully utilized, the throughput of jobs will suffer for all algorithms. Any time a job is assigned to or released from a VM, the states of all active VMs residing on the PM of said VM are adjusted. (i.e. the PC of the VMs and the CPU utilization of active jobs are recalculated.) Other global cluster statistics, such as overall cluster utilization, are collected at these events as well.

Scheduling performance is measured using average (system) utilization and average (job) expansion factor² (XF).

¹For parallel jobs, there is the additional issue of task synchronization, but we leave that discussion for future work

²XF is often referred to as “makespan” or “stretch”

The average utilization is the average overall CPU utilization of the cluster from the first job arrival until the final job completion. XF is essentially the ratio of a job’s actual run time to its ideal run time; the ideal run time is the time it would take the job to execute if it were to run in isolation. Equation 2 shows the basic formula. The total job completion time used in the XF calculation has components from queuing (*queueTime*) and execution time expansion (*actualRuntime*). The normalized difference between *actualRuntime* and *idealRuntime* reflects the elasticity caused by sharing of resources among virtual machines. In a real system it would also include platform overhead which grows with the number of VMs/PM, but modeling the platform overhead is a topic we leave for future work.

$$XF = \frac{(queueTime + actualRuntime)}{idealRuntime} \quad (2)$$

To reduce the bias of XF for jobs of short duration, where short delays cause a large XF, the “bounded slowdown” method described in [4] is applied. Short running jobs are given a minimum *idealRuntime* to avoid small execution or queuing delays to result in large values when calculating the XF. Hence, the formula used for our experiments is shown in Equation 3. We use a *threshold* of 10 seconds.

$$XF = \frac{(queueTime + actualRunTime)}{\max(threshold, idealRuntime)} \quad (3)$$

C. Results

Figures 5 and 6 compare the ratio of XF for EC to PC (XF ratio) for the first 10,000 serial jobs of the Grid’5000 and CTC traces. This corresponds to about 154 and 103 days of job submission data, respectively. For the average expansion factor figures, the data are binned into equal time intervals and each data point represents the average XF of all jobs in that bin. The utilization varies according to the intensity of arrivals and state of the cluster and ranges from 0 (idle) to 1 (full utilization) in this trace segment. The PC metric performs approximately 2 to 4 times better than EC over the trace. The runtime contribution to the XF (i.e. the slowdown due to resource sharing) is expected to be between 1 and 4 for the homogeneous cluster model when there is little or no job queuing. This is because the EC algorithm has a tendency to pack jobs onto the same PM due to its lack of knowledge of the true dynamic capacity of the VMs residing in the PM. So depending on the current state of job assignments in the cluster, EC has a larger chance of assigning jobs to collocated VMs where they may receive as little as 1/4 as many CPU cycles. The utilization data of all figures are from the runs of the PC metric and are within a few percent of the EC numbers. This result is somewhat surprising given the difference in XF. However, consider a simple scenario in which a 1 second job is dispatched on a single machine followed by a 10 second job. The total

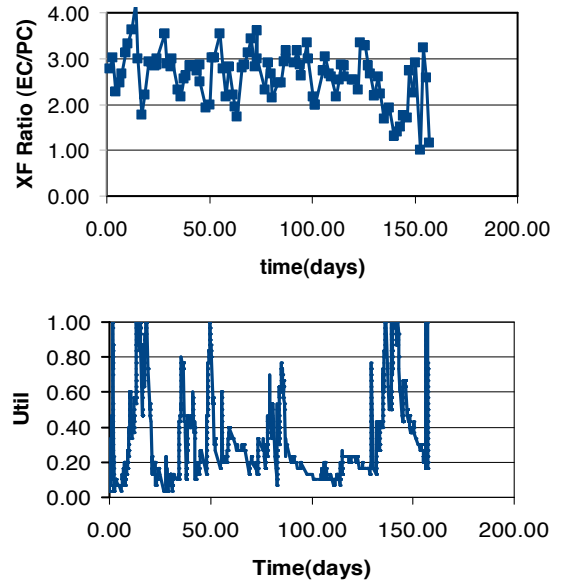


Figure 5. Ratio of XFs(top) and utilization (bottom) as function of job arrival time for Grid’5000 trace.

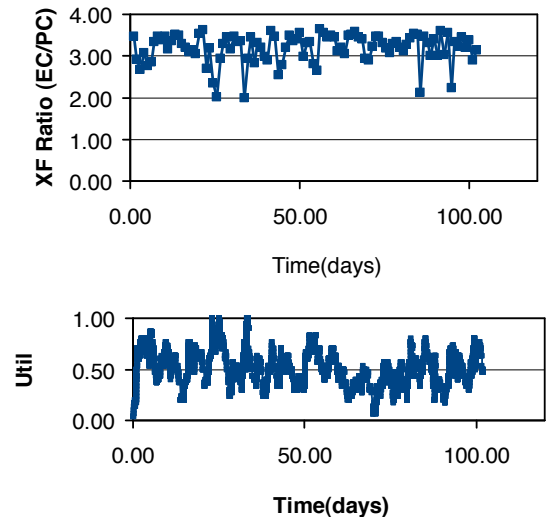


Figure 6. Ratio of XFs(top) and utilization (bottom) as function of job arrival time for CTC trace.

execution time is 11s and the XF of the jobs is 1 and 1.1 respectively. dispatching the jobs in the reverse order now causes an XF of 10 and 1 for the short and long job.

In the asymptotic conditions of low and high utilization, similar performance is expected from PC and EC. At low cluster utilization, the PC and EC algorithms should perform similarly. However, since EC performs no load balancing (i.e. selecting VMs on different PMs) it performs worse than PC even under very low load conditions. With n VMs

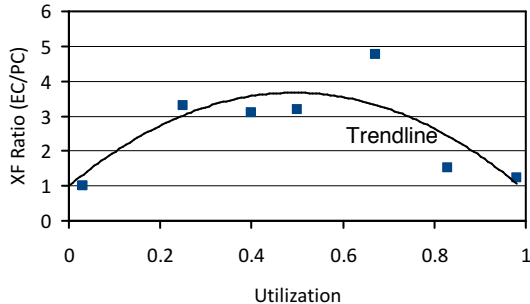


Figure 7. Ratio of XFs as function of average cluster utilization

per PM the frequency of occurrence of jobs assigned to collocated VMs falls as the utilization drops below $1/n$. As cluster utilization approaches 1, the VMs on each PM are generally occupied, there is little resource sharing, and all jobs receive the EC proportion of CPU. For other utilization's the queuing element of XF can play a role depending on the details of job arrivals and execution time and the intricacies of the scheduling algorithm.

In figures 5 and 6, the relationship between utilization and XF ratio is unclear. This is because the utilization depends on multiple job- and system- related properties. Also, the figures do not show the effect of queuing. In some cases, even when the utilization for both algorithms is almost the same, their expansion factors may vary greatly due to the XF bias for small jobs. e.g. if one algorithm queues a short-duration job and the other a long-duration job, the former will have a much higher XF.

Our hypothesis about the relationship between utilization and XF ratio are studied more in the experiments of Figure 7. Here, the average cluster utilization is varied by multiplying the inter-arrival times of the CTC trace by factors of 0.5, 0.55, 0.75, 1, 1.25, 2, and 16. This allows comparison of PC to EC as a scheduling metric under different intensity of job arrivals. The 'trend' line in the figure shows the intuitive expectations of the XF ratio discussed above. Generally the XF ratio improvement is less than 4 as expected for runtime expansion, but the data at util=0.65 (trace compression of 0.75) showed an XF ratio improvement for PC of 4.7. Closer inspection reveals that this is due to 3 intervals during the simulation in which the EC algorithm queued over 20 jobs while the PC algorithm did little or no queuing. As previously mentioned, the performance of the two algorithms should be the same at low utilization. However, since the EC algorithm tends to pack jobs on the same PM, utilization would need to be very low (i.e. under 3%) to reach this point. Instead, we used an EC algorithm with load balancing for the left-most point in Figure 7.

IV. RELATED WORK

Virtualization is a popular subject of research and experimentation in the last few years with the maturing of software and hardware support. While there is concern on the overhead of virtualization, the emergence of clouding computing propels new interest of virtualization for batch applications in addition to web service transactions. Studies of the impact of virtualization to data-centers includes [7], [8], [9]. Unlike these works, we introduce a new set of VM capacity metrics and study applicability to job scheduling of computing resources in clustered environments.

Evaluation of job scheduling requires representative workloads of large data-centers. Feitelson pioneered building archives of workload traces from different HPC sites with large clusters [10]. The research group at the University of Delft have collected another set of workload traces from the grid communities [5]. The workload traces are used widely by job scheduling researchers and engineers. Our paper also used the traces from these collections as input to our simulation, though these workloads represented input to job scheduling systems with physical resource requirements. Moreover, it is important to note that we filtered out all parallel jobs from the traces. Serial jobs are prominent in commercial workloads. However, most commercial workload traces are not readily available for research propose.

The solution presented in this paper is not specific to a particular virtualization technology; it uses the generic constructs for partitioning resources into virtual entities using virtualization technologies such as provided by IBM PowerVM [2], VMWare [3], Citrix Xen [11] and KVM [12].

There are few studies on the placement algorithms and performance of virtual machines mapped to physical machines [13], [14], [15]. Particularly, the work by Stillwell [16] et al. addresses some scheduling issues. Stillwell's work proposes new online 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 previous work, we focused only on job placement. More significantly, as far as we know, we are first to propose a set of new metrics in estimating the dynamic potential capacities of virtual containers and use these metrics for job placement.

V. CONCLUDING REMARKS AND FUTURE WORK

Virtualization offers improved resource utilization by allowing sharing of resources among virtual containers. This sharing makes system state more dynamic, so batch job scheduling algorithms need to account for this dynamic system state. Two new metrics, potential capacity (PC) and equilibrium capacity (EC), of resource capacity that incorporate the dynamic, elastic, and sharing aspects of collocated virtual containers is introduced for the first time. Greedy scheduling algorithms are developed based on these

two new metrics for efficient job placement and to achieve optimal system utilization. We show by simulation that the algorithms based on the new metrics can improve the resource utilization and reduce expansion factor for serial batch jobs selected from the Grid'5000 and CTC workload traces.

In this paper, we present simulation data only on fixed number of virtual to physical machines and uniform virtual machine sharing attributes. Variable number of virtual to physical machines and nonuniform virtual sharing attributes are the subject of our ongoing work.

Our future work will also focus on parallel jobs and expect that the dynamic and sharing aspects of virtual machines present unique challenges and complexity for parallel job scheduling. For example, the execution rate of parallel tasks of a job with frequent synchronization will be limited by the task assigned to a virtual machine with smallest capacity. The calculation of potential capacity (PC) will need to consider such "ripple effect". By applying the our new metrics, we will study the efficiency of different algorithms, in combination with other well-known scheduling algorithms such as back-fill and reservation. In this context, we desire to further study the performance characteristics of assigning jobs of different priorities or classes to different classes of virtual machines.

This paper considered CPU utilization as an example metric for sharing, but VMs can share memory, I/O, and network bandwidth in a similar manner. The algorithms for CPU can be applied easily to these other resources. Applying them to memory poses a unique challenge in that memory allocation and subsequent de-allocation for strict enforcement of fairness results in cost. The cost is in the form of CPU utilization for clean up and the benefit is better application performance with greedy memory allocation. This cost and benefit need considering for memory resources. Note that this is not an issue in CPU, I/O and network allocations because for practical purposes they are stateless resources so the re-allocation cost is negligible with an always positive benefit for the application.

Finally, although this paper focuses on traditional batch scheduling in which the cluster configuration is predefined, the metrics are relevant to new scheduling issues and paradigms such as non-batch or interactive workloads, or dynamic instantiation of new containers and migration of existing work. For example, the available cycle range of $PC - EC$ provides useful input to decisions about adding containers or work when there is concern about container quality of service.

VI. ACKNOWLEDGMENTS

This work was supported in part by IBM, and the National Science Foundation under grants CNS-BPC-AE-1042341, CNS-MRI-R2-0959985, HRD-CREST-0833093, and NSF-OISE-PIRE-0730065.

REFERENCES

- [1] R. Creasy, "The origin of the vm/370 time-sharing system," in *IBM Journal of Research and Development*, 1981.
- [2] "IBM: Advanced power virtualization on ibm system p5." [Online]. Available: <http://www.redbooks.ibm.com/abstracts/sg247940.html>
- [3] "VMWare: Virtualized basics." [Online]. Available: <http://www.vmware.com/virtualization/virtual-machine.html>
- [4] 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.
- [5] "Grid Workload Archive." [Online]. Available: <http://gwa.ewi.tudelft.nl/pmwiki/>
- [6] "Cornel Supercomputer Center." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/1_ctc_sp2/index.html
- [7] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Proceedings of the 20th annual international conference on Supercomputing*. NY, USA: ACM, 2006, pp. 125–134.
- [8] 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.
- [9] J. Brandt, F. Chen, V. De Sapio, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong, "Combining virtualization, resource characterization, and resource management to enable efficient high performance compute platforms through intelligent dynamic resource allocation," in *SMTPS Workshop in Conjunction with IPDPS*, 2010.
- [10] "Parallel Workload Archive." [Online]. Available: <http://www.cs.huji.ac.il/labs/parallel/workload/models.html>
- [11] "Xen server." [Online]. Available: <http://www.citrix.com/lang/English/home.asp>
- [12] "KVM." [Online]. Available: <http://www.citrix.com/lang/English/home.asp>
- [13] 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.
- [14] 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*. NY, USA: ACM, 2008, pp. 175–184.
- [15] 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.
- [16] M. Stillwell, F. Vivien, and H. Casanova, "Dynamic fractional resource scheduling for hpc workloads," in *IPDPS*, 2010.