A multi-dimensional job scheduling

Mehdi Sheikhalishahi[c,*], Richard M. Wallace[b], Lucio Grandinetti[a],
José Luis Vazquez-Poletti[b], Francesca Guerriero[c]
a *Department of Electronics, Comp. Sci. & Sys. University of Calabria, Rende(CS), Italy*
b *Department of Computer Arch. & Automation Complutense University, Madrid, Spain*
c *Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende(CS), Italy*

Abstract

With the advent of new computing technologies, such as cloud computing and contemporary parallel processing systems, the building blocks of computing systems have become multi-dimensional. Traditional scheduling systems based on a single-resource optimization, like processors, fail to provide near optimal solutions. The efficient use of new computing systems depends on the efficient use of several resource dimensions. Thus, the scheduling systems have to fully use all resources. In this paper, we address the problem of multi-resource scheduling via multi-capacity bin-packing. We propose the application of multi-capacity-aware resource scheduling at host selection layer and queuing mechanism layer of a scheduling system. The experimental results demonstrate performance improvements of scheduling in terms of *wait time* and *slowdown* metrics.

1. Introduction

From a scheduling and resource view for computing, there can be a few major issues and problems to consider: low utilization, overloaded systems, poor performance, and resource contention. Solving these issues and problems requires answering complex questions that start with, ‘‘*When*...’’, ‘‘*Which*...’’, and ‘‘*Where*...’’. For instance, ‘‘When should some workloads be migrated to other servers?’’, ‘‘Which types of applications should be consolidated together in a server?’’, and ‘‘Where should a workload be placed?’’These examples are the type of resource management questions to consider and this list has many more resource management questions of this type.

Scheduling algorithms based on *First-Come First Served* schemes (FCFS) pack jobs from the job queue into the system in order of their arrival until a resource is exhausted. If there is a large job at the head of the queue which requires more resources than those left available in the system, the job allocation scheme is blocked from scheduling further jobs until sufficient resources become available for this large job. This results in potentially large resource frag-ments being *under-utilized*. *Back-filling* mechanisms overcome this issue by skipping over jobs that cannot be allocated and by finding smaller jobs that can make use of remaining resources.

Back-filling in addition to being a scheduling algorithm is a queueing mechanism, that is it changes the order of the jobs in the queue for a faster allocation of jobs which can be allocated to the computing system.

With the advent of new computing technologies such as cloud computing as a recent development in the field of computing and massively parallel processing systems such as the most recent *Cray JK7* system (Titan), the *Chinese Tianhe-1A* system (NUDT YH MPP), (http://top500.org/lists/2012/11/) and the quite old *SUN E10000* and *SGI O2K* systems, the building blocks of computing systems have become multi-dimensional. The *Titan* system is installed at Oak Ridge, achieving 17.59 Petaflop/s on the Linpack benchmark with 560,640 processors, including 261,632 *NVIDIA K20x* accelerator cores (https://www.olcf.ornl.gov/titan/). From the processing point of view, according to the *Top500* list, a total of 62 systems on the *Top500* list are using accelerator/co-processor technology including *Titan* and the *Chinese Tianhe-1A* system which use *NVIDIA GPUs* to accelerate its computation.

Moreover, *Stampede* and six other super-computers are accelerated by the new *Intel Xeon Phi* processors (Intel MIC architecture - https://www.tacc.utexas.edu/stampede/). As a result there are multiple computing elements to be taken into account in scheduling at the processor level.

Scheduling techniques in older computer systems, such as the massively parallel processing systems *TMC CM-5* and the *CRAY T3E*, were focused on a single resource dimension allocation (processing nodes) where single capacity bin-packing algorithms were used to solve this problem(http://www.top500.org/system/166997).

In multi-dimensional resource environment a single resource still becomes exhausted while others remain under-used even with the *back-filling* strategy as the scheduling algorithm. This is due to the design of *FCFS* algorithms which are restricted in job selection based on their arrival order and not addressing capac-ity imbalance between resources in a multi-resource environment. *Back-filling* strategy is an instance of FCFS mechanism. Thus, single capacity bin-packing algorithms are inadequate as they are unable to provide optimal scheduling for multi-dimensional resources of *CPU, GPU, memory, shared memory, large disk farms, I/O channels, bandwidth, network input*, *network output*, and even *software li-censes* of current computing system architectures.

Therefore, the scheduling scheme must be free to select any job based on matching all of the jobs' resource requirements with the available system resources in order to address the efficient use of resources in a multi-resource environment. The target of efficient use of new computing architectures depends on efficient usage of all resource dimensions with the scheduling algorithm fully using all resources.

In this paper, we research *multi-resource scheduling* by model-ing the problem as a *multi-capacity bin-packing* problem. First, we propose a new host selection policy for each job based on multi-capacity criteria, that happens at the lowest layer of scheduling (the last fate of job placement). Then, we propose a multi-capacity-aware queuing mechanism. We model multi-resource scheduling as a multi-capacity bin-packing scheduling algorithm at the queue level to reorder the queue in order to improve the packing and as a result to improve scheduling metrics.

The approach was first presented in [1] as part of ARMS-CC-2014 workshop [2]; in which we presented a multi-resource queuing mechanism to provide a higher degree of consolidation in multi-dimensional computing systems. In this paper, we extend the approach at the host selection policy to provide a scheduling system aware of multi-dimensional resources. In summary, our paper makes the following contributions:

- A proposal for scheduling problem based on multi capacity bin packing algorithms.
- A proposal for host selection and queuing mechanism based on multi-capacity bin-packing scheduling algorithm.
- We show experimentally that our multi-resource scheduling system performs more efficiently than the state of the art scheduling system based on back-filling policy as measured by *waittime* and *slowdown* metrics.

The remaining part of this paper is organized as follows. Sec-tion 2 reviews related work. Section 3 presents our multi-resource scheduling approach that is modeled based on a multi-capacity bin-packing algorithm. Then, it details the design of a host selection policy and a queuing mechanism based on multi-capacity bin-packing algorithm. Section 4 explains detailed design and implementation issues such as workload traces, and resource model for experiments of this paper. After that, it discusses simulation experiments and experimental results. Finally, Section 5 presents our conclusions and future work.

2. Related work

Single- and multi-capacity bin-packing problems and their connection to the generalized scheduling problem have been studied in [3–7]. The two-dimensional vector packing problem [7] consists in orthogonally packing a subset of a set of rectangular-shaped boxes, without overlapping, into a single bounding rectangular area, maximizing the ratio between the area occupied by the boxes and the total available area.

The *d*-capacity bin-packing solution approaches extend the single capacity bin-packing solutions, i.e., *First-Fit* (FF), *Next-Fit* (NF), and *Best-Fit* (BF), to deal with the *d*-capacity jobs (items) and nodes (bins). FF, NF, and BF are considered as *Job-To-Node* placement rules. Those *d*-capacity bin-packing algorithms that are extensions of the single capacity bin-packing do not scale well with increasing *d* since they do not take advantage of the information in the additional capacities. The work by Bobroff et al. [8] presents a first-fit approximation algorithm for the bin packing problem. The algorithm was devised for the single resource problem, but tips are given about the extension to multiple resources. Orthogonal to the *Job-To-Node* placement rules is the job queue preprocessing method used before the packing operation. For the single capacity bin-packing algorithm sorting the list based on a scalar value in a non-increasing order with respect to the job resource requirement improves the packing performance. The *First-Fit Decreasing* (FFD) algorithm first sorts the list in a non-increasing order and then applies the FF packing algorithm. The NF and BF algorithms can be extended in a similar manner.

Leinberger et al. [9] proposed a *d*-capacity bin-packing algo-rithm named *Multi-Capacity Bin Packing* (MCBP). It is a particular vector packing algorithm that uses the additional capacity in-formation to provide better packing by addressing the capacity imbalance. The authors show how their algorithms lead to better multi-resource allocation and scheduling solutions.

In addition, the problem of optimally mapping virtual machines (VMs) to servers can be reduced to the *bin packing problem* [10–12]. This problem is known to be NP-hard, therefore heuristic approaches can only lead to sub-optimal solutions. With regard to re-cent work finding an FFD algorithm that has better execution time, the paper [13]

provides an algorithm that maximizes the dot product between the vector of remaining capacities and the vector of remaining or residual capacities of the current open bin, i.e. subtract from the bin's capacity the total demand of all the items currently assigned to it. It places the item that maximizes the weighted dot product with the vector of remaining capacities without violating the capacity constraint vector of demands for the item. This bin-centric method did show better performance. This method is an alternative to our method and is intended for allocation of VM images rather than scientific job placement. The argument can be made that a VM image can have the same processing footprint as a long-lived scientific application.

Moreover, novel job scheduling mechanisms use *d*-capacity bin-packing algorithms. For instance, [14,15] employ an algorithm based on MCBP proposed by Leinberger et al. in [9]. In [14], a novel job scheduling approach for homogeneous cluster computing platforms is proposed. Its key feature is the use of VM technology to share fractional node resources in a precise and controlled manner. Other VM-based scheduling approaches have focused primarily on technical issues or extensions to existing batch scheduling systems, while in [14] authors take a more aggressive approach and seek to find heuristics that maximize an objective metric correlated with job performance. They derive absolute performance bounds and develop algorithms for the online, non-clairvoyant version of scheduling problem. Their results demonstrate that virtualization technology coupled with lightweight online scheduling strategies can afford dramatic improvements in performance for executing high performance computing (HPC) workloads.

Eco4cloud [16] adaptively consolidates the workload using VM migration and balances the assignment of *CPU*- and *RAM*-intensive applications on each server, which helps to optimize the use of re-sources. Live migration of VMs between servers is adopted by the VMware Distributed Power Management system, using lower and upper utilization thresholds to enact migration procedures [17]. The heuristic approaches presented in [11] and in [12] use techniques derived from the Best Fit Decreasing and the First Fit De-creasing algorithms, respectively. In both cases, the goal is to place each migrating VM on the server that minimizes the overall power consumption of the data center. On the other hand, consolidation is a powerful means to improve IT efficiency and reduce power consumption [18–20]. Some approaches – e.g., [21,22] – try to forecast the processing load and aim at determining the minimum number of servers that should be switched on to satisfy the demand, so as to reduce energy consumption and maximize data center revenues. However, even a correct setting of this number is only a part of the problem: algorithms are needed to decide how the VMs should be mapped to servers in a dynamic environment, and how live migration of VMs can be exploited to unload servers and switch them off when possible, or to avoid SLA violations. In [23] the multi-resource scheduling problem is tackled by using a linear programming (LP) formulation that gives higher priority to VMs with more stable workload. *ReCon* [24] is a tool that analyzes the resource consumption of various applications, discovers applications which can be consolidated, and subsequently generates static or dynamic consolidation recommendations. In *ReCon*, only CPU utilization is considered, the complete extension to the multi-resource problem is left to future research.

In comparison to our approach, all aforementioned works are coupled with advanced technologies, like virtualization, to im-prove scheduling metrics. Our approach is based on optimization techniques to improve pure scheduling metrics with simple heuristics.
The framework presented in [25] tackles the consolidation problem by exploiting constraint programming paradigm.

Rule-based constraints concerning SLA negotiation are managed by an optimizer that adopts a branching approach: the variables are considered in a priority descending order, and at each step
one of the variables is set to the value that is supposed to guide the solver to a good solution.

The Entropy resource manager presented in [26] performs dynamic consolidation based on constraint program-ming, where constraints are defined on CPU and on RAM utilization. All these approaches represent important steps ahead for the deployment of green-aware data centers, but they do not model multi-resource aspects of scheduling in their problem completely.

In a game theory allocation model by Ye and Chen [27] they model cluster and cloud computing server load balancing by consolidation of virtual machines (VMs) on physical machines (PMs) by use of non-cooperative game theory for VMs on multi-dimensional resource allocations. They use ''selfish agents'' for each VM where the agent can decide to lessen its payoff by relocating the VM to another PM working toward a pure Nash equilibrium measuring the inefficiency of equilibria by the price of anarchy and the price of stability.

Using linear programming Ferreto, et al. [28] to manage virtual machines consolidation onto a single physical server. Server consolidation involves VM migration, which has a direct impact on service response time. Most of the existing solutions for server consolidation rely on eager migrations, which try to minimize the number of physical servers running VMs. These solutions generate unnecessary migrations due to unpredictable workloads that require VM resizing. The linear programming formulation and heuristics are used to control VM migration prioritizing virtual machines with steady capacity using the TU-Berlin and Google data center workloads to compare their migration control strat-egy against existing eager-migration-based solutions. Their results showed that avoiding migration of VMs with steady capacity re-duces the number of migrations with minimal penalty in the number of physical servers.

An interesting use of Hopfield Neural Network and a ''Jar of Stones'' allocation scheduling method is discussed by Taheri et al. in [29]. This scheduling method depends on a random selection of job metrics for allocation of jobs to systems. While Taheri et al. have a successful performance improvements against the other scheduling methods compared to, they do have issues on the makespan estimation and the method has excessive movement of program and data files among the cluster/cloud systems. Our paper has a more proactive allocation mechanism that is not based on random selection.

A fixed bin packing scheduling method is described by Abawajy [30] uses a two level space-sharing policy (as contrasted to the time-sharing policy of our paper) where the request is either to reschedule a job or to do a reassignment of a job to new resources. In the Modified Adaptive Policy (MAP), a variation of the well-known adaptive scheduling policy, when determining the partition size, the policy considers both running and waiting jobs while the contribution of the scheduled jobs.
By simply disallowing a job to hold onto the processors when it has no use of it, the performance of the jobs can be improved.

Our multi-resource scheduling approach is in line with consolidation approaches in such a way to increase the number of allocated workloads to a node. With that we increase the consolidation degree of nodes leading to improvement of resources utilization, and consequently improving energy efficiency.

In this paper, we exploit MCBP optimization in scheduling system architecture. Our approach is similar to [31], in which the authors use autonomic computing to improve resource contention metric, while we use MCBP optimization to improve packing and as a result improve *waittime* and *slowdown* performance metrics.

3. Multi-resource scheduling

In this section, first we review scheduling system architecture. Then, we introduce the multi-capacity bin-packing problem. We then devise the basics of a multi-capacity bin-packing algorithm to address the problem of multi-resource scheduling. After that, we develop this algorithm as part of the host selection policy and the queuing mechanism of the scheduler.

3.1. Scheduling architecture

Scheduling architecture consists of front-end policies, core scheduler mechanisms, information service, and back-end policies. Fig. 1 summarizes these components in a layered architecture. It shows how a job makes its way through the scheduling system.
Some components of a distributed system scheduler are:

- Front-end policies. Admission control and pricing are placed in this component. Job requests pass via these policies before queuing.
- Core scheduler. A *queue mechanism* and various scheduling algorithms such as back-filling.
- Information service. A scheduler slot table, availability window providing information to be exploited by other scheduling components.
- Back-end policies. *Host selection*, mapping and pre-emption policies constitute back-end policies.

First, a job passes through admission control policy to see if it can get through the scheduling system. If this policy determines that the job can be accepted, then it goes through queue mechanism; that is the entry point of a job into the scheduler. The queue mechanism reorders the queued jobs and pass them one by one to the core scheduling algorithm. In this paper, the ordering criteria is based on a multi-capacity bin packing algorithm.

The core scheduling algorithm uses a host selection policy for the job placement on the most suitable node that has been defined by the policy. In [32], we proposed effective energy aware consolidation policies based on host selection policy. In this paper, we develop multi-capacity bin packing host selection policy for multi-dimensional resource environments.

To address the challenges of multi-dimensional computing systems (capacity imbalance in this paper), we use multi-capacity bin-packing technique. Different layers of scheduling systems can incorporate the MCBP optimization technique to address capacity imbalance at some extent. In this paper, a queueing mechanism and host selection policy have been incorporated into this technique to address capacity imbalance. The queueing mechanism has a more global view on capacity imbalance with respect to the host selection policy. Therefore the queueing mechanism jobs will be reordered to have a more balanced distribution of capacity based on a nodes' status. In other words, this technique minimizes capacity imbalance by moving those jobs that increase the capacity imbalance for resources to the tail of the queue. This improves the packing and as a result improves scheduling metrics.

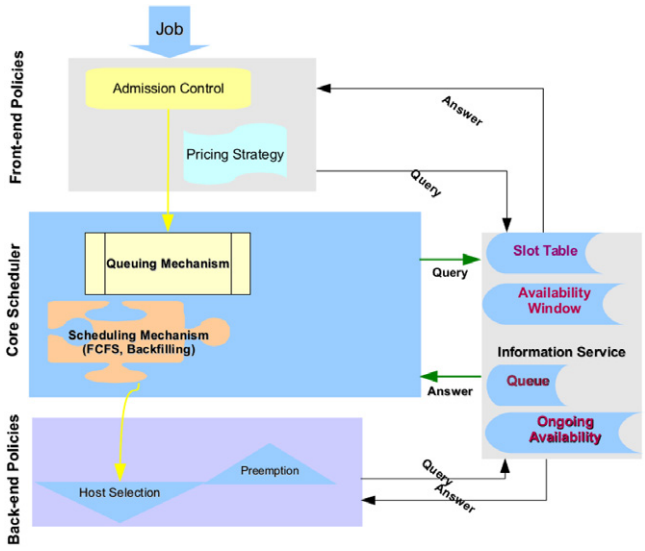


Fig. 1. Scheduling system reference architecture.

3.2. The multi-capacity bin-packing problem

Due to multiple resource dimensions in computing systems, resource allocation problem is related to the *multi-dimensional bin-packing*, or vector packing. Vector packing is bin-packing with multi-dimensional items and bins. In order to model the parallel job scheduling problem as a multi-capacity bin-packing problem the parallel system node is represented by a bin with d capacities, e.g. Bk , corresponding to the multiple resources in the system. And a job (item) is represented by a d-capacity, e.g. Ji , resource requirements vector. Jobs are obtained from a list L, and the total number of jobs to be packed is denoted by n.

In a homogeneous computing system, the capacity of each node is represented by a d-capacity vector, $C = (C_1, \ldots, C_j, \ldots, C_d)$, where C_j, $C_j \geq 0$, represents the jth component capacity, so that $\sum_j C_j > 0$. A job is also represented by a d-capacity vector $Ji = (Ji1, \ldots, Jij, \ldots, Jid)$, where Jij, $0 \leq Jij \leq Cj$ denotes the jth component requirement of the ith job, and $\forall i \mid 1 \leq i \leq n$ and $\sum_j J_{ij} > 0$. Bk represents node k. A job Ji can be packed into a node (bin) if there is enough free capacity for all resources in node Bk , if $Bk + \leq C$, or $\forall j \mid 1 \leq j \leq d$ and $Bkj + Jij \leq Cj$, i.e., Bk for job Ji placement.

The FF algorithm tries to fit the next job to be placed into any of the currently non-empty nodes. If the next job cannot fit into any of the current nodes, then the next node is considered. Or, if it does not fit into any of the nodes, it will return to queue and it will be considered at the next scheduling cycle. The BF algorithm adds a further node selection heuristic to the FF algorithm by scheduling the best-fit job from the queue on a node which minimizes unused resources.

The NF algorithm takes the next d-capacity job Ji and attempts to place it in the current node Bk . If it does not fit: If $Bkj + Jij > Cj$ for some j, then the next node $B\ k+1$ is considered. The point being that no node that does not meet the condition Bl , $1 \leq l < k$ is considered as a candidate for job Ji .

In d-capacity formulation the jobs are sorted based on a scalar representation of the d components; that is the summation of d components. Other extensions include the maximum component, sum of squares of components, etc. The goal is to somehow capture the relative size of each d-capacity item.

3.3. A heuristic to the multi-capacity bin-packing problem

Bin-packing in the computing system scheduling domain is basically an abstraction of a restricted batch processing scenario in which all jobs arrive before processing begins and all jobs have the same execution time. The goal is to process the jobs as fast as possible. Basically, each bin corresponds to a scheduling cycle on the system resources, and the scheduling algorithm must pack jobs onto the system in an order such that all jobs are scheduled using the fewest cycles. Thus, the scheduling goal is to partition the list L into as few nodes (bins) as possible.

At the start of a scheduling cycle, a bin is created in which each component is initialized to reflect the amount of the corresponding machine resource which is currently available. Jobs are selected from the job queue (list L) and packed into the machine until there are not sufficient quantities of resources to fill the needs of any of the remaining jobs.

The prior *Job-To-Node* placement rules described in Section 2 fails to provide a near optimal scheduling solution. For example, in the FF algorithm the node selection mechanism for job placement ignores the resources' requirement (weights) for the job and the current component capacities of the nodes and its only criteria for job placement is that the job fits. Hence, a single capacity of a node may fill up sooner than the other capacities, which leads to lower overall utilization. Based on this analysis, a *Job-To-Node* placement would provide more optimized packing if the current relative weights or rankings of d-capacities are considered; that is, if Bkj has the lowest available capacity, then search for a job$J\,i$ which fits into$B\,k$ and has Jij as its smallest component weight. This reduces pressure on Bkj, which may allow additional jobs to be added to the node$B\,k$. This heuristic attempts to correct a *capacity imbalance* in the node. Thus, the capacities are all kept balanced, so that more jobs will likely fit into the node which gives a multi-capacity aware approach and is the basis of this paper.

Our proposed heuristic attempts to find jobs in which the largest components are exactly ordered with respect to the ordering of the corresponding smallest elements in the current node. For instance, in the case of $d = 5$ with the capacities of the current node $B\,k$ ordered as follows:

$Bk1 \leq Bk3 \leq \mathrm{Bk4} \leq Bk2 \leq Bk5$.

In this instance, the algorithm would first search the list L for a job in which the resource requirements were ranked as follows:

$Ji1 \geq Ji3 \geq Ji4 \geq Ji2 \geq Ji5$

which is exactly opposite of the current node state.

Adding $J\,i$ to $B\,k$ has the effect of increasing the capacity levels of the smaller components more than it increases the capacity levels of the larger components. If no jobs were found with this relative ranking between their components, then the algorithm searches the list again, relaxing the ordering of the smallest components first, working up to the largest components. For example, the next two job rankings that would be searched for are:

$Ji1 \geq Ji3 \geq Ji4 \geq Ji5 \geq Ji2$

and

$Ji1 \geq Ji3 \geq Ji2 \geq Ji4 \geq Ji5$

and finally,

$Ji5 \geq Ji2 \geq Ji4 \geq Ji3 \geq Ji1$.

The algorithm searches each logical sub-list in an attempt to find a job which fits into the current node. If no job is found in the current logical sub-list, then the sub-list with the next best ranking match is searched, and so on, until all lists have been searched.

In summary, these heuristics match jobs to hosts, based on sorting the host resources according to their capacity, and the jobs requirements in the opposite order, such that the largest requirement would correspond to the highest capacity.

3.4. Multi-capacity host selection policy

We extend the proposed multi-capacity bin-packing technique to develop multi-capacity-aware host selection policy. More specifically, a host selection policy
calculates a score to reflect how much a job matches a node if it is going to be scheduled at a specific starting time on that node. The score specifies the ability of the node to accept the job at that starting time.

The heuristic measures the matching of a job resource requirements against a node's free capacity. Thus, it sorts the node's resources based on free capacities (from the highest free capacity to the lowest one). Then, it goes over the node's resource ordering to evaluate how much the job resource requirements match based on this ordering by the summation of the difference between job re-source requirements. This summation reflects the node feasibility degree from the capacity imbalance point of view for the job (an attempt to correct a capacity imbalance).

The pseudo code of multi-capacity aware host selection pol-icy is represented in Algorithm 1.

Algorithm 1 Multi-capacity-host-selection-policy(Job j, Time t, Node n)

job_req = {} **<Populating job resource requirements into job_req dictionary with CPU, MEM, IO, NETIN, NETOUT keys**.>
node_free_capacity_norm = {}
aw = *slottable.get_availability_window(t)* **<Populating resource capacities of a node using a ‘‘slottable" data structure. This will be used for capacity normalization based on resources capacities of a node.>**
for *res* ∈ *job_req.keys()* **do**
end for *node_capacity*[*res*] = *slottable.nodes*[*n*].*capacity.get_by_type(res)*

for *res* ∈ *job_req.keys()* **do**
node_free_capacity_norm[*res*] =
end for *aw.get_availability(t, n).get_by_type(res)/node_capacity*[*res*]
<Sorting node resources based on the free resource capacity, in descending order.>
node_free_capacity_norm.sort()
node_free_capacity_ordering = [res for res, capacity in *node_free_ capacity_norm*]
score = 0 **<Evaluating the job score.>**
for *r*1 ∈ *node_free_capacity_ordering* **do**
del *node_free_capacity_ordering*[0]
for *r*2 ∈ *node_free_capacity_ordering* **do**
then if *node_free_capacity_norm*[*r*1] > *node_free_capacity_norm*[*r*2]
end if *score*+ = *job_req*[*r*1] − *job_req*[*r*2]
end for
end for
return score

The data structures used in Algorithm 1 are:

• *t* as an input parameter is the next scheduling cycle.

• A *slot table* is essentially just a collection of resource reservations. It tracks the capacity of the physical nodes on which jobs can be scheduled, contains the *resource reservations* of all the jobs, and allows efficient access to them.

• A particularly important operation with the slot table is determining the ‘‘*availability window*’’ of resources starting at a given time. Availability window provides easier access to the contents of a slot table by determining the availability in each node starting at a given time.

• *getAvailabilityWindow* function creates an availability window starting at a given time.

In brief, an availability window provides a convenient abstrac-tion over the slot table, with methods to answer questions such as:

• ‘‘If I want to start at least at time *T*, are there enough resources available to start the job?’’

• ‘‘Will those resources be available until time *T* + *t*?’

• ‘‘If not, what is the longest period of time those resources will be available?’’

and so on.

3.5. Multi-capacity queuing mechanism

In this section, we focus on queuing mechanism of scheduling system. We extend the proposed packing technique of the multi-capacity bin-packing algorithm developing a multi-capacity-aware queuing mechanism.

The queuing mechanism heuristic orders jobs based on the free capacity ordering of nodes. Free capacities at the next scheduling cycle are considered by sorting the resources of the nodes based on their free capacity; that is, from highest to lowest free capacity.

The mechanism then processes the ordered list of resources for the nodes and evaluates the best match of job resource requirements to a node by summing the differences between job resource requirements. This summation reflects the degree to which it is feasible to use a node based on the capacity imbalance for a job. This step attempts to correct a capacity imbalance.

The pseudo code of multi-capacity-aware queuing mechanism is represented in Algorithm 2.

Algorithm2. MultiCapacityQueuingMechanism(t: the next scheduling cycle)

```
multi_capacity_queue = [] <Traversing the wait queue,
selecting jobs and calculating their score against node
values.>
while Queue is not empty do
   <Get the job at the head of the queue.>
   job = queue.dequeue()
   score = 0 <Traversing all nodes and evaluating the job
score.>
node)        for node ∈ slottable.nodes.keys() do score        =
   score+ Multi-capacity-host-selection-policy(job, t,
   end for
   <Placing the job and its score in a new queue that is
based on multi-capacity priority.>
   multi_capacity_queue.append((job, score))
end while
<Sorting the multi-capacity queue based on the job score
in descending order
and moving the best matched jobs to the head of the
queue.> multi_capacity_queue.sort()
multi_capacity_queue = [l for (l, s) in multi_capacity_queue]
<Copying the multi-capacity queue into the wait queue.>
for l ∈ multi_capacity_queue do queue.enqueue(l)
end for
```

4. Experiments

In this section, we present some simulation experiments to evaluate the performance of multi-capacity-aware scheduling system in terms of scheduling metrics. For that, we first describe resource model and workload characteristics, then we present workload traces explored. In closing we give specific and precise configuration used.

4.1. Resource model and workload characteristics

We consider commodity cluster infrastructure as resource model in this study; each physical node has CPU, Memory, IO, Net-work input, and Network output as resource types and conventional interconnection between them. Furthermore, the simulated cluster of a configuration is modeled after the corresponding workload trace's cluster.

4.2. Workload traces

For this paper, we construct workloads by adapting the *SDSC Blue Horizon* cluster job *submission* trace5from the *Parallel Workloads Archive*. We alter these derived traces to incorporate all resource dimensions requirements. For that, we simply add *Net-in*, *Net-out*, and *IO* resource types to jobs resource requirements that were missing, and set their resource requirements based on a random uniform distribution to present a random use of resources for jobs.

This is to present multi-dimensional resource requirements for jobs. We treat all resource types the same so that a job can be allocated to a node if all its resources requirements will be satisfied by the node.

4.3. Configurations

We conduct a number of experiments over a wide range of de-rived traces. We extract all *30*-day traces from *SDSC Blue Horizon* to build derived traces for our experiments. Specifically, the extract is from the beginning of day *300* until day *330*. This would be trace one. From day *330* to day *360* would be trace two, and so on. In sum, we build 21 derived traces from day 300 until day 960 in increments of 30 days. For each trace, we carry out two experiments: one for the multi-capacity-aware scheduling (MCBP), and the other for back-filling queuing policy (BKFL).

In addition to the variable parameters, we have fixed parameters such as an *intermediate back-filling* strategy as the packing mechanism. Thus, the scheduling function periodically evaluates the *queue*, using an *intermediate back-filling* algorithm to deter-mine whether any job can be scheduled. In sum, we compare a multi-capacity-enabled back-filling queuing policy against a pure back-filling queuing policy. In all experiments, host selection has been defined based on the multi-capacity host selection policy.

4.4. Results

In simulation experiments, we explore the impact of the multi-capacity-aware queuing mechanism on the *waittime*, and *slowdown* metrics. We performed experiments on the 21 derived workload traces of *SDSC Blue Horizon* according to configurations above.

For each experiment, for each job, we collected time values: *ta*, the arrival time, or time when the job request is submitted; *ts*, the start time of the job; and *te*, the time the job ends. At the end of an experiment, we compute the following metrics:

- *Waittime*: This is time $ts - ta$, the time a job request must wait before it starts running. The time units are in *minute*.
- *Slowdown*: If *tu* is the time the job would take to run on a dedicated physical system, the job's *slowdown* is $(te - ta)/tu$. If *tu* is less than 10 s, the *slowdown* is computed the same way, but assuming *tu* to be 10 s [33].

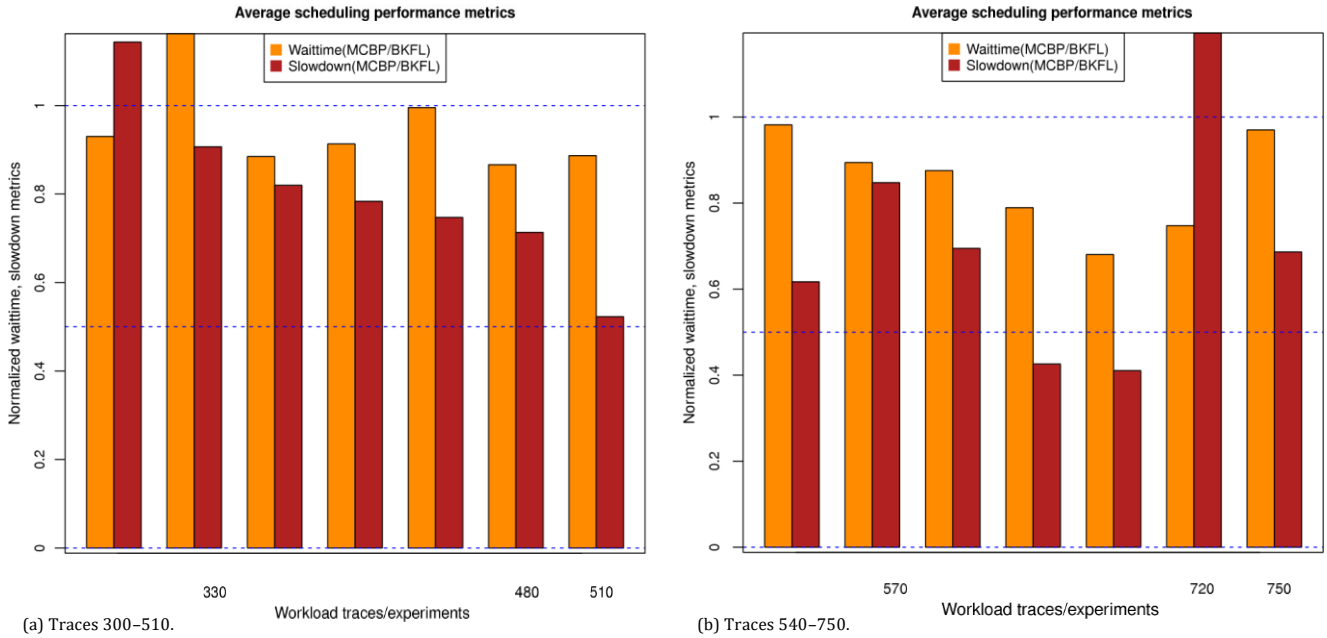
The optimization of these two metrics is a *minimization* prob-lem. We analyze simulation results for each experiment based on mean and standard deviation statistics measure. Mean statistics are illustrated in Figs. 2 and 3, and standard deviations are illus-trated in Figs. 4 and 5. In order to compare the two policies, i.e. **MCBP** and **BKFL**, we normalize **MCBP** results to **BKFL** results, i.e., *MCBP/BKFL*. All values presented in the graphs are based on this normalized value. The transformation of a policy to a value allows a better numeric presentation and objective comparison.

In general, we get better results (smaller standard deviation) for both metrics in terms of mean statistic measure. However, in terms of standard deviation *waittime* metric has a higher deviation value for MCBP policy, while *slowdown* gets a smaller deviation. While on average, we got better results for *waittime* and *slowdown* metrics, we have more discrepancy on *waittime* values for MCBP policy with respect to BKFL policy. Nonetheless, we got more concentrated values for *slowdown* metric for MCBP policy.

This observation implies that in total we have better scheduling with MCBP policy with respect to BKFL policy. This means that with MCBP total jobs get allocated to the system faster (as it is also demonstrated with statistics measure over all experiments in Table 1). In sum, MCBP outperforms BKFL policy in terms of both scheduling metrics.

In addition, Table 1 presents the outcome of mean and standard deviation statistics measures over all experiments. These results demonstrate that the multi-capacity-aware approach provides a consistent performance improvement

over the back-filling one. More specifically, in total we have **54** minutes improvement for the ***waittime*** metric, and **18** unit improvement of the ***slowdown*** metric.



(a) Traces 300–510. (b) Traces 540–750.

Fig. 2. Average of simulation results for experiments 300–750.

Table 1
Statistics measures of simulation results over all experiments.

(a) Mean

Config	Waittime (min)	Slowdown
BKFL	324.58	48.03
MCBP	270.31	30.02

(b) Standard deviation

Config	Waittime (min)	Slowdown
BKFL	606.40	149.24
MCBP	723.23	133.29

5. Conclusions and future work

The building blocks of contemporary computing systems are multi-dimensional. Therefore, architecture of these systems and algorithms which deal with these systems have to take into ac-count this shift from single-dimension resource model. In this paper, we considered scheduling aspects of such systems. Traditional scheduling systems based on single-resource optimization cannot provide optimal solutions. As a result, the efficient utilization of new computing systems depends on the efficient use of all re-source dimensions. The scheduling systems have to fully utilize all resources. To address this problem, we have proposed a multi-resource scheduling system based on multi-capacity bin-packing.

Through exhaustive simulation experimentation on 21 derived workload traces of SDSC Blue Horizon, we have demonstrated that the multi-capacity aware scheduling system addresses multi-dimensional scheduling aspects of computing system resources to achieve improved *waittime*, *slowdown*.

In addition, this approach provides better consolidation degree, that is, it increases the number of allocated workloads to a node leading to an improvement of resources utilization, and energy efficiency. In this paper, we conducted experiments with homogeneous systems based on realistic simulation while our multi-capacity aware scheduling system can support more general instances.

In addition, our solution can be integrated with real frameworks, like Nimbus Toolkit, OpenStack and OpenNebula resource managers.

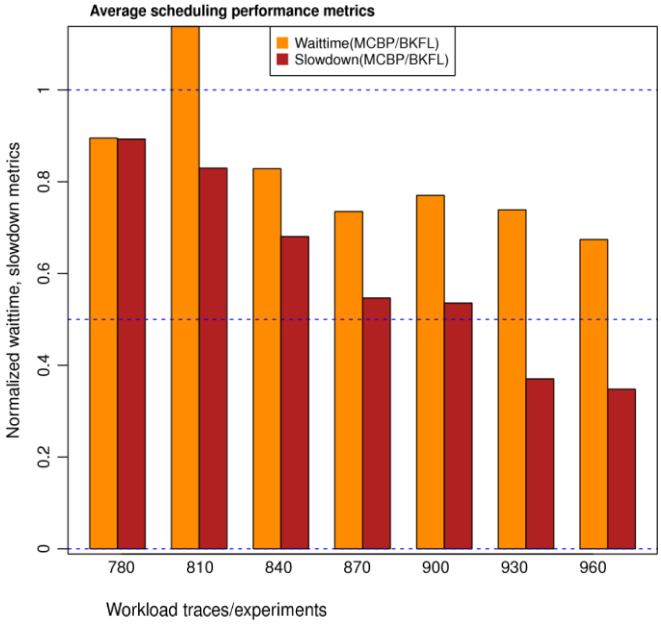
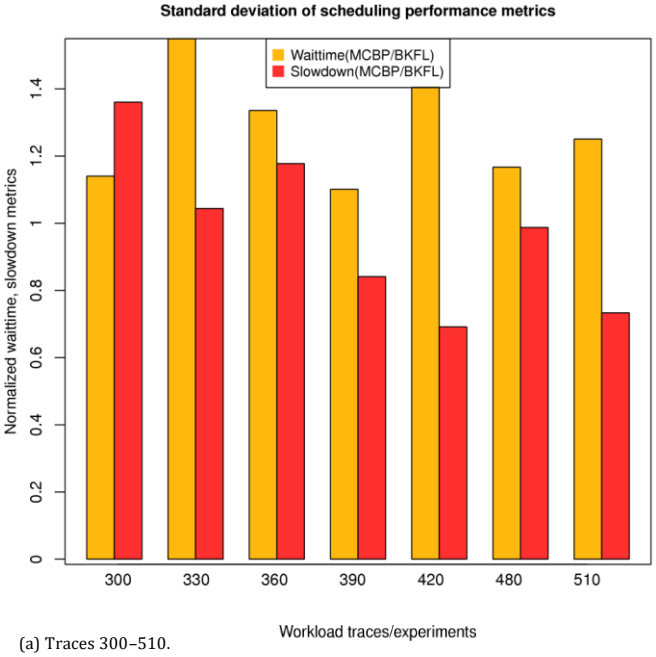
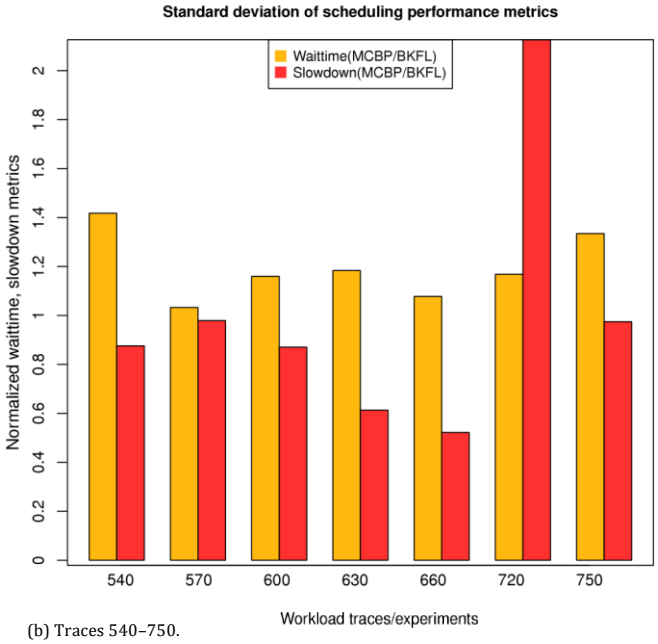


Fig. 3. Average of simulation results for experiments 780–960.

In this paper, we studied multi-capacity aware scheduling system for each single job. We can apply this heuristic on a group of jobs to address capacity imbalance. For example, as a future work we plan to study how to schedule a group of jobs at the queue based on the multi-capacity heuristic.



(a) Traces 300–510.



(b) Traces 540–750.

Fig. 4. Standard deviation of simulation results for experiments 300–750.

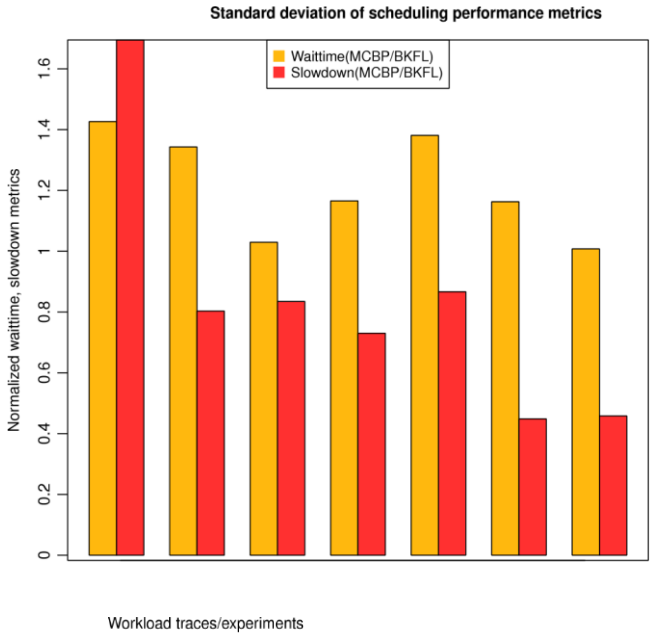


Fig. 5. Standard deviation of simulation results for experiments 780–960.

Acknowledgments

This work was partially performed under the auspices of the Spanish National Plan for Research, Development and Innovation under Contract TIN2012-31518 (ServiceCloud). We acknowledge Carlo Mastroianni from the Italian National Research Council, and Tapasya Patki from the University of Arizona for reviewing this paper

References

[1] M. Sheikhalishahi, R.M. Wallace, L. Grandinetti, J.L. Vazquez-Poletti, F. Guer-riero, A multi-capacity queuing mechanism in multi-dimensional resource scheduling, in: F. Pop, M. Potop-Butucaru (Eds.), Adaptive Resource Man-agement and Scheduling for Cloud Computing, in: Lecture Notes in Com-puter Science, Springer International Publishing, 2014, pp. 9–25. URL: http://link.springer.com/chapter/10.1007/978-3-319-13464-2_2

[2] Adaptive Resource Management and Scheduling for Cloud Computing—First International Workshop. URL: http://www.springer.com/computer/ book/978-3-319-13463-5.

[3] E.G. Coffman, M.R. Garey, D.S. Johnson, An application of bin-packing to multiprocessor scheduling, SIAM J. Comput. 7 (1) (1978) 1–17.

[4] E.G. Coffman, M.R. Garey, D.S. Johnson, Dynamic bin packing, SIAM J. Comput. 12 (2) (1983) 227–258.

[5] M.R. Garey, R.L. Graham, Bounds for multiprocessor scheduling with resource constraints, SIAM J. Comput. 4 (2) (1975) 187–200.

[6] M.R. Garey, R.L. Graham, D.S. Johnson, Resource constrained scheduling as generalized bin packing, J. Combin. Theory Ser. A 21 (3) (1976) 257–298.

[7] Y.-L. Wu, W. Huang, S.-c. Lau, C.K. Wong, G.H. Young, An effective quasi-human based heuristic for solving the rectangle packing problem, European J. Oper. Res. 141 (2) (2002) 341–358. URL: http://ideas.repec.org/a/eee/ejores/ v141y2002i2p341-358.html.

[8] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing SLA violations, in: 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007, pp. 119–128. http://dx.doi.org/10. 1109/INM.2007.374776.

[9] W. Leinberger, G. Karypis, V. Kumar, Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints, in: 1999 International Conference on Parallel Processing, 1999. Proceedings, 1999, pp. 404–412. http://dx.doi.org/10.1109/ICPP.1999.797428.

[10] A. Verma, P. Ahuja, A. Neogi, pMapper: power and migration cost aware application placement in virtualized systems, in: V. Issarny, R.E. Schantz (Eds.), Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference, Leuven, Belgium, December 1–5, 2008, Proceedings, in: Lecture Notes in Computer Science, vol. 5346, Springer, 2008, pp. 243–264.

[11] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.

[12] D.M. Quan, R. Basmadjian, H. de Meer, R. Lent, T. Mahmoodi, D. Sannelli, F. Mezza, L. Telesca, C. Dupont, Energy efficient resource allocation strategy for cloud data centres, in: 26th Int. Symp. on Computer and Information Sciences, ISCIS 2011, London, UK, 2011, pp. 133–141.

[13] Panigrahy, Talwar, Uyeda, Wiede, Heuristics for vector bin packing, 2011. http://research.microsoft.com/apps/pubs/default.aspx?id=147927.

[14] M. Stillwell, F. Vivien, H. Casanova, Dynamic fractional resource scheduling vs. batch scheduling, IEEE Trans. Parallel Distrib. Syst. 99 (2011) (PrePrints). http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.183.

[15] M. Stillwell, D. Schanzenbach, F. Vivien, H. Casanova, Resource allocation algorithms for virtualized service hosting platforms, J. Parallel Distrib. Comput. 70 (2010) 962–974. http://dx.doi.org/10.1016/j.jpdc.2010.05.006.

[16] C. Mastroianni, M. Meo, G. Papuzzo, Probabilistic consolidation of virtual machines in self-organizing cloud data centers, IEEE Trans. Cloud Comput. 1 (2) (2013) 215–228. http://dx.doi.org/10.1109/TCC.2013.17.

[17] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, X. Zhu, Vmware distributed resource management: design, implementation, and lessons learned, VMware Tech. J. (2012) URL: http://labs.vmware.com/publications/ vmware-technical-journal.

[18] M. Cardosa, M.R. Korupolu, A. Singh, Shares and utilities based power consolidation in virtualized server environments, in: Proceedings of the 11th IFIP/IEEE Integrated Network Management, IM 2009, Long Island, NY, USA, 2009, pp. 327–334.

[19] P. Graubner, M. Schmidt, B. Freisleben, Energy-efficient virtual machine consolidation, IT Prof. 15 (2) (2013) 28–34. http://dx.doi.org/10.1109/MITP. 2012.48.

[20] K. Schröder, W. Nebel, Behavioral model for cloud aware load and power management, in: Proc. of HotTopiCS'13, 2013 International Workshop on Hot Topics in Cloud Services, ACM, 2013, pp. 19–26. http://dx.doi.org/10.1145/ 2462307.2462313.

[21] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, N. Gautam, Managing server energy and operational costs in hosting centers, SIGMETRICS Perform. Eval. Rev. 33 (1) (2005) 303–314.

[22] M. Mazzucco, D. Dyachuk, R. Deters, Maximizing cloud providers' revenues via energy aware allocation policies, in: 10th IEEE/ACM Int. Symp. on Cluster Computing and the Grid, CCGrid 2010, Melbourne, Australia, 2010, pp. 131–138.

[23] T. Ferreto, M. Netto, R. Calheiros, C. De Rose, Server consolidation with migration control for virtualized data centers, Future Gener. Comput. Syst. 27 (8) (2011) 1027–1034.

[24] S. Mehta, A. Neogi, ReCon: a tool to recommend dynamic server con-solidation in multi-cluster data centers, in: Proc. of the Network Opera-tions and Management Symposium, IEEE NOMS 2008, 2008, pp. 363–370. http://dx.doi.org/10.1109/NOMS.2008.4575156.

[25] K. Dhyani, S. Gualandi, P. Cremonesi, A constraint programming approach for the service consolidation problem, in: Int. Conf. on Integration of AI and OR Techniques in Constraint Programming, CPAIOR'10, Bologna, Italy, 2010, pp. 97–101.

[26] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: Proc. of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2009, ACM, 2009, pp. 41–50. http://dx.doi.org/10.1145/1508293.1508300.

[27] D. Ye, J. Chen, Non-cooperative games on multidimensional resource allocation, Future Gener. Comput. Syst. 29 (6) (2013) 1345–1352. http://dx.

doi.org/10.1016/j.future.2013.02.004. URL: http://www.sciencedirect.com/ science/article/pii/S0167739X13000320.

[28] T.C. Ferreto, M.A. Netto, R.N. Calheiros, C.A.D. Rose, Server consolidation with migration control for virtualized data centers, Future Gener. Comput. Syst. 27 (8) (2011) 1027–1034. http://dx.doi.org/10.1016/j.future.2011.04.016. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11000677.

[29] J. Taheri, A.Y. Zomaya, P. Bouvry, S.U. Khan, Hopfield neural network for simultaneous job scheduling and data replication in grids, Future Gener. Comput. Syst. 29 (8) (2013) 1885–1900. http://dx.doi.org/10.1016/j.

future.2013.04.020. URL: http://www.sciencedirect.com/science/article/pii/ S0167739X13000800.

[30] J. Abawajy, An efficient adaptive scheduling policy for high-performance computing, Future Gener. Comput. Syst. 25 (3) (2009) 364–370. http://dx.

doi.org/10.1016/j.future.2006.04.007. URL: http://www.sciencedirect.com/ science/article/pii/S0167739X06000859.

[31] M. Sheikhalishahi, L. Grandinetti, R.M. Wallace, J.L. Vazquez-Poletti, Auto-nomic resource contention-aware scheduling, Softw. - Pract. Exp. (2013) 161–175. http://dx.doi.org/10.1002/spe.2223.

[32] M. Sheikhalishahi, I.M. Llorente, L. Grandinetti, Energy aware consolidation policies, in: International Conference on Parallel Computing, IOS Press, Gent, Belgium, 2011, pp. 109–116.

[33] D.G. Feitelson, L. Rudolph, Metrics and benchmarking for parallel job scheduling, in: D. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 1459, Springer, Berlin, Heidelberg, 1998, pp. 1–24. http://dx.doi.org/10.1007/BFb0053978.