A Multi-capacity Queuing Mechanism in Multi-dimensional Resource Scheduling

Mehdi Sheikhalishahi[1(✉)], Richard M. Wallace[2], Lucio Grandinetti[1], José Luis Vazquez-Poletti[2], and Francesca Guerriero[1]

[1] Department of Electronics, Computer Sciences and Systems, University of Calabria, Rende, CS, Italy
mehdi.alishahi@gmail.com
[2] Department of Computer Architecture and Automation, Complutense University, Madrid, Spain

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 algorithms based on a single-resource optimization like processor fail to provide near optimal solutions. The efficient use of new computing systems depends on the efficient use of all resource dimensions. Thus, the scheduling algorithms have to fully use all resources. In this paper, we propose a queuing mechanism based on a multi-resource scheduling technique. For that, 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 improve scheduling metrics. The experimental results demonstrate performance improvements in terms of waittime and slowdown metrics.

Keywords: Multi-resource · Queuing mechanism · Resource management · Scheduling · Bin-packing · Performance

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, "Which types of applications should be consolidated together in a server?", "When should some workloads be migrated to other servers?", 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

F. Pop and M. Potop-Butucaru (Eds.): ARMS-CC 2014, LNCS 8907, pp. 9–25, 2014.
DOI: 10.1007/978-3-319-13464-2_2

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 fragments 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.

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)[1], 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[2].

Scheduling 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[3].

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 uses *NVIDIA GPUs* to accelerate its computation. Moreover, *Stampede* and six other supercomputers are accelerated by the new *Intel Xeon Phi* processors (Intel MIC architecture)[4]. As a result there are multiple computing elements to be taken into account in scheduling at the processor level.

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 capacity 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 licenses* of current computing system architectures.

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. Therefore, 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.

[1] http://top500.org/lists/2012/11/
[2] https://www.olcf.ornl.gov/titan/
[3] http://www.top500.org/system/166997
[4] https://www.tacc.utexas.edu/stampede/

In this paper, we investigate research on *multi-resource scheduling* by modeling this problem as a *multi-capacity bin-packing* problem. We propose a queuing mechanism based on multi-resource scheduling technique. 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.

In summary, our paper makes the following contributions:

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

The remainder of this paper is organized as follows. Section 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 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 experimentations and experimental results. Finally, Sect. 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 [5,6,10,11,25].

The two-dimensional vector packing problem [25] 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. [2] 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 performance of the packing. 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. [15] proposed a *d*-capacity bin-packing algorithm named *Multi-Capacity Bin Packing* (MCBP). It is a particular vector packing algorithm that uses the additional capacity information to provide better packing by addressing the capacity imbalance. 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* [1,20,24]. This problem is known to be NP-hard, therefore heuristic approaches can only lead to suboptimal solutions. With regard to recent work finding a FFD algorithm that has better execution time [19] 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, [22,23] employ an algorithm based on MCBP proposed by Leinberger et al. in [15]. In [23], 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 [23] 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 resources. 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 [13]. The heuristic approaches presented in [1] and in [20] use techniques derived from the Best Fit Decreasing and the First Fit Decreasing 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 [3,12,21]. Some approaches - e.g., [4,17] - 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 [9] the multi-resource scheduling problem is tackled by using an linear programming (LP) formulation that gives higher priority to VMs with more stable workload. *ReCon* [18] 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, these works are coupled with advanced technologies, like virtualization, to improve scheduling metrics. Our approach is based on optimization techniques to improve pure scheduling metrics with simple heuristics. The framework presented in [7] 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 [14] performs dynamic consolidation based on constraint programming, 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.

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.

3 Multi-resource Scheduling

In this section, first we review bin-packing algorithms. 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 queuing mechanism of the scheduler.

3.1 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. $\overrightarrow{B_k}$, corresponding to the multiple resources in the system. And a job (item) is represented by a d-capacity, e.g. $\overrightarrow{J_i}$, 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, $\overrightarrow{C} = (C_1, ..., C_j, ..., C_d)$, where $C_j, C_j \geq 0$, represents the jth component capacity, so that $\sum_{j=1}^{d} C_j > 0$. A job is also represented by a d-capacity vector, $\overrightarrow{J_i} = (J_{i1}, ..., J_{ij}, ..., J_{id})$, where $J_{ij}, 0 \leq J_{ij} \leq C_j$, denotes the j-th component requirement of the ith job, and $\forall i \mid 1 \leq i \leq n$ and $\sum_{j=1}^{d} J_{ij} > 0$.

$\overrightarrow{B_k}$ represents node k. A job $\overrightarrow{J_i}$ can be packed into a node (bin) $\overrightarrow{B_k}$, if $\overrightarrow{B_k} + \overrightarrow{J_i} \leq \overrightarrow{C}$, or $\forall j \mid 1 \leq j \leq d$ and $B_{kj} + J_{ij} \leq C_j$, i.e., there is enough free capacity for all resources in node $\overrightarrow{B_k}$ for job $\overrightarrow{J_i}$ 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 $\overrightarrow{J_i}$ and attempts to place it in the current node $\overrightarrow{B_k}$. If it does not fit: If $B_{kj} + J_{ij} > C_j$ for some j, then the next node $\overrightarrow{B}_{k+1}$ is considered. The point being that no node that does not meet the condition $\overrightarrow{B_l}$, $1 \leq l < k$ is considered as a candidate for job $\overrightarrow{J_i}$.

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.

3.2 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 Sect. 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 B_{kj} has the lowest available capacity, then search for a job $\overrightarrow{J}_i$ which fits into $\overrightarrow{B}_k$ and has J_{ij} as its smallest component weight. This reduces pressure on B_{kj}, which may allow additional jobs to be added to the node $\overrightarrow{B}_k$. 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 $\overrightarrow{B}_k$ ordered as follows:

$$B_{k1} \leq B_{k3} \leq B_{k4} \leq B_{k2} \leq B_{k5}$$

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

$$J_{i1} \geq J_{i3} \geq J_{i4} \geq J_{i2} \geq J_{i5}$$

which is exactly opposite of the current node state. Adding $\overrightarrow{J}_i$ to $\overrightarrow{B}_k$ 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:

$J_{i1} \geq J_{i3} \geq J_{i4} \geq J_{i5} \geq J_{i2}$
and
$J_{i1} \geq J_{i3} \geq J_{i2} \geq J_{i4} \geq J_{i5}$
... and finally,
$J_{i5} \geq J_{i2} \geq J_{i4} \geq J_{i3} \geq J_{i1}$

The algorithm searches each logical sublist in an attempt to find a job which fits into the current node. If no job is found in the current logical sublist, then the sublist 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.

3.3 Multi-capacity Queuing Mechanism

In this paper, 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 bin-packing queuing mechanism.

Our multi-capacity bin-packing 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 transits the resource ordering for the nodes evaluating 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 the algorithm 1. t as an input parameter is the next scheduling cycle. Some description about the data structures used in the algorithm 1 are as the following:

- 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 abstraction 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.

4 Experiments

In this section, we present simulation experiments to evaluate the multi-capacity bin-packing queuing policy 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.

Algorithm 1. MultiCapacityQueuingMechanism(t: the next scheduling cycle)

```
multi_capacity_queue = []
job_res_req = {}
job_res_req[RES_CPU] = 0
job_res_req[RES_MEM] = 0
job_res_req[RES_IO] = 0
job_res_req[RES_NETIN] = 0
job_res_req[RES_NETOUT] = 0
node_free_capacity_norm = {}
node_free_capacity_res_ordering = {}
for res ∈ job_res_req.keys() do
    node_res_capacity[res] = slottable.nodes[1].capacity.get_by_type(res)
end for
aw = slottable.get_availability_window(t)
for node_id ∈ slottable.nodes.keys() do
    node_free_capacity_norm[node_id] = {}
    for res ∈ job_res_req.keys() do
        node_free_capacity_norm[node_id][res] =
        aw.get_availability(t, node_id).get_by_type(res)/node_res_capacity[res]
    end for
    node_free_capacity_norm_items[node_id] = node_free_capacity_norm[node_id].items()
    <Sorting resources for a node_id based on the free resource capacity, in
        descending order.>
    node_free_capacity_norm_items[node_id].sort()
    node_free_capacity_res_ordering[node_id] = [ res for res, capacity in
        node_free_capacity_norm_items[node_id]]
end for
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.>
    for node_id ∈ slottable.nodes.keys() do
        for r1 ∈ node_free_capacity_res_ordering[node_id] do
            del node_free_capacity_res_ordering[node_id][0]
            for r2 ∈ node_free_capacity_res_ordering[node_id] do
                if node_free_capacity_norm[node_id][r1] >
                    node_free_capacity_norm[node_id][r2] then
                    score+ = job_res_req[r1] − job_res_req[r2]
                end if
            end for
        end for
    end for
    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
```

4.1 Resource Model and Workload Characteristics

We consider commodity cluster infrastructure as resource model in this study each physical node has *CPU, Memory, IO, Network 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 trace[5] from 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. That is a job can be allocated to a node if all its resources requirements will be satisfied by the node.

4.3 Configurations

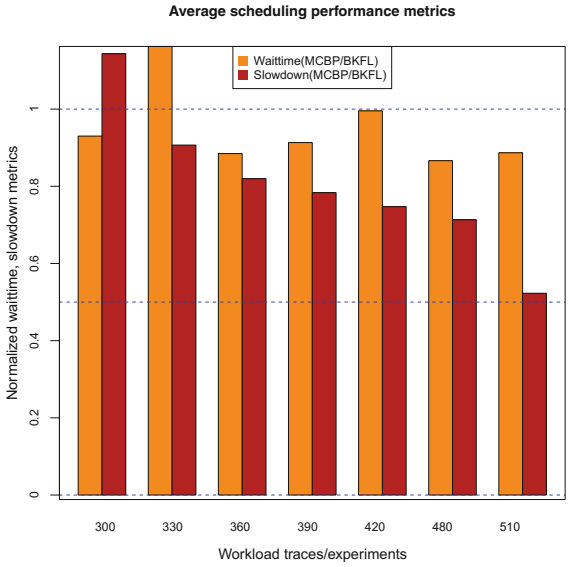
We conduct a number of experiments over a wide range of derived traces. We extract all *30*-day traces from *SDSC Blue Horizon* to build derived traces for our experiments. Specifically, the extract is from the beginning of day *300* until day *330*. This would be trace one. From day *330* to day *360* 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 queuing policy(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 determine 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.

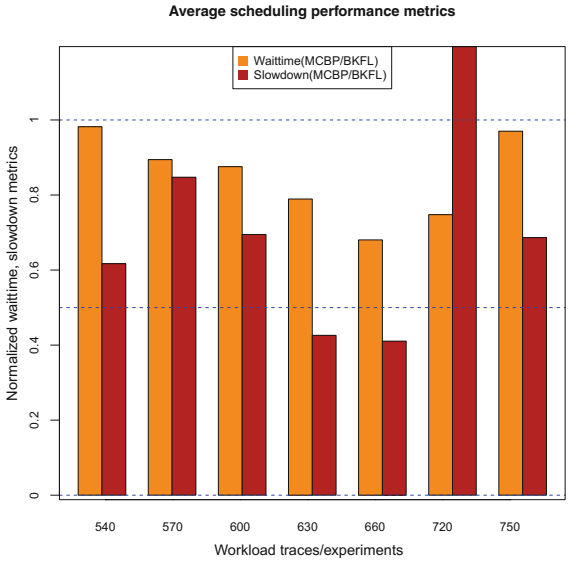
4.4 Results

In simulation experiments, we explore the impact of the multi-capacity bin-packing 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.

[5] http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_blue/index.html



(a) Traces 300 to 510



(b) Traces 540 to 750

Fig. 1. Average of simulation results for experiments 300 to 750

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

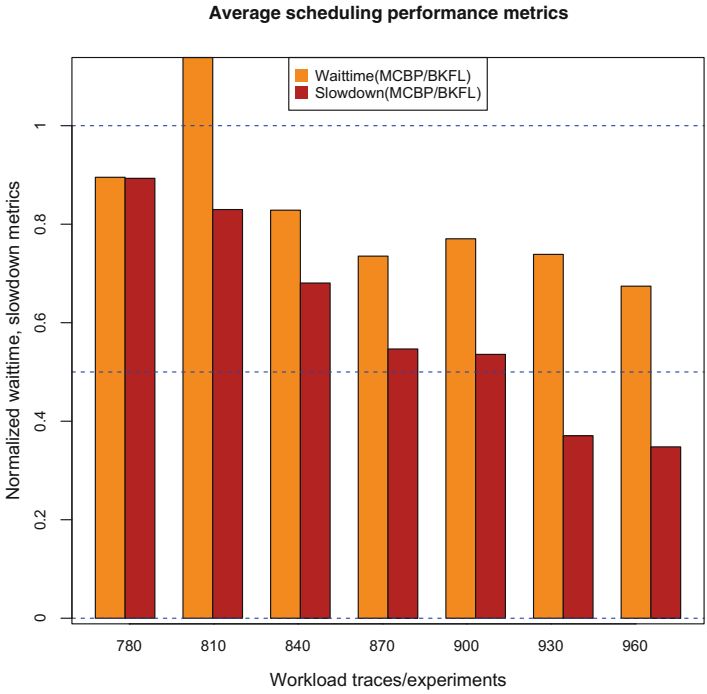
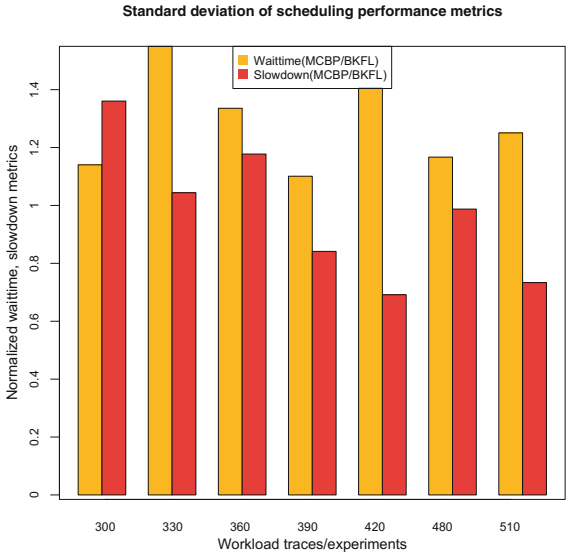


Fig. 2. Average of simulation results for experiments 780 to 960

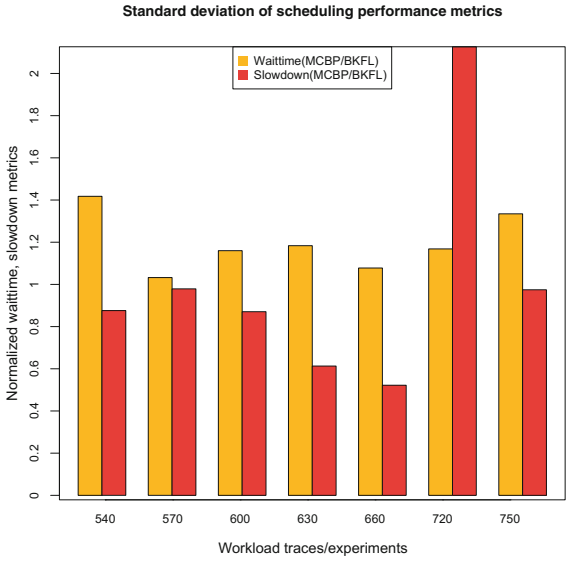
– *Waittime*: This is time $t_s - t_a$, the time a job request must wait before it starts running. The time units are in *minute*.
– *Slowdown*: If t_u is the time the job would take to run on a dedicated physical system, the job's slowdown is $(t_e - t_a)/t_u$. If t_u is less than 10 seconds, the slowdown is computed the same way, but assuming t_u to be 10 seconds [8].

The optimization of these two metrics is a *minimization* problem. We analyze simulation results for each experiment based on mean and standard deviation statistics measure. Mean statistics are illustrated in Figs. 1, 2, and standard deviation are illustrated in Figs. 3, 4. In order to compare two policies, we normalize **MCBP** results to **BKFL** results, i.e., $MCBP/BKFL$. All values presented in the graphs are based on this normalized value. This is to better present and compare two policies with a value.

In general, we have got better results for both metrics in terms of mean statistic measure. However, in terms of standard deviation waittime metric gets higher values for MCBP policy, while slowdown gets lower values. While on average we have got better results for waittime and slowdown metrics, we have more discrepancy of waittime values for MCBP policy respect to BKFL policy. Nonetheless, we have got more concentrated values for slowdown metric for MCBP policy. This observation implies that in total we have better scheduling with MCBP policy respect to BKFL policy. This means that with MCBP total



(a)Traces 300 to 510



(b)Traces 540 to 750

Fig. 3. Standard deviation of simulation results for experiments 300 to 750

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.

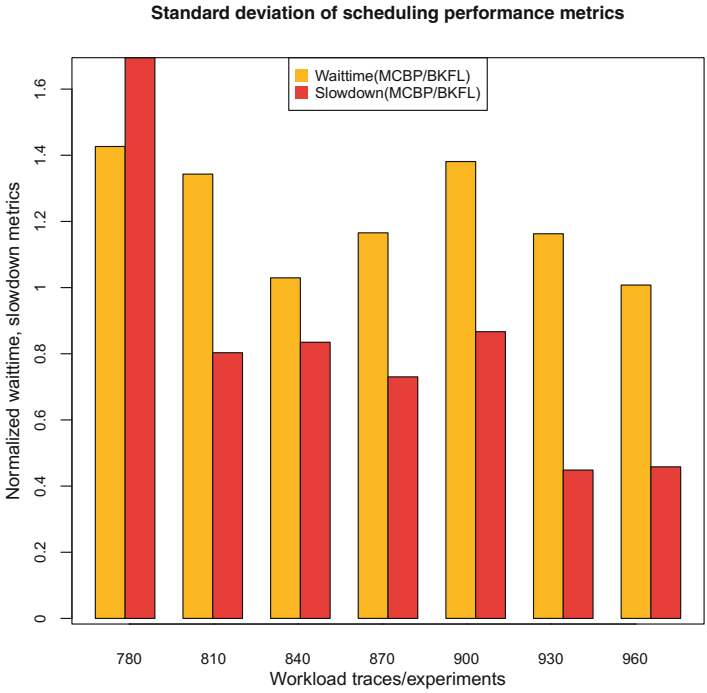


Fig. 4. Standard deviation of simulation results for experiments 780 to 960

Table 1. Statistics measures of simulation results over all experiments

(a) Mean

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

(b) Standard deviation

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

In addition, Table 1 presents the outcome of mean and standard deviation statistics measures over all experiments. These results demonstrate that the multi-capacity queuing 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.

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 account this shift from single-dimension resource

model. In this paper, we considered scheduling aspects of such a systems. Traditional scheduling algorithms 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 resource dimensions. The scheduling algorithms have to fully utilize all resources. To address this problem, we have proposed a multi-resource scheduling mechanism at the queuing mechanism. For that, we studied multi-capacity bin-packing queuing policy.

Through exhaustive simulation experimentation on 21 derived workload traces of SDSC Blue Horizon, we have demonstrated that the multi-capacity bin-packing queuing policy 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 bin-packing queuing policy can support more general instances. In addition, our solution can be integrated with real frameworks, like Nimbus Toolkit, and OpenNebula resource managers.

In this paper, we studied multi-capacity bin-packing queuing policy for each single job. We can apply this heuristic at 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.

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

References

1. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. **28**(5), 755–768 (2012)
2. Bobroff, N., Kochut, A., Beaty, K.: Dynamic placement of virtual machines for managing SLA violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, (IM 2007), pp. 119–128 (2007)
3. Cardosa, M., Korupolu, M.R., Singh, A.: 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, June 2009
4. Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., Gautam, N.: Managing server energy and operational costs in hosting centers. SIGMETRICS Perform. Eval. Rev. **33**(1), 303–314 (2005)
5. Coffman, E.G., Garey, M.R., Johnson, D.S.: An application of bin-packing to multiprocessor scheduling. SIAM J. Comput. **7**(1), 1–17 (1978)
6. Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM J. Comput. **12**(2), 227–258 (1983)

7. Dhyani, K., Gualandi, S., Cremonesi, P.: A constraint programming approach for the service consolidation problem. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 97–101. Springer, Heidelberg (2010)
8. Feitelson, D.G., Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1998, SPDP-WS 1998, and JSSPP 1998. LNCS, vol. 1459, p. 1. Springer, Heidelberg (1998)
9. Ferreto, T., Netto, M., Calheiros, R., De Rose, C.: Server consolidation with migration control for virtualized data centers. Future Gener. Comp. Syst. **27**(8), 1027–1034 (2011)
10. Garey, M.R., Graham, R.L.: Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. **4**(2), 187–200 (1975)
11. Garey, M.R., Graham, R.L., Johnson, D.S.: Resource constrained scheduling as generalized bin packing. J. Comb. Theory Ser. A **21**(3), 257–298 (1976)
12. Graubner, P., Schmidt, M., Freisleben, B.: Energy-efficient virtual machine consolidation. IT Prof. **15**(2), 28–34 (2013)
13. Gulati, A., Holler, A., Ji, M., Shanmuganathan, G., Waldspurger, C., Zhu, X.: VMware distributed resource management: design, implementation, and lessons learned. VMware Techn. J. (2012). https://labs.vmware.com/vmtj/vmware-distributed-resource-management-design-implementation-and-lessons-learned
14. Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009), pp. 41–50. ACM, New York (2009)
15. Leinberger, W., Karypis, G., Kumar, V.: Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In: Proceedings of the International Conference on Parallel Processing 1999, pp. 404–412 (1999)
16. Mastroianni, C., Meo, M., Papuzzo, G.: Probabilistic consolidation of virtual machines in self-organizing cloud data centers. IEEE Trans. Cloud Comput. **1**(2), 215–228 (2013)
17. Mazzucco, M., Dyachuk, D., Deters, R.: Maximizing cloud providers' revenues via energy aware allocation policies. In: 10th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2010), Melbourne, Australia, pp. 131–138, May 2010
18. Mehta, S., Neogi, A.: ReCon: a tool to recommend dynamic server consolidation in multi-cluster data centers. In: Proceedings of the Network Operations and Management Symposium, IEEE NOMS 2008, pp. 363–370 (2008)
19. Panigrahy, R., Talwar, K., Uyeda, L., Wieder, U.: Heuristics for vector bin packing (2011). http://research.microsoft.com
20. Quan, D.M., Basmadjian, R., de Meer, H., Lent, R., Mahmoodi, T., Sannelli, D., Mezza, F., Telesca, L., Dupont, C.: Energy efficient resource allocation strategy for cloud data centres. In: 26th International Symposium on Computer and Information Sciences (ISCIS 2011), London, UK, pp. 133–141, September 2011
21. Schröder, K., Nebel, W.: Behavioral model for cloud aware load and power management. In: Proceedings of HotTopiCS '13, 2013 International Workshop on Hot Topics in Cloud Services (HotTopiCS '13), pp. 19–26. ACM, New York, May 2013.
22. Stillwell, M., Schanzenbach, D., Vivien, F., Casanova, H.: Resource allocation algorithms for virtualized service hosting platforms. J. Parallel Distrib. Comput. **70**, 962–974 (2010)
23. Stillwell, M., Vivien, F., Casanova, H.: Dynamic fractional resource scheduling vs. batch scheduling. IEEE Trans. Parallel Distrib. Syst. **23**(3), 521–529 (2012). doi:10.1109/TPDS.2011.183

24. Verma, A., Ahuja, P., Neogi, A.: pMapper: power and migration cost aware application placement in virtualized systems. In: Issarny, V., Schantz, R. (eds.) Middleware 2008. LNCS, vol. 5346, pp. 243–264. Springer, Heidelberg (2008)
25. Wu, Y.-L., Wenqi, H., Lau, S.-C., Wong, C.K., Young, G.H.: An effective quasi-human based heuristic for solving the rectangle packing problem. Eur. J. Oper. Res. **141**(2), 341–358 (2002)

http://www.springer.com/978-3-319-13463-5

Adaptive Resource Management and Scheduling for Cloud Computing
First International Workshop, ARMS-CC 2014, held in Conjunction with ACM Symposium on Principles of Distributed Computing, PODC 2014, Paris, France, July 15, 2014, Revised Selected Papers
Pop, F.; Potop-Butucaru, M. (Eds.)
2014, XII, 217 p. 68 illus., Softcover
ISBN: 978-3-319-13463-5