Online Assignment and Placement of Cloud Task Requests with Heterogeneous Requirements

Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani† and Ammar Rayes‡
Oregon State University, Corvallis, OR 97331, dabbaghm,hamdaoub@onid.orst.edu
† Qatar University, mguizani@ieee.org | ‡ Cisco Systems, San Jose, CA 95134, rayes@cisco.com

***Abstract*—Managing cloud resources in a way that reduces the consumed energy while also meeting clients demands is a challenging task. In this paper, we propose an energy-aware resource allocation framework that: i) places the submitted tasks (elastic/inelastic) in an energy-efficient way, ii) decides initially how much resources should be assigned to the elastic tasks, and iii) tunes periodically the allocated resources for the currently hosted elastic tasks. This is all done with the aim of reducing the number of ON servers and the time for which servers need to be kept ON allowing them to be turned to sleep early to save energy while meeting all clients demands. Comparative studies conducted on Google traces show the effectiveness of our framework in terms of energy savings and utilization gains.**

***Index Terms*—Resource allocation, cloud computing, energy efficiency, convex optimization, VM placement.**

I. Introduction

Energy efficiency is a problem of a primal concern in cloud centers due to financial [1] and environmental [2] concerns. The main reason behind the large energy consumption of these centers is due to the fact that cloud servers operate, most of the time, at between 10% and 50% of their maximal utilizations, and that idle/under-utilized ON servers consume more than 50% of their peak power [3]. Therefore, to minimize cloud's consumed energy, one needs to reduce the number of ON servers, increase the servers utilization and reduce the duration for which servers are left ON while meeting clients' demands.

A cloud center is made up of a huge number of servers called physical machines (PMs). These PMs are grouped into multiple management units called clusters that are distributed in different geographical locations. Cloud clients may, at any time, submit a request to one of these clusters specifying the amount of computing resources they need in order to perform a certain task. For each received task request, the cluster's scheduler creates a virtual machine (VM), allocates certain amounts of resources to it, and assigns it to one of the PMs in the cluster. The VM is reserved for a period of time, called the execution time, after which the task is accomplished and the VM is released.

Prior resource allocation schemes focused on how to save energy in cloud centers by making efficient VM placements. The fact that the virtualization technology allows scheduling multiple VMs to the same PM was exploited in order to consolidate the submitted requests on fewer PMs, resulting in saving energy by turning to sleep as many redundant PMs as possible. The authors in [4] treated the VM placement problem as the classical online Bin Packing (BP) optimization problem, which views VMs as objects with certain sizes (CPU demands) and PMs as bins with certain capacities (CPU capacities) and where the objective is to pack these objects in as few bins as possible. Since the online BP problem is NP-hard [5], the Best Fit (BF) heuristic was used in [6] to make efficient placements. Dynamic consolidation techniques were also suggested in [7] where VM migration was adapted to perform periodic VM-PM remappings with the objective of producing more compact packing over time. Enhancements over these heuristics were proposed to reduce communication costs [8, 9]. We refer the reader to our prior work [10] for a complete overview on energy efficiency techniques in cloud centers.

Although prior work techniques lead to energy savings, they all handle task requests of the form (w_i^{req}, t_i^{req}) where w_i^{req} is task i's requested amount of CPU resources and t_i^{req} is the duration for which these resources are needed. Such requests are referred to by *inelastic* task requests as they require a fixed amount of CPU resource during their whole execution time and as increasing their allocated resources at any time would not decrease their execution times. An example of an inelastic task would be a task that is responsible for replying to queries received during a certain period. We define the requested computing volume v_i^{req} for an inelastic task i as: $v_i^{req} = w_i^{req} \times t_i^{req}$.

On the other hand, the submitted task requests to the cloud cluster could be elastic, i.e, the amount of allocated resources for these tasks can be increased or decreased during their execution time. Increasing the allocated resources for these tasks would reduce their execution time and vice versa. An example of such tasks would be any task with thread parallelism where the number of allocated threads (CPU resources) determines how fast the task terminates. An elastic task request i is defined by $(v_i^{req}, w_i^{max}, t_i^{max})$ where v_i^{req} is the requested computing volume, w_i^{max} is the maximum amount of CPU resources that can be allocated to the task and depends primarily on its maximum amount of parallelism, and t_i^{max} is a duration period by which the requested task should complete.

Unlike prior work that considered only inelastic tasks, we propose in this paper a resource-allocation framework that handles both elastic and inelastic task requests. Our main contributions are in proposing a framework that:

1) places the submitted tasks (elastic and inelastic) in a way that avoids turning new PMs ON while also reducing the PMs' *uptimes* (i.e. the duration for which PMs need to be kept ON to serve the hosted tasks).

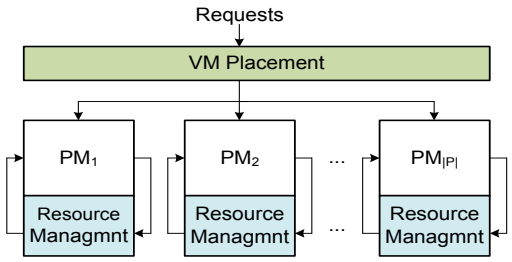


Fig. 1: Proposed Resource Allocation Framework.

2) decides what amount of resources should be assigned initially to the elastic tasks while guaranteeing that their deadlines will be met.
3) tunes the amount of allocated resources for the elastic tasks over time by solving a convex optimization problem whose objective is to reduce the PMs' uptimes while meeting tasks' requirements.

The remainder is organized as follows. Section II presents our proposed energy-aware allocation framework. Section III describes the optimization problem solved by our framework to tune the allocations over time. Section IV evaluates our framework on real traces from a Google cluster. Finally, section V provides conclusions and future work directions.

II. The Proposed Framework

As illustrated in Fig. 1, our proposed resource allocation framework has a two-level control structure as it is made up of a front end *VM Placement* module connected to all the PMs in the cloud cluster and an autonomous *Resource Management* module dedicated to each PM in the cluster. We explain next each one of these control modules:

A. VM Placement Module

Upon receiving a task request (elastic or inelastic), this module creates a VM for the submitted task and decides what PM in the cluster the created VM should be assigned to. These placement decisions are made depending on the current states of the PMs in the cluster and on two task-related quantities: the amount of CPU resources assigned initially to the created VM, and the time after which the VM is expected to be released. In the case of an inelastic task, these two quantities are directly specified by the client to be w_i^{req} and t_i^{req} respectively. On the other hand, there is flexibility in these quantities if the task is elastic as they are only bounded above by w_i^{max} and t_i^{max} respectively. Thus, the module needs to decide the quantity of resources that should be assigned initially to the elastic task in order to make efficient PM placement. Our module allocates $w_i^{min} = v_i^{req}/t_i^{max}$ amount of resources to the elastic task initially. This represents the least amount of resources needed so that the task is accomplished exactly in the client's maximum tolerable period t_i^{max}. The intuition behind this choice is the following. If we allocate less than w_i^{min} initially, then at some point in time we need to increase these resources so that the task terminates within t_i^{max}. However, the PM's capacity constraint and the constraints from the remaining hosted VMs may prevent us from doing that and thus we may risk to miss the task's deadline. Thus at least w_i^{min} amount of resources should be allocated initially to avoid that. On the other hand, if an amount of resources larger than w_i^{min} is assigned to the VM, then there may be no ON PM with enough slack to fit that VM and we will be forced to switch a new PM from sleep to fit this VM. This switch has a high energy overhead [11, 12] and also increases the number of ON PMs which leads to high energy consumption. This costly switch will be unnecessary if an ON PM with a slack of only w_i^{min} is already available in the cluster. Thus w_i^{min} is initially allocated to the elastic task in order to guarantee meeting its deadline while also saving energy. It is worth mentioning that these are only initial assignments for the elastic tasks and will be later tuned in order to maximize the energy savings as will be explained later. Having determined the amount of resources that should be allocated initially to the submitted task's VM and when the VM is expected to be released, we now explain the PM preference criteria that is adapted for efficient placement selection. Based on our preference criteria, the PMs in the cluster are divided into the following disjoint groups:

1) *Group 1:* contains all the PMs that are currently ON and that have an uptime larger than the VM's release time.
2) *Group 2:* contains the remaining ON PMs (the ones with an uptime smaller than the VM's release time).
3) *Group 3:* contains all the PMs in the sleep state.

The module tries first to place the new task's VM in one of the PMs of Group 1. These PMs are mostly favored as their uptime will not be increased after placing the new VM. If multiple PMs of Group 1 can fit the VM, then the one with the least CPU slack is chosen. The intuition behind this preference is to leave larger slacks on the remaining ON PMs so that VMs with larger CPU demands can be supported by these PMs in the future without needing to wake new PMs from sleep. If non of the PMs in Group 1 have enough space to fit the VM, then the PMs of Group 2 are considered. If multiple PMs from Group 2 can fit the VM, then the one whose uptime will be extended the least after placement will be chosen. This is to reduce the extra time for which the PM will be kept ON. Finally, if no fit is found in Group 2, then Group 3 is considered. Thus our preference criteria switches a new PM ON to accommodate the VM only if we have no other choice. The PM in Group 3 with the largest capacity is chosen in that case so that requests with larger demands can be supported by this PM in the future.

B. Resource Management Module

The main role of this module is to make resource and power management decisions for the controlled PM. A PM P_j's module performs the following procedures:

1) Resource Allocation: A flag is dedicated to indicate whether or not the allocated resources for the tasks hosted on the controlled PM need to be retuned. This flag is set true whenever the PM's uptime is increased due to assigning a new elastic task request, or whenever one

of the hosted VMs is released from P_j. The resource management module checks this flag periodically every T_p period. If the flag is true, then the amount of resources w_i to be allocated to each task i hosted on P_j is tuned. This is done by solving an optimization problem with the objective of minimizing the time after which the PM can be turned to sleep while guaranteeing that all tasks' requirements are satisfied. Details on the resource allocation optimization problem are provided in section III. The flag is reset to false after updating these allocations. Ideally, to maximize energy savings, new resource tuning needs to be calculated anytime the PM's uptime is increased due to assigning a new elastic task or anytime a VM is released from the PM. However, this has a high computation overhead. The flag is thus introduced for practical uses in order to limit the number of times the optimization problem is solved and the resources are retuned per PM to be once at most every T_p period. The choice of T_p is left to the cloud provider depending on how much computation overhead it can afford. The smaller the selected value for this parameter, the higher the energy savings but also the higher the computation overhead. In our implementation, T_p is set to 5 minutes as evaluations revealed that great energy savings can be achieved for such choice while keeping the calculation overhead low.

2) Remaining Time/Volume Tracking: for each elastic task i hosted on P_j, the module tracks t_i^{rem}, the amount of remaining time before which the task should be accomplished. The remaining time is initially set to t_i^{max} when the task is first scheduled on the PM and is decremented as time goes by. For each elastic or inelastic task i hosted on P_j, the module also tracks the amount of remaining computing volume still needed to accomplish each one of these tasks. The remaining computing volume for task i, referred to by v_i^{rem}, is initially set to be equal to the task's requested computing volume v_i^{req} when the task is first scheduled on P_j. Let w_i be the amount of resources allocated to task i for period T, then the remaining volume will be updated after the T period is over to be: $v_i^{rem} \leftarrow v_i^{rem} - (w_i \times T)$.
3) VM Termination: a task is accomplished once it is allocated all of its requested computing volume. Thus the module releases the allocated resources for VM i hosted on the PM once its remaining volume v_i^{rem} reaches zero. If no other VM is still hosted on the PM after this release, then the PM is switched to sleep to save energy.

III. PM Resource Allocation

Consider a PM P_j in the cloud cluster which has a certain CPU capacity C_j. Let $\mathbb{E}_j$ and $\mathbb{I}_j$ be respectively the sets of all elastic and inelastic tasks currently hosted on P_j. We explain in this section how to tune w_i, the amount of CPU resources to be allocated to each task i hosted on PM P_j, where $w_i \in \mathbf{R}^{++}$ which is the set of positive real numbers. The allocated resources w_i allows, in turn, to determine $t_i \in \mathbf{R}^{++}$, the time needed to accomplish the i^{th} task. Our framework allocates resources to the requests assigned to P_j by solving the following optimization problem.

Objective Function: The objective of our resource allocation strategy is to reduce the amount of time for which the PM will be left ON. This allows to switch the active PM into the power-efficient sleep state early so as to save energy. Since P_j needs to be kept ON until its last hosted task is accomplished, then our objective is to minimize the accomplishment time of the task with the maximal accomplishment time among those assigned to P_j, i.e., we seek to:

$$\text{Minimize} \quad \max\{t_i : i \in \mathbb{E}_j \cup \mathbb{I}_j\}$$

Constraints: The optimization problem is solved subject to the following constraints. One,

$$\sum_{i \in \mathbb{E}_j \cup \mathbb{I}_j} w_i \leq C_j \tag{1}$$

which states that the aggregate allocated resources for all the VMs hosted on P_j must not exceed the PM's capacity.

Two,

$$w_i = w_i^{req} \quad \forall i \in \mathbb{I}_j \tag{2}$$

which states that any inelastic VM request must be assigned the exact amount of requested CPU resources. Three,

$$w_i \leq w_i^{max} \quad \forall i \in \mathbb{E}_j \tag{3}$$

which states that the allocated resources for any elastic VM must not exceed the maximum amount of resources that the VM's task supports. Four,

$$t_i \leq t_i^{rem} \quad \forall i \in \mathbb{E}_j \tag{4}$$

which states that the accomplishment time of each elastic VM should be within t_i^{rem}, the remaining duration before which the task should be accomplished.

Five,

$$t_i \times w_i = v_i^{rem} \quad \forall i \in \mathbb{E}_j \cup \mathbb{I}_j \tag{5}$$

which states that the resulting allocations should provide the remaining computing volumes v_i^{req} required by the hosted tasks.

Equivalent Problem: We introduced so far the objective and the constraints of the formulated resource allocation optimization problem where the decisions variables are $w_i \in \mathbf{R}^{++}$ and $t_i \in \mathbf{R}^{++}$. We show now how to perform simple transformation on the above introduced original problem in order to transform it into an equivalent problem whose optimal solution can be obtained easily. The transformation consists basically of performing a variable renaming by letting $t_i = 1/f_i$. This variable renaming changes the optimization problem as follows. First, the objective of the original problem becomes:

$$\text{Minimize} \quad \max\{1/f_i : i \in \mathbb{E}_j \cup \mathbb{I}_j\}$$

where the decision variables are now $f_i \in \mathbf{R}^{++}$ and $w_i \in \mathbf{R}^{++}$. The objective function after the variable change becomes a convex function as for any given i, the function $1/f_i$

is convex on $\mathbf{R}^{++}$, and as the maximum of a set of convex functions is a convex function [13].

Constraints 1, 2, and 3 remain the same after the variable renaming as they are each a function of w_i only. Note that all of these three constraints are affine functions with respect to w_i.

Constraint (4) becomes the following affine constraint after renaming:

$$1 \leq f_i \times t_i^{rem} \quad \forall i \in \mathbb{E}_j$$

Observe that constraint (5) in the original problem is not affine as the decision variables w_i and t_i are multiplied by each other. However, after renaming the variables, the constraint is now transformed into the following equivalent linear (and thus affine) constraint:

$$w_i = f_i \times v_i^{rem} \quad \forall i \in \mathbb{E}_j \cup \mathbb{I}_j$$

As a result, by letting $t_i = 1/f_i$, the original problem transforms into a convex optimization problem as the new objective function is now convex (which we aim to minimize), and as all equality and inequality constraints of the new equivalent problem are now affine. The optimal solution for such problems can be found easily and quickly using convex optimization solvers such as CVX [14], which is the one used in our implementation.

An Illustrative Example: Having explained the formulated optimization problem, we now provide an example of how a feasible and an optimal resource allocation would look like for a set of tasks that are hosted on P_j that has a unit capacity (i.e. $C_j = 1$). The hosted requests are three elastic tasks (R_1, R_2, R_3) and one inelastic task (R_4). The specs of these tasks are shown in Table I. w^{max} and w^{req} are reported as a percent of the PM's capacity, the remaining time t^{rem} and the duration for which the inelastic task is needed t^{req} are reported in hours, and the remaining computing volume v^{rem} is reported in CPU percent times hours.

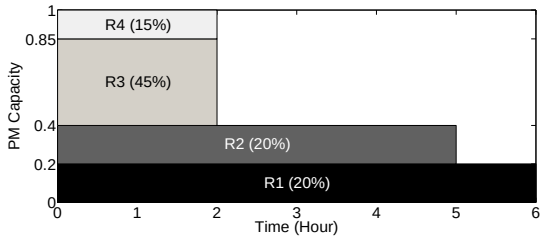
Fig. 2 shows two allocations for these requests: (a) a feasible allocation and (b) the optimal allocation obtained by solving the formulated problem. Both allocations in (a) and (b) satisfy all tasks constraints. However, the allocation in (a) provides more resources to R_3 whereas low resources are allocated to R_1 and R_2. This allocation is not energy efficient as the PM will be kept ON for six hours and can be turned to sleep only after that. Whereas the optimal allocation in (b) allocates more resource to R_1 and R_2 and lower resources to R_3 allowing the PM to be turned to sleep to save energy earlier (after four hours) while all requests still meet their deadlines.

IV. Performance Evaluation

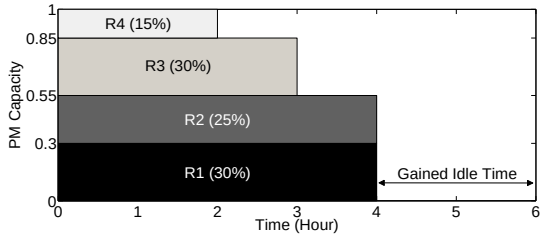
The experimental evaluations presented in this section are based on real cluster traces [15] that were released by Google in November 2011. These traces were collected during a 29-day period from a Google cluster that is made up of more than 12K PMs. The PMs within the cluster have three types in terms of the supported CPU capacity. Table II shows the number of PMs for each one of these types along with their

TABLE I: Specs of the requests in the illustrative example.

Elastic Request	v^{rem} (CPU percent×hour)	w^{max} (CPU percent)	t^{rem} (hour)
R_1	120	30	6
R_2	100	30	6
R_3	90	45	3
Inelastic Request	v^{rem} (CPU percent×hour)	w^{req} (CPU percent)	t^{req} (hour)
R_4	30	15	2



(a) Feasible Allocation



(b) Optimal Allocation

Fig. 2: Resource Allocations for the illustrative example.

CPU capacities normalized with respect to the PM with the largest capacity in the cluster. Since the size of the Google traces is huge (> 39 GB), we limit our evaluations to a chunk of the traces that spans 30 hours.

TABLE II: Configurations of Google cluster PMs.

Number of PMs	CPU Capacity
798	1.00
11658	0.50
126	0.25

For each task request i found in the traces, Google reports:

- a timestamp that indicates when the task was submitted.
- w_i^{trace} the task's reserved amount of CPU resources.
- t_i^{trace} the duration after which the task was accomplished based on the reserved resources.

Unfortunately the traces do not reveal the type or the nature of the submitted tasks and thus we could not infer the elastic tasks from the inelastic ones. In our evaluations, ρ percent of the tasks found in the traces are picked at random and are assumed to be elastic. We use the information revealed by the traces to determine the requested demands of these tasks. For each task i in the traces that is treated as inelastic, the requested amount of computing resources w_i^{req} and the duration for which these resources are needed t_i^{req} are taken

from the trace numbers and are set to be equal to w_i^{trace} and t_i^{trace} respectively. For each task i in the traces that is treated as elastic, the requested computing volume v_i^{req} is calculated from the traces numbers to be $v_i^{req} = w_i^{trace} \times t_i^{trace}$. The duration t_i^{max} within which the elastic task must be accomplished is set to t_i^{trace} (i.e. the elastic tasks have a worst case accomplishment time equal to the one reported in the traces). Finally, we had to make assumptions about w_i^{max}, the maximum amount of resources that can be allocated to task i, as no information is revealed about the nature of these tasks. In our experiments, w_i^{max} is set to: $w_i^{max} = w_i^{trace} + \lambda \times w_i^{trace}$ where λ is the elasticity factor. This makes w_i^{max} proportional to w_i^{trace} where the assumption here that the higher the reserved resources for the tasks in the traces, the higher the maximum range of resources that these tasks can be allocated. In our experiments, ρ is set to 50% meaning that half of the tasks found in the traces are assumed elastic. The elasticity factor for these tasks is set to $\lambda = 0.5$.

Our framework is compared against the following schemes:

- **BF with Min Allocation:** the BF heuristic [6] is used to make PM placement decisions where a submitted request is placed on a PM in sleep only if it can't be fitted in any ON PM and the PM with the largest capacity is chosen in that case for placement. If multiple ON PMs can fit the task, then the one with the least slack is selected for placement. The minimum amount of resources w^{min} is allocated for the elastic tasks throughout their lifetimes so that they finish exactly in t^{max} period.
- **BF with Max Allocation:** the BF heuristic is used to make PM placement decisions, and the maximum amount of the task's supported resources w^{max} are allocated to the elastic tasks throughout their lifetimes so that they are accomplished as fast as possible.
- **BF with Random Allocation:** the BF heuristic is used to make PM placement decisions, and for each requested elastic task i, a value between w_i^{min} and w_i^{max} is selected at random with an equal probability and gets assigned to the elastic task throughout its whole lifetime. It is worth mentioning that any selected value within that range guarantees that the task finishes within t_i^{max} period.

We next discuss our results in terms of the number of active PMs, utilization gains and energy savings.

A. Number of Active PMs

We analyze first the number of ON PMs that were needed to serve the submitted tasks as this number has a direct impact on the amount of consumed energy. Fig. 3 shows the number of ON PMs in the Google cluster over time when different schemes were used to manage the traces tasks. Observe that the BF with Max Allocation scheme used the largest number of ON PMs most of the time. This is due to the fact that by allocating the maximum amount of resources for the elastic tasks, it is true that these tasks were released as quickly as possible, however, they required a large amount of CPU resources all the time and thus the scheme faced many instances where it was forced to switch new PMs ON as there

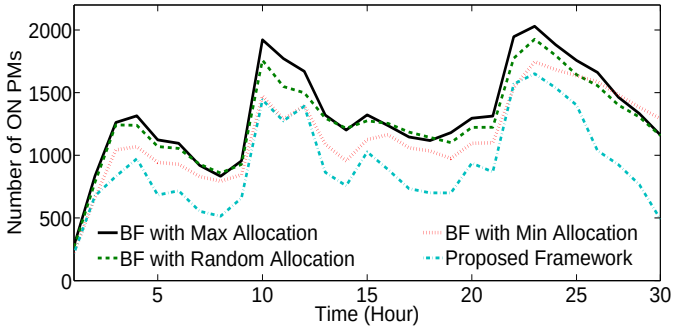


Fig. 3: Number of ON PMs over time for the different resource allocation schemes.

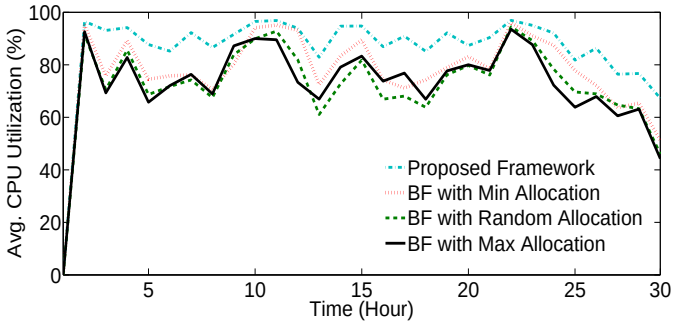


Fig. 4: Average Utilization over time for the different resource allocation schemes.

was no enough slack on any of the currently ON PMs to fit the coming requests. The BF with Random Allocation had a similar performance where many ON PMs were also needed to serve the submitted task requests. This clearly shows that allocating random amount of resources to the elastic tasks is not energy efficient. Observe that the BF with Min Allocation scheme used lesser number of PMs most of the time. Although elastic tasks in that case took longer time to be released, they were allocated smaller amounts of CPU resources leaving larger slacks to support the coming tasks which reduced the need for switching new PMs from sleep. Observe that our framework used the least number of ON PMs all the time compared to all the other schemes. Both the VM placement and the resource management modules were making efficient placement and tuning decisions in order to reduce both the number of ON PMs and the duration for which PMs need to be kept ON.

B. Utilization Gains

We compare next the utilization gains that are achieved by the different resource allocation schemes where the CPU utilization of a PM (referred to by η) is the aggregate amount of CPU resources reserved for all of its hosted VMs divided by the PM's capacity. Fig. 4 shows the average utilization for all of the ON PMs in the cluster over time under the different resource allocation schemes. Observe that the BF with Max Allocation and the BF with Random Allocation had the worst average utilization. This clearly shows that not only

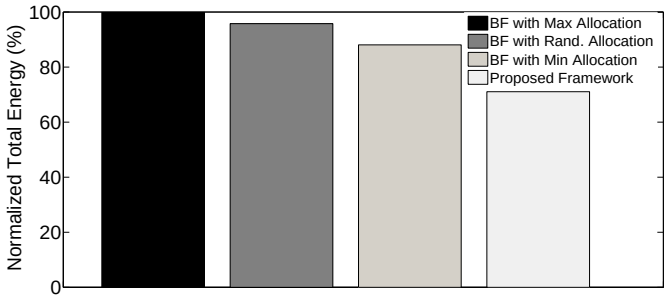


Fig. 5: Total Energy Costs for the different schemes (normalized w.r.t. BF with Max Allocation costs).

those two schemes used a large number of ON PMs, but also the resources of those ON PMs were not utilized efficiently. Although the BF with Min Allocation allocated the least amount of resources for the elastic tasks, it had an improved average utilization over time compared to those schemes as it used less number of ON PMs. Finally, our framework achieved the highest average utilization among all the other schemes. In fact, our framework reached in some cases an average utilization level that is very close to 100%. This is attributed to the VM placement module which packs the submitted tasks tightly and to the resource management module which reduced the wasted resource slacks by increasing the amount of allocated resources for the elastic tasks whenever possible.

C. Energy Savings

We assess next the energy savings that our framework achieves. Experiments in [3] show that the consumed power, P_{on}, of an active PM increases linearly from P_{idle} to P_{peak} as its CPU utilization, η, increases from 0 to 100%. More specifically, $P_{on}(\eta) = P_{idle} + \eta(P_{peak} - P_{idle})$, where $P_{peak} = 400$ and $P_{idle} = 200$ Watts. On the other hand, a PM in the sleep state consumes about $P_{sleep} = 100$ Watts. Switching a PM from sleep to ON consumes $E_{s\rightarrow o} = 4260$ Joules, whereas switching a PM from ON to sleep consumes $E_{o\rightarrow s} = 5510$ Joules. All of these numbers are based on real servers' specifications [16].

We calculate the total energy to run the cluster under the different resource allocation schemes where the total energy is the summation of the energy consumed by both ON and sleep PMs in addition to the switching energy (from sleep to ON and vice versa). Fig. 5 shows these total costs throughout the whole 30-hour period normalized with respect to the total energy cost of BF with Max Allocation scheme. Observe that our proposed framework met the demands of all of the requests within the traces while consuming the least amount of energy compared to all other resource allocation schemes. This proves the efficiency of our framework and highlights how important it is to make efficient placement and resource tuning decisions in cloud centers. It is worth mentioning that we also evaluated our framework and the remaining resource allocation schemes on the Google traces for different values of ρ and λ. In all of those cases, our framework consumed less energy compared to all the remaining schemes. Due to space limitation, we limited our analysis to the case where $\rho = 50\%$ and $\lambda = 0.5$.

V. Conclusions and Future Work

We propose in this paper an energy-aware allocation framework for elastic and inelastic task requests. Our framework makes allocation decisions in a way that reduces the number of ON PMs and the time for which PMs need to be kept ON while meeting all tasks' demands. Evaluations based on real traces from a Google cluster show the effectiveness of our framework when compared to other schemes in terms of energy savings and utilization gains. For future work, we plan to modify the framework in order to handle elastic tasks with different priorities where not only are tasks required to finish before a deadline, but also the faster the task's completion time, the higher the client's satisfaction level and charged cost.

VI. Acknowledgment

This work was made possible by NPRP grant # NPRP 5-319-2-121 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] D. Kliazovich, P. Bouvry, Y. Audzevich, and S.U. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *Proceedings of IEEE GLOBECOM*, 2010.

[2] C. Pettey, "Gartner estimates ICT industry accounts for 2 percent of global co2 emissions," 2007.

[3] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer Journal*, 2007.

[4] Y. Zhang and N. Ansari, "Heterogeneity aware dominant resource assistant heuristics for virtual machine consolidation," in *Proceedings of IEEE GLOBECOM*, 2013.

[5] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Release-time aware VM placement," in *Proc. of IEEE GLOBECOM Workshop on Cloud Computing Sys., Networks, and Applications (CCSNA)*, 2014.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, 2012.

[7] J. Xibo, Z. Fa, H. Songlin, and L. Zhiyong, "Risk management for virtual machines consolidation in data centers," in *Proceedings of IEEE GLOBECOM*, 2013.

[8] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proceedings of IEEE INFOCOM*, 2012.

[9] M. Shojafar, N. Cordeschi, D. Amendola, and E. Baccarelli, "Energy-saving adaptive computing and traffic engineering for real-time-service data centers," *IEEE ICC Workshop on Cloud Computing Systems, Networks and Applications*, 2015.

[10] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Towards energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Network Magazine*, 2015.

[11] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient resource allocation and provisioning framework for cloud data centers," *IEEE Transactions on Network and Service Management*, 2015.

[12] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient cloud resource management," in *Proceedings of IEEE INFOCOM Workshop on Mobile and Cloud Computing*, 2014.

[13] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge University Press*, 2004.

[14] Inc. CVX Research, "CVX: Matlab software for disciplined convex programming, version 2.0," 2012.

[15] *http://code.google.com/p/googleclusterdata/*.

[16] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "'efficient datacenter resource utilization through cloud resource overcommitment'," in *Proceedings of IEEE INFOCOM Workshop on Mobile Cloud and Virtualization*, 2015.