Advance Access publication on 30 November 2015
doi:10.1093/comjnl/bxv106

Virtual Machine Customization and Task Mapping Architecture for Efficient Allocation of Cloud Data Center Resources

Sareh Fotuhi Piraghaj[1]*, Rodrigo N. Calheiros[1], Jeffrey Chan[2], Amir Vahid Dastjerdi[1] and Rajkumar Buyya[1]

[1]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Melbourne, Australia*
[2]*School of Computer Science and Information Technology, RMIT University, Melbourne, Australia*
**Corresponding author: s.fotuhipiraghaj@student.unimelb.edu.au*

Energy usage of large-scale data centers has become a major concern for cloud providers. There has been an active effort in techniques for the minimization of the energy consumed in the data centers. However, most approaches lack the analysis and application of real cloud backend traces. In existing approaches, the variation of cloud workloads and its effect on the performance of the solutions are not investigated. Furthermore, the focus of existing approaches is on virtual machine migration and placement algorithms, with little regard to tailoring virtual machine configuration to workload characteristics, which can further reduce the energy consumption and resource wastage in a typical data center. To address these weaknesses and challenges, we propose a new architecture for cloud resource allocation that maps groups of tasks to customized virtual machine types. This mapping is based on the task usage patterns obtained from the analysis of the historical data extracted from utilization traces. In our work, the energy consumption is decreased via efficient resource allocation based on the actual resource usage of tasks. Experimental results show that, when resources are allocated based on the discovered usage patterns, significant energy saving can be achieved.

Keywords: cloud computing; energy efficiency; workload characterization; virtualization

Received 23 October 2014; revised 3 September 2015
Handling editor: Alan Marshall

1. INTRODUCTION

Cloud computing is a realization of utility-oriented delivery of computing services on a pay-as-you-go basis [1]. As stated by Armbrust *et al.* [2], cloud computing has the potential to transform a large part of the IT industry while making software even more attractive as a service. However, the major concern in cloud data centers is the drastic growth in energy consumption, which is a result of the rise in cloud services adoption and popularity. An average data center consumes as much energy as 25 000 households, as reported by Kaplan *et al.* [3]. This energy consumption results in increased Total Cost of Ownership and consequently decreases the Return of Investment (ROI) of the cloud infrastructure. Apart from low ROI, energy consumption has a great impact on carbon dioxide (CO_2) emissions, which are estimated to be 2% of global emissions [4].

There has been a growing effort in decreasing cloud data centers' energy consumption while meeting Service Level Agreements (SLA). The energy wastage in data centers is caused by various reasons such as inefficiency in data center cooling system [5], network equipments [6] and server utilization [7]. In this paper, we mainly focus on the efficient utilization of computing resources, since servers are still the main power consumers in a data center [8].

One of the key features introduced in data centers that can decrease their energy consumption is virtualization technology. This technology enables efficient utilization of resources and

load balancing via migration and consolidation of workloads. Therefore, a considerable amount of energy is saved with virtual machine migrations from underloaded servers by putting them in a lower power state. Many approaches utilize this technology, along with various heuristics, concentrating solely on virtual machine migrations and VM placement techniques with the objective of decreasing the data center power consumption. However, these approaches ignore tailoring virtual machine configurations to workload characteristics and the effect of such tailoring on the energy consumption and resource wastage in a typical data center.

User-defined virtual machine configuration is an available option for most cloud service models. Therefore, one of the challenges is to propose a method for defining the most efficient virtual machine configuration for a given application.

Apart from VM configuration, the other factor impacting the efficiency of resource utilization is the application of the knowledge obtained from the analysis of the real world clouds trace logs. This analysis enables an understanding of the variance of workloads that should be incorporated in solutions, as they affect the performance of proposed resource management approaches.

In this paper, we propose an end-to-end architecture for energy-efficient resource allocation and management in data centers. Because of the predefined software and applications that are executed in the data center, there exist similarities between the tasks' usage patterns and hence similar tasks can be grouped together using clustering algorithms. Our proposed solution decreases the resource wastage in data centers via virtualization and efficient resource allocation policies.

For defining virtual machine types and their capacity, we leverage similarities in the utilization patterns reported in the Google traces, which is confirmed by previous studies [9–12]. These similarities enable tasks to be grouped based on average resource usage via clustering techniques. Then the clustering output is used for the determination of customized virtual machine types. The actual resource utilization of tasks is considered during the grouping process, since there is a considerable gap between the actual reported resource usage and the requested amount of resources for task execution in the studied trace. In this respect, considering the actual resource utilization during the task execution will result in less resource wastage and consequently less energy consumption, which is one of the objectives of the proposed architecture.

In order to apply information of real cloud backend traces in our solution and on its evaluation, we utilized Google traces. The first Google log provides the normalized resource usage of a set of tasks over a 7-h period. The second version of the Google traces, which was released in 2012, contains more details in a longer time frame. Therefore, the data set used in this paper is derived from the second version of the Google cloud trace log [13] collected during a period of 29 days. The log consists of data tables describing the machines, jobs and tasks.

Recent work analyzing Google traces focused on various objectives such as characterization of task usage [14], task grouping for workload prediction and capacity planning [9], characterization of applications [12], modeling and synthesis of task placement constraints [15] and workload characterization for simulation parameter extraction and modeling [11,16,17]. Our work contributes to the current research area by introducing an architecture that utilizes the knowledge obtained from the workload characterization to determine efficient virtual machine configurations. The key *contributions* of our work are:

(i) We propose an end-to-end architecture for efficient allocation of requests on data centers that reduces the infrastructure's energy consumption.
(ii) We present an approach, applied to the proposed architecture, to identify virtual machine configurations (types) in terms of CPU, memory and disk capacity via clustering tasks, taking into consideration usage patterns of each cluster.
(iii) We propose an approach for the identification of VM task capacity, which is the maximum number of tasks that can be accommodated in a virtual machine, considering different estimates, including the average resource usage of tasks in each cluster.

An evaluation of the proposed architecture shows that the policy that considers the actual reported usage results in less energy consumption in the data center. This policy showed 73% improvement when comparing to a policy that allocates the virtual machine's resources based on the resource estimation provided by users.

The rest of the paper is organized as follows. Section 2 presents the related work in this area. Section 3 introduces the system model, the proposed architecture and its components. In Section 4, the implementation details of the task clustering is presented. In Section 5, we explain how the virtual machine configurations are defined, followed by a brief discussion on the resource allocation policies in Section 6. Section 7 describes the experiment set up including the data center's server configurations and the power consumption model of the servers. Then, the results of the algorithms' performance in terms of the energy and task execution efficiency are discussed in Section 8. Finally, Section 9 presents conclusions and discusses future research directions.

2. RELATED WORK

There is a considerable body of literature on power management in virtualized and non-virtualized data centers via hardware- and software-based solutions [18–20]. Most of the prior research in the area do not apply the knowledge obtained from the analysis of real cloud backend traces, nor the variance of the cloud workloads in their proposed solutions.

In 2009, Yahoo! released traces from a production MapReduce cluster to a selection of universities. In the same year,

Google made the first version of its traces publicly available. Google trace's release resulted in a variety of research investigating the problems of capacity planning and scheduling via workload characterization and statistical analysis of the planet's largest cloud backend traces [13].

2.1. Google trace research works

The research on Google cluster traces falls in three major categories, namely statistical analysis, workload modeling and characterization and simulation and modeling. They are further discussed in this section.

2.1.1. Statistical analysis

The first version of the Google traces contains the resource consumption of tasks, whereas the second version of Google traces covers more details including machine properties and task placement constraints. These constraints limit the machines onto which tasks can be scheduled [13]. In order to measure the performance impact of task placement constraints, Sharma *et al.* [15] synthesized these constraints and machine properties into performance benchmarks of Google clusters in their approaches.

Garraghan *et al.* [21] investigated server characteristics and resource utilization in the Google cluster data. They also explored the amount of resource wastage resulted from failed, killed and evicted tasks for each architecture type over different time periods. The average resource utilization per day lies between 40 and 60% as stated by Reiss *et al.* [22], and the CPU wastage on average server architecture type lies between 4.52 and 14.22%. These findings justify investigation of new approaches for improving resource utilization and reducing resource wastage.

Di *et al.* [23] investigated the differences between a cloud data center and other Grid/HPC systems considering both workload and host load in the Google data center. An analysis of the job length and jobs resource utilization in various system types, along with job submission frequency, shows that the host load in a cloud environment faces higher variance resulted from higher job submission rate and shorter job length. As a result, the authors identified three main differences between cloud and grid workloads: firstly, Grid tasks are more CPU intensive, whereas cloud tasks consume other resources, such as memory, more intensively. Secondly, CPU load is much noisier in clouds than in Grids. Thirdly, the host load stability differs between infrastructures, being less stable in clouds. These differences make the analysis of cloud traces crucial for researchers, enabling them to verify the applicability of heuristics in real cloud backend environments.

2.1.2. Workload modeling and characterization

Mishra *et al.* [9] and Chen *et al.* [10] explored the first version of the Google cluster traces and two approaches were introduced for workload modeling and characterization. Mishra *et al.* [9] used the clustering algorithm K-means for forming the groups of tasks with more similarities in resource consumptions and durations. Likewise, Chen *et al.* [10] used K-means as the clustering algorithm. In their experiments, the authors classified jobs[1] instead of tasks. Di *et al.* [12] characterized applications, rather than tasks, running in the Google cluster. Similarly to the two previous approaches, the authors chose K-means for clustering, although they optimized the K-means result using the Forgy method.

Moreno *et al.* [16] presented an approach for the characterization of the Google workload based on users and task usage patterns. They considered the second version of the Google traces and modeled the workload for two days of it. Later in 2014 [11], authors extended the work with an analysis of the entire tracelog. The main contribution of the work is considering information about users along with the task usage patterns. Moreno *et al.* [11, 16] also used K-means for grouping purpose. They estimated the optimal k with the quantitative approach proposed by Pham *et al.* [24].

The previous study demonstrated that there are similarities in task usage patterns of Google backend traces. Therefore, in our proposed architecture, likewise previous approaches [9,10], we group tasks with similarities in their usage patterns using clustering. In typical clustering, the number of clusters is a variable that is data-dependent and has to be set beforehand. Approaches noted in [11,16] use K-means and vary the number of clusters considering a finite range, for example 1–10. Then, the optimal value of k is derived considering the degree of variability in derived clusters [11,16] and Within cluster Sum of Squares [12]. Although these approaches could be applied here, we aimed to make the architecture as autonomous as possible and thus we avoided manual tuning of the number of clusters for each dataset like previous studies [11,12,16]. Pelleg and Moore [25] proposed X-means, a method that combines K-means with BIC. The latter is used as a criterion to automatic selection of the best number of clusters. Hence, we utilize X-means rather than existing approaches based solely on K-means [11,12,16]. It is worth mentioning that the workload modeling part of the architecture can be substituted, without changes in other components of the proposed architecture, by other approaches available in the literature [9–12,16].

The concept of task clustering has been previously investigated and proved to be effective outside of cloud computing area [26–28]. Our approach is different from them in terms of the objective and the target virtualized environment. For example, Singh *et al.* [26] and Muthuvelu *et al.* [27] utilized the technique for reducing communication overhead for submission of tasks in Grid systems, which are geographically distributed, in contrast with our application for energy minimization in a centralized cloud data center. Task clustering is also utilized by Wang *et al.* [28] to improve energy efficiency

[1] A job is compromised of one or more tasks [13].

in clusters via dynamic frequency and voltage (DVFS) techniques targeting parallel applications. Our approach, on the other hand, is agnostic to the application model and achieves energy-efficiency via consolidation and efficient utilization of data center resources. Furthermore, our work goes beyond these previous approaches on clusters and Grids by leveraging virtualization and mapping groups of tasks to VMs.

2.1.3. Simulation and modeling

Di *et al.* [17] proposed GloudSim as a distributed cloud simulator based on Google traces. This simulator leveraged virtualization technology and modeled jobs and their usage in terms of the CPU, memory and disk. It supports simulation of a cloud environment, that is, as similar as possible to Google cluster.

Moreno *et al.* [11,16] proposed a methodology to simulate the Google data center. Authors leveraged their modeling methodology to build a workload generator. This generator is implemented as an extension of the well-known cloud discrete simulator CloudSim [29] and is capable of emulating the user behavior along with the patterns of requested and utilized resources of submitted tasks in Google cloud data center.

In this paper, we present an end-to-end architecture aiming at efficient resource allocation and energy consumption in cloud data centers. In this architecture, the cloud provider utilizes the knowledge obtained from the analysis of the cloud backend workload to define customized virtual machine configuration along with maximum task capacity of virtual machines.

In the proposed architecture, likewise the discussed related work [11,16,17], we assume availability of virtualization technology and therefore tasks are executed on top of virtual machines instead of physical servers. This architecture can also be implemented utilizing the aforementioned simulation models [11,16,17]. Our work is different since we aim at decreasing energy by defining the virtual machines configurations along with their maximum task capacity.

3. SYSTEM MODEL AND ARCHITECTURE

Our proposed architecture targets Platform as a Service data centers operating as a private cloud for an organization. Such a cloud offers a platform where users can submit their applications in one or more programming models supported by the provider. The platform could support, for example, MapReduce or Bag of Tasks applications. Here, users interact with the system by submitting requests for execution of applications supported by the platform. Every application in turn translates to a set of jobs to be executed on the infrastructure. In our studied scenario, the job itself can be composed of one or more tasks.

3.1. User request model

In the proposed model, users of the service submit their application along with estimated resources required to execute it and receive back the results of the computation. The exact infrastructure where the application executes is abstracted away from users. Parameters of a task submitted by a user are:

(i) scheduling class;
(ii) task priority;
(iii) required number of cores per task;
(iv) required amount of RAM per task and
(v) required amount of storage per task.

All the aforementioned parameters are present in Google Cluster traces [13].

3.2. Cloud model

In the presented cloud model, system virtualization technology [30] is taken into consideration. This technology improves the utilization of resources of physical servers by sharing them among virtual machines [31]. Apart from this, live migration of VMs and overbooking of resources via consolidation of multiple virtual machines in a single host reduce energy consumption in the data center [32]. The other benefit of virtualization is the automation it provides for application development [33]. For example, once a virtual machine is customized for a specific development environment, the VM's image can be used on different infrastructures without any installation hassles. Therefore, as long as the virtual machine is able to be placed on the server, homogeneity of the environment offered by the VM image is independent of the physical server and its configuration. These characteristics and advantages of the virtualization technology persuade us in applying this in our proposed architecture.

Our focus is on data centers that receive task submissions and where tasks are executed in virtual machines instead of physical servers, a model that has been widely explored in the area of cloud computing [34,35]. Since these tasks might be different in terms of running environments, it is assumed that tasks run in containers [13] that provide these requirements for every one of them. However, in our model, these containers run inside the virtual machines instead of the physical machines. This can be achieved with the use of Linux containers or tools such as Docker [36], an open platform for application development and whose containers can run inside the virtual machine or on physical hosts.

3.3. System architecture

The objective of the proposed architecture (shown in Fig. 1) is to execute the workload with minimum wastage of energy. Therefore, one of the challenges is finding optimal VM configurations, in such a way that the accommodated tasks have enough resources to be executed and resources are not wasted during the operation. Since the proposed model has been designed to operate in a private cloud, the different number and types of

applications can be controlled, and there is enough information about submitted tasks so that cloud usage can be profiled.

3.4. System components

The proposed architecture is presented in Fig. 1, and their components are discussed in the rest of this section.

3.4.1. Pre-execution phase

We discuss the components of the proposed architecture that need to be tuned or defined before the system runtime:

(i) *Task classifier:* this component is the entry point of the streaming of tasks being processed by the architecture. It categorizes tasks arrived in a specified time frame into predefined classes. The classifier is trained with the clustering result of the historical data before system start up. The clustering is performed considering average CPU, memory and disk usage together with the priority, length and submission rate of tasks, obtained from the historical data. The time interval for the classification process is specified by the cloud provider according to the workload variance and task submission rate. Once the arriving task is classified in terms of the most suitable virtual machine type for processing it, it is forwarded to the Task Mapper to proceed with the scheduling process. The Task Mapper component is discussed in the execution phase.

(ii) *VM Type Definer:* this component is responsible for defining the virtual machines' configuration based on the provided historical data. Determining the optimal VM configuration requires analysis of task usage patterns. In this respect, the identification of groups of tasks with similar usage patterns reduces the complexity of estimating the average usage for new tasks. These patterns, which identify groups of tasks that have a mutual optimal VM configuration, are obtained with application of clustering algorithms.

(iii) *VM Types Repository:* in this repository, the available virtual machine types, including CPU, memory and disk characteristics, are saved. These types are specified by the *VM Type Definer* considering workload specifications and is derived from historical data used for training the task classifier component.

3.4.2. Execution phase

The components that operate during the execution phase of the system are discussed:

(i) *Task Mapper:* the clustering results from the *Task Classifier* are sent to the *Task Mapper*. The Task Mapper operation is presented in Algorithm 1. Based on available resources in the running virtual machines and the available VM types in the *VM Types Repository*, this component calculates the number and type of new virtual machines to be instantiated to support the newly

Algorithm 1: Overview of the Task Mapper operation process.

```
Input: KilledTasks, AvailablevmCapacity, NewTasks,
       VMTypeRepository
Output: NumberofvmsToInstatiate
1  foreach ProcessingWindow do
2      foreach Task in NewlyArrivedTasks do
3          if There is a vm in AvailablevmCapacity then
4              vm.Assign(Task)
5              vm.CheckStatus
6              Delete Task from NewlyArrivedTasks
7      foreach Task in KilledTasks do
8          if There is a vm in AvailablevmCapacity then
9              vm.Assign(Task)
10             vm.CheckStatus
11             Delete Task from KilledTasks
12     LeftTasks = Append KilledTasks to NewlyArrivedTasks
13     foreach Task in LeftTasks do
14         Calculate the NumberofvmsToInstantiate
```

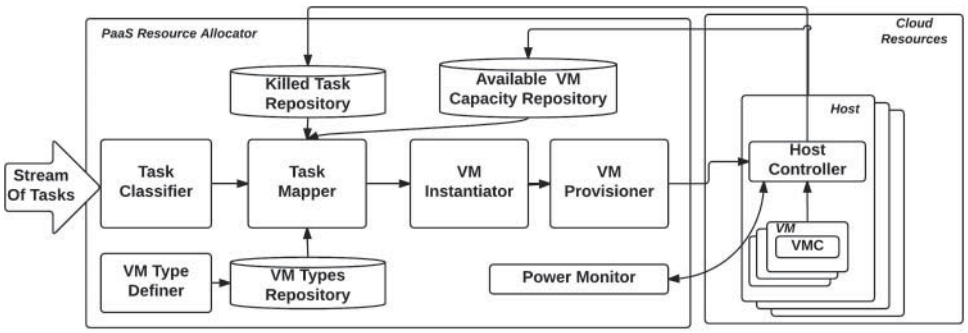

FIGURE 1. Proposed system architecture and its components.

arrived tasks. Apart from new VM instantiation when available VMs cannot support the arriving load, this component also reschedules rejected tasks that are stored in the killed task repository to the available virtual machines of the type required by the VM (if any). This component prioritizes the assignment of newly arrived tasks to available resources before instantiating a new virtual machine. However, in order to avoid starvation of the rejected tasks, the component assigns the newly arrived tasks to the available virtual machines and the killed tasks are assigned to newly instantiated VMs. The operation of this component on each processing window (Algorithm 1) has complexity $O(n \times m)$, where n is the total number of tasks to be mapped (i.e., tasks in the *KilledTaskDictionary* along with the tasks received in the processing window) and m is the number of VMs.

(ii) *Virtual Machine Instantiator:* This component is responsible for the instantiation of a group of VMs with the specifications received from the *Task Mapper*. This component decreases the start-up time of the virtual machines by instantiating a group of VMs at a time instead of one VM per time.

(iii) *Virtual Machine Provisioner:* This component is responsible for determining the placement of each virtual machine on available hosts and turning on new hosts if required to support new VMs.

(iv) *Killed Task Repository:* Tasks that are rejected by the Controller are submitted to this repository, where they stay until the next upcoming processing window to be rescheduled by the *Task Mapper*.

(v) *Available VM Capacity Repository:* IDs of virtual machines that have available resources are registered in this repository. It is used for assigning tasks killed by the *Virtual Machine Controller* along with newly arrived ones to available resource capacity.

(vi) *Power Monitor:* This component is responsible for estimating the power consumption of the cloud data center based on the resource utilization of the available hosts.

(vii) *Host Controller:* It runs on each host of the data center. It periodically checks virtual machine resource usage (which is received from the Virtual Machine Controllers) and identifies underutilized machines, which are registered in the available resource repository. This component also submits killed tasks from VMs running on its host to the *Killed Task Repository* so that these tasks can be rescheduled in the next processing window. Finally, this component also sends the host usage data to the *Power Monitor*.

(viii) *Virtual Machine Controller (VMC):* The VMC runs on each VM of the cloud data center. It monitors the usage of the VM and, if the resource usage exceeds the virtual machine capacity, it kills a number of tasks with low priorities so that high priority ones can obtain the resources they require in the virtual machine. In order to avoid task starvation, this component also considers the number of times a task has been killed. The Controller sends killed tasks to the *Host Controller* to be submitted to the global killed task repository. As mentioned before, killed tasks are then rescheduled on an available virtual machine in the next processing window. The operation of this component is shown in Algorithm 2 and it has complexity $O(n \times m)$, where n is the number of running tasks and m is the number of VMs.

Algorithm 2: Virtual Machine Controller Process.

```
Input: RunningTaskList, TaskUsage
Output: CPUUsage,MemoryUsage,DiskUsage,
        KilledTasksList
foreach Processingwindow do
    foreach Task in RunningTaskList do
        vm.updateUsage()
        vm.updateState()
    foreach vm whose state is OverLoaded do
        foreach Task in RunningTaskList do
            if TaskPriority equals to LowestPriority and
            has MinNumberofKills then
                vm.killTask()
                vm.updateState()
```

4. TASK CLUSTERING

In this section, we discuss the selected clustering feature set and the clustering algorithm utilized for clustering tasks with more details.

4.1. Clustering feature set

As our feature set, we used the following characteristics of each task:

(i) *Task Length*: The time during which the task was running on a machine.

(ii) *Submission Rate*: The number of times that a task is submitted to the data center.

(iii) *Scheduling Class*: This feature shows how latency sensitive the task/job is. In the studied traces, the scheduling class is presented by an integer number between 0 and 3. Tasks with a 0 scheduling class are non-production task. The higher the scheduling class is, the more latency sensitive is the task.

(iv) *Priority*: The priority of a task shows how important a task is. High priority tasks have preference for resources over low priority ones [13]. The priority is an integer number between 0 and 10.

(v) *Resource Usage*: The average resource utilization U_T of a task T in terms of CPU, memory and disk, which is obtained using Equation (1). In this equation, nr is the number of times that the task usage (u_T) is reported in the studied 24 h period and $u_{(T,m)}$ is the mth observation of the value of utilization u_T in the traces.

$$U_T = \frac{\sum_{m=1}^{nr} u_{(T,m)}}{nr}. \quad (1)$$

The selected features of the dataset were used for the estimation of the number of task clusters and determination of the suitable virtual machine configuration for each group. Application of data from other days of the trace, as well as utilization of other parameters from the trace for classification purposes, is the subject of our future work.

4.2. Clustering algorithm

Clustering is the process of grouping objects with the objective of finding the subsets with the most similarities in terms of the selected features. In this respect, both the objective of the grouping and the number of groups affect the results of clustering. In our specific approach, we focus on finding groups of tasks with similarities in their usage pattern so that available resources can be allocated efficiently. For discovering the other factor, namely definition of the most effective number of clusters, the X-means algorithm is utilized.

4.2.1. X-means clustering algorithm

Pelleg *et al.* [25] proposed the X-means clustering method as the extended version of K-means [37]. In addition to grouping, X-means also estimates the number of groups present in a typical dataset, which in the context of the architecture is the incoming tasks.

K-means is a computationally efficient partitioning algorithm for grouping N-dimensional dataset into k clusters via minimizing within-class variance. However, supplying the number of groups (k) as an input of the algorithm is challenging since the number of existing groups in the dataset is generally unknown. Furthermore, as our proposed architecture aims for automated decision making, it is important that the number of input parameters is reduced and that the value of k is automatically calculated by the platform. For this reason, we opted for X-means.

As stated by Pelleg *et al.* [25], X-means efficiently searches the space of cluster locations and number of clusters in order to optimize the Bayesian information Criterion (BIC). BIC is a criterion for selecting the best fitting model amongst a set of available models for the data [38]. Optimization of the BIC criterion results in a better fitting model.

X-means runs K-means for multiple rounds and then clustering validation is performed using BIC to determine the best value of k. It is worth mentioning that X-means has been successfully applied in different scenarios [39–42].

5. IDENTIFICATION OF VM TYPES FOR THE VM TYPE REPOSITORY

Once clusters that represent groups of tasks with similar characteristics in terms of the selected features are defined, the next step is to assign a VM type that can efficiently execute tasks that belong to the cluster. By efficiently, we mean successfully executing the tasks with minimum resource wastage. Parameters of interest of a VM are number of cores, amount of memory and amount of storage. Since tasks in the studied trace need small amount of storage, the allocated disk for virtual machines are assumed to be 10 GB, which is enough for the OS installed on the virtual machine and the tasks disk usage.

5.1. Determination of number of tasks for each VM type

Algorithm 3 details the steps taken for the estimation of the number of tasks for each virtual machine type. In order to avoid overloading the virtual machines, the maximum number of tasks in each VM is set to 150. This amount is allowed to increase if the resource demand is small compared with the VM capacity. Then, for each allowed number of tasks i (i between 1 and 150), i random tasks are selected from the cluster of task and the average CPU utilization is calculated for this selection. The CPU error is then reported and stored in $temp_{error}$.

Next, according to the $temp_{error}$, the algorithm finds the value of i that has the lowest CPU usage estimation error as the VM's number of tasks. This process is repeated for 500 iterations, which enables enough data to be collected for drawing conclusions. The VM's number of tasks in each iteration is then saved in Min_{error}. According to Min_{error}, the number of task for each VM type would be the number which shows the minimum estimation error in most of the iterations. In other words, the algorithm selects the number of tasks that is the most probable to result in less estimation error.

Algorithm 3: Estimation of the optimum number of tasks for each VM Type.

Input: *ClusterofTasks*
Output: *NumberofTasksPerCluster*
foreach *ClusterofTasks* **do**
  AvgCPU ←Average CPU Usage of the *ClusterofTasks*
  for *k from 1 to 500* **do**
    for *i from 1 to 150* **do**
      ClusterSample ← *i* random samples of *TaskCluster* without replacement
      $AvgCPU_s$ ← Average CPU usage for the *ClusterSample*
      $CPU_{Error} \leftarrow \frac{AvgCPU - AvgCPU_s}{AvgCPU}$
      $temp_{error}[i] \leftarrow CPU_{Error}$
    $Min_{Error}[k]$ ← Index of $min(temp_{Error})$
  $NumberofTasksPerCluster \leftarrow mode(Min_{Error})$

TABLE 1. Virtual machine configurations.

VM type	Number of tasks	vCPU	Memory (GB)	VM type	Number of tasks	vCPU	Memory (GB)
TYPE 1	136	3	4.5	TYPE 10	250	1	0.4
TYPE 2	125	1	0.5	TYPE 11	188	3	1.6
TYPE 3	500	1	1.8	TYPE 12	1250	1	1.1
TYPE 4	38	6	11	TYPE 13	118	4	10.3
TYPE 5	139	5	3.4	TYPE 14	126	25	14.2
TYPE 6	250	1	0.9	TYPE 15	100	2	1.9
TYPE 7	143	14	20.6	TYPE 16	136	3	6.8
TYPE 8	150	3	2.4	TYPE 17	143	2	1.1
TYPE 9	154	8	4.3	TYPE 18	500	1	3.8

5.2. Estimation of resource usage of tasks in each cluster

After estimating the maximum number of tasks in each virtual machine with the objective of decreasing the estimation error, the virtual machine types need to be defined. For this purpose, there is a need to estimate the resource usage of a typical task running in a virtual machine. For estimating the resource usage of each task in a cluster, the algorithm uses the average resource usage and variance of each cluster of tasks in our selected dataset. The first step for this is the computation of the average resource usage of each task during the second day of the trace and then for each cluster, the 98% confidence interval of the average utilization of resources of the tasks in the group is used. The upper-bound of the calculated confidence interval is then used as the estimate of the resource demands (*RD*s) for a typical task from a specific cluster.

5.3. Determination of virtual machines configuration

After obtaining the estimates for resource demands (RD) and the number of tasks in a virtual machine type (nT), the specifications of the virtual machine is derived using Equation (2).

$$Capacity = \lceil nT * RD \rceil. \quad (2)$$

Since tasks running in one virtual machine are already sharing the resources, at least one core of the CPU of the physical machine is assigned for each virtual machine. Because of the rounding process in Equation (2), the number of tasks in each virtual machine is estimated again applying the same equation.

The process above was applied to determine VM types for each cluster. VM types resulting from the above process are stored in the VM Types Repository to be used by the Task Mapper for assignment purposes. The application of this process resulted in the VM types described in Table 1. The number of tasks nT obtained via Equation (2) is used as the virtual machines' task capacity for the proposed Utilization-based Resource Allocation (URA) policy, which is briefly discussed in the next section along with the other proposed policies.

6. RESOURCE ALLOCATION POLICIES

The number of tasks residing in one VM varies from one cluster to another. As discussed in the previous section, virtual machine configurations are tailored to the usage pattern of the tasks residing in the VMs. The same virtual machine configurations are used for all the proposed policies. However, these algorithms are different in terms of the task capacity of the virtual machines for each cluster of tasks. These resource allocation policies are detailed:

(i) *URA*: in this policy, the number of tasks assigned to each VM is computed according to the 98% confidence interval of the observed average utilization of resources by the tasks being mapped to the VM. For example, if historical data shows that tasks of a cluster used on average 1 GB, and tasks of such cluster are going to be assigned to a VM with 4 GB of RAM, URA will assign 4 of such tasks, regardless the estimated amount of memory declared by the user when submitting the corresponding job (which is the value obtained from the traces). The task capacity of each virtual machine type is equal to the nT obtained from Equation (2), which is discussed in Section 5.

(ii) *Requested Resource Allocation* (RRA): in this policy, the same virtual machine types from URA are considered; however, the number of tasks assigned to a VM is based on the average requested amount by the submitted tasks. As mentioned before, the requested amount of resources is submitted along with the tasks. RRA is used as a baseline for our further comparisons in terms of data center power consumption and server utilization.

The other four policies are derived from the results of the evaluation of URA. In this respect, the usage of virtual machines is studied to get more insight about the cause of rejections (CPU, memory or disk) and the number of running tasks in each virtual machine when the rejections occurred.

For each virtual machine, the minimum number of running tasks that utilizes more than 90% of the VM's capacity in terms

Algorithm 4: Determination of the minimum number of running tasks for each virtual machine that causes VM resource utilization to be higher than 90% of its capacity without causing rejections.

Input: $vmListsofClusters = \{vmList_1, \ldots, vmList_{18}\}$
$vmList_{clusterIndex} = \{vmID_1, \ldots, vmID_{numberofVMs}\}_{clusterIndex}$
$resourceList = \{CPU, memory, disk\}$
Output: $nT_{(clustrIndex,Res)} = \{nt_{vmID_1}, \ldots, nt_{vmID_{numberofVMs}}\}$

```
for clusterIndex ← 1 to 18 do
    vmIDList ← vmListsofClusters.get(clusterIndex)
    for vmID in vmIDList do
        foreach Res in resourceList do
            Find minimum number of running tasks (nt) that caused the utilization of the resource (Res) to be between 90% to
            100% of its capacity. nT(clustrIndex,Res).add(nt_vmID)
```

of CPU, memory and disk without causing any rejections are extracted. This 90% limit avoids the occurrence of underutilized virtual machines. The extracted number is defined as $nt_{(vmID,resource)}$. The procedure is applied on each cluster and is explained with more details in Algorithm 4.

For each cluster determined by its *clusterIndex* in Algorithm 4, *nt* is obtained for each VM type. Then, *nt* of the VMs in each cluster are gathered in a set named $nT_{clusterIndex,Res}$ for each of the considered resources (*Res*) including CPU, memory and disk. We propose four policies to determine the number of tasks residing in each virtual machine. These policies as described below are based on the estimates (average, median, the first and the third quantile) derived from $nT_{clusterIndex,Res}$ for each cluster.

(i) *Average Resource Allocation policy (AvgRA):* For each cluster of tasks, considering *m* as the length of the set $nT_{clusterIndex,(CPU,memory,disk)}$, for the average number of tasks, we have

$$nT_{Avg,resource} = \left(\sum_{i=1}^{m} nt_{i,resource}/m\right). \quad (3)$$

The nT_{Avg} is estimated for each resource separately. In this policy, the number of tasks residing in each virtual machine type is equal to the minimum *nT* obtained for each resource (Equation (4)).

$$nT_{minimum} = \min(nT_{Avg,CPU}, nT_{Avg,memory}, nT_{Avg,Disk}). \quad (4)$$

(ii) *First Quantile Resource Allocation policy (FqRA):* For this policy, the first quantiles[2] of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. Like AvgRA, the minimum amount obtained for each of the resources is used. By resource, we mean the virtual machine's CPU, memory or disk capacity.

(iii) *Median Resource Allocation policy (MeRA):* for this policy, the second quantiles (median)[2] of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. Like the previous policy, the minimum amount of $nT_{Med,resource}$ obtained for each of the resources is used for determining the VM's task capacity.

(iv) *Third Quantile Resource Allocation policy (ThqRA):* In this policy, the third quantiles[2] of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. As in the previous cases, the minimum amount of $nT_{Med,resource}$ obtained for each of the resources is used for determining the virtual machines task capacity.

[2] The *k* quantile of a sorted set is the value that cuts off the first $(25 * k)\%$ of the data. For first, second and third quantile k is equal to 1, 2 and 3, respectively.

7. EXPERIMENT SET UP

We discuss the setup of the experiments that we conducted to evaluate our proposed approach in terms of its efficiency in task execution and power consumption.

The dataset used in this paper is derived from the second version of the Google cloud trace log [13] collected during a period of 29 days. The log consists of data tables describing the machines, jobs and tasks. In the trace log, each job consists of a number of tasks with specific constraints. Considering these constraints, the scheduler determines the placement of the tasks on the appropriate machines. The event type value in the job and tasks are reported in the event table. The job/task event has two types: events that change the scheduling state such as submitted, scheduled or running and events that indicate the state of a job such as dead [13]. For the purpose of this evaluation, we utilize all the events from the trace log and we assume that all the events are occurring as reported in the trace log. The second day of the traces is selected for evaluation purpose, as it had the highest number of task submissions.

In the available traces, resource utilization measurements and requests are normalized, and the normalization is performed separately for each column. As stated by Reiss *et al.* [13],

TABLE 2. Largest amount of each resource applied for de-normalization.

CPU	Memory (GB)	Disk (GB)
100% of a core of the largest machine CPU (3.2 GHz)	4	1

normalization is carried out in relation to the highest amount of the particular resource found on any of the machines. In this context, to get a real sense of the data, we assume the largest amount of resources for each column as described in Table 2 and multiply each recorded data by the related amount (e.g for recorded memory utilization we have $Real_{util} = Recorded_{Util} * 4$). Then, the total resource utilization and requested amount are calculated for each cluster as discussed in the last section. In order to eliminate placement constraints, only the tasks scheduled on one of the three available platforms are considered and the configurations of the simulated data center servers are further discussed in Section 7.1.

The proposed system is simulated for each cluster and the tasks are assigned to the corresponding virtual machine types during each processing window (1 min for the purposes of these experiments). The simulation runtime is set to 24 h. Cluster resource usage and number of rejected tasks are reported for each cluster of tasks separately. Since the virtual machine placement also affects the simulation result, the same policy introduced in Section 7.2 is used in the Virtual Machine Provisioner component for all the proposed algorithms. In order to show the efficiency of our proposed architecture in terms of power consumption, linear power consumption model is considered for each of the running machines. The power consumption model is further discussed in Section 7.3.

7.1. Data center servers' configuration

We define a data center with three server configurations listed in Table 3. These types are inspired from Google data center and its host configurations during the studied trace period. Hosts in the Google cluster are heterogeneous in terms of the CPU, memory and disk capacity. However, hosts with the same platform ID have the same architecture.

As mentioned in Section 2, there are three types of platforms in Google data center. In order to eliminate placement constraint for tasks, we have chosen the platform with the largest number of task submissions. The server architecture for our implementation is the same for all three types. As suggested by Garraghan *et al.* [43], servers in this platform are assumed to be 1022G-NTF (Supermicro Computer Inc.) inspired from the SPECpower_ssj2008 results [44].

7.2. Virtual machine placement policy

The First Fit algorithm is applied as the placement policy for finding the first available machines for hosting newly instantiated VMs. The algorithm first searches through the running machines to find if there are enough resources available for the virtual machine. It reports the first running host that can provide the resources for the VM. If there is no running host found for placing the virtual machine, a new host is activated. The new host is selected from the available host list, which is obtained from the trace log and contains the hosts IDs along with their configurations. All the proposed algorithms have access to the same host list to make sure that the placement decision does not affect the simulation results.

7.3. Server's power consumption model

The power profile of the selected server[3] from SPECpower is used for determining the linear power model constants in Equation (5) [45]. The power consumption for processing tasks at time t is defined as the accumulative power consumed in all the active servers at that specific time. For each server, the power consumption at time t is calculated based on the CPU utilization and server's idle and maximum power consumption (Eq. (5)). We focus on energy consumption of CPU because this is the component that presents the largest variance in energy consumption regarding its utilization rate [45].

$$P_n(t_i) = (P_{max} - P_{idle}) * n/100 + P_{idle} \quad (5)$$

8. EXPERIMENT RESULTS

X-means algorithm reports the existence of 18 clusters in the tasks. In this section, we go through the specifications of the task clusters and then we compare how efficiently the six algorithms can successfully execute the tasks. Later, in Section 8.3, we discuss the comparison of the proposed algorithms in terms of their energy consumption.

8.1. Characteristics of task clusters

We briefly discuss the characteristics of task clusters in terms of the scheduling class, priority, and the average length of the tasks in each group (Table 4). The population comparison of the clusters is presented in Fig. 2. To enable a better understanding of the characteristics of task clusters, Fig. 3 summarizes Table 4 considering the similarities between task groups.

In Fig. 3, task priority higher than 4 is considered 'high'. In addition, the average task length <1 and <5 h are noted 'short' and 'medium' length, respectively. The average task length higher than 5 h is considered 'long'. Figure 3 shows that almost 78% of the tasks fall in to the short length category. In addition, all long and medium length tasks have higher priorities and are less likely to be preempted. This logic is implemented in the Google cluster scheduler to avoid long tasks getting restarted in

[3] 1022G-NTF (Supermicro Computer, Inc.)

TABLE 3. Available server configurations present in one of the platforms of the Google cluster [43].

Server type	Number of cores	Core speed (GHz)	Memory (GB)	Disk (GB)	P_{idel} (W)	P_{max} (W)
Type1	32	1.6	8	1000	70.3	213
Type2	32	1.6	16	1000		
Type3	32	1.6	24	1000		

TABLE 4. Statistics of the clusters in terms of the scheduling class, priority and the average task length. The star sign (*) shows the dominant priority and scheduling class of the tasks in each group.

	Scheduling class			Priority						Task length (average)
Cluster-1	0	1*	2	8		9*	10	11		
	0.09%	99.76%	0.15%	0.88%		99.10%	0.01%	0.01%		18.19 (h)
Cluster-2	0*			6*						
	100%			100%						6.38 (min)
Cluster-3	0*			8*		9		10		
	100%			63.20%		36.76%		0.04%		5.7 (min)
Cluster-4	0*			4*						
	100%			100%						1.04 (h)
Cluster-5	0*			4*						
	100%			100%						20.32 (min)
Cluster-6	0*			4*						
	100%			100%						5.32 (min)
Cluster-7	0*			0*		1		2		
	100%			97.02%		2.98%		0.01%		56.82 (min)
Cluster-8	0*		1	0*	1	2	4	9		
	94.32%		5.7%	83.44%	8.42%	0.33%	7.80%	0.01%		4.72 (h)
Cluster-9	0*			0		1*		2		
	100%			36.55%		63.23%		0.23%		1.47 (h)
Cluster-10	0*			0*						
	100%			100%						28.04 (min)
Cluster-11	0*			1*			2			
	100%			99.2%			0.8%			19.19 (min)
Cluster-12	0*			0*						
	100%			100%						21.17 (min)
Cluster-13	2*		3	2	4*	6	8	9	10	
	74.9%		25.1%	0.03%	30.97%	16.28%	21.75%	28.17%	2.80%	38.66 (min)
Cluster-14	0	1*	2	1		2	4*	9		
	2.28%	97.62%	0.09%	0.04%		0.02%	99.49%	0.45%		42.9 (min)
Cluster-15	1*			4*		5		6		
	100%			97.67%		0.04%		2.29%		39.83 (min)
Cluster-16	2*		3	0*	1	2	4	9		
	93.67%		6.33%	67.61%	28.14%	4.23%	0.015%	0.001%		1.45 (h)
Cluster-17	1*			0*			1			
	100%			99.2%			0.8%			27.59 (min)
Cluster-18	1*			1			2*			
	100%			2.7%			97.3%			20.49 (min)

the middle of execution, which leads to more resource wastage. Next, we describe the four meta-cluster task groups.

(i) *Short and high priority tasks (Cluster 2, 3, 13):* Tasks in clusters 2 and 3 are all from scheduling class 0. However, tasks in cluster 13 are from higher scheduling classes, which indicates that they are more latency sensitive than the tasks in clusters 2 and 3. Amongst these three clusters, cluster 13, with the average length of 38.66 min, has the longest average length.

(ii) *Short and low priority tasks (Clusters 5, 6, 7, 10, 11, 12, 14, 15, 17, 18):* Comparing to others, this category includes the largest number of clusters. Cluster 7, with the average length of 56.82 min, has the longest tasks in this group. Considering the scheduling class, tasks in clusters 5, 6, 7, 10, 11 and 12 are all from scheduling class 0 while most of the tasks in clusters 14, 15, 17 and 18 are from scheduling Class 1.

(iii) *Medium and low priority tasks (Clusters 4, 8, 9, 16):* In terms of the average task length, Cluster 8, with 4.72 h, has the longest length. Considering the scheduling class, the tasks in Cluster 16 are more latency sensitive and probably belong to the production line, while the tasks from the other three clusters are less latency sensitive.

(iv) *Long and high priority tasks (Cluster 1):* Although Cluster 1 contains <1% of the tasks (Fig. 2), this group has the highest priority tasks with the longest durations as shown in Table 4. Most of the tasks of the group have scheduling Class 1, which shows they are less latency sensitive in comparison with the tasks from higher scheduling classes.

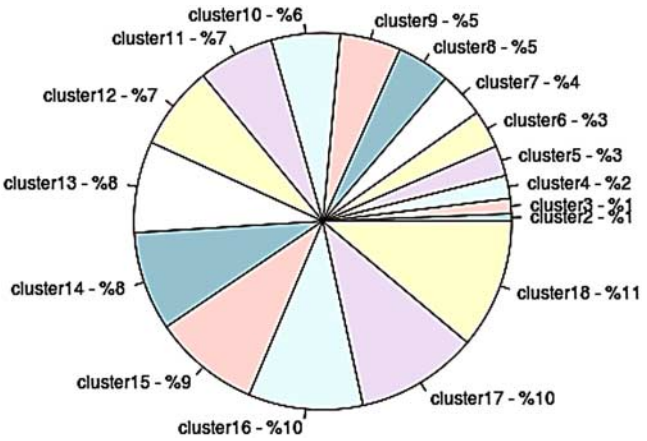

FIGURE 2. Population of tasks in each cluster. Clusters 15–18 are the most populated clusters. Since Cluster 1 population is <1%, it is not shown in the chart.

The results of clustering allowed us to draw conclusions per cluster that help in the design of specific resource allocation policies for each cluster. For example, as depicted in Fig. 3, tasks in Cluster 1 are the longest and have the highest priority. Therefore, one can conclude that the system assigns the higher priority to these long tasks so that if they failed, the system still has time to reschedule them. In contrast, as illustrated by Fig. 3, the majority of tasks with short length have been given low priorities. This is because, in case of both failure or resource contention, the system can delay their execution and still guarantee that they are executed in time.

In addition, as shown in Table 5, for task clusters with larger length, less usage variation is observed. For resource allocation policies, this makes the usage estimation of resources and predictions more accurate and more efficient, as less sampling data are required, while the prediction window can be widened. The opposite holds for clusters with smaller length: in these

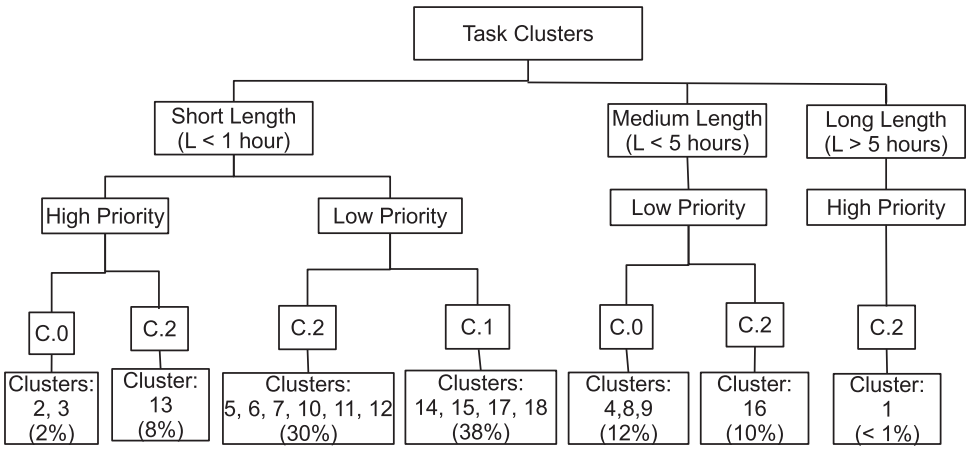

FIGURE 3. Clusters of tasks are categorized on three levels according to the average length, the priority and the scheduling class (*C*) considering the statistics in Table 4.

TABLE 5. Virtual machine task capacity of each cluster for RRA, FqRA, AvgRA, MeRA, ThqRA and URA resource allocation policies.

ClusterIndex	RRA	FqRA	AvgRA	MeRA	ThqRA	URA
Cluster-1	15	29	34	32	38	136
Cluster-2	20	33	39	40	43	125
Cluster-3	20	16 500	16 500	16 500	16 500	500
Cluster-4	41	33	43	38	56	38
Cluster-5	28	32	48	46	56	139
Cluster-6	8	51	88	77	117	250
Cluster-7	80	46	64	53	62	143
Cluster-8	73	45	48	47	48	150
Cluster-9	41	95	104	104	107	154
Cluster-10	6	7	9	7	8	250
Cluster-11	12	91	92	99	105	188
Cluster-12	11	28	53	46	71	1250
Cluster-13	28	13	19	13	13	118
Cluster-14	83	67	69	67	67	126
Cluster-15	18	17	45	33	52	100
Cluster-16	48	74	79	94	101	136
Cluster-17	7	21	28	22	23	143
Cluster-18	73	409	439	478	478	500

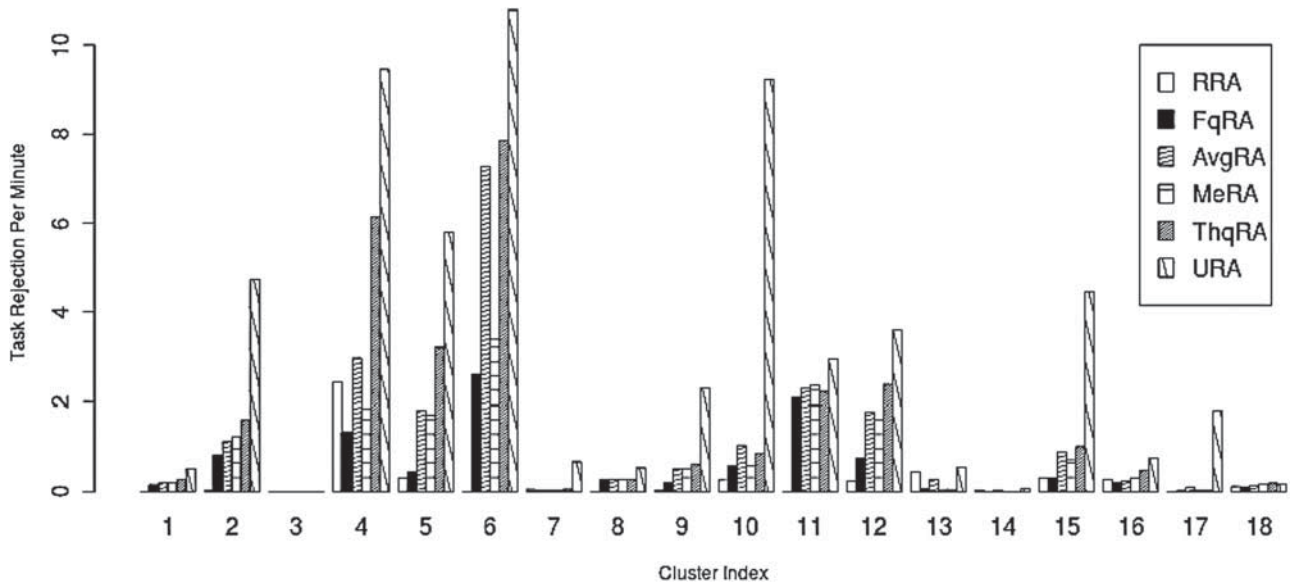

FIGURE 4. Task execution efficiency in the RRA, FqRA, AvgRA, MeRA, ThqRA and URA policies. Efficiency is measured as the task rejection rate per minute.

clusters, more variation is observed, and as a result prediction require more frequent sampling and narrower time window.

8.2. Task execution efficiency of the proposed algorithms

We compare task execution efficiency of our proposed algorithms in terms of task rejection rate. Ideally, the percentage of tasks that need to be rescheduled should be as low as possible, since it results in delays in the completion of jobs. In addition to delays, the increase in task rejection rate increases resource wastage since computing resources (and energy) are spent with tasks that do not complete successfully and thus need to be later executed again. The rejection rate for each policy is presented in Fig. 4.

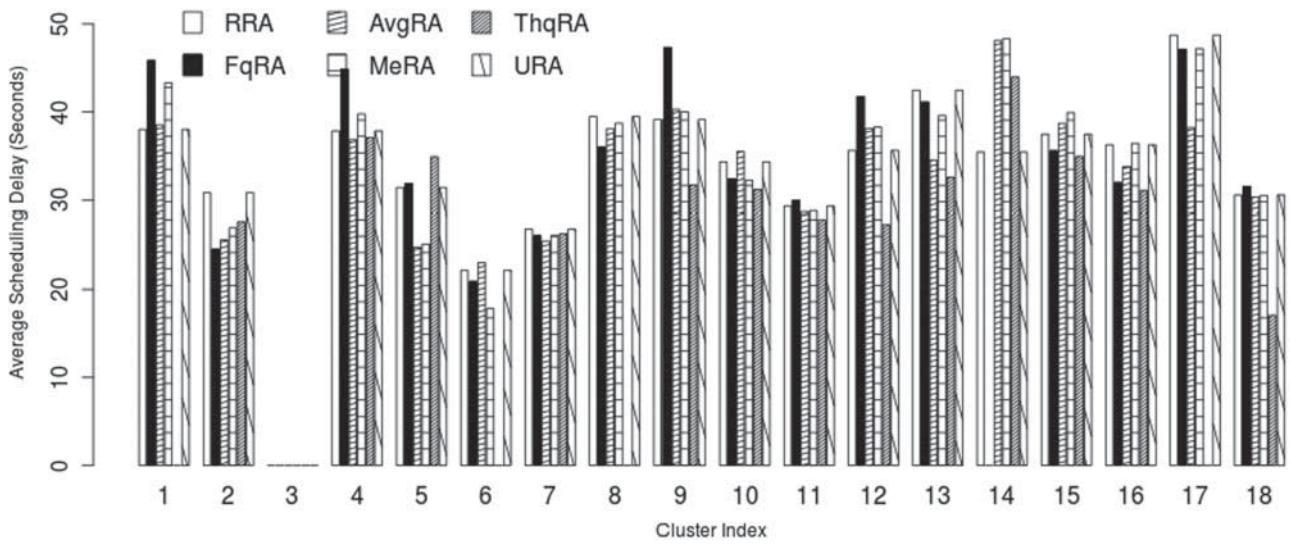

FIGURE 5. Average delay caused by applying the RRA, FqRA, AvgRA, MeRA, ThqRA and URA policies. The delay is estimated by the time it takes for a specific task to be rescheduled on another virtual machine after being rejected.

The virtual machine capacity for each of the algorithms is shown in Table 5. In the URA policy, tasks are allocated based on the actual usage. Because of the gap between requested resources and the actual usage of tasks, in URA the VM task capacity is higher than the other five algorithms. Therefore, in most of the clusters, RRA accommodates the least number of tasks in one virtual machine. Excluding RRA, FqRA has the smallest VM task capacity in comparison to the other four algorithms and excluding the URA policy, ThqRA has the largest amounts in terms of the task capacity.

Considering task rejection rate, the algorithms with larger amounts in terms of VM task capacity have higher rejection rates. Therefore, in most clusters, URA has the highest rejection rate. However, the gap between rejection rates for FqRA, AvgRA, MeRA and ThqRA are almost negligible. As expected, RRA, with the lowest number of tasks in each virtual machine, incurs the least rejections during the simulation.

In addition to rejection rates, the delay caused in the execution of the tasks are reported for the proposed policies. This delay is extracted for rejected tasks that finish during the simulation time (24 h). The delay t_d is equal to the $t_f - t_g$ in which t_f is the time that the execution of the task is finished in our simulation and t_g is the desired finished time reported in the Google traces. In other words, t_d of a typical task is the time it takes for the task to start running after it is rejected. Figure 5 shows that the average delay for all the proposed algorithms is less than 50 s. This delay can be reduced via smaller processing window sizes. The processing window size in our case is assigned to 1 min, therefore tasks should wait in the killed task repository until the next processing window, so that they can get the chance to be rescheduled in another virtual machine.

8.3. Energy efficiency of the proposed algorithms

The experiments presented in the previous section focused on the analysis of the performance of the assignment policies in terms of rejection rate and average delay. Since one of the goals of the proposed architecture is efficient resource allocation, which results in less energy consumption, in this section we analyze the policies in terms of their energy efficiency.

The power consumption incurred by servers are estimated using the power model presented in Equation (5). Figure 6 shows the amount of energy consumption (kWh) for the six applied resource allocation policies. In terms of energy consumption, URA on average outperform RRA, FqRA, AvgRA, MeRA and ThqRA by 73.02, 59.24, 51.56, 53.22 and 45.36%, respectively, considering all the clusters. However, URA in most of the clusters increases the average task rejection rate and results in delays in task execution. Considering this, URA is the selected policy when tasks have low priorities and the delay in the execution is not a concern.

ThrdRA policy is the second most energy efficient algorithm, outperforming RRA, FqRA, AvgRA and MeRA in average 34.41, 25.11, 7.42 and 15.01%, respectively. Apart from energy efficiency, this policy caused less task rejections in comparison with URA. Therefore, when task execution efficiency and energy are both important, this policy is the best choice. RRA in most clusters is the least energy efficient algorithm, although it caused less task rejections. Therefore, RRA can be applied for tasks with higher priorities.

AVgRA and MeRA have almost the same energy consumption for all the clusters. The task capacity of the VM in AvgRA and MeRA is based on the average and the median number of tasks that can run without causing any rejections. In most cases,

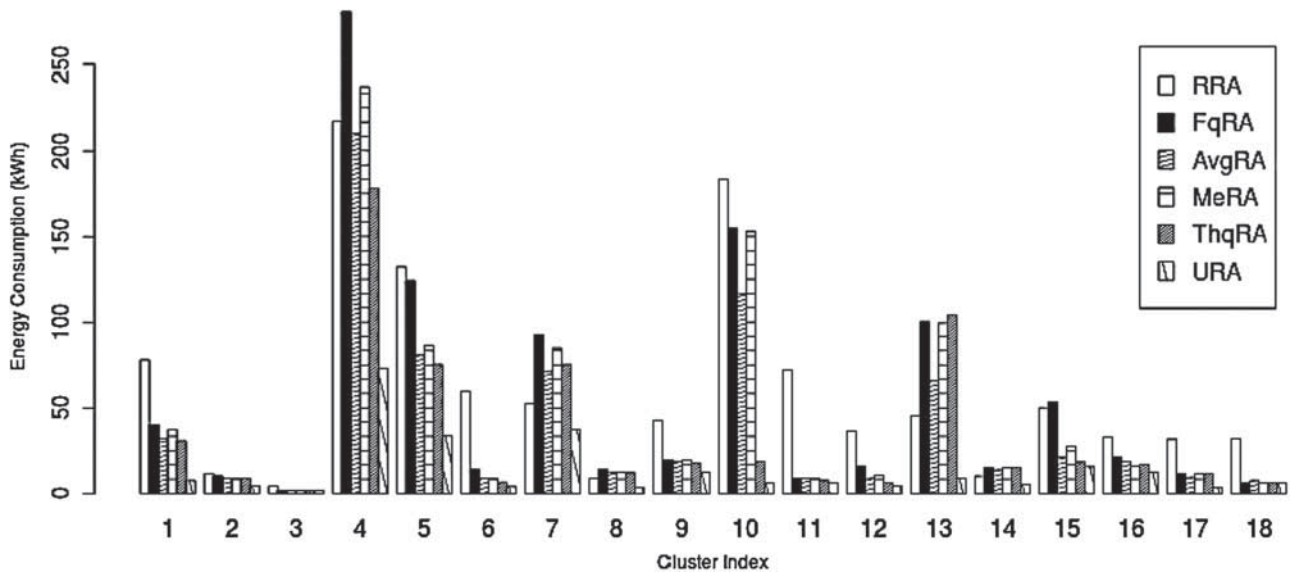

FIGURE 6. Energy consumption comparison of the RRA, FqRA, AvgRA, MeRA, ThqRA and URA policies. URA outperforms the other five algorithms in terms of the energy consumption and the average saving considering all the clusters.

the median and the average of our considered estimate (number of running tasks) are close to each other, therefore the difference in the energy consumption of AvgRA and MeRA is negligible.

To get more insights about the effect of each algorithm, we applied the six policies for each group of tasks separately. However, a combination of these policies could be applied on each cluster when tasks from different clusters run simultaneously. This will be subject of future work.

9. CONCLUSIONS AND FUTURE WORK

We investigated the problem of energy consumption resulted from inefficient resource allocation in cloud computing environments using Google cluster traces. We proposed an end-to-end architecture and presented a methodology to tailor virtual machine configuration to the workload. Tasks are clustered and mapped to virtual machines considering the actual resource usage of each cluster instead of the amount of resources requested by users.

Six policies are proposed for estimating task populations residing in each VM type. In the RRA policy, tasks are assigned to VMs based on their average requested resource. This policy is the baseline for our future comparison since it is solely based on the requested resources submitted to the data center. Resource allocation in the URA policy is based on the average resource utilization of task clusters obtained from historical data. In the other four policies, the assignment is based on the four estimates extracted from the virtual machines' usage logs from the URA policy. The extracted estimates are average, median, first and third quantile of the number of tasks that can be accommodated in a virtual machine without causing any rejections. Compared with RRA, URA, FqRA, AvgRA, MeRA and ThqRA policies show 73.01, 14.68, 34.72, 25.20 and 34.41% improvement in the total energy consumption of the data center, respectively.

The performances of the proposed algorithms are compared for each cluster of tasks separately without considering task placement constraints. As future work, we will investigate energy-aware virtual machine placement algorithms that consider the aforementioned constraints and the characteristics of each group of tasks. Furthermore, the right policy will be selected according to the specifications of each group.

We will also investigate online learning algorithms for defining the task capacity of virtual machines in replacement of the static methods explored in this paper. For more energy savings, VM consolidation techniques and virtual machine resizing options will also be explored.

ACKNOWLEDGEMENTS

We thank Mehran Garmehi, Yaser Mansoori, Adel Nadjaran Toosi and Atefeh Khosravi for their valuable insights for the improvements of the paper.

FUNDING

This work was partially supported by a Discovery Grant from the Australian Research Council.

REFERENCES

[1] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I. (2009) Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comput. Syst.*, **25**, 599–616.

[2] Armbrust, M. *et al.* (2010) A view of cloud computing. *Commun. ACM*, **53**, 50–58.

[3] Kaplan, J.M., Forrest, W. and Kindler, N. (2008) Revolutionizing data center energy efficiency. Technical Report. http://www.mckinsey.com/clientservice/bto/pointofview/pdf/revolutionizing_data_center_efficiency.pdf (accessed June 5, 2015).

[4] Buyya, R., Beloglazov, A. and Abawajy, J. (2010) Energy-efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. *Proc. 16th Int. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, Las Vegas, USA, July 12–15, pp. 6–17. World Academy of Science, Engineering and Technology, San Diego, USA.

[5] Greenberg, S., Mills, E., Tschudi, B., Rumsey, P. and Myatt, B. (2006) Best Practices for Data Centers: Lessons Learned from Benchmarking 22 Data Centers. *Proc. ACEEE Summer Study on Energy Efficiency in Buildings*, Asilomar, USA, August 13–18, pp. 76–87. ACEEE, Washington, USA.

[6] Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S. and McKeown, N. (2010) ElasticTree: Saving Energy in Data Center Networks. *Proc. 7th USENIX Conf. Networked Systems Design and Implementation (NSDI'10)*, San Jose, USA, April 28–30. USENIX, Berkeley, USA,

[7] Greenberg, A., Hamilton, J., Maltz, D.A. and Patel, P. (2008) The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Comput. Commun. Rev.*, **39**, 68–73.

[8] Zheng, K., Wang, X., Li, L. and Wang, X. (2014) Joint Power Optimization of Data Center Network and Servers with Correlation Analysis. *Proc. IEEE INFOCOM 2014*, Toronto, Canada, April 27–May 2, pp. 2598–2606. IEEE, Piscataway, USA.

[9] Mishra, A.K., Hellerstein, J.L., Cirne, W. and Das, C.R. (2010) Towards characterizing cloud backend workloads: insights from Google compute clusters. *ACM SIGMETRICS Perform. Eval. Rev.*, **37**, 34–41.

[10] UCB/EECS-2009-28 (2010) *Analysis and Lessons from a Publicly Available Google Cluster Trace*. University of California at Berkeley, Berkeley, USA.

[11] Solis Moreno, I., Garraghan, P., Townend, P. and Xu, J. (2014) Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. *IEEE Trans. Cloud Comput.*, **2**, 208–221.

[12] Di, S., Kondo, D. and Cappello, F. (2013) Characterizing Cloud Applications on a Google Data Center. *Proc. 42nd Int. Conf. Parallel Processing (ICPP 2013)*, Lyon, France, October 1–4, pp. 468–473. IEEE, Piscataway, USA.

[13] Reiss, C., Wilkes, J. and Hellerstein, J.L. (2011) *Google Cluster-usage Traces: Format+ Schema*. Google, Inc. Mountain View, USA.

[14] Zhang, Q., Hellerstein, J.L. and Boutaba, R. (2011) Characterizing Task Usage Shapes in Google's Compute Clusters. *Proc. 5th Int. Workshop on Large Scale Distributed Systems and Middleware*, Seattle, USA, September 2–3, pp. 1–6. ACM, New York, USA.

[15] Sharma, B., Chudnovsky, V., Hellerstein, J.L., Rifaat, R. and Das, C.R. (2011) Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters. *Proc. 2nd ACM Symp. Cloud Computing (SOCC'11)*, Cascais, Portugal, October 26–28, pp. 3:1–3:14. ACM, New York, USA.

[16] Moreno, I.S., Garraghan, P., Townend, P. and Xu, J. (2013) An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models. *Proc. IEEE 7th Int. Symp. Service Oriented System Engineering (SOSE 2013)*, San Francisco Bay, USA, March 25–28, pp. 49–60. IEEE, Piscataway, USA.

[17] Di, S. and Cappello, F. (2014) GloudSim: Google trace based cloud simulator with virtual machines. *Softw.: Practice and Exper*.

[18] Kansal, A., Zhao, F., Liu, J., Kothari, N. and Bhattacharya, A.A. (2010) Virtual Machine Power Metering and Provisioning. *Proc. 1st ACM Symp. Cloud Computing (SoCC'10)*, Indianapolis, USA, June 10–11, pp. 39–50. ACM, New York, USA.

[19] Nathuji, R. and Schwan, K. (2007) VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. *Proc. 21st ACM SIGOPS Symp. Operating Systems Principles (SOSP'07)*, Stevenson, WA, USA, October 14–17, pp. 265–278. ACM, New York, USA.

[20] Kim, K.H., Beloglazov, A. and Buyya, R. (2009) Power-aware Provisioning of Cloud Resources for Real-time Services. *Proc. 7th Int. Workshop on Middleware for Grids, Clouds and e-Science (MGC'09)*, Champaign, USA, November 30–December 4, pp. 1:1–1:6. ACM, New York, USA.

[21] Garraghan, P., Townend, P. and Xu, J. (2013) An Analysis of the Server Characteristics and Resource Utilization in Google Cloud. *Proc. 2013 IEEE Int. Conf. Cloud Engineering (IC2E 2013)*, San Francisco, USA, March 25–27, pp. 124–131. IEEE Computer Society, Washington, USA.

[22] Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H. and Kozuch, M.A. (2012) Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. *Proc. 3rd ACM Symp. Cloud Computing (SoCC 2012)*, San Jose, USA, October 14–17, pp. 7:1–7:13. ACM, New York, USA.

[23] Di, S., Kondo, D. and Cirne, W. (2012) Characterization and Comparison of Cloud Versus Grid Workloads. *Proc. 2012 IEEE Int. Conf. Cluster Computing (CLUSTER'12)*, Beijing International Convention Center Beijing, China, September 24–28, pp. 230–238. IEEE Computer Society, Washington, USA.

[24] Pham, D.T., Dimov, S.S. and Nguyen, C. (2005) Selection of k in K-means clustering. *Proc. Inst. Mech. Eng. C: J. Mech. Eng. Sci.*, **219**, 103–119.

[25] Pelleg, D. and Moore, A.W. (2000) X-means: Extending K-means with Efficient Estimation of the Number of Clusters. *Proc. 17th Int. Conf. Machine Learning (ICML'00)*, Stanford, USA, pp. 727–734. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA.

[26] Singh, G. *et al.* (2008) Workflow Task Clustering for Best Effort Systems with Pegasus. *Proc. 15th ACM Mardi Gras Conf. (MG'08)*, Baton Rouge, USA, January 31-February 2, pp. 9:1–9:8. ACM, New York, USA.

[27] Muthuvelu, N., Vecchiola, C., Chai, I., Chikkannan, E. and Buyya, R. (2013) Task granularity policies for deploying bag-of-task applications on global grids. *Future Generation Comput. Syst.*, **29**, 170–181.

[28] Wang, L., Tao, J., von Laszewski, G. and Chen, D. (2010) Power Aware scheduling for Parallel Tasks Via Task Clustering. *Proc.*

IEEE 16th Int. Conf. Parallel and Distributed Systems (ICPADS 2010), Shanghai, China, December 7–10, pp. 629–634. IEEE Computer Society, Washington, USA.

[29] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A. and Buyya, R. (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw.: Practice Exper.*, **41**, 23–50.

[30] Barham, P. *et al.* (2003) Xen and the Art of Virtualization. *Proc. 19th ACM Symp. Operating Systems Principles (SOSP'03)*, Bolton Landing, USA, October 19–22, pp. 164–177. ACM, New York, USA.

[31] Zhang, Q., Cheng, L. and Boutaba, R. (2010) Cloud computing: state-of-the-art and research challenges. *Internet Serv. Appl.*, **1**, 7–18.

[32] Beloglazov, A., Buyya, R., Choon Lee, Y. and Zomaya, A. (2011) A taxonomy and survey of energy-efficient data centers and cloud computing systems. In Zelkowitz, M. (ed.), *Advances in Computers*, pp. 47–111. Elsevier.

[33] Chieu, T., Mohindra, A., Karve, A. and Segal, A. (2009) Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *Proc. IEEE Int. Conf. e-Business Engineering (ICEBE'09)*, Macau, China, October 21–23, pp. 281–286. IEEE, Piscataway, USA.

[34] Van den Bossche, R., Vanmechelen, K. and Broeckhove, J. (2010) Cost-optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. *Proc. 3rd Int. Conf. Cloud Computing*, Miami, USA, July 5–10, pp. 228–235. IEEE, Piscataway, USA.

[35] Fang, Y., Wang, F. and Ge, J. (2010) A task scheduling algorithm based on load balancing in cloud computing. In Wang, F., Gong, Z., Luo, X. and Lei, J. (eds.), *Web Information Systems and Mining*, Lecture Notes in Computer Science 6318, pp. 271–277. Springer, Heidelberg, Germany.

[36] Merkel, D. (2014) Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.*, **2014**.

[37] Hartigan, J. and Wong, M. (1979) Algorithm as 136: a K-means clustering algorithm. *Appl. Stat.*, **28**, 100–108.

[38] Schwarz, G. (1978) Estimating the dimension of a model. *Ann. Stat.*, **6**, 461–464.

[39] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B. (2002) Automatically characterizing large scale program behavior. *ACM SIGARCH Comput. Archit. News*, **30**, 45–57.

[40] Kass, R.E. and Wasserman, L. (1995) A reference Bayesian test for nested hypotheses and its relationship to the schwarz criterion. *J. Am. Stat. Assoc.*, **90**, 928–934.

[41] Gu, G., Perdisci, R., Zhang, J. and Lee, W. (2008) Botminer: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection. *Proc. 17th Conf. Security Symp. (SS'08)*, San Jose, USA, July 24–August 1, pp. 139–154. USENIX, Berkeley, USA.

[42] Dy, J.G. and Brodley, C.E. (2004) Feature selection for unsupervised learning. *J. Mach. Learn. Res.*, **5**, 845–889.

[43] Garraghan, P., Moreno, I.S., Townend, P. and Xu, J. (2014) An analysis of failure-related energy waste in a large-scale cloud environment. *IEEE Trans. Emerg. Topics Comput.*, **2**, 166–180.

[44] Corporation, S. P. E. Specpower_ssj2008 results. http://www.spec.org/power_ssj2008/results/ (accessed June 4, 2015).

[45] Blackburn, M. and Grid, G. (eds) (2008) *Five Ways to Reduce Data Center Server Power Consumption*. The Green Grid.