An Energy and Deadline Aware Resource Provisioning, Scheduling and Optimization Framework for Cloud Systems

Yue Gao Yanzhi Wang Sandeep K. Gupta Massoud Pedram
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
yuegao@usc.edu, yanzhiwa@usc.edu, sandeep@usc.edu, pedram@usc.edu

Abstract— **Cloud computing has attracted significant attention due to the increasing demand for low-cost, high performance, and energy-efficient computing. Profit maximization for the cloud service provider (CSP) is a key objective in the large-scale, heterogeneous, and multi-user environment of a cloud system. This paper addresses the problem of minimizing the operation cost of a cloud system by maximizing its energy efficiency while ensuring that user deadlines as defined in Service Level Agreements are met. The workload in the cloud system can be modeled as independent batch requests or as task graphs with dependencies. This paper adopts the latter modeling approach, which provides more opportunities for energy and performance optimizations, thus enabling the CSP to meet user deadlines at lower operation costs. However, these optimizations require additional supporting efforts e.g., resource provisioning, virtual machine placement, and task scheduling, which are addressed in a holistic manner in the proposed framework. In the envisioned cloud environment, users can construct their own services and applications based on the available set of virtual machines, but are relieved from the burden of resource provisioning and task scheduling. The CSP will then exploit data parallelism in user workloads to create an energy and deadline-aware cloud platform.**

I. INTRODUCTION

Cloud computing has been envisioned as the next-generation computing paradigm because of its advantages in terms of on-demand service, ubiquitous network access, location independent resource pooling, and transference of risk [1]. Cloud computing shifts the computation and storage resources from the network edges to a "Cloud" from which users are able to access resources on demand and from anywhere in the world [2, 3, 4]. Cloud service providers (CSPs), who own large data centers and server clusters, are incentivized by profits that they accrue by charging the end users for the service access. Users are attracted by the opportunity to reduce or eliminate costs associated with implementing these services "in-house".

Virtualization technology [5], which is an important driver of the cloud computing, structures the server and data center resources (e.g., CPU, memory, etc.) into virtual platforms, called *virtual machines* (VMs). A VM is the basic deployment and management unit in the cloud computing. Users rent VMs from the CSP to construct their own services and/or applications, whereas the CSP determines *how* to allocate the physical resources to host the VMs and *where* to place the VMs provided to various users.

There are inherent conflicts between the goals of the users and the CSP. The CSP is profit driven: it aims to minimize the operation cost, or equivalently, maximize the operation efficiency, while attracting as many customers as possible. It is well known that the bulk of the CSP operating cost stems from energy consumption [4, 6], both dynamic and static. The situation is exacerbated by the fact that today's servers consume up to 50% of their peak power when idle [7]. Energy consumption may be minimized by (i) selectively shutting down servers, and (ii) balancing the resource utilization for all active servers. Cloud users, on the other hand, demand performance for their submitted workloads.

To bridge the gap between users and the CSP, Service Level Agreements (SLAs) are drafted, which document the mutually agreed service quality, including deadlines, and privacy and security specifications. When multiple users coexist in the public cloud, the CSP has to satisfy their individual needs, while managing the vast, heterogeneous and elastic cloud resources to maximize energy efficiency. When developing a systematic approach to achieve this goal, we follow a set of principles that we coin as the "Triple-A Guidelines":

- **Accurate modeling of the cloud platform**
  The cloud platform modeling should only consider aspects of the physical environment that are crucial to the optimization, so as to reduce the computational complexity while retaining adequate model accuracy.
- **Appropriate modeling of user workloads**
  Modeling of the user workloads determines the optimization processes that are invoked by the CSP. Currently there are two types of workload models in a cloud computing system, which target different application sets, with one key differentiating factor: the inclusion of task dependencies [8]. In the first model, at a coarse granularity, each user workload can be represented as an atomic task, which is independent of other tasks. The entire collection of tasks forms a task batch. Such an abstraction is suitable for certain hosting services. It also greatly lowers the CSP scheduling complexity, which can be attributed to the statistical estimations of workloads, such as Poisson arrival rates and response times [9, 10]. In the second model, each user workload is viewed at a fine granularity by using a task graph with output dependencies. This formulation mainly targets large scientific and engineering applications [11, 12, 13]. The authors in [8] conveniently name the scheduling frameworks under the two workload models "batch mode scheduling" and "dependency mode scheduling", and provide a survey of the related work.
- **Acceptable complexity**
  The CSP should ensure that the optimization process itself does not incur large runtime or power overheads that offset the actual benefits gained from the extensive optimization effort, especially for dependency mode scheduling.

978-1-4799-1417-3/13/$31.00 ©2013 IEEE

In this paper, we adopt the task graph workload model and present a novel cloud resource provisioning, task scheduling and energy cost optimization framework for the CSP, which has the following properties:

(1) Workloads are modeled as a collection of multiple task graphs with output dependencies. Other cloud system optimization frameworks that also operate on a similar task graph based workload model include: Nephele [12], Pegasus [14], VGrADS [15], Particle Swarm Optimization (PSO) based frameworks [16, 17], Surfer [13], DisNet [18], Dryad [11] and BC/Azure [19].

(2) The cloud platform is modeled as a weighted graph, which is capable of representing heterogeneous servers with varied resource capacities, power efficiencies and communication bottlenecks.

(3) Users request VMs according to a pay-as-you-go billing agreement (as in Amazon EC2 [20]), but are relieved from the burden of resource provisioning and task scheduling.

(4) The CSP addresses deadline-aware resource provisioning, VM placement, task scheduling and energy cost optimization in a holistic fashion.

(5) The scheduling algorithm itself is fully parallelizable to take advantage of the cloud resources.

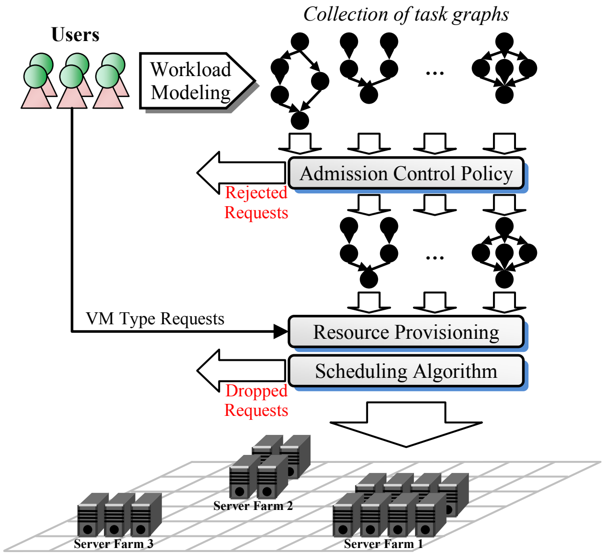


Fig. 1 Overview of the cloud environment

The cloud environment is illustrated pictorially in Fig. 1. Task scheduling is performed by the CSP. The ultimate goal of CSP is to first *selectively accept* workload requests through the admission control policy, and then *allocate* appropriate amount of VMs for each workload request, *place* those VMs on physical servers, *consolidate* VMs when necessary, and finally *schedule* all the accepted requests to meet SLA deadlines, *drop* requests when necessary, while *minimizing the global energy cost*. It harnesses the task parallelism benefits from task graph based workloads, while also taking advantage of global optimizations that to the best of our knowledge, currently only exist in batch mode scheduling frameworks. Our algorithm operates offline, but it can be transformed into an adaptive online algorithm through recursive triggering.

II. RELATED WORK

For the CSP, the first step in cloud operation optimization is resource provisioning, which is essentially allocating appropriate amount of computing resources (in the form of physical servers and VMs) to satisfy user demands. This problem can be formulated and solved differently according to how the cloud platform is modeled. In its simplest form, the cloud platform can be envisioned as disjoint, homogenous servers, while workloads are independent requests with given arrival rates. Under these assumptions, resource allocation can be solved through modified bin packing [6, 21] or custom workload prediction algorithms inspired by queuing theory [9, 10, 22]. A more accurate modeling of the hardware would involve communication capacities. In [23] the authors adopt a graph model for a service enriched cloud environment, and solve the problem of server provisioning and message routing through MILP (mixed integer linear programming). Other variants are also introduced, such as bundled VM requests [24], VM performance variability [25, 26], multiple server sleep states [27], or price auctions [28].

After resource provisioning, the mapping of applications or VMs to physical servers is performed. A key goal of this process is to sustain near-optimal utilization levels for each server in order to achieve high energy efficiency [6]. This problem is similar to the classical load balancing problem in internet services [7], which can be concurrently solved during resource allocation for independent workloads. For example, the bin packing algorithm in [6] not only minimizes the total number of servers deployed, but also prevents servers from being starved or overburdened. Reference [29] formulates the problems of VM provisioning and placement as a constraint satisfaction function. Other classical solutions, such as MCMF (minimum cost maximum flow) [30], can be used. When workloads are not known *a priori*, dynamic workload migration [31] or VM reallocation [32, 33] becomes very beneficial. The temporal and/or spatial variations in electricity pricing make load balancing in the cloud unique. Related approaches include MILP [34], primal-dual [35], bargaining games [36], and probabilistic predictions [37].

Scheduling task graph based workloads with dependencies is very different from batch scheduling in cloud computing systems. At a high level, this problem *loosely* resembles chip multiprocessor (CMP) scheduling from the parallel/cluster computing community [38, 39, 40]. Unfortunately, techniques developed there are not directly applicable for cloud users, mainly due to the opaqueness of the public cloud [12]; they are not suitable for the CSP either, due to the co-existence of multiple competing users and more importantly the elasticity of the underlying cloud computing hardware. Improved scheduling frameworks such as Nephele [12] and Pegasus [13] fully embrace the dynamic nature of the computing resources in cloud systems, but perform optimizations from the perspective of individual users. With a lack of awareness of other competing users and the entire cloud resource map, they cannot capture *global* CSP management opportunities such as admission control, VM placement and consolidation, which are normally only considered in batch mode scheduling. Their results are evaluated based on completion times and resource utilizations for a single user, not global energy consumption.

III. USER WORKLOAD MODEL

User Workload Requests

We use directed acyclic graphs (DAGs) to model users' workload requests. The entire workload is represented as a collection of N disjoint DAGs: $\{G_1(\boldsymbol{V}_1, \boldsymbol{E}_1), G_2(\boldsymbol{V}_2, \boldsymbol{E}_2), ..., G_N(\boldsymbol{V}_N, \boldsymbol{E}_N)\}$. Each DAG G_a ($1 \le a \le N$) represents a workload request, and each vertex T_i^a ($1 \le i \le |\boldsymbol{V}_a|$) in G_a embodies a task. Without the loss of generality, we assume that each workload request is an application that belongs to a separate user. Hence in this paper, an application (e.g. the 100GB integer sort program in [12]) is equivalent to a user workload request.

An edge from T_i^a to T_j^a in G_a denotes that T_j^a is dependent on the output of T_i^a. The weight of the edge $W_{i,j}^a$ represents the amount of data that needs to be passed from the predecessor task (T_i^a) to the successor task (T_j^a). Fig. 2 provides an example illustration of a group of N applications.

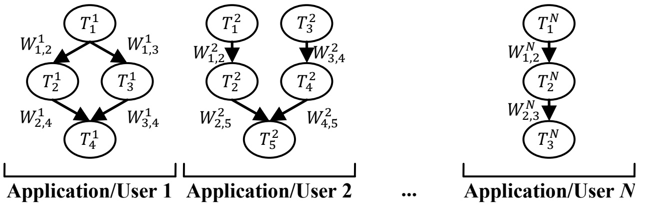


Fig. 2 Application Model

Task Characteristics

Tasks are run on virtual machines (VMs). VMs are categorized into K distinct types: $\{VM_1, VM_2, ..., VM_K\}$, each VM_g ($1 \le g \le K$) is coupled with a two-tuple integer set that specifies the resource requirement for a VM of type g: $\{R_{CPU}^g, R_{MEM}^g\}$. R_{CPU}^g and R_{MEM}^g represent the required amount of CPU and memory resources to host a VM_g, respectively [20].

Each task T_i^a is also coupled with a two-tuple integer set $\{\theta_i^a, L_i^a\}$. θ_i^a represents the type of VM on which T_i^a can only execute. This information can be provided externally as job descriptions, or deduced internally through collected statistics [12]. L_i^a is the worst case execution time (WCET) of T_i^a when it is executed on a VM of type θ_i^a. Both θ_i^a and L_i^a are inputs to the optimization algorithm. We acknowledge the fact that WCET analysis is a common problem that affects the practical implementation of almost all scheduling algorithms. The team behind the Nephele project envisions a learning mechanism that enables the cloud operator to estimate execution times from past traces.

Resource Requests

Besides its workload request, a user must also request computational resources from the CSP. Naturally, the CSP should charge users based on their resource requests according to a predetermined billing contract. Similar to [24], in our cloud system resource requests come in the form of bundles of VM types. Each user only specifies *types* of VMs that are needed, but not the *quantity* of each requested VM type. Therefore the users need not be concerned with details of the resource allocation. We assume that each user has decided about his/her resource requests according to his/her own financial budgets. Note that currently, commercial cloud systems have not yet standardized the resource request format; our assumption is based on past research, and designed in a way so that the elasticity of cloud hardware can be fully exploited.

VM requests are expressed in the following way: each application (G_a) is associated with a binary array $\boldsymbol{U}^a$ of K members (recall that K is the total number of VM types): $\{U_1^a, U_2^a, ..., U_K^a\}$, where $U_g^a = 1$ ($1 \le g \le K$) indicates that VM_g is requested by User a, and $U_g^a = 0$ otherwise. $\boldsymbol{U}^a$ must guarantee that no task is mapped to an unrequested VM type, that is if $g = \theta_i^a$, then it must follow that $VM_g \in \{VM_1, ..., VM_K\}$ and $U_g^a = 1$.

Each user application is then fully defined by its workload request (G_a) and VM type request ($\boldsymbol{U}^a$), but scheduling will not be carried out by the user. The reasoning behind this is threefold: (i) the cloud platform is opaque to the user [12], (ii) the users may not possess the necessary computing power, and (iii) the CSP can achieve higher efficiency with total scheduling freedom.

Deadlines

Although users can neither request more VMs of the same type, nor schedule their own workload, they can still gain control of their workload performance by specifying deadlines in the SLA. The deadline for the workload request from User a (G_a) is denoted as $L_{deadline}^a$. In this paper all deadlines are *hard*.

Generally, when User a gives a more aggressive deadline, the CSP will allocate more VM resources for G_a, so that tasks in G_a can be executed in parallel and complete their execution sooner. However, user workloads are subject to the admission control policy detailed at the end of the next section, so applications with unsatisfiable (too tight) deadlines will be rejected prior to scheduling or they will be dropped during scheduling.

IV. CLOUD PLATFORM MODEL

The cloud consists of a set of M servers: $\{D_1, D_2, ..., D_M\}$, and is modeled as a undirected graph of M vertices, each representing a server. The weight of each edge (D_x, D_y), $B_{x,y}$, represents the communication capacity between D_x and D_y.

Several neighboring servers may form a server farm with local connections. Server farms can communicate with each other through high speed channels. The distance between servers and channel bandwidths will be reflected in the $B_{x,y}$ values. By default $B_{x,x} = \infty$, i.e. tasks executing on the same server do not incur any communication overhead. Also, we assume a *path* exists between any two servers, either through a direct link or through multi-hops. A multi-hop path will be abstracted as a connecting edge with a low $B_{x,y}$ value. Fig. 3 provides an example illustration of a cloud platform consisting of nine servers clustered into two server farms, one with six servers and the other with three servers. Note that local connections may also be heterogeneous, and for clarity not all server connections are shown.

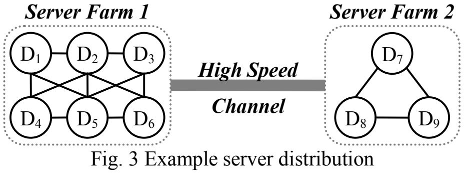


Fig. 3 Example server distribution

Virtual Machine Configurations

During operation, each server D_x is associated with an integer array $\boldsymbol{Q}^x$ of K members: $\{Q_1^x, Q_2^x, \ldots, Q_K^x\}$, where Q_g^x indicates that Q_g^x number of type g VMs (VM_g) are hosted on D_x. $\boldsymbol{Q}^x$ is dynamic since the it may change over time due to VM tear downs and reconfigurations initiated by the CSP. We denote the $\boldsymbol{Q}^x$ configuration at time t as $\boldsymbol{Q}^x(t)$. Each server D_x contains finite amount of resources, namely C_{CPU}^x and C_{MEM}^x amount of CPU and memory, respectively. Clearly, the VM configuration of D_x must abide to the total amount of resources at all times, i.e. $\forall\ t$ we have $\sum_{g=1}^{K} Q_g^x(t) R_{CPU}^g \le C_{CPU}^x$ and $\sum_{g=1}^{K} Q_g^x(t) R_{MEM}^g \le C_{MEM}^x$.

Energy Consumption

The power consumption of D_x at time t includes the static power consumption $P_{static}^x(t)$ and the dynamic power consumption $P_{dynamic}^x(t)$. Both are correlated with the utilization rate of D_x at time t: $Util_x(t)$. We evaluate $Util_x(t)$ by considering only the CPU requirements of the hosted VMs indicated in $\boldsymbol{Q}^x(t)$, and do not differentiate between VMs that are running tasks and idle VMs, since background CPU activities are needed even during idle periods.

$$Util_x(t) = \frac{\sum_{g=1}^{K} Q_g^x(t) \cdot R_{CPU}^g}{C_{CPU}^x} \times 100\%.$$

$P_{static}^x(t)$ is constant when $Util_x(t) > 0$, 0 otherwise. The relationship between $P_{dynamic}^x(t)$ and $Util_x(t)$ is much more complex. Servers have optimal utilization level in terms of performance-per-watt [6], which we define as Opt_x for D_x. It is commonly accepted that for modern servers $Opt_x \approx 0.7$, and the increase in power consumption beyond this operating point is more drastic than when $Util_x(t) < Opt_x$ [4, 7]. Even for identical utilization levels, the energy efficiency of different servers may vary [41]. This is captured by the coefficients α_x and β_x, representing the power consumption increase of D_x when $Util_x(t) < Opt_x$ and $Util_x(t) \ge Opt_x$, respectively. $P_{dynamic}^x(t)$ is then calculated as:

$$\begin{cases} Util_x(t) \cdot \alpha_x & (Util_x(t) < Opt_x) \\ Opt_x \cdot \alpha_x + (Util_x(t) - Opt_x)^2 \cdot \beta_x & (Util_x(t) \ge Opt_x) \end{cases}.$$

We would like to point out that the exact formulations of $P_{dynamic}^x(t)$ do not undermine our analysis, as long as its increase is faster when $Util_x(t) \ge Opt_x$ than when $Util_x(t) < Opt_x$. Fig. 4 plots $P_{dynamic}^x(t)$ for D_x when $\alpha_x = 0.5$, $\beta_x = 10$ and $Opt_x = 0.7$ under different CPU utilization levels.

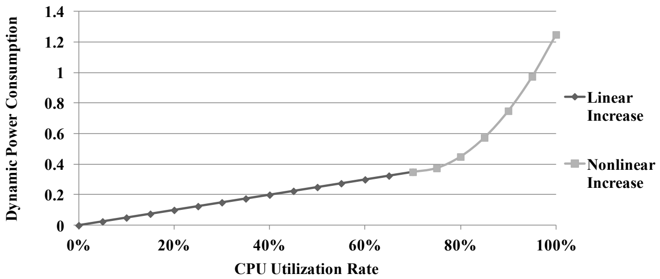


Fig. 4. Dynamic power increase with respect to server utilization

Suppose the upper bound of the maximum schedule length of all applications is L_{max}. The total energy consumption ($COSP$) is the sum of the power consumption across all servers throughout the operation timeline:

$$COSP = \sum_{x=1}^{M} \left(\sum_{t=1}^{L_{max}} \left(P_{static}^x(t) + P_{dynamic}^x(t) \right) \right).$$

Since minimizing energy consumption is equivalent to minimizing energy cost, in this paper we will use the two phrases energy cost and energy consumption interchangeably.

Admission Control

The purpose of admission control is to identify and eliminate user workloads that are overly resource hungry. These users require extensive VM reservations that can potentially cause resource hogging, creating scheduling difficulties for other users and increasing global power consumption. In this paper we adopt a two pass admission control policy that sifts users based on their deadline specifications.

Prior to scheduling, each application will be examined to see whether it can be scheduled under the specified deadline given infinite amount of resources. Unlike the core scheduling procedure, this "schedulability" analysis can be performed in linear time. If the deadline is violated, then the workload request will be *rejected*.

During scheduling, users compete for VM resources under the regulation of the CSP, as a result some deadlines may not be met due to constrained resources. If the deadline violations persist after considerable optimization efforts, the associated workload requests will be *dropped*.

V. CLOUD OPERATION

In this section we identify the key processes during cloud operation and the tradeoffs in cloud operation optimization.

Virtual Machine Provisioning and Placement

The CSP decides the VM allocations and placement of tasks on these VMs. Although the CSP would prefer VMs to be time-shared by all users, we conservatively bind each VM to a specific user to promote user context preservation, security and privacy. As a result, any VM will be dedicated to a single user until the VM is torn down by the CSP.

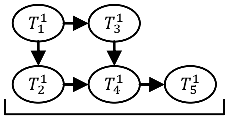
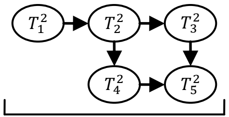
Temporal Scheduling of Tasks

We define a *ready task* as a task with its dependencies satisfied. Suppose for a ready task T_i^a we have $\theta_i^a = g$. If T_i^a were to be scheduled in a type g VM located in D_x, two conditions should be met: (i) the destination VM is available and is dedicated to User a, and (ii) all necessary output data transfers have been completed. Communication will only be needed when one or more predecessors of T_i^a completed execution on a server other than D_x. Suppose a predecessor task of T_i^a, $T_{Pred(i)}^a$, resides on D_y ($x \neq y$), then the output data transmission time is given by $W_{Pred(i),i}^a / B_{x,y}$.

Schedule Evaluation

The quality of a schedule is determined by two factors: (i) the global energy cost ($COSP$) and (ii) the number of dropped requests due to deadline violations. A schedule may be energy efficient, but ultimately infeasible due to excessive deadline violations. When adjusting this schedule to satisfy deadlines and minimize the number of dropped requests, energy efficiency is often sacrificed. We elaborate with a simple example below. Assume the CSP is servicing two users, the workload information are presented in Table I.

Table I. Task graph and task latency

	Task (T)	Latency (L)	VM Mapping (θ)	
User 1	T_1^1	6	1	 Application/User 1
	T_2^1	3	1	
	T_3^1	3	1	
	T_4^1	6	1	
	T_5^1	1	1	
User 2	**Task (T)**	**Latency (L)**	**VM Mapping (θ)**	 Application/User 2
	T_1^2	6	1	
	T_2^2	6	1	
	T_3^2	3	1	
	T_4^2	3	1	
	T_5^2	1	1	

If the workloads are treated as atomic independent entities, then the obvious schedule is to place both applications on the most energy-efficient server, say Server 5 (D_5), as long as the allocated VMs (two Type 1 VMs) will not push $Util_5(t)$ past Opt_5. The result is shown in Fig. 5a, the length of the schedule is 19 time units. If $L^1_{deadline} = L^2_{deadline} = 16$ (indicated by the dotted red line), then this schedule violates deadlines for both users. For the sake of discussion, we calculate the energy cost of this schedule before both user requests are dropped:

$$COSP_a = \left(\frac{2R^1_{CPU}}{C^5_{CPU}} \cdot \alpha_5 + P^5_{static}\right) \cdot 19.$$

To meet the deadline for both users, the CSP can exploit data parallelisms within G_1 and G_2, an opportunity that is not available to batch scheduling. There are several alternatives when doing so, leading to different solutions with different energy costs. A greedy approach would produce the schedule in Fig. 5b, where all VMs are hosted on D_5. The reduced completion time for both users come from the migration of T^1_3 and T^2_4 to newly allocated VMs, which may overburden D_5. If $Util_x(t)$ now surpassed Opt_5 because of the two new VMs, then the energy cost of this schedule would be:

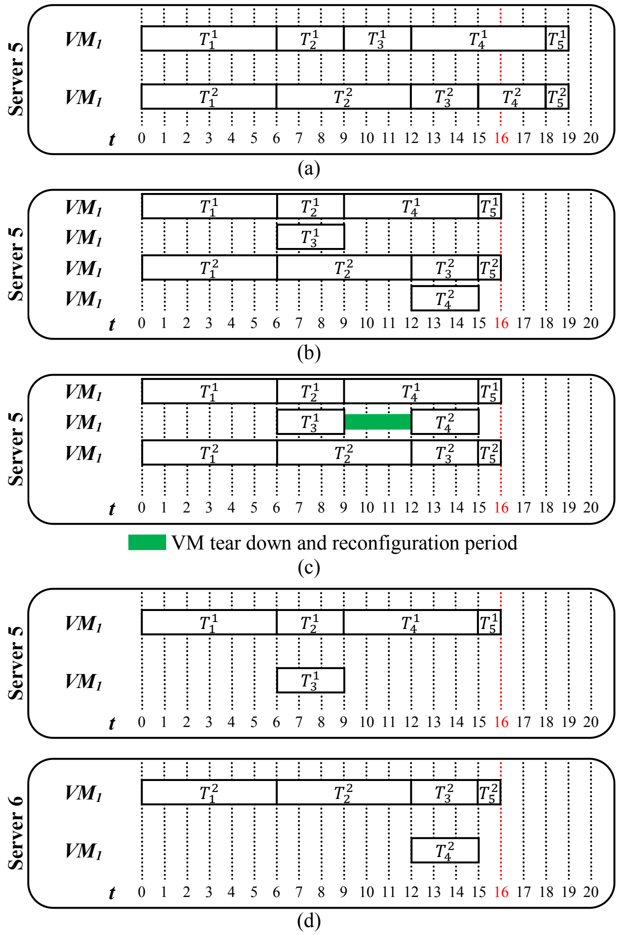
$$COSP_b = \left[Opt_5 \cdot \alpha_5 + \left(\frac{4R^1_{CPU}}{C^5_{CPU}} - Opt_5\right)^2 \cdot \beta_5 + P^5_{static}\right] \cdot 16.$$

The schedule in Fig. 5b can potentially be further refined into the schedule in Fig. 5c. While straightforward in concept, there are actually a series of actions related to this refinement, namely the tear down of the Type 1 VM reserved for User 1 at $t = 9$, followed by a configuration of a Type 1 VM reserved for User 2. We do not delve into the technicalities regarding this process, but accept its overhead. The energy cost of this schedule in Fig. 5c is:

$$COSP_c = \left[Opt_5 \cdot \alpha_5 + \left(\frac{3R^1_{CPU}}{C^5_{CPU}} - Opt_5\right)^2 \cdot \beta_5 + P^5_{static}\right] \cdot 16.$$

Another solution is to explore other servers, say Server 6 (D_6). The CSP can offset an entire user to D_6, so that neither D_5 nor D_6 will be overburdened. The schedule in Fig. 5d chose User 2 to move to D_6. D_6 is not necessarily in the same server farm as D_5. The energy cost of this schedule is:

$$COSP_d = \left[\left(\frac{2R^1_{CPU}}{C^5_{CPU}} \cdot \alpha_5 + P^5_{static}\right) + \left(\frac{2R^1_{CPU}}{C^6_{CPU}} \cdot \alpha_6 + P^6_{static}\right)\right] \cdot 16.$$


Server 5
VM1
T_1^1 T_2^1 T_3^1 T_4^1 T_5^1
T_1^2 T_2^2 T_3^2 T_4^2 T_5^2
t
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Server 6
🟩 VM tear down and reconfiguration period

(a)

(b)

(c)

(d)

Fig. 5 Example schedules

In Fig. 6 we illustrate how server energy efficiency can affect scheduling decisions. Suppose $R^1_{CPU} = 1$, $C^5_{CPU} = C^6_{CPU} = 4$, $\alpha_5 = \alpha_6 = 0.5$, $\beta_5 = \beta_6 = 10$, $Opt_5 = Opt_6 = 0.7$, and the value of P^5_{static} and P^6_{static} is varied. For simplicity, we assume $P_{static} = P^5_{static} = P^6_{static}$. $COSP_a$ is generally the best schedule if only energy consumption is concerned. $COSP_b$ and $COSP_d$ are all greater than $COSP_a$, and the comparison between these two values depends on the P_{static} value. It also depends on other parameters such as β_5/β_6, which is not shown here. If the packing in Fig. 5c can be realized (which is a rare occasion when multiple users with large task graphs are considered), then $COSP_c$ is very promising since it satisfies all deadlines with an energy cost even lower than $COSP_a$.

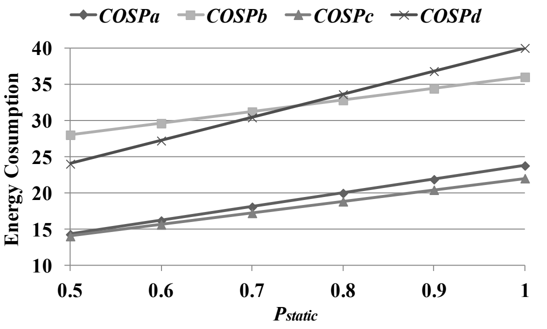


Fig. 6. The *COSP* evaluation

The key takeaways from this case study are:

- Accelerating application execution through additional VM allocations and parallel execution will often incur energy cost overheads for the CSP.
- Many possible schedules exist when performing VM allocations and task migrations, among which a schedule with no deadline violations and a low energy cost is preferred. The best allocation and migration scheme depends on the characteristics of the cloud platform and the workloads. The scheduling framework described in the next section will undertake those challenges.

VI. THE GMaP FRAMEWORK

In this section we illustrate our "Guided Migrate and Pack" (GMaP) scheduling and optimization framework for the CSP phase by phase. GMaP is based on *beam search* in order for it to be fully parallelizable, which is crucial when the CSP is utilizing the cloud resources at its disposal to run GMaP.

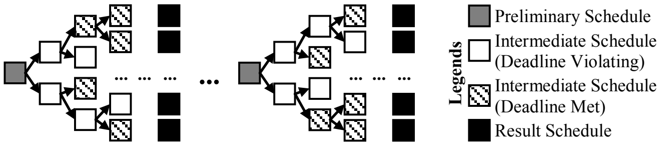

Legends
Preliminary Schedule
Intermediate Schedule (Deadline Violating)
Intermediate Schedule (Deadline Met)
Result Schedule

Fig. 7 Abstraction of the GMaP algorithm

Fig. 7 provides an abstraction of the GMaP algorithm. The basic idea of GMaP is to start with schedules that are deadline oblivious but energy efficient. It will then co-optimize latency for the deadline violating applications and the global energy cost. First, *seed schedules* for each application are generated in parallel, and assembled into the Preliminary Schedule Set (PSS). Then treating each *preliminary schedule* in the PSS as root, the main body of GMaP is executed in parallel, eventually producing multiple result schedules from which the best schedule will be promoted to be the solution. At first, GMaP will be *energy aware* but *deadline oriented*. Once all deadlines have been met, then GMaP will only focus on the energy cost, unless new deadline violations appear.

A valid argument would be that the scheduling process itself may become overly power hungry or time consuming. Fortunately, the resource requirement and run time of the GMaP can be adjusted based on the power portfolio of the target cloud environment in two ways: (i) the number of roots can be sized to any natural number and (ii) each search tree size can be individually tuned.

Phase 1: Seed Schedule Generation

The seed schedule for User a is the schedule for G_a based on $\boldsymbol{U^a}$, while assuming only one VM is instantiated for each requested VM type. Also, all VMs are mapped to a single imaginary server. The seed schedules can be derived in parallel for all applications through any scheduling algorithm [38]. In this paper we used a modified greedy algorithm.

Phase 2: Application Characterization

In this phase, each application is affiliated with different characterizing parameters to aid the following phase. The first parameter is the seed schedule length for G_a, given as L^a_{seed}. The second parameter is the deadline slack $SLACK_a$ = $L^a_{deadline}$ - L^a_{seed}. Applications with larger slack can better cope with energy cost minimization maneuvers by the CSP. Finally, PAR_a, is calculated as L^a_{seed} subtracted by the length of a schedule for G_a under the assumption of infinite VM resources. Three sorted lists of applications based on these parameters in ascending order will be produced: $L_{seed}[\uparrow]$, $PAR[\uparrow]$ and $SLACK[\uparrow]$.

Phase 3: Preliminary Schedule Generation

In this phase, we generate the PSS by overlaying the seed schedules onto the servers. Fig. 5a is an example of a preliminary schedule. Below is the high level pseudo code.

Procedure P3: Preliminary Schedule Set (PSS) Generation

```
D[↑] = sort_servers(α+β);          /*Sort servers by energy efficiency*/
list_total = { n_1{L_seed[↑]}, n_2{PAR[↑]}, n_3{SLACK[↑]}, n_4{shuffle[]} };
/*shuffle[] contains a shuffled list of applications*/
j = 0;
while (list_total ≠ Ø)
    list = pop(list_total) ;
    clear(pre_schedule(j));
    while (list ≠ Ø)
        a = pop(list);
        for (i = 0; i < |D[↑]|; i++)
            if ($\sum_{\theta\ for\ all\ tasks\ in\ G_a} R^{\theta}_{CPU} + \sum Q^{D[i]}_{CPU}(0) < C^{D[i]}_{CPU}$ &&
                $\sum_{\theta\ for\ all\ tasks\ in\ G_a} R^{\theta}_{MEM} + \sum Q^{D[i]}_{MEM}(0) < C^{D[i]}_{MEM}$)
                if ($\sum_{\theta\ for\ all\ tasks\ in\ G_a} R^{\theta}_{CPU} + \sum Q^{D[i]}_{CPU}(0) \geq C^{D[i]}_{CPU} \cdot Opt$)
                    if (match_probability(p_1))
                        map($G_a \rightarrow D[i]$);
                        break;
    output(prelminary_schedule(j));
    j++;
```

Phase 4: Optimization

The beam search based optimization process is the main body of GMaP. In concept, this phase will transfer suitable tasks from one VM to another, in attempt to satisfy deadlines or maximize energy efficiency. In most cases, these two objectives are contradictory.

GMaP is "guided" in the sense that exploration is steered by empirical observations. GMaP also borrows concepts from evolutionary algorithms to break away from local optima. Each iterative optimization pass goes through two steps: *migrate* and *pack*. The pseudo code is presented below.

Procedure P4: Optimization

```
for (all preliminary schedules in the PSS), do in parallel
    S = load(preliminary_schedule);
    initialize(Explore_Depth);
    if (evaluate(energy) < saved_ref_energy)
        saved_ref_energy = evaluate(energy);
        Explore_Depth = Explore_Depth * Depth_Scale;
    for (i = 0; i < Explore_Depth; i++)
        if (check_deadline_violations() == 1)
            A = pop_dealine_violated_application();
        else
            A = choose_application({G_1, G_2, ..., G_x});
        T = choose_task(A, PAR[↑], SLACK[↑], L_seed[↑]);
        if (match_probability(p_2))
            D = pop(mapped_servers(A));
        else
            D = choose_sever(D[↑], mapped_server_farm(A));
        S = migrate(A: T → D); S = adjust(S);
        evaluate(energy, length);
        if (match_probability(p_3, energy, length));
            accept_move(A: T → D);
        else
            reject_move(A: T → D);
            restore_schedule();
    pack(S);
    evaluate(energy, length);
    if (energy < energy_evaluate(solution_schedule))
        update(solution_schedule)
```

The function migrate(), as the name suggests, will migrate a single task T_i^a from its current type g VM (source VM) in D_x (source server) to another VM of the same type (destination VM) in D_y (destination server). The source and destination servers may not be the same, but in most cases they are within the same server farm to avoid high communication latencies.

Each migration attempt makes three important decisions that will collectively dictate the solution quality, namely:

- Which user should be selected for migration?
- Which task should be selected for migration?
- Which server should the task migrate to?

The CSP should select tasks that bottleneck application latencies, and move them to servers that will not incur high energy overheads, while considering the repercussions to other applications. The decision process will typically cross check $L_{seed}[\uparrow]$, $PAR[\uparrow]$ and $SLACK[\uparrow]$ to select the application that has a negative $SLACK$ value and high L_{seed} and PAR values for migration, because this application has violated the deadline resulting in a negative slack, but since it has a high PAR value, there is great potential for length reduction through parallel execution. The destination server will be selected based on the utilization levels and task dependencies. But the server on which the predecessor successor tasks reside will be selected with a high probability since the current priority is to satisfy the deadline, and communications across servers should be minimized. When the deadlines have been met for most applications, GMaP will be more flexible towards manipulating applications with high $SLACK$ values to minimize energy consumption, such as moving tasks to less crowded servers. After migration, the schedule in the source server may need to be adjusted, as well as the schedule in servers where tasks that are direct or indirect successors to the migrated task reside.

Migration may overprovision VMs, which is why the packing step ensues to compress the schedule. Packing is similar to VM consolidation [26].

VII. EXPERIMENTAL RESULTS

We demonstrate the effectiveness of the GMaP via Monte Carlo simulations for randomly generated large scale task graphs.

Experiments on Large Scale Workloads

In this subsection we demonstrate the effectiveness of GMaP through experiments on large scale cloud platforms with large scale workload inputs. For each experiment, the inputs will be generated differently and the cloud platform scaled accordingly. Table II provides the upper and lower bounds of some key parameters.

Table II. Modeling parameters summary

Cloud Platform Parameters		User Workload Characteristics			Algorithm Search Space	
Servers	Server Farms	Total Users	Tasks per User	Task Latency	Roots in the PSS	Nodes per Tree
10 - 30	2 - 6	30 - 100	20 - 100	1 - 10	50	5000

We first compare the final solution schedule with the "Best Deadline Oblivious Preliminary Schedule" (BDOPS), i.e. the best achievable schedule when regarding workload requests as atomic entities and ignoring all deadlines. BDOPS is ideal for energy efficiency but contain many deadline violations. We purposely draft the SLA so that 30% - 80% of the users in BDOPS have their respective deadlines violated. BDOPS is the reference for energy cost overhead calculations.

Second, we compare the solution with the baseline, which is derived from GMaP minus the energy efficiency optimizations. In other words, the baseline schedules are energy cost oblivious.

Table III. Results for large scale and very large scale workload inputs

Index	Number of Users			Energy Cost Improvement		Allocated VMs in BDOPS	Allocated VMs in Solution
		Total	Rejected	BDOPS	Baseline		
1	Large Scale (30 - 50 Users)	30	0	0.77%	13.04%	35	72
2		30	0	-0.02%	36.76%	34	94
3		30	0	-12.41%	29.35%	33	118
4		30	0	-1.25%	27.49%	32	94
5		35	6	-0.47%	27.70%	30	112
6		35	6	4.34%	28.84%	30	116
7		35	5	-29.89%	33.35%	32	121
8		35	0	-10.17%	14.54%	35	127
9		40	0	-34.02%	27.21%	40	147
10		40	0	-18.93%	20.21%	40	134
11		40	0	-28.04%	17.29%	40	163
12		45	2	-7.98%	18.83%	43	126
13		45	2	-4.73%	17.08%	43	195
14		50	7	-54.85%	18.80%	43	222
Average (Large)				**-14.12%**	**23.61%**	**36.4**	**131.5**
15	Very Large Scale (60 - 100 Users)	60	0	-9.09%	6.70%	60	176
16		65	0	-27.30%	16.34%	65	260
17		75	1	-25.55%	19.44%	74	227
18		75	3	-2.53%	16.66%	72	129
19		75	0	-35.88%	13.36%	75	270
20		85	0	-111.19%	8.67%	85	467
21		85	0	-94.04%	8.14%	85	310
22		85	0	-22.42%	14.52%	85	317
23		85	0	-14.91%	8.88%	85	318
24		95	0	-110.47%	0.12%	95	496
25		100	2	-58.25%	6.64%	98	296
26		100	2	-45.72%	4.24%	98	309
27		100	5	-74.06%	10.08%	95	302
28		100	0	-44.11%	8.50%	100	258
Average (Very Large)				**-49.72%**	**9.35%**	**83.7**	**295.4**

Results are shown in Table III. For large scale inputs (30 - 50 users), energy cost improvement compared to the BDOPS is on average -14.12%, in other words, the energy cost overhead is on average 14.12%. This overhead is inevitable, since it is caused by the additional VM allocations used to accelerate deadline violating applications. The CSP allocates appropriate amount of VMs to each user based on their demands, Fig. 8 illustrates the VM distribution for Experiment 6. Compared to the baseline the energy cost improvement is on average 23.61%, which is a very promising result.

To achieve the same level of solution quality for very large scale inputs (60 - 100 users), the algorithm search space should be expanded to match the increase in input size. However for the sake of consistency, we fixed the size of the search space as indicated in Table II. Consequently, there is considerable increase in energy cost overheads to 49.72% on average. The average energy costs improvements with respect to the baseline decreased to 9.35%. From these statistics we can infer that the search space of 50 roots and 5000 nodes per search tree is sufficient for 30 - 50 user workloads, but lacking for 60 - 100 user workloads. In the next subsection we will expand the search space and rerun the experiments for very large scale workload inputs. Nevertheless, GMaP always terminates with a solution schedule with no deadline violations and positive energy cost improvements over the baseline.

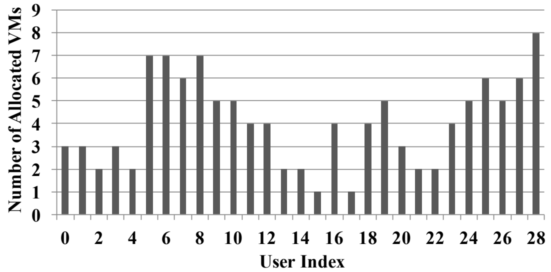
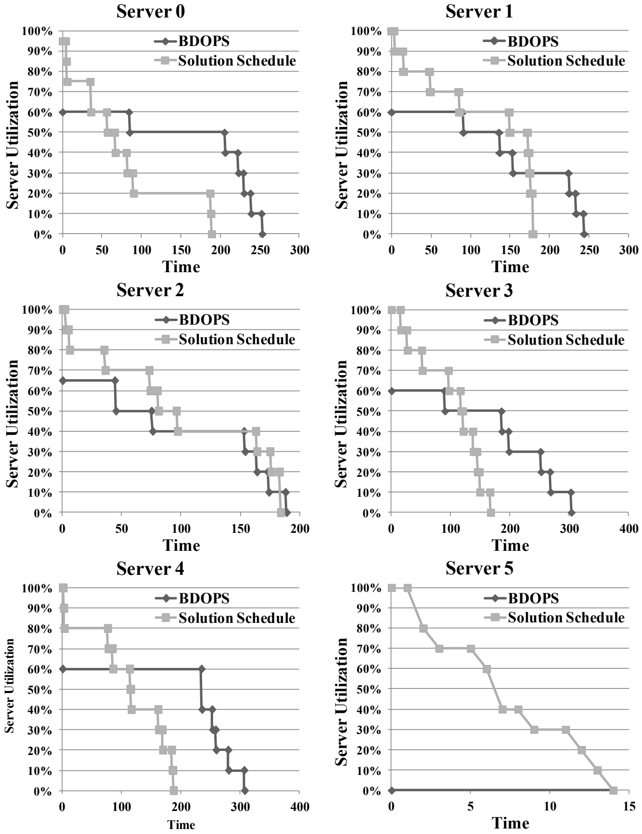


Fig. 8. VM allocations for each user from Experiment 6

Now we show how GMaP balances workloads among servers while allocating additional VMs. Fig. 9 plots the *Util*(*t*) for all ten servers observed during Experiment 6 in Table III. We sorted the servers according to their energy efficiency ratings, with Server 0 being the most energy efficient. We can see fundamental differences in scheduling decisions between the BDOPS and the solution schedule. For the BDOPS, all applications are placed in the most energy efficiency servers, namely Servers 0 - 4, leaving Servers 5 - 9 unused. Although it achieves near optimal energy efficiency, out of the 29 accepted workload requests, 12 violated their deadlines.

For the solution schedule, Servers 0 - 4 saw a utilization percentage increase, which is a result of GMaP allocating additional VMs. Servers 5 - 9 which are less energy-efficient have been brought online to host the new VMs. Their *Util*(*t*) values appears to be high at *t* = 0, but these values only remained high for a short period of time before they dropped.



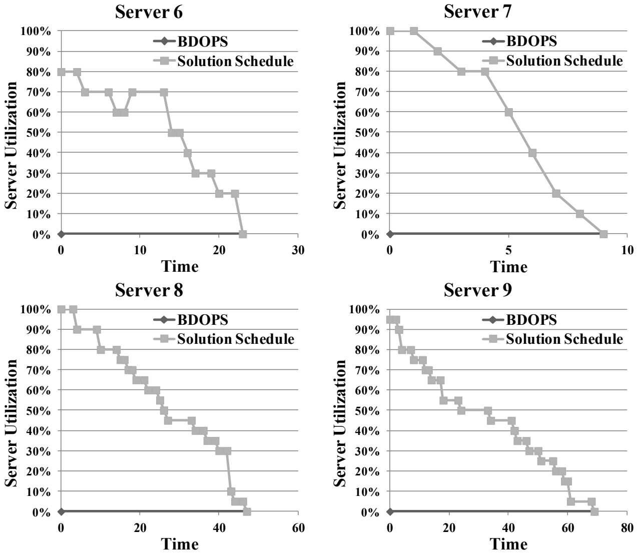


Fig. 9. Server utilization rates for Experiment 6

Impact of Search Space Analysis

Similar to other evolutionary algorithms, for GMaP longer run times will expand the algorithm search space and produce better solutions. But a larger search space needs to be supported by increased computation time, which will lead to increased energy consumption and time of running GMaP. We provide some insights on how to balance the benefits gained from deploying GMaP versus the cost of GMaP itself, by evaluating the improvement in solution quality with respect to expanded algorithm search space.

First we rerun Experiment 15 - 18 from Table III with an expanded search space, namely 50 roots in the PSS and 10^4 nodes per search tree, results are presented in Table IV.

Table IV. Results for expanded search space

Index	Number of Users			Energy Cost Improvement		Allocated VMs in BDOPS	Allocated VMs in Solution
		Total	Rejected	BDOPS	Baseline		
15	Very Large Scale (60 - 100 Users)	60	0	-7.78%	8.15%	60	211
16		65	0	-18.94%	20.23%	65	273
17		75	1	-27.78%	19.92%	74	292
18		75	3	-2.43%	12.14%	72	171
19		75	0	-21.14%	22.69%	75	281
20		85	0	-66.33%	29.05%	85	572
21		85	0	-61.50%	21.24%	85	347
22		85	0	-22.42%	14.52%	85	386
23		85	0	-5.71%	14.24%	85	373
24		95	0	-65.29%	17.20%	95	634
25		100	2	-43.39%	16.90%	98	364
26		100	2	-35.99%	9.75%	98	405
27		100	5	-52.16%	14.78%	95	309
28		100	0	-81.38%	15.06%	100	459
Average (Very Large)				**-36.59%**	**16.85%**	**83.7**	**362.6**

As expected, when the search space is doubled, GMaP produced better results than those presented in Table III, almost doubling the average energy cost improvement from 9.35% in Table III to 16.85% in Table IV.

Next we examine a detailed case study with 20 accepted users on a fixed cloud platform. There will always be 50 roots in the PSS, but we gradually grow the search tree sizes exponentially from 10^2 nodes to 10^5 nodes. Results are shown in Table V.

Table V. Impact of algorithm runtime analysis

Index		Search Tree Size (Number of Nodes)	Energy Cost Improvement		Allocated VMs in BDOPS	Allocated VMs in Solution
			BDOPS	Baseline		
29	a	10^2	-17.77%	-1.38%	20	38
	b		-17.53%	-0.84%	20	35
30	a	10^3	-4.91%	13.20%	20	67
	b		-0.03%	7.46%	20	53
31	a	10^4	12.30%	23.79%	20	91
	b		14.30%	25.35%	20	96
32	a	10^5	16.34%	23.82%	20	93
	b		16.78%	28.94%	20	99

It is clear that as the search tree grows, GMaP terminates with superior solutions. More importantly, GMaP suffers from the common phenomena of diminished returns. The right sizing of the search trees and the PSS is highly dependent on the actual operating context. Currently, GMaP can only offer the flexibility in molding the search space. In the future we intend to develop an adaptive search space sizing procedure.

Impact of Deadline Aggressiveness Analysis

The aggressiveness of deadlines in the SLA has impact on the solution qualities. In this subsection, we investigate a single experiment with a fixed cloud platform of 15 servers in two server farms and a fixed set of 40 workload requests. The deadline of each application is a fraction of the length of its seed schedule: $L^a_{deadline} = \mu \cdot L^a_{seed}\ \forall a$. If $\mu \geq 1$, then the BDOPS would satisfy all deadlines, and immediately becomes the baseline schedule. When $\mu < 1$ GMaP is needed to eliminate deadline violations. Since μ controls the deadline for *all* applications, a small decrease in μ will greatly increase the overall deadline aggressiveness. If μ is too small, then many requests will be dropped. To be consistent in our analysis, in Table VI we only focus on cases with no dropped requests.

Table VI. Impact of deadline aggressiveness analysis

Index		μ (Deadline Aggressiveness)	Energy Cost Improvement		Allocated VMs in BDOPS	Allocated VMs in Solution
			BDOPS	Baseline		
33	a	1.00	20.32%	20.32%	40	54
	b	0.95	-8.94%	20.13%	40	177
	c	0.90	-20.55%	37.82%	40	192
	d	0.85	-20.55%	37.82%	40	192
	e	0.80	-46.50%	38.24%	40	205
	f	0.75	-29.19%	39.27%	40	202
	g	0.70	-53.27%	37.08%	40	192

When $\mu = 1$, we can see that GMaP can devote the entire optimization process to energy cost minimizations, achieving 20.32% improvement over the BDOPS. When μ drops below 1, GMaP becomes constrained by the hard requirement of deadline satisfactions, hence the solution schedules no longer outperform the BDOPS. As the value of μ decreases, the baseline schedule will perform many VM allocations and task migrations that may be detrimental to the energy cost. GMaP successfully recovers near 40% of the loss in energy costs.

VIII. CONCLUSION

In this paper, we consider the problem of global operation optimization in cloud computing from the perspective of the cloud service provider (CSP). Our goal is to provide the CSP with a versatile scheduling and optimization framework that aims to simultaneously maximize energy efficiency and meet all user deadlines, which is also powerful enough to handle multi-user large scale workloads in large scale cloud platforms.

Two types of workload models have been adopted in cloud computing systems: independent batch requests and task graphs with dependencies. In this paper we model the workloads from multiple users as a collection of disjoint task graphs. As for the cloud platform model, it is fully capable of reflecting server resource capacity and energy efficiency heterogeneities. Server communication bottlenecks are also taken into account. This fine-grained treatment of the hardware resources and user workloads provides opportunities for deadline-oriented application acceleration via parallel execution and global energy cost minimization, but also requires additional effort in admission control, resource provisioning, virtual machine placement and task scheduling. In this paper we propose "Guided Migrate and Pack" (GMaP) as a unified scheduling and optimization framework for the CSP that addresses these issues in a holistic fashion. GMaP is also flexible in search space sizing and algorithm run time control. Experimental results show that when GMaP is deployed for the CSP, global energy consumption costs improves by over 23% when servicing 30 - 50 users, and over 16% when servicing 60 - 100 users.

ACKNOWLEGEMENT

This work is supported in part by the Software and Hardware Foundations program of the NSF's Directorate for Computer & Information Science & Engineering.

REFERENCES

[1] B. Hayes, "Cloud Computing," *Communications of the ACM*, 2008.

[2] R. Buyya, "Market-oriented cloud computing: vision, hype, and reality of delivering computing as the 5th utility," *9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid), 2008.

[3] M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, 2010.

[4] M. Pedram, "Energy-efficient datacenters," *IEEE Trans. on CAD*, 2012.

[5] P. Barham et al., "Xen and the art of virtualization," *Proc. of the 19th ACM Symposium on Operating System Principles* (SOSP), 2003.

[6] S. Srikantaiah et al., "Energy aware consolidation for cloud computing," *Cluster Computing*, 12(1):1–15, 2009.

[7] G. Chen et al., "Energy-aware server provisioning and load dispatching for connection-intensive internet services," *NSDI*, 2008.

[8] S. Xavier and S. J. Lovesum, "A survey of various workflow scheduling algorithms in cloud environment," *International Journal of Scientific and Research Publications*, 3(2), 2013.

[9] M. Mazzucco et al., "Maximizing cloud providers revenues via energy aware allocation policies," *IEEE Int'l Conf. on Cloud Computing*, 2010.

[10] M. Lin et al., "Dynamic right-sizing for power-proportional data centers," *INFOCOM*, 2011.

[11] M. Isard et al., "Dryad: distributed data-parallel programs from sequential building blocks," *EuroSys*, 2007.

[12] D. Warneke and O. Kao, "Nephele: efficient parallel data processing in the cloud," *SC-MTAGS*, 2009.

[13] R. Chen et al., "On the efficiency and programmability of large graph processing in the cloud", *Technical Report MSR-TR-2010-44, Microsoft Research*, 2010.

[14] E. Deelman et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, 13(3):219-237, 2005.

[15] L. Ramakrishnan et al., "VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance," *Conf. on High Performance Computing Networking, Storage and Analysis*, 2009.

[16] S. Pandey et al., "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," *Advanced Information Networking and Applications*, 2010.

[17] A. Gorbenko and V. Popov, "Task-resource scheduling problem," *International Journal of Automation and Computing*, 9(4): 429-441, 2012.

[18] R. Lichtenwalter and N. Chawla, "DisNet: A framework for distributed graph computation," *ACM/IEEE Conference on Advances in Social Network Analysis and Mining*, 2011.

[19] M. Redekopp et al., "Performance analysis of vertex centric graph algorithms on the Azure cloud platform," *Workshop on Parallel Algorithms and Software for Analysis of Massive Graphs*, 2011.

[20] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2

[21] D. Tammaro et al., "Dynamic resource allocation in cloud environment under time-variant job requests," *Cloud Computing Technology and Science*, 2010.

[22] J. Bi et al., "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," *IEEE Int'l Conf. on Cloud Computing*, 2010.

[23] R. Aoun et al., "Resource provisioning for enriched services in cloud environment," *Cloud Computing Technology and Science*, 2010.

[24] S. Zaman et al., "An online mechanism for dynamic VM provisioning and allocation in clouds," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[25] M. Bjorkqvist et al., "Opportunistic service provisioning in the cloud," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[26] K. Halder et al., "Risk aware provisioning and resource aggregation based consolidation of virtual machines," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[27] S. Irani et al., "Competitive analysis of dynamic power management strategies for systems with multiple power savings states," *DATE*, 2002.

[28] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," *IEEE Int'l Conf. on Cloud Computing*, 2010.

[29] H. N. Van et al., "Performance and power management for cloud infrastructures," *IEEE Int'l Conf. on Cloud Computing*, 2010.

[30] M. Hadji and D. Zeghlache, "Minimum cost maximum flow algorithm for dynamic resource allocation in clouds," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[31] W. Chen et al., "A profit-aware virtual machine deployment optimization framework for cloud platform providers," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[32] S. He et al., "Improving resource utilisation in the cloud environment using multivariate probabilistic models," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[33] N. M. Calcavecchia et al., "VM placement strategies for cloud scenarios," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[34] L. Rao et al., "Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment," *INFOCOM*, 2010.

[35] N. Buchbinder et al., "Online job-migration for reducing the electricity bill in the cloud," *NETWORKING*, 2011.

[36] H. Xu and B. Li, "A general and practical datacenter selection framework for cloud services," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[37] M. A. Adnan et al., "Energy efficient geographical load balancing via dynamic deferral of workload," *IEEE Int'l Conf. on Cloud Computing*, 2012.

[38] Y.-K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," *Int'l Parallel Processing Symp./Symp. on Parallel and Distributed Processing*, 1998.

[39] H. Topcuoglu et al., "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[40] Y. Gao et al., "Using explicit output comparisons for fault tolerant scheduling (FTS) on modern high-performance processors," *Design Automation & Test in Europe*, 2013.

[41] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011.