Heterogeneity Aware Workload Management in Distributed Sustainable Datacenters

Dazhao Cheng, *Member, IEEE*, Xiaobo Zhou, *Senior Member, IEEE*, Zhijun Ding, *Member, IEEE*, Yu Wang, *Fellow, IEEE*, and Mike Ji

Abstract—The tremendous growth of cloud computing and large-scale data analytics highlight the importance of reducing datacenter power consumption and environmental impact of brown energy. While many Internet service operators have at least partially powered their datacenters by green energy, it is challenging to effectively utilize green energy due to the intermittency of renewable sources, such as solar or wind. We find that the geographical diversity of internet-scale services can be carefully scheduled to improve the efficiency of applying green energy in datacenters. In this paper, we propose a holistic heterogeneity-aware cloud workload management approach, sCloud, that aims to maximize the system goodput in distributed self-sustainable datacenters. sCloud adaptively places the transactional workload to distributed datacenters, allocates the available resource to heterogeneous workloads in each datacenter, and migrates batch jobs across datacenters, while taking into account the green power availability and QoS requirements. We formulate the transactional workload placement as a constrained optimization problem that can be solved by nonlinear programming. Then, we propose a batch job migration algorithm to further improve the system goodput when the green power supply varies widely at different locations. Finally, we extend sCloud by integrating a flexible batch job manager to dynamically control the job execution progress without violating the deadlines. We have implemented sCloud in a university cloud testbed with real-world weather conditions and workload traces. Experimental results demonstrate sCloud can achieve near-to-optimal system performance while being resilient to dynamic power availability. sCloud with the flexible batch job management approach outperforms a heterogeneity-oblivious approach by 37 percent in improving system goodput and 33 percent in reducing QoS violations.

Index Terms—Sustainable datacenter, heterogeneity, job migration, optimization, system goodput, workload placement

1 Introduction

As cloud computing is rapidly growing, services offered by cloud providers such as Amazon, Microsoft, Facebook, and Google are implemented on thousands of servers spread across multiple datacenters. These large datacenters are significant energy consumers due to not only their power-hungry computing equipments but also their cooling and other facilities. Thus, research interests [1], [2], [3], [4] are growing in integrating renewable energy into datacenters to reduce the environmental impact. Researchers envision that in the near future datacenters, at least micro-clouds, can be completely powered by renewable energy and be self-sustainable [3]. However, unlike traditional energy, the availability of green energy (such as solar or wind power generation) may vary widely during the times of a day, seasons of the year, and geographical locations of the power plants. Such intermittency makes it very hard for sustainable datacenters to effectively use green energy.

Meanwhile, major cloud service providers operate multiple geographically distributed datacenters in a wide region for serval reasons, e.g., disaster tolerance and uniform access time for widely distributed clients. On one hand, the power consumption of these datacenters are highly dependent on the resource requirement of dynamic could workloads. One the other hand, the green power generations for distributed sustainable datacenters significantly vary over time and location. In particular, it is difficult to control the system power consumption under a dynamic power supply rather than under a static power budget. Most previous studies assume a static power budget as the constraint for workload management [5], [6]. However, the green power supply in a self-sustainable datacenter highly depends on the natural weather conditions and is often time-varying. It is challenging to effectively match the power supply and demand in a self-sustainable datacenter.

The above characteristic makes the could workload management challenging, especially when the workload is heterogeneous. Most datacenters are commonly shared by many users for quite different uses. They usually support a range of cloud workloads, including critical interactive applications that run 24x7, e.g., Internet services and e-business sites, and batch-style applications, e.g., scientific applications and simulations. For example, most banks have to process the transactional applications for trading stocks and

- D. Cheng and Y. Wang are with the Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223. E-mail: {dazhao.cheng, yu.wang}@uncc.edu.
- X. Zhou is with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918. E-mail: xzhou@uccs.edu.
- Z. Ding is with the Department of Computer Science & Technology, Tongji University, Shanghai 200092, China. E-mail: dingzj@tongji.edu.cn.
- M. Ji is the chief scientist in Valley Technologies Ltd., Saint Louis, MO 63088. E-mail: mji@valleytech.com.

Manuscript received 4 Feb. 2018; revised 15 June 2018; accepted 30 July 2018. Date of publication 17 Aug. 2018; date of current version 16 Jan. 2019. (Corresponding author: Dazhao Cheng.) Recommended for acceptance by F. Cappello. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2018.2865927

the batch workloads for analyzing the model of stock performance at the same time. Consequently, the demand for various workloads has resulted in several composite workload management solutions designed for one or two of these workloads. Fortunately, the geographical distribution of the datacenters not only poses challenges but also opens up opportunities in the cloud workload management.

First, it is challenging to determine the locations where cloud workloads should be placed while their performance requirements are ensured. Datacenters are commonly shared by many users for quite different uses. An important trend is to co-locate heterogeneous workloads, i.e., transactional workloads and batch jobs, on the shared server infrastructure in datacenters for resource utilization efficiency [1]. Transactional requests are relatively flexible to be dispatched to datacenters at different locations for workload management. However, a batch job's processing mostly relies on its data file which is typically not replicated across multiple datacenters. For example, a log analytic job is highly related to its local data and hard to be processed in other datacenters. There is an apparent difference between transactional requests and batch jobs in their data dependence. Most previous studies on distributed datacenter management [5], [7], [8], [9] focus on the transactional workload and pay little attention to batch jobs. For transactional workloads, the goal is to maximize request throughput under a certain response time bound. It requires that resources allocated to transactional workloads be sufficient during the execution of short-lived requests or clients will observe significant decline in service quality. However, batch jobs concern the job completion time in a relatively longer term. It is required that the aggregate resource allocation during the lifetime of batch jobs should guarantee job completions before individual deadlines. Such different characteristics may significantly affect the system performance and lead to serious QoS violations when workloads are heterogeneous.

Second, the dynamic green power supply and geographical distribution of the datacenters also create opportunities for joint performance-power optimizations in distributed self-sustainable datacenters: (1) Although transactional workloads are flexible to be dispatched, their traffic intensities vary widely over time [7]. Matching the computational loads to the actual power supplies of different datacenters could achieve significant power savings. (2) Batch jobs can be migrated from one datacenter to another datacenter where the current power supply is relatively sufficient. (3) Given the dynamic power supply, a careful cloud workload placement and migration planning can maximize the overall system performance.

In this paper, we propose and develop a heterogeneity-aware cloud workload placement and migration approach, sCloud, that aims to maximize the system goodput in distributed self-sustainable datacenters. The key insight of sCloud is that the cloud workload, i.e., transactional requests and batch jobs, can be distributed to different locations based on their time-varying green power availabilities. First, we model the intermittent generation of green energy to predict its production, with respect to the varying weather conditions at the geographical location of each datacenter. Then, we model the performance of transactional workload and batch workload with various resource allocations for each datacenter. Furthermore, we formulate the core objective of sCloud as a constrained optimization problem, in which the constraints capture the QoS requirements, the time-varying workloads and the intermittent availability of green power. Finally, We take advantage of batch job migration to reshape the power demand at different locations when the power supplies of geographically distributed datacenters vary widely.

Specifically, our contributions are as follows. We propose a heterogeneity-aware cloud workload placement and migration approach in distributed self-sustainable datacenters. It maximizes the system performance in terms of system goodput, based on the time-varying green power supply, heterogeneous workload characteristics and QoS requirements. We design an optimization-based algorithm to dynamically place transactional requests to distributed datacenters, with respect to their green power supplies. In order to further improve the system goodput, we integrate another online algorithm to dynamically migrate batch jobs across distributed datacenters when the green power supplies vary widely at different locations.

We extend sCloud design from a heterogeneity aware cloud workload placement approach to a holistic performance aware workload scheduling and resource provisioning system for sustainable datacenters. We propose and develop a flexible batch job manager to dynamically control the job execution progress to maximize the overall system goodput. It employs a novel performance-aware reinforcement learning algorithm to control the job execution progress of batch jobs in each datacenter. It is adaptive to both workload and power changes and works automatically without human interventions.

We have implemented sCloud in our university cloud testbed and performed extensive evaluations with real-world weather conditions and workload traces. Experimental results demonstrate sCloud achieves near-to-optimal system performance while being resilient to dynamic power availability. It outperforms a heterogeneity-oblivious approach GLB [7] by 37 percent in system goodput improvement and 33 percent in reducing QoS violations. The new experimental result shows sCloud with the flexible batch job management approach can further increase the system goodput by additional 11 percent compared to the original sCloud approach.

The rest of this paper is organized as follows. Section 2 describes the design of sCloud. Section 3 presents the optimization-based workload and resource management. Section 4 describes the design of flexible batch job manager. Section 5 gives the testbed implementation. Section 6 presents the experimental results and analysis. Section 7 reviews related work. Section 8 concludes the paper.

2 sCloud Design

sCloud is a holistic workload and resource manager that maximizes the system goodput in the presence of dynamic green power supply. The key insights are transactional workload can be dynamically dispatched to distributed datacenters in order to reshape the power demand of each datacenter, and batch jobs, e.g., MapReduce, can be migrated across datacenters to further improve system performance and green energy usage. As a few batch jobs may be too huge to be migrated, we extend sCloud to temporarily delay these jobs and compensate them later without violating the deadlines. sCloud considers to meet the QoS

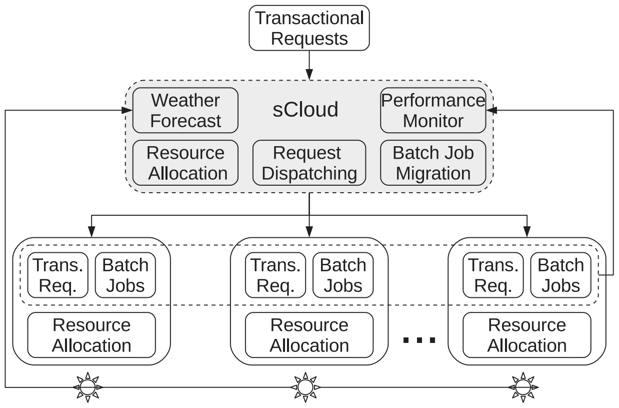


Fig. 1. The architecture of sCloud.

requirements of heterogeneous workloads while making dispatching and migration decisions.

Fig. 1 shows the architecture of sCloud. In each control interval, sCloud performs following three tasks:

- Collects the weather forecast information at each geographical location and predicts its available green power.
- Monitors the performance of applications hosted in each datacenter and the intensity of heterogeneous workloads.
- Decides how to place the transactional requests to geographically distributed datacenters and migrate batch jobs across datacenters.
- Controls the resource allocations between transactional workloads and batch jobs in each datacenter.

We first propose system goodput, a unified metric to quantify the system performance with heterogeneous workloads. We then formulate the workload management and resource provisioning as a constrained optimization problem.

2.1 Quantifying System Performance

Although heterogeneous workloads have individual measures of client-perceived performance, such as request response time and job completion time, a unified metric is needed for cloud providers to quantify the benefit of resource allocation. We define system goodput as the total useful work delivered to users in a certain period of time. Specifically, it is the amount of effective data throughput completed by interactive requests or batch jobs that meet their corresponding service level objectives (SLOs). Similar metrics have been used to quantify system performance for interactive requests [10] and batch jobs [11], respectively.

Formally, we define system goodput $G_i(k)$ at the ith datacenter in time interval k as the sum of two different workloads

$$G_i(k) = G_i^t(k) + G_i^b(k), \tag{1}$$

where $G_i^t(k)$ and $G_i^b(k)$ are the goodput of transactional requests and batch jobs, respectively. For heterogeneous workloads, $G_i(k)$ is uniformly defined as

$$G_i(k) = \frac{(\sum_{l=1}^{m} d_l) * \delta_i(k)}{\Delta t(k)}, \tag{2}$$

where d_l is the data size of task l in a group of m tasks that finish during time period k. $\Delta t(k)$ is the length of interval k. To count only useful work, decay function $\delta_i(k)$ discounts the data throughput from tasks with violated SLOs

$$\delta_i(k) = \begin{cases} 1 & \text{if } t_i(k) < t_{soft}. \\ 1 - \frac{t_i(k) - t_{soft}}{t_{soft}} & \text{if } t_{soft} \le t_i(k) \le t_{hard}. \\ 0 & \text{if } t_i(k) > t_{hard}. \end{cases} \tag{3}$$

We use a SLO with two time bounds, t_{hard} and t_{soft}, in the decay function. While t_{hard} sets a hard deadline for task completion, beyond which no revenue is generated, t_{soft} is a soft deadline whose violation will incur reduction in revenue. We ensure that hard deadlines to be no longer than 2 times of the soft deadlines so that the decay function never becomes negative. Accordingly, $\delta_i(k)$ considers the data delivered by m tasks as useful work if the observed average finish time $t_i(k)$ meets t_{soft}. Violations of t_{soft} and t_{hard} result in a linear decay in the counted throughput and zero work, respectively.

For the transactional workload, we treat requests as individual tasks and estimate the transaction data size $d^t(l)$ as the size of the response sent back to the clients. For batch jobs, we use the input data size to approximate the task size $d^b(l)$. Typically, these jobs are split and processed by many sub tasks distributed across a number of computing slots in a cluster. If there are more tasks than the available slots, the tasks will be processed by multiple waves [12]. Then the SLO requirement is divided into the sub-task level as following:

$$n_{wave} = \left\lceil \frac{n_{task}}{n_{slot}} \right\rceil, \tag{4}$$

$$t_{task}^{SLO} = \frac{t_{job}^{SLO}}{n_{wave}}, \tag{5}$$

where n_{task}, n_{slot} and n_{wave} are the number of tasks, slots and waves for the batch job, respectively. The SLO requirement at the job level t_{job}^{SLO} is divided into task level t_{task}^{SLO}. Many batch jobs provide the interface to query such execution information, such as the *JobTracker* in a Hadoop environment.

2.2 Optimization Problem Formulation

The optimization problem of sCloud is formulated as,

$$\max \sum_{k=1}^{K} \sum_{i=1}^{N} [G_i^t(k) + G_i^b(k)], \tag{6}$$

$$\text{s.t. } P_i(k) = P_i^{solar}(k) + P_i^{wind}(k), \tag{7}$$

$$\lambda^t(k) = \sum_{i=1}^{N} \lambda_i^t(k), \forall i \in N, k \in K, \tag{8}$$

$$\lambda_i^t(k) \ge 0, \lambda_i^b(k) \ge 0, \forall i \in N, k \in K. \tag{9}$$

Objective Eq. (6) aims to maximize the system goodput. Constraint Eq. (7) represents that the power supply of the ith datacenter is determined by the local green power generation amount. Eq. (8) represents that the overall transactional workload arrival rate is the accumulated value of that at each datacenter. This is a complex optimization problem with both nonlinear objective function and constraints. To solve the problem, we first build the green energy model for power supply prediction. We then model the performance of heterogeneous workloads in each datacenter.

TABLE 1
Notations

Symbol	Meaning
N	Total number of datacenters
k	Control interval
P_i	The total power supply at the ith datacenter
P_i^{solar}	The solar power supply at the ith datacenter
P_i^{wind}	The wind power supply at the ith datacenter
$\lambda^t(k)$	The system transactional workload arrival rate
$\lambda_i^t(k)$	Trans. workload arrival rate at the ith datacenter
$\lambda_i^b(k)$	Batch workload arrival rate at the ith datacenter

Finally the optimization problem is transformed to a nonlinear programming problem. The notations used for problem formulation are listed in Table 1.

3 Optimization-Based Management

3.1 Green Power Prediction

In contrast to brown energy, solar and wind energy generation may not provide a reliable, consistent source of energy due to the time-varying weather conditions. We use prediction methods to estimate the amount of green energy in a given interval and utilize that data to make decisions of workload placement and resource provisioning in distributed datacenters.

We use the model introduced by the work in [13] to predict solar and wind power. The solar model is based on the simple premise that energy generation is inversely related to the amount of cloud coverage. It is expressed as: $P^{solar}(k) = B^{solar}(k)(1 - CloudCover)$, where $P^{solar}(k)$ is the amount of solar power predicted for interval k, $B^{solar}(k)$ is the amount of solar power expected under ideal sunny conditions, and $CloudCover$ is the forecasted percentage cloud cover (given as a fraction between 0 and 1). We use historical data from NREL [14] to instantiate B^{solar}.

The wind power model is based on the cubic wind power production function [13]. That is, $P^{wind}(k) = a \times (v^w(k))^3 + b$, where v^w is the windspeed, a and b are parameters depending on the locations. We fit this power curve with the observed data from NREL [14] using the least-squares method to generate the wind power model. The parameters a and b for three locations used in our implementation are obtained as shown in Table 2.

Given the data from NREL [14], we implement solar and wind energy prediction models at the granularity of 10 minutes. We assume that each sustainable datacenter has 7 solar panels and a micro-turbine with capability of producing 1 KW respectively. Fig. 3 demonstrates that the average difference between each observed and predicted value is small.

In this work, we focus on the CPU resource allocation since CPU is the major power contributor in datacenters [1], [2], [3]. There is a linear relationship between the available CPU resource and the power consumption [1], [5]. Thus, the power constraint in the optimization problem is transformed to the resource availability constraint.

3.2 Transactional Workload Placement

As in related studies [8], [15], we use the classic queueing theory to model the performance of heterogeneous workloads in each datacenter. sCloud includes transactional and batch workloads in the model to obtain a more realistic system view in the cloud environment.

TABLE 2
Wind Power Model Parameters

Locations	California	Hong Kong	Dublin
a	0.1732	0.1697	0.1783
b	3.1207	3.0641	2.9372

3.2.1 Transactional Workload Model

For the transactional workload, performance goals are typically defined in terms of the average response time [16] or throughput [10]. We use a multi-server queueing model M/M/c [17] to model the performance of transactional workload in a datacenter. Previous study [8] has shown M/M/c model is accurate to estimate the performance of transactional requests in a cluster environment. The average response time of the requests at the ith datacenter is represented as

$$t_i^t = \frac{P_Q}{c\mu_i^t - \lambda_i^t} + \frac{1}{\mu_i^t}. \tag{10}$$

Here, t_i^t includes the waiting time $\frac{P_Q}{c\mu_i^t - \lambda_i^t}$ and the service time $\frac{1}{\mu_i^t}$. c is the number of servers and μ_i^t is the average service rate of a single server in the ith datacenter. λ_i^t is the request arrival rate in the ith datacenter. P_Q is the probability of requests waiting in the queue. Similar to others [8], [18], we assume that all servers will keep busy. Hence, we have P_Q equal 1.

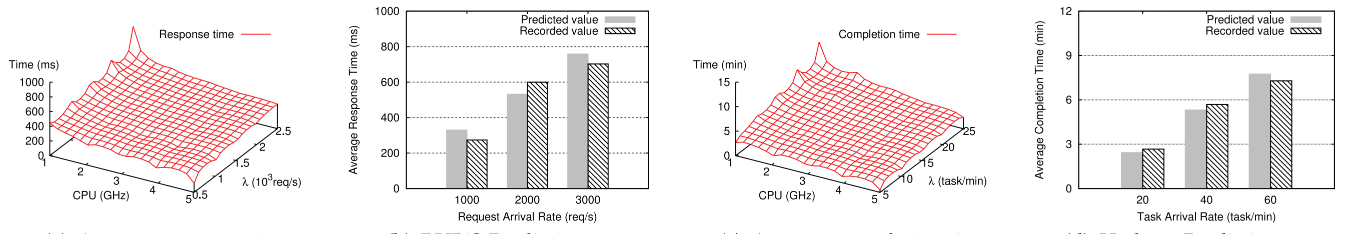
3.2.2 Batch Workload Model

The performance of batch workload concerns the completion time of individual jobs. For the batch workload, schedulers, such as Fair Scheduler and Capacity Scheduler in Hadoop can share the resource among multiple jobs. We consider there are multiple jobs hosted in the cluster. As demonstrated in [15], [19], job arrivals can be modeled by a Poisson process, where the inter arrival times are typically exponentially distributed. Thus, we model the batch workload at the task level by a M/GI/1/PS queueing system [3]. Multiple jobs are hosted in a Hadoop cluster. Each job is divided to multiple tasks for processing. Let t_i^b denote the average completion time of the batch tasks. Based on the queueing foundation, we have

$$t_i^b = \frac{1}{\mu_i^b - \lambda_i^b}. \tag{11}$$

Here λ_i^b is the workload arrival rate and μ_i^b is the average service rate that is determined by its resource allocation.

In order to evaluate the accuracy of the models, we implement RUBiS as the transactional workload and Gridmix2 of Hadoop as the batch workload. As shown in Figs. 2a and 2c, the average response time and the completion time vary when given different amount of resource and workload. The accuracy is measured by the normalized root mean square error (NRMSE), a standard metric for deviation. Figs. 2b and 2d show that the real measured data and predicted data



(a) Average response time. (b) RUBiS Prediction error. (c) Average completion time. (d) Hadoop Prediction error.

Fig. 2. Performance model accuracy evaluation with different amount of resources and workloads.

are very close, with the NRMSE 10.4 and 6.9 percent for RUBiS and Hadoop workloads, respectively.

3.2.3 Placement Algorithm

Given the models above, the optimization problem described in Section 2 is transformed to a nonlinear programming problem. Let $V(k) = \{\lambda_i^t(k), r_i^t(k), r_i^b(k)\}$ denote the optimization variable. As shown in Algorithm 1, at the beginning of each control interval, sCloud takes current weather forecast, workload arrival rate and the service rate as inputs. It generates the optimal transactional workload placement solution and resource allocations at each datacenter in the current interval to maximize the system goodput.

Algorithm 1. The Algorithm of Placement

```
1:  Initial state $v(k)$, $k \leftarrow 1$;
2:  repeat
3:      At the beginning of control interval $k$;
4:      Inputs:
5:      / Weather forecast: $CloudCover_i(k)$, $v_i^w(k)$;/
6:      / Workloads: $\lambda_i^b(k)$, $\lambda^t(k)$;/
7:      / Service rate: $\mu_i^t(k-1)$, $\mu_i^b(k-1)$;/
8:      Update the performance model parameters;
9:      Predict the available green power $P_i(k)$;
10:     Solve the problem by $fmincon$ function in MATLAB;
11:     Output: The optimal point $V^*(k) = \{\lambda_i^t(k), r_i^t(k), r_i^b(k)\}$;
12: until $k = K$
```

We use the *Karush-Kuhn-Tucker* conditions [20] to determine the optimal point $V^*(k)$ for the placement solution $\lambda_i^t(k)$ and resource allocations $r_i^t(k), r_i^b(k)$. We implement the proposed algorithm based on a standard nonlinear programming solver, *fmincon*, which is provided in the optimization toolbox of MATLAB.

3.3 Batch Job Migration

3.3.1 Why Migrate

Although the transactional workload placement can effectively distribute the workload to geographically distributed datacenters, job migration provides another opportunity to further reallocate the workload according to the power availability at individual datacenters. When the power supply is too low for one datacenter, as Algorithm 1 shows, sCloud does not dispatch any transactional requests to the datacenter. Moreover, sCloud may further migrate jobs from this datacenter to other datacenters to meet the QoS requirement and to maximize the system goodput.

3.3.2 Where to Migrate

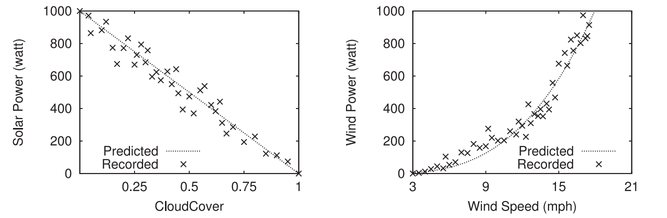
Algorithm 2 determines which job should be migrated and where to migrate. If no new transactional request is placed to a datacenter in the kth interval, the algorithm is initialized when a new batch job is submitted to the datacenter i. If job migration is needed, the batch job is migrated to another datacenter before it starts running at the source datacenter. Fig. 4 illustrates a scenario where a batch job j is submitted to the datacenter but there is no sufficient power at the datacenter to run the job in addition to the currently running jobs and meet their QoS requirements. sCloud performs two steps to decide where to migrate job j.

First, sCloud selects a destination candidate x from the potential destination datacenters. It is done by comparing the completion time of the job j among the possible migration solutions if j would be migrated from the current datacenter to others. Note that batch jobs typically rely on their data files and the job migration incurs additional cost for data transfer.

Specifically, the estimated completion time of the migrated job j includes the data transfer delay and the job process time at the xth datacenter. The delay caused by data transfer between the source datacenter i and the destination datacenter x is represented as

$$t_{i,x}^d(j) = \frac{DataSize_j}{Bandwidth_{i,x}}, (i, x \in \{1, \ldots, N\}). \qquad (12)$$

Here, delay $t_{i,x}^d$ is determined by the bandwidth between two datacenters and the amount of data that needs to be



(a) Solar power model prediction. (b) Wind power model prediction.

Fig. 3. The accuracy of green power prediction.

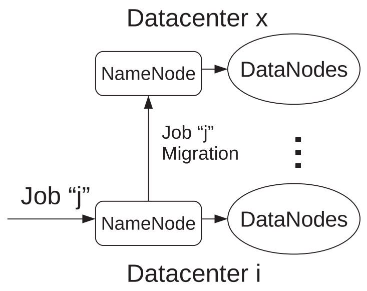
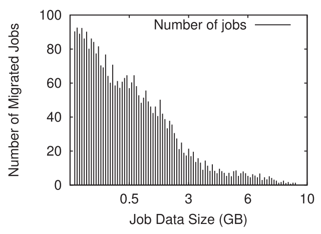
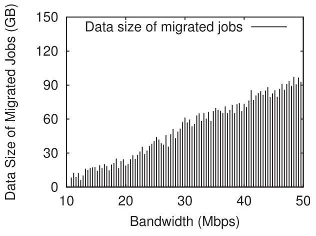


Fig. 4. Batch job migration.



(a) Data size impact.



(b) Bandwidth impact.

Fig. 5. Data size and bandwidth impact on job migration.

transferred. The amount of data is determined by specific job input and output data size. We focus on the batch jobs with large scale input data. Fig. 5 depicts the statistical results of batch job migration in our experiment using FaceD workload [2], [19]. Fig. 5a shows that the migration of the small jobs is more frequent than that of the large jobs. Fig. 5b demonstrates that the data size of migrated jobs is proportionally increased when the bandwidth increases.

The job process time at the xth datacenter is estimated as

$$t_x^{process}(j) = t_x^b \times n_{wave}, \tag{13}$$

where t_x^b is the average task completion time in the xth datacenter based on the performance model described by Eq. (11). n_{wave} is the number of waves for job j, which can be obtained from the *JobTracker* in a Hadoop environment.

Second, the algorithm makes decision of job migration based on the performance comparison between the destination candidate x and the current datacenter i. If the completion time at the current datacenter is longer than that at the destination candidate, job j is migrated to the destination candidate. Otherwise, job j is still hosted in the current datacenter.

Algorithm 2 is implemented as a daemon program at each datacenter, i.e., $NameNode$ of Hadoop cluster.

Algorithm 2. The Algorithm of Job Migration

```
1: Search from the first datacenter, $i \leftarrow 1$;
2: repeat
3:   if $\lambda_i^t(k) = 0$, Job $j$ submits then
4:     /* Find the best destination candidate $x$ */
5:     $x = \arg\min_x [t_{i,x}^d(j) + t_x^{process}(j)], x \in \{1, \ldots, N\}$
6:     /* Make decision of job migration */
7:     if $t_i^{process}(j) > [t_{i,x}^d(j) + t_x^{process}(j)]$ then
8:       Job $j$ is migrated from $i$ to $x$
9:     end if
10:  end if
11: until $i = N$
```

3.3.3 How to Migrate

sCloud uses the *DistCp* in a Hadoop environment to transfer the job data from the source datacenter to the destination datacenter. *DistCp* is a tool used for large inter-cluster data copying. It is implemented as a MapReduce job where the work of copying is done by the maps that run in parallel across the cluster. It tries to give each map approximately the same amount of data, at least 256 MB. Note that the *DistCp* expects the data transfer between the Hadoop clusters to be of the same version.

TABLE 3
Analysis of System Goodput Lost

Workloads	Goodput Lost	% Jobs	Average Size
Trans.	5.3 GB	6.2%	47.2 KB
Batch	76.5 GB	0.4%	4.5 GB

4 Flexible Batch Job Management

To identify the performance impact from different workload types, we analyze a representative workload set from a Fackbook datacenter [2] as shown in Table 3. It shows the batch jobs contribute majority of the system goodput lost since their average data size is much larger than transactional workloads' data size. As these batch jobs may be too huge to be migrated, it is necessary to further explore more flexible approaches to improve system performance. Thus, we extend sCloud to temporarily accelerate or delay these jobs, and compensate them later without violating their deadlines. We propose and develop a flexible batch job manager to dynamically control the job execution progress to maximize the overall system goodput. It takes advantages of time-varying traffic of transactional workloads and delay-tolerance of batch jobs to optimize the overall system performance under dynamic power constraints. It employs a novel performance-aware reinforcement learning algorithm to control the job execution progress of batch jobs in each datacenter. The algorithm prioritizes transactional workloads when possible but also aims to avoid batch job deadline miss. Based on the predictions of the performance model, it performs efficient search in the space of all possible allocation combinations. It is adaptive to both workload and power changes and works automatically without human interventions.

For transactional workloads, the goal is to maximize request throughput under a certain response time bound. It requires that resources allocated to transactional workloads be sufficient during the execution of short-lived requests or clients will observe significant decline in service quality. Batch jobs concern the job completion time in a relatively longer term. It is required that the aggregate resource allocation during the lifetime of batch jobs should guarantee job completions before individual deadlines. We formulate the resource provisioning problem of batch job execution as a reinforcement learning process. We define the state space S and action set A when applying the reinforcement learning approach. We use reinforcement learning optimization approach to obtain an adaptive job progress tuning scheme for large jobs. Based on practical issues, in the experiments sCloud focuses on allocating CPU resources, which are commonly believed to be the major power consumer in datacenters.

4.1 Problem Formulation and Searching Space

Reinforcement Learning (RL) is a process of learning through interactions with an external environment so as to maximize long term rewards defined on a high level goal. The job progress tuning problem of resource allocation can be formulated as a finite `Markov Decision Process` (MDP). Formally, for a set of states S and a set of actions A, the MDP is defined by the transition probability $P_a(s, s')$ and a reward function R. At each step t, the adapter

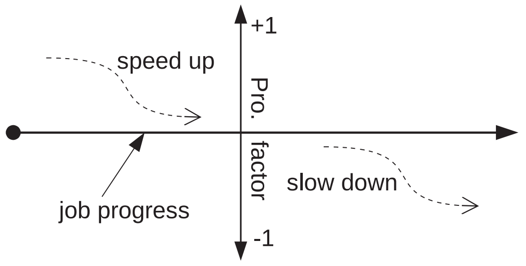


Fig. 6. Flexible batch job progress management.

perceives its current state $s_t \in S$ and the available action set $A(s_t)$. By taking action $a_t \in A(s_t)$, the adapter transits to the next state s_{t+1} and receives a reward R_{t+1} from the environment. The value function of taking action a in state s can be defined as: $Q(s,a) = E\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a\}$, where $0 \leq \gamma < 1$ is a discount factor.

We define the state space S as the set of possible parameter values. States defined on the configurations are deterministic in that $P_a(s,s) = 1$, which simplifies the RL problem. We represent the state space as a collection of state vectors: $s = [\epsilon_i], i \in [1, N]$. The elements in the state vector are the job progress control factors for batch jobs at different datacenters. As shown in Fig. 6, ϵ_i is the progress factor to determine how long the batch job can be delayed or accelerated compared to the job execution progress achieved in the previous control interval at the ith datacenter. Formally, we define $\epsilon_i = Pro_i(k+1)/Pro_i(k)$, where $Pro_i(k+1)$ and $Pro_i(k)$ are the batch job progress in the $k+1$th and kth interval, respectively. We shrink the searching space of these ratios to a reasonable range (i.e., $\epsilon_i \in [-1, +1]$) in order to accelerate the search speed. The tuning granularity of ϵ_i is empirically set to 0.05 in the experiment. The action set (A) is represented as a collection of action vectors (a): $A = [\epsilon_i]$. Only one datacenter is considered at a time and only one-step tuning is allowed. It follows the natural trail-and-error method that searches the configuration state space exhaustively. More importantly, resource adjustment in small steps smooths the tuning process.

4.2 Performance-Aware Reward Function

In the resource sharing configuration problem, the desired configurations are the ones which improve system-wide performance (i.e., goodput for both transactional workloads and batch jobs). The rewards are the summarized performance of individual datacenter feedbacks on the resulted new configuration. The performance is measured by a reward which is the ratio of current goodhput (G_i^b) to a reference goodput plus possible penalties when task level SLOs are violated

$$\begin{aligned} reward &= \frac{G_i^b}{G_i^b(ref)} - penalty, \\ penalty &= \begin{cases} 0 & \text{if } t_{task} \leq SLO, \\ \frac{t_{task}}{t^{SLO}} & \text{Otherwise.} \end{cases} \end{aligned} \tag{14}$$

The reference goodput ($G_i^b(ref)$) value is the maximum achievable batch job processing rate without any task level SLO violations. We obtained the reference for one batch job by giving sufficient resource to guarantee the job execution progress. The task level SLO equals the soft time bound proposed in Section 2 so that the overall job level execution progress is ensured. This constraint guarantees that the job delay latency is not continuously increasing and the system that can remain stable. A low reward indicates this job execution may miss the soft deadline or the hard deadline, both of which should be avoided when tuning resource allocations.

4.3 Solution of Search Algorithm

The solution of the RL problem aims to maximize the cumulative rewards at each state. It is equivalent to finding an estimation of $Q(s,a)$ which approximates its actual value. The experience-based solution is based on the theory that the average of the sample $Q(s,a)$ values collected approximates the actual value of $Q(s,a)$ given sufficiently large number of samples. A sample is in the form of (s_t, a_t, R_{t+1}). The basic RL algorithms in experience based solution are called temporal-difference methods, which update $Q(s,a)$ at each time a sample is collected

$$\begin{aligned} Q(s_t, a_t) = &\, Q(s_t, a_t) \\ &+ \alpha * [R_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \end{aligned} \tag{15}$$

where α is the learning rate and γ is the discount factor. The Q values are usually stored in a look-up table and updated by writing new values to the corresponding entries in the table. Starting from any initial policy, the adapter gradually refines the policy based on the feedback perceived at each step.

5 System Implementation

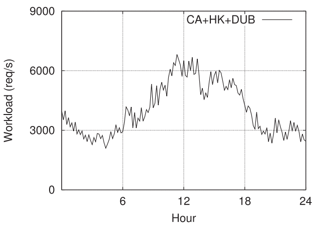
5.1 Testbed

We built a testbed in a university prototype datacenter, which consists of 276-core CPUs, 2.1 TB memory and 102 TB disk storage. VMware vSphere 5.5 was used for server virtualization. VMware vSphere module controls the CPU usage limits in MHz allocated to the virtual machines (VMs). It also provides an API to support the remote management of VMs. The power monitor gathers the real-time measured power consumption of individual VMs through VMware ESXi 5.5 Intelligent Power Management Interface Sensors [21].

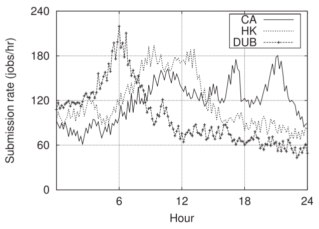
Large corporations built their distributed datacenters at multiple regions worldwide, e.g., North America, Asia and Europe. We built three clusters based on the testbed to mimic three self-sustainable datacenters located at California (CA), Hong Kong (HK) and Dublin (DUB), respectively. The average bandwidth between the clusters is 27 Mbps by measurement. Each cluster includes a number of transactional Web servers and a Hadoop cluster. The Hadoop used in the experiment is 1.0.3 version and configured with 11 VMs, i.e., 1 $MasterNode$ and 10 $SlaveNode$. Fair Scheduler is used in the experiment. As in the work [12], each Slave Node is configured with a single map and reduce slot. Each VM is allocated 1 VCPU and 1 GB memory. The block size is configured 64 MB in our experiments. All VMs use Ubuntu Server 10.04 with Linux 2.6.32.

5.2 Real-World Workloads

For the transactional workload, we use open-source RUBiS as the benchmark and a real Internet trace from Wikipedia.org [22] to mimic the daily dynamics of workload volume.



(a) Transactional workload.



(b) Batch job submission rate.

Fig. 7. The workloads traces at CA, HK and DUB.

The trace represents the users' behavior in visiting the Wikipedia website. Fig. 7a shows the transactional workload used in the experiments. The number of concurrent users dynamically changes from 2,700 to 6,300 in 24 hours. Our experiments set the soft response time bound to be 1,000 ms and the hard response time bound to be 1,500 ms [16].

For the batch workload, we use a synthetic workload, "Facebook-Derived (FaceD)", which models Facebook's production workload [2], [19]. FaceD contains jobs with widely varying execution times and data set sizes, representing a scenario where the cluster is used to run many different types of batch applications. By default Hadoop configuration, we submit all jobs with normal priority in the experiments.

Table 4 shows the FaceD workload characteristics and the SLO soft and hard completion time bounds. We do not run the Facebook code itself. Rather, we mimic the characteristics of the jobs using "loadgen". Loadgen is a configurable MapReduce job from the Gridmix2 benchmark in the Hadoop distribution. Fig. 7b shows a real-world trace from Facebook [23] to mimic the submission rate of jobs.

5.3 Green Power Supply and Weather Conditions

We use the green power data from the National Renewable Energy Laboratory (NREL) database [14]. This database contains time series data in 10-minute intervals from more than 30,000 measurement points worldwide. In experiments we picked three measurement points at CA, HK and DUB to get real green power traces. Fig. 8a shows the amount of green power supply varies over time during one day at CA, HK and DUB. The time used in the figures is Pacific Time.

To emulate the intermittent availability of green energy, we use meteorological data from the Measurement and Instrumentation Data Center of NREL [14]. As shown in Fig. 8, a variety of meteorological data, including sun irradiance, cloud cover, and wind speed, is covered in those records from the NREL. Prior studies [1], [8] have shown that the data from the NREL is quite accurate in weather prediction.

TABLE 4
FaceD Workload Characteristics

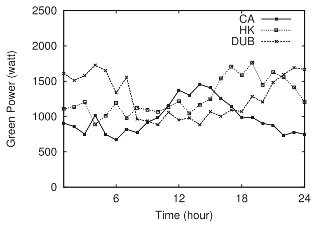
% Jobs	# Maps	# Reds	Data (GB)	SLO (min) (soft-hard)
59	4	1	0.36	1-1.5
9.8	10	1-2	0.69	2-3
8.7	20	1-5	1.65	5-7.5
8.5	40	2-10	2.60	10-15
5.7	80	4-20	5.00	15-22.5
4.4	150	8-38	9.38	30-45
2.5	300	15-75	18.75	50-75
1.3	600	30-150	37.5	100-150

5.4 SCLOUD Components

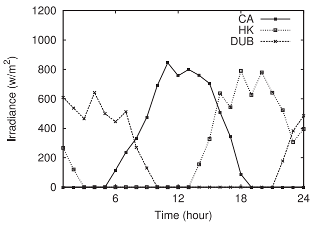
1) sCloud Controller: We implement sCloud control algorithm on a separate VM, which issues commands to the virtualized server cluster using VMware vSphere API 5.0. The algorithm invokes a nonlinear programming solver *fmincon* in the optimization toolbox of MATLAB.
2) Power Monitor: The real-time power consumption of the virtualized cluster is measured at the resource pool level. The power monitor gathers the measurement data through VMware ESX 5.0 Intelligent Power Management Interface sensors.
3) Performance Monitor: For the transactional workload, it uses a sensor program provided by RUBiS client for performance monitoring in terms of request response time and the data size of each request. For MapReduce batch jobs, it measures each task completion time and task size by *JobTracker* on the *NameNode* periodically.
4) Resource Allocator: It uses vSphere API to impose CPU usage limits on the VMs. The vSphere module provides an interface to execute a method *ReconfigVM* to modify a VM's CPU usage limit, ranging from 0 to 2.8 GHz.
5) Flexible Batch Job Manager: The online RL adapter was deployed with a table-based Q function which was initialized to all zeros. We used the Sarsa(0) algorithm [24] to drive the flexible batch job adapter.

6 Performance Evaluation

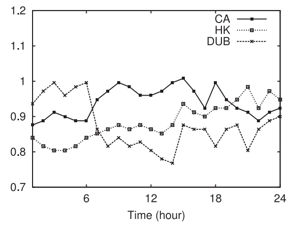
We first demonstrate the system goodput achieved by sCloud and illustrate the effectiveness of self-optimizing workload placement with dynamic green power supply. We then demonstrate the effectiveness of the flexible batch job manager in terms of system goodput and time bound violation. Furthermore, we show the self-adaptiveness of



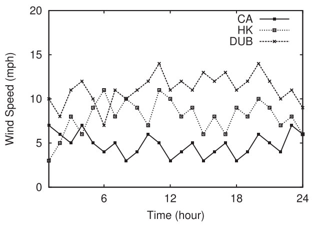
(a) Green power supply.



(b) Sun irradiance.



(c) Could cover.



(d) Wind speed.

Fig. 8. The local green power supply, sun irradiance, cloud cover and wind speed at California (CA), Hong Kong (HK) and Dublin (DUB).

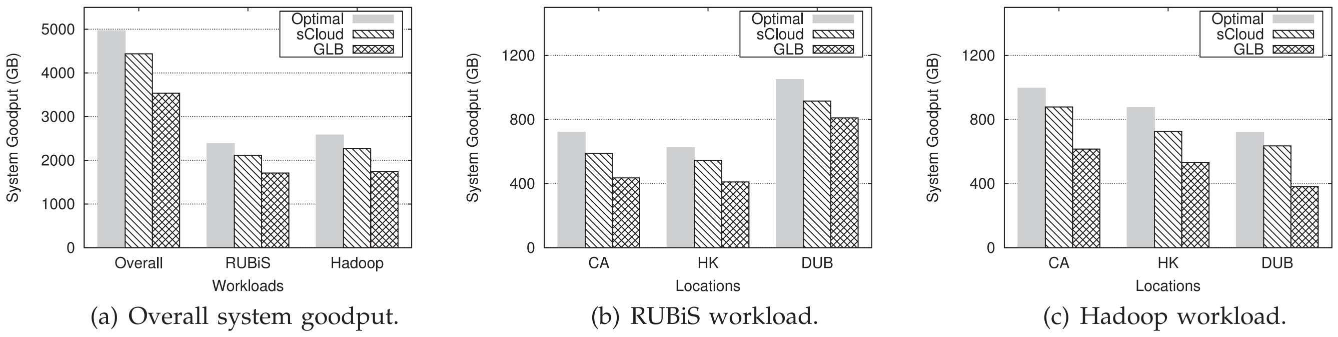


Fig. 9. The goodput improvements in a one-day scenario at California (CA), Hong Kong (HK) and Dublin (DUB).

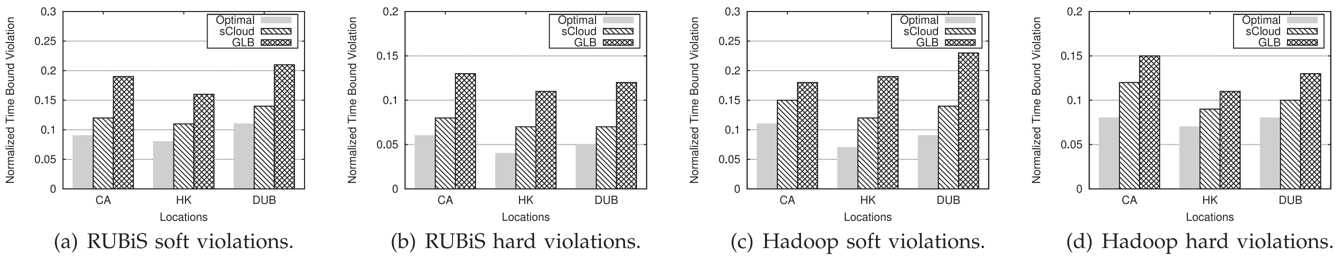


Fig. 10. Soft and hard QoS violations of RUBiS and Hadoop at California (CA), Hong Kong (HK) and Dublin (DUB).

power consumption control and resource allocation by sCloud. Finally, we show the effectiveness of batch job migration control and the power prediction impact on sCloud.

For reference, we obtained the optimal system goodput based on an offline optimization process using the workload trace and power supply trace. Note this offline training based optimization is an ideal but not practical solution. For illustration, we implemented GLB [7], an online migration-oblivious geographical load balancing approach for cloud workload. For fairness, we tailored GLB so that its available resource for provisioning is also dynamically driven by the power supply.

6.1 System Goodput Improvement

Fig. 9 compares the system goodput achieved by Optimal, sCloud and GLB approaches in a one-day green power supply scenario. Fig. 9a shows that the overall system goodput of sCloud is 26 percent more than that achieved by GLB. And sCloud achieves near-to-optimal performance, obtaining 91 percent system goodput of the optimal solution. For the goodput of RUBiS and Hadoop workloads, sCloud outperforms GLB by 23 and 30 percent while achieving 92 and 89 percent of Optimal, respectively.

Figs. 9b and 9c show the performance comparison of three different approaches for RUBiS and Hadoop workloads at different locations. Fig. 9b shows that the goodput of RUBiS workload by sCloud outperforms that by GLB 24, 27 and 17 percent at CA, HK and DUB, respectively. And it achieves 92, 94 and 90 percent of the optimal solution at CA, HK and DUB, respectively. Fig. 9c shows that the goodput of Hadoop workload by sCloud outperforms that by GLB 24, 27 and 17 percent at CA, HK and DUB, respectively. And it achieves 92, 94 and 90 percent of the optimal solution at CA, HK and DUB, respectively.

Fig. 10 compares the QoS time bound violations of RUBiS workload and Hadoop workload by Optimal, sCloud and GLB approaches in the one-day scenario. Figs. 10a and 10b show that sCloud significantly reduces the soft response time bound violation and hard response time bound violation of RUBiS workload by 39 and 43 percent respectively. Figs. 10c and 10d show that sCloud also effectively reduces the soft completion time bound violation and hard completion time bound violation of Hadoop workload by 35 and 29 percent respectively.

6.2 Self-Optimizing Placement

Fig. 11 provides a microscopic view of the dynamic RUBiS workload placement by different approaches in the one-day scenario. It depicts how to place RUBiS requests to CA, HK and DUB by Optimal, sCloud and GLB. In the first eight-hour stage, GLB dispatches more requests to DUB due to more green power supply at DUB. But when the batch workload at DUB is relative high between the 5th and 7th hours, sCloud and Optimal allow more requests to be

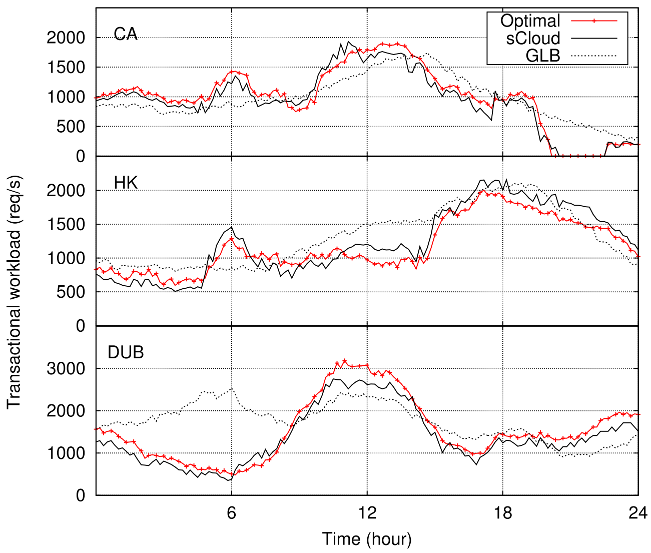
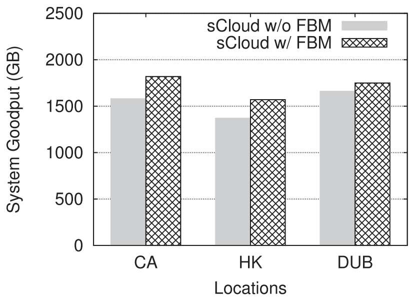
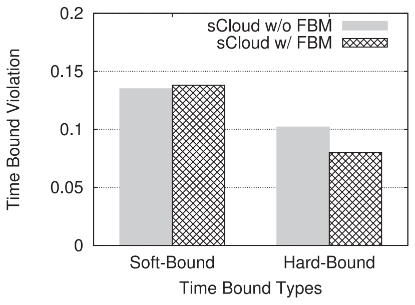


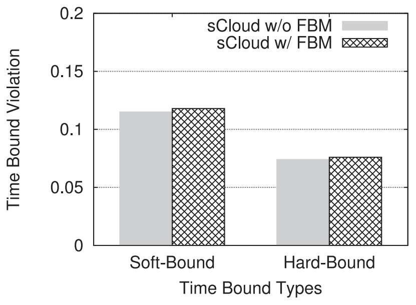
Fig. 11. RUBiS workload placement.



(a) System goodput comparison.



(b) Hadoop time bound violations.



(c) RUBiS time bound violations.

Fig. 12. Effectiveness of flexible batch job manager in terms of system goodput and time bound violation.

placed to CA and HK to improve the overall system goodput. This is due to the fact that sCloud and Optimal take both transactional and batch workloads into account.

In the second stage, the green power supplies of three locations are similar with each other. sCloud and Optimal place more RUBiS workload to DUB than to others since it has the smallest batch workload. Because the batch workload at HK is highest during this period, sCloud and Optimal place smaller RUBiS workload to HK than to GLB.

In the final stage, sCloud and Optimal place more RUBiS workload to HK and DUB while GLB places the requests to three datacenters in proportion to their power supplies. This is because the green power supplies at HK and DUB are more than that at CA. During the 20th to 22th hours, sCloud and Optimal place all transactional requests to HK and DUB to improve the system goodput. GLB still places RUBiS requests to three datacenters in proportion to their power supplies, losing the opportunity for the system goodput optimization.

6.3 Flexible Batch Job Management

Fig. 12 demonstrates the effectiveness of the flexible batch job manager in terms of system goodput and time bound violation. Fig. 12a shows that sCloud with the flexible batch job manager increases the system goodput by 11 percent compared to sCloud without the flexible batch job manager. CA and HK achieve higher performance improvement than DUB since the power and resource availability at CA and HK fluctuates wildly compared to DUB. It provides more opportunities to manage the batch job execution progress in a flexible manner. For example, a batch job at CA may be slow down at the 10th hour and then be speed up at the 11th hour without violating its deadline. This is due to the fact that the power supply at the 11th hour is significantly increased and the job execution may still catch the deadline after a small delay.

Figs. 12b and 12c compare the soft and hard time bound violations for Hadoop and RUBiS workloads respectively. It shows that the proposed flexible batch job manager effectively reduces the deadline violations of Hadoop workload while it slightly increases the time violations of BUBiS workload. Even a single batch job misses the hard deadline, it may loss a large amount of system goodput due to the large input size. For example, we analyze the system goodput improvement of the selected workload set as shown in Table 5. Comparison between Tables 5 and 3 demonstrates only 0.12 percent of batch jobs contribute about 43 percent goodput improvement. Fig. 12b shows the hard time bound violation of Hadoop is significantly improved compared to the soft time bound violation. This is because the designed reward function prefers to allocate more resource to the jobs approaching hard deadlines, which aims to avoid significant system goodput lost.

TABLE 5
Effectiveness of System Goodput Improvement

Workloads	Goodput Lost	% Jobs	Average Size
Trans.	5.7 GB	6.9%	47.8 KB
Batch	42.6 GB	0.28%	4.9 GB

6.4 Benefit by Batch Job Size

Fig. 13 shows there are different deadline violation improvements by the proposed sCloud with Flexible Batch job Manager (FBM) under various batch job sizes. We use the deadline violations achieved by GLB as the baseline. The result demonstrates FBM achieves better soft time bound violation improvement for small jobs and better hard time bound violation improvement for big jobs. Apparently, sCloud with FBM can effectively reduce the hard deadline misses by the adaptive reward setting in Section 4. However, the proposed flexible batch job manager relies on the reinforcement learning process, which typically needs a few time to search the suitable job execution progress by setting the delay factor. These big jobs with long execution process provide more time to speed up or slow down the job progress. This is the reason why batch jobs with large sizes get more benefit from the flexible batch job management.

6.5 Resource and Power Control

6.5.1 Power Consumption Control

Fig. 14 shows sCloud power consumptions respect the dynamic power supplies at CA, HK and DUB, respectively.

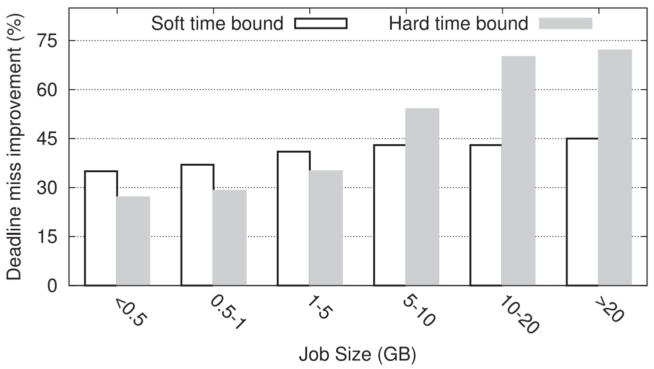


Fig. 13. Effectiveness under various batch job sizes.

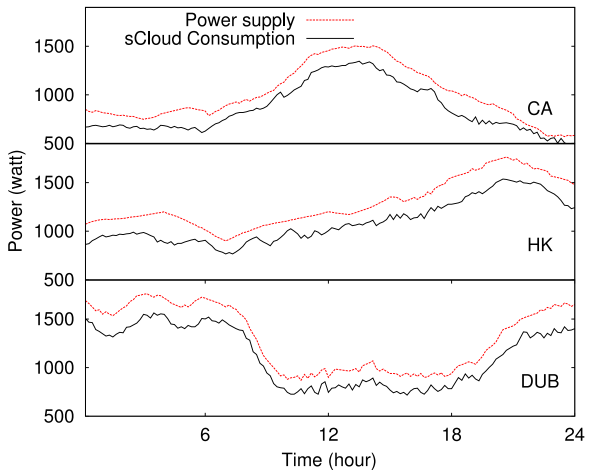


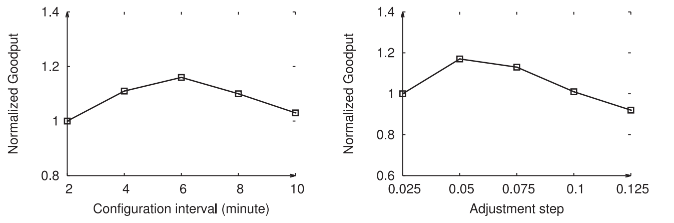
Fig. 14. Self-adaptiveness of power consumption control.

We can observe that sCloud is able to control the power consumption by adaptively increasing or reducing the available resource at each datacenter. This is due to the obtained model in Section 3 for green power prediction. sCloud applies a threshold-based power capping technique [25] to make sure that the real power consumption is below the real power supply. More results about the impact of the green power prediction error on system performance can be found in our preliminary study [26].

6.5.2 *Resource Allocation Control*

Fig. 15 shows the dynamic CPU resource allocations between RUBiS workload and Hadoop workload at CA, HK and DUB, respectively. It shows that the overall available resources are allocated in proportion to their green power supplies. Fig. 15a shows the resource allocated for RUBiS at CA increases between the 11th and 17th hours as RUBiS workload increases. sCloud allows all resource to be allocated to Hadoop workload between the 20th and 22th hours. This is because the Hadoop workload at CA is relative high and sCloud places all RUBiS workload to HK and DUB during this period. A few batch jobs and their data are migrated from CA to HK and CA to DUB between the 20th and 22th hours. More detailed results about the batch job migration can be found in our preliminary study [26].

As shown in Fig. 15b, sCloud allocates more resource to RUBiS workload while Hadoop workload is low during last eight hours. Fig. 15c shows at DUB datacenter, more resource is allocated to Hadoop workload during the beginning stage and the end stage compared to the middle stage. At the beginning stage, it is to satisfy the increased Hadoop workload at DUB datacenter. At the end stage, it is to deal with the batch jobs migrated from CA datacenter.



(a) Impact of tuning interval. (b) Impact of adjustment step.

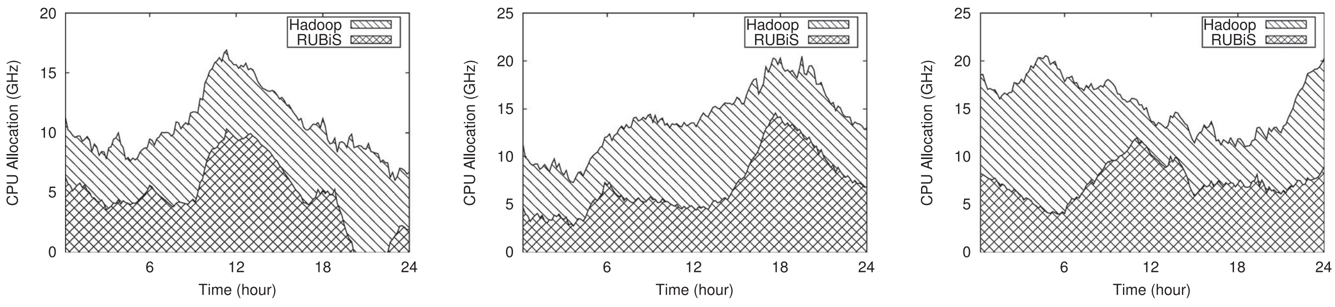
Fig. 16. Sensitivity of control parameter selection.

6.6 Sensitivity of Parameter Selection

We change the values of the flexible batch job management interval and the control factor adjustment step to study their impacts on the performance improvement in terms of the system goodput. Fig. 16a shows that the goodput initially decreases as the tuning interval increases. However, increasing the tuning interval further leads to performance degradation. This tells that a very large tuning interval may lead to the system goodput deterioration. Thus, we empirically set the tuning interval to 6 minutes in the experiment. It is a tradeoff between the search speed and the the system goodput improvement. Fig. 16b shows tuning the adjustment step of the control factor has the similar phenomenon with tuning the tuning interval of the flexible batch job manager in the experiment. A large adjustment step setting (e.g., 0.125) leads significant performance deterioration due to the instability of streaming system data processing. Thus, we empirically set the progress control factor adjustment step to 0.05 without impacting system stability in the experiment. Please note that these parameters are sensitive to workload characteristics. For example, a large batch job may prefer a larger adjustment step so that it can find a good configuration in time.

7 Related Work

Research interests are growing in integrating green energy into datacenters as the environmental concerns and the energy consumption of datacenters continuously grow. Major Internet service operators [1], [7], [8], [27] start to develop sustainable datacenters and treat it as an increasingly important mission. For example, HP Lab built up the Net-Zero Energy datacenter recently [3]. Unlike traditional



(a) Resource allocation at CA. (b) Resource allocation at HK. (c) Resource allocation at DUB.

Fig. 15. Self-adaptiveness of resource allocation by sCloud at CA, HK and DUB.

energy, the intermittency of renewable energy makes it very hard to maintain a stable cluster resource availability to process workloads. Goiri et al. proposed GreenHadoop [2], a MapReduce framework for Parasol, a prototype green cluster built in Rutgers University. It aims to maximize the green energy consumption by job scheduling. It does not consider the potential opportunities to improve the green energy usage by workload distribution across distributed datacenters. The vast majority of the previous studies on the sustainable energy management has focused on the single datacenters. These studies aim to achieve sustainable operation driven by green energy supply partially or completely from following aspects: (1) Studies [1], [2] focus on the energy demand side of a datacenter. (2) Studies [25], [28] focus on matching energy supply of a datacenter server cluster with its energy demand. (3) Studies [29], [30] focus on different energy storage approaches in the sustainable datacenters to improve their green energy usage efficiency.

Many recent studies [25], [31] make efforts to control the renewable energy generation and supply in order to improve renewable energy usage in datacenters. Gmach *et al.*, [25] proposed an approach for designing a power management plan that can match the power supply with the power demand in datacenters using renewable energy. The power management plan defines a choice for a mix of baseline power, e.g., from the power grid, power from renewable energy sources, and stored energy. Stewart *et al.*, proposed renewable energy management approaches to maximize the use of off-grid renewable energy in datacenters [31]. When renewables are intermittently unavailable, datacenters must get power from other energy sources. Recently, Aksanli et al. [1] designed an adaptive job scheduler to increase the green energy usage in a sustainable datacenter, which utilizes short term prediction of solar and wind energy production. This scheduling method may violate the QoS requirement due to unnecessarily delaying batch jobs.

Studies [32], [33] have shown that it is challenging to power a datacenter using only local wind and solar energy without large-scale storage, due to the intermittency and unpredictability of these sources. Thus, a few recent studies start to utilize green energy in distributed datacenters. Deng et al. [5] proposed an adaptive request routing approach to meet the operational cost, QoS, and carbon footprint goals. Zhang et al. [8] proposed GreenWare, a middleware system that dynamically dispatches transactional requests to distributed datacenters to maximize the use of green energy within the allowed operation budget. However, these studies only consider transactional requests that are of low cost for routing and do not consider another important category of cloud workload, i.e., batch jobs. Recently, Liu et al. proposed GLB [7], a geographical load balancing approach that can significantly reduce the required capacity of green energy by using the energy more efficiently with request dispatching. Our work differs from this effort in that we consider the workload heterogeneity and batch job migration across distributed datacenters.

8 Conclusion and Future Work

In this paper, we find heterogeneous cloud workloads should be carefully scheduled in distributed self-sustainable datacenters to maximize the overall system performance. We have proposed and developed a self-optimizing cloud workload management approach, sCloud, that can improve the system goodput with respect to the dynamic green power supply. Furthermore, we have extended sCloud by integrating a flexible batch job manager to dynamically control the job execution progress without violating the deadlines. The main technical novelty of sCloud lies in the integration of request placement, dynamic resource allocation and batch job migration. sCloud can significantly improve the system performance and green energy usage by self-optimizing workload and resource management. The new experimental result shows sCloud with the flexible batch job management approach can further increase the system goodput by additional 11 percent compared to the original sCloud approach. Our future work will integrate the virtualization techniques and the renewable energy distribution methods with sCloud to further improve sustainable computing in green datacenters.

Acknowledgments

A preliminary version of this paper appeared in [26]. Cheng's research was supported by the University of North Carolina at Charlotte.

References

[1] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," in *Proc. USENIX Workshop Power Aware Comput. Syst.*, 2011, Art. no. 5.

[2] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "GreenHadoop: Leveraging green energy in data-processing frameworks," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 57–70.

[3] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," in *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Model. Comput. Syst.*, 2012, pp. 175–186.

[4] H. Dou, Y. Qi, and W. Wei, "Carbon-aware electricity cost minimization for sustainable data centers," *IEEE Trans. Sustainable Comput.*, vol. 2, no. 2, pp. 211–223, Apr.–Jun. 2017.

[5] N. Deng, C. Stewart, D. Gmach, M. Arlitt, and J. Kelley, "Adaptive green hosting," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2012, pp. 135–144.

[6] P. Lama and X. Zhou, "NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2012, pp. 1–12.

[7] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew, "Geographical load balancing with renewables," in *Proc. ACM SIGMETRICS*, 2011, pp. 62–66.

[8] Y. Zhang, Y. Wang, and X. Wang, "GreenWare: Greening cloud-scale data centers to maximize the use of renewable energy," in *Proc. ACM/IFIP/USENIX Int. Middleware Conf.*, 2011, pp. 140–159.

[9] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2015, pp. 485–497.

[10] Y. Guo, P. Lama, and X. Zhou, "Automated and agile server parameter tuning with learning and control," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2012, pp. 656–667.

[11] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. IEEE/ACM Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2011, pp. 390–399.

[12] A. Verma, L. Cherkasova, and R. H. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in *Proc. ACM/IFIP/USENIX Int. Middleware Conf.*, 2011, pp. 160–179.

[13] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy, "Cloudy computing: Leveraging weather forecasts in energy harvesting sensor systems," in *Proc. 7th Annu. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw.*, 2010, pp. 1–9.

[14] NREL, Measurement data center, (2012). [Online]. Available: http://www.nrel.gov/midc/

[15] J. Tan, X. Meng, and L. Zhang, "Delay tails in MapReduce scheduling," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 5–16, 2012.

[16] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 13–26.

[17] Z. Liu and S. Yu, "The M/M/C queueing system in a random environment," *J. Math. Anal. Appl.*, vol. 436, no. 1, pp. 556–567, 2016.

[18] L. Rao, X. Liu, X. L. Xie, and W. Liu, "Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2010, pp. 1–9.

[19] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user MapReduce clusters," Univ. California at Berkeley, Berkeley, CA, Tech. Rep. UCB/EECS-2009-55, 2009.

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

[21] Vmware: Host power management in VMware vSphere 5.5. (2013). [Online]. Available: http://www.vmware.com/files/pdf/techpaper/hpm-perf-vsphere55.pdf

[22] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Elsevier Comput. Netw.*, vol. 53, pp. 1830–1845, 2009.

[23] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 43–56.

[24] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances Neural Inf. Process. Syst.*, pp. 1038–1044, 1996.

[25] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, A. Shah, R. Sharma, and Z. Wang, "Capacity planning and power management to exploit sustainable energy," in *Proc. IEEE Int. Conf. Netw. Serv. Manage.*, 2010, pp. 96–103.

[26] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-aware workload placement and migration in distributed sustainable datacenters," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 307–316.

[27] S. Ren and Y. He, "COCA: Online distributed resource management for cost minimization and carbon neutrality in data centers," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2013, pp. 1–12.

[28] C. Li, R. Zhou, and T. Li, "Enabling distributed generation powered sustainable high-performance data center," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2013, pp. 35–46.

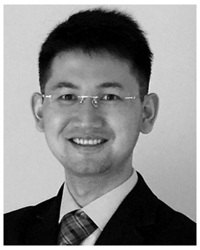
[29] T. Yang, H. Pen, W. Li, D. Yuan, and A. Zomaya, "An energy-efficient storage strategy for cloud datacenters based on variable K-coverage of a hypergraph," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3344–3355, Dec. 2017.

[30] W. Zheng, K. Ma, and X. Wang, "Ahybrid energy storage with supercapacitor for cost-efficient data center power shaving and capping," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1105–1118, Apr. 2017.

[31] C. Stewart and K. Shen, "Some joules are more precious than others: Managing renewable energy in the datacenter*," in *Proc. USENIX Workshop Power Aware Comput. Syst. (HotPower)*, 2009, pp. 45–49.

[32] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," in *Proc. Int. Conf. Archit. Support Programm. Lang. Operating Syst.*, 2012, pp. 75–86.

[33] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. K. Fathy, "Energy storage in datacenters: What, where, and how much?" in *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Model. Comput. Syst.*, 2012, pp. 187–198.



Dazhao Cheng received the BS and MS degrees in electronic engineering from the Hefei University of Technology, in 2006 and the University of Science and Technology of China, in 2009, respectively. He received the PhD degree from the University of Colorado, Colorado Springs, in 2016. He is currently an assistant professor with the Department of Computer Science, University of North Carolina, Charlotte. His research interests include big data and cloud computing. He is a member of the IEEE and ACM.

Xiaobo Zhou received the BS, MS, and PhD degrees in computer science from Nanjing University, in 1994, 1997, and 2000, respectively. Currently he is a professor of the Department of Computer Science, University of Colorado, Colorado Springs. His research lies broadly in computer network systems, specifically, cloud computing and datacenters, BigData parallel and distributed processing, autonomic and sustainable computing, scalable Internet services and architectures. He was a recipient of the NSF CAREER Award in 2009. He is a senior member of the IEEE.

ZhiJun Ding received the MS degree from the Shandong University of Science and Technology, Taian, China, in 2001, and the PhD degree from Tongji University, Shanghai, China, in 2007. Currently he is a professor and the chair of the Department of Computer Science and Technology, Tongji University. His research interests are in concurrency, petri nets, and services computing. He is a member of the IEEE.

Yu Wang received the BEng and MEng degrees in computer science from Tsinghua University, China, and the PhD degree in computer science from the Illinois Institute of Technology. He is a professor of computer science at the University of North Carolina at Charlotte. His research interest includes wireless networks, mobile social networks, smart sensing, and mobile computing. He has published more than 150 papers in peer reviewed journals and conferences, with four best paper awards. He is a IEEE Fellow and senior member of the ACM.

Mike Ji received the PhD degree in computer science from Nanjing University, in 1997. He was a postdoc and computer scientist in Stanford Research International, in 1998. Currently, he is the chief scientist in Valley Technologies Ltd, focusing on data center design and implementation, IoT operating system, and concurrent system formal description.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.