Dynamic Workload Management in Heterogeneous Cloud Computing Environments

Qi Zhang and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON Canada
{q8zhang,rboutaba}@cs.uwaterloo.ca

***Abstract*—Cloud computing is a paradigm that harnesses massive resource capacity of data centers to support applications in a scalable, flexible, reliable and cost-effective manner. Despite its recent success and rapid adoption in the IT industry, recent literature has shown that effective workload management in cloud computing environments remains to be a difficult challenge. A key reason behind this difficulty is that resources and workloads in production environments are both heterogeneous and dynamic. In particular, large cloud data centers often consist of machines with heterogeneous capacities and performance characteristics. At the same time, cloud workloads often show significant diversity in terms of priority, resource requirements, arrival rate and performance objectives. Consequently, it is difficult to devise heterogeneity and dynamicity-aware management scheme that satisfy diverse application performance objectives, while reducing operational expenses such as energy consumption.**

This work addresses several key challenges pertaining to dynamic workload management in heterogenous Cloud environments. Specifically, we first present a scheme that place service application across geographically distributed data centers to meet service demand while minimizing total resource usage cost. Then, we design a heterogeneity-aware dynamic application provisioning technique to minimize energy consumption while satisfying performance objectives. Finally, we study the problem of MapReduce scheduling and present a novel scheme that leverages heterogenous run-time task usage characteristics. Through experiments and simulations, we show our proposed solutions can significantly reduce data center energy consumption, while achieving better application performance in terms of service response time and job completion time.

I. Introduction

The rapid development of Internet technologies in recent years has enabled the delivery of rich services at an unprecedented scale. In this context, Cloud computing has emerged as a model that harnesses massive capacities of data centers to host services in a cost-effective manner. In a cloud computing environment, the traditional role of service provider is divided into two: *cloud providers* who owns the physical data center and lease resources (e.g. virtual machines) to service providers; and *service providers* who use resources leased by cloud providers to execute applications. By leveraging the economies-of-scale of data centers, Cloud computing can provide significant reduction in operational expenditure. At the same time, it also enables new applications such as big-data analytics (e.g. MapReduce [1]) that process massive volumes of data in a scalable and efficient manner.

978-1-4799-0913-1/14/$31.00 © 2014 IEEE

However, as IT organizations continue to deploy and expand their Cloud infrastructures and services, Cloud data centers are also growing in scale and complexity, making them increasingly more difficult to manage. For instance, recent work [2], [3], [4], [5] suggests resource management in large production data centers is an important yet difficult challenge, as these data centers often consist of heterogeneous machines that execute vast number of applications and jobs under dynamic conditions. We summarize the types of heterogeneity and dynamicity found in production data centers as follows:

Infrastructure heterogeneity and dynamicity: Production data centers often comprise several types of machines with heterogenous hardware processor architecture, processor speed, memory, disk size and energy consumption characteristics. This is partly due to the fact that certain types of workload can only be scheduled on particular types of machines. It also stems from the fact that Cloud providers continue to upgrade their data center capacities over time [2], [6]. Indeed, as physical servers may become outdated, it is necessary to purchase new machines over-time and keep old machines while they are still functional. As a result, production data centers often consist of multiple generations of machines with heterogenous characteristics. Finally, cloud providers often build across geographically distributed locations to reduce network latency while improving service reliability. These data centers often operate in dynamic conditions, as electricity price and network conditions may vary over time [7].

Workload heterogeneity and dynamicity: Production data centers typically run vast number of applications with diverse characteristics [3], [4], [5]. In particular, an application can be divided into one or more processes running in dedicated *containers* such as Virtual Machines (VMs). Containers can have different levels of priorities ranging from production (highest priority) to best effort (lowest priority) [5]. Furthermore, the resource requirement differs from container to container. It has been reported that most containers are small in terms of resource requirements, whereas a small fraction of containers can be much larger than the others. The arrival rate of resource requests also vary from time to time, depending on the actual service demand [8]. It has been reported that the arrival rate during busy hours can be several orders of magnitude higher than during idle hours. Furthermore, the life time of applications can vary significantly. Typically user-facing services such as web servers can run for a long time, whereas batch jobs typically have short running time. Finally, run-time resource consumption of processes (e.g. MapReduce tasks) can fluctuate depending on the phases they are in [9].

The heterogeneity and dynamicity of both machines and workloads has profound implications on the design of Cloud workload management schemes. From a service provider's perspective, given a variety of data center locations and time-varying service demand from Internet users, it is challenging to determine where resource requests should be submitted in order to minimize total resource usage while satisfying server level objectives (SLOs). From a Cloud provider's perspective, given heterogenous machine and workload characteristics, it is a challenging to devise efficient scheduling schemes that improves resource utilization, while minimizing operational costs. In particular, energy is a major concern of cloud providers as its accounts for 18% of total data center operational expenditures [10]. Driven by its importance and difficulties, cloud workload management has attracted significant attention from the research community, with many research topics being actively pursued in recent years [3], [4], [5], [11].

This dissertation addresses several key challenges pertaining to dynamic workload management in heterogeneous Cloud environments. Specifically, we address the concerns of service providers by presenting a control-theoretical approach to dynamic service placement across geographically distributed data centers, with the goal of satisfying service demand while minimizing total resource usage cost. From the perspective of cloud providers, we study the problem of heterogeneity-aware capacity provisioning in data centers to minimize energy consumption while achieving low application scheduling (i.e. queuing) delay. Finally, we present a novel scheduling algorithm that leverages the run-time resource usage variability of MapReduce tasks to improve job running time and resource utilization. Through experiments using real workload traces including one from a production Google compute cluster [12], we show our solutions can significantly reduce data center operational costs, while achieving better performance in the terms of service response time and job completion time.

The rest of this paper is organized as follows: Section II, III and IV summarize the technical solutions for the 3 problems addressed in the dissertation, namely, the dynamic service placement in distributed clouds, heterogeneity-aware resource provisioning and fine-grained phase-level resource scheduler for MapReduce. We then conclude the paper in Section V.

II. Service Placement in Geo-Distributed Clouds

Service providers have been increasingly relying on geographically distributed cloud infrastructures for service hosting and delivery. In this context, a key challenge faced by service providers is to determine where service applications should be placed such that the hosting cost is minimized, while key performance requirements (e.g. response time) are assured. This involves solving two problems jointly: (1) deciding on the number of servers placed in each data center, and (2) routing each request to appropriate servers to minimize response time. As cloud providers typically offer elastic (i.e., on-demand) resource access, it is possible to adjust the number of servers to match service demand in a dynamic way. However, the cost of reconfiguration (i.e., the cost of adding and removing servers) must be taken into account. The consideration of reconfiguration cost is important for ensuring the system stability and minimum management overhead and costs. In particular, these operations have costs for setup (e.g., VM image distribution) and tear-down (e.g., data fetching / state transfer). For example, it has been reported that starting up a VM in Amazon EC2 cloud can take between 20 seconds to more than 13 minutes, depending on the VM size and OS running in the VM [13]. Thus, the time it takes to scale up the service needs to be considered when making scaling decisions, as under-provisioning servers during the scaling up process can cause revenue loss. Thus, it is in the interest of service providers to reduce such reconfiguration cost. We call this problem *dynamic service placement problem (DSPP)*. This problem shares many similarities with traditional replica placement problem in [14], [15], [16]; however, the price fluctuation and reconfiguration cost are often neglected.

In this section, we study DSPP using control theoretic methods. Specifically, we model the network as a bipartite graph $G = (L \cup V, E)$, where L denotes the set of data centers, V denotes set of access networks to which customers are connected. We define $E \subseteq L \times V$ the communication paths between customers and data centers. We also assign constant weights d^{lv} to denote the network latency between a data center $l \in L$ and a client location $v \in V$. In our framework, we consider a discrete-time model where time is divided into multiple time intervals. We assume that there is an interval of interest $\mathcal{K} = \{0, 1, 2, ..., K\}$ that consists of $K + 1$ periods. Let $N = \{1, 2, ..., n\}$ denote the set of service providers. We assume that at time $k \in \mathcal{K}$, each customer location $v \in V$ has demand D_k^v in terms of average arrival rate of requests from location v at time k. For simplicity, we assume that all the servers rented by each service provider have identical size and functionality. We define the state variable $x_k^l \in \mathbb{R}_+$ as the number of servers owned by the service provider at location $l \in L$ at time k. To simplify the model, we assume that x_k^l can take continuous values rather than discrete values. This assumption is reasonable for large-scale services that require tens or hundreds of servers, where the weight of each individual server in the overall solution is small. In this case, we can always obtain a feasible solution by rounding up the continuous values to the nearest integer values. Based on this assumption, we can further decouple x_k^l by defining $x_k^{lv} \in \mathbb{R}_+$ as the number of servers at location l serving demand from $v \in V$, and define $x_k^l = \sum_{v \in V} x_k^{lv}$ for all $l \in L, 0 \leq k \leq K$. Let $u_k^{lv} \in \mathbb{R}$ denote the change in x_k^{lv} at time k, we then have:

$$x_{k+1}^{lv} = x_k^{lv} + u_k^{lv}, \quad \forall l \in L, v \in V, 0 \leq k \leq K. \qquad (1)$$

To model the cost of server allocation, we assume that there is a price p_k^l for running a server at data center $l \in L$ at time k. The total resource cost R_k for service hosting at time k is

$$R_k = \sum_{l \in L} x_k^l p_k^l \qquad \forall 0 \leq k \leq K \qquad (2)$$

We also assume that there is a convex function $g : \mathbb{R} \rightarrow \mathbb{R}_+$ that computes the cost of reconfiguration. A possible reconfiguration cost function is

$$G_k = \sum_{l \in L} c^l_{on} (\sum_{v \in V} u^l_k)^+ - c^l_{off} (\sum_{v \in V} u^l_k)^-, \qquad (3)$$

where c^l_{on} and c^l_{off} are the average monetary costs for adding an additional VM and removing a VM, respectively. For example, the c^l_{on} can measure the performance penalty during the startup time of a server. Our framework can also support other reconfiguration cost functions as well.

In addition, there is a SLA performance requirement that specifies a desired average delay $\bar{d}$ that the service should achieve. In our model, we define σ^{lv}_k as the demand arrival rate from v assigned to data center l at time k, such that $\sum_{l \in L} \sigma^{lv}_k = D^v_k$ for all $v \in V$ and $0 \leq k \leq K$. We assume there is a load balancer placed in each data center $l \in L$ such that demand $\sigma^l_k = \sum_{v \in V} \sigma^{lv}_k$ arriving from location v is equally split among the local servers x^l_k. Assuming each server has mean service time μ, the utilization of a server can be defined as $\rho^l_k = \frac{\sigma^l_k}{x^l_k \mu}$. We assume the queuing delay of a server is a convex function $f(\cdot)$ of the server utilization ρ^l_k, i.e. $q^l_k = f(\rho^l_k)$. This is true for most of the queuing systems. For example, if each server can be modeled as a $G/G/1$ queue, then the queueing delay can be approximated by [17] as $q^l_k = \left(\frac{\rho^l_k}{1-\rho^l_k}\right)\left(\frac{c_a^2+c_s^2}{2}\right) \cdot \frac{1}{\mu}$ where c_a and c_s are the coefficient of variation of arrival rate (i.e. $\frac{\sigma^l_k}{x^l_k}$) and service time, respectively. It is clearly convex in ρ^l_k.

Our goal is ensure that for any $(v, l) \in E$ with $\sigma^{lv}_k > 0$, the average delay (i.e., the sum of propagation and queuing delay) is less than $\bar{d}$, i.e. $d^{lv} + q^l_k \leq \bar{d}$ for $0 \leq k \leq K$. In our framework, we can model the performance objective as a SLA penalty function. Assume there is a convex penalty function $h(\cdot)$ that measures revenue loss due to exceeding the delay requirement. By defining $a^{lv} = \frac{1}{f^{-1}(\bar{d}-d^{lv})\mu}$ The total performance penalty can be expressed as:

$$P_k = \sum_{v \in V} h(\sum_{l \in L} \frac{x^{lv}_k}{a^{lv}} - D^v_k) \qquad 0 \leq k \leq K \qquad (4)$$

We also assume each data center possesses R types of resources, let s^r denote the size of a server for a resource type $r \in R$, and C^{lr} is the capacity of data center l for resource type r, we can use a convex function $\pi^r(\cdot)$ to represent the penalty due to violation of capacity constraint.

$$C_k = \pi^r((s^r x^l_k - C^r)^+) \quad \forall l \in L, 0 \leq k \leq K, r \in R \quad (5)$$

In practice, this penalty can be measured as scheduling delay as we shall elaborate in the next section. Finally, the goal of DSPP is to minimize the following objective function:

$$J = \sum_{k=0}^{K} R_k + G_k + P_k + C_k$$

subjects to constraints (1) and $x^{vl}_k \geq 0 \, \forall \, l \in L, k \in \mathcal{K}$.

Algorithm 1 MPC Algorithm for DSPP

```
1: Provide initial state $\mathbf{x}_0$, $k \leftarrow 0$
2: while true do
3:     At beginning of control period $k$:
4:     Predict $D^v_{k+i|k}$ for horizons $i = 1, \cdots, K$
5:     Solve DSPP to obtain $u^{lv}_{k+t|k}$ for $t = 0, \cdots, W-1$
6:     Change the resource allocation according to $u^{lv}_{k|k}$
7:     $k \leftarrow k + 1$
```

The offline version of DSPP is a convex optimization problem that can be solved optimally using standard methods [18]. However, the resource controller must solve this problem online where the future demand is unknown. In this case, we use the Model Predictive Control (MPC) framework that is widely used for solving online control problems. Algorithm 1 is our online MPC algorithm that can be described as follows. At time k, the system forecasts the future demand D^v_k for a time horizon $[k+1, ..., k+W]$. Let $D^v_{k+t|k}$ denote the demand predicted for time $k+t$ at time k. The controller then solves the optimization problem for the horizon $[k, ..., k+W]$, starting with the initial state $x^{lv}_{k|k} = x^{lv}_k$. Even though the solution of the optimization problem will contain a set of values $u^{lv}_{k|k}, ..., u^{lv}_{k+W-1|k}$, the controller will only execute the first step in sequence $u^{lv}_{k|k}$. When the next control period $k+1$ starts, the same procedure is performed again by the controller.

Results: We have implemented and evaluated our solution using a real Internet topology graph from the *Rocketfuel project* [19]. In our experiment, we have created 3 large data centers located in Mountain View, CA, Houston, and TX, Atlanta, GA. To generate realistic service requests, we used the Worldcup 98 dataset [20] which contains HTTP requests for a total duration of 92 days. We use the request regions provided in the dataset to approximate the source of requests. To demonstrate the benefit of our approach, we present the result for a simple scenario involving one access network and two data centers, as shown in Figure 1. It can be seen that the servers are provisioned according to service demand. Finally, we demonstrate the importance of reconfiguration cost. We also compared the result with a greedy algorithm [14] that ignores reconfiguration cost. The output of the algorithm is shown in Figure 2. Indeed, greedy algorithm causes massive migrations when resource price fluctuates. It is evident that reconfiguration cost plays a crucial role in avoiding massive migrations in this scenario.

III. Heterogeneity-Aware Dynamic Capacity Provisioning

Reducing energy consumption has become a major concern of cloud providers, as energy cost accounts for a significant portion of data center operational expenditures. One promising technique for reducing energy consumption is *Dynamic Capacity Provisioning* (DCP), whose goal is to dynamically adjust the number of active machines in a data center to save energy while meeting workload performance objectives. In the context of workload scheduling in data centers, a metric of particular importance is *scheduling delay* [4], [3], [2], [21], which is the time a request waits in the scheduling queue before it is scheduled on a machine. However, while DCP has been studied extensively in the literature, the heterogeneity aspect of both infrastructure and workload has been largely overlooked, resulting in a number of serious deficiencies. For instance, given a rise of workload requests, a heterogeneous-oblivious DCP scheme can turn on wrong types of machines which are not capable of handling these requests (e.g., due to insufficient capacity), resulting in both resource wastage and high scheduling delays. Furthermore, as frequently switching a machine on and off reduce its lifetime [22], it is necessary to consider switching costs while making DCP decisions.

In this section, we present Harmony, a **H**eterogeneity-**A**ware **R**esource **MON**itoring and management s**Y**stem that addresses the aforementioned challenges. We first characterize the workload by dividing tasks into *task classes* using the K-means algorithm. Once the workload characterization has been obtained, we introduce a monitoring mechanism that allows Harmony to capture the run-time workload composition in terms of arrival rate for each task class. To make provisioning decisions, we define a *container* as a logical allocation of resources to a task that belongs to a task class. In our approach, the task containers serve as resource reservations that help the controller to make machine allocation decisions. we call this approach *container-based provisioning* (CBP).

In our model, we assume time is divided into intervals of equal duration. Let $z_t^m \in \mathbb{R}^+$ denote the number of type m machines that are active at time t, and $\delta_t^m \in \mathbb{R}$ the change in δ_t^m at the end of time t. Similarly, define $x_t^{mk} \in \mathbb{R}^+$ as the number of type k containers assigned to machines of type m and $\sigma_t^{mk} \in \mathbb{R}^+$ as the change in x_t^{mk} at time t. We now have:

$$z_{t+1}^m = z_t^m + \delta_t^m \quad (6)$$
$$x_{t+1}^{mk} = x_t^{mk} + \sigma_t^{mn} \quad (7)$$

We also need to ensure that each type of containers can only be assigned to machines that are capable of hosting them. This is achieved by introducing a constant $\psi^{mk} \in \{0,1\}$ that indicates whether a type n container can be scheduled on a type m machine. Let c^{kr} and C^{mr} denote the capacity of a type k container and a type m machine for each resource $r \in R$ respectively, we have the following schedulability constraint:

$$c^{kr} x_t^{mn} \le z_t^m \psi^{mk} C^{mr} \quad \forall m \in M, k \in K, r \in R, t \in \mathcal{T} \quad (8)$$

As total energy usage of a physical machine can be estimated by a linear function of resource utilization [23], let p_t denote the energy price at time t, the energy consumption of all the active machines at time t can be computed as:

$$E_t = p_t \left(z_t^m E^{idle,m} + \sum_{r \in R} \sum_{k \in K} \frac{\alpha^{mr} c^{kr}}{c^{mr}} \cdot x_t^{mk} \right) \quad (9)$$

where $E^{idle,m} \in \mathbb{R}^+$ is the energy consumption of a type m machine when it is idle, and $\alpha^{mr} \in \mathbb{R}^+$ is the slope of the energy consumption function.

To model task scheduling delay, since it is not possible for all containers to be scheduled when demand exceeds data center capacity, we assume there is a utility function $f^k(\cdot)$ that models the monetary gain for scheduling containers. $f^k(\cdot)$ is assumed to be a concave function that can be derived from SLO objectives. For example, $f^k(a^k)$ can model the gain in monetary cost when a^k containers are scheduled for task class k. The total revenue can now be written as:

$$U_t^{perf} = \sum_{k \in K} f^k\big(\sum_{m \in M} \sum_{i \in N_t^m} a_t^{ik} \big) \quad (10)$$

The machine switching cost can be described by:

$$C_t^{sw} = \sum_{m \in M} q^{on,m} (\delta_t^m)^+ + q^{off,m} (\delta_t^m)^- \quad (11)$$

where $q^{on,m} \in \mathbb{R}^+$ and $q^{off,m} \in \mathbb{R}^+$ denotes the cost for switching on and off of a single type m machine, respectively. Finally, equation (12) ensures that provisioned containers should not exceed the provisioned machine capacity:

$$\sum_{n \in N} c_n^r x_t^{mn} \le z_t^m C^{mr} \qquad \forall m \in M, r \in R, t \in \mathcal{T} \quad (12)$$

The heterogenity-aware DCP problem can now be stated as:

$$\max_{\delta_t^m, \sigma_t^{mk}} \sum_{t=0}^{T} U_t^{perf} - C_t^{sw} - E_t,$$

subject to $z_t^m \le N_t^m$ for all $m \in M, t \in \mathcal{T}$ along with constraints (6), (7), (8) and (12). This problem is a convex optimization problem that can be solved using standard methods [18]. Again, we apply the MPC framework to solve the problem online. At a given time t, we predict the future values for electricity price and arrival rate and solve the problem. Once the problem is solved, we simply round up the fractional values of (x_t^{mk}, z_t^m) to obtain an integer solution for DCP, which gives the number of machines to be provisioned (i.e., $\lceil z_t^m \rceil$) and the number of type n tasks that should be scheduled on type m machines (i.e., $\lceil x_t^{mk} \rceil$). However, at run time, the scheduler needs to ensure that the number of type n tasks assigned to type m machines must respect the provisioned capacity $\lceil x_t^{mk} \rceil$. A simple strategy is to ensure the number of type k tasks assigned to m (denoted by $Assign_t^{mk}$) is proportional to the number of containers, i.e. $Assign_t^{mk} = \frac{x_t^{mk}}{\sum_{j \in M} x_t^{jk}}$ This can be achieved easily by using a weighted round-robin scheduling policy. Furthermore, it can be easily integrated with existing scheduling algorithms. For example, variants of first-fit and best-fit algorithms (which are used in Microsoft [24], Google [3] and Eucalyptus [25]) can adopt this mechanism by changing the scheduling policy to weighted round-robin first-fit and best-fit, respectively.

A key drawback of the above rounding scheme is that it often under-estimates the required capacity. The reason is that the fractional solution assumes that each container can be arbitrarily divided and placed on multiple machines, which is not realizable in practice. To account for the under-estimation of machine capacities, we define an over-provisioning factor $\omega^k \in \mathbb{R}^+$ that captures how much extra resource is required to fully pack a given set of type n containers. To account for ω^k, it suffices to replace equation (8) by the following equation:

$$\sum_{k \in K} \omega^k c^{kr} x_t^{mk} \leq z_t^m C^{mr} \qquad \forall m \in M, r \in R, t \in \mathcal{T} \quad (13)$$

The value of ω^n can be obtained through experiments. We found setting $\omega^k = 1.2$ to be a reasonable value in practice.

Results: We have evaluated the performance of our algorithm using Google workload traces [26]. For the purpose of comparison, we have also implemented a baseline (heterogeneity-oblivious) algorithm that greedily switches on and off machines based on energy efficiency. We use an over-provisioning factor of 1.2 to demonstrate the behavior of our algorithms. The number of active servers provisioned by the baseline algorithm and CBP are shown in Figure 3 and Figure 4 respectively. It can be seen that CBP uses fewer machines compared to baseline algorithm. Furthermore, the CDF of task scheduling delays are shown in Figure 5, 6 respectively. CBP clearly achieves substantial reduction in scheduling delay compared to the baseline algorithm. Finally, our work can be extended in the future to address the problem of data center capacity planning and upgrade.

IV. Fine-Grained Resource-Aware Scheduling for MapReduce

MapReduce [1] is a popular programming model for large-scale, data-intensive computation. In MapReduce, a job is a collection of *Map* and *Reduce* tasks that can be scheduled concurrently on multiple machines, resulting in significant reduction in job running time. However, traditional MapReduce systems (e.g. Hadoop MapReduce Version 1) use simple slot-based resource allocation scheme, where physical resources on each machine are captured by the number of identical slots that can be assigned to tasks. In practice, this simple scheme is often inefficient as run-time resource consumption varies from task to task and from job to job. Motivated by this observation, several recent proposals, such as Resource-Aware Adaptive Scheduling (RAS) [27] and Hadoop MapReduce Version 2 (also known as Hadoop NextGen and Hadoop Yarn) [28], have introduced resource-aware job schedulers to the MapReduce framework. These schedulers specify a fixed size for each task in terms of required resources (e. g. CPU and memory), thus assuming the run-time resource consumption of the task is static over its life time. However, this is not true for many MapReduce jobs. In particular, it has been reported that the execution of each MapReduce task can be divided into multiple phases of data transfer, processing and storage [9]. A *phase* is a sub-procedure in the task that has a distinct purpose and can be characterized by the uniform resource consumption over its duration. This phase-level scheduling has the potential to further improve job performance and resource utilization.

In this section, we present PRISM, a **P**hase and **R**esource **I**nformation-aware **S**cheduler for **M**apReduce clusters that performs resource-aware scheduling at the level of phases. In PRISM, when a phase that belongs to a task needs to be executed, the task sends a request to the local task tracker which

Algorithm 2 Phase-Level Scheduling Algorithm

```
1: Upon receiving a status message from machine $n$
2: Compute the resource utilization of machine $n$
3: $PhaseSelected \leftarrow \{\emptyset\}$
4: $CandidatePhases \leftarrow \{\emptyset\}$
5: for each job $j$ in the system do
6:     for each scheduable phase $i \in j$ do
7:         $CandidatePhases \leftarrow CandidatePhases \cup \{i\}$
8: while $CandidatePhases \neq \emptyset$ do
9:     for $i \in CandidatePhases$ do
10:        if $i$ is not schedulable on $n$ given current utilization then
11:            $CandidatePhases \leftarrow CandidatePhases \backslash \{i\}$
12:            continue;
13:        Compute the utility $U(i,n)$ as in equation (14)
14:        if $U(i,n) \leq 0$ then
15:            $CandidatePhases \leftarrow CandidatePhases \backslash \{i\}$
16:    if $CandidatePhases \neq \emptyset$ then
17:        $i \leftarrow$ task with highest $U(i,n)$ in the $CandidatePhases$
18:        $PhaseSelected \leftarrow PhaseSelected \cup \{i\}$
19:        $CandidatePhases \leftarrow CandidatePhases \backslash \{i\}$
20:        Update the resource utilization of machine $n$
21: return $PhaseSelected$
```

forwards received requests to the job scheduler periodically. The job scheduler then decides whether the phase is allowed to proceed, in which case the response is first sent back to the task tracker and then forwarded to the task process.

In order to perform phase-level scheduling, PRISM requires phase-level resource information for each job. Existing state-of-the-art resource profilers, such as Starfish [9], can already provide accurate phase-level resource information for PRISM. This profile-driven scheduling approach has been widely adopted by many MapReduce systems [29], [30], [27]. In the absence of phase-level resource information, PRISM can fall back to use task-level resource information for Hadoop Yarn to schedule phases without introducing deficiencies.

We now describe the scheduling algorithm used by PRISM. When deciding which phase should be scheduled on a machine, the schedule needs to make a trade-off between fairness and performance. Performance is important as we want to take advantage of phase-level scheduling to make jobs run faster. Fairness is also important as we want to allocate sufficient resources to each job so that no job will be severely delayed. Specifically, each job j in the system consists of two types of tasks: map tasks M and reduce task R. Let $\tau(t) \in \{M, R\}$ denote the type of a task t. Given a phase i belonging to a task t that can be scheduled on a machine n, we define the utility of assigning a phase i to machine n as:

$$U(i,n) = U_{fairness}(i,n) + \alpha \cdot U_{perf}(i,n) \qquad (14)$$

where $U_{fairness}$ and U_{perf} represent the utilities for improving fairness and job performance, respectively, and α is an adjustable weight factor. If we set α to a value close to zero, then the algorithm will greedily schedule phases according to the improvement in fairness. Specifically, we define

$$U_{fairness}(i,n) = U_{fairness}^{before}(i,n) - U_{fairness}^{after}(i,n) \qquad (15)$$

where $U_{fairness}^{before}(i,n)$ and $U_{fairness}^{after}(i,n)$ denotes the fairness

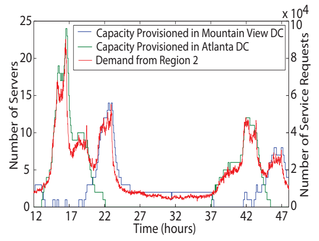
Fig. 1. Result of DCPP Algorithm

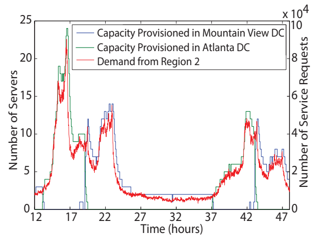
Fig. 2. Result of Greedy Algorithm

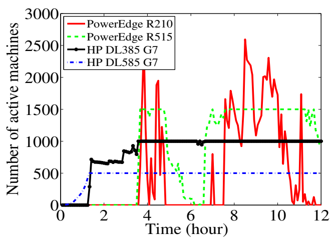
Fig. 3. DCP using baseline

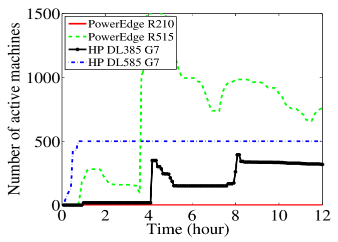
Fig. 4. DCP using CBP

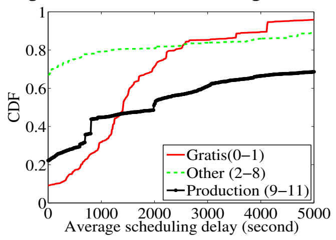
Fig. 5. Scheduling delay for baseline

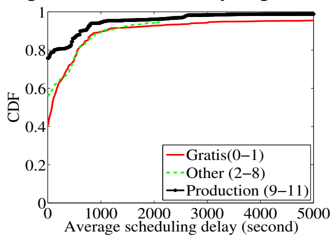
Fig. 6. Scheduling delay for CBP

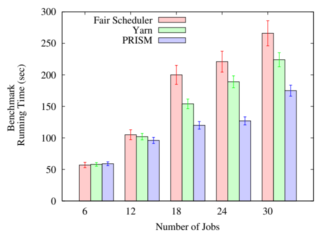
Fig. 7. Benchmark Running Time

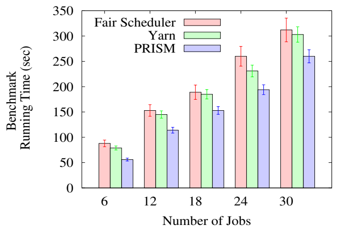
Fig. 8. Benchmark Running Time

measure of the user before and after scheduling i on n, respectively, according to fairness metrics such as Dominant Resource Fairness (DRF) [31]. On the other hand, $U_{perf}(i,n)$ is more difficult to compute. As mentioned previously, if i is the first phase of a map (or reduce) task t, then $U_{perf}(i,n)$ measures the gain in parallelism in terms of the number of running map tasks (or reduce tasks). Otherwise, if i is a subsequent phase of task t, then $U_{perf}(i,n)$ measures the gain in shortening the running time of task t. Formally, we define

$$U_{perf}(i,n) = \begin{cases} U_{task}(i,n) & i \text{ is the first phase of a task} \\ U_{phase}(i,n) & \text{Otherwise} \end{cases}$$

Even though PRISM does not restrict the function for computing the utility of a phase, in our current implementation, we have chosen $U_{task}(i,n)$ to be

$$U_{task}(i,n) = \frac{N_{remaining}}{\max\{N_{current}, \epsilon\}} - \frac{N_{remaining}}{N_{current}+1} \quad (16)$$

where $N_{remaining}$ denotes the number of remaining tasks of type $\tau(t)$, and $N_{current}$ denotes the number of running tasks of type $\tau(t)$. The variable ϵ is used to prevent dividing by 0.

On the other hand, let T^t_{wait} denote the number of seconds that task t has been paused due to phase-based scheduling. The utility for scheduling a non-leading phase i of task t can be expressed as a function $p(\cdot)$ of T^t_{wait}:

$$U_{phase}(i,n) = p(T^t_{wait}) \quad (17)$$

There are many possible choices for $p(\cdot)$. In our implementation, we have chosen $p(\cdot)$ to be a quadratic function $p(T^t_{wait}) = a \cdot p(T^t_{wait})^2 + b$. The intuition is to increase the urgency for scheduling i more rapidly if i has been paused for a long time. However, PRISM can adopt any type of utility function $p(\cdot)$ as long as it is a monotonically increasing function. Finally, the scheduling algorithm used by PRISM is represented by Algorithm 2.

Results: We have implemented PRISM and compared it with Hadoop Yarn 2.0.4 and Hadoop 0.20.2 with fair scheduler. Hadoop Yarn 2.0.4 is a recent version of Hadoop NextGen that allows the users to specify both CPU requirement (i.e. number of virtual cores) and memory requirement (i.e. GB of RAM) of each task. Hadoop 0.20.2 with fair scheduler serves as the baseline for comparison. For analysis purposes, we have implemented a simple job profiler that captures the CPU, memory and I/O usage of both map and reduce tasks at phase-level. For each of the schedulers, we run both the PUMA [32] and Gridmix 2 [33] benchmarks. For each benchmark, we varied the number of jobs $2\times - 10\times$ to create batch workload of different sizes. We found PRISM can achieve up to 24% reduction in job running time as shown in Figure 7 and 8, while achieving slightly higher utilization compared to Yarn.

V. Conclusion

Cloud computing is a model that harnesses massive resource capacity of data centers to support Internet services and applications in a scalable, flexible, reliable and cost-efficient manner. However, despite its recent success, devising efficient workload management schemes for cloud data centers still remains a major challenge, as it requires carefully taking into consideration the heterogeneous characteristics of both data centers and workloads. This dissertation tackles three key challenges pertaining to resource management in cloud computing environments, namely, dynamic service placement in geographically distributed data centers, heterogeneity-aware dynamic capacity provisioning and fine-grained resource-aware MapReduce scheduling. Through experiments, we show our solutions can bring significant benefits in terms of improved application performance and reduced operational cost.

VI. Final Remark

This dissertation can be downloaded from http://uwspace.uwaterloo.ca/bitstream/10012/7992/1/ZhangQi.pdf. The work conducted during the course of the dissertation research has been published in [4], [23], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43].

References

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, 2008.

[2] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, “Towards characterizing cloud backend workloads: insights from Google compute clusters,” *SIGMETRICS Perform. Eval. Rev.*, vol. 37, March 2010.

[3] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, “Modeling and synthesizing task placement constraints in google compute clusters,” in *Proceedings of the* 2^{nd} *ACM Symposium on Cloud Computing (SOCC)*, 2011.

[4] Q. Zhang, J. Hellerstein, and R. Boutaba, “Characterizing task usage shapes in googles compute clusters,” in *ACM Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2011.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Towards understanding heterogeneous clouds at scale: Google trace analysis,” Tech. Rep. ISTC-CC-TR-12-101, April 2012.

[6] G. Lee, B.-G. Chun, and R. H. Katz, “Heterogeneity-aware resource allocation and scheduling in the cloud,” in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, vol. 11, 2011.

[7] L. Rao, X. Liu, and W. Liu, “Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment,” in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[8] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive Internet services,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, “Starfish: A self-tuning system for big data analytics,” in *Proc. of the 5th Biennial Conf. on Innovative Data Systems Research (CIDR11), Asilomar, California, USA*, 2011.

[10] “Technology research - Gartner Inc,” www.gartner.com.

[11] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, “A taxonomy and survey of energy-efficient data centers and cloud computing systems,” *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[12] “Google cluster-usage traces: format + schema,” http://googleclusterdata.googlecode.com/files/Google %20cluster-usage%20traces%20-%20format%20%2B %20schema%20%282011.10.27%20external%29.pdf.

[13] M. Mao and M. Humphrey, “A performance study on the vm startup time in the cloud,” in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 423–430.

[14] L. Qiu, V. Padmandabhan, and V. Geoffrey, “On the placement of web server replicas,” *IEEE INFOCOM*, 2001.

[15] C. Vicari, C. Petrioli, and F. Presti, “Dynamic replica placement and traffic redirection in content delivery networks,” *Proc. of MASCOTS*, 2007.

[16] M. Karlsson and C. Karamanolis, “Choosing replica placement heuristics for wide-area systems,” in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. IEEE, 2004, pp. 350–359.

[17] J. Kingman, “The single server queue in heavy traffic,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 57, no. 04. Cambridge Univ Press, 1961, pp. 902–904.

[18] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge Univ Pr, 2004.

[19] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, “Measuring isp topologies with rocketfuel,” *IEEE/ACM Transactions on Networking (TON)*, 2009.

[20] “World cup 1998 web site requests,” http://ita.ee.lbl.gov/html /contrib/WorldCup.html.

[21] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, “Heterogeneity and dynamicity of clouds at scale: Google trace analysis,” in *ACM Symposium on Cloud Comp.*, 2012.

[22] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, “Managing server energy and operational costs in hosting centers,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005.

[23] Q. Zhang, M. F. Zhani, Q. Zhu, S. Zhang, R. Boutaba, and J. L. Hellerstein, “Dynamic energy-aware capacity provisioning for cloud computing environments,” in *Proceedings of the IEEE/ACM International Conference on Autonomic Computing (ICAC)*, 2012.

[24] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, and U. Wieder, “Validating heuristics for virtual machines consolidation,” *Microsoft Research, MSR-TR-2011-9*, 2011.

[25] “Eucalyptus community,” http://open.eucalyptus.com/.

[26] “Google workload arrival traces,” http://rboutaba.cs.uwaterloo.ca/Cluster-statistics-OneMonth.zip.

[27] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguadé, “Resource-aware adaptive scheduling for mapreduce clusters,” *Middleware 2011*, pp. 187–207, 2011.

[28] “The next generation of apache hadoop mapreduce.”

[29] A. Verma, L. Cherkasova, and R. H. Campbell, “Aria: automatic resource inference and allocation for mapreduce environments,” in *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 2011, pp. 235–244.

[30] A. Verma, L. Cherkasova, and R. Campbell, “Resource provisioning framework for mapreduce jobs with performance goals,” *Middleware 2011*, pp. 165–186, 2011.

[31] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, “Dominant resource fairness: fair allocation of multiple resource types,” in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. USENIX Association, 2011, pp. 24–24.

[32] PUMA Benchmarks. http://web.ics.purdue.edu/ fahmad/benchmarks/ datasets.htm.

[33] GridMix benchmark for Hadoop clusters. http://hadoop.apache. org/docs/mapreduce/current/gridmix.html.

[34] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[35] L. Cheng, Q. Zhang, and R. Boutaba, “Mitigating the negative impact of preemption on heterogeneous mapreduce workloads,” in *7th International Conference on Network and Service Management (CNSM), 2011*. IEEE, 2011.

[36] Q. Zhang, Q. Zhu, and R. Boutaba, “Dynamic resource allocation for spot markets in cloud computing environments,” in *IEEE International Conference on Utility and Cloud Computing (UCC)*, 2011.

[37] R. Boutaba, L. Cheng, and Q. Zhang, “On cloud computational models and the heterogeneity challenge,” *Journal of Internet Services and Applications*, 2012.

[38] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, “Data center network virtualization: A survey,” *IEEE Communications Survey and Tutorials*, 2012.

[39] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba, “Dynamic service placement in geographically distributed clouds,” in *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[40] ——, “Dynamic service placement in geographically distributed clouds,” *IEEE Journal on Selected Areas in Communications (JSAC)*, 2013.

[41] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba, “Vdc planner: Dynamic migration-aware virtual data center embedding for clouds,” *Proceedings of Integrated Network Management (IM)*, 2013.

[42] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, “Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud,” in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013.

[43] ——, “Dynamic heterogeneity-aware resource provisioning in the cloud,” *IEEE Transactions on Cloud Computing (TCC)*, 2014, to appear.