Cost-Minimizing Preemptive Scheduling of MapReduce Workloads on Hybrid Clouds

Xuanjia Qiu*, Wai Leong Yeow†, Chuan Wu*, Francis C.M. Lau*

*Department of Computer Science, The University of Hong Kong, Hong Kong, {xjqiu, cwu, fcmlau}@cs.hku.hk
†Networking Protocols Department, Institute for INFOCOMM Research (I^2R), Singapore, {wlyeow@ieee.org}

***Abstract*—MapReduce has become the dominant programming model for processing massive amounts of data on cloud platforms. More and more enterprises are now utilizing hybrid clouds, consisting of private infrastructure owned by themselves and public clouds such as Amazon EC2, to process their spiky MapReduce workloads, which fully utilize their own on-premise resources while outsourcing the tasks only when needed. With disparate workloads of different MapReduce tasks, an efficient scheduling mechanism is in need to enable efficient utilization of the on-premise resources and to minimize the task outsourcing cost, while meeting the task completion time requirements as well. In this paper, a fine-grained model is described to characterize the scheduling of heterogeneous MapReduce workloads, and an online algorithm is proposed for joint task admission control into the private cloud, task outsourcing to the public cloud, and VM allocation to execute the admitted tasks on the private cloud, such that the time-averaged task outsourcing cost is minimized over the long run. The online algorithm features preemptive scheduling of the tasks, where a task executed partially on the on-premise infrastructure can be paused and scheduled to run later. It also achieves desirable properties such as meeting a pre-set task admission ratio and bounding the worst-case task completion time, as proven by our rigorous theoretical analysis.**

I. Introduction

The MapReduce framework [1] and its open source versions such as Hadoop [2] have become the dominant programming models for data-intensive and computation-intensive applications in cloud data centers. In the MapReduce framework, a *job* may spawn many small Map and Reduce *tasks* that can be executed concurrently on multiple virtual machines, achieving significant fault tolerance and job completion time reduction. With the rapid increase of the number of jobs that are coded based on the MapReduce framework, efficient schedulers for such workloads on cloud computing resources are becoming increasingly important.

Although there have been a number of models and solutions for scheduling MapReduce workloads on cloud platforms [3][4], we point out the following important aspects that are in lack of attention by the existing work, as the motivation for our research:

(1) *Temporally spiky workloads:* Arrival of MapReduce jobs is not only non-uniform but even spiky. Instead of provisioning resources according to the peak-level workload, it is common for an enterprise to dispatch its workloads across a hybrid cloud, consisting of its own on-premise data center and a public cloud, as long as moderate job outsourcing cost can be achieved and job service delay can be guaranteed [5]. It is not trivial to design a scheduler that can strategically dispatch the workloads to guarantee long-term cost optimality, especially when the *a priori* knowledge about the job arrival pattern is unknown. Hajjat *et al.* [6] focus on one-time application deployment on hybrid clouds, without considering dynamic arrivals of workloads. Zhang *et al.* [5] propose an intelligent algorithm to factor workloads and dynamically determine the service placement across an on-premise server and a cloud, without diving into resource scheduling inside a private cloud.

(2) *Heterogenous workloads:* Different MapReduce jobs may contain different numbers of Map and Reduce tasks, which can span several orders of magnitude [7], with highly heterogenous task running times, which generally follow a long-tailed distribution [8]. Scheduling with the assumption that all jobs are the same or all tasks are the same, is not practical and results in low efficiency. We seek to model the MapReduce workloads at a fine-grained level, *i.e.*, to characterize the lifespans and execution sequence of the tasks in each MapReduce job in details, in order to schedule the jobs efficiently onto the available cloud resources.

(3) *Preemptive job execution:* Preemption refers to pausing a running task, checkpointing its status and resuming it later on, making resource available for another task to run first [9]. As a task may span multiple time slots, preemption is a necessary mechanism to guarantee that more urgent jobs, *e.g.*, production jobs, are not starved, while also allowing the cloud to be used for less urgent jobs when available, *e.g.*, experimental and research jobs [7]. On the other hand, different amounts of workloads may remain after a task is preempted, which adds complexity to the optimization model for task scheduling. The Dynamic Priority Scheduler [10] uses preemption to achieve fairness among users, without guaranteeing the completion time of each job. Maguluri *et al.* [11] discuss preemptive algorithms for job scheduling to achieve optimal throughput in cloud computing clusters, without theoretically addressing the job service delays.

(4) *Quality-of-service (QoS) guarantee:* Two types of QoS are important for MapReduce task scheduling over a hybrid cloud. One is the admission rate of the workloads, defined as the ratio between the amount of workloads admitted into the system and the amount of workloads users submit. The other is the maximum allowed completion time of each job (referred to as the *maximum tolerable job completion time*), defined to be the duration from the time when the job is admitted into the

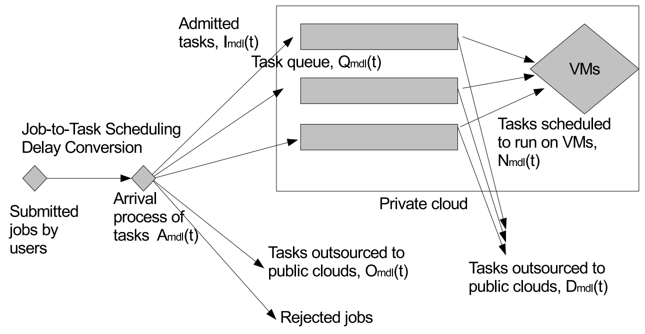
Admitted tasks, $I_{mdl}(t)$
Task queue, $Q_{mdl}(t)$
VMs
Job-to-Task Scheduling Delay Conversion
Tasks scheduled to run on VMs, $N_{mdl}(t)$
Private cloud
Submitted jobs by users
Arrival process of tasks $A_{mdl}(t)$
Tasks outsourced to public clouds, $O_{mdl}(t)$
Tasks outsourced to public clouds, $D_{mdl}(t)$
Rejected jobs

Fig. 1. System Model.

system to the time when it is completed in the private cloud or outsourced to the public cloud. For example, production jobs are typically associated with a tight completion time, while non-production jobs can tolerate more scheduling delay. The scheduler needs to strategically schedule the incoming jobs, in order to satisfy a pre-set workload admission ratio, while completing every admitted job by its completion deadline.

To the best of our knowledge, there are no existing studies that address all the above four aspects in their MapReduce scheduler design. Our contributions in this paper are highlighted as follows:

- ▷ We describe a detailed model to characterize Map and Reduce tasks with different workloads spanning one or more time slots, and the execution sequence of the tasks in a MapReduce job.
- ▷ We build an optimization framework for joint task admission control into the private cloud, task outsourcing to the public cloud, and VM allocation to execute the admitted tasks on the private cloud, such that the time-averaged outsourcing cost is minimized over the long run.
- ▷ We design an online algorithm based on the Lyapunov optimization framework, which features efficient task preemption, and is proven to meet a pre-set job admission ratio and bound the worst-case task completion times, based on our rigorous theoretical analysis.

In the remainder of this paper, we detail the system model and problem formulation in Sec. II, design algorithms for job-to-task arrival process conversion and online scheduling in Sec. III, rigorously prove the guarantee of worst-case task completion time and cost optimality of the algorithms in Sec. IV, and conclude the paper in Sec. V.

II. System Model and Problem Formulation

We consider a time-slotted system. In each time slot, users in the enterprise submit a number of MapReduce jobs to be processed by the hybrid cloud. Description of each job contains Map/Reduce tasks in the job and expected completion time for the job, to be detailed in Sec. II-A. The system will first convert the completion time requirement of each MapReduce job to the completion time requirements of Map and Reduce tasks contained in the job, and then makes

TABLE I
Important Notation

D'	Max. tolerable completion time for a job
J	Set of Map tasks for a job
K	Set of Reduce tasks for a job
T^m_j	Required running time for Map task j
T^r_k	Required running time for Reduce task k
$e(j,k)$	Binary variable indicating whether Map task j depends on Reduce task k $(=1)$ or not $(=0)$
D^m_j	Max. tolerable completion time for Map task j
D^r_k	Max. tolerable completion time for Reduce task k
$C(t)$	Outsourcing cost at t
M	Set of all VM types
D	Set of all max. tolerable completion times
L	Set of all possible task workloads, measured by the number of time slots required to complete a task
$A_{mdl}(t)$	No. of tasks of type (m,d,l) submitted by users, at t
A_{max}	Upper bound of $A_{mdl}(t), \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}$
$I_{mdl}(t)$	No. of tasks of type (m,d,l) admitted to Q_{mdl}, at t
$O_{mdl}(t)$	No. of tasks of type (m,d,l) outsourced to the public cloud directly without being admitted to task queues, at t
$Q_{mdl}(t)$	No. of unit workload in task queue for type-(m,d,l) tasks, at t
$D_{mdl}(t)$	No. of unit workload outsourced to the public cloud from queue Q_{mdl}, at t
$N_{mdl}(t)$	No. of type-(m,d,l) tasks scheduled to run on VMs of the private cloud, at t.
$N_{mdl}(t^-)$	No. of leftover type-(m,d,l) tasks, at t.
$K_{mdl}(t)$	Backlog of virtual queue for guaranteeing the admission ratio of type-(m,d,l) tasks, at t
$Z_{mdl}(t)$	Backlog of virtual queue for guaranteeing worst-case completion time of type-(m,d,l) tasks, at t
α	Pre-set admission ratio of tasks

admission control decisions, *i.e.*, either enqueueing the tasks into appropriate task queues to be potentially handled by the on-premise data center, dispatching tasks to the public cloud, or rejecting the job. For tasks in the task queues, the system further decides whether to schedule their execution on VMs in the on-premise data center, or outsource them to the public cloud, in order to minimize the outsourcing cost while guaranteeing all the QoS requirements. An illustration of the system is given in Fig. 1.

We address the following two problems: (1) Given the scheduling is carried out at the task level rather than the job level, how should the input job completion deadline requirement be converted to individual tasks' completion deadlines for efficient scheduling? (2) How should the system schedule the tasks strategically to achieve minimized outsourcing cost while satisfying all the QoS requirements? We model these problems as follows. Key notation is summarized in Table I for ease of reference.

A. Job-to-task Completion Deadline Conversion

The system receives job submissions from users, in the sequence of H_i, $i = 1, 2, 3, ...$ Since we are going to restrict the discussion in this sub-section within the scope of one job, we drop the subscript that distinguishes the jobs, for simplicity of the notation. Each job can be represented by a 6-tuple $< J, K, D', T^m, T^r, E >$, which is defined as follows:

- J is the set of Map tasks. K is the set of Reduce tasks.

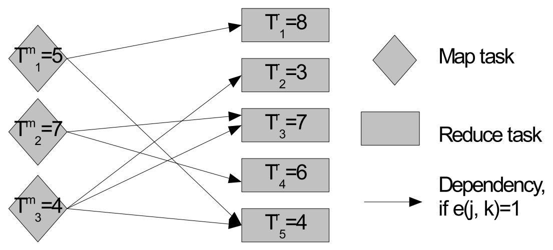

Fig. 2. An illustration of the MapReduce job model.

- D' is the user-specified maximum tolerable completion time for the job.
- $T_j^m, \forall j \in J$, and $T_k^r, \forall k \in K$, are sets of workloads of all the Map tasks and all the Reduce tasks, respectively. The workload of a task is characterized by the number of required time slots to complete the task on a VM of the required type.
- E is a set of binary variables that describe the dependency between any Map task $T_j^m, \forall j \in J$, and any Reduce task $T_k^r, \forall k \in K$. The dependency relationship is determined by the partition functions that assign keys of intermediate pairs to the Reduce nodes. Reduce task T_k^r is dependent on Map task T_j^m if T_j^m generates an intermediate key-value pair whose key is in the specified key range of T_k^r. In that situation, T_k^r cannot be started until T_j^m has been completed, and we have $e(j,k) = 1$; otherwise, $e(j,k) = 0$.

This general job model, as illustrated in Fig. 2, applies to any categories of applications running on the MapReduce platforms. The parameters in a job descriptor can be specified by the user who submits the job, or obtained by profiling [12].

Our optimal scheduling algorithm works at the task level instead of the job level, in order to achieve more efficient resource utilization. To this end, the system needs to convert the job submission process into task submission processes, as well as decides the maximum tolerable completion time requirements for tasks in a MapReduce job based on that of the job. Let $D_j^m, \forall j \in J$, and $D_k^r, \forall k \in K$, denote the maximum tolerable completion time for each Map task and for each Reduce task, respectively. The set of feasible completion times for the tasks should satisfy the following constraints:

$$\begin{aligned} &\max_{j\in J, k\in K} e(j,k)(D_j^m + D_k^r) \le D', \qquad (1)\\ &T_j^m \le D_j^m, \forall j \in J,\\ &T_k^r \le D_k^r, \forall k \in K. \end{aligned}$$

(1) means that the sum of the tolerable completion times of any Map task and any Reduce task with dependency in between should not exceed the tolerable completion time of the job.

We will detail the algorithm that derives the set of feasible task completion times and task submission times in Sec. III-A. Each task can then be characterized by a 3-tuple (m, d, l): $m \in \mathcal{M}$ is the type of VMs the task requires to use; $d \in \mathcal{D}$ is the maximum tolerable completion time for the task, *i.e.*, the duration from the time slot when the task is enqueued to the time slot when the task is completed; $l \in \mathcal{L}$ is the required running time of the task. Here, $\mathcal{M}$, $\mathcal{D}$ and $\mathcal{L}$ are the sets of all possible VM types, all possible tolerable completion times, and all possible running times, respectively. The latter two are in the units of time slots in our system. The tasks submitted in the same time slot t and with the same 3-tuple (m, d, l) produce a task arrival process with arrival rate $A_{mdl}(t)$ in t. We suppose the arrival rates are upper bounded by A_{max}.

B. Queueing Model and Control Decisions

Tasks that are submitted to the system are queued before they are scheduled to run. Especially, tasks with the same (m, d, l) are enqueued in queue Q_{mdl}, $\forall m \in \mathcal{M}$, $\forall d \in \mathcal{D}$, $\forall l \in \mathcal{L}$. Each element in a task queue represents one unit of task workload, *i.e.*, the workload corresponding to one-time-slot execution of the task on a type-m virtual machine. When a task is enqueued into Q_{mdl}, l elements are appended to the tail of the queue; when a task is scheduled to run on its required VM for one time slot, an element belonging to that task departs from the queue; when a decision has been made for outsourcing a task in Q_{mdl}, all the remaining elements associated with that task are removed from the queue. We denote the backlog of task queue Q_{mdl} at t as $Q_{mdl}(t)$, which indicates the number of elements (*i.e.*, the number of unit workload) in the queue.

We formulate decision variables in our scheduling framework as follows:

1) *Task admission or outsourcing upon arrival:* In time slot t, $I_{mdl}(t)$ tasks among the $A_{mdl}(t)$ arrived tasks are admitted into queue Q_{mdl}, by appending $lI_{mdl}(t)$ elements at the end of the queue. $O_{mdl}(t)$ newly arrived tasks are outsourced to run in the public cloud, and the rest $A_{mdl}(t) - I_{mdl}(t) - O_{mdl}(t)$ tasks are rejected.

2) *Task outsourcing after admission:* At time slot t, $D_{mdl}(t)$ units of workloads at the head of the queue Q_{mdl} are outsourced to the public cloud. These $D_{mdl}(t)$ units of workloads may span a number of tasks at the head of the queue, which are partially completed or not scheduled for running at all yet. For outsourced workload, on-demand VMs from the public cloud are rent to run them immediately without occurring any further delay.

3) *Task scheduling on the private cloud:* At the beginning of time slot t, $N_{mdl}(t)$ VMs of type m are scheduled to run tasks of type (m, d, l). $N_{mdl}(t^-)$ is the number of left-over type-(m, d, l) tasks observed at the start of time slot t, *i.e.*, tasks that were scheduled to run on type-m VMs before time slot t, and have not yet been completed until the end of time slot $t-1$. When $t = 1$, we set $N_{mdl}(t^-) = 0$.

If $N_{mdl}(t) \ge N_{mdl}(t^-)$, it denotes that all $N_{mdl}(t^-)$ left-over tasks continue running on their occupied VMs in time slot t, and $N_{mdl}(t) - N_{mdl}(t^-)$ type-m VMs are newly allocated to run the next $N_{mdl}(t) - N_{mdl}(t^-)$ waiting tasks in queue Q_{mdl}. If $N_{mdl}(t) < N_{mdl}(t^-)$, only $N_{mdl}(t)$ among the $N_{mdl}(t^-)$ left-over tasks can continue running, and the other $N_{mdl}(t^-) - N_{mdl}(t)$ left-over tasks are preempted. We maintain the *first-come-first-served (FCFS)* principle at the task level, *i.e.*, between any two running tasks, the one arrived earlier at the queue has higher priority to continue running.

In time slot t, one unit of workload for each of the $N_{mdl}(t)$ head-of-the-queue tasks is removed from queue Q_{mdl}, *i.e.*, $N_{mdl}(t)$ elements are removed from Q_{mdl}. We note that since each task is represented by multiple adjacent elements in the queue, the removed elements belong to different tasks, and may not be neighboring elements. Essentially, we maintain the FCFS principle at the task level only, while it is not necessarily FCFS at the level of elements. An illustration of this queueing model is given in Fig. 3. Devising a scheduling algorithm with guarantee of worst-case completion time under such a queueing model is challenging.

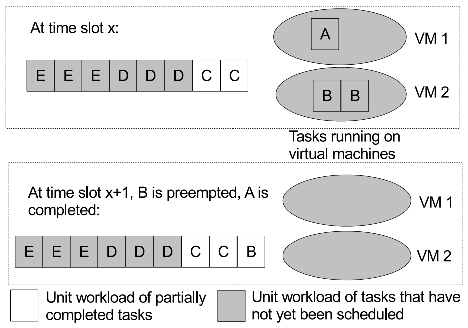
At time slot x:
E E E D D D C C
A
VM 1
B B
VM 2
Tasks running on virtual machines
At time slot x+1, B is preempted, A is completed:
VM 1
E E E D D D C C B
VM 2
Unit workload of partially completed tasks
Unit workload of tasks that have not yet been scheduled

Fig. 3. An illustration of the task queueing model: A—E represent task indices.

The update of queue Q_{mdl} is as follows:

$$Q_{mdl}(t+1) = \max[Q_{mdl}(t) - N_{mdl}(t) \\ -D_{mdl}(t), 0] + l\, I_{mdl}(t), \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}. \quad (2)$$

We note that $Q_{mdl}(t)$ and $D_{mdl}(t)$ are in the number of unit task workload, while the other quantities $A_{mdl}(t)$, $I_{mdl}(t)$, $O_{mdl}(t)$ and $N_{mdl}(t)$ are in the number of tasks.

C. Quality-of-Service Constraints

The first QoS metric under consideration is the task admission ratio, which is the ratio between the number of processed tasks (including outsourced tasks and scheduled tasks) and the number of submitted tasks. The task admission ratio in the system should be no smaller than a given threshold α (set by the enterprise), *i.e.*,

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T}(I_{mdl}(t) + O_{mdl}(t))}{\sum_{t=0}^{T} A_{mdl}(t)} \geq \alpha. \quad (3)$$

The other QoS requirement is to guarantee the worst-case completion times of tasks admitted into the queues, *i.e.*, the tasks are processed within a specified maximum tolerable completion time. Instead of modeling this constraint explicitly, we treat it as a property of our proposed scheduling algorithm, which will be proved by rigorous analysis in Sec. IV.

D. Outsourcing Cost Minimization

Since the on-premise infrastructure is constructed and always maintained by the enterprise regardless of the workloads, our scheduling algorithm focuses on the minimization of the task outsourcing cost, which is the payment to the public cloud, over the long run. Let $H(m)$ denote the charge by the public cloud for processing a unit of task workload on a type-m VM. The overall outsourcing cost in time slot t is

$$C(t) = \sum_{m\in M}\sum_{d\in D}\sum_{l\in L}(l\, O_{mdl}(t) + D_{mdl}(t))H(m). \quad (4)$$

Our objective is to minimize the time-averaged outsourcing cost in the long term as follows:

$$\min \quad \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} C(t) \quad (5)$$

subject to:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T}(I_{mdl}(t) + O_{mdl}(t))}{\sum_{t=0}^{T} A_{mdl}(t)} \geq \alpha, \\ \forall m \in M, d \in D, l \in L, \quad (6)$$

$$\sum_{d\in D}\sum_{l\in L} N_{mdl}(t) \leq N_m^{tot}, \forall m \in M, \forall t \in [0,T], \quad (7)$$

$$I_{mdl}(t) + O_{mdl}(t) \leq A_{mdl}(t), \\ \forall m \in M, d \in D, l \in L, t \in [0,T], \quad (8)$$

$$D_{mdl}(t) \leq D_{mdl}^{max}, \forall m \in M, d \in D, l \in L, t \in [0,T], \quad (9)$$

$$D_{mdl}(t) \in Z^{+} \cup 0, \forall m \in M, d \in D, l \in L, t \in [0,T], \quad (10)$$

$$I_{mdl}(t) \in Z^{+} \cup 0, \forall m \in M, d \in D, l \in L, t \in [0,T], \quad (11)$$

$$O_{mdl}(t) \in Z^{+} \cup 0, \forall m \in M, d \in D, l \in L, t \in [0,T], \quad (12)$$

$$N_{mdl}(t) \in Z^{+} \cup 0, \forall m \in M, d \in D, l \in L, t \in [0,T]. \quad (13)$$

Here, N_m^{tot} is the total number of type-m VMs , and D_{mdl}^{max} is maximum units of outsourced workloads. (6) is to specify the admission ratio requirement. (7) shows that the number of scheduled tasks for each type of VM should not exceed the total number of VMs of that type in the private cloud.

III. The Online Scheduling Algorithm

We next detail the algorithm for converting job submission sequence into task submission sequences, and then design the dynamic scheduling algorithm based on the Lyapunov optimization framework.

A. Converting Job Submission Sequence to Task Submission Processes

For each submitted job, we need to decide the submission time and maximum tolerable completion time for each of the Map and Reduce tasks contained in the job, in order to generate the task arrival process $A_{mdl}(t)$.

Although there exist many feasible solutions satisfying the constraints in (1), we seek to derive a simple and efficient solution method, which provably achieves performance optimality together with our scheduling algorithm, as follows.

For each Reduce task $k \in K$, find the Map task that has the maximum required running time among all the Map tasks that task k depends on, *i.e.*, select

$$j' = \arg\max_{j\in\mathcal{J}} \{e(j,k)T_j^m\}. \quad (14)$$

Hence, the slack time to allow timely scheduling of Map task j' and Reduce task k is $D' - T_{j'}^m - T_k^r$. We allocate one half of this slack time to the scheduling of Reduce task k, and set its maximum tolerable completion time as

$$D_k^r = T_k^r + \frac{D' - T_{j'}^m - T_k^r}{2}. \quad (15)$$

Suppose this MapReduce job is submitted at t_0. The submission time of Reduce task k is set to $\lceil t_0 + D' - D_k^r \rceil$.

After deciding the submission time and maximum tolerable completion time of each of the Reduce tasks using the above method, we then decide the submission time and tolerable time of each of the Map tasks. For Map task $j \in J$, the maximum tolerable completion time is computed as the job's tolerable completion time minus the largest tolerable completion time among those of all the Reduce tasks depending on Map task j, *i.e.*,

$$D_j^m = D' - D_{k'}^r, \text{where } k' = \arg\max_{k \in \mathcal{K}} e(j,k) D_k^r. \quad (16)$$

The submission time of each Map task is set to the same as that of the job, *i.e.*, t_0.

The above method guarantees that a Reduce task will not be enqueued until all of its dependent Map tasks have been completed. As this conversion method is deterministic, when the arrival process of the jobs is ergodic, the task arrival processes into the system are also ergodic.

B. Solving the Online Scheduling Problem

We apply the Lyapunov Optimization framework [13][14] to design an online algorithm to solve problem (5).

To guarantee the satisfaction of admission ratio constraints in (6), we associate a virtual queue K_{mdl} with each task queue $Q_{mdl} (\forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L})$, with initial backlog $K_{mdl}(0) = 0$ and queue update

$$K_{mdl}(t+1) = \max[K_{mdl}(t) + \alpha l \, A_{mdl}(t) - l \, I_{mdl}(t) - l \, O_{mdl}(t), 0]. \quad (17)$$

To bound the worst-case completion time for each task, we apply the *ϵ-persistent service* technique [14] to build a virtual queue Z_{mdl} associated with each task queue Q_{mdl}, $\forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}$, with initial backlog $Z_{mdl}(0) = 0$:

$$\begin{aligned} Z_{mdl}(t+1) = \max[&Z_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}}(\epsilon_{mdl} - N_{mdl}(t) \\ &- D_{mdl}(t)) - 1_{\{Q_{mdl}(t)=0\}} N_m^{tot}, 0]. \end{aligned} \quad (18)$$

Here ϵ_{mdl} is a pre-defined constant whose value is related to the desired worst-case completion time d of tasks in queue Q_{mdl}. We will reveal their relationship in Sec. IV-B.

Let $\Theta(t) = [\mathbf{Q}(t), \mathbf{Z}(t), \mathbf{K}(t)]$ denote a vector consisting of all queues in our system, where $\mathbf{Q}(t) = \{Q_{mdl}(t), \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}\}$, $\mathbf{Z}_{mdl}(t) = \{Z_{mdl}(t), \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}\}$, $\mathbf{K}_{mdl}(t) = \{K_{mdl}(t), \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}\}$. Define the Lyapunov function as

$$L(\Theta(t)) = \frac{1}{2} \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} [Q_{mdl}(t)^2 + Z_{mdl}(t)^2 + K_{mdl}(t)^2].$$

The one-slot conditional drift-plus-penalty is

$$\begin{aligned} &\Delta(\Theta(t)) + VC(t) \\ &\leq B_1 + Z_{mdl}(t)(1_{\{Q_{mdl}(t)>0\}}\epsilon_{mdl} - 1_{\{Q_{mdl}(t)=0\}} N_m^{tot}) \\ &\quad + \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} l \, I_{mdl}(t)[Q_{mdl}(t) - K_{mdl}(t)] \\ &\quad + \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} l \, O_{mdl}(t)[VH(m) - K_{mdl}(t)] \\ &\quad + \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} [D_{mdl}(t)(VH(m) - Q_{mdl}(t) \\ &\quad - 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t))] - \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} N_{mdl}(t)(Q_{mdl}(t) \\ &\quad + 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t)), \end{aligned} \quad (19)$$

where V is a controlling constant the purpose of which will be detailed in Sec. IV, and B_1 is a constant as follows:

$$\begin{aligned} B_1 = &\sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} (D_{mdl}^{max})^2 + 2 \sum_{m \in \mathcal{M}} N_m^{tot} (\sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} D_{mdl}^{max}) \\ &+ |\mathcal{D}||\mathcal{L}| \sum_{m \in \mathcal{M}} (N_m^{tot})^2 + \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} \max\{(1-\alpha)^2, \alpha^2\}(A_{max})^2 \\ &+ \sum_{m \in \mathcal{M}} \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} \max\{\epsilon_{mdl}^2, (N_m^{tot} + D_{mdl}^{max})^2\}. \end{aligned} \quad (20)$$

Based on the Lyapunov optimization framework [13], we derive a dynamic algorithm that observes queues in $\Theta(t)$ in each time slot t and makes control decisions on $I_{mdl}(t)$, $O_{mdl}(t)$, $D_{mdl}(t)$, $N_{mdl}(t)$, $\forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}$, that minimize the RHS (Right-Hand-Side) of (19), such that an upper bound for the time-averaged outsourcing cost is minimized. Except the constant terms, RHS of (19) can be decomposed into three parts, which we seek to minimize respectively in each time slot as follows.

1) *Admission control*: We solve the following optimization problem to derive $I_{mdl}(t)$ and $O_{mdl}(t)$, for each $m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}$:

$$\min I_{mdl}(t)[Q_{mdl}(t) - K_{mdl}(t)] + O_{mdl}(t)[VH(m) - K_{mdl}(t)]$$

Subject to: constraints (8)(11)(12). (21)

(21) is a linear optimization problem. We observe the coefficients of the two variables $I_{mdl}(t)$ and $O_{mdl}(t)$, *i.e.*, $Q_{mdl}(t) - K_{mdl}(t)$ and $VH(m) - K_{mdl}(t)$, in (21), and derive the following solutions:

Case (1): If $Q_{mdl}(t) - K_{mdl}(t) \geq 0$ and $VH(m) - K_{mdl}(t) < 0$, the solution is $I_{mdl}^*(t) = 0$ and $O_{mdl}^*(t) = A_{mdl}(t)$;

Case (2): If $Q_{mdl}(t) - K_{mdl}(t) < 0$ and $VH(m) - K_{mdl}(t) \geq 0$, the solution is $I_{mdl}^*(t) = A_{mdl}(t)$ and $O_{mdl}^*(t) = 0$;

Case (3): If $Q_{mdl}(t) - K_{mdl}(t) \geq 0$ and $VH(m) - K_{mdl}(t) \geq 0$, the solution is $I_{mdl}^*(t) = 0$ and $O_{mdl}^*(t) = 0$;

Case (4): When $Q_{mdl}(t) - K_{mdl}(t) < 0$ and $VH(m) - K_{mdl}(t) < 0$, if $Q_{mdl}(t) - K_{mdl}(t) < VH(m) - K_{mdl}(t)$, *i.e.*, $Q_{mdl}(t) < VH(m)$, the solution is $I_{mdl}^*(t) = A_{mdl}(t)$ and $O_{mdl}^*(t) = 0$; otherwise, the solution is $I_{mdl}^*(t) = 0$ and $O_{mdl}^*(t) = A_{mdl}(t)$.

2) *In-queue task outsourcing*: We solve the following linear minimization problem to derive $D_{mdl}(t)$, for each $m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{L}$:

$$\min D_{mdl}(t)(VH(m) - Q_{mdl}(t) - 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t)) \quad (22)$$

Subject to: constraints (9)(10).

The solution to problem (22) is

$$D_{mdl}^*(t) = \begin{cases} 0 & \text{if } VH(m) \geq Q_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t) \\ D_{mdl}^{max} & \text{otherwise} \end{cases}$$

In our system, if part of the workloads of a task in a task queue is to be outsourced, all the remaining units of workloads of the task should be outsourced together. Hence, we set $D_{mdl}(t)$ to be $D_{mdl}^*(t)$ plus the smallest number of elements that can cover complete tasks. The increment is smaller than l, *i.e.*, $D_{mdl}(t) < D_{mdl}^*(t) + l$.

3) *VM allocation for in-queue tasks*: We derive $N_{mdl}(t)$, for each $m \in \mathcal{M}$, by solving the following maximization problem:

$$\max \sum_{d \in \mathcal{D}} \sum_{l \in \mathcal{L}} N_{mdl}(t)(Q_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t)) \quad (23)$$

Subject to: constraints (13)(7).

As $Q_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t)$ is always non-negative, to maximize (23), we simply need to find

$$\{d', l'\} = \text{argmax}_{d \in \mathcal{D}, l \in \mathcal{L}} Q_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}} Z_{mdl}(t)), \quad (24)$$

and then set $N_{md'l'}^* = N_m^{tot}$ and $N_{mdl}^* = 0, \forall d \neq d', l \neq l'$.

The above dynamic algorithm can be implemented by a controller module in the enterprise as follows: At the beginning of each time slot t, the controller receives submitted jobs and converts them into task arrivals at the current and future time slots. Then the controller solves the three optimization problems (21)(22)(23) to decide the optimal values of control variables on task admission, outsourcing and VM allocation, and schedules the tasks accordingly. Especially for in-queue task outsourcing and VM allocation, the controller first outsources tasks that are covered by D_{mdl} elements at the head of each queue Q_{mdl}, and then schedules the next $N_{mdl}(t)$ in-queue tasks to run on VMs. At the end of the time slot, the controller updates the status of all queues.

IV. Performance Analysis

The proof of all lemmas and theorems in this section can be found in our technical report [15].

A. Strong Stability of Queues

Lemma 1: (Strong Stability of Queues) Our algorithm guarantees in all time slots,

$$K_{mdl}(t) \leq K_{mdl}^{max} = VH(m) + \alpha l\, A_{max}, \tag{25}$$

$$Q_{mdl}(t) \leq Q_{mdl}^{max} = VH(m) + (1+\alpha)l\, A_{max}, \tag{26}$$

$$Z_{mdl}(t) \leq Z_{mdl}^{max} = VH(m) + \epsilon_{mdl}. \tag{27}$$

As Lemma 1 shows the strong stability of queue (17), inequality (6) is always satisfied, *i.e.*, the pre-set admission ratio is guaranteed.

B. Guarantee of Worst-Case Task Completion Time

Theorem 1: (Guarantee of Worst-Case Completion Time) The worst-case completion times of all tasks admitted into queue Q_{mdl} are upper bounded by the constant

$$U_{mdl} = \lceil \frac{(1+l)Q_{mdl}^{max} + Z_{mdl}^{max}}{\epsilon_{mdl}} \rceil, \tag{28}$$

where Q_{mdl}^{max} and Z_{mdl}^{max} are upper bounds of $Q_{mdl}(t)$ and $Z_{mdl}(t)$ defined in (26) and (27).

This shows that we can set ϵ_{mdl} as

$$\epsilon_{mdl} = \frac{(1+l)Q_{mdl}^{max} + Z_{mdl}^{max}}{d},$$

such that the completion times of tasks in Q_{mdl} are no larger than the maximum tolerable completion time of d.

C. Optimality of Time-averaged Outsourcing Cost

For simplification of notation, we use $\overline{X}$ to represent $\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} E(X(t))$.

Lemma 2: (Existence of Optimal Stationary, Randomized Policy): For any ergodic job arrival process, there exists a stationary randomized control policy π that chooses $I_{mdl}(t) \in [0, A_{max}]$, $O_{mdl}(t) \in [0, A_{max}]$, $N_{mdl}(t) \in [0, N_m^{tot}]$, that solve problem (5) augmented with the following constraints:

$$\epsilon_{mdl} \leq \overline{N_{mdl}} + \overline{D_{mdl}}, \forall m \in \mathcal{M}, d \in \mathcal{D}, l \in \mathcal{D}, \tag{29}$$

with optimal time-averaged outsourcing cost $\overline{C}^{\pi}$, only if the new problem is feasible.

Theorem 2: (Optimality of Outsourcing Cost): The time-averaged outsourcing cost achieved by our dynamic algorithm (denoted as $\overline{C}^{*}$ hereafter) is within a constant gap from the cost of any stationary randomized control policy that solve problem (5) augmented with constraint (29), *i.e.*,

$$\overline{C}^{*} \leq \overline{C}^{\pi} + \frac{B_1}{V} + |\mathcal{D}| \sum_{l\in\mathcal{L}} l \sum_{m\in\mathcal{M}} H(m), \tag{30}$$

where B_1 is the constant defined in (20).

The theorem shows that the performance of our dynamic algorithm can approach the optimal cost of the augmented problem within a constant gap, which decreases with the increase of V. Meanwhile, from (26) we see worst-case backlogs grow linearly in V. Therefore, V can be adjusted to achieve a desired tradeoff between the outsourcing cost and the actual completion time.

V. Conclusion

This paper proposes a fine-grained model to characterize the scheduling of heterogeneous MapReduce workloads over a hybrid cloud, and an online algorithm for joint task admission control, task outsourcing and VM allocation. Based on the Lyapunov optimization framework, we show that the online algorithm can achieve close-to-minimal time-averaged task outsourcing cost over the long run, with guarantee of task admission ratio and worst-case task completion time. As on-going work, we are implementing the algorithm in practical systems to further evaluate its performance.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. of OSDI*, Dec. 2004.

[2] *Hadoop*, http://hadoop.apache.org/.

[3] *Fair Scheduler*, http://hadoop.apache.org/mapreduce/docs/r0.21.0/fair_scheduler.html.

[4] *Capacity Scheduler*, http://hadoop.apache.org/mapreduce/docs/r0.21.0/capacity_scheduler.html.

[5] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent Workload Factoring for a Hybrid Cloud Computing Model," in *Proc. of the International Workshop on Cloud Services (IWCS)*, Jun. 2009.

[6] M. Hajjat, X. Sun, Y. E. Sung, D. Maltz, and S. Rao, "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," in *Proc. of IEEE SIGCOMM*, Aug. 2010.

[7] L. Cheng, Q. Zhang, and R. Boutaba, "Mitigating the Negative Impact of Preemption on Heterogeneous MapReduce Workloads," in *Proc. of 7th International Conference on Network and Service Management (CNSM)*, Oct. 2011.

[8] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An Analysis of Traces from a Production MapReduce Cluster," in *Proc. of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010.

[9] T. White, *Hadoop: The Definitive Guide*. Yahoo Press, 2010.

[10] T. Sandholm and K. Lai, "Dynamic Proportional Share Scheduling in Hadoop," in *Proc. of the 15th International Conference on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2010.

[11] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters," in *Proc. of IEEE INFOCOM*, Mar. 2012.

[12] S. Babu, "Towards automatic optimization of MapReduce programs," in *Proc. of the 1st ACM symposium on Cloud computing (SoCC)*, Jun. 2010.

[13] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

[14] ——, "Opportunistic Scheduling with Worst Case Delay Guarantees in Single and Multi-Hop Networks," in *Proc. of IEEE INFOCOM*, Apr. 2011.

[15] X. Qiu, W. L. Yeow, C. Wu, and F. C. M. Lau, "Cost-Minimizing Preemptive Scheduling of MapReduce Workloads on Hybrid Clouds," `http://www.cs.hku.hk/~xjqiu/xjqiu-mrsch.pdf`, The University of Hong Kong, Tech. Rep., Apr. 2013.