Cost-Minimized VM Allocation for MapReduce Jobs on Hybrid clouds

***Abstract*—xxx**

I. Introduction

II. The Task Scheduling Problem

A. System Model

Consider an organization that deals MapReduce jobs submitted from its members. The organization holds a private cloud dealing with most of the workload, and outsources some workload to public clouds when the private cloud is fully occupied. If a job is not accepted by either of the above two means, it is rejected. Tasks admitted into the private clouds are queued, waiting for being scheduled on VMs.

The system model is illustrated in Fig. 1.

The problem we consider is two-folded:

(1) How can we transfer a MapReduce job requirement to the system-required specification?

(2) How can the system schedules the tasks executions with minimized operational cost under the constraint of acceptance ratio?

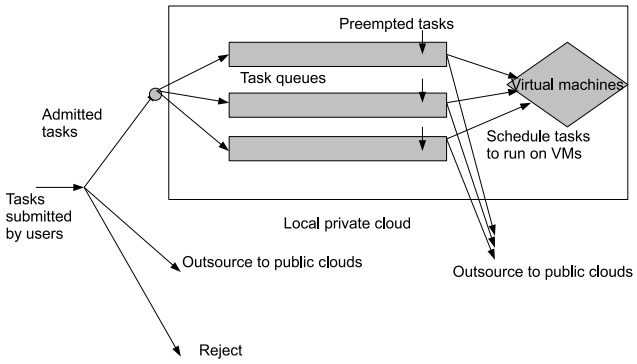

Fig. 1. System Model.

B. The Problem of Allowable Delay Partition

A job of MapReduce has two phases of tasks. One is map phase and the other is reduce phase. There are multiple tasks in each phase. Each *map task* is assigned a portion of the input data file. A map task reads each input record from the input data file and invokes a user specified *map function* with the record as a parameter to produce a list of intermediate key-value pairs. Each *reduce task* is assigned a portion of the key range produced by the map phase. For each distinct key, the *reduce task* fetches the list of all associated intermediate values in the map output, and invokes a user-specified reduce function onto them. The output of all *reduce tasks* is the final result that the user wants.

As reduce tasks need to read intermediate results produced by map tasks, we say a reduce task R is dependent on a map task M if M generates a key-value pair $< K, V >$, and K is in the specified key range of R. If R is dependent on M, R cannot be started until M has finished.

We suppose for each job with the set of map tasks J and the set of reduce tasks K, the user gives

- The allowable delay of the whole job D'.
- Execution duration of each map task, $T_j^m, \forall j \in J$ and each reduce task, $T_k^r, \forall k \in K$.
- If map task j is dependent on reduce task k, then $e(j,k) = 1$; otherwise $e(j,k) = 0$.

We need to give an algorithm to produce allowable delay of each task, i.e., $D_j^r, \forall j \in J$ and $D_k^m, \forall k \in K$.

We can formulate the problem as the following:

$\max \prod_{j \in J}(D_j^m - T_j^m) \prod_{k \in K}(D_k^r - T_k^r)$

Subject to:

$$e(j,k)(D_j^m + D_k^r) \leq D', \forall j \in J, k \in K \quad (1)$$

$$T_j^r \leq D_j^r, \forall j \in J \quad (2)$$

$$T_k^m \leq D_k^m, \forall k \in K \quad (3)$$

The feasibility condition of the problem is as follows:

$$\max_{j \in J, k \in K} e(j,k)(T_j^r + T_k^m) \leq D' \quad (4)$$

(4) means that when we sum up the pair of one map task and one reduce task, where there is a dependency. The largest number of the sums is the bottleneck of the job. If sum of task execution duration at the bottleneck is larger than the total allowable delay, there are now valid scheduling to guarantee the deadline is met.

C. Cost-Minimization Task Scheduling Problem

Suppose at time slot t, $S(t)$ tasks are submitted to the system. Each task k is denoted as a 3-tuple: (m_k, l_k, d_k), where m_k is the type of VM, $m_k \in M$; l_k is the task size (execution duration), $l \in L$; d_k is the allowable delay from the time it is submitted to the system to the time it is finished, $d_k \in D$. Tasks described with same $3 - tuple$ are grouped together, with number $S_{mdl}(t)$, so that $S(t) = \sum_{m \in M} \sum_{d \in D} \sum_{l \in L} S_{mdl}(t)$.

Decision Variables Task queues Q_{mdl} is used to buffer tasks with delay requirement d, VM type m and task size l, $\forall m \in M, d \in D, l \in L$. Each unit of task size is modeled as an element in the task queue. In time slot t, our admission control algorithm will admit $A_{mdl}(t)$ tasks into task queue Q_{mdl}, outsource $O_{mdl}(t)$ tasks to public clouds before admission, and reject the remaining $S_{mdl}(t) - O_{mdl}(t) - A_{mdl}(t)$ tasks. $D_{mdl}(t)$ tasks are outsourced to public clouds after they have been admitted into the queue Q_{mdl}, whose remaining associated elements in the queue at time slot t is $W^{(t)}(D_{mdl}(t))$. $N_{mdl}(t)$ tasks are scheduled to execute on VMs.

The update equation for queue Q_{mdl} is as follows:

$Q_{mdl}(t+1) = \max[Q_{mdl}(t) - N_{mdl}(t) - W^{(t)}(D_{mdl}(t)), 0] + A_{mdl}(t) l, \forall d \in D, m \in M, l \in L, t$

Preemption $N_{mdl}(t^-)$ are the number of left-over tasks, i.e., tasks that are allocated with VMs in time slot $t-1$ and have not been completed.

When $N_{mdl}(t) > N_{mdl}(t^-)$, $N_{mdl}(t) - N_{mdl}(t^-)$ new tasks are allocated with VMs. On the contrast, when $N_{mdl}(t) < N_{mdl}(t^-)$, $N_{mdl}(t^-) - N_{mdl}(t)$ running tasks are preempted.

We notice that the dequeue sequence is non-FIFO. Devising scheduling algorithm with worst-case delay guarantee constitutes one point of contribution of this paper.

Acceptance Ratio Constraint We require that in the long term, the acceptance ratio of all tasks is no smaller than α, *i.e.*,

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T}(A_{mdl}(t) + O_{mdl}(t))}{\sum_{t=0}^{T} S_{mdl}(t)} \geq \alpha \tag{5}$$

Operational cost There are two parts of operational cost: preemption cost and outsourcing cost.

$H(x, y, z)$ is the charge by the public cloud for processing a task with VM type x, allowable delay y, and task size z. Therefore, total cost for outsourcing at time slot t is $\sum_{m\in M}\sum_{d\in D}\sum_{l\in L}(D_{mdl}(t) + O_{mdl}(t))H(m, d, l)$.

The operational cost in time slot t is

$C(t) = (O_{mdl}(t) + D_{mdl}(t))H(m, d, l) + \beta p_m[N_{mdl}(t^-) - N_{mdl}(t)]^+$.

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

$$\min \quad \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T}\sum_{m\in M}\sum_{d\in D}\sum_{l\in L} C(t) \tag{6}$$

subject to:

$$\lim_{T\to\infty} \frac{\sum_{t=0}^{T}(A_{mdl}(t) + O_{mdl}(t))}{\sum_{t=0}^{T} S_{mdl}(t)} \geq \alpha, \quad \forall m \in M, d \in D, l \in L \tag{7}$$

$$\sum_{d\in D}\sum_{l\in L} N_{mdl}(t) \leq N_m^{tot}, \forall m \in M, \forall t \tag{8}$$

$$A_{mdl}(t) + O_{mdl}(t) \leq S_{mdl}(t), \forall m \in M, d \in D, l \in L, t \tag{9}$$

TABLE I
NOTATIONS

$C(t)$	Operational coat, at t
M	Set of all types of VM
D	Set of all allowable delays
L	Set of all task sizes
$Q_{mdl}(t)$	Backlog of queue of task of type (m,d,l)
$S_{mdl}(t)$	Num. of tasks of type (m, d, l) submitted by the users, at t
$A_{mdl}(t)$	Num. of tasks admitted into Q_{mdl}, at t
$O_{mdl}(t)$	Num. of tasks of type (m, d, l) outsourced to public clouds before admission, at t
$D_{mdl}(t)$	Num. of tasks of type (m, d, l) outsourced to public clouds after admission, at t
$N_{mdl}(t)$	Num. of tasks of type (m, d, l) scheduled to run on VMs, at t.
$N_{mdl}(t^-)$	Num. of leftover tasks of type (m, d, l).
$K_{mdl}(t)$	Backlog of virtual queue for guaranteeing acceptance ratio constraint
$Z_{mdl}(t)$	Backlog of virtual queue for guaranteeing worst-case completion time

(7) model the constraint of acceptance ratio. (8) means the number of scheduled tasks for each type of VM should not exceed the number of all VMs of that type in the private cloud.

III. Dynamic Scheduling Algorithm

A. Virtual Queues

To guarantee the acceptance ratio constraint, we use a virtual queue as follows:

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

To guarantee the completion time, we apply *ϵ-persistent service* technique to build a virtual queue associated with each task queue:

$Z_{mdl}(t+1) = \max[Z_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}}(\epsilon_d - N_{mdl}(t) - W^{(t)}(D_{mdl}(t))) - 1_{\{Q_{mdl}(t)=0\}} N^{max}, 0]$.

B. Lyapunov Optimization and Dynamic Algorithm

Define $\Theta(t) = [Q(t), Z(t), K(t)]$ as a collective vector of all queues $Q_{mdl}(t), Z_{mdl}(t), K_{mdl}(t), \forall m \in M, d \in D, l \in L$ and define Lyapunov function as

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

The one-slot drift is

$$\begin{aligned}
&\Delta(\Theta(t)) + VC(t) \\
&\leq B + \sum_{m\in M}\sum_{d\in D}\sum_{l\in L} Q_{mdl}(t)[A_{mdl}(t) - N_{mdl}(t) - W^{(t)}(D_{mdl}(t))] \\
&+ \sum_{m\in M}\sum_{d\in D}\sum_{l\in L} Z_{mdl}(t) 1_{\{Q_{mdl}(t)>0\}}[\epsilon_d - N_{mdl}(t) - W^{(t)}(D_{mdl}(t))] \\
&+ \sum_{m\in M}\sum_{d\in D}\sum_{l\in L} K_{mdl}(t)[\alpha S_{mdl}(t) - A_{mdl}(t) - O_{mdl}(t)] \\
&+ V\sum_{m\in M}\sum_{d\in D}\sum_{l\in L}((O_{mdl}(t) + D_{mdl}(t))H(m, d, l) + \beta p_m[N_{mdl}(t^-) - N_{mdl}(t)]^+) \\
&= B' + \sum_{m\in M}\sum_{d\in D}\sum_{l\in L} A_{mdl}(t)[Q_{mdl}(t) - K_{mdl}(t)] \\
&+ \sum_{m\in M}\sum_{d\in D}\sum_{l\in L} O_{mdl}(t)[VH(m, d, l) - K_{mdl}(t)] \\
&+ \sum_{m\in M}\sum_{d\in D}\sum_{l\in L}[D_{mdl}(t)VH(m, d, l) - W^{(t)}(D_{mdl}(t))(Q_{mdl}(t) + Z_{mdl}(t))]
\end{aligned}$$

$+ \sum_{m\in M}\sum_{d\in D}\sum_{l\in L}((-N_{mdl}(t)])(Q_{mdl}(t) + 1_{\{Q_{mdl}(t)>0\}}Z_{mdl}(t)) + V\beta p_m[N_{mdl}(t^-) - N_{mdl}(t)]^+)$

The following algorithm makes $A_{mdl}(t)$, $O_{mdl}(t)$, $D_{mdl}(t)$, $N_{mdl}(t)$ decisions every slot to minimize the right-hand-side of the above drift-plus-penalty expression.

- (Algorithm for Admission Decision)
  Minimize $A_{mdl}(t)[Q_{mdl}(t) - K_{mdl}(t)] + O_{mdl}(t)[VH(m,d,l) - K_{mdl}(t)]$
  Subject to $0 \le A_{mdl}(t) + O_{mdl}(t) \le S_{mdl}(t)$
  Case (1):
  If $Q_{mdl}(t) - K_{mdl}(t) \ge 0$ and $VH(m,d,l) - K_{mdl}(t) < 0$,
  $A^*_{mdl}(t) = 0$ and $O^*_{mdl}(t) = S_{mdl}(t)$
  Case (2):
  If $Q_{mdl}(t) - K_{mdl}(t) < 0$ and $VH(m,d,l) - K_{mdl}(t) \ge 0$,
  $A^*_{mdl}(t) = S_{mdl}(t)$ and $O^*_{mdl}(t) = 0$
  Case (3):
  If $Q_{mdl}(t) - K_{mdl}(t) \ge 0$ and $VH(m,d,l) - K_{mdl}(t) \ge 0$,
  $A^*_{mdl}(t) = 0$ and $O^*_{mdl}(t) = 0$
  Case (4):
  If $Q_{mdl}(t) - K_{mdl}(t) < 0$ and $VH(m,d,l) - K_{mdl}(t) < 0$,
  at this time, if $Q_{mdl}(t) - K_{mdl}(t) < VH(m,d,l) - K_{mdl}(t)$, *i.e.*, $Q_{mdl}(t) < VH(m,d,l)$, $A^*_{mdl}(t) = S_{mdl}(t)$ and $O^*_{mdl}(t) = 0$, otherwise, $A^*_{mdl}(t) = 0$ and $O^*_{mdl}(t) = S_{mdl}(t)$
  The rationale behind this algorithm is that:
  Observing the coefficient of the two variables $A_{mdl}(t)$ and $O_{mdl}(t)$. If one of the two is negative and the other is positive, i.e., in the case (1) and (2), to achieve the minimum, we simply assign $S_{mdl}(t)$ to the variable whose coefficient is negative, and assign 0 to the other variable. If both coefficients are positive, we simply assign both variables to be 0. If both coefficients are negative, we need to compare the two coefficients, and assign $S_{mdl}(t)$ to the variable whose coefficient is the smaller, and assign 0 to the other variable.
- (Algorithm on VM allocation)
  For each $m' \in M$,
  Minimize

$$\sum_{d\in D}\sum_{l\in L}(-N_{m'dl}(t)(Q_{m'dl}(t) + 1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t)) + V\beta p_{m'}[N_{m'dl}(t^-) - N_{m'dl}(t)]^+) \quad (10)$$

  Subject to:
  $0 \le \sum_{d\in D}\sum_{l\in L} N_{m'dl}(t) \le N^{tot}_{m'}$
  One solution is describe in Algorithm (1). The algorithm is developed with the "water filling" idea: Because $-N_{m'dl}(t)(Q_{m'dl}(t) + 1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t)) + V\beta p_{m'}[N_{m'dl}(t^-) - N_{m'dl}(t)]^+$ is equivalent to $-N_{m'dl}(t)(Q_{m'dl}(t) + 1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t)) + 1_{\{N_{m'dl}(t)<N_{m'dl}(t^-)\}}V\beta p_{m'}$
  we want to fill water of volume $N^{tot}_{m'}$ into the tubes $N_{m'dl}$. For each tube, if the water level is below the threshold $N_{m'dl(t-)}$, the level increment by unit water is $\frac{1}{Q_{m'dl}(t)+1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t)+V\beta p_{m'}}$; otherwise, the level increment by unit water is $\frac{1}{Q_{m'dl}(t)+1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t)}$. Therefore, we try to fill the water step by step. In each step, we pick up the tube that is with smallest increment by unit volume of water, i.e., $N_{m'd^*l^*}$, if the water level of $N_{m'd^*l^*}$ is higher than its threshold $N_{m'dl}(t^-)$, we simply fill all the remaining water into that tube. Otherwise, we fill water into that tube until its level reaches the threshold $N_{m'dl}(t^-)$. And then repeat the selection process.

Algorithm 1: Algorithm solving (10)

1. Set $p = N^{tot}_{m'}$,
2. Select
   $N_{m'd^*l^*}(t) \in \text{argmax}_{d\in D,l\in L} Q_{m'dl}(t) + 1_{\{Q_{m'dl}(t)>0\}}Z_{m'dl}(t) + 1_{\{N_{m'dl}(t)<N_{m'dl}(t^-)\}}V\beta p_{m'}$.
3. If $N_{m'd^*l^*}(t) \ge N_{m'dl}(t^-)$, set $N_{m'd^*l^*}(t) = p$;
   Otherwise, set $N_{m'd^*l^*}(t) = \min\{p, N_{m'd^*l^*}(t^-)\}$.
4. Update $p = p - N_{m'd^*l^*}(t)$
5. If $p > 0$, Go to (2); otherwise, the algorithm ends.

- (Algorithm on Outsourcing Queueing Tasks)
  Minimize

$$D_{mdl}(t)VH(m,d,l) - W^{(t)}(D_{mdl}(t))(Q_{mdl}(t) + Z_{mdl}(t)) \quad (11)$$

  Subject to:
  $0 \le D_{mdl}(t) \le D^{max}_{mdl}$
  Solution:
  When $VH(m,d,l) \ge Q_{mdl}(t) + Z_{mdl}(t)$, $D^*_{mdl} = 0$.
  Otherwise, because $W^{(t)}(D_{mdl}(t) + D_0) \ge W^{(t)}(D_{mdl}(t)) + D_0$, (11) is decreasing on $[0, D^{max}_{mdl}]$. Therefore $D^*_{mdl} = D^{max}_{mdl}$.

IV. Analysis

A. Bounded Queues

Lemma 1: Our algorithm guarantees in all time slots,

$$K_{mdl}(t) \le K^{max}_{mdl} = VH(m,d,l) + \alpha S_{max} \quad (12)$$

$$Q_{mdl}(t) \le Q^{max}_{mdl} = VH(m,d,l) + (1+\alpha)S_{max} \quad (13)$$

$$Z_{mdl}(t) \le Z^{max}_{mdl} = VH(m,d,l) + \epsilon_d \quad (14)$$

Proof:
First we prove (12).
When $K_{mdl}(t) \le VH(m,d,l)$, $K_{mdl}(t+1) = VH(m,d,l) + \alpha S_{mdl}(t+1) \le VH(m,d,l) + \alpha S_{max}$.
When $K_{mdl}(t) > VH(m,d,l)$, according to our *Algorithm for Admission Decision*, $A^*_{mdl}(t) + O^*_{mdl}(t) = S_{mdl}(t)$, then $K_{mdl}(t+1) = \max[K_{mdl}(t) - \alpha S_{mdl}(t) - S_{mdl}(t), 0] \le K_{mdl}(t)$.
Then we prove (13).
When $Q_{mdl}(t) < K_{mdl}(t) \le VH(m,d,l) + \alpha S_{max}$, $Q_{mdl}(t+1) \le VH(m,d,l) + (1+\alpha)S_{max}$.
When $Q_{mdl}(t) \ge K_{mdl}(t)$, according to our *Algorithm on VM allocation*, $A^*_{mdl}(t) = 0$, so, $Q_{mdl}(t+1) \le Q_{mdl}(t)$.

At last we prove (14).

When $Z_{mdl}(t) \leq VH(m,d,l)$, $Z_{mdl}(t+1) \leq VH(m,d,l) + \epsilon_d$.

When $Z_{mdl}(t) > VH(m,d,l)$, according to our *Algorithm on Outsourcing Queueing Tasks*, $Z_{mdl}(t+1) \leq Z_{mdl}(t) + \epsilon_d - D_{mdl}^{max} \leq Z_{mdl}(t)$.

B. Worst-Case Delay

Lemma 2: (Worst-Case Delay) Suppose an algorithm is used that ensures the following for all slots t:

$Q_{mdl}(t) \leq Q_{mdl}^{max}$, $Z_{mdl}(t) \leq Z_{mdl}^{max}$.

where Q_{mdl}^{max} and Z_{mdl}^{max} are finite upper bounds on backlogs of tasks queues and associated virtual queues. Then, the worst-case delay of non-outsourced tasks in Q_{mdl} are bounded by the constant

$$U_d = \lceil \frac{(1+l)Q_{mdl}^{max} + Z_{mdl}^{max}}{\epsilon_d} \rceil \tag{15}$$

Proof:

We prove the guarantee of worst case delay by contradiction as follows:

Fix any slot $t_0 \geq 0$, and let $A_{mdl}(t_0)$ represent the tasks that arrives on this slot. Suppose the $A_{mdl}(t_0)$ tasks are not finished within the following U_d time slots, then it must be $Q_{mdl}(t) > 0$, where $t \in \{t_0+1, ..., t_0+U_d\}$. Then,

for all $t \in [t_0+1, t_0+U_d]$, we have:

$Z_{mdl}(t+1) = \max[Z_{mdl}(t) + \epsilon_d - N_{mdl}(t) - W^{(t)}(D_{mdl}(t))), 0]$.

and hence,

$Z_{mdl}(t+1) \geq Z_{mdl}(t) + \epsilon_d - N_{mdl}(t) - W^{(t)}(D_{mdl}(t)))$.

Summing the above over $t \in [t_0+1, t_0+U_d]$ yields:

$Z_{mdl}(t_0+U_d+1) - Z_{mdl}(t_0+1)$
$\geq \epsilon_d U_d - \sum_{t=t_0+1}^{t_0+U_d}[N_{mdl}(t) + W^{(t)}(D_{mdl}(t))]$

Rearrange terms in the above inequality and using the fact that $Z_{mdl}(t_0+U_d+1) \leq Z_{mdl}^{max}$, $Z_{mdl}(t_0+1) \geq 0$ and the definition of U_d yields:

$\sum_{t=t_0+1}^{t_0+U_d}[N_{mdl}(t) + W^{(t)}(D_{mdl}(t))]$
$\geq \epsilon_d U_d - Z_{mdl}^{max}$
$\geq (1+l)Q_{mdl}^{max}$
$\geq lQ_{mdl}^{max} + Q_{mdl}(t_0+1)$

This means more than $lQ_{mdl}^{max} + Q_{mdl}(t_0+1)$ elements depart Q_{mdl}.

There are $Q_{mdl}(t_0+1)$ elements in front of the $A_{mdl}(t_0)$ tasks, and always at most Q_{mdl}^{max} tasks at the back of the $A_{mdl}(t_0)$ tasks. Focusing on any one of the $A_{mdl}(t_0)$ tasks, and suppose the focused task is not processed in full in the U_d time slots. Then, during any time slot in $\{t_0+1, ..., t_0+U_d\}$, there are two cases:

(1) The focused task is not allocated with a VM, then the tasks behind it are not allocated with VMs. The number of all elements departing the queue in these all these time slots is no larger than $Q_{mdl}(t_0+1)$.

(2) The focused task is allocated with a VM, then the tasks behind it are possibly allocated with VMs. There are always at most Q_{mdl}^{max} tasks in the queue, so in any one time slot, at most Q_{mdl}^{max} element depart the queue. Because we suppose the focused task is not yet processed in full, this case will not happen more than l times. Therefore, totally fewer than lQ_{mdl}^{max} elements will depart the queue.

Summarizing the two cases, fewer than $Q_{mdl}(t_0+1) + lQ_{mdl}^{max}$ element will depart the queue.

Contradiction happens.

(maybe we can further prove this bound for worst-case delay is tight, e.g., by case study.)

C. Performance Optimality

doing.

V. Evaluation

doing.

VI. Note

- The drawback of this model is that there are too many queues: the number is $|D||M||L|$.