Optimal scheduling of tasks in federated hybrid clouds

***Abstract*—**

I. Introduction

Although cloud computing provides an illusion of unlimited resource pool, there still exists inherently limited scalability of single-provider clouds because a cloud provider provisioning capacity to meet the spike demand would suffer from under-utilization of resource. Therefore there is a need to federate multiple cloud providers to form a seamless computing resource pool and resource can be tapped into any cloud provider who can not serve requests locally at any time.

Even if public cloud service prevails, on-premise server cluster would still exist for many years. Idle computing resource at cloud users becomes a waste if it is not pooled into the global cloud and transferred to other users in need. We envision clouds would follow the development path like Internet which have developed from a conventional client-server architecture to Web 2.0. A cloud user equipped with racks of local computing servers would be tapped into the cloud, to form a hybrid architecture consisting of public clouds and locally managed computing resource from users at the edge of the Internet.

Because requests of computing resource could be served at either local data center, VMs contributed by user groups, or routed to other cloud providers, so there exists an optimal scheduling that maximizes the revenue of the community of cloud providers. We first model the system as a queueing network where requests for executing tasks enter the queues and departure when there are proper VMs to process them. We then adopt Lyapunov optimization technique to develop algorithms which achieve performance with a small constant gap away from the optimality and comply with SLA in terms of queueing delay. After that, we show that local revenue maximization meet optimal social welfare.

In summary, we have the following analytical results:

- A ready-to-use algorithm for each cloud provider to maximize its local revenue.
- SLA(in terms of queueing delay) are guaranteed.
- Local revenue maximization meet optimal social welfare.

II. Related work

Reservoir project is motivated by the vision of implementing an architecture that would enable providers of cloud infrastructure to dynamically partner with each other to create a seemingly infinite pool of IT resources while preserving their individual autonomy in making technological and business management decisions [1]. The goal is to facilitate an open service-based online economy in which resources and services are transparently provisioned and managed across clouds on an on-demand basis at competitive costs with high-quality service.

III. The problem

A. System model

We consider a geo-distributed federated hybrid cloud consisting of a number of cloud providers, denoted as set $\mathcal{C}$. Each cloud provider has a group of users, who send requests and sell their idle resource to the cloud provider. We assume at time slot t, user group n send $a_n(t)$ requests to cloud provider n. Upon receipt of a request, a cloud provider determines whether it will process the request or delegate it to other cloud provider. If the cloud provider n decides to process it by itself, the request would be placed in its request queue $Q_{nn}(t)$. Otherwise, it would be routed to one of the other cloud, say, cloud m, where the request would be placed in request queue $Q_n^{(m)}(t)$. In each time slot, cloud provider n determine the departure of the request queues $Q_n^{(m)}(t), \forall n \in \mathcal{C}$. The departure of queue $Q_n^{(m)}(t)$ has two destinations: $c_n^{(m)}(t)$ requests are processed at its data center, and $u_n^{(m)}(t)$ are processed at the VMs contributed by its users.

To simplify the model, we assume each user request consumes one unit of computing resource. When computing resource is tapped from users or other cloud providers, the cloud provider would pay the other entity for processing the request at a pre-set price. Specifically, cloud provider n charges its user $i_n(t)$ dollars per task. Cloud provider n charges cloud provider m at the wholesale rate $b_n^{(m)}(t)$. The operational cost of executing a task in the data center at cloud provider n is $k_n(t)$. Cloud provider n pays its user for executing a task $h_n(t)$.

The important notations are summarized in Table I.

Each cloud provider $n, \forall n \in \mathcal{C}$ maintains $|C|$ queues, i.e., $Q_n^{(m)}, \forall m \in \mathcal{C}$. The law of queue update is as follows:

$$Q_n^{(m)}(t+1) = max[Q_n^{(m)}(t) + r_n^{(m)}(t) - c_n^{(m)}(t) - u_n^{(m)}(t), 0].$$

B. Problem formulation

Because each cloud provider wants to maximize its revenue, each of them would solve the following optimization problem in each time slot, based on the current market information on VMs, and backlogs of related queues. For cloud provider n ($\forall n \in \mathcal{N}$), the local optimization problem is defined as

TABLE I
IMPORTANT NOTATIONS

$a_n(t)$	Number of arrival requests in time slot t.
$r_n^{(m)}(t)$	Number of requests routed from cloud provider n to cloud provider m. If $n == m$, it represents the requests would be processed locally.
$Q_n^{(m)}(t)$	Backlog of request queue buffering requests routed from cloud provider n to cloud provider m.
$c_n^{(m)}(t)$	Number of requests departing from request queue $Q_n^{(m)}$ and being processed at data center.
$u_n^{(m)}(t)$	Number of requests departing from request queue $Q_n^{(m)}$ and being processed at user contributed VMs.
$i_n(t)$	Charging rate from users for a request by cloud provider n at time slot t.
$b_n^{(m)}(t)$	Charging rate for a request routed from cloud provider n to cloud provider m at time slot t.
$k_n(t)$	Cost of processing a request at the data center of cloud provider n at time slot t.
$h_n(t)$	Payment to users for serving a request from cloud provider n at time slot t.
A_n	Maximal number of VMs contributed by users at cloud provider n (should be time variable?)
B_n	Capacity of data center at cloud provider n

follows:

$$\begin{aligned}\max \quad M(t) = \quad & i_n(t)\sum_{m\in\mathcal{C}} r_n^{(m)}(t) \\ & -\sum_{m\in\mathcal{C}} r_n^{(m)}(t)b_n^{(m)}(t) \\ & +\sum_{m\in\mathcal{C}} b_n^{(m)}(t)R_m^{(n)}(t) \\ & -h_n(t)\sum_{m\in\mathcal{C}} u_n^{(m)}(t) \\ & -k_n(t)\sum_{m\in\mathcal{C}} c_n^{(m)}(t)\end{aligned}$$

subject to:

$$\sum_{m\in\mathcal{C}} u_n^{(m)}(t) < A_n \tag{1}$$
$$\sum_{m\in\mathcal{C}} c_n^{(m)}(t) < B_n \tag{2}$$
$$u_n^{(m)}(t) + c_n^{(m)}(t) \le Q_n^{(m)}(t), \forall m \in \mathcal{C} \tag{3}$$
$$\sum_{m\in\mathcal{C}} r_n^{(m)}(t) \le a_n(t). \tag{4}$$

Meanwhile, the whole cloud community hopes the social welfare could be maximized as well. Therefore, the global social welfare optimization problem is defined as follows:

$$\begin{aligned}\max \quad G(t) = \quad & \sum_n [\sum_m r_n^{(m)}(t)i_n(t) \\ & -h_n(t)\sum_{m\in\mathcal{C}} u_n^{(m)}(t) \qquad (5) \\ & -k_n(t)\sum_{m\in\mathcal{C}} c_n^{(m)}(t)], \qquad (6)\end{aligned}$$

subject to:

$$\sum_{m\in\mathcal{C}} u_n^{(m)}(t) < A_n, \forall n \in \mathcal{C} \tag{7}$$
$$\sum_{m\in\mathcal{C}} c_n^{(m)}(t) < B_n, \forall n \in \mathcal{C} \tag{8}$$
$$u_n^{(m)}(t) + c_n^{(m)}(t) \le Q_n^{(m)}(t), \forall n, m \in \mathcal{C} \tag{9}$$
$$\sum_{m\in\mathcal{C}} r_n^{(m)}(t) \le a_n(t), \forall n \in \mathcal{C}. \tag{10}$$

IV. DYNAMIC ALGORITHM

A. Introducing virtual queues

In order to guarantee worst-case queueing delay, we define virtual queues

$$Z_n^{(m)}(t+1) = \max[Z_n^{(m)}(t) + 1_{\{Q_n^{(m)}(t)>0\}}(\epsilon_n^{(m)} - c_n^{(m)}(t) - \mu_n^{(m)}(t))$$

B. Dynamic Algorithm Design via Drift-Plus-Penalty Minimization Method

Define the vector of all queues as

$$\Theta(t) = [\mathrm{Q}, \mathrm{Z}]$$

We first deal with the global optimization problem. Define our Lyapunov function as

$$L(\Theta(t)) = \sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}} \frac{1}{2}(Q_n^{(m)}(t) + Z_n^{(m)}(t)). \tag{11}$$

Then the Lyapunov drift-plus-penalty satisfies the following inequality:

$$\begin{aligned} & \Delta(\Theta(t)) + G(t) \\ \le \quad & B + \sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}} Q_n^{(m)}(t)(r_n^{(m)}(t) - c_n^{(m)}(t) - \mu_n^{(m)}(t)) \\ & + \sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}} Z_n^{(m)}(t)(1_{\{Q_n^{(m)}(t)>0\}}(\epsilon_n^{(m)} - c_n^{(m)}(t) - \mu_n^{(m)}(t)) \\ & \qquad - 1_{\{Q_n^{(m)}(t)=0\}}(A_n + B_n)) \\ & - V\sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}}(r_n^{(m)}(t)i_n(t) - h_n(t)\mu_n^{(m)}(t) - k_n^{(m)}(t)c_n^{(m)}(t)) \\ = \quad & B + \sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}} \\ & [Q_n^{(m)}(t)(r_n^{(m)}(t) - c_n^{(m)}(t) - \mu_n^{(m)}(t)) \\ & + Z_n^{(m)}(t)(1_{\{Q_n^{(m)}(t)>0\}}(\epsilon_n^{(m)} - c_n^{(m)}(t) - \mu_n^{(m)}(t)) \\ & \qquad - 1_{\{Q_n^{(m)}(t)=0\}}(A_n + B_n)) \\ & - V(r_n^{(m)}(t)i_n(t) - h_n(t)\mu_n^{(m)}(t) - k_n^{(m)}(t)c_n^{(m)}(t))] \\ = \quad & B' + \sum_{n\in\mathcal{C}}\sum_{m\in\mathcal{C}} \\ & [r_n^{(m)}(t)(Q_n^{(m)}(t) - Vi_n(t)) \\ & + c_n^{(m)}(t)(Vk_n^{(m)} - Q_n^{(m)}(t) - 1_{\{Q_n^{(m)}(t)>0\}}Z_n^{(m)}(t)) \\ & + \mu_n^{(m)}(t)(Vh_n(t) - Q_n^{(m)}(t) - 1_{\{Q_n^{(m)}(t)>0\}}Z_n^{(m)}(t)] \end{aligned} \tag{12}$$

To minimize the right-hand-side of the above inequality, we may solve the following problem, at each cloud provider n :

$$\begin{aligned}\max \quad & \sum_{m\in\mathcal{C}} [r_n^{(m)}(t)(Q_n^{(m)}(t) - Vi_n(t)) \\ & + c_n^{(m)}(t)(Vk_n^{(m)} - Q_n^{(m)}(t) - 1_{\{Q_n^{(m)}(t)>0\}} Z_n^{(m)}(t)) \\ & + \mu_n^{(m)}(t)(Vh_n(t) - Q_n^{(m)}(t) - 1_{\{Q_n^{(m)}(t)>0\}} Z_n^{(m)}(t))]\end{aligned}$$

Subject to:

(1)(2)(3)(4)

This is easy because it is a linear programming.

C. Heuristic

D. Discussion on practical implementation

V. Performance Analysis

A. Bound of Request Queue Length

B. Bound of Queueing Delay

C. Optimality against the T-Slot Lookahead Mechanism

VI. EMPIRICAL STUDIES

VII. Conclusion

VIII. Discussion for future directions

- "Consider to merge the —C— queues inside each cloud provider, in order to decrease the number of variables": each queue adopts different ϵ to achieve different worst-case queueing delay (because different cloud has different SLA), when cloud A delegate a request to cloud B, cloud B comply the same SAL as cloud A with the end-user. But when there are N zone, there would be N queues in each cloud providers. Such a system architecture doesn't scale well.
  A solution is to build H queues in each cloud provider. H queues represent H levels of SLA.

References

[1] B. Rochwerger, D. Breitgand, E. Levy, a. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 1–11, Jul. 2009.