A Novel Scheduling Mechanism For Hybrid Cloud Systems

Haoran Wei
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware
Email: nancywhr@udel.edu

Fanchao Meng
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware
Email: fcmeng@udel.edu

Large scale and complexity in structure are two major trends in current cloud environments. Bringing cloud computing to its full potential requires a paradigm shift toward cloud resiliency. A major challenge in designing resilient clouds is to avoid single point of failures in cloud management systems. The state-of-the-art cloud management systems are mostly designed by relying on centralized managing systems with complicated master nodes, which are prone to single point of failures. In this paper, we propose a novel resilient hybrid cloud management system to ensure offering consistent cloud services to users, which consists of two tiers: a peer-to-peer (P2P) tier and a centralized tier. In the centralized tier, each local cloud has a centralized management system, while these local clouds form a P2P cloud in the P2P tier. Our hybrid cloud system benefits from auto scaling across the entire system while it easily manages each local cloud. We propose an innovative user request scheduling mechanism, dedicated to our system, considering multi-objectives: minimizing user requests average makespan, while ensuring load balancing. Finally, we conduct extensive experiments to evaluate the performance of our proposed hybrid cloud management system.

Keywords– hybrid cloud management system; peer-to-peer; scheduling mechanism; load balancing; priority algorithm

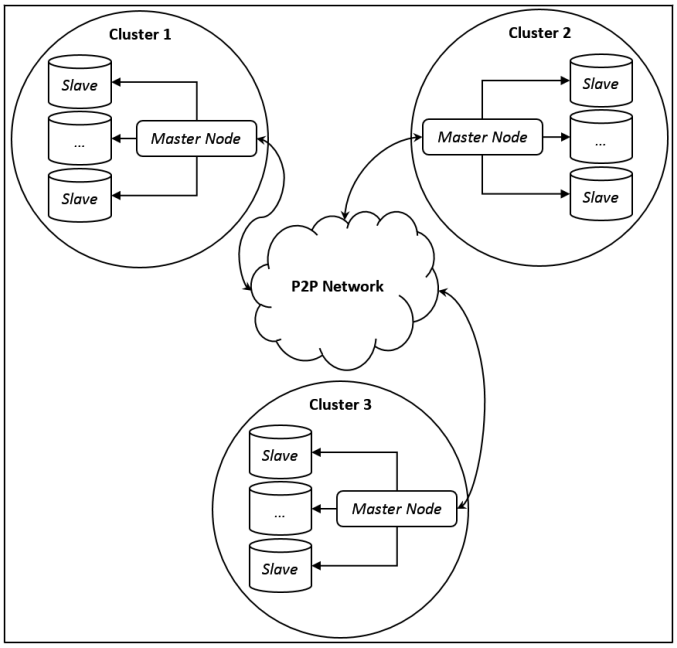


Fig. 1. An example of the hybrid system architecture.

I. Introduction

Clouds have been adopted in sundry scenarios from massive scale organizations across continents to startup companies, and this trend has kept growing rapidly. However, few cloud frameworks have been proposed in the past decade, with centralized clouds ruling the market (e.g. Google Kubernetes [1] and Amazon AWS). In contrast only a few decentralized methods have been proposed for general use (e.g. Storj and MIT's Enigma which are both aimed security issues, and are based on the peer-to-peer (P2P) network and the bitcoin protocol).

Centralized frameworks, though popular, are limited by several drawbacks. As is well known, single points in centralized frameworks may provoke failure of the whole system. Additionally, the weak scalability in large scale networks is another issue for centralized systems. A promising alternative approach is the P2P framework. Unfortunately, P2P frameworks can hardly do optimized resource allocation due to the difficulty in management (which is critical to small scale clouds). Therefore, our goal in this paper is to design a hybrid cloud framework that performs well in scenarios where local and small clouds are aggregated together to form a large cloud, thereby satisfying wider ranges. Such a hybrid cloud framework can take advantage of both centralized and P2P frameworks, so as to accept more user requests and minimize job waiting time.

A. Our Contribution. We propose an innovative hybrid cloud framework which is composed of two tiers as shown in Figure 1. The lower tier deploys a centralized architecture for each local cluster in which machines can be heterogeneous, while the upper tier organizes all clusters into a P2P network. The key component of this framework is the scheduling mechanism consisting of five major components: user request partition, job prioritization, single job scheduling, intra-cluster load balancing and inter-cluster load balancing. Our framework efficiently utilizes computing resources in each cluster, and enhances the cooperation between clusters.

B. Related Work. The dominant cloud frameworks are mostly centralized, such as Borg and Kubernetes [1], in which all the machines are managed by a central controller, which is threatened by the singleton failure risk. The robust requirement to the singleton controller will be extremely strict, which consequently will make it more complex. Iyengar et al. in

[2] have affirmed that added complexity would introduce brittleness to the system by creating multiple single points of failure. In [3], Verissimo et al. point out that to address high-resilience cloud systems, a distributed design is necessary.

On the P2P side, Babaoglu et al. proposed a P2P cloud system in [4] based on the gossip algorithm aims at linking all devices into one network to run large scale tasks. Furthermore, some studies focus on P2P network's easy-to-scale feature to aggregate scattered computing resources. Mayer et al. propose an autonomous cloud platform in [5] built on a voluntary computing mechanism using P2P to serve as a PaaS. However, when concerning resource allocation in finer granularities, pure P2P frameworks perform poorly most of the time [5].

The scheduling problem is the core issue in clouds. That is, determining where each job should be executed. Schwarzkopf et al. in [6] design a parallel scheduler, called "Omega", based on shared states, using lock-free optimistic concurrency control, to achieve both extensibility in implementation and high scalability. From the perspective of scalability and fault-tolerance, Zhang et al. propose a distributed resource management and job scheduling system in [7], called Fuxi deployed in Alibaba.

Additionally, The load balancing is a critical factor in the scheduling problem. Singh et al. in [8] discuss how to leverage the integrated agility for non-disruptive load balancing in data centers across multiple resource layers. To make the analysis more mature, queuing models [9] [10] [11] have been introduced. Li et al. propose a P2P dedicated queuing model in [12]. Khazaei et al. use the Markov Chain model in [13] to evaluate the performance of a cloud computing center.

In reality, user requests are not homogeneous. Users will request for heterogeneous devices. Sharma et al. [14] develop methodologies for incorporating task placement constraints and machine properties into performance benchmarks of large compute clusters. And Tumanov et al. in [15] propose a specific approach for accommodating both soft and hard constraints, and general machine heterogeneity.

C. Organization. The rest of the paper is organized as follows. Section II provides a description of the hybrid cloud framework. Section III gives the details of our scheduling mechanism including prioritization, single job scheduling, intra-cluster load balancing and inter-cluster load balancing. Section IV shows the experiments and the performance evaluation. Sections V, VI discuss the conclusion and our future work.

II. Hybrid P2P Cloud Framework

Informally, we consider a cloud provider managing a set of centralized local clouds, called *Clusters* in this paper, each of which owns a set of heterogeneous machines, and all clusters are connected as a P2P network. The machines in each cluster are partitioned into m groups by the types of computing resources, (e.g. Intel Xeon E5 and Intel i7,) denoted by $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. All the machines in the cluster are managed by a central controller for the cluster, called the *Master Node*. Each master node has a finite waiting queue to cache user requests. Based on the Queuing theory, the arrival of user requests is assumed to follow the Poisson distribution. In this paper each user request $\mathcal{R}_l$ can be partitioned into a set of unit jobs. The scheduler will dispatch requests to appropriate group as shown in Figure2. With respect to the storage resources, we presume that they are always adequate for user requests.

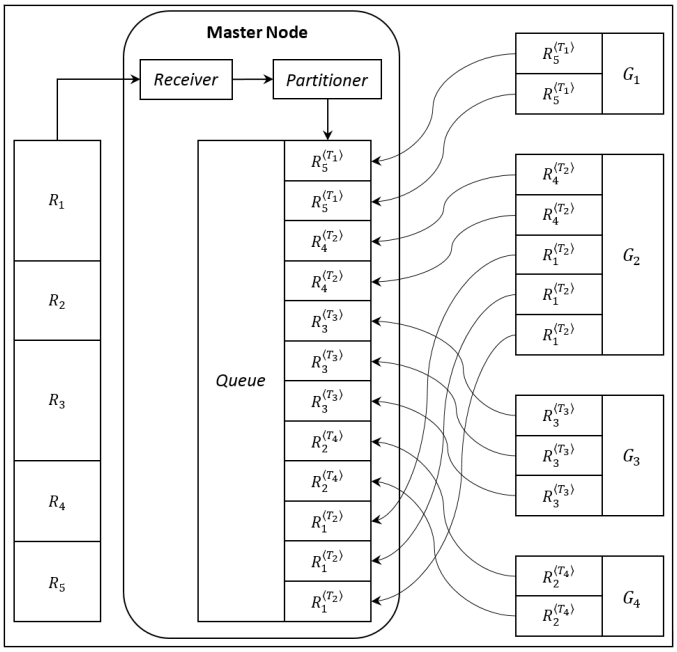


Fig. 2. The master node and groups. R_i is a user request sent to the master node. If it is accepted, then the master node will partition it into a set of unit jobs $\{R^{\langle T_j \rangle}\}_i$, where T_j is the type of requested computing resources. Each unit job then will be sent to the corresponding group G_j according to its type.

The scheduler is responsible for determining the unit jobs and where each job will be executed or queued. There are five major components in the scheduler: request partitioning, prioritization, single job scheduling, intra-cluster load balancing and inter-cluster load balancing. The request partition module will partition a given user request into a set of unit jobs. The prioritization module will assign each unit job a priority value, and this value might be updated along with the changes of the cluster's states. The single job scheduling module will place each unit job in an appropriate location (i.e. a group in the local cluster or in other cluster), according to their priorities and the system's states. The higher the priority, the sooner the job will be executed. Since groups may have different arrival rates, service rates and computing capability, workloads among groups might not be balanced nor might be workloads between clusters. The intra- and inter-cluster load balancing modules address these issues. Based on different workload situations, only when specific criteria are met, the load balancing procedures are triggered.

In the remainder of this section, we provide formal definitions of the concepts outlined above.

Definition 1. User Request $\mathcal{R}_l = \langle \mathcal{R}_l^{\mathcal{T}_i}, \mathcal{R}_l^{\mathcal{L}} \rangle$ denotes a user

request, where $\mathcal{R}_l^{\mathcal{T}_i}$ means the type of requested computing resources, and $\mathcal{R}_l^{\mathcal{L}}$ represents the cardinality of the unit jobs (see **Definition 2**) set derived from the user request. The states of a given user request include:

1. Entry Time Point (ETP): The time point at which the user request is accepted by a cluster.

Definition 2. Unit Job Unit job is the minimum schedulable and executable unit in the system. Each unit job belongs to a user request. The jth unit job of $\mathcal{R}_l$ is denoted by $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$, or $\langle \mathcal{T}_i \rangle_j$ for convenience given the context of $\mathcal{R}_l$, where $\mathcal{T}_i$ denotes the type of requested computing resources. All unit jobs are of the same scale in terms of computation. The states of a unit job include:

1. Finish Time Point (FTP): The time point at which the unit job is finished.

2. Priority (Pri): The higher the priority, the sooner the unit job would be scheduled.

Definition 3. Cluster A cluster is a centralized cloud consisting of one master node (see **Definition 5**) and a set of slave nodes (see **Definition 6**). Slave nodes are organized into different groups (see **Definition 4**) With respect to their computing resource types. $\mathcal{C}_i$ denotes the ith cluster.

Definition 4. Group A group is a collection of slave nodes of the same computing resource type in a given cluster. $\mathcal{G}_{ij}$ denotes the jth group in $\mathcal{C}_i$, or $\mathcal{G}_j$ if the context of $\mathcal{C}_i$ is given.

Definition 5. Master Node A master node is a central workstation of a cluster responsible for receiving incoming user requests, partitioning each user request into a set of unit jobs, scheduling unit jobs, managing slave nodes, and communicating with master nodes in other clusters.

Definition 6. Slave Node A slave node is a physical machine whose only duty is to execute unit jobs. Here we presume that one slave node can only execute one unit job within one execution time. The states of slave node include:

1. Occupied (Ocp): '1' means this slave is working, and '0' means it is idling.

III. Scheduling Mechanism

Each incoming user request will be partitioned into unit jobs, and each unit job will be prioritized. Our scheduler is placing unit jobs based on dynamically adjusting priorities and system states. Three scheduling approaches are engaged in our scheduler, the intra-cluster load balancing, the inter-cluster load balancing and the single unit job scheduling. The scheduler will first check if the system is about to be overwhelmed to determine if the upcoming user requests would be rejected. Then if any unit job going beyond the priority threshold is detected, then the scheduler firstly attempts an intra-cluster load balancing. Thereafter, if the scheduler detects that the system is still not balanced yet internally, then an inter-cluster load balancing will be attempted. If the fact is both the current cluster and other ones in a certain range are "very busy", then the load balancing would not be called endlessly, instead periodically. The single unit job scheduling is about to be called afterward, during which the unit jobs going beyond the priority threshold will be considered to migrate to other groups. The user request reception mechanism is shown in Algorithm 1, and the general scheduling mechanism is shown in Algorithm 2, in which $\hat{N}_{mig}^k$ and $\hat{N}_{acc}^k$ are defined below.

We introduce the estimated baseline makespan ratio eleva-

Algorithm 1 User Request Reception Mechanism

```
INPUT: A user request R_l.
Let Γ_rej be a predefined threshold to reject the user request.
if Σ_k N̂^k_mig − Σ_k N̂^k_acc ≥ Γ_rej then
    Upcoming user request is rejected.
else
    User Request Partitioning.
    for all unit jobs do
        Call Prioritization Algorithm.
    end for
    Call General Scheduling Mechanism.
end if
```

tion (EBMRE) of a group as a measure, and take the values at both the current moment $CurrTP$ and a certain future moment $CurrTP + \Delta t$ into account to evaluate if our system is about to be overwhelmed.

Expected Makespan:

1. Expected Makespan of unit job $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$:

$$M(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}) = t_{\mathcal{T}_i} \tag{1}$$

2. Expected Makespan of a user request $\mathcal{R}_l$:

$$M(\mathcal{R}_l) = \left\lceil \mathcal{R}_l^{\mathcal{L}} / |\mathcal{G}_i| \right\rceil \cdot t_{\mathcal{T}_i} \tag{2}$$

where $|\mathcal{G}_i|$ is the cardinality of slave nodes located in group $\mathcal{G}_i$ where $\mathcal{R}_l$ is originally assigned, and $t_{\mathcal{T}_i}$ is the unit job execution time in $\mathcal{G}_i$. The expected case is the most ideal one without waiting.

Actual Makespan:

1. Actual Makespan of a unit job $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$:

$$M'(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}) = \mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}.FTP - \mathcal{R}_l.ETP \tag{3}$$

2. Actual Makespan of a user request $\mathcal{R}_l$:

$$M'(\mathcal{R}_l) = \max_j \{\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}.FTP\} - \mathcal{R}_l.ETP \tag{4}$$

The actual makespan can only be applied to finished jobs or user requests.

Estimated Makespan:

1. Estimated Makespan of a unit job $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$:

For finished unit jobs, their estimated makespans are equal to the actual ones. For the unit jobs still in the system, their estimated makespans are computed as:

$$\begin{aligned} \hat{M}(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}) = &\left\lceil n_k / |\mathcal{G}_i| \right\rceil \cdot t_{\mathcal{T}_i} + t_{\mathcal{T}_i} \\ &+ (CurrTP - \mathcal{R}_l.ETP) \end{aligned} \tag{5}$$

where n_k is the number of unit jobs waiting on $\mathcal{G}_k$ whose priorities are greater or equal to that of $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$. And when the unit job is executing instead of waiting, $n_k = 0$.

2. Estimated Maksepan of a user request $\mathcal{R}_l$:

$$\hat{M}(\mathcal{R}_l) = \max_j \{\hat{M}(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j})\} \tag{6}$$

Cumulative Makespan:

Let $\mathbb{U}_k$ be a set of unit jobs.

1. Expected Cumulative Makespan of $\mathbb{U}_k$:

$$M_\Sigma(\mathbb{U}_k) = \textstyle\sum_{\langle \mathcal{T} \rangle_i \in \mathbb{U}_k} M(\langle \mathcal{T} \rangle_i) \tag{7}$$

Algorithm 2 General Scheduling Mechanism

INPUT: A unit job $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$.
Let ρ_{ll} and ρ_{gl} be the enabling flags for the intra- and inter- cluster load balancing respectively. Initially, $\rho_{ll} = 1$ and $\rho_{gl} = 1$.
Let Γ_{ll} and Γ_{gl} be the predefined trigger thresholds for the intra- and inter- cluster load balancing respectively.
Let t_{hold} be a predefined period length.
if $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$ can be executed immediately in the current group. **then**
    Place it into an available machine in the current group.
else
    if $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}.Pri < \Gamma_{pri}$ **then**
        $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$ keeps waiting on its current group.
    else
        if $\sum_k \hat{N}_{mig}^k + \sum_k \hat{N}_{acc}^k \geq \Gamma_{ll} \wedge \rho_{ll} = 1$ **then**
            Call Intra-Cluster Load Balancing Algorithm.
            $\rho_{ll} = 0$.
            Update priorities.
            Update $\hat{N}_{mig}^k$ and $\hat{N}_{acc}^k$.
        end if
        if $\sum_k \hat{N}_{mig}^k - \sum_k \hat{N}_{acc}^k \geq \Gamma_{gl} \wedge \rho_{gl} = 1$ **then**
            Call Inter-Cluster Load Balancing Algorithm.
            $\rho_{gl} = 0$.
            Update priorities.
            Update $\hat{N}_{mig}^k$ and $\hat{N}_{acc}^k$.
        end if
        if $\sum_k \hat{N}_{mig}^k > \sum_k \hat{N}_{acc}^k \wedge \rho_{ll} = 0 \wedge \rho_{gl} = 0$ **then**
            Wait t_{hold} before set ρ_{ll} and ρ_{gl} to 1.
        else
            $\rho_{ll} = 1$ and $\rho_{gl} = 1$.
        end if
        Call Single Job Scheduling Algorithm.
    end if
end if

2. Estimated Cumulative Makespan of $\mathbb{U}_k$:

$$\hat{M}_\Sigma(\mathbb{U}_k) = \sum_{\langle \mathcal{T} \rangle_i \in \mathbb{U}_k} \hat{M}(\langle \mathcal{T} \rangle_i) \tag{8}$$

Actual Makespan Ratio:

1. Actual Makepan Ratio of $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$:

$$\mathscr{R}'_M(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}) = {}^{M'(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j})}/_{M(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j})} \tag{9}$$

2. Actual Makepan Ratio of $\mathcal{R}_l$:

$$\mathscr{R}'_M(\mathcal{R}_l) = {}^{M'(\mathcal{R}_l)}/_{M(\mathcal{R}_l)} \tag{10}$$

Estimated Makespan Ratio:

1. Estimated Makepan Ratio of $\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}$:

$$\hat{\mathscr{R}}_M(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j}) = {}^{\hat{M}(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j})}/_{M(\mathcal{R}_l^{\langle \mathcal{T}_i \rangle_j})} \tag{11}$$

2. Estimated Makepan Ratio of $\mathcal{R}_l$:

$$\hat{\mathscr{R}}_M(\mathcal{R}_l) = {}^{\hat{M}(\mathcal{R}_l)}/_{M(\mathcal{R}_l)} \tag{12}$$

Estimated Cumulative Makespan Ratio:

1. Estimated Cumulative Makespan Ratio of $\mathbb{U}_k$:

$$\hat{\mathscr{R}}_M^\Sigma(\mathbb{U}_k) = \hat{M}_\Sigma(\mathbb{U}_k)/M_\Sigma(\mathbb{U}_k) \tag{13}$$

Baseline Makespan Ratio:

Given a set of consecutive user requests finished by the current time, denoted by $\mathbb{R}_B = \{\mathcal{R}_1, \ldots, \mathcal{R}_{N_B}\}$.

$$\mathscr{R}_M^B = {}^{\sum_i [w_i \cdot \mathscr{R}'_M(\mathcal{R}_i)]}/_{\sum_i w_i} \tag{14}$$

where w_i is computed as

$$w_i = {}^{(\mathcal{R}_i^{\mathcal{L}} - \min_j\{\mathcal{R}_j^{\mathcal{L}}\} + 1)}/_{(\max_j\{\mathcal{R}_j^{\mathcal{L}}\} - \min_j\{\mathcal{R}_j^{\mathcal{L}}\} + N_B^{\mathbb{U}})} \tag{15}$$

where $N_B^{\mathbb{U}} = \sum_{i=1}^{N_B} \mathcal{R}_i^{\mathcal{L}}$.

Estimated Baseline Makespan Ratio Elevation (EBMRE):

1. EBMRE of $\mathcal{G}_k$ at a given time t:

$$\begin{aligned} \hat{\mathscr{E}}_M^t(n|\mathcal{G}_k) =& {}^{N_B^{\mathbb{U}}}/_{(N_B^{\mathbb{U}} + n)} \cdot \mathscr{R}_M^B \\ &+ {}^{n}/_{(N_B^{\mathbb{U}} + n)} \cdot \left[\sum_{i=1}^{n} \hat{\mathscr{R}}_M(\langle \mathcal{T}_k \rangle_i)\right] \end{aligned} \tag{16}$$

where n is the number of unit jobs waiting on $\mathcal{G}_k$ at the time t.

Let μ_k and σ_k be the mean and the variance of cardinalities of unit job sets in requests assuming the cardinalities follow the Normal distribution. The reason we make the assumption is that the actual distribution of cardinalities is unknown. However, it is reasonable to assume that it follows the Normal distribution empirically by taking a set of samples and calculating the mean and the variance. Let L_{ur} be the upper bound of the cardinalities. Let η_k be the departure rate of $\mathcal{G}_k$, which is a constant in our study, and $\eta_k = |\mathcal{G}_k| \cdot \bar{\eta}_k$, where $\bar{\eta}_k$ is the departure rate of one slave in $\mathcal{G}_k$. Let $\hat{N}_{I_{\Delta t}}$ represent the estimated number of unit jobs come into $\mathcal{G}_k$ from $CurrTP$ until a future moment $CurrTP + \Delta t$, where Δt is a predefined period length. $\hat{N}_{mig}^k$ and $\hat{N}_{acc}^k$ (defined below) will be updated every Δt. Let λ_k be the user request arrival rate assuming the arrivals follow the Poisson distribution. Let Γ_{pri} be the predefined priority threshold. A unit job's priority going beyond the threshold means it should be scheduled as soon as possible. Let Γ_{lb} be the predefined load balancing threshold. Going beyond it means a load balancing is needed. Therefore,

$$\begin{aligned} \hat{N}_{I_{\Delta t}} &= \lambda_k \cdot \Delta t \cdot E[l | 1 \leq l \leq L_{ur}] \\ &= \lambda_k \cdot \Delta t \cdot \big[\mu_k \cdot \sigma_k \\ &\quad - \sigma^{2\,[f(L_{ur}) - f(1)]}/_{[F(L_{ur}) - F(1)]}\big] \end{aligned} \tag{17}$$

where l is the cardinality, and

$$f(l) = {}^{1}/_{(\sigma_k\sqrt{2\pi})} \cdot e^{-\frac{1}{2}(\frac{l-\mu_k}{\sigma_k})^2} \tag{18}$$

$$\begin{aligned} F(l) &= {}^{1}/_{\sqrt{2\pi}} \cdot \int_{-\infty}^{l} e^{-x^2/2} \mathrm{d}x \\ &= {}^{1}/_{2} \cdot [1 + \varphi_e({}^{(x - \mu_k)}/_{(\sigma_k\sqrt{2})})] \end{aligned} \tag{19}$$

$$\varphi_e(y) = {}^{1}/_{\sqrt{\pi}} \cdot \int_{-y}^{y} e^{-t^2} \mathrm{d}t \tag{20}$$

Let $N_{O_{\Delta t}}$ represent the number of unit jobs leaving $\mathcal{G}_k$ from $CurrTP$ until $CurrTP + \Delta t$.

$$N_{O_{\Delta t}} = \eta_k \cdot \Delta t \tag{21}$$

Let N^{CurrTP} represent the number of unit jobs in $\mathcal{G}_k$ currently. Let $\hat{N}_{mig}^k$ be the number of jobs would be migrated out of $\mathcal{G}_k$, and $\hat{N}_{acc}^k$ be the number of unit jobs at most $\mathcal{G}_k$ could accept from other groups. Given a group $\mathcal{G}_k$, four cases are in consideration.

Case 1: $\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k) \geq \Gamma_{lb}$

Without loss of generality, we assume that $\mathcal{G}_k$ would not migrate unit jobs to other groups nor accept any from others within Δt.

Case 1-1: $\hat{\mathscr{E}}_M^{CurrTP+\Delta t}(m|\mathcal{G}_k) \geq \Gamma_{lb}$

In this case, $\mathcal{G}_k$ needs to migrate a certain number of unit jobs to other groups. The goal is to achieve $\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k) < \Gamma_{lb}$. And apparently, $N^{CurrTP} + \hat{N}_{I_{\Delta t}} > N_{O_{\Delta t}}$. Let $\Delta N_{\Delta t}$ be the number of leftover unit jobs. $\Delta N_{\Delta t} > 0$ in this case.

Step 1: *Estimate when the group starts to be over saturated.* Let $CurrTP+\Delta t'$ be that moment, i.e. $\hat{N}_{I_{\Delta t'}} + N^{CurrTP} = N_{O_{\Delta t}}$. Therefore,

$$\begin{aligned} &\lambda_k \cdot \Delta t' \cdot E[l|1 \le l \le L_{ur}] + N^{CurrTP} = \eta_k \cdot \Delta t \\ &\therefore \Delta t' = {}^{(\eta_k \cdot \Delta t - N^{CurrTP})}/_{\lambda_k \cdot E[l|1 \le l \le L_{ur}]} \end{aligned} \tag{22}$$

Let $\hat{N}_{lo}$ estimate $\Delta N_{\Delta t}$.

$$\hat{N}_{lo} = \lambda_k \cdot (\Delta t - \Delta t') \cdot E[l|1 \le l \le L_{ur}] \tag{23}$$

Step 2: *Compute the number of jobs to migrate.*

Given the goal $\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k) < \Gamma_{lb}$, we firstly compute n,

$$n = \text{argmax}_m\{\hat{\mathscr{E}}_M^{CurrTP}(m|\mathcal{G}_k) < \Gamma_{lb}\} \tag{24}$$

To compute $\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k)$ based on the formula 16, we assume, in the worst case, all leftover jobs come in at the moment $CurrTP + \Delta t'$, and that there is no other arrival or departure during $\Delta t - \Delta t'$, which yields the estimate as follows

$$\hat{\mathscr{R}}_M(\langle\mathcal{T}_k\rangle_i) = {}^{(\Delta t - \Delta t' + \lceil {}^{i}/_{|\mathcal{G}_k|} \rceil \cdot t_{\mathcal{T}_k})}/_{t_{\mathcal{T}_k}} \tag{25}$$

$$\textstyle\sum_{i=1}^{n} \hat{\mathscr{R}}_M(\langle\mathcal{T}_k\rangle_i) \le \frac{n\cdot(\Delta t - \Delta t' + t_{\mathcal{T}_k})}{t_{\mathcal{T}_k}} + \left\lceil \frac{n(n+1)}{2|\mathcal{G}_k|} \right\rceil \tag{26}$$

Hence, we estimate n as

$$\begin{aligned} n = \text{argmax}_m \big\{ &{}^{N_B}/_{(N_B+m)} \cdot \mathscr{R}_M^B \\ &+ {}^{1}/_{(N_B+m)} \big[{}^{m(\Delta t - \Delta t' + t_{\mathcal{T}_k})}/_{t_{\mathcal{T}_k}} + \lceil {}^{m(m+1)}/_{2|\mathcal{G}_k|} \rceil \big] \\ &< \Gamma_{lb} \big\} \end{aligned} \tag{27}$$

$$\begin{aligned} \therefore \hat{N}_{mig}^k &= \hat{N}_{lo} - n \\ \hat{N}_{acc}^k &= 0 \end{aligned} \tag{28}$$

Case 1-2: $\hat{\mathscr{E}}_M^{CurrTP+\Delta t}(m|\mathcal{G}_k) < \Gamma_{lb}$

In this case, $\mathcal{G}_k$ will not migrate any jobs nor accept any from others, hence $\hat{N}_{mig}^k = 0$ and $\hat{N}_{acc}^k = 0$.

Case 2: $\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k) < \Gamma_{lb}$

Case 2-1: $\hat{\mathscr{E}}_M^{CurrTP+\Delta t}(m|\mathcal{G}_k) \ge \Gamma_{lb}$

In this case, $\mathcal{G}_k$ will not migrate any jobs nor accept any from others, hence $\hat{N}_{mig}^k = 0$ and $\hat{N}_{acc}^k = 0$.

Case 2-2: $\hat{\mathscr{E}}_M^{CurrTP+\Delta t}(m|\mathcal{G}_k) < \Gamma_{lb}$

This case indicates that $\mathcal{G}_k$ is able to accept jobs from others.

$$n = \underset{m}{\text{argmin}}\{\hat{\mathscr{E}}_M^{CurrTP}(n|\mathcal{G}_k) \ge \Gamma_{lb}\} \tag{29}$$

$$\begin{aligned} \therefore \hat{N}_{acc}^k &= n - \hat{N}_{lo} + \big|\{s|s \in \mathcal{G}_k \wedge s.Ocp = 0\}\big| \\ \hat{N}_{mig}^k &= 0 \end{aligned} \tag{30}$$

where s means slave machine in $\mathcal{G}_k$.

A. User Request Partitioning

Task partitioning is a common technique in parallel computing. MapReduce is a good example. It partitions a computation task into a batch of small tasks, and runs them in parallel on multiple machines. Thereby, it scales easily to large clusters of inexpensive commodity computers [16]. Furthermore, some studies have been conducted around designing a new job scheduling module aimed at sharing a MapReduce cluster between users to enable statistical multiplexing [17]. In our study, user requests are partitioned into same-scale unit jobs (i.e. in the same expected execution time on machines of a certain type).

B. Prioritization

Priority measures how important a given unit job to the scheduler. Thereby, a hierarchical prioritization algorithm, Algorithm 3, is proposed, which firstly evaluates the importance of a given user request to the cluster, secondly evaluates the importance of each corresponding unit job to the request, and finally combines them together to get the priority of the unit job. We utilize makespan [18] as a parameter to compute the priority value.

Algorithm 3 Prioritization Algorithm

1: **INPUT**: $\mathcal{R}_l$ and its unit jobs in the current cluster.
2: Compute $M(\mathcal{R}_l)$ and $\hat{M}(\mathcal{R}_l)$.
3: **for each** $\langle\mathcal{T}_i\rangle_k$ (its current group is $\mathcal{G}_h$) **do**
4: **(a)** compute $\hat{M}(\langle\mathcal{T}_i\rangle_k)$,
5: **(b)** compute $\hat{M}_\Sigma^k(\{\langle\mathcal{T}_i\rangle_x\})$ assuming $\langle\mathcal{T}_i\rangle_k$ is about to be executed in $\mathcal{G}_h$ immediately,

$$\hat{M}_\Sigma^k(\{\langle\mathcal{T}_i\rangle_x\}) = t_{\mathcal{T}_h} + \hat{M}_\Sigma(\{\langle\mathcal{T}_i\rangle_x\} \setminus \{\langle\mathcal{T}_i\rangle_k\}) \tag{31}$$

6: **(c)** compute its weight via voting,

$$v_k = \textstyle\sum_{j\in\{1,\dots,N_k\}\setminus\{k\}} \hat{M}_\Sigma^j(\{\langle\mathcal{T}_i\rangle_x\}) \tag{32}$$

7: **(d)** normalize the weight,

$$v_k' = {}^{(v_k - \min\{v_j\})}/_{(\max\{v_j\} - \min\{v_j\})} \tag{33}$$

8: **(e)** compute its priority,

$$\begin{aligned} \langle\mathcal{T}_i\rangle_k.Pri =& {}^{w^B}/_{(w^B + w_{\mathcal{R}_l})} \cdot \mathscr{R}_M^B \\ &+ {}^{w_{\mathcal{R}_l}}/_{(w^B + w_{\mathcal{R}_l})} \cdot v_k' \cdot \hat{\mathscr{R}}_M(\mathcal{R}_l) \end{aligned} \tag{34}$$

where $w^B = \sum_{\mathcal{R}_q \in \mathbb{R}_B} w_q$, and

$$w_{\mathcal{R}_l} = \frac{\min\{\max\{\mathcal{R}_l^{\mathcal{L}}, \min\limits_{\mathcal{R}_q\in\mathbb{R}_B}\{\mathcal{R}_q^{\mathcal{L}}\}\}, \max\limits_{\mathcal{R}_q\in\mathbb{R}_B}\{\mathcal{R}_q^{\mathcal{L}}\}\} - \min\limits_{\mathcal{R}_q\in\mathbb{R}_B}\{\mathcal{R}_q^{\mathcal{L}}\} + 1}{\max\limits_{\mathcal{R}_q\in\mathbb{R}_B}\{\mathcal{R}_q^{\mathcal{L}}\} - \min\limits_{\mathcal{R}_q\in\mathbb{R}_B}\{\mathcal{R}_q^{\mathcal{L}}\} + N_B} \tag{35}$$

9: **OUTPUT**: $\langle\mathcal{T}_i\rangle_k.Pri$.

Algorithm 4 Single Job Scheduling Algorithm

1: **INPUT**: A unit job $\mathcal{R}_l^{\langle\mathcal{T}_i\rangle_j}.Pri \ge \Gamma_{pri}$.
2: **for each** $\mathcal{G}_k$ **do** compute $\hat{\mathscr{R}}_M^\Sigma(\mathbb{U}_{\mathcal{G}_k})$.
3: Find the most appropriate group,

$$\begin{aligned} \mathcal{G}_{best} = \underset{k}{\text{argmin}}\{ &\frac{|\mathcal{G}_k|+1}{2|\mathcal{G}_k|+1} \cdot \hat{\mathscr{R}}_M^\Sigma(\mathbb{U}_{\mathcal{G}_k} \cup \{\mathcal{R}_l^{\langle\mathcal{T}_i\rangle_j}\}) \\ &- \frac{|\mathcal{G}_k|}{2|\mathcal{G}_k|+1} \cdot \hat{\mathscr{R}}_M^\Sigma(\mathbb{U}_{\mathcal{G}_k})\} \end{aligned} \tag{36}$$

4: **OUTPUT**: $\mathcal{G}_{best}$.

C. Single Job Scheduling

Seeing a unit job $\mathcal{R}_l^{\langle\mathcal{T}_i\rangle_j}.Pri \ge \Gamma_{pri}$, the scheduler migrates it to a group in the current cluster where it will have the least impact on the estimated cumulative makespan ratio of the group which is computed as shown in Algorithm 4.

D. Intra-Cluster Load Balancing

When doing the intra-cluster load balancing, the scheduler will try to minimize the sum of *EBMRE*s of groups that need to migrate unit jobs to others. The essence of the intra-cluster load balancing algorithm is to seek a best way to migrate unit jobs from "overwhelmed" groups to "free" ones. The algorithm is shown as Algorithm 5, in which the recursion at line 5 can be done by using dynamic programming.

E. Inter-Cluster Load Balancing

After doing intra-cluster load balancing in a cluster, and if

Algorithm 5 Intra-Cluster Load Balancing Algorithm

INPUT: The set $\mathbb{G}_{mig}$ of groups that $\hat{N}^k_{mig} > 0$, and the set $\mathbb{G}_{acc}$ of groups that $\hat{N}^k_{acc} > 0$.
Let $\mathcal{P}(\langle \mathcal{G}_i, \mathcal{G}_j \rangle) = \hat{\mathscr{O}}^t_M(n_i|\mathcal{G}_i) + \hat{\mathscr{O}}^t_M(n_j|\mathcal{G}_j)$, where $\mathcal{G}_i \in \mathbb{G}_{mig}$ and $\mathcal{G}_j \in \mathbb{G}_{acc}$, n_i is the number of unit jobs waiting in $\mathcal{G}_i$ and n_j is the number of unit jobs, if any, waiting in $\mathcal{G}_j$.
Let $\hat{\mathcal{P}}(\langle \mathcal{G}_i, \mathcal{G}_j \rangle) = \hat{\mathscr{O}}^t_M(n_i - \min\{\hat{N}^i_{mig}, \hat{N}^j_{acc}\}|\mathcal{G}_i) + \hat{\mathscr{O}}^t_M(n_j + \min\{\hat{N}^i_{mig}, \hat{N}^j_{acc}\}|\mathcal{G}_j)$.
Let $\Delta P(\langle \mathcal{G}_i, \mathcal{G}_j \rangle) = \mathcal{P}(\langle \mathcal{G}_i, \mathcal{G}_j \rangle) - \hat{\mathcal{P}}(\langle \mathcal{G}_i, \mathcal{G}_j \rangle)$.
Let $\Delta P_\Sigma(\langle \mathbb{G}_{mig}, \mathbb{G}_{acc} \rangle) = \max_{\substack{\mathcal{G}_i \in \mathbb{G}_{mig} \\ \mathcal{G}_j \in \mathbb{G}_{acc}}} \{\Delta P(\langle \mathcal{G}_i, \mathcal{G}_j \rangle) + \Delta P_\Sigma(\langle \hat{\mathbb{G}}_{mig}, \hat{\mathbb{G}}_{acc} \rangle)\}$, where $\hat{\mathbb{G}}_{mig}$ and $\hat{\mathbb{G}}_{acc}$ are $\mathbb{G}_{mig}$ and $\mathbb{G}_{acc}$ respectively after migrating $\min\{\hat{N}^i_{mig}, \hat{N}^j_{acc}\}$ number of unit jobs from $\mathcal{G}_i$ to $\mathcal{G}_j$.
$\hat{N}^j_{acc} = \hat{N}^j_{acc} - m$ after m unit jobs being migrated to $\mathcal{G}_j$.
if $\hat{N}^j_{acc} \leq 0$ **then**
    $\hat{\mathbb{G}}_{acc} = \hat{\mathbb{G}}_{acc} \setminus \{\mathcal{G}_j\}$.
end if
if $\hat{\mathbb{G}}_{acc} = \emptyset$ **then**
    return a set of migrations $\{\langle \mathcal{G}_i, \mathcal{G}_j \rangle\}$
end if

$\sum_k N^k_{mig}$ is still larger than $\sum_k N^k_{acc}$ by a specified amount, then inter-cluster load balancing will be attempted. The intra-cluster load balancing tries to balance all groups in one cluster, thus the inter-cluster load balancing will focus on the unbalance in the same type of groups between clusters. We use *"heavy"* to describe clusters whose workload is above the average level. Otherwise, we use *"light"* to express the relatively idle ones. The inter-cluster load balancing mechanism will try to find light clusters within a certain range, e.g. one hop. This mechanism is inspired by Gossip-based protocols and the Ant Colony algorithm. Some notations are shown below:

Notation 1. Load ($\mathcal{L}^t_{i,j}$): The total number of executing and waiting jobs in $\mathcal{G}_i$ of a cluster $\mathcal{C}_j$ at a time point t.

Notation 2. Capacity ($\mathcal{N}_{i,j}$): The cardinality of slave nodes in $\mathcal{G}_i$ within a cluster $\mathcal{C}_j$.

Notation 3. Gossip Message (g): A broadcast message generated by a heavy cluster who has migrated a set of jobs out successfully.

A. Measures

Inspired by [19], we use the expected average workload in $\mathcal{G}_i$ of each cluster $\mathcal{C}_j$ as a baseline value, denoted by $\mathcal{E}_{i,j}$. For example, given a time point t and N clusters, the expected average workload can be computed as:

$$\mathcal{E}_{i,j} = (\bar{\mathcal{L}^t_i}/\bar{\mathcal{N}_i} \pm \delta) * \mathcal{N}_{i,j} \tag{37}$$

Here, the $\bar{\mathcal{N}_i}$, $\bar{\mathcal{L}^t_i}$ represent the average capacity and the average workload of all clusters respectively:

$$\bar{\mathcal{L}^t_i} = \textstyle\sum_j \mathcal{L}^t_{i,j}/N \tag{38}$$

$$\bar{\mathcal{N}_i} = \textstyle\sum_j \mathcal{N}_{i,j}/N \tag{39}$$

$\mathcal{E}_{i,j}$ is the expected workload that should be allocated in a group $\mathcal{G}_i$ of cluster $\mathcal{C}_j$ when all the workloads are close to balanced at the time point t. Comparing the actual workload $\mathcal{L}_{i,j}$ to $\mathcal{E}_{i,j}$, if $\mathcal{L}_{i,j} > \mathcal{E}_{i,j}$, $\mathcal{C}_j$ is a *heavy* cluster with respect to $\mathcal{G}_i$, otherwise it is a *light* cluster as shown in Algorithm 6. Without insertion or removal of clusters from the system, the $\bar{\mathcal{N}}$ shall be a constant but the $\bar{\mathcal{L}}$ is a dynamic value. δ is an adjustable parameter. For a heavy cluster, $\mathcal{E}_{i,j}$ takes the upper bound of its value, otherwise, the lower bound. For a given group $\mathcal{G}_i$, the amount of jobs the cluster $\mathcal{C}_j$ (heavy or light) needs to forward or accept is denoted by $\mathcal{U}_{i,j}$:

$$\mathcal{U}_{i,j} = \left\lfloor \left| \mathcal{L}^t_{i,j} - \mathcal{E}_{i,j} \right| \right\rfloor \tag{40}$$

Algorithm 6 Cluster Status Update Algorithm

INPUT: The set of groups $\{\mathcal{G}_n\}$ and clusters $\{\mathcal{C}_m\}$
$\bar{\mathcal{N}_i} = \sum_j \mathcal{N}_{i,j}/N$
for each cluster j to M **do**
    for each group i to N **do**
        Broadcast current load $\mathcal{L}_{i,j}$ to its neighbors
        Records all received broadcast of load
        $\bar{\mathcal{L}^t_i} = \sum_j \mathcal{L}^t_{i,j}/N$, $\mathcal{E}_{i,j} = (\bar{\mathcal{L}^t_i}/\bar{\mathcal{N}_i} \pm \delta) * \mathcal{N}_{i,j}$
        if $\mathcal{L}_{i,j} > \mathcal{E}_{i,j}$ **then**
            Mark cluster $\mathcal{C}_j$ as heavy
        else
            Mark cluster $\mathcal{C}_j$ as light
        end if
    end for
end for

B. Mechanism

The objective is that of migrating jobs from heavy clusters to light ones to keep the workloads across the system balanced. The migration decisions are made based on gossip messages, which are generated and broadcast by heavy clusters who transfer jobs out successfully. One gossip message includes: the target light cluster's address, the group type, its available computing capacity and the life time of this message. It can be deemed as a positive gossip message to guide the job migration

Algorithm 7 Inter-Cluster Load Balance Algorithm

INPUT: The set of heavy groups $\{\mathcal{G}_k\}$, and the set of corresponding clusters $\{\mathcal{C}_m\}$
for Each type of Group $\mathcal{G}_i$ **do**
    for Each Cluster $\mathcal{C}_j$ **do**
        while $\mathcal{U}_{i,j} \geq 0$ and one light cluster can be found in the gossip table **do**
            Traverse gossip table.
            Pick the most frequent cluster as the target cluster, say $\mathcal{C}_{tar}$, and compute its $\mathcal{U}_{i,tar}$
            if $\mathcal{U}_{i,j} - \mathcal{U}_{i,tar} \leq 0$ **then**
                Forwards $\mathcal{U}_{i,j}$ from $\mathcal{G}_{i,j}$ to $\mathcal{G}_{i,tar}$
                Update $\mathcal{U}_{i,tar} \leftarrow \mathcal{U}_{i,tar} - \mathcal{U}_{i,j}$
                Broadcast one gossip message g with this target cluster $\mathcal{C}_{tar}$, and update the gossip table.
            else
                Forward $\mathcal{U}_{i,tar}$ to $\mathcal{G}_{i,tar}$
                Update the $\mathcal{U}_{i,j} \leftarrow \mathcal{U}_{i,j} - \mathcal{U}_{i,tar}$
                Update the gossip table
            end if
        end while
    end for
end for

TABLE I
THE MACHINE GROUP COMPUTING CAPACITY

Group	Execution Time(millisecond)
$\mathcal{G}_1$	50
$\mathcal{G}_2$	250
$\mathcal{G}_3$	1250
$\mathcal{G}_4$	6250

in the system. The algorithm is shown in Algorithm 7.

IV. EXPERIMENTATION

A. Settings

We consider one cloud provider managing four clusters, with four types of machine groups each. We utilize four clusters since it is a reasonable estimation of the number of clusters that one cloud provider can build and operate. With the four groups, $G = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$, represent extra fast, fast, medium and slow machine groups respectively. And the different computing capacities in these four groups are reflected in the execution time settings, shown in Table I.

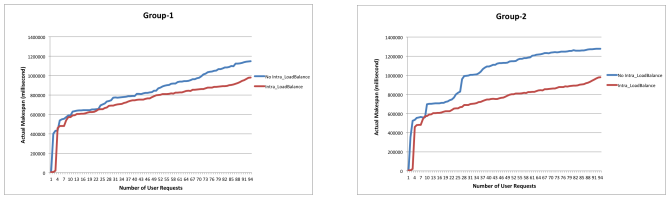
The other parameters used in our experiments are as follows: $\lambda = 20$ is the mean value of the number of random arrival user requests, which follows the Poisson Distribution. In the system, the new coming user requests are generated continuously and the arrival interval follows the exponential distribution. The requests for the four different groups are generated randomly. Meanwhile the number of unit jobs each request may be partitioned into is also generated randomly and limited within the range of $[1 \sim 20]$. When doing inter-cluster load balancing, we define a communication protocol for the inter-clusters communication based on UDP. There are mainly two kinds of communications between pairs of clusters: one for inter-cluster load balancing request and reply, another for realizing jobs transmission. And we use the survive time for one UDP, which is how long this message has been generated, to measure its reliability. Each time doing inter-cluster load balancing, clusters tends to follow the most reliable UDP to make transmission decision.

B. Experimental Results

Intra-Cluster load Balancing We compare the average actual user request makespan over four groups, $M(\mathcal{R}_l)$, between the case with the intra-cluster load balancing and the case without it. The experimental results are shown in Figure3.

After deploying the intra-cluster load balancing, the $M(\mathcal{R}_l)$ is growing more slowly in all the four groups, and the average of $M(\mathcal{R}_l)$ decreases significantly. All the four groups are running smoothly and the entire cluster keeps balanced better than the original case where no intra-cluster load balancing is involved.

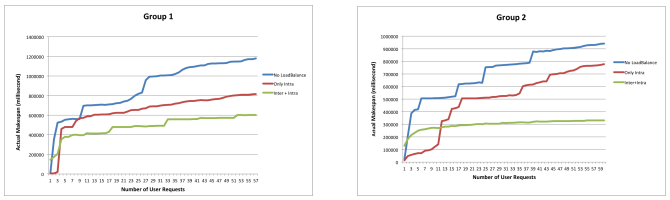
Inter-Cluster load Balancing We still use the average actual user request makespan, $M(\mathcal{R}_l)$, as the performance measure to make comparison over the four groups between the case with inter-cluster load balancing and the case without inter-cluster load balancing.



(a) Group 1 (b) Group 2

(c) Group 3 (d) Group 4

Fig. 3. The actual User Request Makespan with intra-cluster load balancing(red) vs actual User Request Makespan without intra-cluster load balancing(blue)



(a) Group 1 (b) Group 2

(c) Group 3 (d) Group 4

Fig. 4. Comparison on actual user requests makespan of without inter or intra-cluster load balancing(blue) vs inter-cluster load balancing(red) vs intra-cluster load balancing (green).

The experimental results in Figure4 show that our intra-cluster load balancing and inter-cluster load balancing can keep the entire system balanced effectively, and prevent the average actual user request makespan to increase significantly. We use the average actual user request makespan without intra- or inter-cluster load balancing as the baseline. When the intra-cluster load balancing is deployed, the average acutal makespan decreases around 20% on average compared to the baseline, and the maximum decrease is up to 35%. When the inter-cluster load balancing is added, the average actural makespan decreases more significanlty compared to the baseline. The decrease is around 50% on average, and the maximum decrease is up to 65%. Moreover, the intra- and inter-cluster load balancing will make the average actural makespan increase much more slowly than the case without intra- or inter-cluster load balancing does along with the increasing of the number of incoming user requests.

V. Conclusion

Our hybrid cloud system combines the centralized organization and Peer-to-Peer network hierarchically, and thus integrates their advantages. Our system gives a flexible method to balance the workload across the entire system via the intra-cluster and inter-cluster load balancing and also can avoid the singleton failure problem. Moreover, we propose a priority algorithm for individual job scheduling. Our simulation results show that the proposed system can execute user requests with lower Makespan. In so doing each cluster is internally balanced, and there is an effective cooperation between clusters across the entire system.

VI. Future Work

In this paper, we presume that our hybrid system is deployed by a single cloud provider who manages multiple clusters. However, in the future work, we will take the federation of multiple cloud providers into consideration, in which the benefit maximization problem will become more complicated because the cost and the expected revenue are varying in different providers. Therefore, we will add a feasible profit sharing mechanism into our system to get a win-win cooperation, and aimed at maximizing the social welfare as well as minimize the average user requests makespan and keep system balanced.

On the other hand, our inter-cluster load balancing algorithm relies on unit jobs transmitting between clusters to keep the workload balanced, which tends to local balance whereas global unbalance, especially when the Peer-to-Peer network is expanded to large scale. Moreover, it is inevitable that the cloud system is growing towards large-scale and heterogeneity, and thus the importance of cooperation between providers will be more critical. Therefore, we are improving the inter-cluster load balancing mechanism to make it more feasible and efficient in a large-scale cloud network.

Acknowledgment

We would like to thank Dr. Lena Mashayekhy, whose class introduced us to cloud computing and whose encouragement initiated this paper. We also thank Dr. Errol Lloyd who has been a consistent and positive sources of ideas, perspective, and encouragement.

References

[1] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[2] S. S. Iyengar and R. R. Brooks, *Distributed Sensor Networks: Sensor Networking and Applications*. CRC press, 2012.

[3] P. Verissimo, A. Bessani, and M. Pasin, "The tclouds architecture: Open and resilient cloud-of-clouds computing," in *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*. IEEE, 2012, pp. 1–6.

[4] O. Babaoglu, M. Marzolla, and M. Tamburini, "Design and implementation of a p2p cloud system," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 412–417.

[5] P. Mayer, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, R. Pugliese, J. Keznikl, and T. Bures?, "The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing," in *Self-Adaptation and Self-Organizing Systems Workshops (SASOW), 2013 IEEE 7th International Conference on*. IEEE, 2013, pp. 89–94.

[6] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 351–364.

[7] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1393–1404, 2014.

[8] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 53.

[9] H. CONSTANTIN, "Markov chains and queueing theory," 2011.

[10] T. L. Saaty, *Elements of queueing theory*. McGraw-Hill New York, 1961, vol. 423.

[11] A. O. Allen, *Probability, statistics, and queueing theory*. Academic Press, 2014.

[12] T. Li, M. Chen, D.-M. Chiu, and M. Chen, "Queuing models for peer-to-peer systems."

[13] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 5, pp. 936–943, 2012.

[14] 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 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 3.

[15] A. Tumanov, J. Cipar, G. R. Ganger, and M. A. Kozuch, "alsched: Algebraic scheduling of mixed workloads in heterogeneous clouds," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 25.

[16] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in *OSDI*, vol. 8, no. 4, 2008, p. 7.

[17] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55*, 2009.

[18] E. G. Coffman and J. L. Bruno, *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.

[19] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured p2p systems," in *Peer-to-Peer Systems II*. Springer, 2003, pp. 68–79.