Fairly Sharing the Network for Multitier Applications in Clouds

Xiaolin Xu, Huazhong University of Science and Technology, Wuhan, China

Song Wu, Huazhong University of Science and Technology, Wuhan, China

Hai Jin, Huazhong University of Science and Technology, Wuhan, China

Chuxiong Yan, Huazhong University of Science and Technology, Wuhan, China

ABSTRACT

A significant trend caused by cloud computing is to aggregate applications for sharing resources. Thus, it is necessary to provide fair resources and performance among applications, especially for the network, which is provided in the best-effort manner in current clouds. Although many studies have made efforts for provisioning fair bandwidth, they are not sufficient for network fairness. In fact, for interactive applications, response time is more sensitive than bandwidth, and users expect a fair response time not just bandwidth. In this study, the authors want to investigate whether the traditional methods of sharing bandwidth can help the fairness of response time. They show that: (1) bandwidth has little relationship to response time, and adjusting bandwidth hardly affects response time in most cases. Thus, the traditional methods cannot help the fairness of response time much; and (2) the fairness between components is different from the one between transactions, and many prior studies only consider the former while ignoring the latter. Thus, the authors cannot help much for multitier applications consisting of multiple transactions either. As a result, they construct a model with two metrics to evaluate the fairness status of the network sharing, while considering the applications' characteristics on both the response time and throughput. Based on the model, they also propose a mechanism to improve the fairness status. The evaluation results show that the authors' mechanism improves the fairness status by 26.5%–52.8%, and avoids performance degradation compared to some practical mechanisms.

Keywords: Cloud Computing, Fairness, Multitier Application, Network, Performance, Response Time

INTRODUCTION

In cloud environment, many applications are aggregated to share resources. Meanwhile, sharing leads to potential resource contentions, such as CPU contention (Li et al., 2012; Rao, Wei, Gong, & Xu, 2013), I/O contention (Rao et al. 2013), and network contention (Bourguiba, Haddadou, El Korbi, & Pujolle, 2014; Singh, Shenoy, Natu, Sadaphal, & Vin, 2011). Owing to the *best-effort* provisioning manner of cloud networks, the network contention may cause significant

DOI: 10.4018/IJWSR.2015100103

unfair sharing of network resources among applications. In fact, some prior studies have made efforts to guarantee the fairness of network bandwidth (Guo, Liu, Zeng, Lui, & Jin, 2013; Popa et al., 2012; Wei, Vasilakos, Zheng, & Xiong, 2010), providing bandwidth fairly to applications. However, it is not sufficient to share networks fairly by considering the bandwidth only.

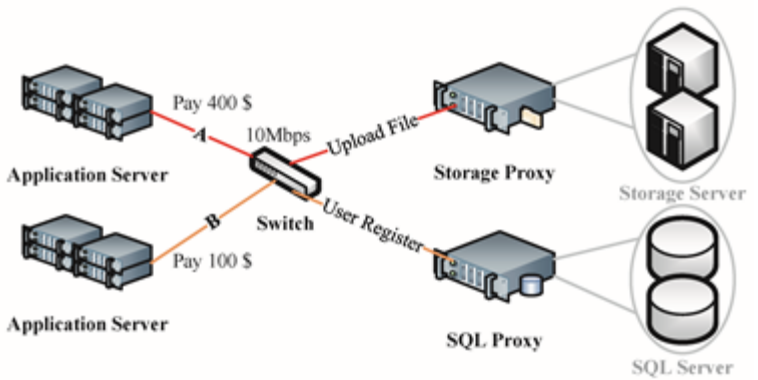
Many interactive applications in clouds concern not only the bandwidth, but also the response time, which would be experienced directly by end-users. The owners of such applications expect fair response time, rather than the bandwidth. In fact, they may consider it unfair when their applications express bad response time, even if the applications are guaranteed with fair bandwidth. Thus, we should guarantee the fair response time as well. Naturally, we investigate whether the traditional methods (Guo et al., 2013; Popa et al. 2012; Shieh, Kandula, Greenberg, Kim, & Saha, 2011) can share the network fairly for multitier applications, considering not only the bandwidth but also the response time.

First, can the methods for sharing bandwidth help the response time? Unfortunately, these works do not help the fairness of response time much. Intuitively, they consider an application with more bandwidth obtaining better performance. As shown in Figure 1, assuming application *A* pays 4 times as much as *B*, *A* gets 8Mbps bandwidth of the link and *B* gets 2Mbps. However, according to the queuing theory, packets of *A* going through this link would cost the same time as packets of *B,* even though *A* gets more bandwidth (under FIFO scheduling policy). In fact, the response time is mainly affected by the forwarding ability, an inherent attribute of switches that is unchangeable, and the link load, the aggregated load transferring over that link. Although the link load can be changed by adjusting bandwidth, and the change may potentially affect response time, we will show that such effect is non-significant or even unwanted.

Second, can the methods toward components help multitier applications? Unfortunately, the traditional methods that focus on the fairness toward components cannot help multitier applications either. In practice, on one hand multitier applications contain multiple transactions, and on the other hand, the cloud contains many multitier applications. This causes a lot of transactions with different degrees of importance mixed together. Only considering the components ignores the requirements of transactions, and causes unfair performance for applications.

In this paper, we focus on the fairness of network sharing for multitier applications. We first analyze potential problems if considering requirements of response time and transactions. Then we construct a model to describe the fairness of network sharing, considering both response time and throughput. Two metrics in this model evaluate the fairness status of the cloud, indicating

Figure 1. Two sample transactions sharing a switch link



the fairness degree and the average performance, respectively. Based on the model, we propose a scheduling mechanism to improve the fairness status and reduce the performance degradation. The evaluation results show that, our method can improve fairness by 26.5%–52.8%, compared to some practical mechanisms, and avoid performance degradation or even improve performance.

In summary, we make the following contributions in this paper:

- We analyze the difficulties of network fairness if considering response time, and explain why the traditional ways cannot help much;
- We construct a model to describe the fairness of network sharing, and build two metrics to evaluate the fairness status of the cloud;
- We propose a mechanism to improve fairness status of the cloud and avoid performance degradations.

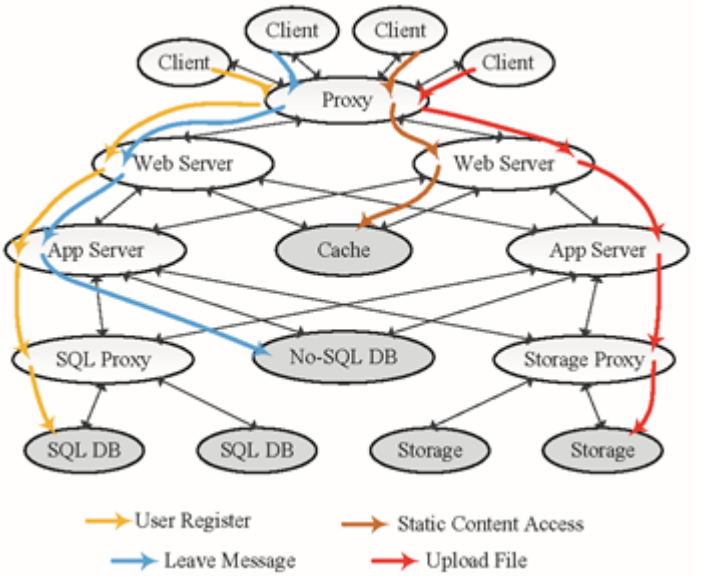
PROBLEM STATEMENT

In this section, we analyze potential problems of a fair sharing network if consider response time and transactions, and explain why the traditional ways cannot help much.

Response Time and Bandwidth

Generally, response time is the time interval between submitting a request and receiving the response. Take the *User Register* transaction of a simplified web application, shown in Figure 2 as an example, its response time begins at the request entering the *Proxy*, along the yellow path to the *SQL DB*, then back to the *Proxy*, and ends at returning the respond to the *Client*. In fact, the response time consists of the latency of multiple network links. Although the latency of a link

Figure 2. A simplified web application with core components and transactions



is composed of multiple factors, such as the processing time, queuing time, transmission time, and propagation time (Anonymous, 2013), here we only focus on the queuing time, because it is more sensitive than other factors in a congested cloud network.

We investigate whether the traditional methods of guaranteeing fair bandwidth help the response time from two perspectives. First, is the latency of one link determined by the bandwidth of that link? Second, which is more challenging; can controlling the bandwidth of one link help to guarantee the fairness of latency over that link? For the first question, considering the M/M/1 queuing model with formula $W=1/(\mu-\lambda)$. W denotes the staying time of a packet in the switch, λ denotes the requests' arriving rate, and μ denotes the service rate of switch. We can see the latency is only affected by the load of the switch and forwarding ability. By controlling the bandwidth, we can limit the load of that switch, and thus affect the latency. As a result, *the latency is not determined by the bandwidth, but may be affected by it indirectly*.

For the second question, as controlling bandwidth may affect latency, does it possibly help to guarantee the fairness of latency by assigning desirable bandwidth to applications? We analyze it with regard to two types of links: multiplexed links and independent links. For the scenario of multiple logical links sharing a physical link, we take the physical link connected through *Switch* in Figure 1 as an example. It is shared by the logical link of *App Server-SQL Proxy* in *User Register* transaction, which is latency sensitive, and the logical link of *App Server-Storage Proxy* in *Upload File*, which is non-latency sensitive. Reasonably, we should adjust the bandwidth to provide the former one with a better latency than the latter one. However, because of the sharing, they both contribute their throughputs to the same switch, and their latencies are impacted by the sum of throughputs simultaneously, rather than their own throughputs. According to the queuing theory, these two logical links always have equal latency, no matter how we change the bandwidth assignment. In other words, although controlling bandwidth can adjust the throughputs, the latencies of these two links are always the same, which violates their characteristics. In practice, PS-L (Proportional Sharing at Link-level) (Popa et al., 2012) is a popular allocation policy, which provides the bandwidth fairness by assigning different weight to each logical link. Based on above analysis, the fairness of latency cannot be fulfilled by only adjusting the bandwidth weight. Thus, *it hardly guarantees the fairness of latency on multiplexed links if the logical links have different demands*.

For the independent scenario that transactions take different physical links, we take the link *Web Server - App server* in the transaction *Leave Message* and the link *Web Server - Cache* in *Static Content Access* as an example. If both switches have same forwarding ability and both transactions require same latency, we should guarantee the throughputs of two links at the same level, according to the relation of latency and throughput. Unfortunately, if the original throughput of *Leave Message* is low while *Static Content Access* is high, the adjusted throughput of the latter one will be reduced greatly, to make it equal to the former one. As a result, it extremely impairs the performance of the high throughput transaction for a seemingly fair latency, and furthermore, it greatly decreases the resource utilization of the cloud. Therefore, *it is unreasonable to guarantee the fairness of latency by controlling bandwidth, even for independent links*.

In summary, we cannot directly guarantee the fairness of latency (or response time) by simply controlling bandwidth. It is noticeable that we focus on the FIFO scheduling strategy in switches, although the WFQ scheduling strategy can provide independent queues for different virtual links. In a cloud, a physical link can be multiplexed by many flows, which has a risk of approaching the upper bound of flow limitation of the WFQ. Meanwhile, the processing complexity of WFQ is very high as pointed out by Bennett and Zhang (1996), which is enlarged in the cloud owing to the high dynamic nature.

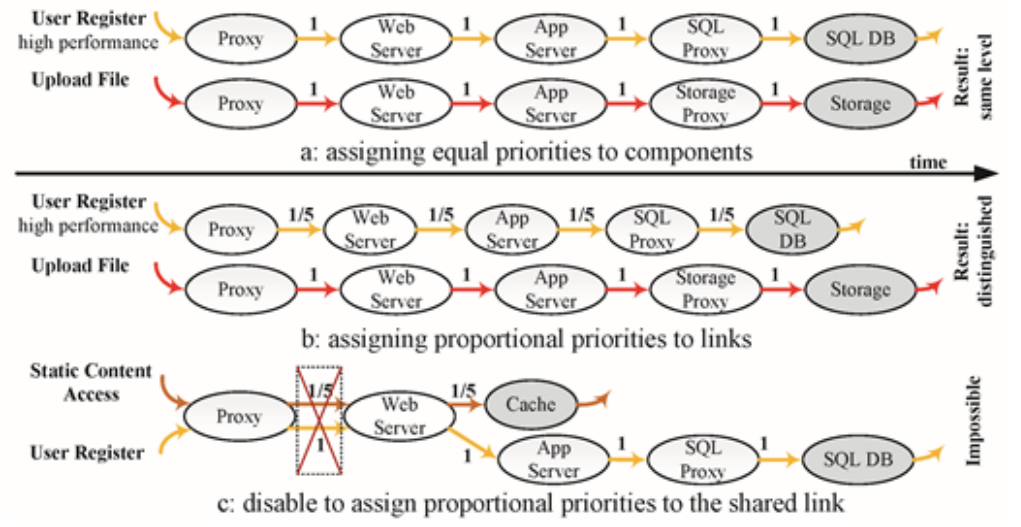
Transactions and Components

Multitier applications consist of many transactions. These transactions have different requirements according to their own characteristics. In fact, ignoring transactions' requirements may cause significant problems for the fairness.

First, assigning equal priority to components cannot guarantee the fairness for transactions. Traditionally, managers may assign priorities directly to components if they are unaware of transitions. Thus, the only way of guaranteeing the fairness in such situation is to assume and provide equal priority to all components. Unfortunately, such a method cannot distinguish the different requirements of transactions. Taking the transaction *User Register* and *Upload File* in Figure 2 as an example, the former requires high-level performance as it is an interactive task, while the latter does not. Ideally, all the links are guaranteed with equal priority, and thus have almost the same link latency. In this way, a request of *User Register* will go through four links among *Proxy*, *Web Server*, *App Server*, *SQL Proxy* and *SQL DB*, and its aggregated latency is about 4 times the link latency. Meanwhile, the *Upload File* gets about 4 times the link latency as well. As a result, these two transactions get the same level of response time, which obviously violates the initial demands, as shown in Figure 3a. Therefore, it is necessary to map requirements of transactions to components for actual fairness.

Second, mapping requirements proportional to links cannot guarantee the fairness for transactions as well. Someone might indicate that if we adjust the weights of User Register and Upload File to a very big ratio, such as 5:1, then the User Register would enjoy a better response time than the Upload File, thus their requirements would be distinguished and satisfied. Actually, this is feasible. The proportional mapping can make the transactions well on independent links, as shown in Figure 3b. However, it is uncommon that many independent links exist. In fact, many transactions share links, such as Static Content Access and Upload File. Since these two transactions share the link between Proxy and Web Server, the mapping is useless for that link, as shown in Figure 3c. An alternative is to only adjust unshared links. However, as the physical machines are multiplexed, the links of one transaction almost always have neighbors, thus it is nearly impossible to find an absolutely unshared link.

Figure 3. Possible ways of mapping requirements



FAIRNESS MODEL

We construct the model for a fair sharing network through two critical steps. The first is to build metrics to evaluate fairness status. The second is to map requirements, enabling the fairness metrics to fully express the requirements of transactions.

Fairness Metrics

Our goal is to guarantee fairness towards the whole cloud. Thus, we need proper metrics to evaluate the global fairness status. When we design fairness metrics, the first consideration is that which conditions should be regarded as unfair. A natural idea is that, once the performance status of a component has deviated far from the average performance of all components, then this component should be considered an exceeded or insufficient performance level, which means an unfair scenario. Based on this idea, we first design a performance metric as the basis, and then design fairness metrics towards the whole cloud.

Performance Metric toward Components

Generally, the performance status of a component is related to three core factors: priority, latency, and throughput. The priority is considered as the weight of the component for a certain level of performance. It can be defined in many ways, but for simplicity, we will just think of it as the payment of components. Thus, a larger priority means more payment, which should be provisioned with better performance. Meanwhile, the latency and the throughput can directly indicate the performance level of components; a component with lower latency and higher throughput definitely obtains a better performance status. Thus, logically, the performance status of components is negative for the priority and response time, while positive for the throughput.

Formally, let F be the performance metric toward components, H denotes the throughput, P denotes the performance priority, and L denotes the latency (although the latency is applied to a link rather than a component, we use it here just like the average latency of all the links that contain this component). Considering the relations between above factors, if we ignore the constant term, F can be expressed as:

$$F = P \times L / H$$

This is very similar to the *Power* (Peterson & Davie, 2007), a most accepted indicator of link ability, described as *Power=Throughput/Latency*. The formula implies that a stronger *Power* needs higher *Throughput* and lower *Latency*. Borrowing the concept of *Power*, we have *F=P/Power*. If we minimize the gap between F_A and F_B ($F_A \approx F_B$), we achieve:

$$\frac{Power_A}{Power_B} \approx \frac{P_A}{P_B}$$

which is a desirable fair situation. We use to express the performance status of a component. Ideally, if the cloud is absolute fair, the performance status of all components should be equal to each other.

Fairness Metrics toward the Cloud

To guarantee the fairness of the whole cloud, an intuitive thought is that we should reduce the deviation of performance status between components. In another words, we want to keep performance status of any component around the average status. Thus, we adopt the deviation of the performance status of all components as a metric to denote the fairness status of the cloud.

Meanwhile, from a different perspective, the average performance status of all components can express the performance status of the cloud. Intuitively, a lower value of the average indicates better performance level, while a higher one indicates a worse performance level, which means we degrade the performance when we increase the fairness. Thus, we adopt the average of performance status of all components as the other metric to denote the performance status of the cloud.

Average latency is a widely used metric to describe the performance of the network with multiple links (Radonjic & Radusinovic, 2010). In our paper, transactions are assigned with weights according to demands of cloud users. Thus, formally, we define $\overline{F}$ as the average performance status, which we call the *Weighted Mean Performance (WMP)* of the cloud. According to the statistic theory, we get, in which, F_c denotes the fairness metric of component c, and count(c) is the function to calculate amount of the components:

$$\overline{F} = \sum_{c \in C} F_c \,/\, count(c)$$

Then, we define *SD* as the standard deviation of the performance status:

$$SD = \sqrt{\sum_{c \in C} \left(F_c - \overline{F}\right)^2 \,/\, count\left(c\right)}$$

which reflects the similarity of the performance status between components. As mentioned before, a low *SD* means the performance status of all components almost standing at the same level; i.e. the fairness is well guaranteed.

However, we do not directly use *SD* as the fairness metric, because when we compare the fairness status guaranteed by different strategies or in different clouds, the average value may lead to an unreasonable comparison. Thus, in order to eliminate side effects on the average values, we adopt the *coefficient of variation* (Anonymous, 2014) of the performance status as the fairness metric of the cloud, which we call the *Fairness Coefficient of Variation (FCV)*. Let *CV* denote it as:

$$CV = \frac{SD}{\overline{F}} = \frac{1}{\overline{F}} \sqrt{\sum_{c \in C} \left(F_c - \overline{F}\right)^2 \,/\, count(c)}$$

Based on the above metrics, we can describe the fairness status from two aspects. However, until now, we have not considered the requirements of transactions. For a more precise description of the fairness metrics, we describe how to map requirements of transactions to related components in the next section.

Mapping Transactions' Requirements

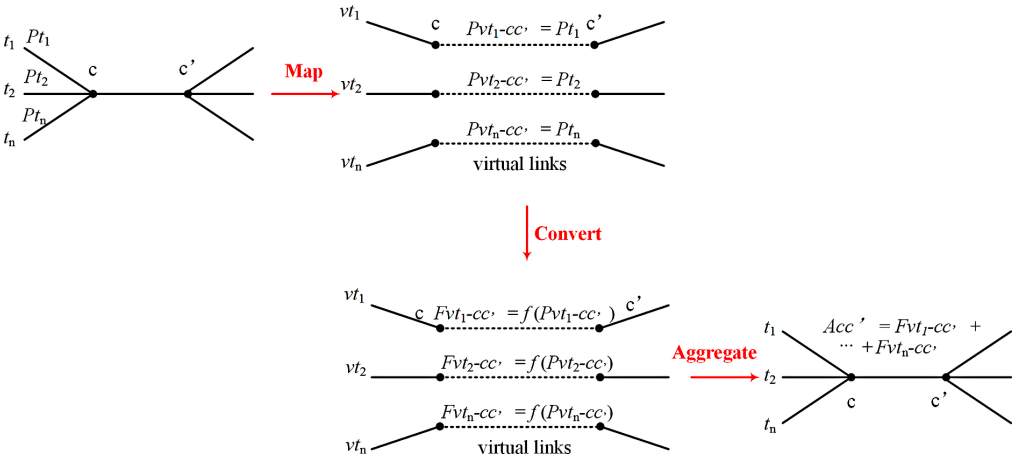
Generally, different transactions may have different requirements of performance, each of which would like to be assigned with a priority to express its requirement (the priority can be proportional to its payment). These priorities need to be mapped to related components, thus the manager can schedule the components properly, to maintain their performance at a certain level. However, the requirements require a fine-grained mapping policy instead of the course-grained ones described in the section of *Problem Statement*. Our solution is shown in Figure 4. The key is to resolve the requirements mapping on shared component pairs. Since many transactions may share the same component pair, and such components may have different priorities at the same time, we need an approach to aggregate these priorities to just one, as the final priority of the component. We first break down the requirements of the transactions to related component pairs, thus each component pair has multiple temporary requirements that we call subrequirements. Then we convert the subrequirements to corresponding temporary fairness metrics that we call subfairness. Finally, we aggregate the subfairness to form the final fairness metric of each component.

Break Down Requirements

As a physical link may be multiplexed by many transactions as in Figure 1, logically we divide it into multiple *virtual links*, each of which correlates to a distinct transaction. Intuitively, the entire throughput of one transaction is transmitted via the related virtual link, thus the requirement of this transaction should be expressed on this link as well. As a result, the transactions that multiplex a component pair will dispatch their requirements to the virtual links of this component pair separately, as shown in the *Map* process in Figure 4.

Formally, let T and C be the sets of transactions and components, and cc' be the component pair comprised of $c, c' \in C$. For the transaction t that communicates through cc', a virtual link should be created, marked as $v_{t\_}cc'$. For each t, a priority P_t is assigned as its requirement. Owing to arbitrary virtual link $v_{t\_}cc'$ should express the requirement of transaction t, so we simply set the priority of $v_{t\_}cc'$ to be equal to the priority of t:

Figure 4. Requirements mapping process for one pair of components



$$P_{v_t\_cc'} = P_t$$

Thus, the requirements related to *cc'* are broken down to multiple subrequirements of the virtual links of *cc'*. In fact, along with breaking down the requirements, the fairness metric of a component is also broken down to many subfairness metrics of the virtual links. Let $F_{v_t\_cc'}$ be the subfairness metric of v_t*_cc'*. Similar to, it is a function of $P_{v_t\_cc'}$, $L_{cc'}$ the latency of the link between the component pair *cc'*, and $H_{v_t\_cc'}$ the throughput of the virtual link v_t*_cc'*. Thus:

$$F_{v_t\_cc'} = P_{v_t\_cc'} \times L_{cc'} / H_{v_t\_cc'}$$

By now, after the *Convert* process in Figure 4, we get the subfairness metrics of virtual links of component pair *cc'*, then we aggregate them together to obtain the final fairness metric of *c*.

Aggregate Subfairness

Forming the final fairness metric of one component requires knowing all the subfairness metrics related to this component. In other words, we need to find all the component pairs that contain this component, and all the transactions that communicate through these component pairs. In fact, we can easily get this information from the application topology, or by monitoring the communication patterns if unaware of topology.

Let $A_{cc'}$ denote the aggregation result of the subfairness for *cc'*, and *T_cc'* denotes the set of the transactions that share the component pair *cc'*. According to, we know the subfairness for each $t \in T\_cc'$ is $F_{v_t\_cc'}$, then as the *Aggregate* process in Figure 4, we get $A_{cc'} = \sum_{t \in T\_cc'} F_{v_t\_cc'}$. In addition, let C' denote the set of the components that communicate with the component *c*, and A_c denotes the aggregation result of the subfairness for *c*. Thus, $A_c = \sum_{c' \in C'} A_{cc'}$.

However, A_c cannot directly express the fairness metric of a component, because the simple summation of submetrics may give more benefit to the component contained in more transactions. Thus, we divide it by the amount of related virtual links:

$$F_c = A_c / count\left(v_t\_cc'\right)$$

in which, count(v_t*_cc'*) is the function that provides the amount of virtual links related to component *c*. According to the above formulas, we get the final fairness metric of *c* as:

$$F_c = \frac{1}{count(v_t\_cc')} \sum_{c' \in C'} \sum_{t \in T\_cc'} P_t \times L_{cc'} / H_{v_t\_cc'}$$

We use to calculate fairness metrics for all components. Intuitively, it relates to the amount and the priorities of the transactions that contain this component, as well as the latency and

throughput of the component pairs. If a component relates to more transactions with high priority, it would be provisioned with low response time or high throughput.

GUARANTEE THE FAIRNESS

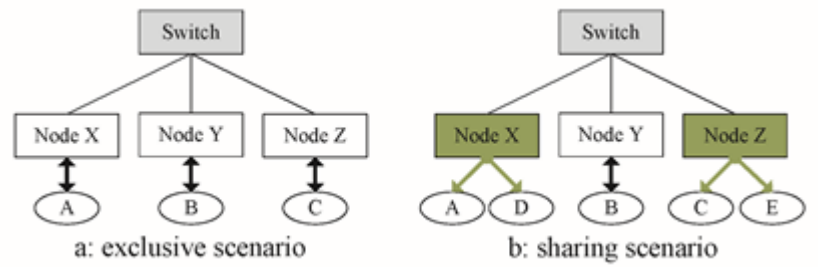
We propose a mechanism to schedule components for guaranteeing the fairness. The idea of this mechanism is simple. We only give components reasonable placements, by which they can get a proper performance level. The mechanism contains three steps of *component consolidation*, *priority-ability match*, and *set affinity match*. The first step aggregates components according to their communication patterns, and the following two steps provide solutions for component placement, with regard to two typical scenarios, respectively.

Components Consolidation

Usually, some components in the same application have intensive communications. Thus, scheduling these components simultaneously and placing them on nearby locations can improve performance and reduce network pressure significantly (Shrivastava et al., 2011). We call these components with strong dependences a *dependence set (DS)*. Unfortunately, in many prior studies, the basic unit of scheduling is an independent component. If a pair of components with strong dependence is separated by a long distance, it will cause high latency, and thus decrease the performance of transactions. Moreover, since the communication between the components finally acts on the physical equipment, the longer distance will lead to expensive costs and heavy global network loads.

Therefore, we propose consolidating components with strong dependences into a DS, and make it the basic unit for scheduling. We think it can obviously improve the performance. In practice, we aggregate components by judging their communication throughput with a threshold. If the throughput exceeds the threshold, we merge them into a set. This procedure can be formulated as following: For any component T_i, we initialize the dependence set DS_i that contains only this component. Afterwards, for any pairs of component T_i and T_j, where $T_i \in DS_i$ and $T_j \in DS_j$, if the throughput between them exceeds the threshold, we merge the two dependence sets DS_i and DS_j into a new DS. Owing to the consolidation, there are two possible scenarios: the *exclusive scenario* and the *sharing scenario*. In the exclusive scenario, the DS occupies a whole node by itself, as in Figure 5a, where *A*, *B*, and *C* occupy the nodes of *X*, *Y*, and *Z*, respectively. In the sharing scenario, as in Figure 5b, *A* and *D* share node *X*, and *C* and *E* share node *Z*. For these two scenarios, we introduce corresponding approaches for improving the fairness status.

Figure 5. Dependence sets in two possible scenarios



Scheduling in the Exclusive Scenario

In the exclusive scenario, resources are sufficient for dependence sets, thus each of them can occupy a location exclusively, even though we still need to choose which location is best for dependence sets.

Since the loads of one dependence set would not interfere the locations occupied by other dependence sets, we just match them to the location that provides the best performance at that moment. On one hand, for the same dependence set, a location that transmits packets faster can provide better performance than the one that transmits packets more slowly. On the other hand, for the same location, a dependence set with higher priority can utilize its capacity more efficiently than the one with lower priority. Thus, for multiple dependence sets in the exclusive scenario, we schedule them simultaneously by matching the one with higher priority to the location with better forwarding ability, in order to improve the performance and the utilization maximally. We call this strategy the *priority-ability match*. The procedure can be fulfilled in three steps. First, sort all dependence sets according to their priorities. Second, sort all nodes according to their forwarding abilities. At last, map the sets to the nodes at the sorted order and try best to let dependence sets are evenly scheduled to all nodes.

Since resources are sufficient in the exclusive scenario, the sets could always get free nodes. However, this does not exist in the sharing scenario, thus we introduce another strategy as below.

Scheduling in the Sharing Scenario

In the cloud, many locations have to host more than one dependence set. Once the resources cannot satisfy all dependence sets simultaneously in the exclusive situation, part of dependence sets should be scheduled to the same location. Thus, we need to place proper dependence sets together, while ensuring the expected fairness and performance.

Simply speaking, we suggest scheduling the dependence sets with similar priority-throughput ratio into the same location, which we call the *set affinity match*. The characteristic of set affinity is that if the sets have more similar priority-throughput ratios, they have a higher affinity. In order to explain more precisely, we give four dependence set samples, which are: A with high priority and low throughput; B with high priority and high throughput; C with low priority and high throughput; and D with low priority and low throughput. Their priority-throughput ratios θ are assumed as $\theta_A > \theta_B \approx \theta_D > \theta_C$.

For A with high priority and low throughput, the system should provide it with a low L_{cc}. If A shares resources with B or C that have high throughput, the loads of the shared location may increase dramatically, which increases the L_{cc} and F_c of A. Therefore, sharing resources with high throughput dependence sets seriously impairs the performance of A. If A shares resources with D, they may get similar load increment ΔH and thus a similar latency increment ΔL. However, since the priority of A and D differs significantly, it still leads to an unbalanced fairness increment ΔF. As a result, the similar ΔF can only be achieved when A shares resources with sets such as A.

Thus, sharing resources between the dependence sets with similar set affinities can guarantee the performance and fairness efficiently; that is, the dependence sets with high set affinity should be scheduled together in the sharing scenario.

EVALUATION

We perform the evaluation in two steps. First, we verify the subprocesses of our strategy by three small-scale experiments. Second, we compare our strategy with some practical strategies using

a large-scale simulation. In each experiment, we evaluate the fairness metrics of FCV and WMP described in the section of *Fairness Model*, in which, a lower FCV means better overall fairness, while a lower WMP means better network performance. In all simulations, we set the resource demand for each component (VM) with 500MB memory and 2 CPU cores, while the capacity of each node has 4GB memory and 16 CPU cores. Thus, each node can host 8 VMs at most.

Basic Experiments

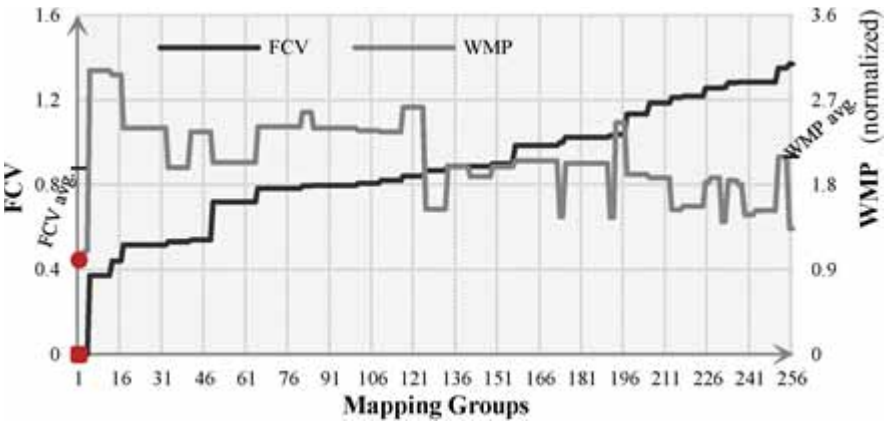
Components Consolidation

In this experiment, we evaluate the effectiveness of the *component consolidation* process. We emulate two homogeneous nodes *A* and *B* with forwarding ability of 10000 (packets/s), and two dependence sets *a* and *b* with four homogeneous components in each of them. The throughput of the components in same set is configured as 100 (packets/s), with the fairness priority of 1. We perform the permutation of the mapping between components and nodes, and compare the results of all possible placements. Since each component can be deployed on either *A* or *B*, there are 256 possible mappings in all for 8 components.

According to our analysis, it is better to deploy the components belonging to the same dependence set in the nearby or same node, so it could predict that two of the possible mappings would achieve the best result. One is to place all components of set *a* to node *A* and meanwhile place all components of set *b* to node *B*. The other is to place all components of set *a* to node *B,* and meanwhile place all components of set *b* to node *A*.

The evaluation result is shown in Figure 6. Coinciding with the prediction above, two mappings achieve the best result, which are 0 for FCV and 1 for WMP (for simplicity, we normalize all the results of WMP, based on the result of our strategy, as below). In contrast, the results of the worst case are 1.37 and 3.01, and the average of all cases is 0.88 and 2.09. Thus, the *component consolidation* makes great improvements compared to the average (100% for FCV and 52.2% for WMP) and the worst (100% for FCV and 66.8% for WMP) cases. Therefore, scheduling at the dependence set level obtains a better fairness result than scheduling at the component level. We should make consolidating components the first consideration when scheduling for fairness and performance.

Figure 6. Simulation results of components consolidation. The red marked pointes are the results of our strategy.



Priority-Ability Match

In this experiment, we evaluate the effectiveness of *priority-ability match*. We emulate three nodes with forwarding ability of 10000, 5000, and 1000 (packets/s), respectively, and three dependence sets with priorities of 3, 2 and 1, respectively. Again, we perform the permutation between dependence sets and nodes. With 6 possible mappings in all, each mapping is denoted by a 3-tuple (x, y, z), which means the dependence sets are scheduled in order to the nodes x, y and z, respectively.

According to our analysis, scheduling dependence sets in the exclusive scenario should comply with the order of high priority sets to high ability nodes. Therefore, it could predict that our strategy (*priority-ability match*) would schedule the dependence sets to the nodes as the mapping of (0, 1, 2), which is expected to achieve satisfied fairness result.

The evaluation result is shown in Figure 7. The mapping of (0, 1, 2) obtains the metrics 3.56 and 1. In contrast, the worst case gets 4.68 and 2.69, and the average results of all cases are 4.25 and 1.84. Among all mappings, ours achieves the best result, which improves the metrics of the average by 16.2% and 45.7%, and of the worst by 23.9% and 62.8%. Therefore, we consider that performing *priority-ability match* in the exclusive scenario can achieve significant improvement for fairness and performance.

Sets Affinity Match

In this test, we evaluate the effectiveness of the *set affinity match*. We emulate 6 dependence sets as shown in Table 1. According to the definition of set affinity, the sets 0 and 3 have the same affinity, as well as the sets 1 and 4, and sets 2 and 5.

Figure 7. Simulation results of priority match. The red marked pointes are the results of our strategy.

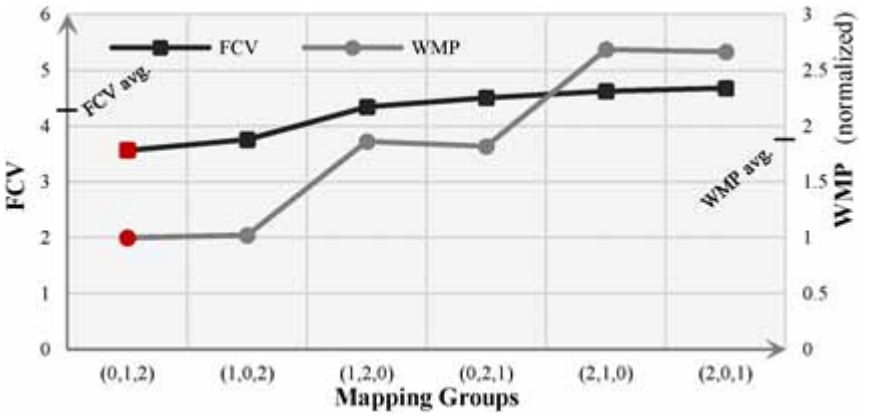


Table 1. Simulation data for set affinity match

DS ID	0	1	2	3	4	5
Throughput (packets/s)	1000	100	10	8000	800	80
Priority	1	1	1	8	8	8

Based on the permutation, we get 90 possible mappings in all. According to the strategy, since dependence sets pairs 0–3, 1–4 and 2–5 have the strongest affinity, they should be aggregated together for better fairness and performance.

The evaluation result is shown in Figure 8. Among all possible combinations, our dependence sets combination (0–3, 1–4, 2–5) achieves the results of 5.00 for FCV and 1 for WMP, respectively. In contrast, the worst case gets 5.74 and 1.43, and the averages are 5.20 and 1.20. Our strategy shows better fairness than most cases, and meanwhile shows the best performance among all cases, which improves both metrics by 4.0% and 16.7% for the average, and improves the worst by 14.8% and 30.1%. Therefore, scheduling the dependence sets with similar priority-throughput ratio into the same location can improve the fairness and performance for the sharing scenario, and benefits performance more than fairness.

Strategies Comparision

In this section, we compare our strategy proposed with some other strategies by a large-scale simulation with 360 nodes and 2880 components. The strategies that we consider include Random, Priority-First, Performance-First, and the variety of our proposed strategy. In order to emulate the nodes with various forwarding ability and dependence sets that have different priority and throughput, we divide all nodes into 15 clusters, and all components into 15 batches for simplifying configurations. All the dependence sets are classified into 3 types, respectively, containing 16, 8, and 4 components (abbreviated as 16-comp.DS, and so on). The detail configurations are shown in Table 2. We intend to verify the effect of our strategy by comparing it with the other practical strategies in a large-scale simulated environment.

Proposed

In our simulation, every node hosts 8 VMs evenly. Therefore, the DS that contains equal to or more than 8 components cannot share the node with another DS. In this way, our strategy handles the 16-comp.DS and 8-comp.DS in the exclusive scenario, and handles the 4-comp.DS in the sharing scenario. Since our proposed mechanism integrates all three steps mentioned before, we expect it to achieve the best result among all strategies in our evaluation. In fact, the evaluation result shows that the proposed one achieves 11.13 for FCV and 1 for WMP, which is the best result, and proves our prediction.

Figure 8. Simulation results of set affinity match. The red marked pointes are the results of our strategy.

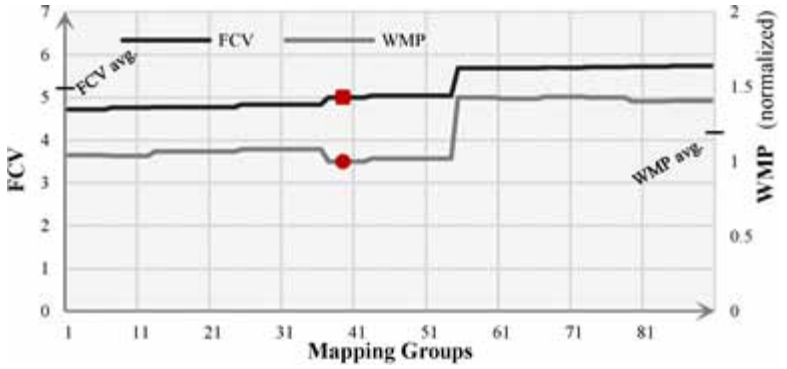


Table 2. Configurations for strategies comparison

<table>
<tr><td rowspan="2">Amount</td><td>Zone</td><td>Batch</td><td>16-comp.</td><td colspan="2">8-comp.</td><td>4-comp.</td></tr>
<tr><td>15</td><td>15</td><td>1/batch</td><td colspan="2">10/batch</td><td>24/batch</td></tr>
<tr><td colspan="3" rowspan="2">Forwarding Ability (packets/s)</td><td colspan="2">Switch n</td><td colspan="2">Nodes in Zone n</td></tr>
<tr><td colspan="2">$100000*0.98^n$</td><td colspan="2">$10000*0.95^n$</td></tr>
<tr><td colspan="3" rowspan="3">Components in Batch n</td><td>Priority</td><td colspan="3">Throughput (packets/s)</td></tr>
<tr><td rowspan="2">$10*0.95^n$</td><td>n%3=0</td><td>n%3=1</td><td>n%3=2</td></tr>
<tr><td>100</td><td>80</td><td>50</td></tr>
</table>

Random

Random scheduling is the simplest strategy requiring minimum decision cost, in which, the components are scheduled to random locations without any other considerations. Thus, it just schedules all sets independently to arbitrary network locations, where the resources are sufficient. Since there is not a strategy to help improve the fairness and performance, we predict that the result of random scheduling might be the worst. We execute 50 groups of random scheduling, in which the best metric results are 22.33 and 1.05, the worst results are 24.56 and 1.09, and the averages are 23.61 and 1.07. The results fluctuate in a very narrow range, which is reasonable for its random nature. Compared to our strategy, both the fairness and performance metrics are greatly decreased.

Priority-First

Priority-First scheduling is a practical strategy that only concerns the fairness priority. It sorts all dependence sets according to their priorities, and matches them to the network locations according to their forwarding abilities. It looks like Proposed, but it does not aggregate the dependency sets with similar priority-throughput ratio. In fact, it only adopts priority-ability match step in Proposed. As a result, the dependence set with higher fairness priority is always scheduled to the location with higher forwarding ability, which ignores the potential influence caused by high-throughput components. Since this strategy integrates a part of mechanism we propose, it is reasonable that its result comes between the Random and Proposed. The actual results are 15.14 and 1.02, which are a great improvement compared to Random, but are still worse than Proposed.

Performance-First

Performance-First scheduling is another practical strategy, which only concerns the network performance. Compared to the Proposed and Priority-First strategy, it does not set priorities to components, but searches the location that can obtain the best performance just at the moment of scheduling. We emulate it based on the principle of *optimal adaptation search* presented by Jung, Joshi, Hiltunen, Schlichting, and Pu (2009). In each iteration, we randomly pop a dependence set and schedule it to the location with the lowest latency. Similar to the Priority-First strategy, Performance-First is a local optimal strategy. Thus, it should behave between the Random and Proposed, like Priority-First. The evaluation results of the two metrics in Performance-First are 23.51 and 1.06. Surprisingly, this strategy only obtains a slightly better result compared to

Random, but a far worse one than Priority-First and Proposed. Therefore, Performance-First is a far from ideal strategy for guaranteeing fairness and performance.

A-RandomB and RandomA-B

For a deep exploration, we divide our strategy into two stages. In *Stage A*, since resources are sufficient, dependence sets are scheduled in the exclusive scenario. In *Stage B*, the remaining dependence sets are aggregated in the sharing scenario. We evaluate each stage by replacing the other stage with the random scheduling policy.

First, we maintain the original strategy of Stage A and replace Stage B with the random scheduling (A-randomB). Then, we recover to the original Stage B and replace Stage A with the random scheduling (randomA-B). Since both of them are varieties of Proposed, we guess that the result of each lies between Random and Proposed. In fact, A-randomB obtains the results of 17.08 and 1.03, and randomA-B gets 20.45 and 1.04. The metrics in both of them are reasonably located between Random and Proposed, as we expected. These results demonstrate that both Stage A and Stage B can improve fairness and performance.

Summary

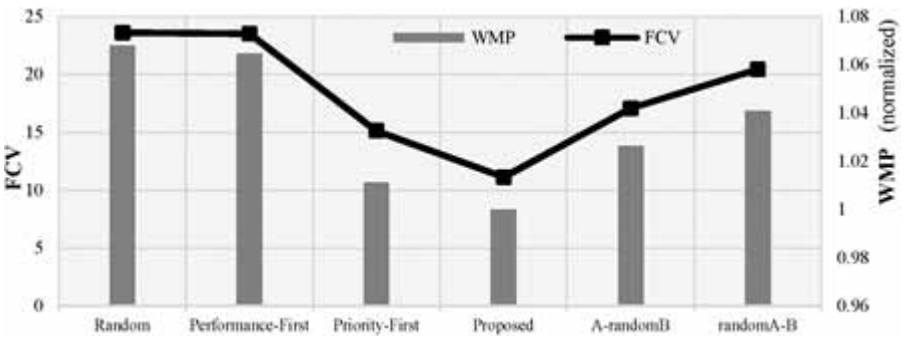
As shown in Figure 9, among all strategies, ours (Proposed) obtains the best result. It improves the metrics by 52.8% and 6.6% compared to Random, 26.5% and 1.1% with regard to Priority-First, and 52.7% and 6.1% compared to Performance-First. Moreover, by exploring Proposed in depth, both Stage A and Stage B in it make an obvious contribution on improving the fairness and performance, while Stage A performs better than Stage B.

RELATED WORK

Many works are related to ours in two aspects: improving performance and guaranteeing fairness.

Some works strive to improve the network performance by optimizing the placement of VMs. Tsakalozos, Roussopoulos, Floros, and Delis (2010) adopt the users' knowledge to guide VM scheduling, by relying on the clues given by users, managers could meet individual performance requirements of applications. In contrast, Meng, Pappas, and Zhang (2010) take action to learn

Figure 9. Comparison of different guarantee strategies



lots of characteristics. These works improve network performance depending on the knowledge of applications, but they do not quantify the dependence degree of components precisely, which may cause potential congestion and high latency. To adapt the dynamic nature of clouds, Jung et al. (2009) use a run-time scheduling engine to redeploy applications automatically. It tests all possible scheduling polices via offline experiments. However, it is too expensive for scheduling applications to all candidate places, and is thus impractical in real clouds. In contrast, Singh et al. (2011) establish a queue model to estimate performance, eliminating the cost produced by offline experiments, which is similar to ours.

Some other works focus on application fairness. Chen, Feng, Li, and Li (2014) introduce the performance-centric fairness for the data parallel applications. They make effort on maximizing the application performance while maintaining the performance-centric fairness. Their work focuses on the bandwidth and the tradeoff between resources and performance, while our work focuses on the latency and the transaction level fairness for the latency. Guo et al. (2013) analyze the negative influence produced by unfair bandwidth assignments from the perspective of tenants. It adopts the game theory to assign fair bandwidth to VMs. Carrera, Steinder, Whalley, Torres, and Ayguadé (2008) try to guarantee performance fairness between web applications, in which, the CPU time is considered as the dominant resource. Isard et al. (2009) guarantee the fairness for parallel computing in clusters, and aim to guarantee fair serving time to parallel jobs. Depending on a queue-based scheduler, the jobs could share processing time fairly. Some other works focus on fairness problems in the cloud (Doulamis, Doulamis, Varvarigos, & Varvarigou, 2007; Huh, Yoo, Kim, & Hong, 2012; Wei et al. 2010). However, these works rarely consider the fairness for response time, and ignore the requirements of transactions, which may potentially affect the real fairness. In fact, the above problems are exactly what we focus in this paper.

CONCLUSION

Traditional methods of guaranteeing network fairness focus mainly on the bandwidth, which cannot help the response time that can be experienced directly by end-users. In addition, many works provide fair performance only to components, ignoring the requirements of transactions, which may cause weak fairness for multitier applications. In this paper, we focus on the fairness of the network toward multitier applications. We analyze the potential problems, considering the specific requirements of response time and transactions, construct a fairness model, and build two metrics to evaluate the fairness status of the cloud, and propose a mechanism for improving the status. We evaluate our work by simulation, and the results show that our method can greatly improve the fairness of the cloud, and has obvious advantages compared to some other mechanisms.

ACKNOWLEDGMENT

This research is supported by National Science Foundation of China under grant 61232008, National 863 Hi-Tech Research and Development Program under grant 2013AA01A213, Chinese Universities Scientific Fund under grant 2013TS094, Research Fund for the Doctoral Program of MOE under grant 20110142130005, and EU FP7 MONICA Project under grant 295222.

REFERENCES

Anonymous. (2014, January 28). Network delay. Retrieved from http://en.wikipedia.org/wiki/Network_delay

Anonymous. (2013, July 7). Coefficient of variation. Retrieved from http://en.wikipedia.org/wiki/Coefficient_of_variation

Bennett, J. C., & Zhang, H. (1996). Why WFQ is not good enough for integrated services networks. *Proceedings of ACM conference on Network and Operating Systems Support for Digital Audio and Video* (pp. 524-532). ACM.

Bourguiba, M., Haddadou, K., El Korbi, I., & Pujolle, G. (2014). Improving network I/O virtualization for cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, *25*(3), 673–681. doi:10.1109/TPDS.2013.29

Carrera, D., Steinder, M., Whalley, I., Torres, J., & Ayguadé, E. (2008). Utility-based placement of dynamic web applications with fairness goals. *Proceedings of IEEE/IFIP Network Operations and Management Symposium* (pp. 9–16). IEEE. doi:10.1109/NOMS.2008.4575111

Chen, L., Feng, Y., Li, B., & Li, B. (2014). Towards Performance-Centric Fairness in Datacenter Networks. *Proceedings of IEEE International Conference on Computer Communications* (pp. 1599-1607). IEEE.

Doulamis, N. D., Doulamis, A. D., Varvarigos, E. A., & Varvarigou, T. A. (2007). Fair scheduling algorithms in grids. *IEEE Transactions on Parallel and Distributed Systems*, *18*(11), 1630–1648. doi:10.1109/TPDS.2007.1053

Guo, J., Liu, F., Zeng, D., Lui, J., & Jin, H. (2013, April). A cooperative game based allocation for sharing data center networks. I *Proceedings of IEEE International Conference on Computer Communications* (pp. 2139-2147). IEEE. doi:10.1109/INFCOM.2013.6567016

Huh, S., Yoo, J., Kim, M., & Hong, S. (2012, June). Providing Fair Share Scheduling on Multicore Cloud Servers via Virtual Runtime-based Task Migration Algorithm. *Proceedings of IEEE International Conference on Distributed Computing Systems* (pp. 606-614). IEEE. doi:10.1109/ICDCS.2012.33

Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., & Goldberg, A. (2009, October). Quincy: fair scheduling for distributed computing clusters. *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 261-276). ACM. doi:10.1145/1629575.1629601

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., & Pu, C. (2009). A cost-sensitive adaptation engine for server consolidation of multitier applications. *Proceedings of ACM/IFIP/USENIX Middleware conference* (pp. 163-183). Springer-Verlag. doi:10.1007/978-3-642-10445-9_9

Li, J., Wang, Q., Jayasinghe, D., Malkowski, S., Xiong, P., Pu, C., & Kawaba, M. (2012). Profit-based experimental analysis of IaaS cloud performance: impact of software resource allocation. *Proceedings of IEEE Ninth International Conference on Services Computing* (pp. 344-351). IEEE. doi:10.1109/SCC.2012.85

Meng, X., Pappas, V., & Zhang, L. (2010). Improving the scalability of data center networks with traffic-aware virtual machine placement. *Proceedings of IEEE International Conference on Computer Communications* (pp. 1154–1162). IEEE. doi:10.1109/INFCOM.2010.5461930

Peterson, L. L., & Davie, B. S. (2007). *Computer networks: a systems approach*. Elsevier.

Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., & Stoica, I. (2012). FairCloud: sharing the network in cloud computing. *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (pp. 187-198). ACM. doi:10.1145/2342356.2342396

Radonjic, M., & Radusinovic, I. (2010). Average latency and loss probability analysis of crosspoint queued crossbar switches. *Proceedings of 52nd International Symposium ELMAR* (pp. 203-206). IEEE.

Rao, J., Wei, Y., Gong, J., & Xu, C. Z. (2013). Qos guarantees and service differentiation for dynamic cloud applications. *IEEE eTransactions on Network and Service Management*, *10*(1), 43–55. doi:10.1109/TNSM.2012.091012.120238

Shieh, A., Kandula, S., Greenberg, A. G., Kim, C., & Saha, B. (2011). Sharing the Data Center Network. *Proceedings of USENIX conference on Networked Systems Design and Implementation* (pp. 23-23). USENIX.

Shrivastava, V., Zerfos, P., Lee, K. W., Jamjoom, H., Liu, Y. H., & Banerjee, S. (2011). Application-aware virtual machine migration in data centers. *Proceedings of IEEE International Conference on Computer Communications* (pp. 66-70). IEEE. doi:10.1109/INFCOM.2011.5935247

Singh, R., Shenoy, P., Natu, M., Sadaphal, V., & Vin, H. (2011). Predico: a system for what-if analysis in complex data center applications. *Proceedings of the 12th International Middleware Conference* (pp. 120-139). International Federation for Information Processing. doi:10.1007/978-3-642-25821-3_7

Tsakalozos, K., Roussopoulos, M., Floros, V., & Delis, A. (2010). Nefeli: Hint-based execution of workloads in clouds. *Proceedings of IEEE International Conference on Distributed Computing Systems* (pp. 74-85). IEEE. doi:10.1109/ICDCS.2010.66

Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, *54*(2), 252–269. doi:10.1007/s11227-009-0318-1