A Distributed QoS-Constraint Task Scheduling Scheme in Cloud Computing Environment: Model and Algorithm

Bing Li, A Meina Song, Junde Song

*1,2,3 PCN&CAD Center, Beijing University of Posts and Telecommunications, Beijing, 10086, China
Email: vincent.lisiyi@yeah.net1, mnsong@bupt.edu.cn2, jdsong@bupt.edu.cn3

Abstract

Improving resource utility could increase the benefits of Cloud Computing service providers and decrease the price paid by users in the end. Therefore, how to scheduling tasks for resource is a key problem. Most of scheduling schemes proceeding nowadays have no QoS differentiation, which is necessary for Cloud Computing service operation. More than that, they are central architecture. The central node would be overloaded when the number of users is huge. And the central scheduler relies on messages to update the global status information. If the status changes a lot, the messages will influence the network performance. Inspired by Grid Computing and P2P network, this paper presents a QoS-differentiated system model for sharing resource between different QoS-constrained users in Cloud Computing system at first. And second, this paper brought an optimizing chord algorithm to scheduling tasks submitted by users with lower QoS-constrained. The experiment result shows that this algorithm improves the system utility much more than the traditional way in this model.

Keywords: Cloud Computing, Resource Sharing, Scheduling, QoS, Chord

1. Introduction

Cloud Computing is considered as a significant approach to solve the problem of computing and storage in a large-scale cluster environment. However it is not a new concept in the field of computer science. It has an intricate connection to Grid Computing paradigm, Utility Computing and Cluster Computing [1]. It is an evolution of resource multiplexing and a result of rethinking the resource organization. Cloud Computing maintain application and data by Internet and remote server. It separates the detail of system resource management and application design. The reason why users choose Cloud Computing Service is it could provide computing and storage service access anytime and anywhere with a low price as the electricity provision [2]. Cloud Computing Service Provider (CCSP) could reduce the marginal cost of operation by size effect. Users could order the system resource from service provider as they want and change them dynamically. The most important thing is users will not have to buy lots of expensive hardware to satisfy the peak requirement of access even if their utility rate is low in most of the days. Therefore the concept of Cloud Computing is welcomed since the day of emergence. A lot of huge companies e.g. Google [3], Yahoo! [4], Amazon[5] and Facebook[6] have participated in the race spontaneously. Who will win this competition depends on the price in addition to service itself. It’s obviously that the provider offering the lowest price with the same quality of service will win most users. Cloud Computing Service Provider could lower their price by multiplexing the resource as far as possible to improve average utility. And a delicate task-scheduling scheme can make the CCSP satisfy more demand with less resource relatively, so it has become the core issue. In this paper, we introduce a QoS-constraint task-scheduling scheme.

Inspired by Grid Computing and P2P Network, this paper proposes a distributed QoS-constraint task-scheduling (DQCTS) scheme based on Chord. It has several advantages: decentralization, scalability, and robustness. Users with idle resource could share with others. And chord is used to storage the information of nodes with idle resources. Users could rent the idle resource according to chord information.
2. Related Work

Tradition ways of task scheduling is not fit to Cloud Computing [1][7][8]. At present, there are lots of task scheduling schemes implemented in different cloud framework. Hadoop[9] adopts the FIFO (First In First Out)[10] scheme by default. The advantage of FIFO is simple and low overhead. All the jobs from different users are submitted to an only queue. And they will be scanned according to the priority and the order of submission time. The first job with highest priority will be selected to processing. The disadvantage of FIFO is poor fairness. Those jobs with lower priority have little chance to process with lots of higher priority jobs. In order to improve the fairness, Facebook presented Fair Scheduling Algorithm [6]. The objective of fair scheduling is that all tasks can achieve their resource as the time goes by. This algorithm lets short tasks finish in reasonable time while not starving long tasks. Task occupies the whole resource with no other tasks in the system. And the system will allocate the idle time slot to those new tasks and make each of them could get equal CPU time. Fair Scheduling defines deficit of tasks. Tasks with larger deficit mean they got more unfair treatments, so they have more probability to obtain resource. Beyond this, fair scheduling algorithm guaranteed the minimized shared resource. It means task with lowest priority could have its turn even if there are many tasks with higher priority. Yahoo! presents Capacity Scheduling for Hadoop as well [11]. It allows for multiple-tenants to securely share a large cluster such that their applications are allocated resources in a timely manner under constraints of allocated capacities. This scheme allows sharing a large cluster while giving each organization a minimum capacity guarantee. Clusters will be partitioned among multiple organizations and each organization can access any excess capacity no being used by others. All the algorithms introduced above focus on tasks of computing oriented, and not fit for tasks of service oriented. In addition, Zhongyuan Lee etc. presented a dynamic priority-scheduling algorithm on service request scheduling [12]. This paper builds a non-preemptive priority M/G/1 queuing model for the tasks and the system cost function for this model. After that, the author gave the corresponding strategy and algorithm to get the approximate optimistic value of service. QuXilong and Hao Zhongxiao etc researched the distributed software resource sharing in Cloud Manufacturing system and implemented the sharing scheme in a cloud platform [15]. However, the scheduling schemes introduced above are centralized algorithms and will become bottleneck in large scale Cloud Computing environment. Moreover, they are designed for a specific computing paradigm, which is performance oriented and not suit for other Cloud Computing Services, which is service oriented. The former one executed with short period and high utility and the later one executed with long term and lower utility.

The rest of this paper is organized as follows. Section 3 gives the system model and presents the problem formulation. In section 4, we proposed the distributed resource scheduling scheme and algorithm. The numerical result and analysis are provided in section 5 and section 6 concludes the paper.

3. System Model and Problem Formulation

3.1. System Model

The architecture of Cloud Computing system is central style, which determined by its commercial model. Fully distributed architecture is hard to management, monitoring and pricing. From the systemic viewpoint, we can treat the Cloud Computing environment as a huge server with unlimited resource. It means the system will allocate a bundle of resource to a specific task according to user's requirement. All operation details are hidden behind the cloud and users were no necessary to understand how the system works.

Normally designers use different QoS classifications in system [16]. In this paper, we divided tasks into two categories: performance oriented tasks which apply resource in a short period with high utility; and service oriented tasks which rent resource in a long period with lower utility, which is
represented by $C_1$ and $C_2$ respectively. MapReduce is the representative of $C_1$ and web service is the representative of $C_2$. In order to provide differentiated service, CCSP signed the Service Level Agreement (SLA) contract with users. Different SLA denotes different quality of service (QoS) and different prices. In this paper, we consider the waiting time as one aspect of QoS. Users apply appropriate resource from CCSP to handle the requests under certain quality of service. High QoS constraint means short waiting time request. According to their payment, users can be given relative priorities. For simplicity of expression, we assumed that there are two different priorities named $P_1$ and $P_2$ for the users of performance oriented. $P_1$ is higher than $P_2$. The higher priority has larger chance to obtain resource allocation and we will discuss it in section 3. It is easy to increase the number of priorities with the same idea. $U_1$ denotes the users of $C_1$ with $P_2$, and $U_3$ denotes the users of $C_2$. $U_1 = \{u_{11}, u_{12}, \ldots, u_{1x}\}$, $U_2 = \{u_{21}, u_{22}, \ldots, u_{2y}\}$, and $U_3 = \{u_{31}, u_{32}, \ldots, u_{3z}\}$. For $U_1$ and $U_3$, Cloud Computing system allocates resource to them as soon as tasks submitted because they have higher priority. For $U_2$, they will search the idle resources and rent them just like the way of Grid Computing.

There are more parameter definitions in Table 1 and the system model is showed in Fig 1.

### Table 1. Model parameter definitions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{max}$</td>
<td>It is the maximal waiting time of tasks. According to the SLA, we assume that users of $U_2$ have the max waiting time $w_{max}$. If it exceeds $w_{max}$, Cloud Computing system will allocate resource requested immediately in order to satisfy the SLA contract.</td>
</tr>
<tr>
<td>$\tau$</td>
<td>It is the system processing time for each task.</td>
</tr>
<tr>
<td>$r$</td>
<td>It is the fundamental unit of resource and the total amount of $r$ is much larger than user’s demand.</td>
</tr>
<tr>
<td>$R_j$</td>
<td>The total amount of resource in each node of $U_j$.</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>It represents the portion of fundamental resource requested by $u_{ij}$, and $\sum_{i=1}^{n} \alpha_i \leq \sum_{j=1}^{m} R_j$.</td>
</tr>
<tr>
<td>$\beta_j$</td>
<td>It represents the portion of fundamental resource provided by $u_{ij}$, and $\sum_{j=1}^{m} \beta_j \leq \sum_{i=1}^{n} R_j$.</td>
</tr>
<tr>
<td>$\mu$</td>
<td>It represents the utility of resource.</td>
</tr>
<tr>
<td>$p$</td>
<td>The price of renting a unit bundle of resource per minute. It is determined by SLA contract signed between CCSP and users.</td>
</tr>
<tr>
<td>$c$</td>
<td>The cost of system operation per minute for unit resource and it is a constant. The cost is generated by electricity, hardware investment, human resource cost and so on.</td>
</tr>
<tr>
<td>$B$</td>
<td>It is the benefit of CCSP.</td>
</tr>
</tbody>
</table>

**Figure 1.** Cloud Computing system model
3.2. Problem Formulation

As the description above, we can see the benefit of users from $U_1$ and $U_2$ is fixed. So we only consider the utility $U_3$ of benefit of $U_2$.

$$Utility = \frac{R_j - \beta_j + \alpha_j}{R_j}$$  \hspace{1cm} (1)

$$B_{U_2} = \xi \cdot \sum_{i=1}^{y} [\alpha_j \cdot \tau \cdot p - c_3 \cdot \beta r \cdot (w + \tau)]$$

$$\xi = \begin{cases} 1, & w \leq w_{\max} \\ 0, & w > w_{\max} \end{cases}$$  \hspace{1cm} (2)

The purpose of task scheduling is maximizing of $Utility$ and $B_{U_2}$. In formula (2), we have to minimize the waiting time $w$ to improve $B_{U_2}$. In section 4, we introduce a distributed QoS-constrained task-scheduling scheme to address this problem.

4. Scheme and Algorithm

4.1. Scheduling Scheme

Because Cloud Computing system is a commercial model and satisfies user's request with lowest cost to achieve maximal benefit, it is necessary to utilize idle resource fully. As we introduced in section 3, tasks of $C_2$ (e.g. web service) rent resources for a long time with low utility. Users from $U_3$ always rent more resource than they needed in order to handle emergency situations. However, parts of time slots of the resource are idle and wasted. Inspired by Grid Computing, the system utility will be improved greatly if we can use those idle resources.

Meanwhile, Cloud Computing is a QoS-constrained system. CCCP has to provide corresponding service according to contract signed with users. In this paper, we wait the waiting time as QoS. It's easy to enlarge the scale of QoS parameters with the same idea and method. Therefore, CCCP has to provide resource required by users on time within the stipulated time, which is $w < w_{\max}$.

System will check the user's group information and priority at first when users submit tasks. If users belong to $U_1$ or $U_3$, system will allocate proper resource to them immediately. If users belong to $U_2$, system will check whether there is sufficient resource provided by users from $U_3$. If they did, the minimal resource, which satisfied the requirement will be allocated to applicant. If they didn't or waiting time reaches the upper bound, which is $w = w_{\max}$, system will allocate free resource to applicant immediately.

4.2. Algorithm Description

Normally, there will be a central scheduler to schedule tasks. However, the scheduler could be overloaded in a large-scale cluster system especially under the condition of huge amount of tasks. On the other hand, the central scheduler needs lots of heartbeat messages to update the global status. These messages could cause network congestion. Consequently, central scheduler could be bottleneck of the system. In this section, we present a distributed QoS-constrained task-scheduling algorithm (DQCTS) to share responsibility for central scheduler. DQCTS simplifies the design of Cloud Computing system and applications based on it by addressing these difficult problems:

- Decentralization: DQCTS takes the advantage of distributed hash function and spreads keys evenly around the system; there is no master node in the system and all the users have the
equal position. Users find the routing according to the distributed hash table and negotiate with destination node directly.

- **Scalability**: Because the cost of Chord lookup grows as the log of the node numbers, DQCTS need no global parameter and system is feasible for large-scale situation. Node can join and quit the system whenever they want.

- **Robustness**: DQCTS will automatically change the content of chord table and reflect those newly joined nodes and failures. This feature goes especially well with Cloud Computing because node failure is common phenomenon.

According to chord [16], consistent hashing uses base hash function such as SHA-1 to assign an m-bit identifier to each node and key. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible [15]. The length of SHA function is normally larger than 160. So we considered the each identifier should be unique. Key is a binary array, which contains the amount of resource $\beta_j$ and IP address of the machine, key=$\{\beta_j, \text{IP address}\}$. Key[0]=$\beta_j$, key[1]=IP address. Cloud Computing System assigned an nodeID to those users from $U_3$ who have the ability to share a part of resource, nodeID=H(D$_n$). D$_n$ means the description of the IP address and port of node. H represents the consistent hashing function. Moreover, id$=H(\text{key}[0])$. The consistent hashing assigns keys to nodes as follows. Identifiers are arranged in an identifier circle modulo $2^m$ from small to large. Key k is assigned to the first node whose identifier is equal to or follows the identifier of k, which is $\text{id}_k$ in the identifier space. We call this node the successor node just as in the traditional chord. There is a finger table with m entries saved on each node. The i$^{th}$ entry saves the n's successor of $(n + 2^{i-1}) \mod 2^m$. We call it $n.\text{finger}[i].\text{node}$. Each finger table entry concludes the IP address and Port number of specific node. Beyond this, successor[i] represents the successive node and predecessor[i] represents the former node of node n.

![Figure 2. An identifier circle consisted of 81 nodes](image-url)
When users submitted tasks to Cloud Computing, system would check the priority of users at first and allocate proper resource as they were requested if users belong to $U_1$ and $U_3$. If users were from $U_2$, they will calculate the id according to their request, $id'=H(Dr')$ and $Dr'$ means the description of resource amount $\alpha_i$. After the tasks have been submitted to System, it will access to a node $ns$ in the chord ring randomly. At first, user will compare $\alpha_i$ with the keys[0] which is saved on ns. If $\alpha_i \leq keys[0]$, then user would access the node according to keys[1] and keys[2] and negotiate with it for sharing the resource. Second, if $\alpha_i > keys[0]$, user will check the finger table of the ns to find the first node $n_j$ which satisfied $\alpha_i \leq \beta_j$, $1 \leq j \leq m$. Then user would choose the previous node of $n_j$ and check i'ts finger table as the same way. This procedure can be iteratively until $\alpha_i = \beta_j$ or node $n_j'$ is the smallest one bigger than $\alpha_i$. In the condition of no node in the finger table satisfied $\alpha_i \leq \beta_j$, $1 \leq j \leq m$, then user invokes $n.finger[m].node$ to locate the farthest node it can reach and repeat the process above until find the appropriate node. The reason of recursion is the closer node is to key, the more it will know about the identifier circle in the region of key. Third, if all the nodes' resource cannot meet the requirement of user, system will allocate free resource at last. The algorithm complexity is $O(\log N)$.

![Figure 3. Flow chart](image)

5. Performance Analysis

Because there is no any scheme like DQCTS to share idle in Cloud Computing system, an extensive simulation shows the system utility and benefit of the DQCTS In this section. We set the parameters in the simulations as follow: $R=20r$, $p=10$, $c_1=1$, $r=1$, $m=5$, $w_{\text{max}} = 2,000\text{ms}$. We assume that searching
each table of a node cost 120ns and \( w \) equals to the sum of searching time. In addition, \( w \), \( \tau \), \( \alpha_i \) and \( \beta_j \) were generated randomly with normal distribution. \( w \) was generated between 1ms to 2,500ms with the expectation of 1000 and the standard deviation of 500. \( \tau \) was between 1,000ms to 300,000ms with the expectation of 200,000 and the standard deviation of 50,000. Here, we vary \( \alpha_i \) from 1 to 100 with the expectation of 50 and the standard deviation of 25. And we vary \( \beta_j \) from 1 to 1,000 with the expectation of 500 and the standard deviation of 200.

We assumed that each node of \( U_3 \) has the same amount resource with 5000r. Therefore the occupied resource of node \( j \) is 5000 - \( \beta_j \). Fig. 4 shows the utility of \( U_3 \). We can see in the figure that DQCTS improved system utility obviously. The deviation of the two lines, which means the improvement of utility became larger as the increasing of \( z \).

![Figure 4. The Utility of \( U_3 \)](image)

Fig. 5(a) shows the system benefit of DQCTS when all the resource request from \( U_2 \) were satisfied by idle resource from \( U_4 \) where \( y=50 \) and \( z=100 \). And Fig. 5(b) shows the system benefit when the resource request from \( U_2 \) were satisfied by idle resource from \( U_4 \) partially where \( y=50 \) and \( z=25 \). We did not consider the resource collector when tasks finished. From this figure, we can find out that the system benefit in DQCTS is higher than any other existing scheme and the reason is obvious that they did not consider sharing the idle resource and the \( B_{ij} = 0 \) for them. Tasks submitted to other scheme were assigned to free resource directly. If there were no free resource, tasks would wait in the queue.
6. Conclusion

In this paper, we focused on task scheduling in Cloud Computing environment with certain QoS constraint, which is waiting time. First, we modeled the scheduling process and then presented a distributed searching scheme based on chord to address the problem of central bottleneck. Finally, extensive simulation results illustrated significant improvement of system utility and the benefit of CCSP. In this paper, we have not considered the influence of network and the interaction with storage resource. We will focus on it in the future work.

7. Acknowledgments

This work is supported by the National Key project of Scientific and Technical Supporting Programs of China (Grant Nos.2008BAH24B04, 2009BAH39B03); the National Natural Science Foundation of China (Grant No.61072060); the Program for New Century Excellent Talents in University (No.NECET-08-0738); the Co-construction Program with Beijing Municipal Commission of Education; Engineering Research Center of Information Networks, Ministry of Education.

8. References