Bottleneck-aware Allocation of Multiple Resources with Time Constraint

Caihong Bao[1][2]
College of Further Education, Qujing Normal University, Qujing, 655011, China
E-mail: `752551340@qq.com`

The ever-growing demand of cloud resources places the resource management at the heart of the design and decision-making processes in the cloud computing environment. A new allocation model: Balancing Fairness and Efficiency with Bottleneck-Aware Allocation(BAA) has found sound appropriate balance between fairness to the clients and utilization maximization of the system. We study the problems of BAA and improve BAA to design a new mechanism which is Bottleneck-Aware Allocation with time constraint. BAA with time constraint allows that users can join the system at any time and satisfies the sharing incentive(SI), envy freness(EF), pareto efficiency(PE) and Strategy-proofness (SP). An approximation algorithm is hereby presented that can efficiently compute a near-optimal profit schedule which time complexity is o(n). It can achieve good resource utilization based on the greedy. We believe that our work informs the design of superior multiusers system while expanding the scope of fair division theory by initiating the dynamic study and the fair resource allocation mechanism.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker
[2]The work was supported by Chinese Natural Science Foundation Grant No. 11361048.

1. Introduction

The cloud computing paradigm offers users rapid access to the computing resources such as CPU, memory and storage with minimal management overhead. The recent commercial cloud platforms, exemplified by Amazon EC2[1], Microsoft Azure and Linode[2], organize a shared resource pool for serving their users. The emergence of cloud system as a common computation resource gives rise to plenty of optimization problems. The elastic and on-demand nature of cloud computing is to assist cloud users in meeting their dynamic and fluctuating demands with the minimal management overhead.

The resource allocation is a key building block of any shared computer system. Current production resource management and scheduling systems often use some mechanisms to guarantee fair sharing of computational resources among different users of the system. One of the most popular allocation policies proposed so far has been the max-min fairness [3], which can maximize the minimum allocation received by a user in the system. Assume that each user has an equal share of the resources. Dominant Resource Fairness(DRF) [3] suggests performing the max-min fair share algorithm over so called dominant user's share, which is the maximum share that a user has been allocated of any resource. Simultaneous and fair allocation of multiple continuously divisible resources called as the bottleneck-based fairness(BBF) is proposed [4]. A new model called Bottleneck Aware Allocation (BAA) based on the notion of local bottleneck sets to maximize the system utilization while providing fairness in the allocations of the competing clients [5]. The model provides clients that are bottlenecked on the same device with allocations that are proportional to their fair shares, while allowing allocation ratios between clients in different bottleneck sets to be set by the allocator to maximize the utilization [6]. In addition, all these approaches assume that all jobs and/or resources are continuously divisible.

2 Bottleneck-aware Allocation

The main goal of DRF is to make the maximum number of users to run the tas in a fair situation, which ignores the utilization of resources; and the system resource utilization might appear anomaly. DRF and BBA are not satisfied with SP. We propose a new design of BAA with time constraint for improving the utilization based on BBA. It meets the SP properties and make maximum utilization of the system resources. The thought of bottleneck allocation lies in the users to get the same resources who have the same bottleneck resources type. Strategy-proofness (SP) is an important property that the users should not be able to get benefit by lying about their resource demands; however, the current allocation mechanism rarely satisfies this property according to our research. We used to make the Bottleneck-Aware allocation to meet the SP. We assume there are two users $U=(u_1,u_2)$ and system resources $C=(c_1)=(100)$. The resources allocation are $A=(A_1^{t_1},A_2^{t_1})=(40,60)$ at time t_1. u_2 shares resources more than u_1 at time t_1. When u_1 needs more resources at time t_2, u_1 can get more resources than u_2 ($A=(A_1^{t_1},A_2^{t_2})=(60,40)$).

2.1 Basic Setting

(1) $U=(u_1,u_2,\cdots,u_k)$ the set of users. $R=(r_1,r_2,\cdots,r_k)$: resources types. We define a variety of resources which types are not limited. $C=(c_1,c_2,\cdots,c_n)$ resource constraints which it is resource systems

(2) For every user i, we normalize the resource demand vector to d_i, where d_{ir} is the fraction of resource r required by each task of user i over the entire system. For the sake of simplicity, we assume the positive demands for all users, $d_{ir}>0$, $\forall i\in U$, $r\in R$. We assume the system have two resource types, CPU and Memory; therefore, $m_i=\frac{d_{i1}}{d_{i1}+d_{i2}}$ and $h_i=\frac{d_{i2}}{d_{i1}+d_{i2}}$.

(3) Let x_i be the number of tasks processed on the server for the user i.

(4) Let A_i^t denote the allocation of client i at time t under some resource partitioning. The total throughput of the system is $A = \sum_i A_i = \sum_i \int_0^t A_i^t dt$. User $\forall i \in U$ shares resources of CPU and Memory are $A_i^t \cdot m_i$ and $A_i^t \cdot h_i$ at time t.

(5) The load of a client i on the Memory is h_i and on the CPU is m_i. Partition the clients into two sets S and D based on their hit ratios. D and S is the set of users who have the same bottleneck resource CPU for $\forall i \in D$ and Memory for $\forall j \in S$.

For now, we assume that the users have infinite number of tasks to be scheduled. Infinite users join the system at different times.

For example, we assume $U = (u_1, u_2, u_3, u_4)$, $R = (r_1, r_2) = (CPU, Memory)$. The resources of the system are $C = (c_1, c_2) = (100,\ 200)$, we have $m = (m_1, m_2, m_3, m_4)$ $(0.2, 0.4, 0.7, 0.9)$ and $h = (h_1, h_2, h_3, h_4) = (0.8, 0.6, 0.3, 0.1)$. In this case, $h_{bal} = \frac{100}{100+200} \approx 0.33$, thus $u_1, u_2, u_3 \in D$ and $u_1 \in S$.

2.2 Bottleneck-aware Policy

(1) Fairness between clients in :

$$\forall i, j \in D_k,\ A_i \cdot m_i \ge A_j \cdot m_j \ge p_d. \tag{2.1}$$

Because the users belong to the same set, they have the same dominant/bottleneck resource type. Our goal is that the users who have the same set shared dominant/bottleneck resources are no less than the other users from the equity considerations. The dominant/bottleneck resource type of users in the set D is CPU.

(2) Fairness between clients in S:

$$\forall i, j \in D_k,\ A_i \cdot h_i \ge A_j \cdot h_j \ge p_s. \tag{2.2}$$

The dominant/bottleneck resource type of users in the set S is Memory.

(2.3)Fairness between a client in D and a client in S: the dominant/bottleneck resource type of users in the set D is CPU, and users in S is Memory. Allocation mechanism to make the share resources CPU of users in set D are greater than the in set S, and the memory of users in set S are greater than in set D in (2.3)(2.4).

$$\forall i \in D, j \in S, A_i \cdot m_i \ge A_j \cdot m_j. \tag{2.3}$$

$$\forall i \in D, j \in S, A_j \cdot h_j \ge A_i \cdot h_i. \tag{2.4}$$

2.3 Bottleneck-aware Allocation with Time Constraint

We joined the constraints of time in BAA. If the current user shares the resources are low, the system will give priority to allocate resources to this user so as to ensure fairness without any starvation phenomenon.

There are many algorithms objective functions, such as the maximum utilization of the system resources, the maximum total number of tasks and the minimum number of tasks of each user. In this paper, we set the objective function for maximum resource utilization per unit time in (2.5).

$$\text{Maximize } \sum_i \frac{A_i}{t}. \tag{2.5}$$

Subject to

$$\forall i, j \in D,\ \int_0^t A_i^t dt \cdot m_i \ge \int_0^t A_j^t dt \cdot m_j \ge \int_0^t p_d dt. \tag{2.6}$$

$$\forall i, j \in S,\ \int_0^t A_i^t dt \cdot h_i \ge \int_0^t A_j^t dt \cdot h_j \ge \int_0^t p_s dt. \tag{2.7}$$

$$\forall i \in D,\ j \in S,\ \int_0^t A_i^t dt \cdot m_i \ge \int_0^t A_j^t dt \cdot m_j. \tag{2.8}$$

$$\forall i \in D,\ j \in S,\ \int_0^t A_i^t dt \cdot h_i \le \int_0^t A_j^t dt \cdot h_j. \tag{2.9}$$

$$\sum_{i \in U} A_i \cdot m_i \le C_{CPU}. \tag{2.10}$$

$$\sum_{i\in U} A_i \cdot h_i \le C_{Memory}. \tag{2.11}$$

Theorem1. $\int_0^t A_i^t dt \cdot m_i = \int_0^t p_d dt$ and $\int_0^t A_i^t dt \cdot h_i = \int_0^t p_s dt$ to satisfy (2.6) and (2.7).

Proof. We have $\int_0^t A_i^t dt \cdot m_i \ge \int_0^t A_j^t dt \cdot m_j \ge \int_0^t p_d dt$ for $i, j \in D$ .according to (2.6). We have

$$\int_0^t A_i^t dt \cdot m_i \ge \int_0^t A_j^t dt \cdot m_j \text{, and } \int_0^t A_j^t dt \cdot m_j \ge \int_0^t A_i^t dt \cdot m_i \text{ for } \forall j \in D,. \tag{2.12}$$

so we have $\int_0^t A_i^t dt \cdot m_i = \int_0^t A_j^t dt \cdot m_j = \int_0^t p_d dt$ for $i, j \in D$. We can also get:

$$\int_0^t A_i^t dt \cdot h_i = \int_0^t A_j^t dt \cdot h_j = \int_0^t p_s dt,\ i, j \in S. \tag{2.13}$$

We obtain the following constraint.

$$\int_0^t A_i^t dt \cdot m_i = \int_0^t p_d dt,\ \forall i \in D. \tag{2.14}$$

$$\int_0^t A_i^t dt \cdot h_i = \int_0^t p_s dt,\ \forall i \in S. \tag{2.15}$$

Theorem 2. $p_d + p_s \ge \max(A_i)$ to satisfy (2.8)(2.9).

Proof. We have

$$A_i \cdot m_i \ge A_j \cdot m_j \Leftrightarrow A_j \cdot m_i \ge A_j(1-h_j) \Leftrightarrow A_i \cdot m_i + A_j \cdot h_j \ge A_j$$
$$\Leftrightarrow p_d + p_s \ge A_j \text{ for } i \in D,\ j \in S. \tag{2.16}$$

according to (12)(13).

$$A_i \cdot h_i \le A_j \cdot h_j \Leftrightarrow A_i(1-m_i) \le A_j \cdot h_j \Leftrightarrow A_i \cdot m_i + A_j \cdot h_j \ge A_i$$
$$\Leftrightarrow p_d + p_s \ge A_i \text{ for } i \in D,\ j \in S. \tag{2.17}$$

according to (2.12) and (2.13); so we have $p_d + p_s \ge \max(A_i)$ to satisfy (2.8) and (2.9).

$$p_d + p_s \ge \max(A_i). \tag{2.18}$$

$$\int_0^t p_d dt + \int_0^t p_s dt \ge \max(\int_0^t A_i^t dt). \tag{2.19}$$

Theorem 3. The incremental value of p_d or p_s is

$$tmp = \max(\frac{p_d + p_s - A_i}{(1/m_i)-1}) \text{ or } \max(\frac{p_d + p_s - A_i}{(1/h_i)-1}). \tag{2.20}$$

Proof. If $p_d' = p_d + tmp$, $i \in D$.

$p_d' + p_s \ge \max(A_i) \Rightarrow p_d + tmp + p_s \ge \max(A_i) \Rightarrow p_d + tmp + p_s \ge \max(A_i + tmp/m_i)$

$\Rightarrow tmp \cdot (1 - 1/m_i) \ge A_i - p_d - p_s \Rightarrow tmp \le \max(\frac{p_d + p_s - A_i}{(1/m_i - 1)})$.

If $p_s' = p_s + tmp$, $i \in S$, we have $p_d + p_s' \ge \max(A_i) \Rightarrow p_d + tmp + p_s \ge \max(A_i) \Rightarrow$ $tmp \le \max(\frac{p_d + p_s - A_i}{(1/h_i - 1)})$.

$$tmp = \begin{cases} \max(\frac{p_d + p_s - A_i}{(1/m_i - 1)}), & i \in D \\ \max(\frac{p_d + p_s - A_i}{(1/h_i - 1)}), & j \in S \end{cases} \tag{2.21}$$

$$M = \sum_{i\in D} \frac{h_i}{m_i},\ H = \sum_{i\in S} \frac{m_i}{h_i}. \tag{2.22}$$

Theorem 4. The objective function $E = \sum_{i\in U} A_i = \sum_{i\in U}(p_d + p_s) + \sum_{i\in D} p_d \cdot M + \sum_{i\in S} p_s \cdot H$..

Proof. $E = \sum_{i\in U} A_i = \sum_{i\in D} A_i + \sum_{i\in S} A_i = \sum_{i\in D}(p_d + p_d \cdot \frac{h_i}{m_i}) + \sum_{i\in S}(p_s + p_s \cdot \frac{m_i}{h_i})$

$= \sum_{i\in U}(p_d + p_s) + \sum_{i\in D} p_d \cdot \frac{h_i}{m_i} + \sum_{i\in S} p_s \cdot \frac{m_i}{h_i} = \sum_{i\in U}(p_d + p_s) + \sum_{i\in D} p_d \cdot M + \sum_{i\in S} p_s \cdot H$.

We propose an online approximation algorithm to solve BBA with the time allocation problem. We use a greedy strategy to maximize the resource utilization per unit of time. We have

$E=\sum_{i\in U}(p_d+p_s)+\sum_{i\in D}p_d\cdot M+\sum_{i\in S}p_s\cdot H$ to maximum (2.5) according to Theorem 3. If $M\geq H$, we increase the value of p_d priority, whereas the increase in the value of p_s. Algorithm 1 is the online approximation algorithm, which is waiting for new users to join the system, and then in turn calls Algorithm 2 and 3. Algorithm 2 is to allocate resources to the new user and the other users who share the bottleneck resources. The algorithm did not achieve p_d or p_s. The complexity of the algorithm is o(n).

```
ALGORITHM1 BAA with Time Online Approximation Allocation
Input: $U=(u_1,u_2,\cdots,u_n)$, $C=(c_1,c_2)$, resources CPU and Memory of the system
1: wait new user $u_i$ arrives the system
2: Add $u_i$ to the set of D or S based on her bottleneck resources
3: Algorithm2
4: Algorithm3
5: goto 1
```

```
ALGORITHM2 Set $A_i$ value
1: if $u_i\in D$
2: $A_i=\min(\frac{p_d}{m_i},c)$ //c is the remaining resources in system.
3: if $u_i\in S$
4: $A_i=\min(\frac{p_s}{h_i},c)$
5: for $u_j$ in $U$
6: if $u_j\in D$ and $A_i\cdot m_i<p_d$
7: $A_i=A_i+\min(\frac{p_d-A_i\cdot m_i}{m_i},c)$
8: else if $u_j\in S$ and $A_i\cdot h_i<p_s$
9: $A_i=A_i+\min(\frac{p_s-A_i\cdot h_i}{h_i},c)$
10: end for
```

```
ALGORITHM3 Set $p_d$ or $p_s$ value
1: for $u_i$ in $D$
2: $M=M+\frac{h_i}{m_i}$
3: for $u_i$ in $S$
4: $H=H+\frac{m_i}{h_i}$
5: int k=0
6: while c>0 // c is remaining resources in the system.
7: if $M\geq H$ or $K>0$
8: compute tmp value
9: $p_d=p_d+tmp$
10: k=k+1
11: End if
12: if $M\geq H$ or $K>0$
```

```
13: compute tmp value
14: $p_s = p_s + tmp$
15: k=k+1
16: End if
17: End while
```

3. Fairness Properties

The important and desirable properties of a fairness are shown as below:

(1) Sharing incentive(SI)[3]. Each user should be better off sharing the cluster. Each user should not be allocated more tasks in a cluster partition consisting of $1/n$ of all resources. Each user shall get at least the throughput he/she would get from the statically partitioned resource equally among them. This throughput will be referred to as the fair share of the user. In this paper, we will use fair share defined by equal partition of the resources

$$\forall i,j \in D\,,\ A_i \cdot m_i = A_j \cdot m_j \ \text{and}\ \forall i,j \in S\,,\ A_i \cdot h_i = A_j \cdot h_j\,. \tag{3.1}$$

(2) Envy-freeness(EF) [3]. A client cannot increase its throughput by swapping its allocation with any other client, that is, users prefer their own allocation over the allocation of any other user.

(3) Pareto efficiency(PE) [3]. It should not be possible to increase the allocation of a user without decreasing the allocation of at least another user. This property is important as it maximize the system utilization to satisfy other properties.

(4) Strategy-proofness (SP) [3]. Users should not be able to get benefited by lying about their resource demands. It provides incentive compatibility, as a user that cannot improve his/her allocation by lying. In this paper, we will use the proofness defined by users should not be able to get benefited by lying in the long time.

Theorem 5. BAA with time satisfies the SI property

Proof. We have

$$\int_0^t A_i^t dt \cdot m_i = \int_0^t A_j^t dt \cdot m_j = \int_0^t p_d dt\,,\ \forall i,j \in D\,. \tag{3.2}$$

and

$$\int_0^t A_i^t dt \cdot h_i = \int_0^t A_j^t dt \cdot h_j = \int_0^t p_s dt\,,\ \forall i,j \in S\,. \tag{3.3}$$

according to (2.6), (2.7), (2.14) and (2.15).

Theorem 6. BAA with time satisfies the EF property

Proof. If user u_i envies the user u_j, they will be the same to set D or S. If $u_i, u_j \in D$, we have $\int_0^t A_i^t dt \cdot m_i < \int_0^t A_j^t dt \cdot m_j$. The system will priority allocate resources to the user u_i according to the algorithm. If and only if $\int_0^{t_2} A_i^t dt \cdot m_i = \int_0^{t_2} A_j^t dt \cdot m_j$, the system will allocate resources to u_j, thus u_i will not envy u_j in the end.

Theorem 7. BAA with time satisfying the PE property

Proof. We assume the system have resources c to increase the allocation of a user without decreasing the allocation of at least another user. So $u_i \in U$, $A_i = A_i + c'$, $c' \le c$. We have $\sum_{i\in U} A_i + c' > \sum_{i\in U} A_i$. It contradicts with (2.5).

Theorem 7. BAA with time satisfying the SP property

Proof. If the user u_i gets resources $A_i' > A_i$ by deception for $\forall i \in D$. So $\exists j \in U$, $A_j' < A_j$, we have $A_i' \cdot m_i > A_j' \cdot m_j$ or $A_i' \cdot h_i > A_j' \cdot j_j$. The system will priority allocate resources to the user u_j. If and only if $A_i' \cdot m_i \le A_j' \cdot m_j$ or $A_i' \cdot h_i \le A_j' \cdot j_j$, the system will allocate resources to u_i. If the system will not allocate resources to in (t_1, t_2), the user will not get benefited by deception for the long time.

4. Experimental Results

We perform extensive experiments in order to investigate properties of the proposed algorithms. BAA with time constraint guarantees the users who have the same set to have the same dominant resource type, so we use the efficiency of system resources to analyze the performance of the algorithm. To analyze the performance, we present real data in experiments. We assume that each user brings the user 1/n of resources to join the system. As our data we use to trace the real workloads on a Google compute cell, for seven hours in [6]. The workloads consists of tasks, each of which runs on a single machine, consumes memory and one or more cores; the demands fit our model with two resources. For various values of n, we sampled n random positive demand vectors from the traces and analyzed the value of the three objective functions.

For the sake of convenience, we assumed each user joined to the system one by one. We selected n=1000 users to add system. Each user submitted the computing jobs, divided into a number of tasks with each requiring a set of resources. The experimental platform environment was using C# in Visual studio 2013.

The dominant resources of users in D is CPU and the users of S is Memory. The CPU resource usage of D is higher than S and the memory resource usage of S is higher than D according to (2.8) and (2.9). Fig.1 shows the CPU and memory utilization. BAA with time often ensures the utilization of two resources close to 100%. Fig. 2 shows the ratio of CPU utilization and Fig. 3 shows the ratio of Memory utilization. The blue line is the set of D and the yellow line is the set of S. From the Fig. 2 and Fig. 3, we can see that the CPU resource usage of D is higher than S and the memory resource usage of S is higher than the D.

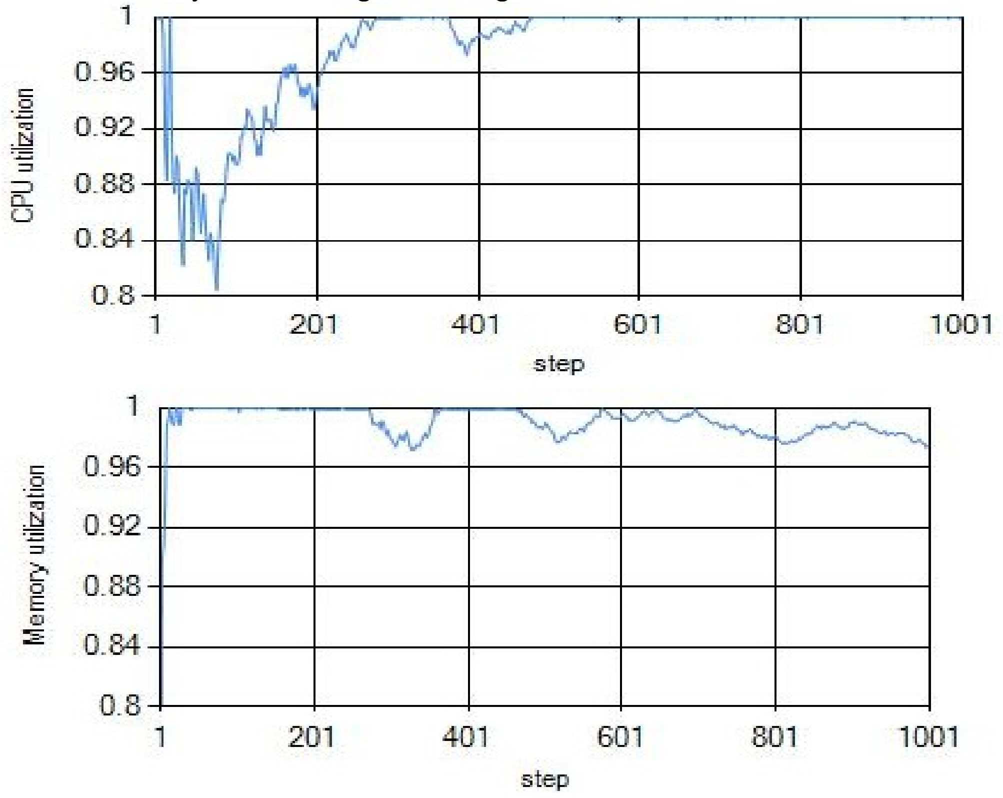

Figure 1: CPU and Memory Utilization

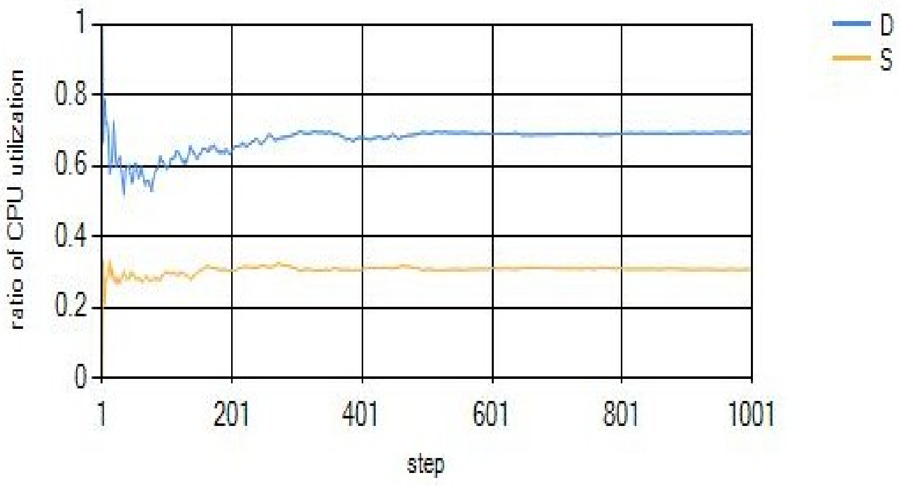

Figure 2: Ratio of CPU Utilization

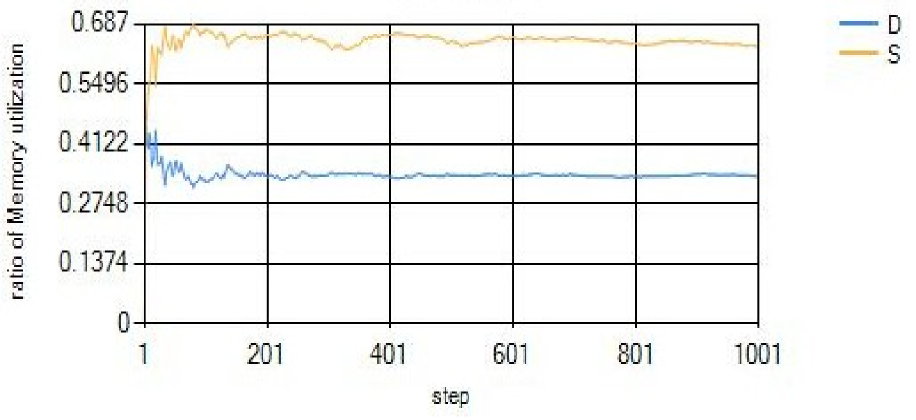

Figure 3: Ratio of Memory Utilization

5. Conclusion and Future Work

In this paper, we design a new mechanism, the Bottleneck-Aware Allocation with time constraint. BAA with time constraint allows that the users can join the system at any time and satisfy the constrained sharing SI, EF, PE and SP. An approximation algorithm was presented that can efficiently compute a near-optimal profit schedule. This algorithm computationally scales very well as the number of users grow and the resource utilization close to the offline algorithms There are several challenges that we need to address in order to complete the research. Firstly, we did not take into account the number of tasks. When allocating the same bottleneck resource to each user, the number of tasks to run is different. Secondly, the system allocates resources to each user which cannot guarantee the users can run the integer number of tasks. When the excess resources cannot run a task, the waste resources will exist. Thirdly, we assume that the task resources submitted by the users do not change, which is not satisfied with the actual situation. We plan to further analyze the performance and suitability of the production solution as well as possible problems that many appear in the future. As for future work, we use this allocation mechanism in the real system (e.g., Hadoop, yarn).

References

[1] M.ambrust, A. Fox. *Above the Clouds: A Berkeley View of Cloud Computing*[EB/OL].(2011-01-25). http://www.eecs.berkeley.edu/pubs/ techrpts /2009/EECS-2009-28. pdf.

[2] L.Kash, D.Ariel. *No Agent Left Behind: Dynamic Fair Division of Multiple Resources*[J]. Journal of Articial Intelligence Research. 51(2):579-603(2014)

[3] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski., *Dominant resource fairness: air allocation of multiple resource types*[J]. In Proceedings of the USENIX Conference on Networked Systems Design and Implementation. 23 (1):24-28(2011)

[4] M. Isard, M. Budiu, A. Birrell, D. Fetterly. *distributed data-parallel programs from sequential building blocks*[J]. Engineering Analysis. 32(1): 67–75(2007)

[5] H. Wang, P. J. Varman. *Balancing Fairness and Efficiency in Tiered Storage System with Bottleneck-Aware Allocation*[J]. In Proceedings of the USENIX Conference on File and Storage Technologies. ,12 (4):229-242(2014)

[6] M.Isard, V. Prabhakaran, J. Currey, U. Wieder. *Fair Scheduling for Distributed Computing Clusters*[J]. Storage Technologies. 16(2):261-276(2009)