PROCEEDINGS OF SCIENCE

Multi-Objective Bottleneck-Aware Allocation of Multiple Resources

Jun Liu[12]
College of Mathematics and Information Science, Qujing Normal University
Qujing, 655011, China
*E-mail:*liujunxei@126.com

The ever-growing demand for cloud resources places the resource management at the heart of design and decision-making processes in the cloud computing environment. In this paper, we consider multi-objective allocation to optimize the max min(x_i), maximize job numbers, and maximize resources utilization simultaneously. Firstly, a greedy online framework is presented to allow the scheduling decisions to be made based on any well-defined value function. In order to tackle the possibly conflicting objectives, we propose a fuzzy-based priority approach to explore the tradeoffs of two or more objectives at the same time; secondly, a novel algorithm is designed to find the nearest integer solution efficiently while maintaining the constraints and tightly bounding the optimal solution. In addition, this algorithm has very desirable runtime and solution quality properties when the number of tasks and machines become large.

CENet2015
12-13 September 2015
Shanghai, China

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

© *Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).*

PoS(CENet2015)043

1. Introduction

Cloud computing, such as Amazon EC2 [1], Microsoft Azure [2], and GoGrid [3], has become a hot research application and is a new computing model. The ability of cloud computing for users to immediately demand and obtain remote resources to help with computing and storage draws much interest from the computing and community. The cloud's key features include the pay-as-you-go model and elasticity. Users can instantly scale resources up or down according to the demand or the desired resources time. Current production resource management and scheduling systems often use some mechanism to guarantee fair sharing of computational resources among different users of the system.

There are several works that propose novel multi-resource aware scheduling methods. For example, one of the most popular allocation policies proposed so far is max-min fairness[4], a flexible resource allocation mechanism used in the most datacenter scheduler. The efficient and fair design algorithms for sharing multiple resources is becoming increasingly important. Dominant Resource Fairness (DRF)[5] suggests performing 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. A new allocation model Balancing Fairness and Efficiency with Bottleneck-Aware Allocation(BAA) [6] has found greatly appropriate balance between fairness to the clients and the maximized system utilization. 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. No agent left behind [7] proposes a dynamic resource allocation mechanism. Also, all these approaches make the assumption that all jobs and/or resources are continuously divisible. This paper presents an allocation with time limit on the bottleneck resources. This mechanism can meet some good properties, improve the robustness of the algorithm and have more practical value.

The reminding of the paper is organized as follows: in the following section, we will present Bottleneck-aware allocation with fuzzy-based priority approach and its inspiration. We give some good fairness properties and prove algorithm meet in Section3. Section 4 analyzes the performance algorithm on real data. The paper will be concluded in Section 5.

2 Bottleneck-aware Allocation

2.1 Basic Setting

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

(2) As to 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. As to the simplicity, we assume positive demands for all users, $d_{ir}>0$, $\forall i\in U$, $r\in R$.

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

(4) Let A_i denote the allocation of client i under some resource partitioning. The total throughput of the system is $\sum_i A_i$.

(5) We assume to have two resources: CPU and Memory. 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 $\forall j\in S$.

PoS(CENet2015)043

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

For example, we assume $U=(u_1,u_2)$ and $R=(r_1,r_2,r_3,r_4)$. The resources of the system are C=(CPU, Memory)=(1,2). The resources demand： $d_1=(0.8,0.2)$ and $d_2=(0.2,0.6)$. In this case, $h_{bal}=\frac{1}{3}\approx 0.33$, $m_1=\frac{0.8}{0.8+0.2}=0.8$ and $m_2=\frac{0.2}{0.2+0.6}=0.25$. So $u_1\in D$ and $u_2\in S$.

2.2 Bottleneck-aware Policy

(1) Fairness between clients in :

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

As the users belong to the same set, they have the same dominant/bottleneck resource type. Our goal is that the same set of users sharing 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:

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

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

(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 those in set S, and resources memory of users in set S are greater than those in set D.

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

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

(4) resources restriction

The total resources of users' share are no more than the total resources of the system.

$$\sum_{i\in U}A_i\cdot m_i\leq C_{CPU}. \tag{2.5}$$

$$\sum_{i\in U}A_i\cdot h_i\leq C_{Memory}. \tag{2.6}$$

2.3 Mono-objective Allocation

Mono-objective allocation considers a single optimization objective when deciding where to assign each user. In this subsection, we present three mono-objective allocations to maximize the job numbers, max min (x_i), and maximize resources utilization, respectively. The following describes the two objective.

Maximize job numbers: To maximize the total number of tasks of users can run. The validity range $E_s\in[E_s^{\min},E_s^{\max}]$ can be calculated using a greedy algorithm by simply mapping resources into the user who has the minimum resource requirements.

$$E_s=\sum_{i\in U}\min(A_i\cdot\frac{m_i}{d_{i1}},\ A_i\cdot\frac{h_i}{d_{i2}}). \tag{2.7}$$

Max min(x_i): to maximize the number of tasks of the user who has the minimum number of tasks. No user left behind: let the number of a user's task to run will not be reduced because of new users joining in the system; however, the resource requirements per task of each user are not same. This approach allows users who have bigger resources demand per task have the advantage. To estimate a feasible valid interval $E_x\in[E_x^{\min},\ E_x^{\max}]$, we use the simple algorithm by allocation resources to user who has the minimum dominant/bottleneck resources.

$$E_x=Max\min(x_i),\ i\in U. \tag{2.8}$$

Maximize resources utilization: in this sense, upon the allocation of resources, the CPU and memory can maximize the utilization of the system so as to improve the use efficiency of the system resources. This value represents the lower bound $E_A^{\max}$ of the (2.9) constraint

PoS(CENet2015)043

$E_A \in [E_A^{\min}, \ E_A^{\max}]$, where $E_A^{\max}$ is again calculated by allocating each resource on the users to maximize resource utilization.

$$E_A = \sum_{i \in U} A_i \ . \tag{2.9}$$

2.4 Multi-objective Allocation with A Fuzzy-based Priority Approach

In order to optimize two or more objectives at the same time, we propose a novel fuzzy-based priority approach to perform resources allocation.

(1) Dual-objective allocation: we first consider optimizing two objectives, for which, we use the following composite value function.

$$E^{x,y} = \left\langle \overline{E}^x(f), E^y \right\rangle. \tag{2.10}$$

The first objective X up to the specified fuzzy factor is selected to improve the second objective Y. The simple priority approach, on the other hand, would have allocation for the best X with much worse Y. The value of fuzzy factor as well as the priority should depend on the relative importance of the two objectives for optimization, which can be set by the user on the system administrator.

To implement the fuzzy-based priority, the value function for the first objective X is normalized between 0 and 1 in order to take the fuzzy factor into account, i.e.,

$$\overline{E}^X = \frac{E^X - E^X_{\min}}{E^X_{\max} - E^X_{\min}} \ . \tag{2.11}$$

Where $E^x_{\max}$ and $E^x_{\min}$ denote the maximum and minimum value in terms of objective X.

(2) Multi-objective allocation: With more than two objectives ,we can take a similar approach of optimizing one objective after another, but with combined value functions that consist of two or more objectives. For instance, the weighted sum method can be used to combine maximize resources utilization and max min (x_i)to form a single objective, i.e.,

$$E^{A,X} = \alpha \overline{E}^A + (1-\alpha)\overline{E}^X \ . \tag{212}$$

Where $\alpha \in [0, \ 1]$ denotes the relative weight assigned to resources utilization. Note that the functions for both objectives are format 0 and 1 to form a meaningful combination. Then, to optimize the maximum job numbers with the combined value(of maximize resources utilization and max min(x_i)), the following composite value function can be constructed

$$E^{S,AX} = \left\langle \overline{E}^S(f), E^{AX} \right\rangle. \tag{2.13}$$

In the case where the first objective is a combination of two or more objectives as in Equation(12), the combined value needs to be normalized while taking into account the fuzzy factor, i.e.,

$$\overline{E}^{AX} = \frac{E^{AX} - E^{AX}_{\min}}{E^{AX}_{\max} - E^{AX}_{\min}} \ . \tag{2.14}$$

The fuzzy-based priority rule described previously can then be applied in the same way as before. The exact priorities among different objectives and the fuzzy factor again depend on their relative importance.

2.5 Static Bottleneck-Aware Allocation

In this section, we present algorithm for static resources allocation. We assume all users join the system at same time in the static allocation. Optimization for allocation

$$\text{Maximize } E^{S,AX} \ . \tag{2.15}$$

According to Equation (2.1-2.6).

As the result is fractional solution from the linear programming, resources which users share are not the integer multiple of resources request per task and the extra part is meaningless; therefore, we propose Algorithm 1 to solve this problem so that the user is assigned to the resources needed just an integer multiple of the resources which will not be wasted.

PoS(CENet2015)043

In the original problem, tasks are not divisible; so one must determine an integer number of tasks. Algorithm is needed to compute an integer solution from this real-valued solution. The following algorithm finds x_i such that it is near x_i while maintain the resources constraint. Algorithm 1 finds x_i that minimizes $\|x_i - x_i^*\|_1$ for a given user i.

ALGORITHM1 Round to the nearest integer solution while maintaining the constraints

Input: $U = (u_1, u_2, \cdots, u_n)$ set of users, $x = (x_1, x_2, \cdots, x_n)$ set of tasks of each user, $C = (c_1, c_2)$ resources CPU and Memory of the system

Output: allocation x

1: for u_i in U

2: $x_i^* = \min(\frac{A_i m_i}{d_{i1}}, \frac{A_i h_i}{d_{i2}})$

3: $f_i = x_i^* - \lfloor x_i^* \rfloor$

4: $c_1 = c_1 - \lfloor x_i^* \rfloor d_{i1}$, $c_2 = c_2 - \lfloor x_i^* \rfloor d_{i2}$

5: end for

6: Sort the users in descending order of f_i

7: if $d_{i1} \le c_1$ and $d_{i2} < c_2$

8: $x_i = \lfloor x_i^* \rfloor$

9: Update resources c_1 and c_2

10: else

11: $x_i = \lfloor x_i^* \rfloor$

12: end if

Algorithm 1 operates on each row of x_i^* independently. Let f_i be the fractional part of the number of tasks that must be rounded up to satisfy the resources constraints. The algorithm simply rounds up those tasks that have the largest fractional parts. Everything else is rounded down. The result is an integer solution x_i until all tasks have been assigned properly.

For the complexity of Algorithm 1, the sorting and initialization takes 0 (nlogn) time. In the loop, algorithm need 0 (n) time to allocate resources; therefore, the overall complexity is 0 (nlogn).

2.6 Online Bottleneck-aware Allocation

The user may join the system at any time in practical applications of the system, so the allocation of resources is dynamic. We assume that a user is added to a system with bring $1/n$ of resources, and resources are allocated to users who join the system by the system. The system has k users and the total resources of the system are k / n at step k.

ALGORITHM2 Greedy Online Allocation pseudo-code

Input: $U = (u_1, u_2, \cdots, u_n)$, $C = (c_1, c_2)$ resources CPU and Memory of the system

Output: allocation x

1: Wait new user u_i joins to system

2: Sort the users of D and S in ascending order of dominant resource

3: for u_i in U

4: pick u_i is in D or S

5: if (satisfy the constraints)

6: allocate resources to u_i

7: end for

PoS(CENet2015)043

For the complexity of Algorithm 2, the sorting and initialization takes 0 (nlogn) time. The system can dynamically adjust the order of S or D in each allocation, so the time complexity is 0 (n). In the loop, algorithm need 0 (n) time to allocate resources. Therefore, the overall complexity is 0 (n).

3. Fairness Properties

The following are important and desirable properties of a fairness:

(1) Sharing incentive(SI). Each user should be better off sharing the cluster. In this paper we will use fair share defined by user's share resources more than 1/n total resources.

(2) Envy-freeness(EF). 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). It should not be possible to increase the allocation of a user without decreasing the allocation of at least another user.

Theorem 1. Satisfies the SI property

Proof. $\forall u_1 \in D$, we have $A_1 \cdot m_1 \geq A_2 \cdot m_2$ for $u_2 \in D$ according to (2.1), and $A_1 \cdot m_1 \geq A_2 \cdot m_2$ for $u_2 \in S$ according to (2.3). The system has n users, so u_1 share dominant resources more than 1/n total resources in the system.

Theorem 2. satisfies the EF property

Proof. $\forall u_1 \in D$, if $\forall u_2 \in D \text{ or } S$, we have $A_1 \cdot m_1 \geq A_2 \cdot m_2$ or $A_1 \cdot h_1 \geq A_2 \cdot h_2$ according to (2.1) (2.2).

Theorem 3. Satisfies the PE property

Proof. We assume system have resources c to increase the allocation of a user without decreasing the allocation of at least another user. So $u_i \in U$, $A_i = A_i + c'$, $c' \leq c$,. We have $\sum_{i \in U} A_i + c' > \sum_{i \in U} A_i$. It contradict with (2.9)(2.15).

4. Experimental Results

We've presented the potential and efficient algorithm with the complexity of 0(n) to realize the allocation mechanism. Our next goal is to analyze the performance on real data. We perform extensive experiments in order to investigate the properties of the proposed algorithms. In order 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 to our data, we use traces of real workloads on a Google compute cell for a 7 hour period [8]. The workload consists of tasks, where each task ran on a single machine, and consumed 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 assume each user joining to the system one by one. Each user submits the computing jobs, which are thus divided into a number of tasks with each requiring a set of resources. Experimental platform environment is using C# in Visual studio 2013.

PoS(CENet2015)043

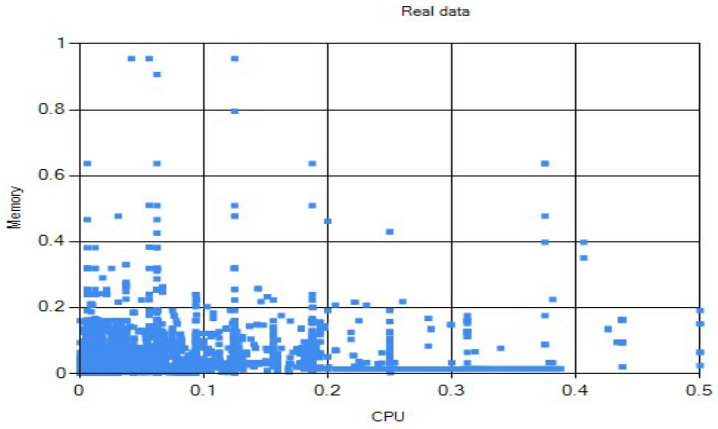

Figure 1: CPU and Memory Needs of Each Task in Real Data

Fig. 1 shows the CPU and Memory needs of each task for the actual architecture form [8]. The CPU or Memory need range of each task is (0,1).

Fig. 2 and 3 show that real users added to the system with n=1000. Fig. 4 and 5 show the real users added to the system with n=2000. Fig. 2 and Fig. 4 show CPU utilization, and figure 3 and figure 5 show memory utilization. The utilization of CPU and memory are all close to 100%.

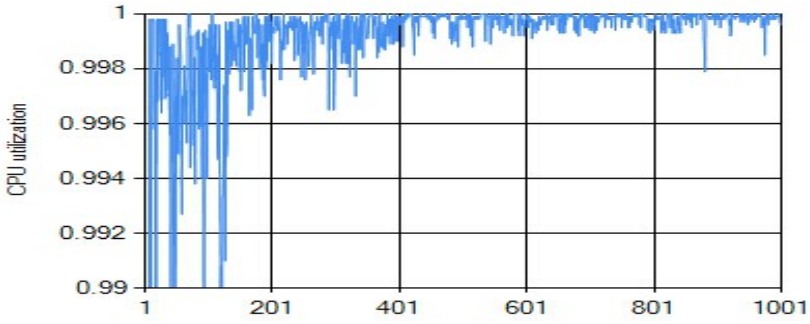

Figure 2: CPU Utilization with n=1000

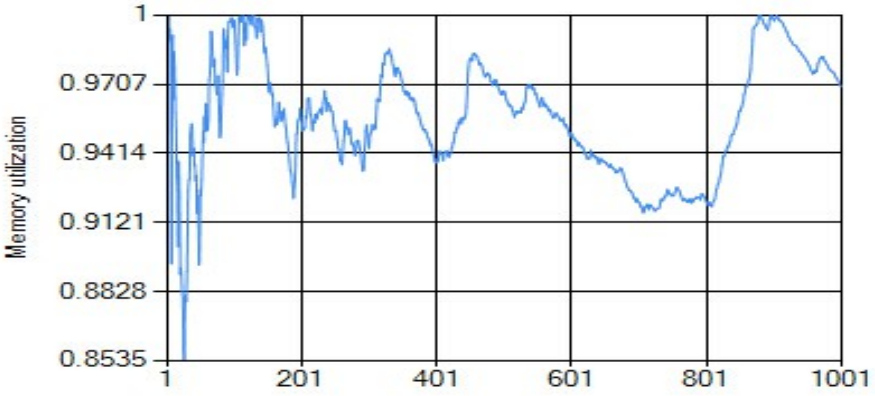

Figure 3: Memory Utilization with n=1000

PoS(CENet2015)043

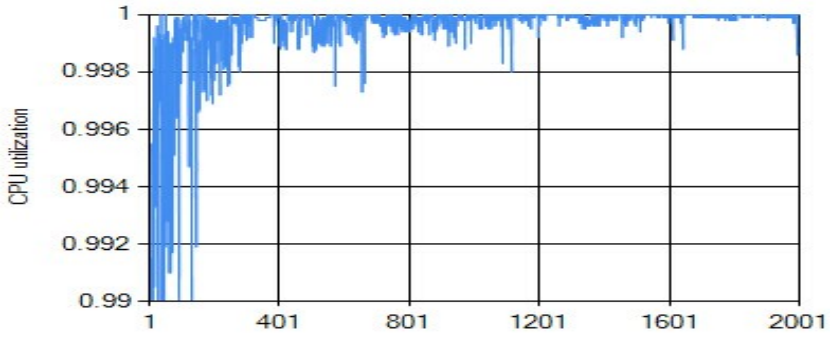

Figure 4: CPU Utilization with n=2000

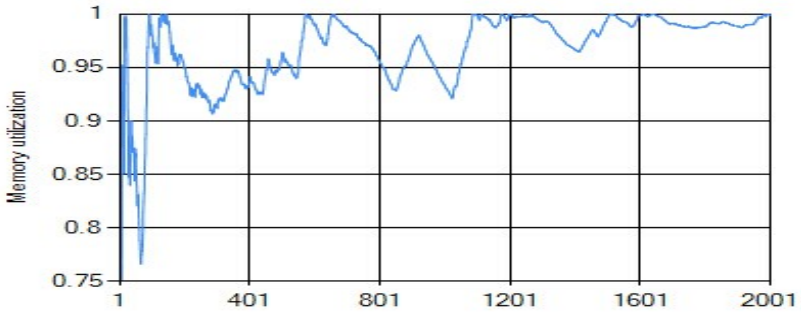

Figure 5: Memory Utilization with n=2000

5. Conclusion and Future Work

In this paper, we have considered online resources allocation and static Bottleneck-Aware allocation. We applied a novel fuzzy-based priority approach to a greedy online allocation framework for optimizing multiple objectives simultaneously, including the max min (x_i), maximized job numbers and maximized resources utilization. The result has also demonstrated the effectiveness of our approach for exploring and optimizing the tradeoffs between two or more objectives. A novel Algorithm 1 was presented that could efficiently compute a near-optimal profit schedule. This algorithm computationally scales very well as the number of tasks grows.

There are several challenges that we need to address in order to complete the research. Firstly, we assume that the user is added to the system without leaving the system in our model, but the reality is that the user is able to leave. Secondly, we assume that the task resources submitted by users do not change, which does not conform to the actual situation. We plan to further analyze the performance and suitability of the production solution as well as possible problems that may appear in the future. As to the 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] M.Isard, V.Prabhakaran, J.Currey, U.Wieder. *Fair Scheduling for Distributed Computing Clusters*[J]. Storage Technologies. 16(2):261-276(2009)

[3] J. dean,, S. Ghemawa. *MapReduce: Simplified Data Processing on Large Clusters*[J]. Compute Science. 26(2): 467-475,(2004)

PoS(CENet2015)043

[4] A. Ghodsi, M, Zaharia, S.Shenker, I.Stoica., , *Choosy:Max-Min Fair Sharing for Datacenter Jobs with Constraints*[J].,Computer Science. 32(4):124-135(2013)

[5] 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)

[6] H. Wang, H.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. 35(4):229-242(2014)

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

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

PoS(CENet2015)043