2014 IEEE 6th International Conference on Cloud Computing Technology and Science

Towards Economic Fairness for Big Data Processing in Pay-as-you-go Cloud Computing

Shanjiang Tang, Bu-Sung Lee, Bingsheng He
School of Computer Engineering
Nanyang Technological University, Singapore
{*stang5, bslee, bshe*}*@ntu.edu.sg*

***Abstract*—Recent trends indicate that the pay-as-you-go Infrastructure-as-a-Service (IaaS) cloud computing has become a popular platform for big data processing applications, due to its merits of accessibility, elasticity and flexibility. However, the resource demands of processing workloads are often varying over time for individual users, implying that it is hard for a user to keep the high resource utilization for cost efficiency all the time. Resource sharing is a classic and effective approach to improve the resource utilization via consolidating multiple users' workloads. However, we show that, current existing fair policies such as max-min fairness, widely adopted and implemented in many popular big data processing systems including YARN, Spark, Mesos, and Dryad, are *not* suitable for pay-as-you-go cloud computing. We show that it is because of their *memoryless* allocation feature which can arise a series of problems in the pay-as-you-go cloud environment, namely, cost-inefficient workload submission, untruthfulness and resource-as-you-pay unfairness. This paper presents these problems and outlines our plans to address them for pay-as-you-go cloud computing. We introduce our preliminary work done on the single-resource fairness and our ongoing work for multi-resource fairness, and outline our future work.**

***Keywords*-Long-Term Resource Fairness, Multi-Resource Fairness, Pay-as-you-go, Cloud Computing, YARN, LTYARN, Spark.**

I. Introduction

We are in the era of 'big data', which necessitates the need of efficient big data processing platforms [11]. Infrastructure-as-a-Service (IaaS) cloud computing has emerged as a promising platform for big data processing, due to the abundant elastic computing resources and flexible billing models [34]. It has been widely supported by many cloud providers such as Amazon EC2 and Rackspace. At any time, there are tens to thousands of users running their data-intensive workloads (e.g., MapReduce [8], [26], [27], [28], Spark [36], log processing [15], computational Biological processing [29], [30] and graph processing systems [7], [37]) on IaaS cloud concurrently.

Achieving the high resource utilization can improve the cost efficiency and in turn lead to the cost saving for users. However, in practice, the resource demand of a user is often varying over time, indicating that it is difficult for individual users to keep the high resource utilization all the time. Resource sharing has shown to be a classic and effective approach to maximize the resource utilization [12]. With resource sharing, a higher utilization can be achieved than non-sharing by allowing overloaded users to use the unneeded resources of underloaded users. To make users share their resources willingly, *fairness* is an important issue in resource allocation (e.g., sharing incentives).

Max-min fairness [22] is one of the most popular fair allocation policies. It achieves the fairness by maximizing the minimum allocated resources for users. It has been widely adopted by many big data processing systems including Hadoop [2], YARN [3], Spark [36] and Mesos [16]. Typically, the policy implemented in these systems is *memoryless* [24], i.e., it considers the fair allocation instantly without taking into account the historical allocation.

A. Motivation

Because of the *memoryless* characteristic in existing fair allocation policies, we argue that they are *not* suitable for pay-as-you-go cloud computing by identifying the following severe problems.

Resource-as-you-pay Unfairness Problem. In pay-as-you-go environment, we should ensure that the total amount of resources obtained by each user is proportional to her payment. However, the *memoryless* max-min fairness cannot guarantee this due to the varying resource demands of individual users over time. Consider a computing system consisting of 100 resources (e.g., CPUs) shared by two users U_1 and U_2 equally. At time t_0, the demand for user U_1 is 20, smaller than its share of 50. Then U_1's unused resources of 30 will be given to U_2 (i.e., U_2 lends to U_2) in terms of the work conserving property, supposing that U_2's demand is 80. Next at time t_1, assume that U_1's resource demand is 70, larger than its share. However, due to *memoryless* allocation, U_1 can only get her share of 50 resources (i.e., U_1 cannot get the lent resources of 30 back from U_2 at t_0) if U_2's demand (e.g., 100) is larger than U_2's share. If this scenario occurs frequently, it is *unfair* for U_1 to get the total amount of resources that she is entitled to from a long-term view.

Untruthfulness Problem. In the real world, a good policy should ensure that no cheating users can get benefits by falsely reporting their demands to the system. However, we show that the memoryless max-min fairness does not have such a guarantee. Consider three users U_1, U_2, U_3 share a computing system of 120 resources equally. Assume the *true* demands for U_1, U_2, and U_3 are $40, 30$, and 60, respectively. Suppose that U_2 and U_3 are honest users whereas U_1 is not. The allocation

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.120

results are shown in Figure 1. When U_1 is honest, the resulting allocation is illustrated in Figure 1(a). All 10 unused resources of U_2 will be given to the overloaded user U_3. In contrast, if U_1 is cheating the system by falsely report her demands (e.g., 100), then the final allocation will be Figure 1(b), where U_1 and U_3 equally get the same amount of 5 resources from U_2 in competition. Thus, U_1 get benefits by cheating the system, violating the truthfulness requirement.

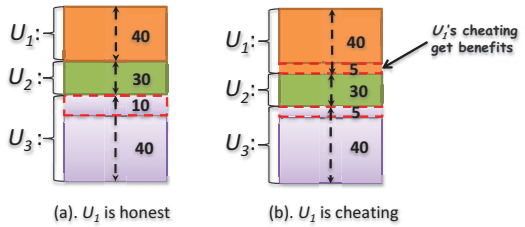

Fig. 1: A counter-example to illustrate that the memoryless max-min fair policy violates the truthfulness requirement.

Cost-inefficient Workload Submission Problem. Cloud resources are not free. It is important for users to submit cost-efficient workload[1] for cost efficiency. Therefore, we should have a mechanism to discourage users not to submit cost-inefficient workload. However, we observe that the memoryless policy does not have such a mechanism. Let's consider the example of Figure 1 in the paragraph of *Untruthfulness Problem*. If U_2 is a selfish user, it is possible that U_2 might possess its unneeded resources without doing anything or submitting some dirty jobs (e.g., running some duplication jobs). Because there is no benefit for U_2 to release the resources to others in *memoryless* allocation. Thus, it is prone to have the cost-inefficient workload submission problem for the *memoryless* fair policy.

B. Goals and Objectives

To address the above-mentioned problems, the goal of the project is to propose new fair policies suitable for pay-as-you-go cloud computing. To achieve that goal, we have the following detailed objectives:

- Study existing fair allocation policies, including its classification, algorithms, and applications.
- Have a deep understanding of current pay-as-you-go cloud computing, including its pricing plans, provided services, and resource specification. All of these factors can directly impact the design of cloud-oriented fair polices.
- Have a comprehensive analysis and classification of specific scenarios that need to be considered with regard to cloud computing and big data processing systems. It can enable us to know numerous specific topic issues and have a detailed project plan for them.
- Propose new fair policies for pay-as-you-go cloud computing under different scenarios.
- Implement the proposed new fair policies in current big data processing systems.
- Analyze and assess the proposed fair policies theoretically and experimentally.

The organization of the paper is as follows. Section II reviews the related work. Section III presents several desirable properties for pay-as-you-go cloud computing. We formulate our research problems in Section IV. We introduce our preliminary work and results in Section V, and describe our ongoing work in Section VI. Finally, Section VII concludes the paper and gives out our future work.

II. Related Work

Fairness is an essential issue in resource allocation for any shared computing system. In traditional High Performance Computing (HPC) and grid computing, max-min fairness [22] is a widely used allocation policy. The classical algorithms such as round-robin, weighted fair queueing [9] and proportional resource sharing [32], are all specific instances of max-min fairness [12]. They have been used for the allocation of single resources (e.g., CPU [6], memory [1], network I/O [9] and disk storage [4]). Moreover, the max-min fairness policy has been implemented in many current popular big data processing and management systems, including Hadoop [2], YARN [3], Spark [36], Mesos [16], Choosy [13], Quincy [19]. Hadoop [2] abstracts resources into map/reduce slots and allocates them fairly between pools and jobs. In contrast, YARN [3] organizes cluster resources into containers (i.e., a set of various resources such as memory, vcores), and ensures the resource fairness across queues. Spark [36] groups jobs into *pools* and guarantees resource fairness across pools. Mesos [16] allows multiple diverse computing frameworks such as Yarn, Spark to share a single psychical computing system. Choosy [13] considers the constraint max-min fairness, of which the placement constraints are taken into account. Quincy [19], a fair scheduler for Dryad [20], achieves the fair scheduling of multiple jobs by formulating it as a min-cost flow problem.

Besides the single-resource allocation, there are other studies focused on the multi-resource allocation (i.e., resources of multiple types). Dominant Resource Fairness (DRF) [12] is a most popular fair policy. It is a generalization of max-min fairness for multi-resource allocation by considering the fairness of the dominant resource between users. The attractiveness of DRF lies in its good properties including sharing incentive, pareto-efficiency, envy-freeness. It has been supported by current systems such as YARN [3], Mesos [16] for multi-resource allocation. Moreover, there has been lots of extension and generalization work for DRF, including [5], [21], [34], [23]. Liu et al. [17] developed multi-resource fairness for tenants in IaaS clouds.

Despite the various effort, all of these existing policies and their implementation on existing systems are indeed *memoryless*. We have shown in Section I-A that they are

[1]Cost-efficient workload: refers to the non-trivial workload that users truly want to submit and run on the cloud.

prone to suffering from three serious problems (i.e., cost-inefficient workload submission, untruthfulness and resource-as-you-pay unfairness). Thus, they are *not* suitable for pay-as-you-go cloud computing. Therefore, there is a need for our research project to propose new policies suitable for pay-as-you-go cloud computing and implement them in existing big data processing systems.

III. Desirable Properties for Cloud Computing

We identify the following *good* properties that are desirable for pay-as-you-go cloud computing. None of existing fairness policies can satisfy all those properties.

Sharing Incentive: Each user in the shared system should perform at least as good as that of exclusively using her own partition of resources. Otherwise, users would prefer to use her own partition of resources without sharing.

Truthfulness: Users should not be able to obtain benefits by cheating the system. This property is consistent with sharing incentive.

Resource-as-you-pay Fairness: In pay-as-you go environment, we should ensure that the total amount of resources received by each user is proportional to her payment.

Cost-efficient Workload Incentive: Cloud resources are not free. We should have a mechanism to discourage users to submit cost-inefficient workloads, especially in the case when they have some unused resources. Otherwise, selfish users would submit dirty workloads to possess idle resources, decreasing the cost efficiency and in turn wasting the money.

Pareto Efficiency: This property refers that it is impossible for a user to get more resources without reducing the resource of at least one user. Any allocation policy satisfying this property can guarantee that the resource utilization is maximized.

IV. Research Problem Formulation

We investigate how the good properties given in Section III can be used to guide the development of fair policies for pay-as-you-go cloud computing. Moreover, we need to have a summary of various scenarios in exploring fair policies in a more systematic manner. Two major dimensions should be considered, namely, *IaaS cloud computing* and *big data processing system*.

A. IaaS Cloud Computing

To meet different users' needs, current cloud providers generally offer several options of pricing plans and instance types. Consider the Amazon EC2 for example as follows:

Different Pricing Plans. There are three different pricing plans provided by Amazon EC2, namely, *on-demand price*, *spot price*, and *reservation price*. For on-demand price plan, users pay for the compute resources (e.g., instances) by the hour (i.e., less than an hour will be automatically rounded to one hour). For reservation price plan, users need to pay a one-time fee for a certain period of time and in turn get a significant discount on the hourly charge. So far, Amazon EC2 provides two types of reservation plans, namely, 1-year term and 3-year term, to satisfy different users' preferences. In contrast, the spot price plan allows users to bid on the unused resources and possess those resources for as long as their bid exceeds the current spot price.

Heterogeneous Instance Types. To satisfy the needs for different workloads, there are numerous types of instances (e.g., virtual machines) offered by Amazon EC2. Broadly, it classifies the instances into the following categories, namely, general purpose, compute-optimized, memory-optimized, and storage-optimized. For each category, it further refines them into multiple specific instances. For example, the general purpose instances includes t2.micro, t2.small, t2.medium, and m3.large. Different instances are with distinct resource configurations and different prices. Moreover, even for the same instance, its price is different under different pricing plans.

B. Big Data Processing System

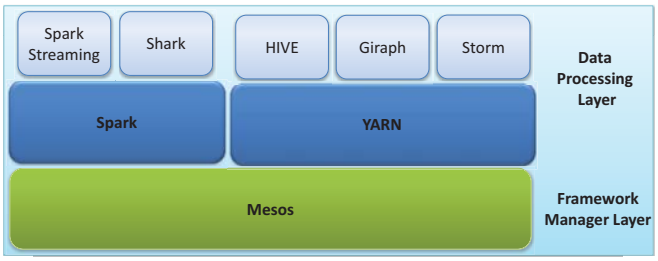

Fig. 2: The stack overview of big data processing systems for cloud computing.

To support big data processing, in recent years, more and more distributed data processing systems are being developed and open source by industrial companies. Figure 2 shows an emerging software stack for distributed data processing systems. Particularly, YARN (Hadoop MRv2) [3] and Spark [36] are two of the most popular ones. YARN, as a new generation of Hadoop, has become the de facto distributed resource management and data operating system [10]. Besides the MapReduce applications, it now can support a variety of other applications and systems, such as Giraph, Storm, HIVE. In contrast, Spark [36] is a lightning-fast in-memory data processing system. It can support a variety of data processing including streaming processing, batch processing and interactive processing. Both YARN and Spark support the fair resource allocation and task scheduling for multiple users. Moreover, to achieve the resource sharing for multiple big data processing systems/frameworks in a single physical cluster, a resource management system called Mesos [16] is proposed for fair resource allocation between multiple computing frameworks. As illustrated in Figure 2, with Mesos, we can make both Spark and YARN co-run in a single cluster.

C. Summary of Research Problems

The two dimensions (IaaS Cloud Computing and Big Data Processing System) have significant impact on the design of fair policy. For each of these aspects, we summarize the challenging issues and their co-relationship in Figure 3.

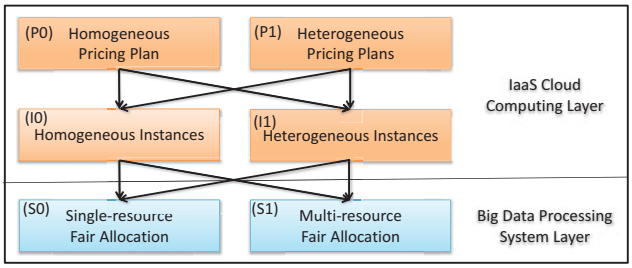

Fig. 3: The overview of undertaken research issues.

IaaS Cloud Computing Layer. Section IV-A tells us that there can be same/different pricing plan(s) for same/different instance(s) for users. We define *Homogeneous Instances* (I0) when they are of the same type. Otherwise they belong to *Heterogeneous Instances* (I1). If all instances provided by users are from the same pricing plan, we say *Homogeneous Pricing Plan* (P0). Otherwise, they are *Heterogeneous Pricing Plans* (P1). Therefore, as illustrated in Figure 3, there are four combinations between the pricing plans and instances.

Big Data Processing System Layer. We classify the fair resource allocations provided by data processing systems into two types, i.e., *single-resource fair allocation* (S0) and *multi-resource fair allocation* (S1). The single-resource fair allocation considers the fair allocation on the single resource (e.g., CPU) of the same type. In contrast, the multi-resource fair allocation refers to the fair resource allocation of multiple resource types (e.g., CPU, memory).

According to the classification at each layer, there are at least 2^3 combinations of scenarios, as arrows shown in Figure 3. In the following, we describe some detailed research problems based on their importance and simplicity (The abbreviation labels in the upper-left corner of each rectangle denote the corresponding categorizes in Figure 3).

Research Problem 1: P0+I0+S0. This problem focuses on the fairness of single-resource allocation. It can be used at the granularity of instances or specific computing elements (e.g., CPU, memory). The problem here considers a simple case where all instances are of the same types and with the same pricing plan contributed by users. Particularly, it is worthy mentioning that the same pricing plan does not mean the same and fixed price for all instances. We can further split the problem into two subproblems based on *P0*:

· *Research Problem 1-1: Fixed price for P0.* When the P0 is either on-demand pricing plan or reservation pricing plan, all instances of the same types will have the same price. In this case, the total amount of resources received by users is just proportional to their payment. We thus only need to consider the total amount of resources.

· *Research Problem 1-2: Dynamic price for P0.* If P0 is the spot pricing plan, the instances of the same types often have varying prices over time. In this case, we need to take such a dynamic pricing across instances and total amount of resources for each user into consideration. Compared to *Problem 1-1*, the additional challenging issue is that we need to have a deep understanding on resource biding and its impact on our resource allocation.

Research Problem 2: P0+I0+S1. It considers multi-resource fair allocation for users when we take into account the heterogeneous resource demands of each resource type for tasks in practice. It has the same assumption as *Problem 1* on cloud instance, pricing plan, and workloads. For multi-resource fair allocation, there are several more challenging issues than the single-resource fair allocations. The first issue is about how to define the fairness when we take into account the multiple resource types in allocation for cloud computing. Second, exploring a fair policy that can satisfy all the desired properties mentioned in Section III is a big research challenge. Lastly, similar to *Problem 1*, we also need to divide the problem into two subproblems by considering the fixed price and dynamic price separately for multi-resource fair allocation.

Research Problem 3: P1+I0+S0. It tackles the single-resource fair allocation under mixed pricing plans for the same typed instances. In practice, it is possible that users purchase instances of the same type (e.g., c3.xlarge) with different pricing plans (e.g., on-demand, reservation), and they form a shared group by contributing their resources. It means that there can be different prices for different users on the same instances. Being a fair resource allocation, we need to be aware of different users' pricing plans in counting their consumption. Otherwise, there will be *unfair* for those users contributing to the cheapest instances. For this problem, we need to have a mechanism that can monitor the resource allocation of each user and have a dynamic price estimation for overused resources. Still, we need to have a separate consideration for mix pricing plans between the fixed price and dynamic price as *Problem 1*.

Research Problem 4: P1+I0+S1. It focuses on a more complicate scenario on top of *Problem 2* by taking into account the same typed instances from different pricing plans of users. To achieve the multi-resource fair allocation, we first need to have an equivalent transformation between the different resource types of different pricing plans. Next like *Problem 2*, we need to propose a multi-resource fair policy based on this transformation.

In addition to the aforementioned four problems, there are other issues to consider. For example, the problem of considering the single/multiple-resource fair allocation under the heterogeneous (different) instances types (e.g., *P0+I1+S0*, *P0+I1+S1*).

Fair Policy Implementation. After proposing specific fair policies, the next step for us is to implement them in big data processing systems (e.g., YARN, Spark) for evaluation and practical use. We can first consider the single-level resource allocation for systems like Spark and Mesos. We also need to support the hierarchical resource allocation (i.e., multi-level resource allocation) for systems such as YARN.

Other Issues. There are other practical issues for achieving economic fairness on IaaS clouds. We describe several of them. First, resource sharing can lead to performance inter-

ference. Performance interference is a challenging problem on IaaS clouds. Some preliminary work [14] has attempted to address this problem. Second, it is also a practical and challenging problem [35] to extend the economic fairness to different price schemes and systems.

V. Preliminary Work and Results

We present our preliminary work to address *Problem 1-1* in Section IV-C on the single-resource fair allocation for pay-as-you-go cloud computing.

A. Policy Design and Analysis

Observing that the current fair policy has some serious flaws due to its *memoryless* feature in Section I-A, we have proposed a new policy called *Long-Term Resource Fairness (LTRF)* [24]. In contrast to the memoryless policy which considers the fairness of current resource allocation at instant time, LTRF takes into account the historical allocation by focusing on the total resource allocation over time between users for fairness. The core idea is learned from the *loan (lending) agreement* [18] with free interest in financial domain. That is, an overloaded user can release her unused resources to others as a *lending* manner. When that user needs more resources in future, she can get the amount of lent resources back from others (i.e., *returning* manner).

We use one example to illustrate the core idea of LTRF. Assume that there is a computing system of 200 resources (e.g., 200 GB RAM) shared by two users U_1, U_2 equally. Suppose that the new requested resource demands for U_1 are 40, 80, 160, 120, and for U_2 are 200, 120, 100, 100 at time t_1, t_2, t_3, t_4, respectively. The allocation results are presented in Table I. With LTRF, we can witness in Table I(a) that, at t_1, U_1's resource demand (e.g., 40) is smaller than her share of 100. U_1's unused resources of 60 are then lent to U_2, making U_2 have a resource allocation of 160. The allocation is similar at t_2. However, at t_3, t_4, U_1's demands are larger than her share. According to the *lending agreement*, LTRF allocates more resources to U_1 than to U_2 so that U_1's total allocated resources are equal to U_2 at t_4 finally, making fair between U_1 and U_2. In comparison, if we adopt the current memoryless fair policy, the resulting allocation will be shown in Table I(b). There will be *unfair* for U_1 and U_2 at t_4 for their total resource allocations (i.e., $U_1 : 320, U_2 : 480$) due to its *memoryless* property.

Finally, we have shown in our prior work [3] that, 1). LTRF can satisfy all the desirable properties mentioned in Section III, and thereby is *suitable* for pay-as-you-go cloud computing. The detailed analysis and proof on the properties can be found in [3]; 2). LTRF can guarantee Service-level Agreement (SLA) by minimizing the sharing loss and bring much sharing benefit for each user, whereas memoryless fair policy not; 3). The sharing methods using either LTRF or memoryless fair property can achieve better performance than non-sharing none, or at least as fast in the sharing case as they do in the non-sharing partitioning case.

	User U_1					User U_2				
	Demand		Allocation		Preempt	Demand		Allocation		Preempt
	New	Total	Current	Total		New	Total	Current	Total	
t_1	40	40	40	40	-60	200	200	160	160	$+60$
t_2	80	80	80	120	-20	120	160	120	280	$+20$
t_3	160	160	160	280	$+60$	100	140	40	320	-60
t_4	120	120	120	**400**	$+20$	100	200	80	**400**	-20

(a) Allocation results based on *LTRF*. *Total Demand* refers to the sum of the new demand and accumulated remaining demand in previous time.

	User U_1					User U_2				
	Demand		Allocation		Preempt	Demand		Allocation		Preempt
	New	Total	Current	Total		New	Total	Current	Total	
t_1	40	40	40	40	-60	200	200	160	160	$+60$
t_2	80	80	80	120	-20	120	160	120	280	$+20$
t_3	160	160	100	220	0	100	140	100	380	0
t_4	120	180	100	**320**	0	100	140	100	**480**	0

(b) Allocation results based on the *memoryless* max-min fairness.

TABLE I: A comparison example to show the resource allocation for a system of 200 resources equally shared by two users U_1, U_2 under the LTRF and memoryless fair policy.

B. Implementation and Evaluation

Given the proposed LTRF policy, we have implemented it in YARN by developing a prototype system named *LTYARN*, by generalizing the default memoryless max-min fairness to long-term max-min fairness. Currently, we support the memory resource fair allocation. Our implementation was done on top of Hadoop-2.2.0 and its source code has been opened at [31]. We performed experiments with a set of macro-benchmarks (e.g., synthetic Facebook workload, Purdue workload, TPC-H workload, and Spark worload). The experimental results show that LTYARN achieves a better resource fairness than existing Hadoop Fair Scheduler.

VI. Work in Progress

We are now working on *Problem 2* for multi-resource fair allocation. Learning from our prior work of LTRF on the single-resource allocation, we are extending existing multi-resource fair policy with LTRF. Particularly, in Section II, we have reviewed that the DRF is the most popular multi-resource fair policy implemented by many big data processing systems. It implies that we can try to have a new policy by combining DRF and LTRF.

DRF introduces the concept of a user's dominant share, defining as the maximum share of any typed resource that is allocated to the user. The resource corresponding to the dominant share is called *dominant resource*. DRF achieves the fairness by equalizing the dominant shares across all users. For example, if there are 40 CPUs and 40 GB memory in a shared system that is equally shared by two users U_1 and U_2. Assume that the resource demand per task for U_1 is $< 1CPU, 2GB >$ and for U_2 is $< 2CPUs, 1GB >$. Then the dominant resource for U_1 is memory with the dominant share of $2/40$, whereas for U_2 is CPU with the dominant share of $2/40$. With DRF, the resulting allocation is shown in Figure 4, where users U_1 and U_2 achieve the same dominant share of $26/40$.

Following the methodology of addressing Problem 1-1, we start by identifying the shortcomings for DRF and then introduce our new policy on how to address all these flaws.

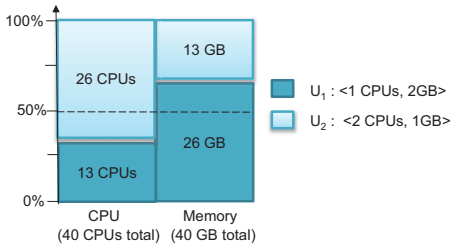

Fig. 4: DRF resource allocation for two users U_1 and U_2. The dominant share of U_1's dominant resource (i.e., memory) is equal to the dominant share of U_2's dominant resource (i.e., CPU).

We have implemented it in the big data processing systems such as YARN and evaluate it experimentally. Our initial results demonstrate the promising results of extending long-term fairness to multi-resource sharing. More details are given in our technical report [25].

VII. Conclusion and Future Work

Resource sharing is an effective and efficient approach to improve the resource utilization and cost efficiency for pay-as-you-go cloud computing. However, this research shows that the existing *memoryless* fair policy, widely used in many popular big data processing systems such as YARN, Spark, has some serious flaws in economic fairness. This paper first motivates our research study by having a detailed analysis for the problems on existing *memoryless* fair policies. Then it conducts a research problem formulation for each possible scenario. We particularly outline four problems that we believe are most important. The preliminary work on the single-resource fairness and ongoing work on the multi-resource fair allocation demonstrate the promising results of this research.

In future, we plan to address all the remaining research problems mentioned in Section IV-C.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. This work is partly supported by a MoE AcRF Tier 1 grant (MOE 2014-T1-001-145) in Singapore.

References

[1] A. K. Agrawala and R. M. Bryant. *Models of memory scheduling*. In SOSP'75, 1975.
[2] Apache. *Hadoop*. http://hadoop.apache.org.
[3] Apache. *YARN*. https://hadoop.apache.org/docs/current2/index.html
[4] J. Axboe. Linux Block IO Present and Future (Completely Fair Queueing). In OLS'04, pp. 5161, 2004.
[5] A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, I. Stoica. *Hierarchical Scheduling for Diverse Datacenter Workloads*. SOCC'14, 2014.
[6] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng. *Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems*. In ATC'05, 2005.
[7] R. Chen, M. Yang, X.T. Weng, B. Choi, B.S. He, X.M. Li. *Improving large graph processing on partitioned graphs in the cloud*, SoCC'12, pp.3-15, 2012.
[8] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, In Proceedings of the 6th Symposiumon Operating SystemsDesign and Implementation (OSDI), 2004.
[9] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In SIGCOMM89, pp. 112, 1989.
[10] *Develop with YARN as the Data Operating System for Enterprise Hadoop*. http://hortonworks.com/get-started/yarn/, 2014.
[11] C. Doulkeridis and K. NOrvag, *A survey of large-scale analytical query processing in MapReduce*. In VLDB Journal, vol. 23, pp. 355-380, 2014.
[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Schenker,I. Stoica. *Dominant Resource Fairness: Fair Allocation of Multiple Resource Types*. In NSDI'11, pp. 24-37, 2011.
[13] A. Ghodsi, M. Zaharia, S. Shenker and I. Stoica. *Choosy: Max-Min Fair Sharing for Datacenter Jobs with Constraints*, EuroSys 2013, April 2013.
[14] Y.F. Gong, B.S. He, D. Li, *Finding Constant From Change: Revisiting Network Performance Aware Optimizations on IaaS Clouds*. IEEE/ACM SC'14, 2014.
[15] B.S. He, M. Yang, Z.Y. Guo, R.S. Chen, B. Su, W. Lin, L. D. Zhou, *Comet: batched stream processing for data intensive distributed computing*, ACM SoCC'10, pp.63-74, 2010.
[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker and I. Stoica, *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*, NSDI 2011, March 2011.
[17] H.K. Liu, B.S. He. *Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds*, IEEE/ACM SC'14, 2014.
[18] Loan agreement. http://en.wikipedia.org/wiki/Loan_agreement.
[19] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg. *Quincy: Fair Scheduling for Distributed Computing Clusters*, In SOSP'09, pp 261-276, 2009.
[20] M. Isard, M. Budiu, Y. Yu, A. Birell, D. Fetterly. *Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks*. Eurosys'07, pp.59-72, 2007.
[21] I. Kash, A. D. Procaccia, N. Shah. *No agent left behind: dynamic fair division of multiple resources*. In AAMAS'13, PP. 351-358, 2013.
[22] Max-Min Fairness (Wikipedia). http://en.wikipedia.org/wiki/Max-min_fairness.
[23] D. C. Parkes, A. D. Procaccia, N. Shah. *Beyond Dominant Resource Fairness: Extensions, Limitations, and Indivisibilities*. In ACM Conference on Electronic Commerce, pp. 808-825, 2012.
[24] S.J. Tang, B.S. Lee, B.S. He and H.K. Liu, *Long-Term Resource Fairness: Towards Economic Fairness on Pay-as-you-use Computing Systems*, in ICS'14, pp.251-260, 2014.
[25] S.J. Tang, Z.J. Niu, B.S. Lee, and B.S. He, *Multi-Resource Fair Allocation in Pay-as-you-go Cloud Computing*, http://pdcc.ntu.edu.sg/xtra/tr/Technical_Report-07-2014, 2014.
[26] S.J. Tang, B.S. Lee, B.S. He. *Dynamic slot allocation technique for MapReduce clusters*, Cluster'13, pp.1-8, 2013.
[27] S.J. Tang, B.S. Lee, B.S. He. *DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters*, in IEEE TCC'14, 2014.
[28] S.J. Tang, B.S. Lee, B.S. He. *MROrder: Flexible Job Ordering Optimization for Online MapReduce Workloads*, in Euro-Par'13, 2013.
[29] S.J. Tang, C. Yu, J.Z. Sun, B.S. Lee, T. Zhang, Z. Xu, and H.B. Wu. *EasyPDP: An Efficient Parallel Dynamic Programming Runtime System for Computational Biology*, in IEEE TPDS'12, 2012.
[30] S.J. Tang, C. Yu, B.S. Lee, C. Sun, and J.Z. Sun, *Adaptive Data Refinement for Parallel Dynamic Programming Applications*, in IEEE IPDPSW'12, 2012.
[31] *LTYARN: A Long-Term YARN Fair Scheduler*. http://sourceforge.net/projects/ltyarn/.
[32] C. A. Waldspurger and W. E. Weihl. *Lottery scheduling: flexible proportional-share resource management*. In OSDI 94, 1994.
[33] Y. Wang, W. Shi, *Budget-Driven Scheduling Algorithms for Batches of MapReduce Jobs in Heterogeneous Clouds*, IEEE Transaction on Cloud Computing, 2014
[34] W. Wang, B. C. Li, B. Liang. *Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers*. INFOCOM'14, 2014.
[35] H.Y. Wang, Q.F. Jing, R.S. Chen, B.S. He, Z.P. Qian, and L.D. Zhou. *Distributed systems meet economics: Pricing in the cloud*. HotCloud'10, 2010.
[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. *Spark: Cluster Computing with Working Sets*. HotCloud'10, pp. 10-16. 2010.
[37] J. Zhong, B. He *Towards GPU-Accelerated Large-Scale Graph Processing in the Cloud*, In CloudCom'13, pp.9-16, 2013.