Supplemental Material: Fair Resource Allocation for Data-Intensive Computing in the Cloud

Shanjiang Tang, Bu-Sung Lee, Bingsheng He

Abstract—To address the computing challenge of 'big data', a number of data-intensive computing frameworks (e.g., MapReduce, Dryad, Storm and Spark) have emerged and become popular. YARN is a de facto resource management platform that enables these frameworks running together in a shared system. However, we observe that, in cloud computing environment, the fair resource allocation policy implemented in YARN is *not* suitable because of its *memoryless* resource allocation fashion leading to violations of a number of good properties in shared computing systems. This paper attempts to address these problems for YARN. Both single-level and hierarchical resource allocations are considered. For single-level resource allocation, we propose a novel fair resource allocation mechanism called *Long-Term Resource Fairness (LTRF)* for such computing. For hierarchical resource allocation, we propose Hierarchical Long-Term Resource Fairness (H-LTRF) by extending LTRF. We show that both LTRF and H-LTRF can address these fairness problems of current resource allocation policy and are thus suitable for cloud computing. Finally, we have developed LTYARN by implementing LTRF and H-LTRF in YARN, and our experiments show that it leads to a better resource fairness than existing fair schedulers of YARN.

Keywords—MapReduce, Hadoop, Fair Scheduler, YARN, Cloud Computing, Long-Term Resource Fairness.

Appendix A
Proof of Theorem 1

Theorem 1: When $T_{wait} = 0$, H-LTRF allocates resources the same as the single-level LTRF. In contrast, when $T_{wait} = +\infty$, H-LTRF is the same as the naive approach.

Proof: In Algorithm 2 of the main file, H-LTRF detects/assumes a leaf node to be a starvation based on the following two conditions: (1). it has the lowest total allocated resources among all users; (2). it has been waiting for a longer time than T_{wait} without being allocated since its last time allocation.

If $T_{wait} = 0$, H-LTRF will choose the leaf node with the lowest total allocated resources to allocate resources immediately without delay, which is just equivalent to LTRF in the single-level resource allocation. On the other hand, when $T_{wait} = +\infty$, according to the condition (2), the starvation node has to wait (i.e., to be starved) until no other leaf nodes need resources. The resulting effect is exactly the same as the naive approach. Therefore, it holds for Theorem 1. □

Appendix B
Property Analysis for LTRF

Theorem 2: LTRF satisfies the sharing incentive property.

Proof: Consider a shared pay-as-you-use cloud computing system of R resources contributed by n clients with equal share (or monetary cost) over t period time. When pursuing individually with the same amount of money, 1). the amount of resources R_1 a client can receive is $\frac{R}{n}$; 2). Under R_1 resources, she can get at most $t \cdot R_1$ resources, i.e., $t \cdot \frac{R}{n}$. In contrast, with fair allocation of LTRF, a client can get at least $t \cdot \frac{R}{n}$ resources. Thus LTRF satisfies sharing incentive property. □

Theorem 3: (Cost-efficient Workload Incentive) Any client who submits cost-efficient workloads to the shared pay-as-you-use computing system could get benefits under LTRF.

Proof: Recall that LTRF focuses on the fairness over total resources with *lending agreement*. When a client's resource demand is less than its current share, she can *lend* unneeded resources out. Later when she needs more resources in the future, she can get extra amount of resources back from others that she lent before. Reversely, if she submits lots of dirty workloads to the system when her *true* demand is less than her share, she will loose opportunity to get more extra sources, especially when she has lots of important and urgent workloads to compute later. Hence, LTRF meets the cost-efficient workload incentive property. □

Theorem 4: LTRF is resource-as-you-pay fairness in a shared computing system.

Proof: Each client in a *shared* computing system has right to enjoy at least the amount of resources she paid. One key factor that affects resource-as-you-pay fairness is the varied client's demands at different time (i.e., *unbalanced* workload which can be either less or larger than her current share). LTRF overcomes the unbalanced workload problem by considering the fairness at the level of total allocated resources and following *lending agreement*. It adjusts the current allocation of resources to each client dynamically according to her historical total allocated resources and current demand, making sure that the total resources a client received are *fair* with each other. Thus LTRF is resource-as-you-pay

• *S.J. Tang is with the School of Computer Science&Technology, Tianjin University, China. B.S. Lee and B.S. He are with the School of Computer Engineering, Nanyang Technological University, Singapore.*
E-mail: tashj@tju.edu.cn, {ebslee, bshe}@ntu.edu.sg.

fairness. □

Theorem 5: LTRF satisfies strategy-proofness property.

Proof: Theorem 3 has demonstrated that LTRF satisfies cost-efficient workload incentive property that can make a client be *truly* willing to yield out her unused resources when she does not need. On the other hand, it is possible that an overloaded client lies about her true demands to let her get more allocated resources in preemption with others at a time. But due to *lending agreement* requirement under LTRF, the consequence of lying in fact is a pre-overconsumption of her resources and she needs to *pay back* at a later time to others. Thus, lying cannot benefits her at all and our proof completes. □

Theorem 6: LTRF satisfies pareto efficiency property.

Proof: In the LTRF algorithm [40], there is a *discount*-based approach to incentivize users to preempt extra unused resources from others actively. It indicates that the utilization of system is fully maximized whenever there are pending tasks. That is, it is impossible for a client to get more resources without decreasing the resources of others and thereby our proof completes. □

Appendix C
Property Analysis for H-LTRF

Theorem 7: H-LTRF is sharing incentive.

Proof: The hierarchical resource allocation is a top-to-down recursively collective resource allocation across children of the same parent node. Consider a hierarchy example of Figure 1(b) of the main file. With H-LTRF, starting from the root node n_r, it allocates resources across its children n_1, n_2 with LTRF. According to Theorem 2, it can guarantee that n_1, n_2 are at least good or better than that without sharing. Similarly, under each internal node (e.g., n_1), LTRF is further adopted to perform resource allocation across its children (e.g., $n_{1,1}, n_{1,2}$). Such recursive LTRF allocation continues until the leaf node occurs. We see that at each time allocation, based on LTRF, we can guarantee that each child node of the same parent perform at least good or better than non-sharing case. Thus, H-LTRF meets the sharing incentive property. □

Theorem 8: H-LTRF satisfies cost-efficient workload incentive and resource-as-you-pay fairness properties.

Proof: H-LTRF performs the resource allocation recursively at each level with LTRF, based on total allocated resources, i.e., the historical resource allocation is taken into accounted. At each time allocation, according to Theorem 3, LTRF has a mechanism which enables to benefit nodes that yields unused resources to others in future time and punishes those nodes submitting cost-inefficient workloads. Thus, H-LTRF is cost-efficient workload incentive. Moreover, LTRF can ensure that the total allocated resources received by each node is proportional to its payment according to Theorem 4. Hereby, H-LTRF also meets resource-as-you-pay fairness property. □

Theorem 9: H-LTRF satisfies strategy-proofness property.

Proof: According to Theorem 5, LTRF can guarantee that users cannot get benefits by falsely reporting their demands. H-LTRF is an extension of LTRF for hierarchical resource allocation which allocates resources at each level with LTRF. It means that at each time allocation, H-LTRF can ensure that no nodes can get benefits by lying the system. Therefore, H-LTRF satisfies strategy-proofness property. □

Theorem 10: H-LTRF is pareto efficiency.

Proof: The *discount*-based approach of LTRF encourages users to preempt extra unused resources from others actively. H-LTRF is on top of LTRF for hierarchical resource allocation that allocates resources collectively with LTRF at each level. It implies that the utilization of system is fully maximized whenever there are pending tasks. Thereby H-LTRF is pareto efficiency. □

Finally, Table 1 summarizes the properties that are satisfied by MLRF, LTRF, and H-LTRF, respectively. We can see that MLRF is *not* suitable for pay-as-you-use computing system due to its lack of support for three important desired properties. In contrast, LTRF and H-LTRF, are *suitable* for pay-as-you-use computing system.

Property	Allocation Policy		
	MLRF	LTRF	H-LTRF
Sharing Incentive	√	√	√
Cost-Efficient Workload Incentive		√	√
Resource-as-you-pay Fairness		√	√
Strategy-Proofness		√	√
Pareto Efficiency	√	√	√

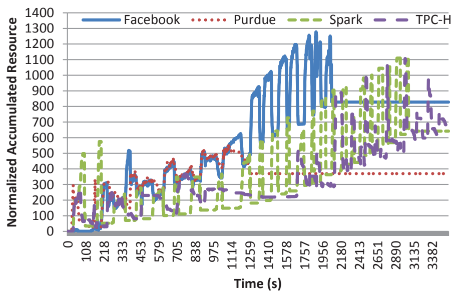
TABLE 1: List of properties for MLRF, LTRF and H-LTRF.

Appendix D
Adaptive Task Quantum Policy Evaluation

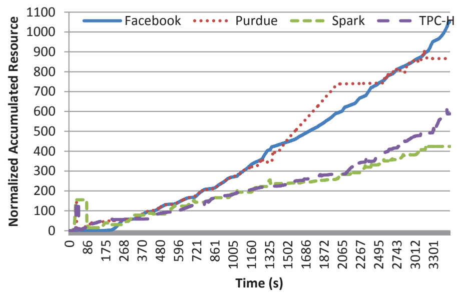
To demonstrate the importance and effectiveness of adaptive task quantum policy for YARN, this section performs a contrast experiment by showing the effects of accumulated resource results over time under the fixed time quantum and the adaptive task quantum mechanism proposed in Section 6.2.1 of the main file.

We consider a scenario where the configured task quantum (e.g., 600s) is much larger than the real task execution time of workloads. Figure 1 shows the compared accumulated results for LTYARN over time within one hour, which are normalized with respect to the system capacity. We have the following observations:

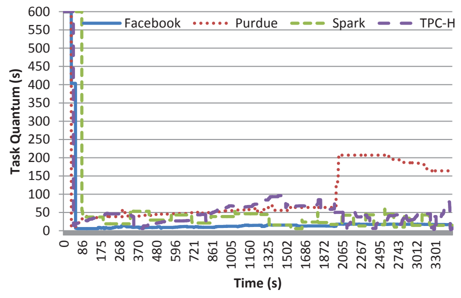
First, Figure 1(a) illustrates that the accumulated resource under the fixed task time quantum policy fluctuates up and down (i.e., unstable) significantly over time, making it unable to be an indicator for resource-as-you-pay fairness and failed to converge fast for fairness. This is due to the computation method for assumed execution time in the time quantum-based approach: 1). the assumed execution time for the completed task is equal to its real execution time; 2). for the running task, we compute its assumed execution time using the maximum value of the configured time quantum and its real execution time. Take Facebook workload for example. Its average task execution time is about 11s. At time 1439s, there are 107 running tasks, whose assumed execution time is 600, and its normalized accumulated resource is 1019. However, at time



(a) Normalized accumulated resources under the fixed task time quantum of 600s, with respect to the system capacity.



(b) Normalized accumulated resources with adaptive task quantum mechanism, with respect to the system capacity.



(c) Adaptive task quantum, initialized to be 600s.

Fig. 1: The adaptive task quantum results for LTYARN in one hour.

1450s (i.e., after 11s), there are 31 running tasks, indicating that at least 76 tasks completed during this period, causing a significant drop for its normalized accumulated resource (e.g., 630).

In contrast, with adaptive task quantum policy, as shown in Figure 1(b), the curves of accumulated resource become much smoother, making it good as an indicator for resource-as-you-pay fairness. Figure 1(c) shows the adaptive task quantum results over time for four workloads. We see that each workload has varied task quantum and our policy can adjust them dynamically, validating the effectiveness of our adaptive approach.

References

[1] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan, *Muppet: Mapreduce-style processing of fast data*, In VLDB Endow, vol.5, pp.1814–1825, 2012.

[2] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04, 2004.

[3] V.K. Vavilapalli, A.C. Murthy, C. Douglas, et al. *Apache Hadoop YARN: Yet Another Resource Negotiator*. In SOCC'13, pp.1-16, 2013.

[4] M. Ming and H., Marty. *Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows*, in SC'11, pp.1-12, 2011.

[5] H.Y. Wang, Q.F. Jing, R.S. Chen, B.S. He, Z.P. Qian, L.D. Zhou. *Distributed Systems Meet Economics: Pricing in the Cloud*, in HotCloud'10, pp.6-12, 2010.

[6] 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.

[7] A. Fight. *Introduction to Project Finance*, Butterworth-Heinemann, 2005.

[8] A. Ghodsi, M. Zaharia, S. Shenker and I. Stoica. *Choosy: Max-Min Fair Sharing for Datacenter Jobs with Constraints*, in EuroSys'13, 2013.

[9] C. Zhou, B.S. He. *Transformation-based monetary cost optimizations for workflows in the cloud*, in TCC'14, vol.2, pp.85-98, 2014.

[10] 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.

[11] Y. Shen, G. Chen, H. Jagadish, W. Lu, B.C. Ooi, B. Tudor, *Fast Failure Recovery in Distributed Graph Processing Systems*. In VLDB'15, 2015.

[12] Apache. *Storm*. http://storm-project.net/.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. *Spark: Cluster Computing with Working Sets*. In HotCloud'10, pp. 10-16. 2010.

[14] Apache. *TPC-H Benchmark on Hive*. https://issues.apache.org/jira/browse/HIVE-600.

[15] Apache. *Hadoop*. http://hadoop.apache.org.

[16] A. K. Agrawala and R. M. Bryant. *Models of memory scheduling*. In SOSP 75, 1975.

[17] L. Qin, J.X. Yu, L.J Chang, H. Cheng, C.Q. Zhang, and X.M. Lin. *Scalable Big Graph Processing in MapReduce*, In SIGMOD'14, pp.827-838, 2014.

[18] A. Turk, R.O. Selvitopi, H. Ferhatosmanoglu, and C. Aykanat. *Temporal Workload-Aware Replicated Partitioning for Social Networks*, In TKDE'14, vol. 26, pp.2832-2845, 2014.

[19] S. Wu, F. Li, S. Mehrotra, B.C. Ooi, *Query Optimization for Massively Parallel Data Processing*, In SOCC'11, pp.1-13, 2011.

[20] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy,S. Schenker,I. Stoica, *Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling*. In Proceedings of EuroSys, pp. 265-278, 2010.

[21] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. *Proportionate progress: A notion of fairness in resource allocation*. In Algorithmica, vol.15, pp.600625, 1996.

[22] Apache. *Hadoop MapReduce Next Generation-Fair Scheduler*. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[23] 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.

[24] A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, I. Stoica. *Hierarchical Scheduling for Diverse Datacenter Workloads*. SOCC'13, 2013.

[25] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu. *Hive- A Petabyte Scale Data Warehouse Using Hadoop*. ICDE, pp. 996-1005, 2010.

[26] G. Sabin, G. Kochhar, P. Sadayappan. *Job Fairness in Non-Preemptive Job Scheduling*. ICPP, pp. 186-194, 2004.

[27] R. Jain, D. M. Chiu, and W. Hawe. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Technical Report EC-TR-301, 1984.

[28] H.K. Liu, B.S. He. *Reciprocal Resource Fairness: Towards Cooperative Multiple-Resource Fair Sharing in IaaS Clouds*, in SC'14, 2014.

[29] J. Ngubiri, M. V. Vliet. *A Metric of Fairness for Parallel Job Schedulers*. Journal of Concurrency and Computation: Practice & Experience. Vol 21, PP. 1525-1546, 2009.

[30] H. N. Zhao, R. Sakellariou. *Scheduling multiple DAGs onto heterogeneous systems*. IPDPS, pp. 159-172, 2006.

[31] M.A. Bhuiyan, M.A. Hasan, *An Iterative MapReduce based Frequent Subgraph Mining Algorithm*. In TKDE'14, 2014.

[32] H. Arabnejad, J. Barbosa. *Fairness Resource Sharing for Dynamic Workflow Scheduling on Heterogeneous Systems*, ISPA, pp. 633-639, 2012.

[33] F. Ahmad, S. Y. Lee, M. Thottethodi, T. N. Vijaykumar. *PUMA: Purdue MapReduce Benchmarks Suite*. ECE Technical Reports, 2012.

[34] GitHub. *Facebook workload traces*. https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository.

[35] Apache. *Hive performance benchmarks*. https://issues.apache.org/jira/browse/HIVE-396.

[36] PUMA Datasets. http://web.ics.purdue.edu/~fahmad/benchmarks/datasets.htm.

[37] C.A. Waldspurger, W. E. Weihl. *Lottery Scheduling: Flexible Proportional-Share Resource Management*. In OSDI'94, 1994.

[38] A. Demers, S. Keshav, S. Shenker. *Analysis and Simulation of a Fair Queuing Algorithm*. In SIGCOMM'89, pp. 1-12, 1989.

[39] N. Cao, C. Wang, M. Li, K. Ren. *Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data*. In TPDS'14, vol. 25, pp.222-233, 2014.

[40] S.J Tang, B.S. Lee, B.S. He, H.K Liu, *Long-Term Resource Fairness: Towards Economic Fairness on Pay-as-you-use Computing Systems*. In ICS'14, pp. 251-260, 2014.

[41] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. *Multi-resource fair queueing for packet processing*, SIGCOMM'12, pp.1-12, 2012.

[42] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, I. Stoica. *FairCloud: Sharing the Network in Cloud Computing*. In Sigcomm'12, pp. 187-198, 2012.

[43] F. Fassetti, G. Greco, G. Terracina. *Mining Loosely Structured Motifs from Biological Data*, in IEEE TKDE'08, vol. 20, pp. 1472-1489, 2008.

[44] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, G. OShea. *Chatty Tenants and the Cloud Network Sharing Problem*. In NSDI'13, pp.171-184, 2013.

[45] D. Shue, M.J. Freedman, and A. Shaikh. *Performance Isolation and Fairness for Multi-Tenant Cloud Storage*. In OSDI'12, 2012.

[46] Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, Q. Xu, *Es 2: A cloud data storage system for supporting both oltp and olap*. In ICDE'11, pp. 291-302. 2011.

[47] A. Greenberg, J. Hamilton, D.A. Maltz and P. Patel. *The Cost of a Cloud: Research Problems in Data Center Networks*. ACM SIGCOMM Computer Communication Review. vol. 39, pp. 68-73, 2009.

[48] C. Delimitrou, C. Kozyrakis. *Quasar: Resource-Efficient and QoS-Aware Cluster Management*, ASPLOS'14, pp. 127-144, 2014.

[49] R.S. 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. 1-13, 2012.

[50] H.K. Liu, B.S. He. *F2C: Enabling Fair and Fine-grained Resource Sharing in Multi-tenant IaaS Clouds*, TPDS '12, pp. 1-1, 2015.

[51] S.J. Tang, B.S. Lee, B.S. He. *Towards Economic Fairness for Big Data Processing in Pay-as-You-Go Cloud Computing*, Cloudcom'14, pp. 638-643, 2014.