Long-term Fair Scheduler for Multiple-scheduling Classes

Abstract

1 Introduction

Cloud computing workload introduction: batch jobs and production jobs. Jobs are classified into multiple scheduling classes.

Introduce multiple goals: performance and fairness.

Motivation example: strictly ensuring performance hurts fairness and vise versa.

Highlight our focus like Figure 1: long-term fairness plus bounded priority.

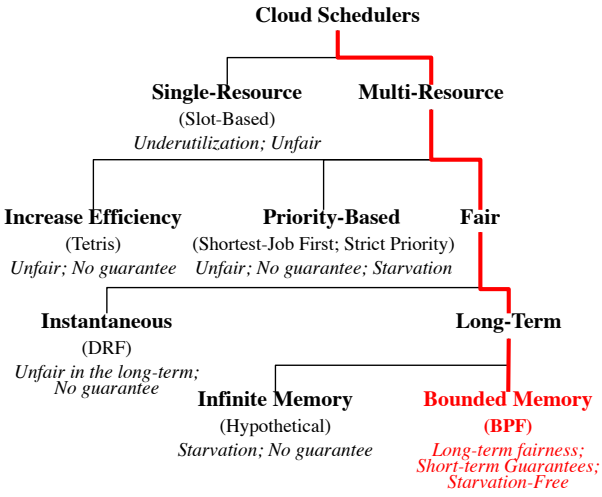

Figure 1: Design space.

Introduce high level of challenges:

1. fairness and performance do not co-exist.
2. Multiclasses of priorities.

Summarize the contributions:

Algorithm Design

System Design

Evaluation based both testbed experiments and large scale simulations

2 Motivation

2.1 Benefits of Long-term Fairness scheduler

Performance is guaranteed for priority jobs

Long-term fairness is guaranteed for low priority jobs.

2.2 Desired Properties

2.3 Existing Policies

3 Approach

Our goal is to design a long-term fair scheduler that supports different types of queues and achieve good utilization.

It is widely known that service curve, based on network calculus, provides performance guarantee for communications networks. More than just a formalism, it enables to analyze complex systems and to prove deterministic bounds on delays, backlogs and other Quality-of-Service (QoS) parameters. However, under multi-resource environment, the service curve, which is based on worst-case performance, is too conservative, especially the type of resources become large.

To resolve the problem, we propose a fair scheduling algorithm based on queue-based service curve.

Queue-i is said to be guaranteed a service curve $S_i(\cdot)$, if $\forall$ time t_2, there exists time t_1 where $t_1 \in A_i^{t_2}$, and $A_i^{t_2}$ is the set of arrival times between $(t_2 - T, t_2]$ of all job from Queue-i, so that $S_i(t_2 - t_1) \succeq W_i(t_1, t_2)$, where T is a burst parameter defined by the system. The main difference between traditional service for bandwidth allocation is that we have to take queue's demand into consideration. For example, sometimes the demand, compared to the

service curve, is not enough. Then we should allow users to have bursts later.

Intuitively, we ask systems to have bounded memory, where queues can have burst for limited time within a preset interval (with length T) from look-back.

Xiao says: Updated: Now $[L_i]$ only represents all users at priority level l_i

Algorithm:

Presetting

Specifying below parameters:

1. Burstiness parameter $T \in (0, \infty]$;

Let the level set be $\mathscr{L} = \{l_1, l_2, \cdots, l_L\}$ where l_1 is the highest priority. Denote $[L_j]$ as the set of all admitted queues with priority level l_j.

Admission Control

$\forall$ incoming latency sensitive queue Q_i, let its level be l_i, system first reports parameter T to Q_i and ask Q_i for a service curve between time $[0, T]$, where the service curve of queue Q_i is defined as a two-piece linear function

$$S_i(t) = \begin{cases} \alpha_i t & \text{if } 0 \le t < T_i' \\ \beta_i t & \text{if } T_i' \le t \le T \end{cases}$$

with $\alpha_i \ge \beta_i$. $\alpha = \beta$ for the batch job users. (TODO: how to compute β).

1. (Resource condition) Check if Queue-i is admitted, whether the sum of burst period from the set $[L_i]$ exceeds the capacity of the cluster : $\sum_{k \in [L_i]} \alpha_k \preceq C$.

2. (Fairness condition) Check whether the resource allocated to this service curve is fair at aggregated $[0, T]$ time by verifying $S_i(T) \preceq \min(CT - \sum_{j \in [L_i]} S_j(T), \frac{CT}{n})$ where n is the size of $[L_i]$. (Tan says it must be $n+1$)

3. If either step is violated, accept queue with best-effort performance.

4. If both guarantee and fairness conditions are satisfied, (Admit Queue-i to guarantee level l_i.) and if the level of Queue Q_i is l_1, system accept Queue i with unlimited time (hard) guarantee ; if not, system accept Queue i with temporary (soft) guarantee;

Resource Allocation

(TODO: how to deal with priority levels here?)

1. For all queues from H and S, starting from level l_1 to l_L, At every time slot $[t, t+1)$, $Y_i(t)$ is the minimum amount of resource queue i should receive from its service curve: $Y_i(t) = \min_{t_1 \in [t-T,t)}(S_i(t+1-t_1) - \sum_{\tau=t_1}^{t-1} a_i^{sc}(\tau), \alpha_i)$. Let the demand of Queue i at time t be $d_i(t)$, then the allocation from its service curve should be $a_i^{sc}(t) = \min\{Y_i(t), d_i(t)\}$.

2. Calculate leftover resource R, allocate them based a round-robin within $\mathbb{H}, \mathbb{S}$ with strict priority. For example, assume there are 5 queues Q_1, Q_2, Q_3, Q_4, Q_5 from priority level l_1, then the round-robin allocation will give all remaining resource to queue Q_1 during $[0, T)$, to queue Q_2 during $[T, 2T]$ etc. unless there is not enough demand

3. Within every queue, FIFO will be applied.

Algorithm 1 NAME Scheduler

```
procedure DynamicSchedule()
    Admit(ℚ)
    Allocate(ℍ, 𝕊, 𝔼)
end procedure

function Admit(LQs ℚ)
    for all LQ Q ∈ ℚ do
        if fairness condition and resource condition satisfied then
            Admit Q to guarantee G
        else
            Admit Q to elastic 𝔼
        end if
    end for
    Sort {ℍ, 𝕊}, {𝔼} seperately based on priority level, break ties based on arrival time;
    return {ℍ, 𝕊, 𝔼}
end function

function Allocate(ℍ, 𝕊, 𝔼, t)
    R = C
    for all LQ Q_i ∈ {ℍ, 𝕊} do
        Y_i(t) = min_{t1∈[t−T,t)}(S_i(t + 1 − t1) − Σ_{τ=t1}^{t−1} a_i^sc(τ), α_i);
        a_i^sc(t) = min{Y_i(t), d_i(t), R};
        R = R − a_i^sc(t);
    end for
    if R > 0 then
        Allocate remaining resource with from Queue 𝔼 with DRF
        Update R
    end if
    if R > 0 then
        Allocate remaining resource with from Queue {ℍ, 𝕊} with strict priority
    end if
end function
```

4 System Design and Implementation

We implement our solution on Yarn Scheduler of Hadoop 2.7.2 [33].

Overview on cluster resource manager.

Admission control

Resource allocation

5 Evaluation

Setup. We use the real-trace workloads from Google.

Workload

Experimental setup

Simulation setup

Metrics

5.1 Experimental Results

5.1.1 Performance Guarantee

5.1.2 Long-term Fairness

5.2 Simulations

6 Related Work

Bursty Applications in Big Data Clusters Big data clusters experience burstiness from a variety of sources, including periodic jobs [16, 2, 5, 30], interactive user sessions [4], as well as streaming applications [35, 3, 1]. Some of them show predictability in terms of inter-arrival times between successive jobs (e.g., Spark Streaming [35] runs periodic mini batches in regular intervals), while some others follow different arrival processes (e.g., user interactions with large datasets [4]). Similarly, resource requirements of the successive jobs can sometimes be predictable, but often it can be difficult to predict due to external load variations (e.g., time-of-day or similar patterns); the latter, without NAME, can inadvertently hurt batch queues (§**??**).

Multi-Resource Job Schedulers Although early jobs schedulers dealt with a single resource [36, 6, 25], modern cluster resource managers, e.g., Mesos [23], YARN [33], and others [30, 34, 11], employ multi-resource schedulers [19, 18, 20, 21, 10, 28, 8] to handle multiple resources and optimize diverse objectives. These objectives can be fairness (e.g., DRF [19]), performance (e.g., shortest-job-first (SJF) [18]), efficiency (e.g., Tetris [20]), or different combinations of the three (e.g., Carbyne [21]). However, *all* of these focus on instantaneous objectives, with instantaneous fairness being the most common goal. To the best of our knowledge, NAME is the first multi-resource job scheduler with long-term memory.

Handling Burstiness Supporting multiple classes of traffic is a classic networking problem that, over the years, have arisen in local area networks [17, 31, 32, 9], wide area networks [29, 26, 24], and in datacenter networks [27, 22]. All of them employ some form of admission control to provide quality-of-service guarantees. They also consider only a single link (i.e., a single resource). In contrast, NAME considers multi-resource jobs and builds on top this large body of existing literature.

BVT [15] was designed to work with both real-time and best-effort tasks. Although it prioritizes the real-time tasks, it cannot guarantee performance and fairness.

Expressing Burstiness Requirements NAME is not the first system that allows users to express their time-varying resource requirements. Similar challenges have appeared in traditional networks [32], network calculus [13, 14], datacenters [27, 7], and wide-area networks [29]. Akin to them, NAME requires users to explicitly provide their burst durations and sizes; NAME tries to enforce those requirements in short and long terms. Unlike them, however, NAME explores how to allow users to express their requirements in a multi-dimensional space, where each dimension corresponds to individual resources. One possible way to collapse the multi-dimensional interface to a single dimension is using the notion of *progress* [12, 19]; however, progress only applies to scenarios when a user's utility can be captured using Leontief preferences.

References

[1] Trident: Stateful stream processing on Storm. http://storm.apache.org/documentation/Trident-tutorial.html, 2017.

[2] S. Agarwal, S. Kandula, N. Burno, M.-C. Wu, I. Stoica, and J. Zhou. Re-optimizing data parallel computing. In *NSDI*, 2012.

[3] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. MillWheel: Fault-tolerant stream processing at Internet scale. 2013.

[4] S. Alspaugh, B. Chen, J. Lin, A. Ganapathi, M. Hearst, and R. Katz. Analyzing log analysis: An empirical study of user log mining. In *LISA*, 2014.

[5] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed popularity content in MapReduce clusters. In *EuroSys*, 2011.

[6] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in MapReduce clusters using Mantri. In *OSDI*, 2010.

[7] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *OSDI*, 2014.

[8] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica. Hierarchical scheduling for diverse datacenter workloads. In *SoCC*, 2013.

[9] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), Dec. 1998. Updated by RFC 3260.

[10] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *OSDI*, 2014.

[11] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and efficient parallel processing of massive datasets. In *VLDB*, 2008.

[12] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *NSDI*, 2016.

[13] R. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[14] R. Cruz. A calculus for network delay, Part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

[15] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. *ACM SIGOPS Operating Systems Review*, 33(5):261–276, 1999.

[16] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *Eurosys*, 2012.

[17] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, 1995.

[18] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

[20] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *SIGCOMM*, 2014.

[21] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *OSDI*, 2016.

[22] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can JUMP them! In *NSDI*, 2015.

[23] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI*, 2011.

[24] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[25] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP*, 2009.

[26] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[27] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. Silo: Predictable message completion time in the cloud. In *SIGCOMM*, 2015.

[28] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *USENIX ATC*, 2015.

[29] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, et al. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *SIGCOMM*, 2015.

[30] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *EuroSys*, 2013.

[31] S. Shenker, D. D. Clark, and L. Zhang. A scheduling service model and a scheduling architecture for an integrated services packet network. Technical report, Xerox PARC, 1993.

[32] I. Stoica, H. Zhang, and T. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority service. In *SIGCOMM*, 1997.

[33] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In *SoCC*, 2013.

[34] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *EuroSys*, 2015.

[35] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant stream computation at scale. In *SOSP*, 2013.

[36] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI*, 2008.