BPF: Mitigating the Burstiness-Fairness Tradeoff in Multi-Resource Clusters

Poster #33: , 2 Pages

1 Motivation

Even though batch, interactive, and streaming applications all care about performance, their notions of performance are different. For instance, while the average completion time can sufficiently capture the performance of a throughout-sensitive batch-job queue (TQ) [4], interactive sessions and streaming applications form latency-sensitive queues (LQ): each LQ is a sequence of small jobs following an ON-OFF pattern. For these jobs [6], individual completion times or latencies are far more important than the average completion time or the throughput of the LQ.

Indeed, existing "fair" schedulers are inherently unfair to LQ jobs: when LQ jobs are present (ON state), they must share the resources equally with TQ jobs, but when they are absent (OFF state), batch jobs get all the resources. In the long run, TQs receive more resources than their fair shares because today's schedulers make *instantaneous* decisions [2].

Therefore, we ponder a fundamental question: *can we enable latency-sensitive LQs and throughput-sensitive TQs to coexist, where LQs are permitted short-term resource bursts while ensuring long-term fairness between the two, as well as maximizing LQs served with resource guarantees?*

In this work, we explore opportunities and challenges in achieving three desired properties: (i) allowing LQs to enjoy short-term high resource usage during their ON periods to minimize *individual* job response times; (ii) maintaining the same *long-term* fairness as existing "fair" allocation policies by preventing arbitrarily large bursts; and (iii) maximizing system utilization and LQs served with resource guarantees.

The key opportunity lies in the observation that TQs do not care about allocated resources in the short term, as long as the long-term averaged resource share is the same. However, there are several challenges. There is a fundamental tradeoff between "hard" resource guarantee to LQs and isolation protection for TQs. In addition, naive admission control may result in very few LQs admitted.

2 BPF: Bounded Priority Fairness

In this section, we provide a high-level overview of BPF, which aims at a balance on the potentially conflicting properties. Please refer to our technical report [2] for details.

2.1 Problem Settings

We consider a system with K types of resources. The capacity of resource k is denoted by C^k. The system resource capacity is therefore a vector $\overrightarrow{C} = \langle C^1, C^2, ..., C^K \rangle$.

LQ-i's demand comes from a series of jobs during her ON status. We denote by $T_i(n)$ and $t_i(n)$ the starting time and duration of her n-th ON period, respectively. Therefore, her n-th OFF period is $[T_i(n) + t_i(n), T_i(n+1)]$. We also denote the demand during her n-th ON period by a vector $\overrightarrow{d_i}(n) = \langle d_i^1(n), d_i^2(n), ..., d_i^K(n) \rangle$, where $d_i^k(n)$ is the demand on resource-k. We assume users report their estimated demand for presentation simplicity.

To enforce the long-term fairness, her total demand during each period $[T_i(n), T_i(n+1)]$, i.e., $\overrightarrow{d_i}(n) * t_i(n)$, should not exceed her fair share, which can be calculated by fair scheduler, i.e., $\frac{\overrightarrow{C}}{N}$ when there are N queues admitted by BPF.

Each LQ requests the resource demand $\overrightarrow{d_i}(n)$ for a duration that equals to the length of her ON period $t_i(n)$. TQs' jobs are queued at the beginning with much larger demand than each burst of LQs.

2.2 Solution Approach

Admission control procedure BPF classifies admitted LQs and TQs into three classes: LQs admitted with hard resource guarantee ($\mathbb{H}$), LQs admitted with soft resource guarantee ($\mathbb{S}$), and elastic queues that can be either LQs or TQs ($\mathbb{E}$).

Before admitting LQ-i, BPF checks if adding it invalidates any resource guarantees committed for LQs in $\mathbb{H} \cup \mathbb{S}$, i.e., the following *safety condition* needs to be satisfied:

$$\overrightarrow{d_j}(n) * t_j(n) \leq \frac{\overrightarrow{C}\,(T_j(n+1) - T_j(n))}{|\mathbb{H}| + |\mathbb{S}| + |\mathbb{E}| + 1}, \forall n, \forall j \in \mathbb{H} \cup \mathbb{S}, \quad (1)$$

where $|\mathbb{H}| + |\mathbb{S}| + |\mathbb{E}|$ is the number of already admitted queues. If (1) is not satisfied, LQ-i is rejected. Otherwise, it is safe to admit LQ-i into one of the three classes.

For LQ-i to have some resource guarantee, either hard or soft, her own total demand should not exceed her long-term fair share. Formally, the *fairness condition* is

$$\overrightarrow{d_i}(n) * t_i(n) \leq \frac{\overrightarrow{C}\,(T_i(n+1) - T_i(n))}{|\mathbb{H}| + |\mathbb{S}| + |\mathbb{E}| + 1}, \forall n. \quad (2)$$

If only condition (1) is satisfied but (2) is not, LQ-i is added to $\mathbb{E}$. If both conditions (1) and (2) are satisfied, it is

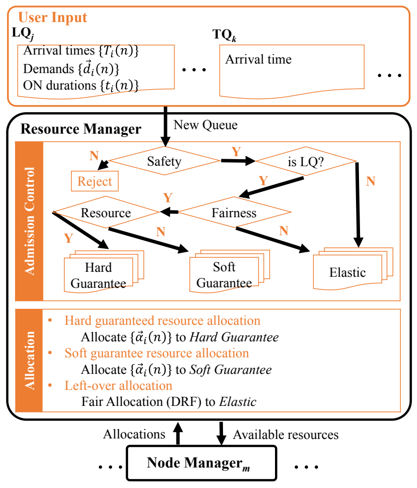

User Input
LQ$_j$
Arrival times $\{T_i(n)\}$
Demands $\{\vec{d}_i(n)\}$
ON durations $\{t_i(n)\}$
TQ$_k$
Arrival time
Resource Manager
New Queue
Admission Control
Safety
is LQ?
Reject
Resource
Fairness
Hard Guarantee
Soft Guarantee
Elastic
Y
N
Allocation
Hard guaranteed resource allocation
Allocate $\{\vec{a}_i(n)\}$ to Hard Guarantee
Soft guarantee resource allocation
Allocate $\{\vec{a}_i(n)\}$ to Soft Guarantee
Left-over allocation
Fair Allocation (DRF) to Elastic
Allocations
Available resources
Node Manager$_m$

Figure 1: Enabling bounded prioritization with long-term fairness in a multi-resource cluster. BPF-related changes are shown in orange.

safe to admit LQ-i to $\mathbb{H}$ or $\mathbb{S}$. If there are enough uncommitted resources (*resource condition* (3)), LQ-i is admitted to $\mathbb{H}$. Otherwise it is added to $\mathbb{S}$.

$$\vec{d}_i(n) \leq \vec{C} - \sum_{j \in \mathbb{H}} \vec{d}_j(t), \forall n, t \in [T_i(n), T_i(n) + t_i(n)]. \tag{3}$$

For TQ-j, BPF simply checks the safety condition (1). If it is satisfied, TQ-j is added to $\mathbb{E}$. Otherwise TQ-j is rejected.

Guaranteed resource provisioning procedure For each LQ-i in $\mathbb{H}$, during each first stage $[T_i(n), T_i(n) + t_i(n)]$ of her ON/OFF cycles, BPF allocates resources according to her demand $\vec{a}_i(t) = \vec{d}_i(n)$. In practice, LQ-$i$ may not fully receive the guaranteed resources and more resources are provided until her resource consumption reaches $\vec{d}_i(n) * t_i(n)$.

For each LQ-i in $\mathbb{S}$, the procedure is similar except that resources are allocated when there are uncommitted resources.

After every LQ in $\mathbb{H}$ and $\mathbb{S}$ is satisfied, remaining resources are allocated to queues in $\mathbb{E}$ using DRF [3].

Spare resource allocation procedure If some allocated resources are not used, they are further shared by TQs and LQs with unsatisfied demand. This maximizes system utilization.

2.3 Properties of BPF

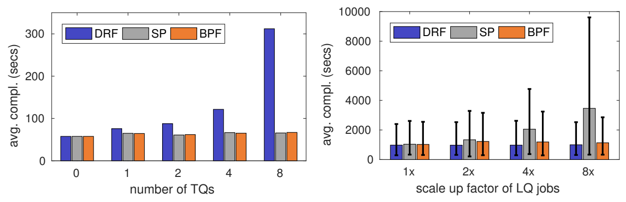
The safety condition and fairness condition ensure the long-term fairness for all TQs. For LQs in $\mathbb{H}$, they have hard resource guarantee and therefore the optimal completion time. For LQs in $\mathbb{S}$, they have resource guarantee whenever possible, and only need to wait after LQs in $\mathbb{H}$ when there is a conflict. Therefore, their performance is near optimal and better than that under fair allocation policies. The addition of $\mathbb{S}$ allows more LQs to be admitted with resource guarantee. Finally, we allocate spare resources whenever there is some, so system utilization is maximized.

2.4 Enabling BPF in Cluster Managers

We now describe how we have implemented BPF on Apache YARN. We use standard techniques for demand estimation, and additional details can be found in [2]. A key benefit of BPF is its simplicity of implementation: we have implemented it in YARN using only 600 lines of code. We made three main changes for user input, admission control, and resource scheduler – all in the *resource manager* (RM). We do not modify *node manager* (NM) or application master (AM). Figure 1 depicts our design.

3 Evaluation

We evaluated BPF using the big data benchmark – BigBench (BB) [1]. We ran experiments on a 40-server CloudLab cluster. We setup Tez atop YARN for the experiment.



(a) Avg. compl. of LQ jobs **(b)** Avg. compl. of TQ jobs

Figure 2: BPF can closely approximate the LQ performance of Strict Priority and the long-term fairness for TQs of DRF.

We compare BPF to two baseline policies: Strict Priority (SP) [5] and DRF [3]. While SP prioritizes LQ whenever possible, DRF ensures instantaneous fairness.

In Figure (2a), there are a single LQ and multiple TQs. When there are no TQs, the average completion times of LQ jobs across three schedulers are the same (57 seconds). As the number of TQs increases, the performance of DRF significantly degrades because DRF tends to allocate less resource to LQ jobs. In contrast, BPF and SP give the highest priority to LQs and the average completion times of LQ jobs are almost unchanged (65 seconds).

When we have too many LQ jobs, Figure (2b) highlights that SP allocates too much resource to LQ jobs that significantly hurts TQ jobs, while DRF maintains similar average completion times of TQ jobs. BPF performs closely to DRF.

In summary, BPF speeds up latency-sensitive jobs by $4.66\times$ compared to DRF, while still maintaining long-term fairness. In the meantime, BPF improves the average completion times of throughput-sensitive jobs by up to $3.05\times$ compared to Strict Priority. More results can be found in [2].

4 References

[1] Big-Data-Benchmark-for-Big-Bench. https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench.

[2] BPF: Mitigating the Burstiness-Fairness Tradeoff in Multi-Resource Clusters – Technical Report. https://goo.gl/e5oypF.

[3] 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.

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

[5] L. Kleinrock and R. Gail. *Queueing systems: Problems and solutions*. Wiley, 1996.

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