===
SIGCOMM '17 Review #212A

Paper #212: BPF: Mitigating the Burstiness-Fairness Tradeoff in
Multi-Resource Clusters

Overall merit: 2. Top 50% but not top 25% of
submitted papers
Reviewer expertise: 3. Knowledgeable
Technical Quality: 3. Only minor issues

===== Paper summary =====

This paper is dealing with admission control in datacenter environments. The problem that the authors aim to solve is that data centers tend to have two very different types of jobs: i) latency sensitive, and ii) throughput sensitive. Prior work has focused on either strict prioritization that benefits latency sensitive jobs or fairness that ignores low latency requirements. The authors propose BPF that is seen as an intermediate solutions that prioritizes low latency jobs but ensures that across a longer time scale fairness is also achieved, such that throughput sensitive jobs can still achieve their objective. The authors implement their approach and evaluate it in a testbed and in simulation.

===== Reasons to accept the paper =====

- finding a solution that can accommodate both latency and throughput jobs is interesting
(We can strengthen this by allowing multiple classes from the theoretical perspective.)
- the authors have implemented their solution and tested it in a testbed and in simulation

===== Issues that could prevent acceptance =====

- the proposed solution does not really present any difficult technical challenge (technical challenge: more than 2 objectives, e.g., by service curve; theoretical guarantees instead of heuristic, e.g., compared to Carbyne as pointed by some sigmetrics reviewer)
- BPF requires as input the duration of the ON periods for low latency traffic. It is unclear to me whether jobs can know this in advance so that they can submit it to the scheduler (Spark setting, or estimation. As pointed by some sigmetrics reviewer, if estimation is used, simply reporting the mean may not be the best choice. This can be another potential theoretical contribution.)
- It is unclear to me how well a latency sensitive job would perform when it transitions from the prioritized to the fair allocation. Is this a meaningful way to enable such jobs to execute well? (We didn't make this point clear enough. We should highlight that an LQ, once admitted, would perform exactly the same as under SP as long as it abides by its original request.)

===== Comments for author =====

In general, I like the premise of the paper. Datacenters accommodate two different types of workloads. Existing schemes focus on optimization criteria that cannot accommodate latency sensitive workloads well. Therefore, the authors propose a scheduler that prioritizes latency sensitive jobs but ensures that fairness across latency and throughput sensitive jobs is achieved at longer time scales.

The fundamental assumption here is "if bursts are not too large to hurt the long-term fairness, they are given higher priority so jobs can be completed as quickly as possible".

The authors propose a solution that given knowledge of how long the burst will be (duration of ON period) they can come up with such a schedule. My main question behind the solution is how easy it is for latency sensitive jobs to express their ON duration. This topic is never addressed in the paper.
(Spark setting, or estimation. As pointed by some sigmetrics reviewer, if estimation is used, simply reporting the mean may not be the best choice. This can be another potential theoretical contribution. Given this reviewer emphasizes this point and the stochastic optimization using estimation (distribution) is doable but nontrivial, I think we should try to finish this in the following weeks.)

(Meanwhile, is it possible to predict the length of the ON duration from some workload traces? This can be a very strong motivation and greatly improve the practicality of our approach.)

Second, I am unclear what the performance implications will be when a latency sensitive job gets strict priority at the beginning but then falls back to its fair allocation. Has the job managed to perform according to its requirements? In a sense, I could see situations that unless the latency sensitive job gets prioritized across its lifetime, it does not matter whether it was prioritized at the beginning.

(Obviously, this reviewer didn’t get the point of “bounded” priority. This point raised highlights for the system community especially, we should make everything crystal clear as otherwise they cannot follow.)

()

In Section 4.2 the authors comment on the impact of inaccuracies in the demand estimation. They also need to address inaccuracies in the duration of the ON periods.

In the experimental setup TQ jobs are submitted all at the beginning while LQ jobs arrive sequentially. Why is this a good experimental setup? TQ jobs can also arrive at different points in time. Do you mean that you are holding them back until the next scheduling decision?

(This sounds reasonable. Tan, how long will it take to add some TQs arrives over time?)

In 5.2.2 you mention that average completion time for LQs is 57 seconds while the average stage 1 duration is 27 seconds "because of inefficient resource packing and allocation overheads". 57 seconds is more than 2x27 seconds! In such a case, I wonder whether the right problem to work on is the scheduling itself or addressing the overheads.

(I think we discussed this point before. If I recall correctly, Tan said that it takes time to release the resources. This sounds like a great opportunity to present some system challenge and make some contribution. Is it possible to make those TQs release resources earlier to prepare for the upcoming bursts of LQs?)

In Section 5 you discuss different configurations for LQs and TQs. How can a data center operator decide how many LQs and TQs they need?

(As Xiao also pointed out, instead of using FCFS, some backpack algorithm can be used to decide the admission.)

(Mosharaf: we actually discussed one great potential. From game theoretic perspective, if users can change their resource demands, say over multiple links, the backpack algorithm may incentivize users to adjust to improve the overall utilization of the system.)

All in all, the problem is interesting, but I am not convinced about the practicality of the work. The difficulty in collecting inputs, the difficulty in configuring the number queues and associated implications (I'm not sure what this means), the unknown impact of prioritizing and then lowering the rate of a latency sensitive job (this can be solved by better presentation), the overheads that the authors have observed experimentally and that overwhelm the performance improvement that they achieve, make me hesitant to recommend acceptance.

More detailed comments:
- using "she" and "her" for queues is very distracting in the text (fixed already in the sigmetrics submission)
- In Section 4.1 the authors state "the BPF scheduler needs additional parameters for LQs; namely, arrival times and demands". You also need the durations of the ON periods.
- Figure 8: the y-axis should say "avg. completion time of TQs" **(still unchanged!)**

(Overall, quite a few comments can be addressed by improving the presentation, while this reviewer did give some good suggestions!)

==
SIGCOMM '17 Review #212B

Paper #212: BPF: Mitigating the Burstiness-Fairness Tradeoff in

Multi-Resource Clusters

Overall merit: 2. Top 50% but not top 25% of submitted papers
Reviewer expertise: 1. No familiarity
Technical Quality: 3. Only minor issues

===== Paper summary =====

This paper presents BPF, a scheduler for datacenters that supports latency-sensitive and throughput-sensitive applications. BPF allocates resources to latency-sensitive applications as soon as possible to allow for fast completion times, but also enforces long-term fairness between latency-sensitive and throughput-sensitive applications. Evaluation in a real testbed and using simulations shows that BPF performs better than two competing approaches.

===== Reasons to accept the paper =====

+ Experimental evaluation of real prototype, simulations
+ Solution to practical problem

===== Issues that could prevent acceptance =====

- BPF solves a very specific, known problem (but there is no solution yet, which we should highlight using examples in the intro/motivation sections. Moreover, we should show by examples simple solutions do not work. There are significant challenges from both theory and system sides.); unclear how it generalizes

===== Comments for author =====

Although this paper presents a working solution to a practical problem, the scientific contributions are unclear. Scheduling of latency-sensitive processes has been studied for two decades. BVT [A] was proposed back in 1999 and is analogous to BPF for CPU scheduling; it even considers resource reservation and admission control (for "hard real-time applications"). There is a large body of work we could draw on to schedule latency-sensitive applications on datacenters, but the paper does not seem to take it into account. (We should show BVT does not solve the problem. In HUG, we motivated by the fact that the utilization of DRF can be arbitrarily low. We need something similar here.)
Another issue is that BPF, as proposed, addresses scheduling in one specific scenario. If new applications or computing abstractions

come along, BPF might not generalize. (I think this can be solved later or a follow-up paper, e.g., service curves)

Other comments

* The introduction mentions that prioritizing LQs would incentivize
  them to submit arbitrarily large jobs to starve TQs. I do not see
  the reason. (We should highlight strategyproofness, private vs public cloud at the very beginning as this is one of the main challenges.)

* If LQs are more valuable than TQs, why is it unacceptable to
  starve TQs? (Both of them are valuable. Perhaps we should give a weight/utility to each queue and let the system to maximize the total utility, which seems like a better optimization problem?)

* Section 3 refer to "stages", but these are not clearly defined.

* What happens for resource demands that are non elastic (say, e.g.,
  an application wants 6GB of RAM)? More generally, if an
  application requests X units of a resource, but only Y < X units
  are available, does the application wait or is it allocated
  partial resources before it can start? (We should clarify this)

* Are the LQ tasks in E worse-off than tasks in TQ? (Given H and
  S will use a large fraction of resources, LQ tasks in E might
  receive a very small fraction of resources.) (This reviewer didn't get our point regarding long-term fairness)

* Is dominant resource fairness enforced only for tasks in E or does
  it cover tasks in H and S too? (We should clarify this)

* What happens to rejected LQs and TQs? Do they get resubmitted?
  The rate of rejected LQs and TQs could be an interesting
  evaluation metric. (This can be solved largely by the utility maximization)

* In Section 5.1, how are LQ jobs scaled to reach maximum capacity
  of a single resource? (We should highlight there can be an upper bound, which can be accommodated by BPF)

* Why focus on a scenario with 1 LQ and 8 TQs?

* If you assume jobs in each class arrive periodically, it implies
  loss of generality. (NO, BPF can accommodate general case)

* Could one arbitrarily vary the fair share of LQ processes by

simply creating more LQs? (Yes, which is true for all resource allocations based on queues)

* I guess terminology is related to how datacenters schedulers work, but it might be a good idea to provide abstractions for queue, job, task, phase, stage, period... seems convoluted. (Good point!)

* In Section 5.3.2, it seems each LQ admits process into one of H, S, and E. I found this confusing. (Don't the admission control and allocation procedures get applied for each queue?) (We need to clarify this)

* In Section 5.4.1, why do overestimated jobs not suffer any delays? What if they come after an underestimated job? (We can add some sensitivity analysis if time/space allows. Meanwhile, adding the estimation into our current scheduler would help.)

* In Section 5.4.2, if task durations are given by the dataset, how do you increase average task run time? (We should clarify this. Tan, could you tell us what you mean by task and why it can change? Is waiting time included? Should we use task completion time or job completion time?)

[A] http://dl.acm.org/citation.cfm?id=319169

(This reviewer pointed out BVT, which is helpful. Other points are mainly related to presentation clarity.)

===
SIGCOMM '17 Review #212C

Paper #212: BPF: Mitigating the Burstiness-Fairness Tradeoff in Multi-Resource Clusters
----

Overall merit: 2. Top 50% but not top 25% of submitted papers
Reviewer expertise: 2. Some familiarity
Technical Quality: 3. Only minor issues

===== Paper summary =====

Existing distributed job schedulers optimize either for fairness or for priority. This leads to the dilemma that either throughput-sensitive jobs prevail over latency-sensitive jobs or vice versa. BPF is a new scheduler that solves this problem by allowing latency-sensitive jobs to "burst" in a short-term while guarding long-term fairness for throughput-sensitive jobs.

Evaluations show that, compared to Dominant Resource Fairness (DRF) and Strict Priority (SP), BPF can get the best of both worlds.

===== Reasons to accept the paper =====

Latency-sensitive and throughput-sensitive are two important classes of distributed computing jobs. It is nice that BPF is able to simultaneously optimize for both of them.

===== Issues that could prevent acceptance =====

Distributed cluster management has received lots of attention in recent years, with the number of proposed schedulers continually growing over time. From this perspective, BPF is just another scheduler with a different performance goal. While the solution looks reasonable, the technical contribution is a bit incremental.

BFP needs accurate estimates of resource demands and their duration to schedule jobs properly. This looks like a very strong assumption to me. Given the complexity of distributed computing jobs, I am not even sure whether obtaining such accurate estimates is generally feasible. Even if it is feasible, the extra complexity may well outweigh the potential benefits. (Given all three reviewers consider demand estimation as a key challenge, we should spend more space/efforts addressing it at least to some extent.)

You should compare BFP to "SP+DRF", e.g., latency-sensitive jobs get high priority and throughput-sensitive jobs get low priority, the job scheduling within the same priority uses DRF. To guard against latency-sensitive jobs that become too large (e.g., exceeding certain pre-defined threshold), they can be converted into throughput-sensitive jobs. To me, this simple two-tier scheduler will likely achieve the same effect as BFP, and is much easier to implement. (Tan, can you tell us the difference between this proposal and what you have implemented?)

```
Summer Deadline for Sigmetrics 2018 Paper #2 Reviews and Comments
===========================================================================
Paper #30 BPF: Mitigating the Burstiness-Fairness Tradeoff in
Multi-Resource Clusters
```

Review #30A
==

Overall merit
----
2. Weak reject

Reviewer expertise
----
4. Expert

Paper summary

This paper considers job scheduling algorithms for cloud datacenters that consider both longer throughput-sensitive (TQ) and shorter latency-sensitive (LQ) jobs. The authors first point out that existing schedulers do not enforce fair resource sharing among these job types. To resolve these problems, they introduce BPF (bounded priority fairness), a scheduling algorithm that admits LQ jobs only when their resource demands are small enough to not interfere with TQ jobs' long-term fair resource allocation. The authors show that BPF has several desirable properties (e.g., guaranteeing long-term fairness) and build a YARN-based implementation of BPF. Using three different benchmark workloads, they show that BPF leads to better completion times for LQ jobs with a minimal loss in completion time for TQ jobs.

Areas for improvement

The authors should more clearly explain the scenario under consideration. For instance, it was not immediately clear to me what constituted a "queue" (a stream of jobs from the same user? A set of jobs with some similar characteristics?). I also did not realize until the optimization formulation in Section 4.2 that both queue admission and resource allocation were being considered, and even then, it was initially unclear if the admission was for a queue or for an individual job in a given queue. Moreover, some variable relationships that don't seem fully articulated. For instance, in (1), the authors use R_i(n) to denote the LQ jobs' completion times; how does R_i(n) relate to the allocated resources a_i^k(t) and e_i^k(t)? (Presentation clarity)

Comments for author

--In the introduction, the authors say that one of the things that makes cloud scheduling harder than network scheduling is the multi-resource aspect. This is true to some extent (though one could view individual links in a network as multiple resources). However, it was unclear to me how the multi-resource aspect complicated the BPF

framework—wouldn't the problem be just as hard to solve if there were only a single resource? I can appreciate that having multiple resources might complicate the safety condition (2), but it's not in fact clear how multiple resources are factored into (2)—is the safety condition simply checked for each resource? (Good point! Is there any challenge besides we need to have DRF as a component?)

-- I did not understand what was meant by "on" and "off" periods—the authors seem to suggest in defining R_i(n) that multiple jobs can be completed during a single "on" period; does that mean that multiple jobs arrive at the same time to each queue, at which point resources are allocated to that queue? (Presentation clarity)

--In Figure 5, the authors should include error bars or standard deviations on the average completion times, to give a better sense of the overall performance. (Tan, how long will this take?)

--In Figure 1, it wasn't immediately clear why BPF was better than SP—I can see that BPF leads to shorter completion times for LQ A, but it seemed like BPF also starved TQ B while satisfying LQ A, which is exactly what the authors say is the problem with SP. If anything, SP seems to provide more proportional resource allocation than BPF in this example, though memory is less fully utilized.(Presentation clarity)

--The optimization problem (1) is stated a bit strangely. I understand that the authors have 3 objectives to fulfill, but it's not clear if these can all be realized at once (or if the proposed BPF algorithm does so)—in that case, it would be better to have a single objective in the optimization problem and then explain that this is a combination of multiple possible objectives. (This formulation does not work. Let's change it.)

--It is nice to see that BPF is strategy-proof, but there seems to be some nuance here that is missing from the discussion. The authors correctly point out that users may not know their job requirements in advance, meaning that even if BPF is strategy-proof, they might inadvertently lie about their jobs' requirements. The authors suggest that users should just submit their estimation of future demands, it's not clear which estimate they should submit—the average estimate? The maximum likelihood estimate? The worst-case scenario? It would be nice to see a discussion of these points. (Again, estimation…)

--Relating to one of my points above about what constitutes a queue, can a queue contain jobs of different sizes? It doesn't appear so, but would be good to make this explicit. (Presentation clarity: system -> queue -> job -> task)

--The authors mention service level agreements with respect to burst guarantees in Section 2.2. What would these SLAs say? It would be good to explain them more. (Presentation clarity)

--In (2), the authors seem to imply that the capacity is a function of the interarrival times—do they mean that C denotes an instantaneous capacity, and this must be integrated over time in (2)? (Presentation clarity)

(Besides presentation clarity, multi-resource, optimization, and estimation are three main concerns)

Review #30B
==

Overall merit
----
2. Weak reject

Reviewer expertise
----
3. Knowledgeable

Paper summary

The paper deals with multi-resource fairness: given on-off traffic sources with specific resources and possibly deadlines demands, the paper designs an admission control policy which roughly aims at 1) meeting the deadlines (for the traffic sources with such a requirement) and 2) guaranteeing fairness (for traffic sources requiring only long-term throughput).

The proposed scheme claims to attain the key properties of DFR (a generalization of max-min to multi-resources), and in addition a "burst guarantee" property (i.e., guarantee deadlines to time-sensitive traffic).

The performance of the scheme is evaluated in both testbed and simulation environments; it is shown in particular that it (significantly) outperforms DFR with respect to the performance of time-sensitive traffic.

Areas for improvement

This is a good "systems paper" but the analytical part is very weak in my opinion.

Is this paper suitable for one-shot revision? (Optional)
----

1. No

Comments for author

There are multiple "issues" with the analytical part of the paper:

1. I very much doubt that the solution from Sec. 3.3 solves for the optimization problem from Eq. (1), especially in the case of dynamic scenarios and without dropping already admitted "queues"; if so then why state (1)?

2. I am not sure that the last condition from Eq. (1) does indeed provide the desired guarantees: in the case of a single TQ, many LQs, and a_i^k=e_j^k, I don't see how that condition only can guarantee the long-term protection for the TQ. (Presentation clarity, but I guess this reviewer actually means the number of queues can increase, so the resources allocated to a TQ may decrease.)

3. I am very suspicious about the fair share condition calculated by a "simple fair scheduler", i.e., d * t <= C/N. Consider a single resource with C=10, T_i(n)=n for all i, n; t_j(n)=1, one TQ with d1=1, and one LQ with d2=6. Then according to the conditions (2) and (3) the LQ is not even admitted. (If T_i(n)=n, then LQ is requesting more than 1/2 of the system resources on average, of course it should not be admitted. I didn't understand exactly why this is a concern. Perhaps this reviewer wants to say in this case, we should admit the LQ instead of the TQ, which can be solved by utility optimization. In this case, it makes sense as FCFS admission is not good at all.)

Review #30C
===

Overall merit
----
2. Weak reject

Reviewer expertise
----
3. Knowledgeable

Paper summary

Today's large-scale clusters run a variety of jobs each of which may have a different performance goal. For instance, one may wish to optimize average completion time for (long-running) batch processing jobs; for interactive jobs, tail latency or throughput per-server may be more important.

However, most existing cluster schedulers tend to achieve fairness --- sharing resources among all jobs according to some fairness measure (instantaneous/long-term) --- while ignoring the various classes of jobs.

This paper makes a case for incorporating individual performance goals of jobs while providing long-term fairness to long-running jobs. The paper proposes BPF, a scheduling mechanism that is shown to achieve long-term fairness for long-running jobs, performance for short-running jobs, Pareto efficiency and strategy proofness.

Areas for improvement

- The paper could use a clearer exposition of classes of jobs that fit into latency-sensitive jobs (e.g., are web services contained in this class; see more below)

- For the case of online job arrivals, any fair mechanism would not provide resource guarantees. The paper, however, claims providing resource guarantees (by not admitting some jobs). Since some jobs are not admitted (or are admitted with soft guarantees), Theorem 3.1 holds only for jobs admitted with hard guarantees. Further strengthening of this result would make the results significantly more interesting.

- Evaluation could be significantly improved. In the current shape, it is hard to understand all the tradeoffs involved in BPF (see detailed comments).

Is this paper suitable for one-shot revision? (Optional)
----
1. No

Comments for author

The tradeoff between fairness and performance, at least intuitively, is rather well understood in the cluster scheduling literature. For instance, see Introduction of "Altruistic Scheduling in Multi-Resource Clusters" paper from OSDI'16. However, it is true that existing fair cluster schedulers ignore various classes of jobs and their individual performance goals when operating at certain point in the fairness-performance tradeoff curve. Incorporating these individual goals into performance objectives seems like an interesting goal.

The above said, I think the paper would improve significantly with a little more work, both in terms of motivation and in terms of technical content. Here are a few suggestions:

Possible improvement in definitions: (Great suggestions! This review is the most valuable outcome from the sigmetrics submission!)

-- Latency-sensitive jobs: What are the types of jobs that we should call latency-sensitive? While the question may sound naive, it is perhaps an important one in the context of this paper. Consider web services (key-value stores, Facebook wall, Google search, etc.). These kind of services, in my opinion, are not ON/OFF type jobs considered in the paper and are yet very latency-sensitive; these services typically are always ON and in fact, require performance in terms of both tail latency and per-server throughput. Existing cluster schedulers would assign these jobs just as fairly as batch processing jobs (which they perhaps shouldn't due to these services being user-facing), but they do not match the characteristics of latency-sensitive jobs considered in the paper. So, what are the kind of ON/OFF jobs considered in the paper? Just ad-hoc queries? (Spark streaming, any others?)

-- Long-term fairness for online job arrivals: Fairness is such a subjective term, that I think it would be useful to define what one means by long-term fairness especially in the context of online job arrivals. The one defined in Section 3.1 seems to be applicable for offline case only. Note that DRF definition (instantaneous fairness) is well-defined in online case. Consider a simple case of all jobs being of type TQ only. If job 0 and 1 arrive at time 0, job 2 arrives at time 1 and job 3 arrives at time 2, job 4 arrives at time 3, and so on (suppose each job is of length 10 units) ... how would one define long-term fair allocation? (This is exactly what we want to highlight. At the queue level, it makes sense to consider the long-term instead of instantaneous allocation. Moreover, we should highlight if instantaneous allocation such as DRF is used, it is impossible to accommodate both workloads.)

-- Burst guarantees: The case of burst guarantees is also unclear for online arrivals. Suppose TQ jobs are arriving at 1 hour inter-arrival time, with each TQ job being 1.5 hour long. And, LQ jobs are continuously arriving with inter-arrival time of 10 minutes and each LQ job is 15 min long. What would burst "guarantee" mean here? (We should define it.)

Fairness and guarantees:

-- Initially, I got fairly excited by the result of Theorem 1. However, during second round of reading (while reading proofs), I realized that the statement may be incomplete. The BPF policy guarantees the four properties only for LQ jobs admitted in H category (hard guarantees). Since BPF in itself admits LQ jobs in H or S based on a relatively simple heuristic, I am not entirely sure if BPF actually provides this guarantee. In fact, the proof in Appendix implicitly admits this --- "For LQs in S, they have resource guarantees *whenever possible* ....". (We should highlight that it is impossible to provide guarantee to all queues if their total demand is too high. Therefore admission control is necessary)

-- What happens in BPF if LQs keep arriving? Eg, suppose each LQ runs for 1 time unit, and LQs have inter-arrival time of 1 time unit (these are different LQs). What will happen to TQs? (The message of isolation guarantee *by* admission control does not pass…)

Evaluation and related work:

-- The discussion on weights in DRF was fairly unconvincing (Section 2.1). Why can't wait assignments for LQs be proportional to their expected run time? This would provide similar guarantees as BPF, at least intuitively. Nevertheless, this requires proper evaluation and/or careful exposition with various weight assignment techniques. (Tan, could you please let us know the difference between this proposal and your implementation?)

-- I think the evaluation could really use some more work. In its current form, it seems very contrived with carefully chosen parameters that make BPF look good. For instance, it would be nice to perform sensitivity analysis against inter-arrival times and running times of LQs. It would also be nice to perform sensitivity analysis against ratio of #LQ jobs and #TQ jobs with fixed inter-arrival times and running times. (Yes, sensitivity analysis could help. Tan, can you come up with a list of figures for sensitivity analysis as we did before, e.g., axis, baseline, etc?)

-- In terms of evaluation, choosing inter-arrival time 10x larger than running times for LQ jobs is precisely a good scenario for BPF. It is unclear what the results would look like if this number was much smaller. (Again, sensitivity analysis could help.)

-- In terms of metrics, the earlier part of the paper seems to suggest that "average job completion time" is not a good metric for LQs. Why is it that the paper decided to evaluate average job completion time even for LQs? Why not compare the performance for LQs against the best possible completion time (lets say, oracle time when LQ jobs have the entire cluster to themselves, and plot a CDF for each of the schedulers)? (We need to clarify between jobs and tasks.)

Overall, I think this is a neat direction and could lead to an interesting research direction. I am excited about reading next version of the paper.