ServerFilling: A better approach to packing multiserver jobs

(invited paper)

Isaac Grosof[*]
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
igrosof@cs.cmu.edu

Mor Harchol-Balter[†]
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
harchol@cs.cmu.edu

ABSTRACT

Ever since the advent of "multiserver jobs" (jobs that require more than one server or core simultaneously), practitioners have been faced with the question of how to pack these jobs into a compute cluster. While many policies have been proposed, including First-Come-First-Served (FCFS), Back-Filling (BF), MaxWeight, and Most Servers First, it is not well understood which policies simultaneously achieve (1) throughput-optimality and also (2) both low and theoretically predictable mean queueing times.

This paper reviews some very recent work from [8, 9] on an alternative packing policy called ServerFilling (SF) and some extensions of this policy. The SF policy achieves both goals (1) and (2) above. This paper discusses and evaluates existing policies in comparison to SF, in order to prompt discussion on the tradeoffs between different scheduling policies.

ACM Reference format:
Isaac Grosof and Mor Harchol-Balter. 2023. ServerFilling: A better approach to packing multiserver jobs. In *Proceedings of Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems, Orlando, Florida, USA, June 2023 (ApPLIED '23),* 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MULTISERVER JOBS

Most computing centers today, be they a public cloud or a private server farm, run *multiserver jobs*. A multiserver job requests some number of servers, typically more than one, and holds onto those servers for some amount of time. In this paper, the term "server" is an abstraction; a server might refer to a CPU, a GPU, or some other processor. A multiserver job has two components: (i) its *server need*, which is the number of servers requested by the job, and (ii) its *duration*, which is the time that the job will hold onto those servers. The *size* of the multiserver job is the product of its server need and its duration, and is expressed in units of server-hours. Importantly, the server need can vary across jobs, often differing by orders of magnitude [13, 14]. While a job's exact server need is known by the system, the job's duration may not be known at all, or sometimes only an estimate or upper bound is known.

Figure 1 illustrates what we will refer to as the **multiserver job queueing model**. Here there are a total of k servers. Jobs arrive with average rate λ. With probability p_i an arrival is of class i. An arriving job of class i requests n_i servers and holds onto these servers for X_i time, where X_i is a random variable. In Figure 1, the scheduling policy is FCFS. However, in general any scheduling policy can be used, and we describe several in this paper.

2 FCFS SCHEDULING AND ITS DRAWBACKS

The most common scheduling policy for multiserver jobs is FCFS; see for example the CloudSim, iFogSim, EPSim and GridSim cloud computing simulators [10], or the Google Borg Scheduler [13]. Unfortunately, FCFS scheduling results in servers being left idle, as seen in Figure 1. This happens when the job at the head of the queue does not "fit" into the available servers (the job's server need is greater than the number of available servers) so servers are left idle until the job can fit. This results in wasted serves. Under FCFS, depending on the particular server needs, half of all servers might be wasted [6]. A second drawback of FCFS is that it is currently *unknown* how to estimate the mean waiting time of multiserver jobs under FCFS beyond the case of $k = 2$ servers [2, 5].

3 FORMALIZING THE MODEL

We define the **load** ρ of a system to be the time-average fraction of total server capacity in use. In particular,

$$\rho = \frac{\lambda \sum_i n_i p_i E[X_i]}{k}, \qquad 0 < \rho < 1 \tag{1}$$

Note that the numerator of (1) can be viewed as the rate of work arriving to the system, while the denominator is

[*]Supported by NSF-CMMI-1938909 and NSF-CSR-1763701, and by a Siebel Scholar Award 2022-2023.
[†]Supported by NSF-CMMI-1938909 and NSF-CSR-1763701.

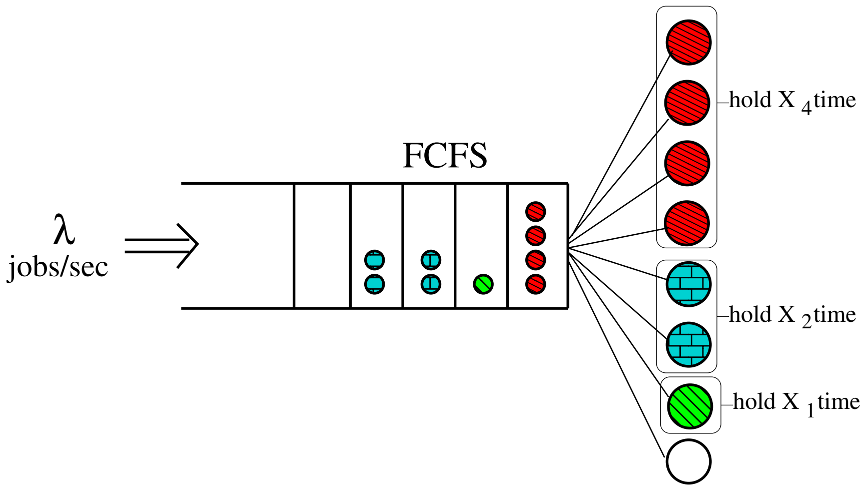

Figure 1: ***The multiserver job queueing model with $k = 8$ servers and FCFS scheduling policy. An arriving job of class i requests n_i servers and holds onto these servers for X_i time. In this particular illustration, $n_i = i$.***

the maximum possible rate of work completion. Note that if the rate of work arriving exceeds k, instead jobs back up indefinitely, and formula (1) no longer holds. This behavior is called "instability", and we focus in this paper on stable settings only. Observe that loads ρ approaching 1 are reached only when the system is always working at capacity, i.e. all servers are occupied.

Importantly, for some scheduling policies π and workloads w, loads ρ near 1 are unachievable: For λ high enough to bring ρ close to 1, the system is already unstable.

For example, consider the setting where the number of servers $k = 2$ and consider a workload w in which half of the jobs require 1 server and half require 2 servers, and where $E[X_1] = E[X_2] = 1$. Here loads ρ near 1 are not achievable by the FCFS policy. The resulting system is unstable. However, loads ρ arbitrarily close to 1 are achievable by the Most Servers First (MSF) policy, which prioritizes 2-server jobs over 1-server jobs. FCFS wastes servers unnecessarily, while MSF keeps all servers busy whenever possible.

As another example, consider a setting with $k = 4$ and a workload w' in which all jobs require 3 servers. In this setting, no scheduling policy π can achieve loads ρ near 1. In particular, no scheduling policy can achieve a load above $3/4$.

We say that a policy π is **throughput-optimal** for a particular workload w if π can achieve the highest ρ of any scheduling policy working on workload w.

4 GOALS

Ideally a scheduling policy should have these two properties:

(1) The policy should be throughput-optimal for commonly encountered workloads w.
(2) It should be possible to theoretically analyze the expected waiting time under the policy. This is important when load balancing across different clusters. Specifically, it allows one to balance the load to ensure comparable mean waiting time at each cluster.

Ideally, we also want mean waiting time to be low.

5 OTHER SCHEDULING POLICIES IN THE LITERATURE

EASY BackFilling: To mitigate the idle servers which result from FCFS scheduling, BackFilling is sometimes used [3, 4, 12]. Two of the most common versions of BackFilling are called Conservative and EASY BackFilling. Under EASY BackFilling, if the job j at the head of the queue has server need is larger than the number of available servers, the system tries to estimate a time t at which point enough servers will become available for j to run. The system then allows jobs that arrived after job j to run if their server needs allow them to fit and if they will complete before time t. Conservative BackFilling is similar, but performs this reservation process for all jobs in the queue, not just the job at the head of the queue. There are several problems with these versions of BackFilling. First, they are hard to implement, in that they require knowing the durations of the jobs. Next, although BackFilling is better than FCFS at avoiding idle servers, it is not understood under which workloads each of these BackFilling policies is throughput-optimal. Finally, just as there is no queueing analysis of waiting time under FCFS, it is even harder to imagine analyzing waiting time under these BackFilling policies.

FirstFit BackFilling: There are also alternative versions of BackFilling that do not incorporate duration information, such as FirstFit BackFilling. These policies allow the backfilling process to look arbitrarily far back in the queue and consider jobs with arbitrarily long duration. As a consequence, these policies may excessively delay jobs with large server need, leading to poor and unpredictable waiting times. For this reason, these policies are rarely used in practice.

Most Servers First (MSF): In this policy, one prioritizes jobs based on server need, at all times preemptively running the jobs with highest server need. This policy also goes by the name "BestFit", which can either refer to the preemptive or nonpreemptive version of the policy [1, 11]

The preemptive version of MSF has the advantage of being throughput-optimal for many common workloads w because it packs jobs into servers well. Unfortunately, in order to pack well, MSF emphasizes jobs with large server need, which often results in high mean waiting time. Moreover, there is no known analysis of mean waiting time under MSF. Finally, preemptive MSF has the disadvantage of performing many preemptions.

In contrast, the nonpreemptive version of MSF is not throughput-optimal for almost any workload. Moreover, the problem of high waiting time and no waiting time analysis both still exist. Nonpreemptive MSF does have the advantage of not requiring preemption.

MaxWeight: The MaxWeight policy is designed to be throughput-optimal for all workloads. It is a preemptive policy which at all times t searches over all possible packings to pick a packing z which maximizes

$$\max_z \sum_i N_i(t) z_i,$$

where $N_i(t)$ is the number of jobs in the system with server need i and z_i is the number of jobs with server need i that are served by packing z. While MaxWeight is provably throughput-optimal [11], it is prohibitively complex to implement, and there is no analysis of its mean waiting time. Empirically, the waiting time can be high under moderate load (when queue lengths are short) because in this regime MaxWeight has only limited information to make its decisions.

6 A NEW IDEA: SERVERFILLING

In this paper we summarize a very new approach to scheduling multiserver jobs which achieves both of our goals [8]. The ServerFilling policy operates under the regime where the total number of servers, k, is a power of two, and the server need of each job, is also a power of two. The power of two setting is common for computing jobs. ServerFilling can also handle certain other workloads, such as the case where all jobs require either 1 or k servers. Additionally, there is a variant of ServerFilling, called DivisorFilling (see [7, Appendix] or [9, Appendix]), which allows for the more general case where all job server needs are divisors of k.

ServerFilling is actually not that different than FCFS scheduling. Jobs join a queue in FCFS order. ServerFilling only serves jobs that are near the head of the queue. ServerFilling designates a "candidate set", M, which consists of the minimal prefix of jobs in arrival order which collectively require $\geq k$ servers. Notice that $|M| \leq k$ because all server needs are at least 1. Once the set M has been determined, the jobs within M are ordered by their server needs, the n_i's, from largest to smallest, tie-broken by arrival order. Jobs from M are then placed into service in order of largest server need first. Note that because M is small and consists only of the oldest jobs in the system in arrival order, ServerFilling serves jobs in near-FCFS order.

While preemption is needed under ServerFilling, it is not frequent. When jobs complete, M changes, and the set of jobs in service is recomputed, which can lead to a preemption. By contrast, when jobs arrive, they do not change M unless M was previously not full (total server need $< k$).

The property that makes ServerFilling so powerful is that, whenever the set M is full (total server need $\geq k$), ServerFilling will result in all k servers being full. Thus, the ServerFilling policy has the property that whenever it is possible to utilize all servers, the policy does so. Hence, the ServerFilling policy is able to achieve throughput-optimality, or equivalently $\rho \to 1$.

It is further shown in [8] that the ServerFilling policy has very predictable performance. Specifically, let W_{SF} denote the *waiting time* under ServerFilling, namely the entire amount of time from when a job arrives until it completes service during which the job is waiting in the queue. Then the mean waiting time under ServerFilling is bounded as follows:

$$\frac{\rho}{1-\rho}\frac{\mathbf{E}\left[S^2\right]}{2k\mathbf{E}\left[S\right]} + c_{\text{low}} \leq \mathbf{E}\left[W_{\text{SF}}\right] \leq \frac{\rho}{1-\rho}\frac{\mathbf{E}\left[S^2\right]}{2k\mathbf{E}\left[S\right]} + c_{\text{up}}, \quad (2)$$

where S represents the job size (the product of the job's server need and its duration) and ρ represents the system load, see (1). In (2), c_{low} and c_{up} are constants that do not grow with load. One can show from this bound that as $\rho \to 1$, the ServerFilling system operates like a system where all k servers have been aggregated into a single powerful server and all jobs are run in FCFS order on that single powerful server with zero waste. While the bound in (2) is stated for overall mean waiting time, one can use the same approach as in [8] to prove a bound on the mean waiting time of each class of jobs.

There are several possible extensions of the ServerFilling idea, most notably the ServerFilling-SRPT (SF-SRPT) policy from [9]. SF-SRPT is less practical because it requires much more preemption of jobs, but we describe it here because it results in asymptotically optimal mean waiting time for multiserver jobs as $\rho \to 1$. SF-SRPT is similar to ServerFilling, except that the jobs are ordered in terms of their Shortest-Remaining-Processing-Time (SRPT), where a job's

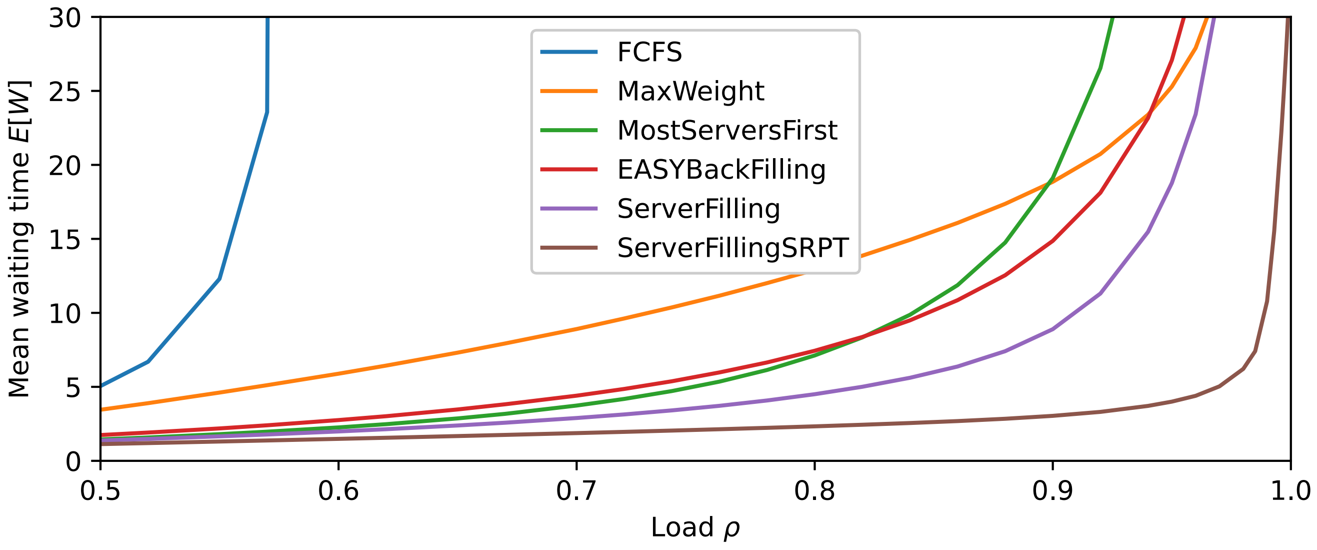

Figure 2: Mean waiting time in a system with $k = 8$ servers, and a workload consisting of server need 1 with probability $2/3$, otherwise server need 8. Duration is distributed $Exp(1)$ independent of server need. We simulate 10^6 arrivals and loads up to $\rho = 0.999$. EASY BackFilling and ServerFilling-SRPT make use of exact size information.

processing time is its size, which is the product of its server need and duration. One now defines the candidate set M in the same way based on that ordering, and again runs jobs in set M in order of largest server need first. New arrivals cause the M set to change, which can result in a job preemption.

7 COMPARISON OF POLICIES

Figure 2 shows a comparison of the different scheduling policies that we have discussed. The setting is $k = 8$ servers, and the workload consists of jobs of server needs 1 and 8. While $2/3$ of jobs require 1 server, $1/3$ require 8 servers. Service duration is $Exp(1)$, independent of server need.

Unsurprisingly, FCFS has by far the worst performance, becoming unstable around $\rho = 0.6$. The EASY BackFilling policy, on the other hand, performs much better than FCFS, due to its better packing. However, EASY BackFilling requires knowing the durations of jobs, and we have optimistically assumed perfect knowledge of this duration. ServerFilling (SF) outperforms EASY BackFilling and does not require knowledge of job durations. The difference between SF and EASY BackFilling can be explained with the following observation: EASY BackFilling often serves just a few jobs with server need 1 when a job with server need 8 is at the head of the queue, leaving the other servers idle; this waste can never happen under SF.

The algorithm that sounds the closest to SF is Most Servers First (MSF), because it too minimizes idle servers by favoring the jobs with most server need. However, SF only consider jobs among the M set of oldest jobs, while MSF considers all jobs. Consequently, the bias in MSF is much more extreme, causing small *sized* jobs to have to wait behind too much work of large sized jobs, resulting in high mean waiting times.

Recall that MaxWeight is provably throughput-optimal. However, as shown in Figure 2, its performance only approaches the good performance of SF at high loads ($\rho \geq 0.95$). For moderate loads, MaxWeight simply does not have enough jobs in the queue to make sound scheduling decisions.

Finally, ServerFilling-SRPT (SF-SRPT) has extremely good mean waiting time at all loads. This is unsurprising, since it both benefits from the good packing properties of SF, while also prioritizing small sized jobs, which is known to improve mean response time. However, SF-SRPT requires exact job size information and preemption when new jobs arrive.

8 CONCLUSION

Our purpose in this paper was to compare scheduling policies with respect to our two goals of (1) throughput-optimality and (2) predictable and low mean waiting times. After examining a wide variety of scheduling policies, the ServerFilling (SF) and ServerFilling-SRPT (SF-SRPT) policies stand out in their ability to meet these goals. On the other hand, SF requires some minimal preemption, while SF-SRPT requires frequent preemption. This leads one to the open question of whether it is possible to achieve all the benefits of SF with even less preemption.

REFERENCES

[1] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 577–578. IEEE, 2010.

[2] Percy H. Brill and Linda Green. Queues in which customers receive simultaneous service from a random number of servers: A system point approach. *Management Science*, 30(1):51 – 68, January 1984.

[3] Danilo Carastan-Santos, Raphael Y. De Camargo, Denis Trystram, and Salah Zrigui. One can only gain by replacing EASY backfilling: A simple scheduling policies case study. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 1–10, 2019.

[4] Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Parallel job scheduling—a status report. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–16. Springer, 2004.

[5] D. Filippopoulos and H. Karatza. An M/M/2 parallel system model with pure space sharing among rigid jobs. *Mathematical and Computer Modelling*, 45(5):491–530, 2007.

[6] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. Stability for two-class multiserver-job systems. arXiv:2010.00631, October 2020.

[7] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. WCFS: A new framework for analyzing multiserver systems. *arXiv preprint arXiv:2109.12663*, 2021. Electronic companion (full version) of the paper by the same name in Queueing Systems.

[8] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. WCFS: a new framework for analyzing multiserver systems. *Queueing Systems: Theory and Applications*, 102:143–174, June 2022.

[9] Isaac Grosof, Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. Optimal scheduling in the multiserver-job model under heavy traffic. *Proceedings of ACM on Measurement and Analysis of Computer Systems (POMACS/SIGMETRICS)*, 6(3):1–32, December 2022.

[10] Syed Hamid Hussain Madni, Muhammad Shafie Abd Latiff, Mohammed Abdullahi, Shafi Muhammad Abdulhamid, and Mohammed Joda Usman. Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. *PLOS ONE*, 12(5):1–26, 05 2017.

[11] Siva Theja Maguluri, R. Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proceedings of IEEE INFOCOM*, pages 702–710, 2012.

[12] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings. International Conference on Parallel Processing Workshop*, pages 514–519, 2002.

[13] Muhammad Tirmazi, Adam Barker, Nan Deng, MD E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*, pages 1–14, Greece, April 2020.

[14] John Wilkes. Google cluster-usage traces v3, November 2019. http://github.com/google/cluster-data.