A Delicate Union of Batching and Parallelization Models in Distributed Computing and Communication

Sounak Kar
Technical University of Darmstadt
sounak.kar@kom.tu-darmstadt.de

Amr Rizk
Universität Ulm
amr.rizk@uni-ulm.de

***Abstract*—The Fork-Join (FJ) model has been extensively studied in the past due to its natural ability to capture computation and communication systems that employ split and merge techniques, i.e., boosting performance through splitting and parallelizing the input and finally merging the results. This model finds applications ranging from multipath communications to distributed databases.**

In this work, we explore the effect of batching within FJ systems, i.e., when servers collect the input to benefit from a so-called speedup, observed as a service time reduction due to batching. We numerically compare the performance when the speedup assumes different analytical forms. Our numerical evaluation shows that the steady-state waiting time of such an FJ system is heavily dependent upon the form of the speedup achieved through batching while the optimization thereof is non-trivial.

I. Introduction

With the advent of multipath communication and large-scale data processing, there has been a rising interest in modelling such systems in order to provide performance guarantees. A large class of parallel systems employs split-and-merge techniques to enhance performance where incoming *jobs* are split into *tasks* which are served by the parallel servers and the served tasks are merged in the end to complete the process. While such systems benefit from parallelization, the gain from it is not unmitigated, owing to the output synchronization constraint these systems pose. FJ models have been in the forefront among the modelling frameworks due to its natural ability to account for the dynamics of this class of parallel systems [1] [2]. One of classical examples of a classical system under the split-and-merge paradigm is MapReduce [3] and its implementations, e.g., Hadoop and Spark [4], are indispensable part of the modern IT world.

Service batching has long been known as a key technique to cutback overhead, especially in communication and computing systems. This involves accumulating incoming *requests/packets* to form a *batch* which is served as a single entity. Due to overhead amortization, service batching has been found to have favourable impact on the system throughput and a varied impact on the delay of the incoming requests, strongly influenced by the underlying dynamics of the system and the size of the batch. Prominent examples where service batching facilitated enhanced throughput are: data-packet batching in Linux-based systems [5] and query-batching in databases [6].

In this work, we seek to model an FJ system which employs batching of *tasks* to enhance the performance further. To this end, we ascribe certain analytical forms to the gain obtained from service batching, termed as speedup in the following, and analyze the resultant waiting time. Such technique might prove to be useful in distributed databases where after splitting a complex query into smaller queries for constituent servers, smaller tasks can be batched to enhance performance further. We simulate an FJ system with high utilization and observe the pattern of the steady state waiting time as the batch size grows and the speedup form and the size of the system is varied. While the formal model is given in Sect. II, the numerical results of the experiments are presented in Sect. III. We also discuss the related work in Sect. IV and summarize our findings in Sect. V.

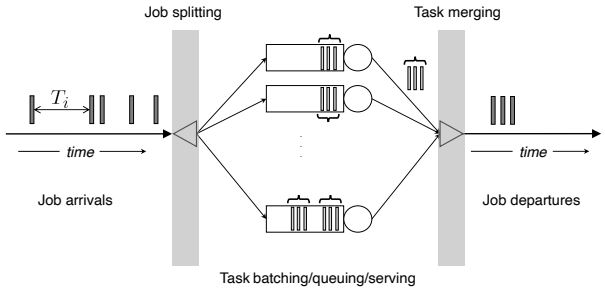


Fig. 1: Sketch of an FJ system with task-batching: Incoming *jobs* (dark grey units) arrive according to a renewal process and upon arrival are split into N *tasks* of equal size (light grey atoms). In each server, k *tasks* are batched and served together. Batches of tasks are finally merged and jobs with tasks from the same batch leave together.

II. Batching in a Fork-Join System

We schematically depict the general FJ model of a parallelized communication / computation system in Fig. 1. In such a system, *jobs* arrive to the splitting station according to a certain arrival process. These jobs may resemble data

This work has been funded by the German Research Foundation (DFG) as part of project B4 within the Collaborative Research Center (CRC) 1053 – MAKI. Computational facilities provided by the Lichtenberg - High Performance Computer at TU Darmstadt are gratefully acknowledged.

chunks that need to be transported over different paths or computations that are spilt and mapped to multiple servers for execution. The jobs are split into tasks and sent to the parallel servers for processing. The servers extend service according to independent service processes and once processed, the completed tasks are finally merged in the merging station and the job is said to leave the system. In this work, we restrict our attention to the case where incoming jobs have unit size and are equally split among the homogeneous servers. Note that equal split, which happens to be proportional split for the case with homogeneous servers, might be favoured over adaptive allocation due to its ease of implementation.

With the description above, we aim to evaluate the performance of FJ systems when the tasks are batched in each server after being split. Batching is a technique that is often employed to boost performance, e.g., throughput, and can be found in many applications ranging from packet transmission and reception on network cards to database query executions. This is because service batching is seen to reduce the total service time. This phenomenon is loosely termed as service speedup. In the following, we assume that the batch size is kept fixed over time and across servers, motivated by the fact that keeping batch size fixed makes it easier to implement in real-world systems. It is also seen to be beneficial in other type of batch-processing systems such as databases where cached batch queries might be utilized for fixed batch sizes.

Let us denote the service time of a batch consisting of k tasks of unit size as S_k. Now, without loss of generality, we can assume $\mathbf{E}(S_1) = 1$ and scale other relevant variables for the system. For the sake of brevity, we express the speedup through the function $g : \mathbb{N} \mapsto \mathbb{R}_+$ where $g(k) = \mathbf{E}(S_k)$. Formally put, the speedup is equivalent to sub-additivity, i.e., $g(k_1 + k_2) \leq g(k_1) + g(k_2)$. In this work, we consider the following forms of speedup function:

- $g_1(k) = ak + b$ with $0 < a < 1$ and $a + b = 1$,
- $g_2(k) = 1 - \log(c + 1) + \log(c + k)$ with $c > 0$,
- $g_3(k) = k^\alpha$ with $\alpha < 1$.

Note that the parameters are chosen appropriately so that the subadditivity criterion is met. We investigate the effect of batching in terms of throughput of the system and the waiting time of jobs in the steady state which, we assume, is attained.

Let us further assume that the jobs arrive according to a renewal process, i.e., the interarrival times T_i's are i.i.d. with $P(T_i > 0) = 1 \quad \forall i$. Let us denote this common distribution by $F(\lambda)$, while λ being the parameter of the distribution. As stated earlier, jobs of unit size are split into N tasks of size $1/N$, where N denotes the number of available servers/paths. If we denote the service time at the nth server for the jth batch consisting of k such tasks by $S^k_{j,n}$, we have $S^k_{j,n} \sim G(\mu_k/N)$, $n \in [N]$. Here, $[N] = \{1, 2, \ldots, N\}$ and $\mathbf{E}(S^k_{j,n}) = g(k)/N = \mathbf{E}(S_k)/N$ and G denotes the common service distribution which satisfies $G(\mu/x) \stackrel{D}{=} G(\mu)/x$. Note that the families $\{T_i\}$ and $\{S^k_{j,n}\}$ are assumed to be independent. Further, we assume the splitting and batching process to be instantaneous.

To account for the batching of tasks, we can formulate an equivalent system where the every kth arrival is considered an arrival of a job of size k and the service process is described by the same family $\{S^k_{j,n}\}$. Let us assume the interarrival times $\{B_i\}$ of the equivalent system are i.i.d. with distribution $H(k, \lambda)$, i.e., $H(k, \lambda) \stackrel{D}{=} \sum_{i=1}^{k} F_i$, with $F_i \sim F(\lambda)$. Let us denote by U_l the waiting time of the lth job in the equivalent system. If we take $U_1 = 0$, i.e. we start the system with first incoming job, we have,

$$U_l = \max\big\{0, \max_{1\leq m\leq l-1}\big\{\max_{n\in[N]}\big\{\sum_{i=1}^{m} S^k_{l-i,n} - \sum_{i=1}^{m} B_{l-i}\big\}\big\}\big\}, \tag{1}$$

where $\sum_{i=1}^{0} B_i$ equals 0 by convention. Further, if we denote the waiting time of the lth job in the original system as W_l, we have $U_l = W_l$ whenever l is a multiple of k. Otherwise,

$$W_l = U_{\lfloor l/k \rfloor} + \sum_{i=\lfloor l/k \rfloor k+1}^{l} T_i. \tag{2}$$

If we average over uniformly chosen values of $mod(l, k)$, the average waiting time is given by

$$\bar{W}_\iota = U_\iota + \frac{1}{k}\sum_{j=1}^{k}\sum_{i=1}^{j-1} T_i, \tag{3}$$

where $\iota = \lfloor l/k \rfloor$ and $\sum_{i=1}^{0} T_i$ equals 0 by convention. Note that the usual interpretation of $1/k \sum_{j=1}^{k}\sum_{i=1}^{j-1} T_i$ is batch formation time whereas U_ι is the waiting time after batch formation. In line with [7], the steady state waiting times of the equivalent system can be written as

$$U \stackrel{D}{=} \max_{m\geq 0}\big\{\max_{n\in[N]}\big\{\sum_{i=1}^{m} S^k_{i,n} - \sum_{i=1}^{m} B_i\big\}\big\}.$$

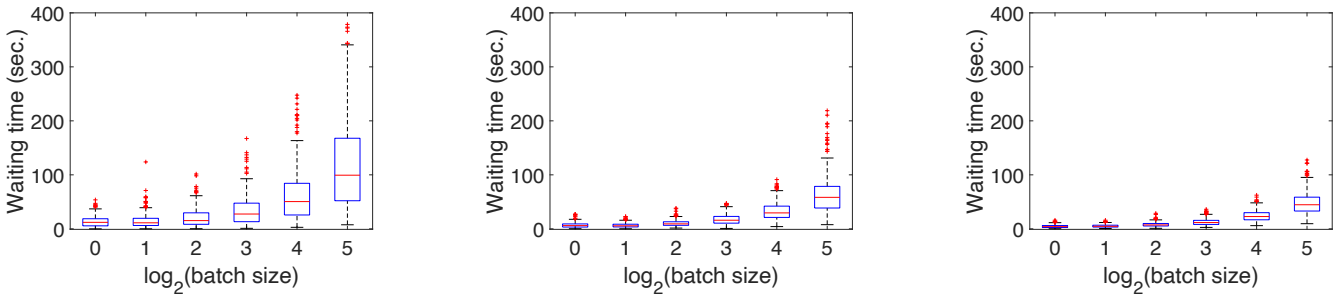
Hence, following (3), the average steady-state waiting time for the original system can be written as

$$\bar{W} \stackrel{D}{=} U + \frac{(k-1)}{2} T_1.$$

Observe that the second component of $\bar{W}$ increases linearly in expectation with batch size k whereas the relation of U with k depends on the speedup and the distribution of interarrival and service times as well. For a strong enough speedup, the reduction in $\mathbf{E}(U)$ can even offset the increment in $\mathbf{E}((k-1)T_1/2)$, at least for a certain range of the batch size k.

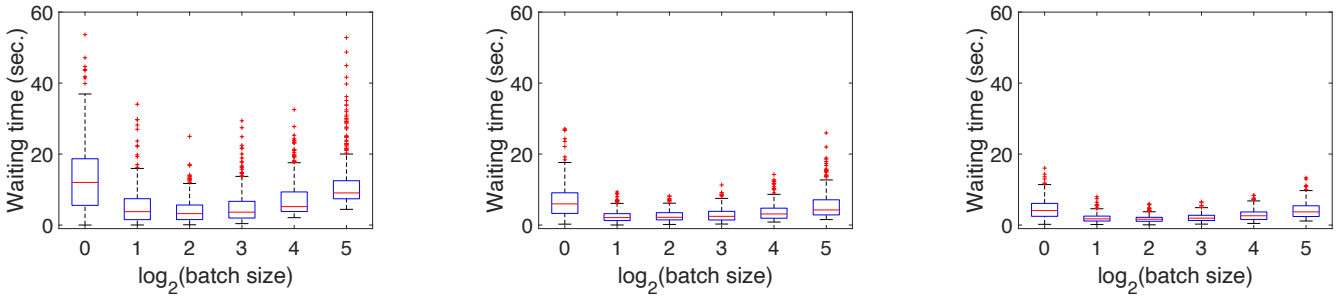
III. Evaluation

In this section, we first describe our evaluation setup and subsequently present our observations on the numerical performance of the system under different forms of batching speedup. Although our framework is valid for any interarrival and service distribution, for simulating the FJ system, we assume the families $\{T_i\}$ and $\{S^k_{j,n}\}$ are exponentially distributed. Note that, since F is $Exp(\lambda)$, $H \sim \Gamma(k, \lambda)$, see Chap. 6 of [8] for details. Further, the parameter λ is suitably chosen to have $0.95N$ arrivals/second, indicating high arrival



(a) $\bar{W}_\iota$ for a system with 2 servers. (b) $\bar{W}_\iota$ for a system with 6 servers. (c) $\bar{W}_\iota$ for a system with 10 servers.

Fig. 2: Average steady state waiting time for a job in an FJ system when the service speedup assumes linear form: The waiting time gets longer as the batch size increases. The rate of increase diminishes with an increasing number of constituent servers.



(a) $\bar{W}_\iota$ for a system with 2 servers. (b) $\bar{W}_\iota$ for a system with 6 servers. (c) $\bar{W}_\iota$ for a system with 10 servers.

Fig. 3: Average steady state waiting time for a job under power form of service speedup: On average, the waiting time reaches a minimum and increases subsequently, although the rate of increase flattens as the number of constituent servers grows.

regime, N being the number of servers in the system. Servers are assumed to be homogeneous and the service time of a *job* is assumed to be $Exp(1)$ distributed. Note that jobs are split into tasks of equal size and for a system with N servers, *task* service times are distributed as $Exp(1)/N$.

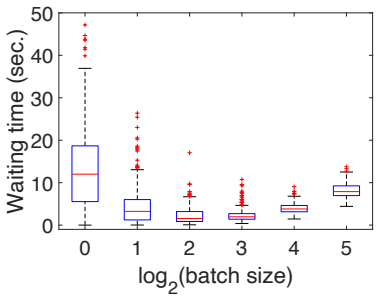
For our evaluation, we take $N = 2, 6, 10$ to illustrate the effect of batching on performance as the number of servers grows. As described in Sect. II, we run experiments separately for three forms of speedup, namely linear, logarithmic and power. The exact forms for these speedups used for the first set of experiments are:

- $g_1(k) = 0.95k + 0.05$,
- $g_2(k) = \log k + 1$,
- $g_3(k) = k^{0.8}$,

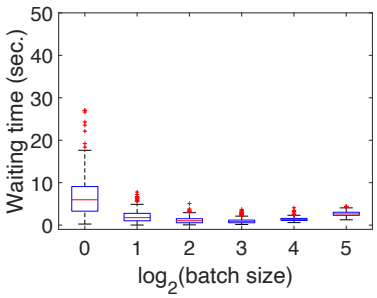
respectively. We simulate the job arrivals, splitting of jobs, instantaneous batching in each server given appropriate amount of available tasks, task service and finally instantaneous merging of tasks. We assume the steady state is attained after 10^5 arrivals and look at the throughput of the system and the average job waiting time $\bar{W}_\iota$ as defined in (3). We also compute the component expressing the waiting time after batch formation denoted as U_ι in (3). Simulation runs are repeated 500 times to generate the distribution of the waiting time. The batch sizes for each run are given by $\{1, 2, 4, 8, 16, 32\}$.

We observe that the average steady state throughput equals the arrival rate and that the dispersion around the mean decreases fast as the batch size is increased, irrespective of the form of service speedup and the number of parallel servers. The waiting time $\bar{W}_\iota$, however, shows a different pattern depending upon the the form of the speedup and the number of servers and, hence, is more interesting as a performance metric. In Fig. 2, we see that, for linear form of speedup, the waiting time increases as the batch size grows. This is because both the batch formation time and the waiting time of the batch grow with increasing batch size under linear speedup. This, however, changes when the speedup assumes logarithmic or power form. Note that the average batch formation time is the same for all cases and grows linearly in batch size irrespective of the form of the speedup. The batch waiting time U_ι, however, decreases for both power and logarithmic speedup and even overcompensates until a threshold batch size depending on the experiment parameters.

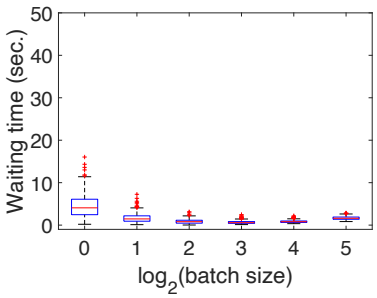
In Fig. 3 and Fig. 4, we see that the rate at which the waiting time initially changes or eventually increases with batch size is naturally dependent upon the number of constituent servers. Further, in Fig 3, the deviation of the waiting time around mean increases with batch size beyond the minima. We see a similar centrality pattern for the waiting time even for logarithmic speedup as seen in Fig. 4, although the dispersion around mean steadily decreases with increasing batch size.



(a) $\bar{W}_\iota$ for a system with 2 servers.

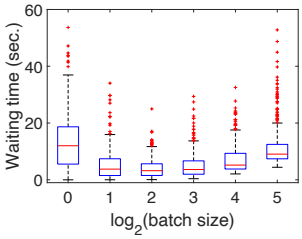


(b) $\bar{W}_\iota$ for a system with 6 servers.

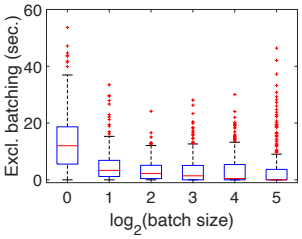


(c) $\bar{W}_\iota$ for a system with 10 servers.

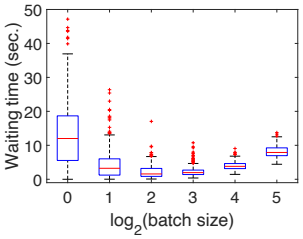
Fig. 4: Average steady state waiting time for a job under logarithmic service speedup also reaches a minimum and then increases on average as the batch size grows. Unlike the power speedup, the variance of waiting time goes down sharply.



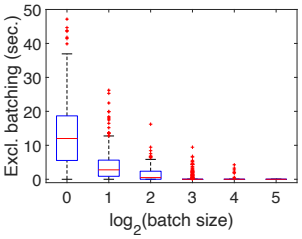
(a) $\bar{W}_\iota$ for a system with 2 servers under power speedup.



(b) U_ι for a system with 2 servers under power speedup.



(c) $\bar{W}_\iota$ for a system with 2 servers under logarithmic speedup.



(d) U_ι for a system with 2 servers under logarithmic speedup.

Fig. 5: The waiting time $\bar{W}_\iota$ obtains a minimum due to two offsetting factors: The increasing batch formation time with increased batch size and the reduced waiting time after batch formation U_ι due to service speedup. For logarithmic speedup, U_ι dies down to zero, leading to comparatively shorter waiting times and visibly lower variance.

This can be attributed to the fact that U_ι, the waiting time after batch formation, diminishes much faster in case of logarithmic speedup which happens to be major source of variance in the total waiting time $\bar{W}_\iota$. We depict this behaviour in Fig. 5 where both the total waiting time $\bar{W}_\iota$ and the batch waiting time U_ι are compared for logarithmic and power speedup. We see in Fig. 5d that the batch waiting time eventually dies down which is not the case for the power speedup as seen in Fig. 5b. Although the average waiting time gets shorter, it shows comparatively higher variance, resulting in higher variance of the total waiting time in Fig. 5a.

To understand the consequence of the speedup form, we run another set of experiments with a different parametrization given below:

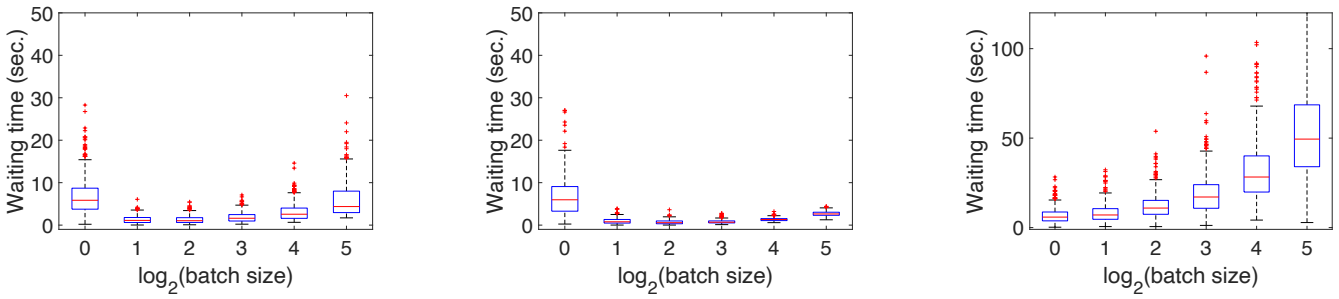
- $g_1(k) = 0.5k + 0.5$,
- $g_2(k) = 1 - \log(2) + \log(1 + k)$,
- $g_3(k) = k^{0.98}$.

The remaining setup is kept same as before. Our experiments show that while logarithmic speedup is strong enough to retain the similar pattern for the variation of the waiting time with respect to batch size, the variation is strongly dependent on the parametrization in case of linear and power speedup. E,g., this particular power speedup fails to produce an optimum batch size for waiting time while the linear speedup yields one.

IV. Related Work

Literature related to ours can be segregated into two different areas, namely *(i)* analysis of queueing systems with batching execution and *(ii)* the performance analysis and applications of FJ systems.

One of the earliest examples of batching in the queueing literature is [9] where the author derives the expected value of steady state queue length and waiting time assuming exponential inter-arrival and Chi-squared service time. In [10], the authors consider a queueing system with Poisson arrivals and general batch service time independent of the batch size. Certain forms of holding and serving cost are also assumed. Batching in the context of running a shuttle service between two end points where customers arrive according to independent Poisson processes has been investigated in [11]. The author provides an optimal batching policy for minimizing the expected total discounted cost over infinite horizon. In [12], the authors consider a system where the incoming jobs have a strict delay guarantee. Given a certain form of serving cost incentivizing batching and a particular distribution of the arrivals, they prescribe a strategy that minimizes expected long term cost per unit time. Further, a queueing system with bulk service at scheduled times has been considered in [13] where the customers can pick the arrival time to minimize the waiting time. Under certain conditions, the authors establish that it is



(a) $\bar{W}_\iota$ under linear speedup. (b) $\bar{W}_\iota$ under logarithmic speedup. (c) $\bar{W}_\iota$ under power speedup.

Fig. 6: The waiting time $\bar{W}_\iota$ for a system with 6 servers under the second set of speedup forms: The linear case shows similar pattern to the logarithmic form due its strong speedup implication whereas the power form fails to produce an optimum.

optimal to arrive just the instant before a service starts. Further, in [14], order batching in a synchronised zone order picking system has been analyzed to design and formulate the selection process in such a system. Note that, unlike our work, none of these articles consider batching in the context of a queueing system that employs split-and-merge technique.

Although there are useful results on adaptive allocation in FJ systems, we restrict our attention here to non-adaptive allocation which has been the focus of our work. It has been long known that exact analysis of FJ systems is hard [15]. Exact results are only known for joint workload distribution when there are two queues and under the assumption of Poisson arrival and independent exponential service times [16]. Hence, most of the results aim to bound the tail distribution of the waiting time for single stage [7], [17], [15] or multi-stage systems [18]. FJ system with batch arrivals have been looked into in [19], although task batching has been largely unexplored to the best of our knowledge.

V. Conclusion

In this work, we have looked at a Fork-Join model of parallel computation/communication systems where the input jobs are split into tasks that are *batched* for performance enhancement. This is a natural extension to the observation that task-batching has been observed to be beneficial for simpler non-parallelized systems. We have explored three different analytical forms for speedup, i.e., the batching gain, namely, linear, logarithmic and power law gain and looked at the corresponding steady state performance of the system in the high utilization regime. While there is not much effect on the average throughput, there exists an optimal batch size for which the job waiting time is minimized for a strong enough speedup. This optimal batch size is, however, non-trivial as it may vanish for some combinations of the arrivals and the batching configuration.

References

[1] A. Thomasian, "Analysis of fork/join and related queueing systems," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–71, 2014.

[2] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 2, no. 2, pp. 1–30, 2017.

[3] I. A. T. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan, "Mapreduce: Review and open challenges," *Scientometrics*, vol. 109, no. 1, pp. 389–422, 2016.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[5] E. Cree, "Linux Kernel path: "handle-multiple-received-packets-at-each-stage"," Retrieved October 27, 2018 from https://github.com/torvalds/linux/commit/2d1b138505dc29bbd7ac5f82f5a10635ff48bddb, 2018, accessed: 2018-10-27.

[6] R. Rehrmann, C. Binnig, A. Böhm, K. Kim, W. Lehner, and A. Rizk, "OLTPshare: The case for sharing in OLTP workloads," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 1769–1780, Aug. 2018.

[7] A. Rizk, F. Poloczek, and F. Ciucu, "Computable bounds in fork-join queueing systems," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 335–346.

[8] P. Hoel, S. Port, and C. Stone, *Introduction to Probability Theory*, ser. The Houthton Mifflin series in statistics. Houghton Mifflin, 1973. [Online]. Available: https://books.google.de/books?id=m1YmtAEACAAJ

[9] N. T. Bailey, "On queueing processes with bulk service," *J. R. Stat. Soc. B. Met.*, pp. 80–87, 1954.

[10] R. K. Deb and R. F. Serfozo, "Optimal control of batch service queues," *Advances in Applied Probability*, vol. 5, no. 2, pp. 340–361, 1973.

[11] R. K. Deb, "Optimal dispatching of a finite capacity shuttle," *Manag. Science*, vol. 24, no. 13, pp. 1362–1372, 1978.

[12] M. Berg, F. van der Duyn Schouten, and J. Jansen, "Optimal batch provisioning to customers subject to a delay-limit," *Management science*, vol. 44, no. 5, pp. 684–697, 1998.

[13] A. Glazer and R. Hassin, "Equilibrium arrivals in queues with bulk service at scheduled times," *Transportation Science*, vol. 21, no. 4, pp. 273–278, 1987.

[14] L. Pan, J. Z. Huang, and S. C. Chu, "Order batching and picking in a synchronized zone order picking system," in *2011 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, 2011, pp. 156–160.

[15] F. Baccelli, A. M. Makowski, and A. Shwartz, "The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds," *Advances in Applied Probability*, vol. 21, no. 3, pp. 629–660, 1989.

[16] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands i," *SIAM Journal on Applied Mathematics*, vol. 44, no. 5, pp. 1041–1053, 1984.

[17] W. R. KhudaBukhsh, A. Rizk, A. Frömmgen, and H. Koeppl, "Optimizing stochastic scheduling in fork-join queueing models: Bounds and applications," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[18] M. Fidler and Y. Jiang, "Non-asymptotic delay bounds for (k, l) fork-join systems and multi-stage fork-join networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[19] Y. Sun, C. E. Koksal, and N. B. Shroff, "Near delay-optimal scheduling of batch jobs in multi-server systems," *Ohio State Univ., Tech. Rep*, 2017.