Robustness Comparison of Scheduling Algorithms in MapReduce Framework

Amirali Daghighi\textsuperscript{1} · Jim Q. Chen\textsuperscript{2}

\textbf{Abstract} Parallel computing is the fundamental base for MapReduce framework in Hadoop. A big data is split into small data chunks, where Map task is referred to processing a data chunk. Each data chunk is replicated over three servers in Hadoop for increasing the availability of data and decreasing the probability of data loss. As a result, the three servers that have the Map task stored on their disk are the fastest servers to process them, which are called local servers. All the servers in the same rack as local servers are called rack-local servers that are slower than local servers in processing Map tasks since the data chunk associated with the Map task should be fetched through the top of the rack switch. All the other servers are called remote servers that are the slowest servers for processing a Map task since they need to fetch data from a local server in another rack, so data should be transmitted through at least two top of the rack switches and a core switch. Note that the number of switches in the path of data transfer depends on the internal network structure of data centers. The First-In-First-Out (FIFO) and Hadoop Fair Scheduler (HFS) algorithms do not take the rack structure of data centers into account, so they are known to not be heavy-traffic delay optimal or even throughput optimal. The recent advances on scheduling for data centers considering the rack structure of them and the heterogeneity of servers resulted in the state-of-the-art Balanced-PANDAS algorithm that outperforms the classic MaxWeight algorithm and its derivation, JSQ-MaxWeight algorithm. In both Balanced-PANDAS and MaxWeight algorithms, the processing rate of local, rack-local, and remote servers are assumed to be known. However, with the change of traffic over time in addition to estimation errors of processing rates, it is not realistic to consider the processing rates to be known. In this work, we study the robustness of Balanced-PANDAS and MaxWeight algorithms in

\begin{footnotesize}
\begin{itemize}
\item \textsuperscript{1} adaghighi@stcloudstate.edu
\item St. Cloud State University
\item \textsuperscript{2} jchen@stcloudstate.edu
\item St. Cloud State University
\end{itemize}
\end{footnotesize}
terms of inaccurate estimations of processing rates. We observe that Balanced-PANDAS is not as sensitive as MaxWeight on the accuracy of processing rates, making it more appealing to use in data centers.

Keywords Hadoop · MapReduce · Data center · Scheduling · Load balancing · Robustness

1 Introduction and Related Work

Parallel computing for big data has different applications from online social networks, health-care industry, advertisement placement, and machine learning to search engines and data centers. The most popular big data parallel computing framework is MapReduce which is broadly used in Hadoop [60], Dryad [25], Google [14], Deep Learning [11], and grid-computing [26, 46, 47]. Before talking about MapReduce, we present some details on the network structure of data centers. Data centers used to mainly consist of two parts, storage and computing, and these two parts were connected to each other through a network link with high bandwidth. At the moment that this structure was being used, not much of big data was present, so the communication of data from storage to processing unit was not creating any bottleneck. However, by emergence of big data in health-care industry, machine learning applications, search engines, etc., the network between the two units was unable to accommodate fast and reliable data transfer. Hence, scientists come up with the idea of bringing data into processing unit by splitting both data and processing units into hundreds of small units and combining each computing unit with a small storage unit. As a result, each small unit consisting of storage and processing units, called a server, is capable of storing small pieces of data and processing them at the same time. In other words, data does not need to be transferred from storage unit to processing unit since they are already together. However, note that the big data cannot be stored in the storage unit of a single server. The solution is to split the big data into small chunks of data, typically 68-128 MB, and storing them on multiple servers. In practice, each data chunk is stored on three servers to increase availability and decrease data loss probability. As a result, processing of big data consists of processing of multiple data chunks that make the whole big data, and concatenating the data chunk processing results for the completion of the big data process. The processing of each data chunk is called Map task and the concatenation of the results on all the Map tasks is called Reduce task, which make up the MapReduce processing framework for big data. Note that at least for Reduce tasks, servers need to be connected and cannot completely be isolated. We later see that even for executing Map tasks, servers may need to exchange data. Hence, servers are connected to each other through links and switches. The structure of switches connecting servers is a complete field of research in computer science, but servers are generally connected to each other by top of the rack switches as well as core switches in the following way. The hundred servers are grouped into batches of 20-50 servers, where each batch is inter-connected to
each other by a switch, called rack switch, and all rack switches are connected to one or more core switches which make all the servers connected.

The rack structure of data centers brings a lot of complexity for load balancing. As a result, most theoretical work on load balancing for data centers either consider homogeneous model of servers or ignore the rack structure and only consider data locality, where data locality refers to the fact that the data chunk associated to a Map task is stored on three servers, so is not available on other servers immediately. Examples of works that consider homogeneous server model are [31], [52], [29], and [32]. A branch of research on homogeneous servers is utilization of the power of two or more choices for load balancing that lowers the messaging overhead between hundreds of servers and the core load balancing scheduler. For example you can refer to [13], [37], [19], [43], [6], [16], [54], [10], [53], and [20]. Although there has been a huge body of work on heuristic algorithms for heterogeneous server model, examples of which are [59], [26], [70], [27], [23], [42], [71], [12], [28], [38], and [49], there has been a few recent works on algorithms with theoretical guarantees on such more complicated models.

The scheduling problem for a data center with a rack structure is a specific case of the open affinity scheduling problem, where each task type can be processed by each server but with different processing rates. The classic MaxWeight algorithm [54], [48], [53], c-µ-rule [55], and the work by Harrison and Lopez [22], [21] and Bell and Williams [4], [3] have different approaches on solving the load balancing problem for the affinity scheduling, but they either not solve the delay optimality or have unrealistic assumptions including known task arrival rates and existence of one queue per task type. The state-of-the-art on scheduling for data centers considering the rack structure, no knowledge of task arrival rates, and having queues on the order of servers not the number of task types is presented by Xie et al. [63] and Yekkehkhany [65], which is extended for a general number of data locality levels by Yekkehkhany et al. [67]. The central idea to all algorithms in [63], [65], and [67] is to use weighted workload on servers instead of the queue lengths, which leads to a better performance in terms of average delay experienced by submitted tasks. The Balanced-PANDAS algorithm, where PANDAS stands for Priority Algorithm for Near-Data Scheduling, is the name for the weighted-workload based algorithms proposed in [63], [65], and [67]. The Join-the-Shortest-Queue-MaxWeight (JSQ-MaxWeight, JSQ-MW) proposed by Wang et al. [57] that only considers data locality is also extended to the case where rack structure is considered by Xie et al. [63]. The priority algorithm proposed by [62] is another work that only considers data locality, not the rack structure of data centers which is interesting in its own rights since both throughput and heavy-traffic delay optimality are proved for it; however, it is not even throughput optimal for a system with rack structure.

All of the algorithms mentioned above consider complete knowledge about the processing rates of different task types on different servers. However, the reality is that the processing rates are mostly not known due to errors in estimation methods and the change of the system structure over time or the
change of traffic which can change the processing rates. Hence, it is important that the algorithm that is used for load balancing is robust to estimation errors of processing rates. In this work, we run extensive simulations to evaluate the robustness of the state-of-the-art algorithms on load balancing with different levels of data locality. It is observed that the Balanced-PANDAS algorithm not only has a better heavy-traffic delay performance, but it also is more robust to changes of processing rate estimations, while MaxWeight based algorithm does not perform as well as Balanced-PANDAS under processing rate estimation errors. In order to estimate the processing rates of tasks on servers and better model the data center structure, reinforcement learning methods can be used as it is discussed in [69], [40], [41], [39]. A recent work by Yekkehkhany and Nagi [68] considers an exploration-exploitation approach as in the reinforcement learning method to both learn the processing rates and exploit load balancing based on the current estimation of the processing rates. They propose the Blind GB-PANDAS algorithm that is proven to be throughput-optimal and have a lower mean task completion time than the existing methods. A more sophisticated risk-averse exploration-exploitation approach can be considered for this problem as discussed in [66].

The rest of the paper is organized as follows. Section 2 presents the system model that is used throughout the paper, section 3 summarizes the preliminary materials including the description of Balanced-PANDAS, Priority, and MaxWeight based algorithms that are needed before we present the robustness comparison among different algorithms in section 4.

2 System Model

We consider the same system model described in [63] and [67] as follows. A discrete time model is considered, where time is indexed by $t \in \mathbb{N}$. Assume a data center with $M$ servers and denote the set of all servers as $\mathcal{M} = \{1, 2, \cdots, M\}$. Without loss of generality, assume that the first $M_R$ servers are connected to each other with a top of the rack switch and are called the first rack, the second $M_R$ servers are connected to each other with another top of the rack switch and are called the second rack, and so on. Hence, there are $N_R = \frac{M}{M_R}$ racks in total. All the top of the rack switches are connected to each other with one or more core switches in a symmetric manner. As a result, there are three levels of data locality as described below. Recall that the data chunk associated to a map task is stored on three servers by Hadoop’s default, so all those three servers are called local servers for the map task or in other words the map task can receive service locally from those three servers. Since servers have the data chunk of local tasks, the processing is immediately started after servers are assigned to process them. Note that the three servers storing the data for a map task is normally different for different map tasks. Hence, we associate a type to each map task, which is the label of the three servers, i.e. $(m_1, m_2, m_3) \in \mathcal{M}^3$, such that $m_1 < m_2 < m_3$. This gives us a unique and
informative way of representation for different task types as follows:

\[ L \in \mathcal{L} = \{(m_1, m_2, m_3) \in \mathcal{M}^3 : m_1 < m_2 < m_3 \}, \]

where a task type is denoted by \( L = (m_1, m_2, m_3) \) given that \( m_1, m_2, \) and \( m_3 \) are the three local servers for task of type \( L \) and the set of all task types is denoted by \( \mathcal{L} \).

A map task is not limited to receive service from one of the local servers. It can receive service from one of the servers that are in the same rack as the local servers with a slightly lower service rate. The slight depreciation of processing rate for such servers, which are called rack-local servers, is for the travel time of the data associated to a map task from a local server to the rack-local server that is assigned for processing the map task rack-locally. Finally, all other servers other than the local and rack-local servers, which are called remote servers, have the lowest processing rate for a map task, since data needs to be transmitted through at least two of the top rack switches and a core switch, so the server cannot immediately start processing the task when it is assigned to do so remotely. In order to formally define the rack-local and remote servers, we need to propose a notation. Let \( R(m) \in \{1, 2, \ldots, N_R\} \) denotes the label of the rack that the \( m \)-th server belongs to. Then, the set of rack-local and remote servers to task of type \( L = (m_1, m_2, m_3) \), denoted by \( \tilde{L}_k \) and \( \tilde{L}_r \), respectively, are as follows:

\[ \tilde{L}_k = \left\{ m \in \mathcal{M} : m \notin (m_1, m_2, m_3), R(m) \in \left( R(m_1), R(m_2), R(m_3) \right) \right\}, \]

\[ \tilde{L}_r = \left\{ m \in \mathcal{M} : R(m) \notin \left( R(m_1), R(m_2), R(m_3) \right) \right\}. \]

The service and arrival process of tasks is described below.

**Task arrival process:** Let \( A_L(t) \) denote the number of tasks of type \( L \) that arrive to the system at time slot \( t \), where \( \mathbb{E}[A_L(t)] = \lambda_L \), and it is assumed that \( A_L(t) < C_A \) and \( P(A_L(t) = 0) > 0 \). The set of arrival rates for all task types is denoted by \( \lambda = (\lambda_L : L \in \mathcal{L}) \).

**Service process:** The processing of a task on a local server is assumed to be faster than on a rack-local server, and the processing of a task on a rack-local server is faster than on a remote server. This fact is formalized as follows. The processing time of a task on a local, rack-local, and remote server has means \( \frac{1}{\alpha} \), \( \frac{1}{\beta} \), and \( \frac{1}{\gamma} \), respectively, where \( \alpha > \beta > \gamma \). Note that the processing time of a task can have any distribution with the given means, but the heavy-traffic delay optimality of Balanced-PANDAS algorithm is only proven under Geometric service time distribution, while MaxWeight based algorithm does not have a general heavy-traffic delay optimality under any distribution for service time.

**Capacity region characterization:** An arrival rate for all task types is supportable for service by the \( M \) servers if and only if the load on each server is strictly less than the capacity of the server. Considering a processing rate of
one for each server, an arrival rate vector \( \lambda = (\lambda_L : L \in \mathcal{L}) \) is in the capacity region of the system if and only if:

\[
\sum_{L,m \in \mathcal{L}} \frac{\lambda_{L,m}}{\alpha} + \sum_{L,m \in \mathcal{L}_k} \frac{\lambda_{L,m}}{\beta} + \sum_{L,m \in \mathcal{L}_r} \frac{\lambda_{L,m}}{\gamma} < 1, \quad \forall m \in \mathcal{M},
\]

where \( \lambda_{L,m} \) is the rate of incoming tasks of type \( L \) that are processed by server \( m \).

### 3 Load Balancing Algorithms

In this section, we introduce the main three algorithms on scheduling for data centers with more than or equal to two levels of data locality that have theoretical guarantees on optimality in some senses and under some conditions. In order to introduce the load balancing algorithm of each method, we also need to present the queueing structure required for that method. The three algorithms are:

1. **Priority algorithm** [62], which is best fit for applications with two levels of data locality, e.g. for the cases that only data locality is taken into account. An example is scheduling for Amazon Web Services inside a rack.
2. **Balanced-PANDAS** [63], [65], and [67], which is the state-of-the-art for scheduling applications with multiple levels of data locality and is observed to perform better in terms of average task completion time by fourfold in comparison to MaxWeight based algorithms. It is proven by Xie et al. [63] that under mild conditions, Balanced-PANDAS is both throughput and heavy-traffic delay optimal for a system with three levels of data locality and a rack structure.
3. **MaxWeight based algorithms** [53] and [58], which can be used for multiple levels of data locality and are throughput optimal, but not heavy-traffic delay optimal, and it is observed that they generally have poor performance at high loads compared to weighted workload based algorithm used in Balanced-PANDAS algorithm.

The following three subsections present a complete introduction to these three main algorithms.

#### 3.1 Priority algorithm

The Priority algorithm is designed for a system with two levels of data locality. In other words, it only considers data locality, but not the rack structure. Hence, there are only local and remote servers from the perspective of a task. The queueing structure under this algorithm is to have a single queue per server, where the queue corresponding to a server only keeps tasks that are local to that server. At the arrival of a task, a central scheduler routes the incoming task to the local server with the shortest queue length. An idle server
is scheduled to process a task in its corresponding queue as long as there is one, and if the idle server’s queue length is zero, it is scheduled to process a task from the longest queue in the system. The priority algorithm is proved to be both throughput and heavy-traffic delay optimal. However, its extension to more than two levels of data locality is not even throughput optimal, let alone heavy-traffic delay optimal.

3.2 Balanced-PANDAS algorithm

The Balanced-PANDAS algorithm can be used for a system with multiple levels of data locality, but here we propose the algorithm for a data center with a rack structure with three levels of data locality. The queueing structure under using this algorithm is to have three queues per server, one queue for storing local tasks to the server, another queue for storing rack-local tasks to the server, and a third queue for storing remote tasks to the server. Hence, server $m$ has a tuple of three queues denoted by $(Q^l_m, Q^k_m, Q^r_m)$, where they refer to the queues storing local, rack-local, and remote tasks respectively. The corresponding queue lengths at time $t$ are denoted by $(Q^l_m(t), Q^k_m(t), Q^r_m(t))$.

The workload on server $m$ at time slot $t$ is defined as follows:

$$W_m(t) = \frac{Q^l_m(t)}{\alpha} + \frac{Q^k_m(t)}{\beta} + \frac{Q^r_m(t)}{\gamma}.$$

An incoming task of type $\bar{L}$ is routed to the corresponding queue of the server with the minimum weighted workload, where ties are broken randomly, in the set below:

$$\arg \min_{m \in M} \left\{ \frac{W_m(t)}{\alpha \cdot 1_{\{m \in \bar{L}\}} + \beta \cdot 1_{\{m \in \bar{L}_k\}} + \gamma \cdot 1_{\{m \in \bar{L}_r\}}} \right\}.$$

An idle server $m$ at time slot $t$ is scheduled to process a local task from $Q^l_m$ if $Q^l_m(t) \neq 0$; otherwise, it is scheduled to process a rack-local task from $Q^k_m$ if $Q^k_m(t) \neq 0$; otherwise, it is scheduled to process a remote task from $Q^r_m$ if $Q^r_m(t) \neq 0$; otherwise, it remains idle until a task joins one of its three queues. The Balanced-PANDAS algorithm is throughput optimal. It is also heavy-traffic delay optimal for a system with a rack structure of three levels of data locality if $\beta^2 > \alpha \cdot \gamma$, which means that the rack-local service is faster than the remote service in a specific manner.

3.3 MaxWeight based algorithm

The MaxWeight algorithm is proposed by [53] and a modification of it called JSQ-MaxWeight algorithm is proposed by [58], which is described below. Consider one queue per server, i.e. server $m$ has a single queue called $Q_m$, where its queue length at time slot $t$ is denoted by $Q_m(t)$. The routing policy is as
the Priority algorithm, i.e. an incoming task of type $\bar{L}$ is routed to the queue of the shortest length in $\bar{L}$. This routing policy is called join-the-shortest-queue (JSQ). An idle server $m$ at time slot $t$ on the other hand is scheduled to process a task from a queue with the maximum weighted queue length, where ties are broken at random, in the set below:

$$\text{arg max}_{n \in M} \left\{ (\alpha \cdot 1_{\{n=m\}} + \beta \cdot 1_{\{n \neq m, R(n) = R(m)\}} + \gamma \cdot 1_{\{R(n) \neq R(m)\}} \right\} \cdot Q_n(t).$$

The JSQ-MaxWeight algorithm is throughput optimal, but it is not heavy-traffic delay optimal.

4 Robustness Comparison of Scheduling Algorithms

In this section, we present the results on our extensive simulations on robustness of scheduling algorithms presented in section 3. To this end, we run the algorithms with parameters that have error to study which algorithm can tolerate errors better than others. More specifically, we use incorrect $\alpha, \beta,$ and $\gamma$ in the algorithms for calculating weighted workloads or weighted queue lengths and observe the average task completion time under these scenarios. The arrival process is a Poisson process and the processing time has an exponential distribution. We have also tested the algorithms for processing times with heavy-tailed distributions and observed similar results. We make these parameters 5%, 10%, 15%, 20%, 25%, and 30% off their real value, either greater than the real value or smaller than the real value, and evaluate algorithms under these cases. The traffic load is assumed the same under all algorithms so that the comparison makes sense. We further compare all the three algorithms mentioned in section 3 with the Hadoop’s default scheduler which is First-In-First-Out (FIFO). Figure 1 shows the comparison between the four algorithms when precise value of parameters are known by the central scheduler. As we see, Balanced-PANDAS algorithm has the lowest average task completion time at high loads. A closer look of high loads is presented in figure 2, where Balanced-PANDAS obviously outperform JSQ-MaxWeight in terms of average task completion time.

The performance of the four algorithms are compared to each other when the parameters are lower than their real values by certain percentages, where the results are shown in figure 3. As is seen, Balanced-PANDAS has best performance among all algorithms by changing the parameters’ error from 5% to 30%. In fact, figure 4 shows that the Balanced-PANDAS has the least sensitivity against change of parameters while JSQ-MaxWeight’s performance varies notably by the increase of error in parameter estimations.

Comparison of the algorithms when the parameters are off for some percentages, but higher than their real values are given in figure 5. It is again observed that the Balanced-PANDAS algorithm has consistent better performance than the JSQ-MaxWeight algorithm. The sensitivity comparison of the Balanced-PANDAS and JSQ-MaxWeight algorithms in this case is presented in figure 6.
5 Conclusion and Future Work

In this work, we did a literature review on both classical and state-of-the-art scheduling algorithms for the affinity scheduling problem. Data center load balancing is a special case of the affinity scheduling problem. Considering the rack structure of data centers, there are three levels of data locality. The priority algorithm that is heavy-traffic delay optimal is not even throughput optimal for three levels of data locality. The Balanced-PANDAS algorithm is the state-of-the-art in heavy-traffic delay optimality. We investigated the robustness of Balanced-PANDAS and JSQ-MaxWeight algorithms with respect to errors in parameter estimation. We observe that Balanced-PANDAS keeps its better performance even in the absence of precise parameter values versus JSQ-MaxWeight. Note that the JSQ-MaxWeight algorithm is also robust under parameter estimation errors, but it is more sensitive than Balanced-PANDAS, specially at high loads close to the boundary of the capacity region. For future work, one can use machine learning tools to estimate the system parameters and make them more precise in the meanwhile that the load bal-
Fig. 3 Robustness comparison of algorithms when parameters are off and lower than their real values.

Fig. 4 Sensitivity comparison of Balanced-PANDAS and JSQ-MaxWeight against parameter estimation error.
Fig. 5 Robustness comparison of algorithms when parameters are off and higher than their real values.

Fig. 6 Sensitivity comparison of Balanced-PANDAS and JSQ-MaxWeight against parameter estimation error.
ancing algorithm is working with the estimated parameters. The scheduling algorithms presented in this work can also be applied to a vast number of applications including but not limited to healthcare and supermarket models [61], [9], [17], [24], web search engines [51], [50], [5], [30], [64], electric vehicle charging [1], [18], [56], [15], [8], [2], [7], [44], [45] and so on.
References

1. Bahram Alinia, Mohammad Sadegh Talebi, Mohammad H Hajiesmaili, Ali Yekkehkhany, and Noel Crespi. Competitive online scheduling algorithms with applications in deadline-constrained EV charging. In *North American Power Symposium (NAPS)*, 2015. IEEE, 2015.
2. Mishari Metab Almalki, Maziar Isapour Chehardeh, and Constantine J Hatziadoniu. Capacitor bank switching transient analysis using frequency dependent network equivalents. *Electronic Journal of Probability*, 10:1044–1115, 2005.
3. Steven Bell, Ruth Williams, et al. Dynamic scheduling of a parallel server system in heavy traffic with complete resource pooling: Asymptotic optimality of a threshold policy. *The Annals of Applied Probability*, 11(3):608–649, 2001.
4. Andrei Broder. A taxonomy of web search. In *ACM Sigir forum*, volume 36, pages 3–10. ACM, 2002.
5. John W Byers, Jeffrey Considine, and Michael Mitzenmacher. Geometric generalizations of the power of two choices. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 54–63. ACM, 2004.
6. Maziar Isapour Chehardeh, Mishari Metab Almalki, and Constantine J Hatziadoniu. Remote feeder transfer between out-of-phase sources using stxs. In *Power and Energy Conference at Illinois (PECI)*. IEEE, 2016.
7. Steven L Bell, Ruth J Williams, et al. Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: asymptotic optimality of a threshold policy. *The Annals of Applied Probability*, 11(3):608–649, 2001.
8. Andrei Broder. A taxonomy of web search. In *ACM Sigir forum*, volume 36, pages 3–10. ACM, 2002.
9. John W Byers, Jeffrey Considine, and Michael Mitzenmacher. Geometric generalizations of the power of two choices. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 54–63. ACM, 2004.
10. Maziar Isapour Chehardeh and Constantine J Hatziadoniu. A systematic method for reliability index evaluation of distribution networks in the presence of distributed generators. In *2018 North American Power Symposium (NAPS)*. IEEE, 2018.
11. Robert Clower and Axel Leijonhufvud. The coordination of economic activities: a keynesian perspective. *The American Economic Review*, 65(2):182–188, 1975.
12. Colin Cooper, Robert Elsasser, and Tomasz Radzik. The power of two choices in distributed voting. In *International Colloquium on Automata, Languages, and Programming*, pages 435–446. Springer, 2014.
13. Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1631–1642. SIAM, 2016.
14. Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 149–158. ACM, 2011.
15. Elizabeth Eisenhauer. In poor health: Supermarket redlining and urban nutrition. *Geojournal*, 53(2):125–133, 2001.
16. Lingwen Gan, Ufuk Topcu, and Steven H Low. Optimal decentralized protocol for electric vehicle charging. *IEEE Transactions on Power Systems*, 28(2):940–951, 2013.
17. J Michael Harrison. Heavy traffic analysis of a system with parallel servers: asymptotic optimality of discrete-review policies. *Annals of Applied Probability*, pages 822–848, 1998.
22. J Michael Harrison and Marcel J López. Heavy traffic resource pooling in parallel-server systems. Queueing systems, 33(4):339–368, 1999.
23. Chen He, Ying Lu, and David Swanson. Matchmaking: A new mapreduce scheduling technique. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 40–47. IEEE, 2011.
24. Mohammad Hosseini, Yu Jiang, Ali Yekkehkhany, Richard R Berlin, and Lui Sha. A mobile geo-communication dataset for physiology-aware dash in rural ambulance transport. In Proceedings of the 8th ACM Multimedia Systems Conference. ACM, 2017.
25. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In ACM SIGOPS operating systems review, volume 41, pages 59–72. ACM, 2007.
26. Michael Isard, Vijayan Prabahakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 261–276. ACM, 2009.
27. Jiahui Jin, Junzhou Luo, Aibo Song, Fang Dong, and Runqun Xiong. Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 295–304. IEEE Computer Society, 2011.
28. Mohammadamir Kavousi. Affinity scheduling and the applications on data center scheduling with data locality. arXiv preprint arXiv:1705.03125, 2017.
29. Joseph Kreimer. Real-time system with homogeneous servers and nonidentical channels in steady-state. Computers & Operations Research, 29(11):1465–1473, 2002.
30. K Hari Krishna and Kosuru Anusha Rani. Reducing the energy consumption energy-efficient query processing node in web search engines. 2018.
31. Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. In ACM SIGMETRICS Performance Evaluation Review, volume 11, pages 47–55. ACM, 1982.
32. James Craig Lowery, Mark Andrew Collins, and Brent Schroeder. Systems and methods for provisioning homogeneous servers, May 22 2008. US Patent App. 11/562,921.
33. Malwina J Luczak, Colin McDiarmid, et al. On the power of two choices: balls and bins in continuous time. The Annals of Applied Probability, 15(3):1733–1764, 2005.
34. Steve Lumetta and Michael Mitzenmacher. Using the power of two choices to improve bloom filters. Internet Mathematics, 4(1):17–33, 2007.
35. Avishai Mandelbaum and Alexander L Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized $c\mu$-rule. Operations Research, 52(6):836–855, 2004.
36. Sean Meyn. Stability and asymptotic optimality of generalized maxweight policies. SIAM Journal on control and optimization, 47(6):3299–3294, 2009.
37. Michael Mitzenmacher. The power of two choices in randomized load balancing. IEEE Transactions on Parallel and Distributed Systems, 12(10):1094–1104, 2001.
38. Amir Moaddeli, Iman Nabati Ahmadi, and Negin Abhar. The power of d choices in scheduling for data centers with heterogeneous servers. arXiv preprint arXiv:1904.00447, 2019.
39. Negin Musavi. A game theoretical framework for the evaluation of unmanned aircraft systems airspace integration concepts. arXiv preprint arXiv:1904.08477, 2019.
40. Negin Musavi, Deniz Onural, Kerem Gunes, and Yildiray Yildiz. Unmanned aircraft systems airspace integration: A game theoretical framework for concept evaluations. Journal of Guidance, Control, and Dynamics, pages 96–109, 2016.
41. Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. Combinatorial Optimization, 9:255–304, 2001.
44. Sepehr Saadatmand, Sima Azizi, Mohammad Amir Kavousi, and Donald Wunsch. Autonomous control of a line follower robot using a q-learning controller. In 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), pages 0556–0561. IEEE, 2020.

45. Sepehr Saadatmand, Mohammad Amir Kavousi, and Sima Azizi. The voltage regulation of boost converters using dual heuristic programming. In 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), pages 0531–0536. IEEE, 2020.

46. Sepehr Saadatmand, Mohammad Saleh Sanjarinia, Pourya Shamsi, and Mehdi Ferdowsi. Dual heuristic dynamic programming control of grid-connected synchronverters. North American Power Symposium (NAPS), 2019, pages 1–6, 2019.

47. Sepehr Saadatmand, Mohammad Saleh Sanjarinia, Pourya Shamsi, Mehdi Ferdowsi, and Donald C Wunsch. Heuristic dynamic programming for adaptive virtual synchronous generators. North American Power Symposium (NAPS), 2019, pages 1–6, 2019.

48. Bilal Sadiq and Gustavo De Veciana. Throughput optimality of delay-driven maxweight scheduler for a wireless system with flow dynamics. In Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on, pages 1097–1102. IEEE, 2009.

49. Mehdi Salehi. Optimal physiology-aware scheduling of clinical states in rural ambulance transport. In 2017 International Conference on Inventive Computing and Informatics (ICI CI), pages 247–252. IEEE, 2017.

50. Sara Salehi, Jia Tina Du, and Helen Ashman. Use of web search engines and personalisation in information searching for educational purposes. Information Research: An International Electronic Journal, 23(2):n2, 2018.

51. Candy Schwartz. Web search engines. Journal of the American Society for Information Science, 49(11):973–982, 1998.

52. Vijendra P Singh. Two-server markovian queues with balking: heterogeneous vs. homogeneous servers. Operations Research, 18(1):145–159, 1970.

53. Alexander L Stolyar et al. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. The Annals of Applied Probability, 14(1):1–53, 2004.

54. Peter van de Ven, Sem Borst, and Seva Shneer. Instability of maxweight scheduling algorithms. In INFOCOM 2009, IEEE, pages 1701–1709. IEEE, 2009.

55. Jan A Van Mieghem. Dynamic scheduling with convex delay costs: The generalized c—mu rule. The Annals of Applied Probability, pages 809–833, 1995.

56. Chwei-Sen Wang, Oskar H Stielau, and Grant A Covic. Design considerations for a contactless electric vehicle battery charger. IEEE Transactions on industrial electronics, 52(5):1308–1314, 2005.

57. Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. A throughput optimal algorithm for map task scheduling in mapreduce with data locality. ACM SIGMETRICS Performance Evaluation Review, 40(4):31–42, 2013.

58. Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. IEEE/ACM Transactions on Networking (TON), 24(1):190–203, 2016.

59. Tom White. Hadoop: The definitive guide, yahoo, 2010.

60. Tom White. Hadoop: The definitive guide. "O'Reilly Media, Inc.", 2012.

61. Fedelma Winkler. Consumerism in health care: beyond the supermarket model. Policy & Politics, 15(1):1–8, 1987.

62. Qiaomin Xie and Yi Lu. Priority algorithm for near-data scheduling: Throughput and heavy-traffic optimality. In Computer Communications (INFOCOM), 2015 IEEE Conference on, pages 963–972. IEEE, 2015.

63. Qiaomin Xie, Ali Yekkehkhany, and Yi Lu. Scheduling with multi-level data locality: Throughput and heavy-traffic optimality. In INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, IEEE, 2016.

64. Xiaohui Xie, Yiqun Liu, Maarten de Rijke, Jiying He, Min Zhang, and Shaoping Ma. Why people search for images using web search engines. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pages 655–663. ACM, 2018.
65. Ali Yekkehkhany. Near data scheduling for data centers with multi levels of data locality. *(Dissertation, University of Illinois at Urbana-Champaign)*.

66. Ali Yekkehkhany, Ebrahim Arian, Mohammad Hajiesmaili, and Rakesh Nagi. Risk-averse explore-then-commit algorithms for finite-time bandits. *arXiv preprint arXiv:1904.13387*, 2019.

67. Ali Yekkehkhany, Avesta Hojjati, and Mohammad H Hajiesmaili. Gb-pandas:: Throughput and heavy-traffic optimality analysis for affinity scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 45(2), 2018.

68. Ali Yekkehkhany and Rakesh Nagi. Blind gb-pandas: A blind throughput-optimal load balancing algorithm for affinity scheduling. *IEEE/ACM Transactions on Networking*, 2020.

69. Yildiray Yildiz, Adrian Agogino, and Guillaume Brat. Predicting pilot behavior in medium scale scenarios using game theory and reinforcement learning. In *AIAA Modeling and Simulation Technologies (MST) Conference*, page 4908, 2013.

70. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, pages 265–278. ACM, 2010.

71. Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Osdi*, volume 8, page 7, 2008.