Reproducing Results from Dominant Resource Fairness

Keshav Santhanam
Stanford University
keshav2@stanford.edu

Homero Roman Roman
Stanford University
homero@stanford.edu

1 INTRODUCTION

Cloud computing clusters have grown to tens of thousands of nodes serving hundreds of thousands of jobs [8]. Allocating cluster resources in a way that respects fairness while minimizing job response times is challenging at this scale. For example, Facebook found that the jobs submitted to the naive Hadoop FIFO scheduler suffered from prohibitively high queuing delays as the number of users of their data warehouse grew [9].

To address this challenge, researchers proposed new scheduling algorithms that used policies such as max-min fairness and statistical multiplexing to compute more effective allocations [4, 9]. However, these algorithms made scheduling decisions at the granularity of fix-sized slots which obscure the individual resources offered by each node (e.g. CPU, memory, I/O). As a result, algorithms that allocate slots fail to capture the heterogeneous resource requirements of modern data processing jobs.

The Dominant Resource Fairness (DRF) algorithm improves upon slot-based policies by providing a generalization of max-min fairness for jobs with diverse resource demands [1]. DRF computes each user's *dominant share* by selecting the largest aggregate proportional consumption of each resource among all the user's running jobs. Then at each scheduling step DRF selects the job that will maximize the minimum dominant share across all users. Ghodsi, et al. show that DRF is able to both improve resource utilization and reduce average job completion time compared to the previous slot-based Hadoop fair scheduler.

In this work we aim to revisit DRF and evaluate its lasting impact on the field of cluster scheduling. In particular, we seek to answer the following questions:

(1) Are the results reported in the DRF paper accurate?
(2) What challenges remained unsolved by DRF and how have more recent papers addressed those challenges?

To answer the first question, we reproduce Figures 9, 10, 13, and 14 in the DRF paper using a simulation-driven approach. Our goal is to measure metrics such as resource utilization and job completion time achieved by DRF compared to the Hadoop slot-based algorithm. We study the behavior of DRF using a microbenchmark trace derived from the paper and a large synthetic trace and find that DRF does in fact outperform the naive slot based policy for most use cases.

Next, we provide a literature review to analyze the progression of cluster scheduling algorithms published both before and after DRF. Our objective here is to identify the specific problems different cluster scheduling algorithms have solved and how these algorithms have evolved over time. In this way we can contextualize the results of DRF and understand how it has influenced the current state-of-the-art approach.

The rest of this paper proceeds as follows. § 2 provides more background information about the DRF algorithm. § 3 presents our simulation-based evaluation of DRF. § 4 offers a comprehensive study of the evolution of cluster scheduling algorithms, and § 5 concludes.

2 DOMINANT RESOURCE FAIRNESS

The DRF algorithm improves upon fixed-size slot-based policies by providing a generalization of max-min fairness for jobs with diverse, heterogeneous resource requirements. This means that DRF can compute allocations that more effectively partition cluster resources with respect to the individual resource demands of different tasks. DRF works in two main steps: First, DRF computes each user's *dominant resource share* which is given by the maximum proportion of a user's consumed resources with respect to the total available resources in the cluster. For example, consider a cluster with available resources of 9 CPU cores and 18 GB of memory. If user A's currently running tasks consume 3 CPU core and 3 GB memory then user A's dominant share is $\max\left(\frac{3}{9}, \frac{3}{18}\right) = \frac{1}{3}$. Then DRF selects the job that will maximize the minimum dominant share across all users. Consider the same clusters of with 9 CPU cores and 18 GB of memory available. Now suppose user B submits a task requiring 1 CPU core and 4 GB of memory, while user C requests 3 CPU cores and 1 GB memory. Then user B receives an allocation of [x CPU, $4x$ GB] and user B receives [$3y$ CPU, y GB] where x and y can be computed as 3 and 2 respectively by equalizing each user's dominant share (see Section 4.1 of the DRF paper for a more thorough walkthrough of this example).

3 EVALUATION

In this section we attempt to reproduce a subset of the results described in the DRF paper.

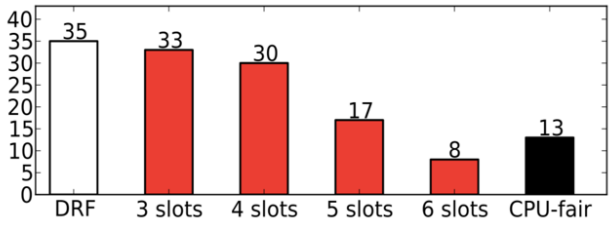

Figure 1: Original Figure 9

3.1 Implementation

We implemented a simulator and synthetic trace generator in Python to enable us to quickly model large scale experiments without the overhead of configuring a cluster. Our simulator models jobs as a series of dependent phases with each phase composed of a series of tasks. Each task has an individual resource demand vector and a specified duration. For simplicity, our trace generator assigns every task in the same phase an identical resource demand vector.

The scheduler accepts as input a set of jobs, a cluster configuration, and a policy. It then initializes the cluster by adding the jobs' initial tasks to a queue of pending tasks. In a loop, the scheduler handles any finished tasks, dispatches any newly arrived tasks, and runs any ready tasks. The loop continues until all jobs have completed. At each event, the scheduler calls a function exposed by the policy to handle any policy-specific state (for example, updating user resource vectors as in DRF).

Our implementation of DRF follows Algorithm 1 from the DRF paper [1]. For the baseline policy, we use the naive sharing algorithm presented in [9] (Algorithm 1).

3.2 Microbenchmark

We begin by attempting to reproduce the microbenchmark described in Section 7.2 of the DRF paper. The goal of these experiments is to demonstrate how DRF outperforms a naive slot-based policy regardless of how many slots are configured for each server. In addition, the authors compare DRF against a scheme that performs max-min fairness on only the CPU resources. In this setup there are 48 nodes, each with 8 CPU cores and 6 GB memory available. One group comprised of four users launches "small" tasks that request 0.5 GB memory and 1 CPU, and a second group also comprised of four users launches "large" tasks that request 2 GB memory and 2 CPU. Each job contains 80 tasks, and jobs are launched sequentially over a period of 10 minutes. This experiment measures how many jobs are able to complete within this allotted time.

Figures 1 and 3 present the original results from the DRF paper (Figures 9 and 10, respectively). The original Figure 9 illustrates how many large jobs were able to complete while

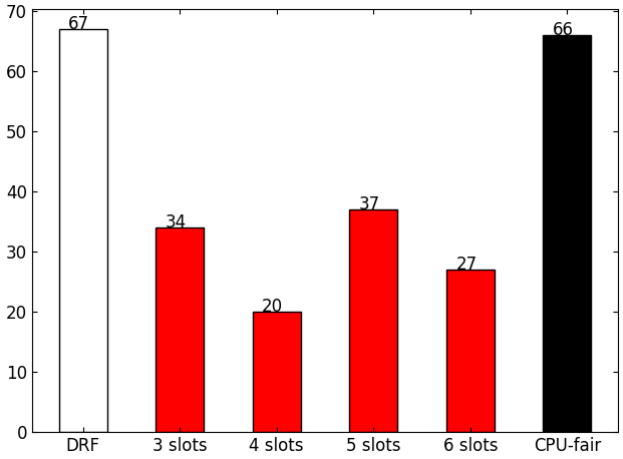

Figure 2: Reproduced Figure 9

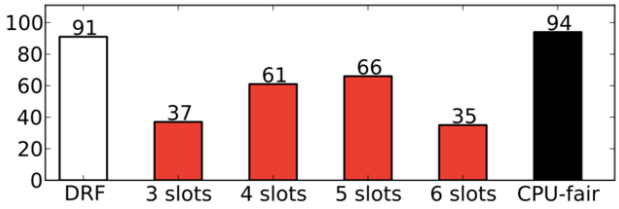

Figure 3: Original Figure 10

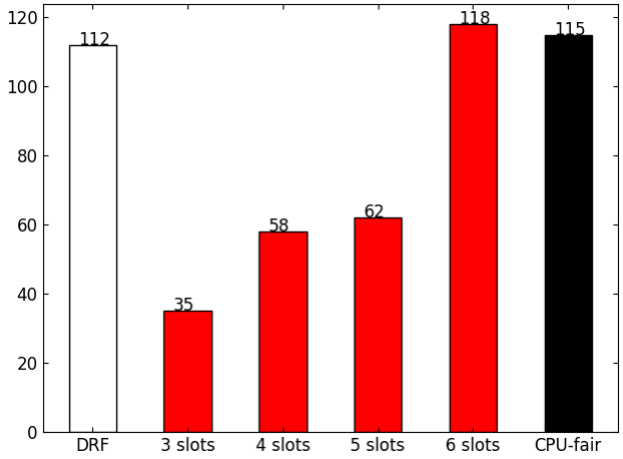

Figure 4: Reproduced Figure 10

Figure 10 shows how many small jobs complete. Figures 2 and 4 present our reproduced results. We observe that for small jobs, DRF and CPU-fairness performing approximately equally well - this is reflected in the original results, as well. Furthermore, with 3, 4, or 5 slots, our reproduction also

closely matches the original results. However, we find that our implementation of the naive slot-based baseline actually performs the best among the different schemes, completing a total of 118 jobs. We believe this is because our simulator does not accurately capture the effects of over-subscription as described in the paper. That is, with a large slot count, the original baseline policy would be able to place tasks onto nodes such that the aggregate resource consumption of the scheduled tasks exceeded the available resources on the node. As a result, the local OS scheduler would have to context-switch between tasks and introduce additional overhead. Given the complexity of modeling inter-task contention, our baseline implementation instead conservatively declines to schedule a task on a node if the node does not have available capacity for that task.

For large jobs, we see a much more significant disparity between the original results and the reproduced results. First, we see that DRF performs roughly twice as well as reported in the original paper. Furthermore, the CPU-fairness policy completes about 5× more jobs with our implementation. The performance of the slot-based fairness algorithm does not appear to align closely with Figure 9.

3.3 Synthetic Trace

Figures 13 and 14 of DRF presented results evaluated using a production trace provided by Facebook. We spoke with an author of the DRF paper who informed us that this trace was not available for public use. Instead, we evaluate using a large synthetic traces generated according to the following properties. The only explicit detail provided by the paper was that the cluster included 400 nodes, each with 32 GB available memory, 16 available CPU cores, and 12 slots. The remaining details are speculation on our part based on a best-effort attempt to replicate a realistic workload. In total our trace contains 1000 jobs. Each job is composed of two phases corresponding to a map phase and reduce phase. The number of tasks in each phase is determined by first uniformly randomly selecting an interval in the range [1, 500], [501, 1000], ... [3001, 3500] and then the precise number of tasks is again uniformly randomly sampled from the chosen interval. The number of CPU cores requested by each task is chosen uniformly randomly between 1, 2, and 4, and the amount of memory requested is also chosen uniformly randomly between 1, 2, and 4 GB. The duration of each phase is chosen uniformly randomly from the range [1, 100] seconds.

The DRF paper evaluated their trace in simulation to measure the affect of DRF on resource utilization as well as job completion time compared to the naive slot-based policy. We include Figures 5 and 7 as the original Figures 13 and 14 in the DRF paper. Figures 6 and 8 present our results using the synthetic trace.

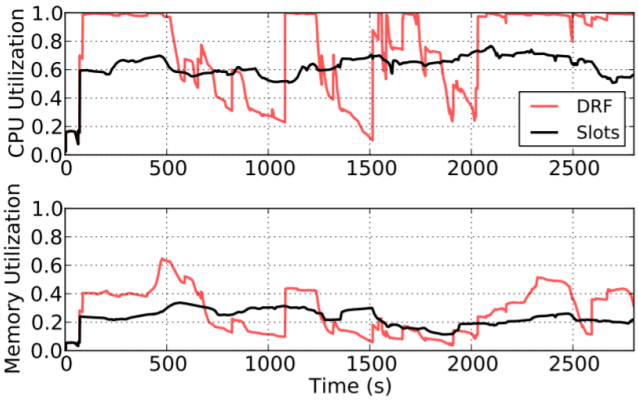

Figure 5: Original Figure 13

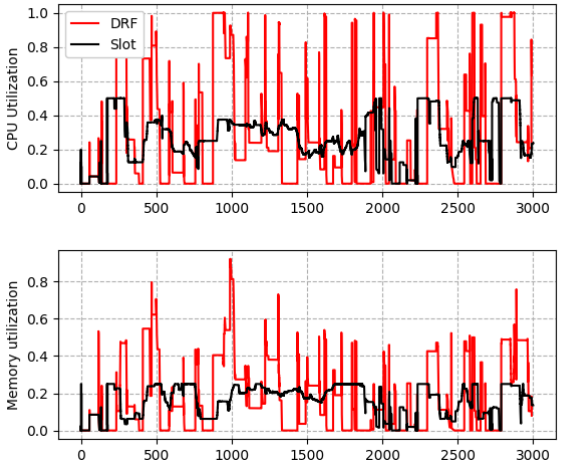

Figure 6: Reproduced Figure 13

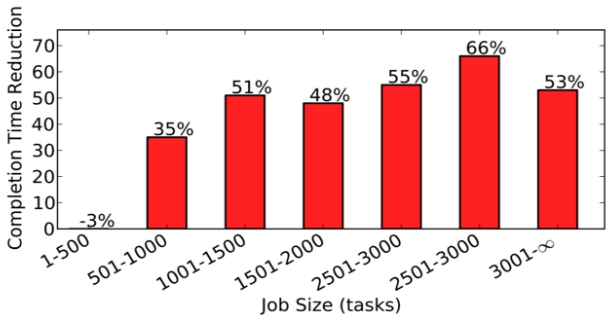

Figure 7: Original Figure 14

First with respect to resource utilization, we observe very similar trends as in the original paper. That is, we see in both cases that DRF resource utilization fluctuates heavily

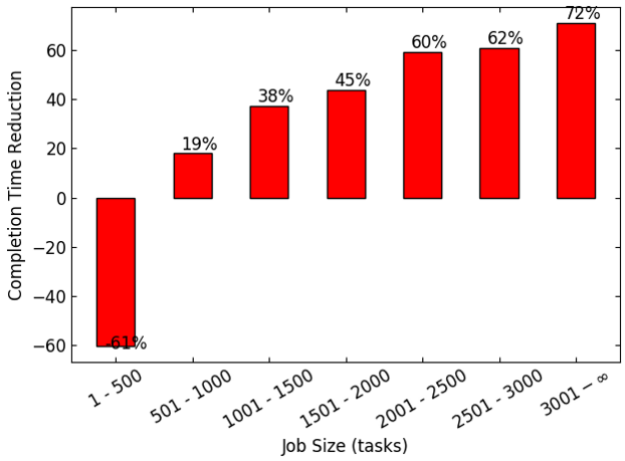

Figure 8: Reproduced Figure 14

but is able to occasionally reach peak or close-to-peak resource utilization. On the other hand, resource utilization with the slot-based policy remains relatively stable throughout, though at a fairly low utilization level. We do note that our reproduced graphs are far less smooth than the original. One possible explanation for this is the original graphs plotted far fewer samples than our graphs. Another possible explanation is that the resource requests of tasks in our trace are more diverse than the tasks in the Facebook trace, thereby resulting in more drastic swings in utilization.

We also see that our synthetic trace closely matches the results of the original Figure 14, with some exceptions. In particular, we observe that using DRF for our trace results in a significant performance downgrade for jobs with 1-500 tasks. We do not believe that this is a significant result however because only 35 of the 1000 total jobs were comprised of 500 tasks or less. Furthermore, the paper explains that for such short jobs DRF will intuitively not be useful because they do not have long-term implications on a user's fairness measurement. On the other hand, the remaining job sizes more closely match the original results.

3.4 Discussion

Overall our reproduced results show that DRF is able to outperform the naive slot-based scheduling policy. As stated by Ghodsi, et al., the DRF policy is able to better utilize cluster resources and reduce job completion time. However, we do not use a simulation-based method for modeling interference between jobs in this work - exploring this further is an interesting avenue for future research.

We did find that reproducing this paper was more difficult than expected given the lack of details about the experiments run in the paper. We did not have access to the Facebook trace and the paper provided minimal details about the types of jobs included in the trace. As a result, we had to speculate about what a realistic trace would look like. The authors also did not provide many details about the baseline slot-based policy, which made certain behavior hard to interpret (for example, the effects of node over-subscription).

4 LITERATURE REVIEW

In this section we examine work in the cluster scheduling domain that appeared both before and after DRF to understand how the scheduling problem and its corresponding solutions have evolved over time.

4.1 Related Work

4.1.1 Quincy fair scheduling. The Quincy [4] scheduling policy improves upon previous queue-based approaches by modeling the scheduling problem as a flow network. This enables Quincy to model data transfer costs associated with each scheduling decision. Quincy then uses a min-cost flow solver to determine the optimal schedule. Jobs submitted to Quincy are subject to preemption, which enables the scheduling algorithm to prioritize the execution of less-serviced jobs (thereby improving fairness). Quincy's evaluation demonstrates that this flow-based approach is able to reduce network utilization and increase data locality compared to previous queue-based techniques.

4.1.2 Delay scheduling in Hadoop Fair Scheduler (HFS). Delay scheduling [9] also aims to improve data locality and fairness for cluster jobs. However, unlike Quincy, delay scheduling moves away from a preemptive strategy as preemption adds complexity and can unnecessarily waste computation. Instead, the delay scheduler purposefully waits to schedule tasks when a slot with the appropriate data is unavailable. This strategy takes advantage of the insight that scheduling a particular task on the first available slot may result in poor data locality, but with high probability a slot with the required data will become available in a short period of time. This intuition is validated empirically as delay scheduling is able to maintain fairness while significantly improving throughput compared to the default Hadoop scheduler at the time. DRF then built upon delay scheduling by taking into account diverse resource requirements.

4.1.3 Tetris multi-resource packing. In contrast to DRF, Tetris [2] breaks the assumption that the most fair schedule is the one that will most reduce average job completion time. Instead, Tetris generalizes the fairness parameter such that the cluster administrator can minimally relax the fairness constraint while greatly improving throughput. Tetris also differs from DRF in that it aims to *pack* tasks to nodes to improve resource utilization. While multi-dimensional bin

packing is a known APX-hard problem, Tetris is able to *learn* heuristic parameters online to make computing an allocation computationally tractable.

4.1.4 Graphene dependency-aware packing. The authors of Graphene [3] make the argument that schedulers making decisions in an online fashion must necessarily be simple and driven by heuristics in order to compute allocations in real time. The authors also note that modern data processing jobs have complex dependency structures and schedulers like Tetris do not incorporate this information. The Graphene scheduler addresses these issues by dividing the scheduling problem into (1) an offline phase which computes the optimal schedule for a single DAG and (2) an online phase that ensures currently running jobs adhere to their optimal schedules. The offline scheduler is able to optimize for *troublesome* tasks that have unusual behavior or respond poorly to packing.

4.1.5 Reinforcement Learning in Decima. Given the recent success of machine learning methods, and in particular reinforcement learning (RL), when applied to systems problems [6, 7], it is no surprise that RL is able to improve cluster scheduling, as well. Decima [5] removes the need for human-designed scheduling algorithms and heuristics and instead uses RL to learn a policy that directly optimizes for the specified objective (e.g. minimizing average job completion time). Using a novel RL state/action space formulation designed for this problem domain, Decima is able to outperform both Tetris and Graphene on production workloads.

4.2 Discussion

Our literature review illustrates that although DRF was state-of-the-art for its time, it could have modified its assumptions to improve its overall performance. The Tetris paper, for example, relaxes the fairness constraint and the assumption of task isolation on machines by introducing packing. Tetris also helps demonstrate the impact of online learning to aid scheduling decisions. Neither DRF nor Tetris consider task dependencies; the Graphene paper shows that including this information in the scheduling algorithm can greatly improve performance. Finally the Decima paper shows that a purely learned approach can be greatly effective. We believe that the overall trend of cluster scheduling research is therefore trending to a space that effectively combines empirically verified heuristics and online learning to optimize for the common case while also being flexible when faced with complex task patterns.

5 CONCLUSION

We reproduced several results from the Dominant Resource Fairness paper and showed that this cluster scheduling policy clearly outperformed the naive slot-based fairness policy that was commonly used at the time DRF was published. We also analyzed cluster scheduling policies that succeeded DRF and identified how the ideas of DRF have been improved over time. Overall we conclude that while the analysis provided in the DRF paper was correct at the time, cluster scheduling research as a whole has trended away from a fixed scheduling policy such as DRF towards a combination of human-designed heuristics and online learning.

REFERENCES

[1] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types.. In *Nsdi*, Vol. 11. 24–24.

[2] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2015. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 455–466.

[3] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. {GRAPHENE}: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 81–97.

[4] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. 2009. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 261–276.

[5] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2018. Learning scheduling algorithms for data processing clusters. *arXiv preprint arXiv:1810.01963* (2018).

[6] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V Le, and Jeff Dean. 2018. A hierarchical model for device placement. (2018).

[7] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2430–2439.

[8] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 18.

[9] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*. ACM, 265–278.