Summary of "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types"
Andrew Levin (anlevin), William Schmitt (schmittw), Zhao Zhang (jackzh)

Problem and Motivation

With the advent of warehouse scale computing, many algorithms have been developed to efficiently and fairly allocate resources to the various users running tasks in the same datacenter. Many of these algorithms attempt to distribute resources via the concept of *max-min fairness*. This approach aims to maximize the minimum allocation that a user receives. Additionally, certain tasks may be deemed more important than others, so *max-min fairness* has been adjusted to *weighted max-min fairness*, where the resources distributed to each user are proportional to that user's weight.

Schedulers for Hadoop and Dryad use *weighted max-min fairness* by allocating resources using fixed-size partitions of nodes. These partitions are referred to as *slots*, and they each come with identical amounts of CPU, memory, and other resources. Unfortunately, this strategy does not allow users to receive resources specific to their task needs. Some tasks are memory intensive while others need processing power. By only divvying up resources at the *slot* level, resources cannot be distributed efficiently. Memory intensive tasks will end up with more CPUs than they need, and processing intensive tasks will end up with more memory than they need. Cloud computing with multi-core processors has increased the need to allocate resources based on heterogeneous user demand.

Because cluster compute time is expensive and highly limited, ensuring efficient allocation of resources is extremely important from an economic perspective. Altering the number of slots cannot ensure proper distribution. Lowering the *slot* count by giving each slot more resources could lead to underutilization as small tasks may run and not use many of the resources in their provided *slots*. Increasing the *slot* count by reducing the number of resources per *slot* could cause large tasks to share nodes and regularly swap out. For example, launching 4 large tasks on a 6 GB node where each task needs 2 GB will lead to inefficient swapping. The only way to ensure high efficiency is by accounting for heterogeneous user demands when distributing resources.

Hypothesis

This paper proposes that the best means of distributing resources is a solution of designed for heterogeneous demands. The solution is called *Dominant Resource First* or *DRF*. To reach this solution, the researchers identified four criteria that any algorithm distributing multiple resources for heterogeneous demands should meet. These properties then help guide the development of *DRF*. Firstly, each user should prefer sharing the cluster's resources than receiving a *slot* or partition of the cluster. Next, there should not be any incentive for a user to lie about its resource demands. Users will seek to cheat, and it is common for users to attempt to manipulate schedulers by lying about their true demand for resources. Additionally, no user should prefer the allocation of any other user. Lastly, it should be impossible to to increase the resources of one user without decreasing the resources of another user. *DRF* is designed to satisfy these four criteria - and a few others - and it does so while efficiently distributing resources based on heterogeneous user demands. These properties make it the best method for distributing resources.

Solution Overview

1. The concept of dominant resource:
    a. Dominant share: the maximum share of all the resources allocated to the user. Essentially it is the percentage calculated by dividing the amount of resource required by the total amount the system provide for that resource.
    b. Dominant resource: the resource allocated to one user with dominant share is called this user's dominant resource.
2. Demand vectors and the DRF algorithm
    a. Consider a computational model with n users and m resources. Each user runs individual tasks and each task is characterized by a demand vector. A demand vector specifies the amount of resources required by one task.
    b. DRF simply applies max-min fairness across users' dominant shares. I.e. Maximizing the smallest dominant share in the system, then the second-smallest, and so on. Basically, the user with the smallest dominant share gets the highest priority to be allocated resources by the system, and if the system cannot satisfy the demand, we check the task with the second smallest dominant share, and so on.
2. Target properties of a resource scheduler:
    a. **Sharing-incentive**: Each user is be better off sharing resources, rather than using it on their own.
    b. **Strategy-proofness**: in DRF users cannot better themselves by lying about their true resource needs.
    c. DRF is **Pareto efficient** because increasing the allocation of user i's saturated resource will definitely decrease the allocations another user j, who's also using that saturated resource of i, thus decrease the dominant share of user j.
    d. DRF satisfies the **sharing incentive** and **bottleneck fairness** properties. This is because progressive filling increases the allocation of each users' dominant resource at the same rate. And if all the users have the same dominant resource, each user gets exactly 1/n of that resource.
    e. Every DRF allocation is **envy-free**. Because if user i envies user j, then user j must have a strictly higher share of every resource that i wants, so that i can run more tasks under j's allocation. This means user j's dominant share is strictly larger than user i's dominant share, which contradicts the properties of progressive filling.
    f. **Population monotonicity**: (Given strictly positive demand vectors) in DRF when users leave the system and relinquish their resources, none of the allocations of the other users decrease.
    g. DRF does not satisfy **resource monotonicity**. DRF does not guarantee that resources allocation per user is monotonic because this directly conflicts with Pareto efficiency. Since you cannot have both properties in a scheduler, DRF is designed for Pareto efficiency because adding new resources to a cluster is a rare event.
3. Overview of performance against previous resource schedulers:
   The paper have evaluated DRF by implementing it in the Mesos resource manager, and

comparisons from the experimental results between DRF and alternative allocation policies (slot-based fair scheduling and fair sharing applied only to a single resource) shown that DRF can lead to better overall performance, either with respect to jobs completions or average response times. It is also analyzed and proved that allocation policies like Asset Fairness and Competitive Equilibrium from Equal Incomes (CEEI) violate at least one of the 4 desired properties.

Limitations and Possible Improvements

One limitation of DRF is that in a workload consisting of many small jobs, DRF can experience a drop in performance. DRF achieves resource fairness at an instantaneous granularity. However, As shown by Carbyne[1], seeking to create a long term fairness rather than instantaneous fairness can lead to an overall speed up in performance when a workload contains many small jobs (or jobs that can easily be reshuffled).

It should be noted that many papers that seek to solve the multiple-resource allocation problem have traded the strategy-proofness in DRF to achieve theoretically better performance. Papers such as Tetris[2], HUG[3] are examples of this.

The paper proposes a the concept of “weighted dominant resource fairness”: the ability to assign weights to different jobs; however, it is unclear which properties weighted DRF maintains. A more formal discussion of this or experimental analysis of this is not critical to the paper or the core proposition, but greatly improve the discussion of weighted DRF.

While sound and convincing, the experimental results could be marginally improved. Experiments on the large and small cluster are done . While a reader can likely assume these are MapReduce jobs, it would have improved the quality of the paper if a standardized benchmark was used for future reproducibility.

Summary of Class Discussion

In class we discussed that the reason DRF is an influential paper because it is the first scheduler to satisfy seven of the eight proposed properties of an ideal scheduler and it considers a variety of factors from across computer science and economics. The importance of strategy-proofness was a key part of this discussion. We discussed that in scenarios without an inherent pricing scheme, DRF is key to achieving maximum resource efficiency not only because it prevents users from cheating but because users often do not know their true requirements and DRF will seek to minimize the overconsumption.

[1] “Altruistic Scheduling in Multi-Resource Clusters” Grandl, Chowdhury, Akella, Ananthananaryanan.
[2] “Multi-Resource Packing for Cluster Schedulers” Grandl, Ananthananaryanan, Kandula, Rao, Akella.
[3] “HUG: Multi-Resource Fairness for Correlated and Fairness Demands” Chowdhury, Liu, Ghodsi, Stoica.