Exploring the Fairness and Resource Distribution in an Apache Mesos Environment

Pankaj Saha,Angel Beltre, and Madhusudhan Govindaraju

Cloud and Big Data Laboratory, State University of New York (SUNY) at Binghamton
{psaha4, abeltre1, mgovinda}@binghamton.edu

***Abstract*—Apache Mesos, a cluster-wide resource manager, is widely deployed in massive scale at several Clouds and Data Centers. Mesos aims to provide high cluster utilization via fine grained resource co-scheduling and resource fairness among multiple users through Dominant Resource Fairness (DRF) based allocation. DRF takes into account different resource types (CPU, Memory, Disk I/O) requested by each application and determines the share of each cluster resource that could be allocated to the applications. Mesos has adopted a two-level scheduling policy: (1) DRF to allocate resources to competing frameworks and (2) task level scheduling by each framework for the resources allocated during the previous step. We have conducted experiments in a local Mesos cluster when used with frameworks such as Apache Aurora, Marathon, and our own framework Scylla, to study resource fairness and cluster utilization. Experimental results show how informed decision regarding second level scheduling policy of frameworks and attributes like offer holding period, offer refusal cycle and task arrival rate can reduce unfair resource distribution. Bin-Packing scheduling policy on Scylla with Marathon can reduce unfair allocation from 38% to 3%. By reducing unused free resources in offers we bring down the unfairness from to 90% to 28%. We also show the effect of task arrival rate to reduce the unfairness from 23% to 7%.** [1]

***Keywords*—**Apache Mesos, Dominant Resource Fairness (DRF)

I. Introduction

Widely known fair resource sharing policies such as max-min fairness and the generalized variation, weighted max-min fairness, are designed to provide a `fair share guarantee` [1]. However, these only work satisfactorily when a single resource type, such as CPU or memory, is taken into account. In data centers and clouds, where applications could be co-scheduled on the same physical nodes, resource fairness needs to extend to multiple resource types such as memory, disk I/O, and network bandwidth. The Dominant Resource Fairness (DRF) [2] algorithm was introduced to address this requirement for resource management of large-clusters and cloud environments.

The key tenant of DRF is that it takes into account different resource types (CPU, Memory, Disk I/O) requested by each application and determines the share of each cluster resource that could be allocated to the applications. DRF has been adopted by Apache Mesos [3], a widely used cluster-wide operating system that can efficiently manage very large clusters. Mesos is estimated to seamlessly scale to more than 10K nodes in some commercial settings [4].

[1]This work was supported in part by NSF grant OAC-1740263.

Mesos pools all the resources in a cluster and allows fine-grained resource sharing by allowing and enforcing multiple applications (called Mesos frameworks) to co-schedule their tasks on VMs/nodes. Mesos employs DRF to allocate resources to frameworks, and then the frameworks use scheduling algorithms to schedule tasks within the allocated resources. The frameworks that are widely deployed to work in concert with Apache Mesos are Apache Aurora [5] for long-running services, Mesosphere Marathon [6] for container orchestration, and Chronos [7] for cron jobs. In previous work, we developed a Mesos framework, Scylla [8], for MPI based HPC jobs.

While DRF is well intentioned, there are several use cases where the Mesos framework's internal scheduling policy, and attribute settings that govern interaction with DRF, prevent it from meeting the desired fairness objectives. In this paper, we have identified a few key attributes in a framework that affect access to a fair share of resources, when used in an Apache Mesos cluster. These include interaction with the Mesos resource offer cycles, offer holding period, task arrival rate, and task duration. We provide suggestions for how cluster administrators can control these attributes to ensure fair distribution of resources across all users of a cluster.

We assume that cluster managers and framework developers will use `off-the-shelf` Apache Mesos and its DRF modules, as it has the advantage of receiving support from service providers and the developer community, and can leverage seamless upgrades to the core Mesos tools. Modifying the DRF implementation within Mesos requires cluster managers and framework developers to maintain custom versions and manually keep track and incorporate the patches to Mesos and DRF modules.

The key contributions of this paper are the following:

- We have identified the differences between the well-known DRF algorithm and its variation that is available with the widely used Apache Mesos distribution.
- We have studied and analyzed the behavior of an Apache Mesos based cluster and determined the key attributes that control the resource distribution among multiple users.
- We have developed recommendations for cluster administrators for configuring key attributes to significantly improve resource distribution among all the frameworks, for varying workloads.

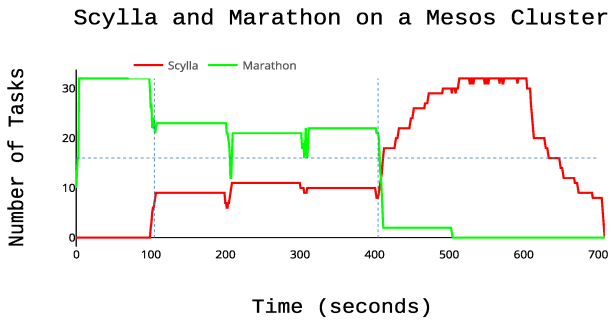

Figure 1. *Unfair Distribution: Scylla and Marathon competing for resources in a Mesos cluster. Marathon gets significantly more resources and is thus able to launch several more tasks than Scylla. Scylla's tasks face long wait times due to unfair distribution.*

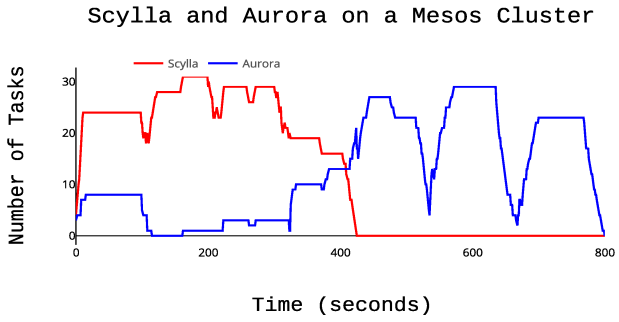

Figure 2. *Starvation: Scylla and Aurora competing for resources in a Mesos cluster. Scylla gets significantly more resources in the beginning. Aurora is able to launch its tasks only after Scylla is done launching all its tasks. Aurora's tasks are starving for resources when launched along with Scylla.*

II. Motivation

Figure 1 and 2 show how the presence of a pair of frameworks can result in an unfair resource distribution, when the frameworks are launched without DRF and Mesos awareness. The frameworks have been given 100 tasks, each with requirements of 1 CPU and 1 GB of RAM. The two frameworks are launched on a cluster with 32 CPUs and 64 GB RAM. In this experiment, as all tasks are identical in terms of resource demands, the number of tasks per second shows resource distribution across the frameworks. We have observed, over repeated runs, that with the default configuration of Marathon, Aurora, and Scylla, the resource distribution never converges to a fair share distribution. Either Aurora (in Figure 2) or Scylla (in Figure 1) faces starvation, and is only able to launch tasks after Scylla does not have any pending tasks. Even though the Mesos Master tries to distribute the resource in a fair way, the individual framework's configurations and scheduling policy affects the overall distribution.

Quantifying Unfairness: We measure the unfairness U_A to framework A by using the following formula.

$$U_A = \left(\frac{Area_{i,j}\ by\ framework_A}{Area_{i,j}\ by\ fair\ graph}\right) * 100$$

$Area_{i,j}$ is the area under the curve from point i to j

In Figure 1, the horizontal dotted line is the fair distribution threshold and the two vertical dotted lines are the start and end points where fairness is not achieved. The graph outside the starting and ending points are showing usage while either one framework has not started launching any tasks or done with launching all the tasks. In Figure 1, Scylla's fair share is reduced by 38%, whereas in Figure 2 Aurora's fair share is reduced by 67%.

III. Background

A. Apache Mesos Architecture

Apache Mesos is composed of three primary components (1) Mesos Master, (2) Mesos Agent, and (3) Mesos Framework. The Mesos Master manages resources during resource negotiation between the Mesos Agents and Mesos frameworks. Mesos Agents are responsible for executing requested tasks with available resources. All the available free resources from a single Mesos agent is included in a resource offer. Mesos frameworks make their own scheduling decisions to map one or multiple tasks to the offers allocated to them.

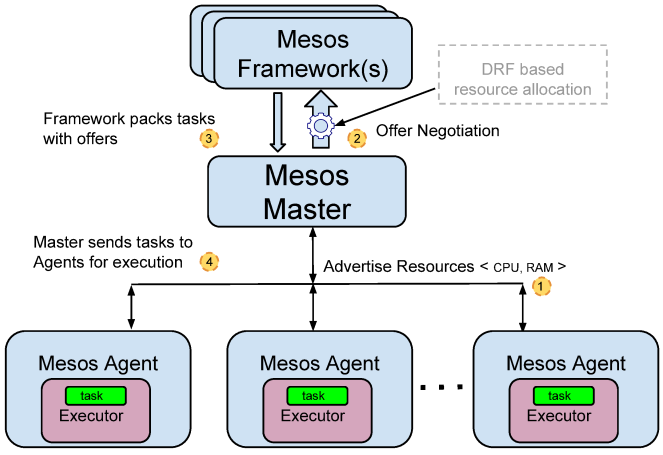

Figure 3. *Apache Mesos Architecture: This diagram shows the components of an Apache Mesos cluster and the steps involved in allocation of resources to frameworks*

In step 1, Mesos Agents periodically advertise to the Mesos Master regarding the available free resources (e.g. CPU, RAM, disk, I/O), which can be utilized for launching tasks. In step 2, Mesos Master's allocation module starts offering the available resources to active frameworks. During each allocation cycle, Mesos Master sorts the frameworks based on the DRF algorithm and the framework with lowest dominant share is the first to receive the offer. After resource allocation, an individual framework can decide whether to accept or decline an offer. The declined offer goes back to the resource pool and will be offered to other frameworks in the next allocation cycle. In step 3, if the offered resources satisfy a framework's resource demands, the framework creates a task list against offers. Based on the individual framework's policy, one or multiple tasks can be packed against a single resource offer. In step 4, Mesos Master launches them on the chosen agents and if the required resources exceed the available resources in

the offer, Mesos Master responds with an error. Otherwise, it sends a set of tasks to the individual agents corresponding to the offer.

B. Mesos Framework

A framework communicates with the Mesos Master for resource negotiation, and upon receiving the desired resources it executes its tasks on Mesos Agents. A Mesos Framework has two key modules: (1) Framework Scheduler and (2) Framework Executor.

1) Framework Scheduler: It is responsible for resource negotiations. The Mesos master decides how many offers will be given to each framework, but the framework's scheduling policy determines which offers will be picked among the available offers. Once offers are accepted, the framework's scheduler creates a map of tasks with the offers and then informs the Mesos Master to launch the tasks on the Mesos Agents associated with the offer. Apache Mesos implements a list of filters corresponding to each registered framework. After a framework rejects or partially uses an offer, it can prevent the same resources from being allocated to the framework by temporarily placing that agent in the filter list. The framework owner can configure the duration an agent should stay in the filter list. During that wait period, resources from the same agent are not offered. This wait time is known as `Offer Refusal` and it is configured in units of seconds.

2) Framework Executor: It is a process that resides on each Mesos Agent node and it is contacted whenever an agent node accepts tasks from frameworks.

C. Framework's Scheduling Policy

Apache Mesos' allocation module allocates resources to available frameworks based on the DRF fairness policy. The resources are listed in an `offer` that has details of the agent node and the share of each resource type that has been allocated. Each framework scheduler can implement its custom scheduling policy. Typically, a framework has a queue of tasks to schedule, and it uses a policy to decide how to pack tasks into offers. For example, `Bin Packing` and `First Fit` are two commonly used scheduling policies.

IV. Dominant Resource Fairness

A. How DRF works

Apache Mesos provides two-level scheduling for resource allocation to frameworks. The Mesos Master's allocation module decides the amount of resources that will be offered to each framework during every DRF offer cycle. The dominant resource of a framework is defined as the resource type that is used the most and is computed in terms of percentage of the overall availability of that resource. To calculate the dominant share S_i of user u_i, DRF uses the following formula:

$$S_i = max_{j=1}^{m}(\frac{u_{i,j}}{r_j})$$

$m =$ available types of resources

$r_j =$ total available resources of type j

$u_{i,j} =$ amount of resource of type j, being use by user u_i

Apache Mesos uses the DRF algorithm to decide the first level of resource distribution among the frameworks. In the second level of scheduling, a framework can choose which offer to accept and which one to decline. This second level of scheduling is pluggable and can be varied based on the requirements of the cluster. In each offer cycle, the Mesos Master tries to allocate resources through DRF to the framework that has the smallest dominant share. Then, it proceeds to offer to the second smallest share, and so on, until all the resources have been allocated. The DRF allocation module is not exercised in an environment where only one framework is in use. However, when multiple frameworks are competing, which is a common use case in very large clusters and data centers, Mesos uses DRF to attempt a fair allocation of resources based on each frameworks' current demands and usage.

B. DRF - Implementation in Mesos

Once resources are allocated to a framework, it can choose which offers to accept and which ones to reject. The DRF implementation within Apache Mesos does not exactly follow the original algorithm.

1) Single Node vs Pool of Resources in Cluster: The classical DRF algorithm [9] is designed to allocate resources from a single node to competing users. However, Mesos pools resources from a heterogeneous set of nodes and presents a single view of the cluster-wide resources. After each offer has been allocated, which could be for resources spread across nodes, the Mesos Master recalculates the dominant share of all the users for allocating the next offer available in the cluster. The Master keeps allocating all the offers as long as resources are available in the cluster. Once all the available offers have been allocated and accepted, one cycle of allocation is considered to be completed.

2) Resource Demands from Users: The Mesos Master's allocation module does not consider any demands from users. In the classical DRF algorithm, a demand (D_i) of user (u_i) is considered as a vector $D_i =< d_{i,1}, d_{i,2}, ..., d_{i,m} >$ where 'm' is the number of resources in the cluster. A user receives resource offer as a multiple of the demand vector – $Offer_i \implies D_i * n =< d_{i,1} * n, d_{i,2} * n, ..., d_{i,m} * n >$ where 'n' is the multiplication factor. This $Offer_i$ is capable of launching 'n' tasks from user u_i. Further, DRF assumes that all the tasks, received form a user, have identical resource demands.

However, unlike the classical DRF, Mesos Master allocates resources to users only based on the dominant share and offers all the available resources in an agent node. It allows users to reject part of the offer, or the entire offer, based on the requirement.

C. DRF-based Resource Allocation

Apache Mesos Master offers resources to frameworks during each cycle of resource distribution and the framework with minimum dominant share is served first. After the framework uses a portion of the offered resources, the rejected and unused portion of the offer, if any, goes back to the resource pool and

is eligible for allocation in the next cycle. After a resource is assigned to a framework, the Mesos Master resets the priority of the available frameworks based on their dominant share. A framework with the lowest dominant share holds the highest priority.

This resource allocation cycle runs against each unallocated offer in the cluster. For each unallocated offer, picked randomly from the list of offers, the allocation module finds the framework with the lowest dominant share. If the offer is in the filtered list of offers at that point of time, then the allocation module picks the next available framework in the DRF-sorted list of frameworks. Otherwise, it assign the resources to the framework.

Algorithm 1 Resource Allocation Cycle in Mesos

```
for each agent in the randomSorted agents do
  for each role in drfSorted roles do
    for each framework in drfSorted frameworks do
      if framework already filtered these resources then
        continue and skip the current framework
      else
        allocate the resources to the framework
      end if
    end for
  end for
end for
```

After each DRF cycle of resource allocation by Mesos Master, individual frameworks can accept or reject offers based on the 2nd level scheduling policy and more specific resource constraints. For example, a "Bin Packing" algorithm will utilize more offers than a "First Fit" and "One Task per Cycle" task allocation policy.

So in a cycle, if Mesos Master allocates all the resources to a Framework-A, which uses "One Task Per Cycle" policy, then the majority of the allocated offers will be rejected and may be allocated to Framework-B, which uses the Bin-Packing policy to map tasks to offers. To counter this resource distribution behavior, Framework-A needs more cycles to allocate more tasks than Framework-B. User level task allocation is one of the key attributes that drives a cluster towards unfair resource allocation. Other factors are (1) Refuse offer cycle, (2) Resource holding period, (3) Task completion rate, and (4) Task arrival rate. We discuss how these attributes contribute towards the fairness of the overall resource distribution.

1) **Refuse Offer Seconds:** This attribute defines the number of seconds for which resources from an agent cannot be offered again after the resources from the same agent were rejected or partially used.
2) **Offer Holding Period:** A few frameworks hold resources for a specified period to make better scheduling decisions. An offer can be terminated from a framework when either a task is launched with the offer or the offer holding period is over. Mesos Master can also set the offer timeout and reclaim after the timer expires If no timeout period is set, then the framework can hold an offer as long as it wants. A longer offer holding time decreases the resource utilization and can make other frameworks starve.
3) **Task Duration:** Long-running tasks block resources for a longer period and can make other users starve for resources. Apache Mesos does not kill tasks once they are started; it waits until the tasks finish even though other users may have more tasks to run.
4) **Task Arrival Rate:** Frameworks keep on launching tasks as long as the Mesos Master offers resources to them and there are tasks waiting in the queue. So, it is beneficial to frameworks that initiate tasks at a faster rate from a queue of tasks. A framework that launches tasks at a slower speed may face starvation because the resources are being blocked by the frameworks that are launching tasks at a higher rate.

V. Evaluation and Experimental Results

We ran experiments to determine the effect of various framework configurations and attributes on resource distribution and fairness. We used two off-the-shelf popular Mesos frameworks, Apache Aurora and Marathon, along with our MPI framework Scylla [8] to study and analyze the experimental results. Our experimental cluster has four nodes with a pool of 32 CPUs and 64 GBs of RAM. All tasks sent to the frameworks are identical in terms of resource requirements but in some experiments they differ in runtime.

Software	Version
Ubuntu	Ubuntu 16.04.2 LTS (Xenial)
Apache Aurora	17.06.0-ce
Marathon	1.4.0
Apache Mesos	1.3.0

Table 1. *Software Stack and Version*

A. DRF based fairness on a multi-user cluster

In this experiment, we ran two instances of Scylla (Scylla-A and Scylla-B), each with a queue of 100 tasks. Each task required $\langle 1CPU, 1GB\ memory \rangle$ and runtime of 100 seconds. We first launched Scylla-A and waited for all its tasks to be launched on the cluster before launching Scylla-B.

We can observe that after some fluctuations, the resource distribution is fair and both the frameworks are using close to 1/n of the cluster resources (which is 50% in this case). In the cluster, for the requested configurations by the frameworks, at most 32 tasks can be launched. The fair share for each framework is 32 tasks. Figure 4 shows that each framework is running 10 to 20 tasks. This setup achieves fair distribution compared to the results shown in Figures 1 and 2.

The Refuse Offer seconds attribute of one framework increases the opportunity for other frameworks to use resources that were rejected or partially used by a framework. So, to achieve better allocation, we ran the same experiment as Figure 4 but changed the Refusal Offer seconds by gradually increasing it from zero to five seconds. In Figure 5, we can observe that both the frameworks are executing around 15-16 tasks for longer period time, which again maintains a better resource distribution compared to the previous experimental

Figure 4. *Moderate Distribution: Number of tasks running every second by each Scylla instances setup with similar configurations. This setup has a moderate resource distribution among the frameworks when offer refusal period for both the Scylla instances is set to none.*

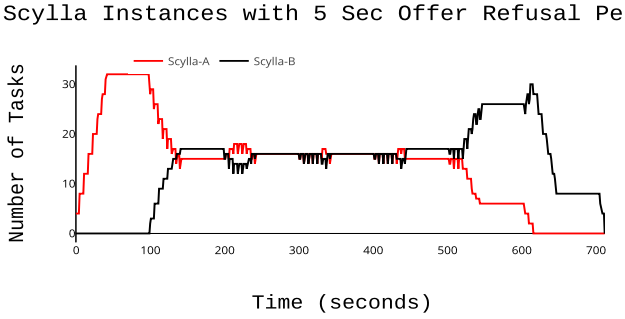

Figure 5. *Preferred Distribution: Number of tasks running every second by each Scylla instances setup with similar configurations. The refuse offer period is set to 5 seconds.*

setup shown in Figure 4. We continued the experiment to further increase the offer refusal seconds to 7 and 10 seconds, but did not see much improvement. So, in our experimental cluster, we kept the configuration of 5 seconds as the offer refusal period is optimal.

B. Fairness with Marathon and Scylla Frameworks

The second level scheduling policy can impact the clusters resource distribution and lead to unfair distribution of resources. For this experiment, we deployed Marathon and Scylla, wherein Scylla employs First-Fit scheduling policy. Both the frameworks were given a queue with 100 tasks, each requiring 1 CPU and 1 GB of RAM.

In Figure 1, we can observe how Marathon's greedy resource consumption policy consumes more resources and launches more tasks even though another framework was waiting for resources. Marathon used resources that exceeded the fair share, whereas Scylla received a smaller share. This greedy approach of Marathon caused a 38% reduction in fair resource allocation to Scylla, as shown in Figure 1. We changed the configuration and instantiated Scylla with Bin-Packing as second level scheduling policy. In Figure 6, we can see how each framework started using close to fair share of the cluster resources. Due to Scylla's Bin-Packing policy, it receives 5% more resources than the fair share limit, which

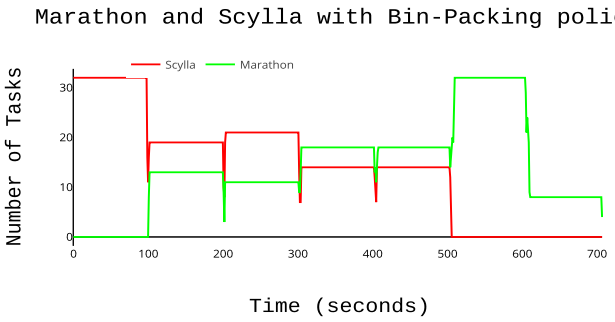

Figure 6. *Moderate Distribution: Marathon and Scylla are vying for resources in a Mesos cluster when Scylla is configured with Bin-Packing as the second level task allocation policy. This resource distribution is moderately better compared to the unfair distribution we noticed in Figure 1.*

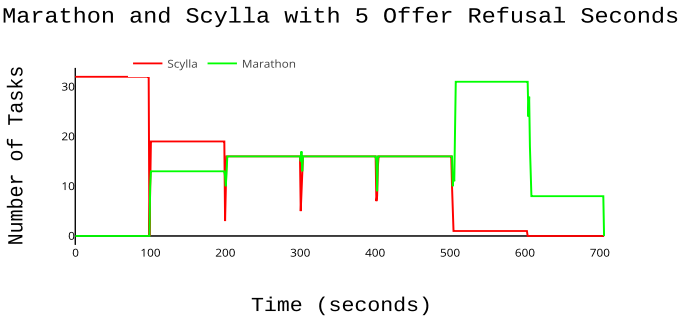

Figure 7. *Preferred Distribution: Marathon and Scylla are vying for resources in a Mesos cluster when Scylla is configured with Bin-Packing as second level task allocation policy and 5 second refuse offer period is set as a framework attribute of Scylla, for better resource distribution.*

is better than the loss of 38% fairness in allocation shown in Figure 1. This situation is further improved to 3% above the fair share limit by increasing the offer refusal period to 5 seconds, which was previously determined to be the optimal value for this cluster in Figure 7.

C. Fairness on Apache Aurora and Scylla based cluster

Figures 8, 9 and 10 show how we can gradually achieve better resource distribution with Apache Aurora and Scylla. With Apache Aurora, we faced some challenges in attaining fairness as it holds all the offered resources for a specified period. When a framework holds resources for a while, even if it does not launch any tasks, those held offers are weighed against the framework in computing the dominant share of the framework. Due to the increment of dominant share, Aurora's priority to receive further offers goes down, and another framework gets those offers.

To explain this scenario in more detail, consider the following example – a cluster with four nodes, each one containing the following resource configuration: 8 CPUs, 16GB of memory, and 32GB of disk space. Mesos Master will receive advertisement of offers from all the Agents in the following manner:

Table 2. *100% CPU and Memory resource is consumed*

CPU	Memory	Disk
100%	100%	2.5%

Table 3. *Resource allocation in presence of both frameworks*

Framework	CPU	Memory	Disk
Framework-A	87%	87%	2.8%
Framework-B	0%	0%	97.81%

- Offer 1 $\langle 8CPU, 16GB\ memory, 32000MB\ disk \rangle$
- Offer 2 $\langle 8CPU, 16GB\ memory, 32000MB\ disk \rangle$
- Offer 3 $\langle 8CPU, 16GB\ memory, 32000MB\ disk \rangle$
- Offer 4 $\langle 8CPU, 16GB\ memory, 32000MB\ disk \rangle$

Now consider Framework A launches 32 tasks, each $\langle 1CPU, 2GB\ memory, 100MB\ disk \rangle$, in the cluster before Framework B is started. Table 2 shows the current status of the cluster where we can see that it is 100% occupied by framework A, for CPU and memory resources, and exhibits a dominant share of 100%. As a result, any available resources should subsequently go to framework B. During this period there will be resource offers like $\langle 0CPU, 0GB\ memory, 31200MB\ disk \rangle$. Let us consider that we have framework B, which holds resources for a specified amount of time hoping to make better task allocation in its internal scheduling. During this offer holding period, let us say framework A completes 4 tasks and as a result a total offer of $\langle 4cpu, 8gb - mem, 400gb - disk \rangle$ will be freed. Mesos Master will then allocate the offers through next DRF cycle. Mesos Master will compute the resource share of each framework as shown in Table 3.

Now any framework that has a lower dominant share will get the opportunity to use these available offers before framework A acquires it. Framework B's CPU and memory share is 0%, and disk share is 97.81%. Even though Framework B is not launching any tasks, due to the disk resource on hold, Mesos Master will determine its dominant (disk) share to be 97.81%, which is higher than frameworks A's dominant share. So framework A gets the chance to use the available offers before framework B. This causes starvation for framework B until framework A is done with executing all the tasks.

To demonstrate with an experiment, we setup a cluster with the same resource configuration and used Scylla as Framework A. We deployed Apache Aurora as framework B, which holds resources for 5 minutes by default [10]. In Figure 8 we can see that Aurora is able to launch less number of tasks in the presence of Scylla as Aurora faces 89% fairness reduction in resource allocation. Scylla receives more CPU and memory resources and is able to launch all the tasks ahead of Aurora.

Table 4. *100% Disk Resource is Allocated*

CPU	Memory	Disk
100%	100%	100%

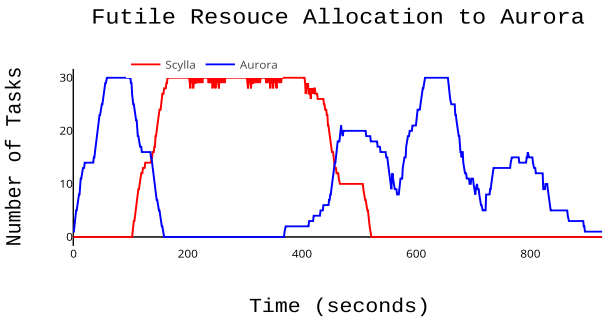

Figure 8. *Poor Distribution: Apache Aurora is launching a small number of tasks in the presence of Scylla. Entire cluster's resources are being used by Scylla to launch most of its tasks, which is leading to poor resource distribution.*

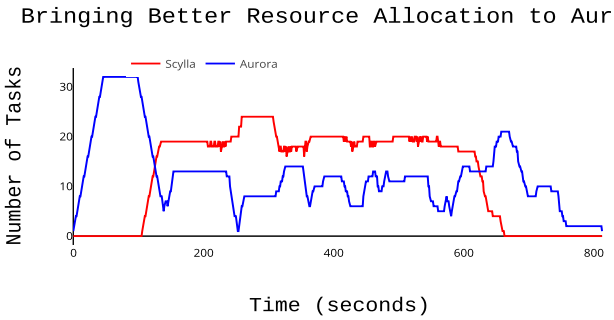

Figure 9. *Better Distribution: Resource distribution is improved between Apache Aurora and Scylla by addressing the problem due to Aurora's resource holding feature.*

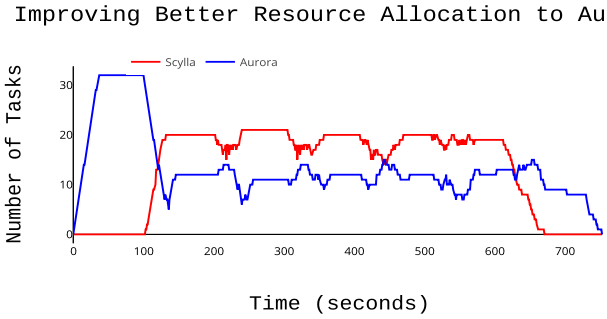

Figure 10. *Improved Distribution: Increased offer refusal period of Scylla improves resource distribution to Aurora.*

To improve resource distribution, we tried to reduce the free disk resource in the cluster. Instead of launching tasks with $\langle 1CPU, 2GB\ memory, 100MB\ disk \rangle$, we launched each task with $\langle 1CPU, 2GB\ memory, 4096MB\ disk \rangle$ to combine more disk resources in the task requirements. Now, at the beginning when Scylla is running 32 tasks, its resource share metric will be as shown in Table 4. As there exist no offers due to the unavailability of resources, Aurora will not receive offers with big disk space like in the previous case. As Aurora is not holding any resources, any freed up resources will be offered to it, which will bring a better resource distribution in the cluster.

Figure 9 shows a comparatively better resource distribution

where Aurora has 35% reduction in fairness. We have seen in previous experiments (see Figures 5 and 7) an increment of Refuse Offer seconds of a framework improves the chances of another framework to get relatively better resources. Figure 10 shows relatively better results than Figure 9 and unfairness is reduced to 28% for Aurora.

D. Impact of Idle Users on Resource Distribution

In this experiment, we study how the presence of idle frameworks in a Mesos cluster can cause low resource utilization. We launched a single instance of Scylla and recorded the time it takes to complete 100 tasks. We launched all the tasks in a fixed interval of two seconds throughout these experiments. Next, we increased the number of framework instances and observed how the makespan increases, in presence of other idle frameworks, due to the DRF resource allocation policy.

Figure 11. *Impact of Idle Frameworks: The required time to launch all the tasks of an active framework rises as the number of idle frameworks in the cluster increase. This could be addressed by increasing the offer refusal period of idle frameworks.*

Figure 11 shows how makespan increases as we increase the number of idle frameworks. From Figure 11A to Figure 11C, we increased the number of idle frameworks in the cluster and observed that the makespan doubles when the number of idle framework goes up to five.

DRF allocation favors the framework that has received less resource allocations. Thus, Mesos Master offers the resources to the idle framework in the current cluster environment even though these frameworks do not have any tasks to launch. Their dominant share is 0%. DRF fairness always prefers framework with the lower dominant share and this can cause starvation for a framework that has a pending list of tasks to launch. To avoid starvation of an active framework, we can increase the offer refusal duration for the idle frameworks that do not have pending tasks. When a task appears on those idle framework, the filter can be removed for it to accept offers for the pending tasks. To study this case, we increased the offer refuse duration of the idle frameworks from five to 10 seconds, and reduced the active frameworks' offer refuse duration to two seconds. We can observe in Figures 11D and 11E how this change improves the makespan. Any further reduction in the offer refusal period for the active framework, however, did not produce further improvements.

E. Long running tasks towards unfair distribution

Mesos lets a framework exceed its fair share if the lower share framework does not want the resources. As Mesos does not revoke resources until a task is completed, a framework may have to wait for those resources to be released before it can get its share. Thus, improvement in utilization comes at the cost of not providing a guarantee that a user can get its fair share without waiting. Mesos allows the use of a `quota`, for role-based static and dynamic reservation of resources, to ensure that a framework will always get its share no matter what is being requested by other frameworks. However, the use of quota can lead to underutilization of the cluster resources if a reserved resource is not being used by a framework.

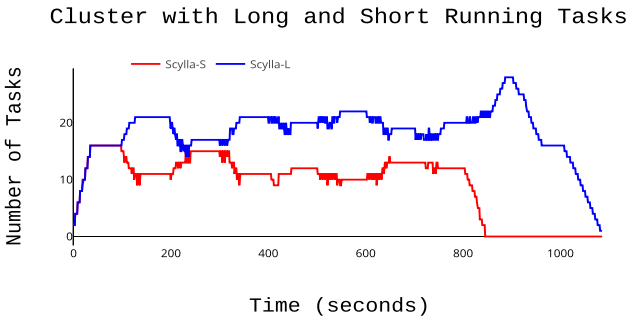

Figure 12. *Poor Distribution: Resource distribution when two identical frameworks launch tasks of different durations but at the same launching rate.*

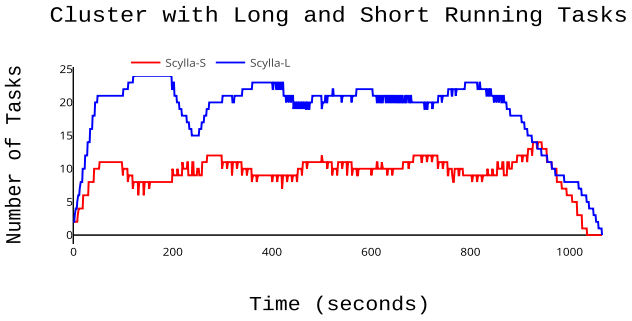

Figure 13. *Worse Distribution: Cluster with long and short Resource distribution scenario when long running tasks are launched at a higher rate and short running tasks are launched at a slower rate.*

In this experiment, we setup two instances of the Scylla framework with similar configuration (Scylla-L and Scylla-S). Scylla-L launches long-running tasks whereas Scylla-S runs short-running tasks. Both the frameworks allocate tasks with "First-Fit" as the second level scheduling policy and receive tasks in an interval of 5 seconds. Short running tasks required 1CPU and 1GB of memory resources and run for 100 seconds. Whereas, long running tasks required similar amount

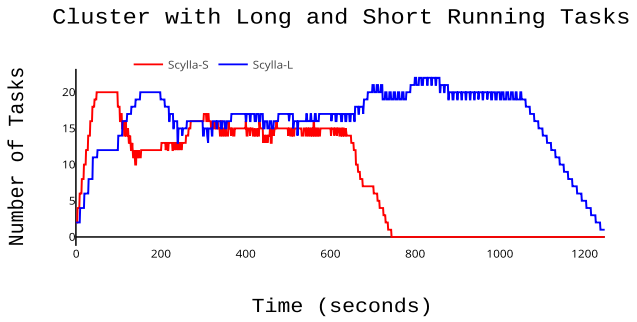

Figure 14. *Improved Distribution: Cluster with long and short running tasks. This figure illustrates an improved resource distribution when long running tasks are launched at a slow rate and short running tasks are launched at a high rate.*

of resources but run for 200 seconds. We launched a total of 100 short running tasks and only 50 long running tasks. As the runtime of long running tasks is double that of the short running tasks, it keeps the total runtime of both the types of tasks the same. Our experimental results in Figure 12 show how this experiment results in an unfair distribution in the cluster. Framework-S starves for a longer period in the Mesos cluster as it receives 23% unfair resource allocation.

To observe how this resource distribution is impacted by task arrival rate, we changed the frequency at which tasks appear in the queue for each framework. Figure 13 shows how resources are distributed among frameworks when Framework-L launches long-running tasks every 5 seconds and Framework-S launches tasks every 10 seconds. The lower task arrival rate of Framework-S results in 37% reduction in fairness. To improve the scenario, we interchanged the task arrival rate of both the frameworks. We see an improvement in the resource distribution shown in Figure 14 where the unfairness is reduced to 7% for Framework-S.

VI. Related Work

The work on evaluation of DRF is in its early stages.

Ghodi et al. [2] introduced DRF as a generalization of the well-known Max-Min traditional problem. Our work complements their research as we are evaluating how the implementation of DRF in Apache Mesos differs from the proposed DRF, and how framework policies affect resource distribution.

Dimopoulos et al. [11] show how big data frameworks (Hadoop, Spark, and Storm) hinder each other in a Mesos cluster under resource constraints. They compare the frameworks with different data sizes to see how performance varies with data volume.

Wang et al. [1] generalized DRF to work on multiple heterogeneous servers. Saha et al. [12] show how Apache Mesos can be integrated into a scientific cluster to leverage DRF based fair resource distribution. A Mesos framework, Scylla [8], was developed to orchestrate scientific MPI tasks on a Mesos cluster.

VII. Conclusion

- The DRF based allocation module may not provide enough resources to frameworks such as Apache Aurora, which holds on to resources instead of immediately using them. Framework specific attributes, like offer holding period, are critical and informed decisions based on the existence of other frameworks can reduce an unfair allocation from 90% to 28%.
- Frameworks can refuse to accept offers from the Mesos allocation module for a configurable period of time. This offer refusal period increases the opportunity for other frameworks to use the refused offer, and in turn leads to better resource distribution. For a change in offer refusal period from 0 to 5 seconds, the gains in fairness range can from 85% to 99%.
- The second level scheduling policy of a framework can incorporate greediness and make other frameworks starve for resources. In such cases, competing frameworks that allow change in their scheduling policy, to bin packing for example, can reduce the unfairness from 38% to just 3%.
- Once Mesos launches tasks, it does not terminate them even if the frameworks' dominant share exceeds the fair share limit of the cluster. This feature can lead other frameworks to starve. Increased arrival rate of short running tasks and decreased arrival rate of long-running tasks from two different frameworks respectively can reduce the unfair resource distribution from 23% to 7%.

References

[1] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 583–591.

[2] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, vol. 11, no. 2011, 2011, pp. 24–24.

[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," pp. 295–308, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972488

[4] "Mesos has been simulated to scale to 50,000 nodes, although it is not clear ho... — Hacker News." [Online]. Available: https://news.ycombinator.com/item?id=10228820

[5] "Apache Aurora." [Online]. Available: http://aurora.apache.org/

[6] "Marathon: A container orchestration platform for Mesos and DC/OS." [Online]. Available: https://mesosphere.github.io/marathon/

[7] "Chronos: Fault tolerant job scheduler for Mesos." [Online]. Available: https://mesos.github.io/chronos/

[8] P. Saha, A. Beltre, and M. Govindaraju, "Scylla: A Mesos Framework for Container Based MPI Jobs," in *MTAGS17: 10th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers*, Denver, 2017.

[9] "Dominant Resource Fairness: Fair Allocation of Heterogeneous Resources in Datacenters — EECS at UC Berkeley." [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-55.html

[10] "Aurora Scheuler Configuration." [Online]. Available: http://aurora.apache.org/documentation/latest/reference/scheduler-configuration/

[11] S. Dimopoulos, C. Krintz, and R. Wolski, "Performance Interference of Multi-tenant, Big Data Frameworks in Resource Constrained Private Clouds." [Online]. Available: https://www.cs.ucsb.edu/sites/cs.ucsb.edu/files/docs/reports/paper_6.pdf

[12] P. Saha, M. Govindaraju, S. Marru, and M. Pierce, "Integrating Apache Airavata with Docker, Marathon, and Mesos," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 1952–1959, 5 2016. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3708