2018 IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems

Scheduling Distributed Resources in Heterogeneous Private Clouds

George Kesidis, Yuquan Shan, Aman Jain, Bhuvan Urgaonkar
School of Elelctrical Engineering and Computer Science
Pennsylvania State University
University Park, PA
{gik2,yxs182,axj182,buu1}@psu.edu

Jalal Khamse-Ashari and Ioannis Lambadaris
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
{jalalkhamseashari,ioannis}@sce.carleton.ca

***Abstract*—We first consider the static problem of allocating resources to (*i.e.*, scheduling) multiple distributed application frameworks, possibly with different priorities and server preferences, in a private cloud with heterogeneous servers. Several fair scheduling mechanisms have been proposed for this purpose. We extend prior results on max-min fair (MMF) and proportional fair (PF) scheduling to this constrained multiresource and multiserver case for generic fair scheduling criteria. The task efficiencies (a metric related to proportional fairness) of max-min fair allocations found by progressive filling are compared by illustrative examples. In the second part of this paper, we consider the online problem (with framework churn) by implementing variants of these schedulers in Apache Mesos using progressive filling to dynamically approximate max-min fair allocations. We evaluate the implemented schedulers in terms of overall execution time of realistic distributed Spark workloads. Our experiments show that resource efficiency is improved and execution times are reduced when the scheduler is "server specific" or when it leverages characterized required resources of the workloads (when known).**

***Index Terms*—private cloud, scheduling, heterogeneity, progressive filling**

I. Introduction

We consider a cloud provider that needs to run multiple software application frameworks on its IT infrastructure. The cloud provider's infrastructure consists of multiple servers (slaves, workers) connected by a network. A server may be a physical machine or virtual machine (*e.g.*, an instance or a container). Each framework desires multiple IT resources (CPU, memory, network bandwidth, *etc.*) for each of its tasks. The provider's challenge then is to determine who should get how many resources from which servers. Our interest is in a *private* cloud setting wherein notions of fairness have often been used as the basis for this resource allocation problem.

Previously proposed fair schedulers include Dominant Resource Fairness (DRF) [12] extended to multiple servers[1], Task Share Fairness (TSF) [34], Per Server Dominant Share Fairness (PS-DSF) [17], [16], among others, *e.g.*, [5]. DRF is resource based, whereas TSF and "containerized" DRF [11] are task based[2]. In the following, we additionally consider variants of these schedulers that employ current residual (unreserved) capacities of the servers in the fairness criteria (somewhat similar to "best fit" variants [35]).

The contributions and organization of this paper is as follows. In Section II, we give background on the multiserver, multiresource scheduling problem. In Section III, we generalize static optimization problems whose solutions correspond to max-min fairness (MMF) and proportional fairness (PF) to multiple resources and multiple servers, the latter related to resource efficiency. In Section IV, an illustrative numerical example is given using progressive filling to compare the task efficiencies of different schedulers, including variants using residual/unreserved server resource capacities specified herein. In Section V, we consider online scheduling (potentially in the presence of framework churn) and describe our open-source prototypes of different schedulers implemented on Apache Mesos, a practical middleware platform to manage plural distributed application frameworks for a private cloud. We give the results of a comparative performance study (assessing total execution times) in a heterogeneous setting using realistic Spark workloads.

II. Background

Typically static problem formulations are considered under a variety of simplifying assumptions on framework behavior. It is assumed that frameworks congest all the available servers. That is, it is assumed that there is sufficient work to completely occupy at least one resource in every server. It is also assumed that the frameworks' required resources (presumably to achieve certain performance needs) are well characterized, *e.g.*, [9], [18], [20], [3], [25], [36], [7], [1], [19]. Frameworks are assumed to have linearly elastic resource demands in the following sense. Each task has a known requirement $d_{n,r}$ for the resource type r. Therefore, if $x_{n,i}$ were the number of tasks of framework n placed on server i, the framework would consume $x_{n,i}d_{n,r}$ amount of resource r on server i. $x_{n,i}$ may

This research was supported in part by NSF CNS 1526133, NSF CNS 1717571 and a Cisco Systems URP gift.

[1]DRF was originally defined for a single server in [12]. The multiple-server version, called DRFH in [35], [11], is also commonly called just DRF as done in Apache Mesos [13] and as we do herein also.

[2]Containerized DRF has a "sharing-incentive" property not possessed by DRF, and TSF possesses "strategy-proofness" and "envy-freeness" properties which are not possessed by containerized DRF [34]. Unlike DRF and TSF, PS-DSF is not necessarily Pareto optimal but is "bottleneck" fair. These properties are not addressed herein

2375-0227/18/$31.00 ©2018 IEEE
DOI 10.1109/MASCOTS.2018.00018

take on non-negative real values rather than being restricted to be non-negative integer valued[3]. Finally, frameworks may have different service priorities and server preference constraints (as in *e.g.*, service-quality constraints [37] or cache-affinity constraints), see also [34].

Note that in some settings (not considered herein), a goal is to minimize the number of servers to accommodate workloads with finite needs, again as in multidimensional bin-packing problems [4], [6], [8]. Such problem formulations are typically motivated by the desire to economize on energy. However, frequently cycling power to (booting up) servers may result in software errors and there are energy spikes associated with boot-up resulting in increased electricity costs [10].

Typically in existing papers, max-min fairness (MMF) with respect to a proposed criteria is specified assuming the aforementioned congested regime under the following (linear) capacity constraints:

$$\forall i, r, \ \sum_n x_{n,i} d_{n,r} \leq c_{i,r} \text{ s.t. } x_{n,i} > 0 \ \Rightarrow \ \delta_{n,i} = 1; \quad (1)$$

where $c_{i,r}$ is the amount of available resource r in server i, and frameworks n indicate preferences for servers i by $\delta_{n,i} \in \{0, 1\}$. Note that quantization (containerization) issues associated with workload resource demands are considered in [11].

MMF allocation may be expressed as the solution of a constrained centralized optimization problem. Alternatively, max-min fairness with respect to the proposed fairness criteria may be approximated by a greedy, iterative "progressive filling" allocation. The latter approach is often preferred because of the benefits this offers for online implementations. Moreover, progressive filling arguments can be used to establish other potentially desirable fairness properties of schedulers defined for private clouds [12], [17].

In this paper, we relate this task efficiency objective to "proportional" fairness (PF).

III. Mulitresource, multiserver MMF and PF

To generalize previous results on MMF and PF to multiple resource types on multiple servers, consider the following general-purpose fairness criterion for framework n,

$$U_n \ = \ \phi_n^{-1} \sum_i u_{n,i} x_{n,i}, \quad (2)$$

for scalars $u_{n,i} > 0$ and priorities $\phi_n > 0$. Under DRF [12], [35], frameworks n are selected using criterion

$$M_n \ = \ \frac{1}{\phi_n} x_n \max_r \frac{d_{n,r}}{\sum_j c_{j,r}}, \quad (3)$$

where $x_n = \sum_i x_{n,i}$. The PS-DSF criterion can be written as

$$K_{n,j} \ = \ \frac{\sum_i x_{n,i} d_{n,\rho(n,j)}}{\phi_n c_{j,\rho(n,j)}} \quad (4)$$

[3]With x integer valued, such problems belong to the class of combinatorial-optimization Multidimensional Knapsack Problems (MKPs), *e.g.*, [4], [26], [33], [6], which are NP-hard. They have been extensively studied, including relaxations to simplified problems that yield approximately optimal solutions, *e.g.*, by Integer Linear Programs solved by iterated/online means.

where bottleneck resource ρ is such that

$$d_{n,\rho(n,j)}/c_{j,\rho(n,j)} \ := \ \max_r d_{n,r}/c_{j,r} \ \text{ when } \delta_{n,j} = 1. \quad (5)$$

The feasibility conditions are (1).

Regarding fully booked resources in server i under allocations $x = \{x_{n,i}\}$, let

$$R_i \ := \ \{(x, r) \mid \sum_n x_{n,i} d_{n,r} = c_{i,r}\}.$$

Definition 1: A feasible allocation $\{x_{n,i}\}$ satisfying (1) is said to be U-MMF if:

$$U_\ell > U_m, \ x_{m,i} > 0, \text{ and } \exists r \text{ s.t.} \sum_n x_{n,i} d_{n,r} = c_{i,r}$$

implies that $x_{\ell,i} = 0$.

Note that if instead $x_{\ell,i} > 0$ in this definition, then $x_{\ell,i}$ can be reduced and $x_{m,i}$ increased to reduce $U_\ell - U_m$. Also, if $\{x_{n,i}\}$ is U-MMF and $x_{m,i}, x_{\ell,i} > 0$ for some server i then $U_m = U_\ell$. Consider the optimization problem

$$\max_x \sum_n \phi_n g(U_n) \text{ s.t. (1)}, \quad (6)$$

for strictly concave and increasing g with $g(0) = 0$.

Proposition 1: A solution $x = \{x_{n,i}\}$ of the optimization (6) s.t. (1) has at least one resource r fully booked in each server i. In addition, there is a unique U-MMF solution if also

$$\delta_{m,i} = 1 = \delta_{l,i} \text{ and } (x, r) \in R_i \ \Rightarrow \ d_{m,r} = d_{l,r}. \quad (7)$$

Note that for uniqueness, the proof given in [15] requires the strong assumption (7) that frameworks that can share servers have identical demand parameters d for fully utilized resources.

For weighted PF, consider the objective

$$\max_x \sum_n \phi_n g_a(x_n), \quad (8)$$

i.e., without dividing by ϕ_n in the argument of g_a [24]. For parameter $a > 0$ specifically take

$$g_a(X) \ = \ \begin{cases} \log(X) & \text{if } a = 1 \\ (1-a)^{-1} X^{1-a} & \text{else} \end{cases}$$

i.e., $g_a'(X) = 1/X^a$, again see [24]. Obviously, in the case of $a = 1$ ($g = \log$), whether the factor ϕ is in the argument of g is immaterial.

The following generalizes Lemma 2 of [24] on Proportional Fairness. See also the proportional-fairness/efficiency trade-off framework of [14] for a single server.

Proposition 2: A solution x^* of the optimization (8) s.t. (1) is uniquely (weighted) (ϕ, a) x-PF, *i.e.*, for any other feasible solution x,

$$\Phi(x, x^*) := \sum_n \phi_n \frac{x_n - x_n^*}{(x_n^*)^a} \ \leq \ 0. \quad (9)$$

The proof is given in [15]. A possible definition of the *efficiency* of a feasible allocation is

$$\sum_n \phi_n \sum_i x_{n,i} \ = \ \sum_n \phi_n x_n, \quad (10)$$

(corresponding to $a = 0$), *i.e.*, the weighted total number of tasks scheduled. The optimization of Proposition 2 with $a = 1$ gives an allocation x^* that is related to a task efficient allocation. Clearly, x^* satisfying (9) for all other allocations x with $a = 1$ does not necessarily maximize (10). This issue is analogous to estimating the mean of the ratio of positive random variables $\mathsf{E}(X/X^*)$ using the ratio of the means $\mathsf{E}X/\mathsf{E}X^*$, see *e.g.* p. 351 of [32] or (11) of [27] . For simplicity in the following, we use (10) instead of (9).

Note that the priority ϕ_n of framework n could factor its resource footprint $\{d_{n,r}\}_r$. Alternatively, the resource footprints of the frameworks can be explicitly incorporated into the main optimization objective via a fairness criterion. The proof of the following corollary is just as that of Proposition 2. Recall that the generic fairness criterion U_n (2) is a linear combination of $\{x_{n,i}\}_i$.

Corollary 1: A solution x^* of the optimization problem

$$\max_x \sum_n \phi_n \log(U_n) \quad \text{s.t.} \quad (1)$$

is uniquely $(\phi, 1)$ U-PF, *i.e.*, for any other feasible x,

$$\sum_n \phi_n \frac{U_n - U_n^*}{U_n^*} \quad \leq \quad 0.$$

Again, optimal $\{U_n^*\}$ would be unique but $x^* = \{x_{n,i}^*\}_{n,i}$ may not be.

Recall for DRF and PS-DSF, the $K_n = \sum_i K_{n,i}$ and M_n, respectively, are proportional to x_n. Thus, using $U_n = K_n$ or $U_n = M_n$ in Corollary 1 reduces to the result of Proposition 2 when $a = 1$.

IV. Illustrative numerical study of fair scheduling by progressive filling

In the following evaluation studies, we use *progressive filling* [2], [12], an online scheduling approach for approximating fair allocation under a certain fairness criterion. In progressive filling, resources are incrementally (taskwise) allocated to frameworks n in a greedy fashion: simply, the framework n with smallest fairness criterion U_n (or $U_{n,i}$), based on *existing* allocations $\{x_{n,i}\}_{n,i}$, will be allocated a resource increment $\{\varepsilon d_{n,i}\}_i$ for small[4] $\varepsilon > 0$. If a framework's resource demands cannot be accommodated with available resources, the framework with the next smallest fairness criterion will be allocated. Also, for brevity, we consider only cases with frameworks of equal priority ($\forall n, n',\ \phi_n = \phi_{n'}$) and without server-preference constraints (*i.e.*, $\delta_{n,i} \equiv 1$).

In this section, we consider the following typical example of our numerical study with two heterogeneous distributed application frameworks ($n = 1, 2$) having resource demands per unit workload:

$$d_{1,1} = 5,\ d_{1,2} = 1,\ d_{2,1} = 1,\ d_{2,2} = 5; \qquad (11)$$

[4]Typically $\varepsilon = 1$ when allocations x are measured in "tasks".

sched. \ (n,i)	(1,1)	(1,2)	(2,1)	(2,2)	total
DRF [12], [35]	6.55	4.69	4.69	6.55	22.48
TSF [34]	6.5	4.7	4.7	6.5	22.4
RRR-PS-DSF	19.44	1.15	1.07	19.42	41.08
BF-DRF [35]	20	2	0	19	41
PS-DSF [17]	19	0	2	20	41
rPS-DSF	19	2	2	19	42

TABLE I
WORKLOAD ALLOCATIONS $x_{n,i}$ FOR DIFFERENT SCHEDULERS UNDER PROGRESSIVE FILLING FOR ILLUSTRATIVE EXAMPLE WITH PARAMETERS (11) AND (12). AVERAGED VALUES OVER 200 TRIALS REPORTED FOR THE FIRST THREE SCHEDULERS OPERATING UNDER RRR SERVER SELECTION.

sched. \ (n,i)	(1,1)	(1,2)	(2,1)	(2,2)
DRF [12], [35]	2.31	0.46	0.46	2.31
TSF [34]	2.29	0.46	0.46	2.29
RRR-PS-DSF	0.59	0.99	1	0.49

TABLE II
SAMPLE STANDARD DEVIATION OF ALLOCATIONS $x_{n,i}$ FOR DIFFERENT SCHEDULERS UNDER RRR SERVER SELECTION WITH. AVERAGED VALUES OVER 200 TRIALS REPORTED.

and two heterogeneous servers ($i = 1, 2$) having two different resources with capacities:

$$c_{1,1} = 100,\ c_{1,2} = 30,\ c_{2,1} = 30,\ c_{2,2} = 100. \qquad (12)$$

For DRF and TSF, the servers i are chosen in round-robin fashion, where the server order is randomly permuted in each round; DRF under such randomized round-robin (RRR) server selection is the default Mesos scheduler, *cf.* next section. One can also formulate PS-DSF under RRR wherein RRR selects the server and the PS-DSF criterion only selects the framework for that server. Frameworks n are chosen by progressive filling with integer-valued tasking (x), *i.e.*, whole tasks are scheduled.

Numerical results for this illustrative example are given in Table I. & II. Similar results for *unused* (stranded) resources are given in [15]. 200 trials were performed for DRF, TSF and PS-DSF under RRR server selection. With Table II, we can obtain confidence intervals for the averaged quantities given in Table I for schedulers under RRR. Note how PS-DSF's performance under RRR is comparable to when frameworks and servers are jointly selected [17] (and with low variance in allocations). We also found that RRR-rPS-DSF performed just as rPS-DSF over 200 trials.

We found task efficiencies improve using *residual* forms of the fairness criterion. For example, the residual PS-DSF (rPS-DSF) criterion is

$$\tilde{K}_{n,j,\underline{x}_j} \quad = \quad x_n \max_r \frac{d_{n,r}}{\phi_n(c_{j,r} - \sum_{n'} x_{n',j} d_{n',r})}$$

That is, this criterion makes scheduling decisions by progressive filling using *current residual* (unreserved) capacities based on the *current* allocations x. From Table I, we see the improvement is modest for the case of PS-DSF. Improvements are also obtained by *best-fit* server selection. For example,

best-fit DRF (BF-DRF) first selects framework n by DRF and then selects the server whose residual capacity most closely matches their resource demands $\{d_{n,r}\}_r$ [35].

V. Online Scheduling of Spark Workloads with Mesos using Different Fair Allocation Algorithms

In the following, we present example experimental results comparing different schedulers implemented on Mesos. The execution traces presented in the figures were collected though Mesos' REST API by our external script and are typical of the multiple trials we ran.

A. Introduction including background on Mesos

The Mesos master (including its resource allocator, see [22]) works in dynamic/online environment with churn in the distributed computing/application frameworks it manages. When all or part of a Mesos slave[5] becomes available, its resources are offered to the frameworks in progressive filling fashion. The framework accepts the offered allocation in whole or part. When a framework's tasks are completed, the Mesos master may be notified that the corresponding resources of the slaves are released, and then the master will make new allocation decisions. We consider two implementations of fair resource scheduling algorithms in Mesos.

In **oblivious**[6] allocation, the allocator is not aware of the resource demands of the frameworks[7] so the allocator will offer all the remaining resources of the server to the framework. A framework running an uncharacterized application may be configured to accept all resources offered to it.

In **workload-characterized** allocation, each active framework n simply informs the Mesos allocator of its resource demands per task, $\{d_{n,r}\}_r$. The Mesos allocator selects a framework and allocates a single task worth of resources from a given slave with unassigned (released) resources.

In the following, we compare different scheduling algorithms implemented as the Mesos allocator. Given a pool of slaves with unused resources, PS-DSF [17], rPS-DSF and best-fit (BF) [35] allocations will depend on particular slaves. Initially, the slaves are always scheduled by the Mesos allocator as a pool.

B. Running Spark on Mesos

We use Spark as an example that may have heterogeneous resource demands, so Spark can certainly be replaced by any other workload generators. Each Spark job (Mesos framework) is divided into multiple tasks (threads). Multiple Spark *executors* will be allocated for a Spark job. The executors can simultaneously run a certain maximum number of tasks depending on how many cores on the executor and how many cores are required per task; when a task completes, the executor informs the task orchestrator (Spark driver) to request another, *i.e.* executors pull in work. Each executor is a Mesos task in the default "coarse-grained" mode [30] and an executor resides in a container of a Mesos slave [21]. Plural executors can simultaneously reside on a single Mesos slave. When starting a Spark job, the resources required to start an executor (d) and the maximum number of executors that can be created to execute the tasks of the job, may be specified. An executor usually terminates as the entire Spark job terminates [29] and hence its resources are released.

[5]A Mesos slave is a server that provides resources.

[6]Called "coarse grain" in Mesos.

[7]Indeed, the frameworks themselves may not be aware.

C. Experiment Configuration

In our experiments, there are two Spark submission groups ("roles" in Mesos' jargon): group Pi submits jobs that accurately calculate $\pi = 3.1415...$ via Monte Carlo simulation; group WordCount submits word-count jobs for a 700MB+ document. The executors of Pi require 2 CPUs and about 2 GB memory (Pi is CPU bottlenecked), while those of WordCount require 1 CPU and about 3.5 GB memory (WordCount is memory bottlenecked)[8]. Each group has five job submission queues, which means there could be ten jobs running on the cluster at the same time. Each queue initially has fifty jobs. Again, each job is divided into tasks and tasks are run in plural Spark executors (Mesos tasks) running on different Mesos slaves.

The Mesos slaves are six servers (AWS c3.2xlarge virtual-machine instances), two each of three types in our cluster. A type-1 server provides 4 CPUs and 14 GB memory, so it would be well utilized by 4 WordCount tasks. A type-2 server provides 8 CPUs and 8 GB memory, so it would be well utilized by 4 Pi tasks. A type-3 server provides 6 CPUs and 11 GB memory, so it would be well utilized by 2 Pi and 2 WordCount tasks. The Mesos master operates in a c3.2xlarge with 8 cores and 15 GB memory.

D. Prototype implementation

We modified the allocator module of Mesos (version 1.5.0) to use different scheduling criteria. We also modified the driver in Spark to pass on a framework n's resource needs per task ($\{d_{n,r}\}$) in workload-characterized mode. Our code is available online [23], [31].

E. Experimental Results for Different Schedulers

We ran the same total workload for the different Mesos allocators under Randomized Round-Robin (RRR) slave selection, except for BF-DRF. (In this section, we drop the "RRR" qualifier). A summary of our results is that overall execution time[9] is improved under workload characterization and allocations that are server specific.

1) DRF vs. PS-DSF in oblivious mode: The resource allocation under these two different fairness criteria are shown in Figure 1. It can be seen that PS-DSF can achieve higher resource utilization than DRF because its criterion naturally

[8]In fact, what is actually running inside Spark is not of interest in this section, where we are only interested in the allocation efficiency of different fairness criteria in Mesos.

[9]Note that minimizing execution time is the aim of CPU scheduling algorithms given individual task execution times.

matches the task demands to a specific server (a server-specific criterion), so it "packs" tasks better into heterogeneous servers. As a result, the entire job-batch under PS-DSF finishes earlier. Also note that at the end of the experiment, there is a sudden drop in allocated memory percentage. This is because the memory-intensive Spark WordCount jobs finish earlier and CPU is the bottleneck resource for the remaining Spark Pi jobs.

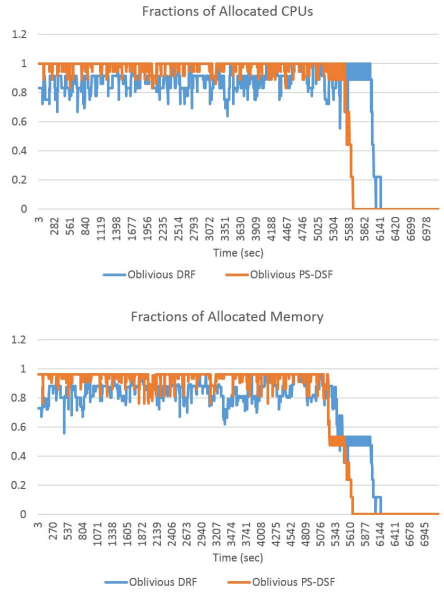

Fig. 1. Comparison between DRF and PS-DSF in oblivious mode.

2) Schedulers in workload-characterized mode: The experimental results under workload-characterized mode, as shown in Fig. 2, are consistent with their oblivious counterparts - PS-DSF has higher resource utilization than DRF. Also note that the resource utilizations in workload-characterized mode have less variance than those in oblivious mode, which will be explained in Sec. V-E3.

In Figure 3, we compare TSF [34] under RRR[10], rPS-DSF (under RRR), and BF-DRF (again, "best fit" is an slave-selection mechanism when there is a pool of slaves to choose from). From the figure, the execution times of BF-DRF and -rPS-DSF are comparable to PS-DRF (but *cf.* Section V-F) and shorter than TSF (which is comparable to DRF).

3) Oblivious versus Workload Characterized modes: We also compared oblivious and workload-characterized allocation for the same scheduling algorithm. Note that when a Spark job finishes, its executors may not simultaneously release resources from the Mesos allocator's point-of-view. So under oblivious allocation, it's possible that multiple Spark jobs can share the same server, as is typically the case under workload-characterized scheduling. However, oblivious allocation is a coarse-grained enforcement of progressive filling, where the

[10]Note that [34] also describes experimental results for a Mesos implementation of TSF.

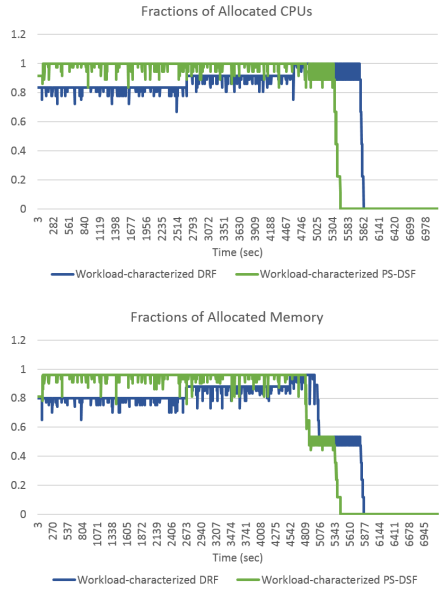

Fig. 2. Comparison between DRF and PS-DSF in workload-characterized mode.

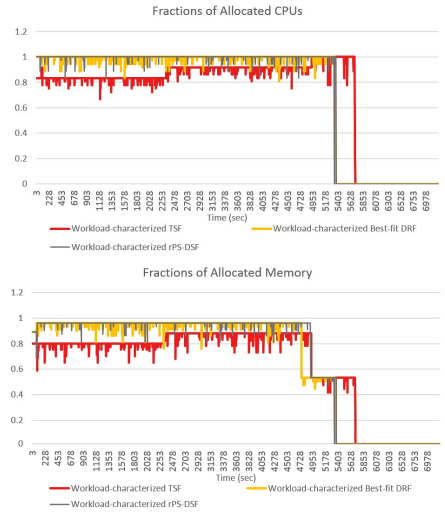

Fig. 3. Comparison among TSF, Best-fit DRF and rPS-DSF (in the workload characterized mode).

resources are less evenly distributed among the frameworks - some frameworks may receive the entire remaining resources on a slave in a single offer, leaving nothing available for others. From Figure 4, note how under oblivious allocation the amount of allocated resources drops more sharply when a Spark job ends, and variance of utilized resources under oblivious allocation is larger than under workload-characterized. Consequently, the entire job-batch tends to finish sooner under workload-characterized allocator. Similar results hold for PS-DSF [28].

In another group of experiments, we show that for homo-

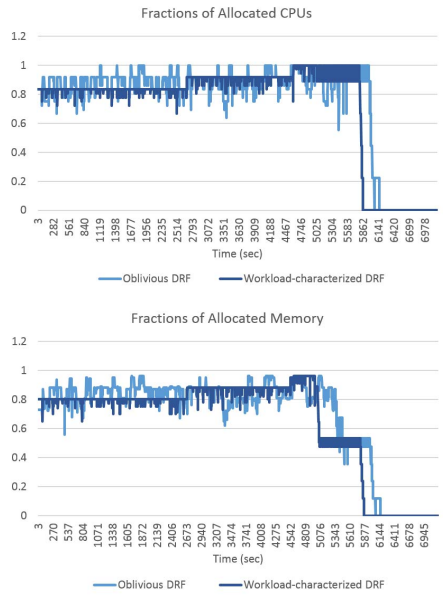

Fig. 4. Comparison between oblivious and workload-characterized modes under DRF.

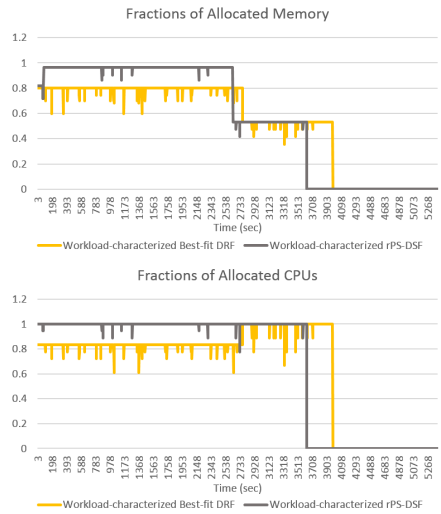

Fig. 5. Performance of Best-fit DRF and rPS-DSF given initial suboptimal allocation.

geneous servers, PS-DSF and DRF have comparable performance [28].

F. BF-DRF versus rPS-DSF

Finally, with a different experimental set-up, we compare BF-DRF (which first selects the framework and then selects the "best fit" from among available slaves) and a representative of a family of slave-specific schedulers, rPS-DRF under RRR. Consider a case where there are three servers, one of each of the above server types (types 1-3).

Suppose under a current allocation, we have one Spark-Pi and two Spark-WordCount executors on the type-1 server, two Spark-Pi and one Spark-WordCount executors on the type-2 server, and two Spark-Pi and two Spark-WordCount executors on the type-3 server. So, whenever a Pi or WordCount framework releases its executor's resources back to the cluster, its DRF "score" is reduced so the scheduler will always sends a resource offer to the same client framework in this scenario. On the other hand, rPS-DSF will make a decision considering the amount of (remaining) resources on the slave, and so will make a more efficient allocation.

We illustrate this with a numerical example in Figure 5. In this experiment, we let each group submit their Spark jobs through five queues with 20 jobs each. To create the above scenario, instead of exposing all the slaves to the clients, we register servers one by one from type-1 to type-3. From the figure, note that both rPS-DSF and BF-DRF have an initial inefficient memory allocation, but rPS-DSF is able to adapt and quickly increase its memory efficiency, while BF-DRF does not.

VI. Summary and Future Work

For a private-cloud setting, we considered scheduling a group of heterogeneous, distributed frameworks to a group of heterogeneous servers. We extended two general results on max-min fairness and proportional fairness to this case for a static problem under generic scheduling criteria. Subsequently, we assessed the efficiency of approximate max-min fair allocations by progressive filling according to different fairness criteria. Illustrative examples in heterogeneous settings show that max-min fair PS-DSF and rPS-DSF scheduling, are superior to DRF in terms of task efficiency performance (a metric related to proportional fairness) and that the efficiency of these "server specific" schedulers did not significantly suffer from the use of randomized round-robin server selection. Task efficiency was also improved when either the "best fit" approach to selecting servers was used or the fairness criteria was modified to use current residual/unreserved resource capacities. In [28], we give the full results of an online experimental study using our implementations of different schedulers on Spark and Mesos [23], [31] for benchmark workloads considering an execution-time performance metric.

In future work, we will consider scheduling (admission control and placement) problems in a public cloud setting. To this end, note that similar objectives to those considered herein for a private-cloud setting, particularly (10), may be reinterpreted as overall revenue based on *bids* ϕ for virtual machines or containers with fixed resource allocations d. Also, as profit margins diminish in a maturing marketplace, one expects that public clouds will need to operate with greater resource efficiency.

REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002.

[2] D. Bertsekas and R. Gallager. *Data Networks, 2nd Ed.* Prentice Hall, 1992.

[3] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, 2003.

[4] C. Chekuri and S. Khanna. On multi-dimensional packing problems. *SIAM Journal of Computing*, 33(4):837–851, 2004.

[5] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *Proc. USENIX NSDI*, March 2016.

[6] H. Christensen, A. Khan, S. Pokutta, and P. Tetali. Multidimensional Bin Packing and Other Related Problems: A Survey. https://people.math.gatech.edu/∼tetali/PUBLIS/CKPT.pdf, 2016.

[7] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, 2004.

[8] M. Cohen, V.Mirrokni, P. Keller, and M. Zadimoghaddam. Overcommitment in Cloud Services Bin packing with Chance Constraints. In *Proc. ACM SIGMETRICS*, Urbana-Campaign, IL, June 2017.

[9] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, 2003.

[10] Duke utility bill tariff, 2012. http://www.considerthecarolinas.com/pdfs/scscheduleopt.pdf.

[11] E. Friedman, A. Ghodsi, and C.-A. Psomas. Strategyproof allocation of discrete jobs on multiple machines. In *Proc. ACM Conf. on Economics and Computation*, 2014.

[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. USENIX NSDI*, 2011.

[13] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proc. USENIX NSDI*, 2011.

[14] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM Trans. Networking*, 21(6), Dec. 2013.

[15] G. Kesidis, Y. Shan, Y. Wang, B. Urgaonkar, J. Khamse-Ashari, and I. Lambadaris. Scheduling distributed resources in heterogeneous private clouds. https://arxiv.org/abs/1705.06102, June 28, 2018.

[16] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, and Y. Zhao. An Efficient and Fair Multi-Resource Allocation Mechanism for Heterogeneous Servers. http://arxiv.org/abs/1712.10114, Dec. 2017.

[17] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, and Y. Zhao. Per-Server Dominant-Share Fairness (PS-DSF): A Multi-Resource Fair Allocation Mechanism for Heterogeneous Servers. In *Proc. IEEE ICC, Paris*, May 2017.

[18] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In G. Goldszmidt and J. Schönwälder, editors, *Integrated Network Management VIII: Managing It All*, pages 247–261. Springer US, 2003.

[19] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Proceedings of the Seventh Real-Time Technology and Applications Symposium*, RTAS '01, 2001.

[20] D. A. Menasce. Web server software architectures. *IEEE Internet Computing*, 7(6):78–81, 2003.

[21] Apache Mesos - Containerizers. http://mesos.apache.org/documentation/latest/containerizer-internals/.

[22] Apache Mesos - Mesos Architecture. http://mesos.apache.org/documentation/latest/architecture/.

[23] Mesos multi-scheduler. https://github.com/yuquanshan/mesos/tree/multi-scheduler.

[24] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Trans. Networking*, Vol. 8, No. 5:pp. 556–567, 2000.

[25] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the Second International Conference on Automatic Computing*, ICAC '05. IEEE Computer Society, 2005.

[26] J. Puchinger, G. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, Spring 2010.

[27] H. Seltman. Approximation of mean and variance of a ratio. http://www.stat.cmu.edu/ hseltman/files/ratio.pdf.

[28] Y. Shan, A. Jain, G. Kesidis, B. Urgaonkar, J. Khamse-Ashari, and I. Lambadaris. Online Scheduling of Spark Workloads with Mesos using Different Fair Allocation Algorithms. https://arxiv.org/abs/1803.00922, March 2, 2018.

[29] Apache Spark - Dynamic Resource Allocation. https://spark.apache.org/docs/latest/job-scheduling.html.

[30] Apache Spark - Running Spark on Mesos. https://spark.apache.org/docs/latest/running-on-mesos.html.

[31] Spark with resource demand vectors. https://github.com/yuquanshan/spark/tree/d-vector.

[32] A. Stuart and K. Ord. *Kendall's Advanced Theory of Statistics*. Arnold, London, 6th edition, 1998.

[33] M. Varnamkhasti. Overview of the algorithms for solving the multidimensional knapsack problems. *Advanced Studies in Biology*, 4(1):3747, 2012.

[34] W. Wang, B. Li, B. Liang, and J. Li. Multi-resource fair sharing for datacenter jobs with placement constraints. In *Proc. Supercomputing*, Salt Lake City, Utah, 2016.

[35] W. Wang, B. Liang, and B. Li. Multi-resource fair allocation in heterogeneous cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2822–2835, Oct. 2015.

[36] W. Xu, P. Bodik, and D. Patterson. A flexible architecture for statistical learning and data mining from system log streams. In *Proceedings of Workshop on Temporal Data Mining: Algorithms, Theory and Applications at the Fourth IEEE International Conference on Data Mining*, Brighton, UK, 2004.

[37] K.-K. Yap, T.-Y. Huang, Y. Yiakoumis, S. Chinchali, N. McKeown, and S. Katti. Scheduling packets over multiple interfaces while respecting user preferences. In *Proc. ACM CoNEXT*, Dec. 2013.