An Appropriate Parameterized Utility Technique On Heterogeneous Server Dependencies

NAVA RAVI KIRAN
Student of M.Tech (CSE), Department of Computer Science & Engineering, Kakinada Institute of Engineering & Technology, Korangi, AP

D . SRINUVAS
Asst. Prof, Dept. of Computer Science & Engineering, Kakinada Institute of Engineering & Technology, Korangi, AP

Abstract: A new server-based approach incorporated in Heterogeneous Servers. Current cloudinfrastructures are mostly homogeneous composed of a large number of machines of the same type – centrally managed and made available to the end user. In a cloud computing pattern, multiple resources types were utilizing. Users may have diverse resource needs. Furthermore, diversity in server properties/capabilities may mean that only a subset of servers may be usable by a given user. In platforms with such heterogeneity, we identify important limitations in existing multi-resource fair allocation mechanisms, notably Dominant Resource Fairness and its follow-up work. To overcome such limitations, we propose a new server-based approach; each server allocates resources by maximizing a per-server utility function. We propose a specific class of utility functions which, when appropriately parameterized, adjusts the trade-off between efficiency and fairness, and captures a variety of fairness measures. We establish conditions for the proposed mechanism to satisfy certain properties that are generally deemed desirable, e.g., envy-freeness, sharing incentive, bottleneck fairness, and Pareto optimality. To implement resource parameterized mechanism, we develop an iterative algorithm which is shown to be globally convergent on Heterogeneous server dependencies.

Key words: Heterogeneous Server; Cloud computing pattern; Utility function;

INTRODUCTION

A. Cloud computing

It is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet. Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server. Clouds may be limited to a single organization, or be available to multiple organizations (public cloud). Cloud computing relies on sharing of resources to achieve coherence and economies of scale. Advocates of public and hybrid clouds note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand, providing the burst computing capability: high computing power at certain periods of peak demand.

Cloud computing has become increasingly popular as high-performance computing systems. As the workloads to data-centers housing cloud computing platforms are intensively growing, developing an efficient and fair allocation mechanism which guarantees quality-of-service for different workloads has become increasingly important. Efficient and fair resource allocation in such a shared computing system is particularly challenging because of (a) the presence of multiple types of resources, (b) diversity in the workloads’ needs for these resources, (c) heterogeneity in the resource capacities of servers, and (d) placement constraints on which servers may be used by a workload. In the following four paragraphs we briefly elaborate on each of these complexities.

The multi-resource needs of cloud workloads imply that conventional single-resource oriented notions of fairness are inadequate [1]. Dominant Resource Fairness (DRF) is the first allocation mechanism which describes a notion of fairness for allocating multiple types of resources for a single server. Using DRF users receive a fair share of their dominant resource [1]. Of all the resources requested by the user (for every unit of work called a task), its dominant resource is the one with the highest demand when demands are expressed as fractions of the overall resource capacities. DRF is shown to achieve several properties that are commonly considered desirable from a multi-resource fair allocation mechanism.

Heterogeneity of workloads’ resource demands is another complexity which results in a trade-off between efficiency and fairness. Specifically, heterogeneity of users’ demands may preclude some resources from being fully utilized. Hence, the DRF allocation may result in a poor resource utilization even when there is only one server [2], [3], [4]. To address this issue, [2] proposed to allocate resources by applying the so-called α-proportional fairness (instead of max-min fairness [5]) on dominant shares. The proposed mechanism, when appropriately parameterized, adjusts the trade-off between efficiency...
and fairness. However, it is applicable only to a single server/resource-pool. In the case of multiple heterogeneous servers, there are several studies investigating/ extending DRF allocation where there is no placement constraint [6], [7], [8]. In all of these works, fairness is defined in terms of a global metric, a scalar parameter defined in terms of different resources across all servers. E.g., [7] presents an extension to DRF where the dominant resource for each user is identified as if all resources were concatenated at one server, and subsequently the resources are allocated by applying max-min fairness on the dominant shares. Since such a global metric may not perfectly capture the impact of server heterogeneity, such approaches may lead to an inefficient resource utilization. Moreover, such mechanisms may not be readily implementable in a distributed fashion [9], as each server needs information on the available resources over all servers. Such information may not be available at each server, especially in a cloud computing environment where the resource capacities (and even activity of servers) might be varying.

There are limited works in the literature investigating multi-resource fair allocation in the presence of user placement constraints [10], [11]. In this case, it is yet unclear how to globally identify the dominant resource as well as the dominant share for different users, as each one may have access only to a subset of servers. Work in [11] presents an extension to DRF identifying the user share by ignoring placement constraints and applying a similar approach as in an unconstrained setting. We show that this approach may not achieve fairness in the specific case that one of the resources serve as a bottleneck (see Section 2.2). In [12] we proposed a multi-resource fair allocation mechanism, called Per-Server Dominant Share Fairness (PSDSF), which is applicable to heterogeneous servers in the presence of placement constraints. The intuition behind PSDSF is to capture the impact of server heterogeneity by measuring the total allocated resources to each user explicitly from the perspective of each server. Specifically, PSDSF identifies a virtual dominant share (VDS) for each user with respect to each server (as opposed to a single system-wide dominant share in DRF). The VDS for user \( n \) with respect to server \( i \) is defined as the ratio of \( \frac{n}{\sum_{n} - \text{the total number of tasks allocated to user } n} \) over the number of tasks executable by user \( n \) when monopolizing server \( i \). Then the resources at each server are allocated by applying max-min fairness on VDS. This approach is amenable to a distributed implementation. It results in an enhanced performance over the existing mechanisms, and satisfies certain properties essential for fair allocation of resources [12].

B. Contributions

In this paper, we build upon and generalize our proposed PS-DSF allocation mechanism [12] to capture the trade-off between efficiency and fairness. We concisely summarize our contributions.

- We propose a new server-based formulation (which includes PS-DSF as a special case) to allocate resources while capturing server heterogeneity. The new formulation can be viewed as a concave game among different servers, where each server allocates resources by maximizing a per-server utility function.

- We study a specific class of utility functions which results in an extension of a-proportional fairness on VDS. We show how the resulting allocation, which we call \( \alpha \)-PF-VDS, captures the trade-off between efficiency and fairness by adjusting the parameter \( \alpha \). We show that \( \alpha \)-PF-VDS satisfies bottleneck fairness, envy-freeness, and sharing incentive properties (as defined in Section 2.1) for \( \alpha \geq 1 \), and Pareto optimality for \( \alpha = 1 \).

- We develop a (centralized) convergent algorithm to implement our proposed mechanism. Towards this, we introduce an equivalent formulation for which we derive an iterative solution (Section 4 and 5.1).

- We propose a simple heuristic to develop a distributed implementation for our resource allocation mechanism (Section 5.2).

- We carry out extensive simulations, driven by real-world traces, to show the enhanced performance of our proposed mechanism (Section 6).

C. Game – Theoretic Approach:

There are several works in the literature which study the resource allocation problem in a cloud computing environment with a game-theoretic approach. Among these, [13], [14], [15] are limited to a single-resource setting, while [16], [17], [18] consider a multi-resource environment. In these studies, the multi-resource allocation problem is formulated as a game, where each server strives to maximize a per-server utility function. The utility function at each server is defined as the summation of resource utilization, minus the variance of a fairness-related metric for different users.

The work in [16] aims at minimizing the variance of global dominant shares for different users. However, since it chooses DRF as the underlying notion of fairness, it has the same limitations as DRF for heterogeneous servers (see Section 2.2 for a discussion of such limitations). In [17], [18], a two-stage mechanism is proposed wherein each user is initially assigned to a server/coalition-of-servers. Then, each server/coalition-of-servers strives to minimize the variance of the local dominant shares for the assigned users, while maximizing the resource utilization. Such a local implementation of DRF, however, may not satisfy bottleneck fairness in the
whole system. Moreover, these works need to solve an extensive form game with a huge strategy set space [16], [18], which may not be implementable in a distributed fashion.

D. Single-resource fair scheduling

There are some recent works investigating max-min fair scheduling for one type of resource while respecting placement constraints [19], [20], [21], [22], [23]. A deadline-aware scheduler is also proposed in [24] which assign CPU cores to different users in a way that their deadlines are met in a fair manner. These simple resource schedulers could be useful in a multi-resource setting when one of the resources (e.g. CPU) is dominantly requested by all users.

Implementation:

Consider a set $K$ of $K$ heterogeneous servers/resourcepools each containing $M$ types of resources. We denote by $c_i, r \geq 0$, the capacity (i.e., amount) of resource $r(1, 2, \ldots, M)$ on server $i$. We make thereasonable assumption that all resources on each server are arbitrarily divisible among the users running on it. Let $N$ denote the set of Nactive users. Let $w_n > 0$ denote the weight associated with user $n$. The eights reflect the priority of users with respect to each other. Let $d_{in} = [d_{in}, r]$ denote the per task demand vector for user $n \in N$, i.e., the amount of each resource required for executing one task for user $n$. Let $x_n, r \in R$ denote the number of tasks that are allocated to user $n$ from server $i$. Assuming linearly proportionate resource-needs, $x_n, id_{in} = [x_n, id_{in}, r]$ gives the amounts of different resources demanded by user $n$ from server $i$. Due to heterogeneity of users and servers, each user may be restricted to get service only from a subset of servers.

$$d_i = [1.5, 1.10] \quad d_j = [1.2, 1.10] \quad d_k = [0.5, 1.0] \quad d_l = [1, 0.5, 0]$$

Fig. 1: A heterogeneous multi-resource system with two servers and four equally weighted users.

For example, users may not run tasks on servers which lack some required resources. Furthermore, each user may have some special hardware/software requirements (e.g., public IP address, a particular kernel version, GPU, etc.) which further restrict the set of servers that the user’s tasks may run on. Let $N_i = \emptyset$ denote the set of eligible users for server $i$. The placement constraints imply that $x_n, i = 0, n \in N_i, \forall i$. Such constraints are referred to as hard placement-constraints.

Soft-constraints, such as data-locality, are another type of constraints which describe preferences of each user over different servers [19]. For instance, consider the example in Fig. 1, where three types of resources, CPU, RAM, and network bandwidth are available over two servers in the amounts of $c_1 = [12 \text{ cores}, 4\text{GB}, 75\text{MB/s}]$ and $c_2 = [8 \text{ cores}, 16\text{GB}, 0\text{MB/s}]$, where no communication bandwidth is available over the second server; four users with their own demand vectors are also shown in the figure. In this example, the first two users require network bandwidth for execution of their tasks, so they are not eligible to run tasks on the second server. However, the last two users may run tasks on both servers.

DOMINANT RESOURCE FAIRNESS

Multi-resource fair allocation was originally studied in [1] under the assumption that all resources are aggregated in one resource pool. Specifically, let $c$ denote the total capacity of resource $r$. Let $\mathcal{A} = [a_n, r]$ denote the amount of different resources allocated to user $n$ under some allocation mechanism. The utilization of user $n$ of its allocated resources, $\mathcal{U}(\mathcal{A})$, is defined as the number of tasks, $x_n$, which could be executed using $\mathcal{A}$, that is: $\mathcal{U}(\mathcal{A}) = x_n = \min\mathcal{A}, \forall n, \forall r$.

In [1] the following properties are deemed desirable for a multi-resource allocation mechanism.

• Sharing incentive: Each user is able to run more tasks compared to a uniform allocation where each user $n$ is allocated a $\phi(n)/m$ fraction of each resource.

• Envy freeness: A user should not prefer the allocation vector of another user when adjusted according to their weights, i.e., it should hold that $\mathcal{U}(\mathcal{A}) \geq \mathcal{U}(\mathcal{A}^n)$ for all $n, m$.

• Bottleneck fairness: If there is one resource which is dominantly requested by every user, then the allocation satisfies max-min fairness for that resource.

• Pareto optimality: It should not be possible to increase the number of tasks $x_n$ for any user $n$, without decreasing $x_m$ for some other user(s).

• Strategy proofness: Users should not be able to increase their utilization by erroneously declaring their resourcedemands.

The reader is referred to [1] or [25] for further details. Sharing incentive provides some sort of performance isolation, as it guarantees a minimum utilization for each user irrespective of the demands of the other users. Envyfreeness embodies the notion of fairness. Bottleneck fairness describes a necessary condition which applies to a specific case that one resource is dominantly requested by every user, so that a single-resource notion of fairness is applicable. These three properties are essential to achieve fairness. So, we refer to them as essential fairness-related properties. Pareto optimality is a
benchmark for maximizing system utilization. Finally, strategy proofness prevents users from gaming the allocation mechanism. In our view these properties are applicable mainly for private settings. In public settings, users pay explicit costs for their usage or allocations and the provider’s goal is to maximize its profits subject to allocation guarantees for users. Even for private clouds, strategy proofness would only be necessary in settings where users act selfishly. In many private settings, strategy proofness is not needed. In view of this, we will not consider strategy proofness.

DRF is the first multi-resource allocation mechanism satisfying all the above properties. Specifically, for every user n, the Dominant Resource (DR) is defined as \[ \rho(n) := \arg\max_{\forall r} x_{n,r} \] (2) that is, the resource whose greatest portion is required for execution of one task for user n. The fraction of the DR that is allocated to user n is defined as its dominant share \( sn := an, \rho(n) c\rho(n) \) (3).

Without loss of generality, we may restrict ourselves to non-wasteful allocations, i.e., \( an = xndn, \forall n \). Hence, an allocation \( \{xn\} \) is feasible when: \( nxdn, r \leq cr, \forall r \). (4)

**Definition 1.** An allocation \( \{xn\} \) satisfies DRF, if it is feasible and the weighted dominant share for each user, \( sn/\rho\) cannot be increased while maintaining feasibility without decreasing some resource \( m \) with \( sm/\rhom \leq sn/\rho n \). (1)

DRF is a restatement of min fairness in terms of dominant shares. What make it appealing are the desirable properties which are satisfied by this allocation mechanism.

### A. Existing challenges with heterogeneous servers and placement constraints

In case of heterogeneous servers (whether there are any placement constraints or not), a natural approach to extend DRF is to identify a system-wide dominant resource for each user, as if all resources were concatenated within a single virtual server. Specifically, let \( cr := icir \) denote the total capacity of resource \( r \) within such a virtual server. Then, one may identify the dominant resource for each user \( n \) according to (2). Furthermore, the global dominant share for user \( n \) is given by: \( sn := xnm\max_{\forall r} x_{n,r} \) (5) where \( xni \) is the total number of tasks that are allocated to user \( n \) from different servers, that is \( xni := ixi \). As in Definition 1, one may find an allocation \( \{xn,i\} \) which satisfies max-min fairness in terms of the global dominant shares [7]. Such an allocation, referred to as DRFH (DRF for Heterogeneous servers [7]), is shown to achieve Pareto optimality and envy freeness.

However, it fails to provide sharing incentive [7]. We believe that the definition of bottleneck fairness employed by DRFH (with respect to a single virtual server that aggregates all resources) is also controversial. Specifically, if all users have the same dominant resource (with respect to the above mentioned virtual server), then DRFH satisfies max-min fairness with respect to such a resource [7].

In case of heterogeneous servers with placement constraints, however, one may consider other conditions under which a resource serves as a bottleneck.

**Definition 2.** A resource \( p \) is said to be a bottleneck if for every server \( dnp, ci, p \geq dnr, ci, r, n, Ni \). (6)

If there exists a bottleneck resource, then the allocation should satisfy max-min fairness with respect to that resource. Unfortunately, DRFH does not satisfy bottleneck fairness in the sense of Definition 2. To appreciate this shortcoming of the DRFH mechanism, consider the example in Fig. 1, where the second resource (RAM) is dominantly requested by eligible users at each server. According to Definition 2, RAM is identified as the bottleneck resource in this example.

To allocate the RAM resources in a fair manner, each user should be allocated \( x1 = x1,1 = 2 \) (i.e., two tasks from the first server), \( x2 = x2,1 = 6 \), \( x3 = x3,2 = 8 \) and \( x4 = x4,2 = 8 \) tasks, respectively (This allocation results from our proposed PS-DSF allocation mechanism [12]).

On the other hand, the DRFH mechanism would instead identify network bandwidth as the dominant resource for the first two users and RAM as the dominant resource for the last two users. To achieve max-min fairness in terms of dominant shares, the DRFH mechanism allocates \( x1 = x2 = 3 \) and \( x3 = x4 = 8 \) tasks to each user. Under such an allocation, the RAM resources are not allocated in a fair manner to the first two users.

Yet another extension of DRF, which applies to heterogeneous servers in the presence of placement constraints, is TSF [11].

As in [11], we let \( i, j \) be defined as the number of tasks executable for user \( n \) when monopolizing all servers as if there were no placement constraints. An allocation is said to satisfy Task Share Fairness (TSF), when \( xn/\rho n \) satisfies max-min fairness [11]. When there is only one server, then \( xn/\rho n \) results in the dominant share for each user \( n \). In such a case, TSF reduces to DRF. In case of heterogeneous servers with placement constraints, TSF is shown to satisfy Pareto optimality, envy freeness and sharing incentive properties [11]. However, we show by example that this mechanism may not satisfy bottleneck fairness (neither in the sense of Definition 2, nor in the conventional sense...
For instance, consider again the example in Fig. 1, where the second resource is identified as a bottleneck according to Definition 2. The number of tasks that each user may run in the whole cluster is: $y_1 = 4$, $y_2 = 12$, and $y_3 = y_4 = 16 = 20$ tasks, respectively. Hence, each user is allocated $x_1 = x_1, l = 5/3$, $x_2 = x_2, l = 5$, $x_3 = x_3,1 + x_3,2 = 8 + 1/3 = 25/3$ and $x_4 = x_4,1 + x_4,2 = 8 + 1/3 = 25/3$ tasks, according to the TF mechanism, which differs from the fair allocation in this example.

### B. Per-server dominant share fairness (PS-DSF)

In this subsection, we describe PS-DSF which we introduced in [12]. PS-DSF is an extension to DRF which is applicable for heterogeneous servers in the presence of placement constraints. The core idea of this mechanism is to introduce a “virtual dominant share” for every user, with respect to each server. Towards this, we first identify the dominant resource for every user $n$ with respect to each server, $\rho(n,i) := \arg\max d_{n,i}, r(i,l)$. (7)

It is assumed that $\gamma_n = 0$ for all $n / N$. We set $\gamma_n, i = 0$ if $n / N$

\[
\gamma_n, i := \min_r \frac{c_{i,r}}{d_{n,r}} = \frac{c_{i,\rho(n,i)}}{d_{n,\rho(n,i)}}, n \in N_i.
\]

**Definition 3.** The Virtual Dominant Share (VDS) for user $n$ with respect to server $i$, $s_{n,i}$, is defined as:

\[
s_{n,i} := \frac{x_n}{\gamma_n, i} = \frac{x_n d_{n,\rho(n,i)}}{c_{i,\rho(n,i)}}, \quad (9)
\]

where $x_n = j \in Xn, j$ is the number of tasks that are reallocated to user $n$ from every server $j \in K$. We have the following conditions on allocation, $x :=$

\[
\sum_{n \in N_i} x_{n,i} d_{n,r} \leq c_{i,r}, \forall i, r.
\]

\[
x_{n,i} = 0, \quad n \notin N_i, \forall i.
\]

**Definition 4.** An allocation $x$ satisfies PS-DSF, if it is feasibileand the allocated tasks to each user, $x_n$,cannot be increased (while maintaining feasibility) without decreasing $x_m$, for some user mand server $i$ with $s_{m,i} / q_m < s_{n,i} / q_n$. Intuitively, $s_{n,i}$ gives a measure of the total allocated resources to user $n$ from the perspective of server $i$. In particular, $s_{n,i}$ gives the normalized share of the dominant resource for user $n$ with respect to server $i$ (which should be allocated to it as if $x_{n,i}$ tasks were allocated resources solely from server $i$ see the right hand side of (9)). The reader may note that $s_{n,i}$ could be possibly greater than 1, as some tasks might be allocated to user $n$ from other servers. According to PS-DSF, the available resources at each server $i$ are allocated by applying (weighted) max-min fairness on $(s_{n,i})$. It can be seen that PS-DSF reduces to DRF when there is only one server.

To gain more intuition, consider again the example in Fig. 1, but this time let $d_4 = [1, 0.5, 0]$. In this case, each user may run $y_1 = 4, y_2 = 12, y_3 = 4, y_4 = 8$ tasks when monopolizing server 1. The third and the fourth users each may run $y_3,2 = 16$ and $y_4,2 = 8$ tasks when monopolizing server 2. In order to satisfy PS-DSF, each user should reallocate $x_1 = x_1, l = 2, x_2 = 2, x_2, l = 5$, $x_3 = x_3,1 + x_3,2 = 32/3$ and $x_4 = x_4,1 + x_4,2 = 16/3$ tasks, respectively. Therefore, the VDS (c.f. Definition 3) for each user with respect to the firstserver is $s_{1, i} = 1, s_{2, i} = 1, s_{3, i} = 8/3$ and $s_{4, i} = 2/3$. Also, the VDS for user 3 and 4 with respect to the second server is $s_{3, i} = 2/3$. The reader can verify that for each server, the allocated tasks to any user may not be increased without decreasing the allocated tasks to another user with less or equal VDS. Intuitively, the RAM resources are dominantly requested by eligible users of the first server in this example.

To achieve per-server dominant share fairness, the first server strives to allocate the RAM resources in a fair manner. Towards this, the last two users are not allocated resources from the first server, since there exists enough RAM for them over the second server. On the other hand, the second server identifies RAM as the dominant resource for User 3 and CPU as the dominant resource for User 4. To achieve PS-DSF, the second server needs to balance the respective dominant shares for User 3 and User 4. The resulting PS-DSF allocation is shown in Fig. 2. The DRF and TSF allocations for this example are also illustrated in Fig. 2. Besides its desirable performance in terms of fairness, the PS-DSF allocation is observed to be more
efficient in utilizing different resources compared to the DRFH and TSF mechanisms.

Table 1: Properties of different allocation mechanisms in case of heterogeneous servers with placement constraints: sharing incentive (SI), envy freeness (EF), Pareto optimality (PO), and bottleneck fairness (BF).

<table>
<thead>
<tr>
<th>Property</th>
<th>DRFH</th>
<th>TSF</th>
<th>PS-DSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BF</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Compares essential sharing properties which are satisfied under different allocation mechanisms [12]. The reader may note that PS-DSF does not satisfy Pareto optimality in general. It is worth noting that Pareto optimality may not also be satisfied in other works, e.g., [17], [18], which aim at developing a distributed implementation for DRFH. PS-DSF not only is amenable to distributed implementation (as we show in [12]), but also may lead to more efficient utilization of resources compared to the DRFH and TSF mechanisms [12] (as also can be observed in Fig. 2, or in the trace-driven simulations in Section 6). The intuitive reason for this is that each of the DRFH and TSF allocation mechanisms allocates resources based on a global metric.

Since a global metric throws away information about the actual distribution of resources across servers, approaches based on it may not perfectly capture the impact of server heterogeneity, and therefore may lead to an inefficient resource utilization in heterogeneous settings.

In summary, PS-DSF has been shown to satisfy the essential fairness-related properties, i.e., envy-freeness, sharing incentive and bottleneck fairness, has been observed to offer highly efficient utilization of resources, and is amenable to distributed implementation [12].

PARAMETERIZED UTILITY TECHNIQUE ON HETEROGENEOUS SERVER DEPENDENCY

As already discussed, in most of the existing multi-resource allocation mechanisms, fairness is defined in terms of a global metric, a scalar parameter defined for each user in terms of different resources across all servers. Such mechanisms may not succeed in satisfying all the essential fairness-related properties (c.f. Section 2.2), may not readily be implementable in a distributed fashion, and may lead to inefficient resource utilization. In this section, we propose a new formulation for multi-resource allocation problem which is based on a per-server metric (as opposed to a global metric) for different users, so that server heterogeneity is captured. The proposed allocation mechanism is built upon our proposed PS-DSF allocation mechanism [12], which was briefly described in the previous section. It generalizes PS-DSF in order to address the trade-off between efficiency and fairness. Furthermore, it inherits all the properties that are satisfied by PS-DSF.

The properties of the αPF-VDS allocation mechanism In this section, we investigate different properties which are satisfied by the αPF-VDS mechanism. In case of heterogeneous servers with placement constraints, we need to extend the notion of sharing incentive property. The notion of bottleneck fairness has been extended by Definition 2. Other properties, Pareto optimality and envy freeness follow the same definitions as described in Section 2. To generalize the sharing incentive property, consider a uniform allocation, where a fraction qn/m of the available resources over each server (whether this server is eligible or not) is allocated to each user n. An allocation is said to satisfy sharing incentive, when each user is able to run more tasks compared to such a uniform allocation.

A. Adjusting the resource utilization

As discussed in Section 3.2 and shown in the example of Fig. 3, the resource utilization improves as the parameter α in the αPF-VDS mechanism gets smaller. In this subsection, we further investigate this effect when applying this mechanism to real-world workloads.

In the Bitbrain workload, users become active/incactive with a relatively low rate. So, the resources could be allocated to different users/virtual-machines in a semi-static manner. In this case, we do several experiments for different sets of active users chosen at random instants of time. In particular, for each set of active users we find the αPF-VDS allocation for α = 1, α = 3 and α = ∞. In case of α = 1 and α = 3 we employ the distributed iterative algorithm proposed.

Fig. 2: A data-center distributed over three different locations.

There are three types of servers, where the configuration of resources (CPU and memory
respectively) for each type of server is as follows: (4 GHz, 12 GBytes) for type 1, (8 GHz, 8 GBytes) for type 2, (16 GHz, 4 GBytes) for type 3. The storage devices are of two different types, where the read/write bandwidth for type 1, used at the first and the last cluster locations, is 32 MB/s, and for type 2, used at the second location, is 100 MB/s. Finally, there are two types of broadband connections, where the first type provides a bandwidth of 100 Mb/s (and 1 Gb/s respectively) to send (receive) data, while the second type provides a bandwidth of 1 Gb/s (and 2 Gb/s respectively) to send (receive) data.

For $\alpha = \infty$, we use the customized algorithm proposed in [12] to implement PS-DSF. The latter is also available open-source at [34], where it is prototyped for cluster scheduling with Apache-Mesos. Fig. 5 shows the average processing time to find the PS-DSF allocation and the $\alpha$PF-VDS allocation for the computing cluster of Fig. 4, and for an expanded cluster where the number of users and servers are doubled. It can be observed that the convergence time for the distributed iterative algorithm of Section 5.2 increases as $\alpha$ gets larger. Such an overhead (which is less than 1 second in a cluster with thousands of users) is acceptable for modest values of $\alpha$, especially in a semi-static setting where the same allocation can be used for at least a few minutes. For the case $\alpha = \infty$, the customized PS-DSF allocation algorithm offers much less processing time (around 0.03 second), which remains the same range even for the expanded cluster.

In Fig. 2 we report the overall resource utilization that is achieved on average over different servers and over 100 runs, for different variants of $\alpha$PF-VDS. As expected, the $\alpha$PF-VDS results in a greater utilization of different resources for smaller values of $\alpha$. In this experiment, the improvement in utilization could be significant as $\alpha$ ranges from $\infty$ to 1.

For the Google workload, we allocate resources in a semi-dynamic manner. In particular, consider the computing cluster described in Table 3, where 2% of users from the Google traces are randomly chosen as the input workload.

In such a setting, we decide to (re)allocate resources from the servers to demanding jobs (at least) every 5 minutes.

Specifically, given the resource usage for different tasks of each job by the Google traces, we may find the demand vector for each job (at the beginning of each 5 minutes interval) as the summation of the resource usage for different tasks (different tasks of the same job usually have proportional demands [32]). Given the total demand for each job, $d_n = [d_n,r]$, we define an execution quantum for job $n$ as a block of resources in the amount of $d_n := d_n/r \max rd m,r$ that is allocated to job $n$ for 1 second. Accordingly, job $n$ demands $q_n := 300 \max rd m,r$ execution quanta for the next 5 minutes interval. We use the normalized demand vectors, $\{d_n\}$, as the input to the $\alpha$PF-VDS mechanism in order to find the number of tasks that are allocated to each job under this mechanism. Given the allocated tasks to each job, the completion time for job $n$ is given by $q_n/x_n$. If a job leaves the system during the 5 minutes period, the released resources are reallocated among the remaining jobs.

The number of execution quanta demanded by different jobs, and also their activity duration, span a quite wide range. Our observations over an interval of 24 hours show that around 38% of jobs are completed within a 5 min period, while 16% of them are active for more than 24 hours.

![Fig. 3: The average processing time to find the PS-DSF allocation and the $\alpha$PF-VDS allocation in the computing cluster of Fig. 2, and in an expanded cluster where the number of users and servers are doubled. The average is calculated over 10 different runs. The 95% confidence interval is shown at the top of each bar.](image)

![Fig. 4: The overall resource utilization, averaged over different servers of the computing cluster of Fig. 4 (serving the Bitbrain workload) and over 100 runs, for different variants of $\alpha$PF-VDS: $\alpha = 1$ (proportional fairness), $\alpha = 3$, $\alpha = \infty$ (PS-DSF). The 95% confidence interval is shown at the top of each bar.](image)

**COMPARISON WITH EXISTING MECHANISMS**

In this subsection, we compare our proposed mechanism in terms of resource utilization against the known-proposed multi-resource fair allocation mechanisms, which briefly described in Section 2.2.
Specifically, we compare the PS-DSF mechanism (which is the least efficient variant of αPF-VDS), against the DRFH and TSF allocation mechanisms (which are applicable to heterogeneous servers in the presence of placement constraints). First, consider the computing cluster of Fig. 3, each of which is fed by the Bitbrain workload. We employ each of the aforementioned mechanisms to allocate resources from the servers in Fig. 3 to different sets of active users chosen at random instants of time. The overall utilization that is achieved by each of these mechanisms for different resources, is shown in Fig. 4, when averaged over different servers and over 100 runs. It can be observed that the PS-DSF allocation mechanism outperforms the other mechanisms in terms of the achieved utilization for different resources. In particular, the resource utilization is enhanced by the PS-DSF mechanism for up to 20% for some resources.

We make similar observations with the Google traces. Specifically, consider again the computing cluster described in Table 3, where 2% of jobs in the Google traces are randomly chosen as the input workload. We employ each of the PS-DSF, DRFH and TSF allocation mechanisms to allocate resources of the specified servers in Table 3 to demanding jobs over an interval of 24 hours. Fig. 4 compares the overall resource utilization (averaged over different servers) that is achieved by each of the allocation mechanisms. It can be observed that PS-DSF is again more efficient in utilizing different resources, compared to the DRFH and TSF allocation mechanisms, while the achieved resource utilization by DRFH and TSF mechanisms is almost the same. The overall resource utilization that is achieved on average over the 24-hour period is shown in Fig. 4 for different allocation mechanisms. The resource utilization over the last two classes of servers is also shown in Fig. 5. It can be observed that the PS-DSF allocation improves the resource utilization over the last two classes of servers more significantly.

Intuitively, the PS-DSF allocation mechanism allocates resources at each server based on the per-server virtual dominant shares. So, at each server it gives more priority to users which may run more tasks (c.f. (9)). Hence, one may expect that the PS-DSF allocation mechanism results in a greater resource utilization compared to the DRFH and TSF mechanisms, especially when the resources are heterogeneously distributed over different servers. That is the reason why a more significant increase in utilization is achieved by the PS-DSF allocation over the last two classes of servers, where the resources are more heterogeneously distributed (the available resources over the first two classes of servers).

CONCLUSION

We studied an Appropriate Parameterized Utility Technique on Heterogeneous Server Dependencies in the presence of placement constraints. We identified potential limitations in the existing multi-resource fair allocation and parameterized mechanisms, DRF and
its follow up work, when used in such environments. In certain occasions, they may not succeed in satisfying all of the essential fairness-related parameterized properties, may not be readily implementable in a distributed fashion, and may lead to inefficient resource utilization. We proposed a new server-based approach to efficiently allocate resources while capturing server heterogeneity. We showed how our proposed pPF-VDS mechanism could be parameterized (by \( \alpha \)) to adjust the trade-off between efficiency and fairness.

Distributed parameterized implementation usually comes at the price of degrading the performance. Our proposed mechanism not only is amenable to distributed implementation, but also results in an enhanced resource utilization compared to the existing mechanisms. We carried out extensive simulations, driven by real-world traces, to demonstrate and implementable in a distributed fashion.

REFERENCES


AUTHOR’s PROFILE

NAVA RAVI KIRAN is pursuing M.Tech(CSE) in the department of CSE from Kakinada Institute of Engineering & Technology, Korangi.

D.SRINUVAS is working as an Assistant Professor in the Department of CSE, Kakinada Institute of Engineering & Technology, Korangi, Kakinada.