O Passado Também Importa: Um Mecanismo de Alocação Justa de Múltiplos Tipos de Recursos ao Longo do Tempo

Hugo Sadok, Miguel Elias M. Campista, Luís Henrique M. K. Costa*

Universidade Federal do Rio de Janeiro – GTA/PEE/COPPE

{sadok, miguel, luish}@gta.ufrj.br

Resumo. *Sistemas compartilhados de computação são compostos por vários tipos de recursos, como CPU e memória, e utilizados por usuários com requisitos distintos. Enquanto alguns usuários executam tarefas pequenas, em que a alocação rápida é fundamental, outros executam tarefas longas que requerem mais recursos. Dentre as diferentes propostas para alocar recursos nesse cenário, a justiça de recurso dominante (DRF) se destaca por possuir propriedades fundamentais, como verdade e eficiência de Pareto. Contudo, essas propostas focam apenas em justiça instantânea, ignorando a heterogeneidade de usuários. Este trabalho propõe o DRF com estado (SDRF), que mantém as propriedades fundamentais do DRF, além de garantir um novo conceito de justiça que considera a alocação de recursos passada. O SDRF é verificado por análises teóricas e simulações usando traços de um mês do cluster do Google. Os resultados mostram que o SDRF reduz a espera média dos usuários e melhora a justiça, ao aumentar o número de tarefas completas para usuários com demanda menor, tendo impacto pequeno nos de demanda maior.*

Abstract. *Shared computing systems are composed by different resource types, such as CPU and memory, and hold users with different resource constraints. While some users execute short workloads in which fast allocation is essential, others execute long workloads that require more resources. Among the different proposals to allocate resources in this scenario, Dominant Resource Fairness (DRF) is notable for satisfying some desirable properties, such as truthfulness and Pareto efficiency. However, these proposals focus only on instantaneous fairness, ignoring users heterogeneity. This paper proposes DRF with state (SDRF). SDRF satisfies the fundamental properties of DRF, besides enforcing a new notion of fairness that look at past resource allocations. We verify SDRF with both theoretical analysis and simulations using Google cluster traces. Results show that SDRF reduces users' average waiting time and improves fairness by increasing the number of completed tasks for users with lower demand with low impact on high-demand users.*

1. Introdução

A alocação de recursos é um problema fundamental em qualquer sistema de computação compartilhado. Nuvens e *clusters* costumam ser compartilhados por usuários com diferentes demandas por recursos [Reiss et al., 2012]. A quantidade de recursos

*Este trabalho foi parcialmente financiado pela CAPES, CNPq, FAPERJ e pelos processos nº 15/24494-8 e nº 15/24490-2, da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

alocada para os usuários impacta diretamente o desempenho tanto do ponto de vista de justiça entre eles quanto do de eficiência do sistema [Joe-Wong et al., 2013]. Em sistemas onde há apenas um tipo de recurso, a justiça max-min é a política de alocação mais comum [Radunovic e Le Boudec, 2007]. Ela foi proposta inicialmente como forma de garantir que os fluxos de uma rede recebessem a mesma capacidade nos enlaces [Jaffe, 1981]. Desde então, a justiça max-min vem sendo aplicada, de forma individual, a vários tipos de recursos, incluindo CPU, memória e disco [Ghodsi et al., 2011].

No entanto, quando é necessário alocar vários tipos de recurso ao mesmo tempo, a alocação max-min não garante justiça [Ghodsi et al., 2011, Dolev et al., 2012]. Alguns mecanismos foram propostos para a alocação de vários tipos de recursos [Ghodsi et al., 2011, Dolev et al., 2012, Bonald e Roberts, 2015], sendo o mais prominente, a Justiça de Recurso Dominante (*Dominant Resource Fairness* – DRF) [Ghodsi et al., 2011]. O DRF generaliza a justiça max-min para o cenário com múltiplos tipos de recursos, dando a cada usuário uma parcela igual de seu recurso de maior demanda – o recurso dominante. Usando essa abordagem o DRF consegue obter várias propriedades desejadas. Mesmo com a vasta literatura em justiça de alocação, a maioria das políticas – incluindo o DRF – foca apenas em justiça instantânea ou de pequeno prazo. Porém na prática, as demandas dos usuários são dinâmicas [Reiss et al., 2012] e ignorar esse fato causa injustiça e ineficiência a longo prazo.

Este trabalho propõe o mecanismo de Justiça de Recurso Dominante com Estado (*Stateful Dominant Resource Fairness* – SDRF), que garante justiça a longo prazo em cenários com múltiplos recursos, pois considera o comportamento passado dos usuários. A ideia é fazer com que usuários com baixa utilização média tenham prioridade em relação a usuários com alta utilização média. Quando usado para escalonar tarefas, o SDRF garante que usuários que usam o sistema esporadicamente têm suas tarefas escalonadas mais rapidamente do que os que usam o sistema continuamente. A intuição é que quando um usuário usa mais recursos do que tem direito, ele se compromete a usar menos no futuro, caso um outro usuário precise. O SDRF guarda os comprometimentos e garante que, quando os recursos do sistema são insuficientes, esses comprometimentos são cumpridos.

O SDRF é avaliado usando modelos teóricos e simulações com traços reais de alocação de tarefas em um *datacenter* compartilhado. A avaliação teórica mostra que o SDRF satisfaz as propriedades fundamentais do DRF. Ele é à prova de estratégia, já que os usuários não conseguem melhorar suas alocações ao mentir. O SDRF incentiva a participação, uma vez que nenhum usuário se beneficiaria caso os recursos do sistema fossem divididos igualmente. Além disso, o SDRF possui eficiência estrita de Pareto já que não é possível melhorar a alocação de um usuário sem piorar a de outro. Além da avaliação teórica, o SDRF é posto a prova com uma simulação de larga escala, usando dados de 30 milhões de tarefas ao longo de um mês na nuvem do Google. Os resultados mostram que o SDRF reduz o tempo médio que os usuários esperam para suas tarefas serem escalonadas. Isso melhora a justiça ao aumentar o número de tarefas completadas para usuários de demanda menor, com impacto pequeno nos usuários de demanda maior.

2. Trabalhos Relacionados

A alocação justa de recursos é um tópico de pesquisa extenso. No entanto, o foco geralmente está na alocação de um único tipo de recurso. Ghodsi *et al*. são os primeiros

a investigar o cenário com múltiplos recursos computacionais compartilhados, propondo o DRF [Ghodsi et al., 2011]. Dolev *et al*. propõem uma alternativa ao DRF baseada na chamada "justiça de gargalo" [Dolev et al., 2012], no entanto, além de computacionalmente cara, não é à prova de estratégia [Ghodsi et al., 2012]. Joe-Wang *et al*. estendem a noção de justiça do DRF para desenvolver um modelo que captura o compromisso entre justiça e eficiência [Joe-Wong et al., 2013]. O DRF também é estendido em outros trabalhos para considerar o cenário com servidores heterogêneos [Wang et al., 2014a] e usuários com demandas nulas e pesos diferentes [Parkes et al., 2015]. Embora os trabalhos mencionados considerem múltiplos tipos de recursos, eles ignoram a dinamicidade de demandas.

Bonald e Roberts [Bonald e Roberts, 2015] propõem a Justiça de Gargalo Máximo (BMF), que também não é à prova de estratégia, mas melhora a utilização de recursos em relação ao DRF. Apesar deles considerarem demandas dinâmicas na análise do BMF, a alocação em si considera apenas o uso de curto prazo. Kash *et al*. adaptam o DRF para um cenário dinâmico [Kash et al., 2014], mas assumem que as demandas são constantes. Há ainda trabalhos que adaptam o DRF para a processamento de pacotes [Ghodsi et al., 2012, Wang et al., 2014b] e consideram o passado recente. No entanto, o objetivo é contornar as limitações de escalonamento de pacotes – em que os recursos devem ser compartilhados no *tempo* – e não garantir justiça e eficiência a longo prazo. Finalmente, alguns trabalhos focam em melhorar eficiência de longo prazo mas não a justiça [Grandl et al., 2016, Chen et al., 2017]. Mesmo que alguns trabalhos considerem a dinâmica dos usuários, nenhum deles se preocupa em garantir justiça de longo prazo, objetivo da presente proposta.

3. Modelo do Sistema

O modelo consiste de um conjunto de usuários $\mathcal{N} = \{1, \ldots, n\}$ que compartilham um conjunto de recursos $\mathcal{R} = \{1, \ldots, m\}$. Sem perda de generalidade, normaliza-se o total de cada tipo de recurso no sistema em 1. Assim, se o sistema tem um total de 1000 núcleos de CPU e 100 TB de memória, $0{,}1$ CPU equivale a 100 núcleos, enquanto $0{,}1$ de memória equivale a 10 TB. Cada usuário i tem um vetor de demandas $\boldsymbol{\theta}_i^{(t)} = \langle \theta_{i1}^{(t)}, \ldots, \theta_{im}^{(t)} \rangle$ que representa a demanda do usuário para cada tipo de recurso no instante t. São consideradas demandas positivas para cada recurso, assim em cada instante t, $\theta_{ir}^{(t)} > 0, \forall i \in \mathcal{N}, r \in \mathcal{R}$. A exigência de demandas positivas simplifica o modelo. Contudo, sempre é possível considerar demandas suficientemente próximas de zero.

O mecanismo deve ser capaz de decidir uma alocação baseado nas demandas declaradas pelos usuários. A demanda declarada pelo usuário i no instante t é representada de forma análoga ao vetor de demanda, $\hat{\boldsymbol{\theta}}_i^{(t)} = \langle \hat{\theta}_{i1}^{(t)}, \ldots, \hat{\theta}_{im}^{(t)} \rangle$. Quando os usuários não tentam manipular o mecanismo e anunciam a verdade, $\hat{\boldsymbol{\theta}}_i^{(t)} = \boldsymbol{\theta}_i^{(t)}$. A alocação retornada pelo mecanismo para o usuário i, no instante t e para todos os tipos de recursos é representada pelo vetor $\boldsymbol{o}_i^{(t)} = \langle o_{i1}^{(t)}, \ldots, o_{im}^{(t)} \rangle$. A alocação de todos os usuários do sistema é representada por uma matriz com todos os vetores de alocação individuais $\boldsymbol{O}^{(t)} = \langle \boldsymbol{o}_1^{(t)}, \ldots, \boldsymbol{o}_n^{(t)} \rangle$. Para garantir que as alocações são viáveis, a quantidade de recursos alocada não pode ser maior do que a quantidade presente no sistema, ou seja, para todo instante t, $\sum_{i \in \mathcal{N}} o_{ir}^{(t)} \leq 1, \forall r \in \mathcal{R}$.

As preferências dos usuários são representadas por uma função de utilidade. Dada

uma alocação arbitrária $\boldsymbol{o}_i^{(t)}$, para todo usuário i e tempo t, a função utilidade é

$$u_i^{(t)}(\boldsymbol{o}_i^{(t)}) = \min\left\{\min_{r\in\mathcal{R}}\{o_{ir}^{(t)}/\theta_{ir}^{(t)}\}, 1\right\}. \tag{1}$$

Intuitivamente, os usuários preferem alocações que maximizam a quantidade de tarefas que conseguem alocar, sendo indiferentes em relação a alocações que resultam na mesma quantidade de tarefas. Por exemplo, caso um usuário recebesse 10% do que ele precisa para um dos recursos e 100% para os demais, ele só conseguiria alocar 10% das tarefas. Por isso sua utilidade nesse caso é $0{,}1$.

3.1. Jogo Repetido

Na subseção anterior, várias notações usam um instante t, no entanto, a influência do tempo nas alocações e preferências dos usuários foi omitida. Como é comum em teoria dos jogos, assume-se que em cada instante t há um estágio em que os usuários declaram suas demandas ($\hat{\boldsymbol{\theta}}_i^{(t)}, \forall i \in \mathcal{N}$) e o mecanismo decide uma alocação ($\boldsymbol{o}_i^{(t)}, \forall i \in \mathcal{N}$). A sequência de estágios define o jogo repetido. Para avaliar a utilidade de longo prazo esperada, considera-se que os usuários descontam suas utilidades futuras usando um fator de desconto $\delta_i \in [0, 1)$. Assim, a utilidade de longo prazo esperada do usuário i para um instante t do jogo repetido é dada por

$$u_i^{[t,\infty)} = \mathbb{E}_{u_i}\left[(1-\delta_i)\sum_{k=t}^{\infty}\delta_i^{k-t}u_i^{(k)}(\boldsymbol{o}_i^{(k)})\right]. \tag{2}$$

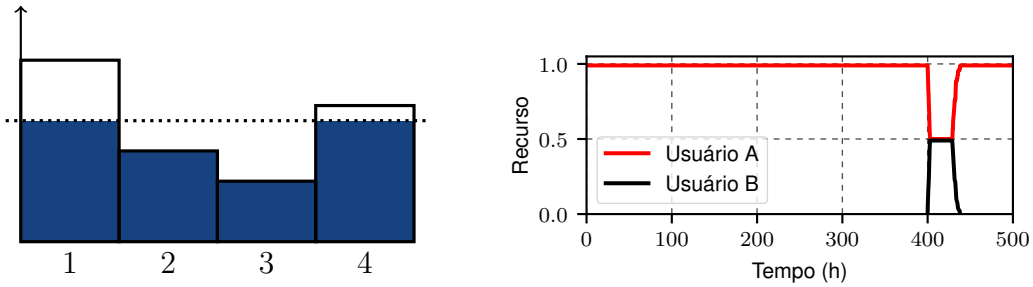
O fator de normalização $(1-\delta_i)$ ajusta a unidade de modo que ela seja comparável com a utilidade nos estágios. O fator de desconto δ_i é comumente referido como a "paciência do usuário". Quanto mais perto de 1, mais o usuário se importa com o futuro, quanto mais perto de 0, mais o usuário se importa com o estágio atual. Caso a utilidade fosse constante nos estágios, a utilidade esperada também seria constante e igual a essa utilidade.

4. DRF com Estado

Primeiro, esta seção introduz a alocação max-min, apresentando suas limitações em garantir justiça ao longo do tempo. Essas limitações motivam o conceito de comprometimento. A partir desse conceito, cria-se o mecanismo de alocação max-min com estado, que considera as alocações passadas a partir dos comprometimentos dos usuários. Finalmente, o SDRF é construído ao expandir a alocação max-min com estado para múltiplos recursos usando o conceito de recurso dominante, a mesma abordagem do DRF.

4.1. Alocação Max-Min

Suponha que exista uma quantidade finita de um recurso a ser dividido entre usuários (p. ex., núcleos de CPU). A forma mais justa é dar uma parcela igual para cada (p. ex., mesmo número de núcleos de CPU). Porém, alguns usuários podem não precisar de tudo o que têm direito, neste caso o excedente é redistribuído entre os demais – esse é o princípio da justiça max-min. Uma das formas de obter a justiça max-min é usando um algoritmo de preenchimento progressivo [Radunovic e Le Boudec, 2007]. O algoritmo aumenta a alocação de recursos gradualmente para todos os usuários, quando



(a) Preenchimento progressivo. **(b)** Alocação ao longo do tempo.

Figura 1. Justiça Max-Min. (a) Diagrama de preenchimento progressivo. (b) Alocação de dois usuários ao longo do tempo.

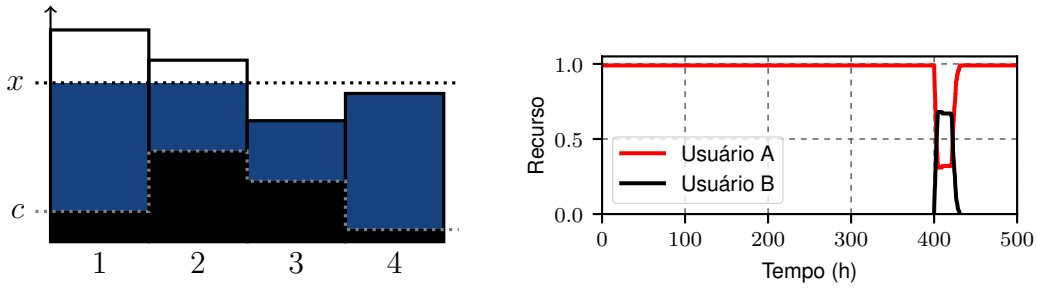
a demanda de um usuário é satisfeita, ele deixa de receber mais recursos e o algoritmo continua aumentando a alocação dos demais. A Figura 1a mostra um diagrama de preenchimento progressivo para uma alocação max-min. Cada coluna (ou tanque) representa a quantidade total do recurso que cada usuário deseja. O recurso é finito e enche os tanques de forma progressiva até que ele acabe. No exemplo, os usuários 2 e 3 têm suas demandas completamente atendidas enquanto os usuários 1 e 4 não.

Embora a alocação max-min seja justa para um cenário estático, quando aplicada a um cenário dinâmico ela não considera o passado e portanto não garante justiça ao longo do tempo. Para ilustrar o problema, considere um exemplo com 2 usuários compartilhando um sistema com escalonador de tarefas max-min (Figura 1b). O usuário A é ávido por recursos e submete uma grande quantidade de tarefas, em contraste com o outro usuário que usa o sistema esporadicamente. Depois de o usuário A já ter usado o sistema por um tempo, o usuário B tem um pico de uso. Mesmo o usuário B nunca tendo usado o sistema, ele recebe a mesma quantidade de recursos que o usuário A. Essa característica privilegia a avidez do usuário A. Caso a alocação do usuário B fosse maior, ele completaria suas tarefas mais rapidamente e o sistema voltaria a ficar livre para o usuário A mais cedo, tendo pouco impacto em a sua carga de trabalho (ver Figura 2b).

4.2. Comprometimentos dos Usuários

Trabalhos anteriores [Ghodsi et al., 2011, Wang et al., 2014a] consideram que os usuários alocam infinitas tarefas com a mesma demanda para cada tipo de recurso. Fazendo isso, somente a parcela de recursos que cada tarefa precisa é considerada, o tempo se torna irrelevante e a alocação equivale à estática. Contudo, na prática, os usuários são heterogêneos e suas demandas dinâmicas [Reiss et al., 2012]. Como forma de distinguir entre usuários que constantemente usam mais recursos do que suas parcelas de direito e usuários que usam o sistema esporadicamente, introduz-se o conceito de comprometimento. O comprometimento é uma medida da propensão dos usuários em usar mais recursos do que suas parcelas de direito.[1] A intuição é que usuários que usam mais recursos que suas parcelas de direito (recursos adicionais) se comprometem a usar menos se outros usuários com perfil de uso inferior à parcela de direito precisarem. Usuários que usam recursos adicionais por pouco tempo devem ter comprometimento menor do que usuários que constantemente precisam de mais recursos. Ademais, usuários que usam

[1]É comum assumir que cada um dos n usuários do sistema tem direito a $1/n$ dos recursos. No entanto, as definições podem ser alteradas para incluir pesos diferentes para a parcela de direito de cada usuário.



(a) Preenchimento progressivo. **(b)** Alocação ao longo do tempo.

Figura 2. Max-Min com estado. (a) Diagrama de preenchimento progressivo com os comprometimentos c. (b) Alocação de dois usuários ao longo do tempo.

poucos recursos a mais devem ter comprometimento menor do que os que usam muitos. Todo usuário $i \in \mathcal{N}$ tem um comprometimento separado para cada recurso $r \in \mathcal{R}$. Os comprometimentos são definidos usando uma média móvel exponencial do uso adicional de recursos. Assim, o comprometimento do usuário i para o recurso r no tempo t é

$$c_{ir}^{(t)} = (1-\delta)\sum_{k=-\infty}^{t} \delta^{t-k}\bar{o}_{ir}^{(k)}, \quad \text{onde } \bar{o}_{ir}^{(k)} = \max\left\{\left(o_{ir}^{(k)} - \frac{1}{n}\right), 0\right\}. \tag{3}$$

O termo $\bar{o}_{ir}^{(k)}$ representa a quantidade adicional de recurso r usada pelo usuário i no tempo k. Quando este termo é zero, o usuário não usou recursos adicionais. Quanto mais no passado os usuários usaram os recursos, menos isso influencia os seus comprometimentos atuais. O parâmetro δ é análogo ao parâmetro δ_i da utilidade de longo prazo esperada (Eq. 2). No entanto, quanto mais próximo de 1, mais o *passado* tem influência.

4.3. Max-Min com Estado

A intuição para o max-min com estado é melhor ilustrada através de um exemplo. *"Se a parcela de direito de cada usuário é 3 CPUs e um usuário tem comprometimento de 1 CPU, então o usuário deve ter direito a receber pelo menos 2 CPUs"*. Essa noção também pode ser implementada usando o algoritmo de preenchimento progressivo, bastando adicionar os comprometimentos como "base para os tanques". A Figura 2a mostra um diagrama para a alocação max-min com estado. As demandas são as mesmas do exemplo da Figura 1a, mas agora há uma base com comprometimentos escolhidos arbitrariamente.

Formalmente, a alocação max-min com estado pode ser representada usando um problema de otimização. Como ela é válida apenas para um único tipo de recurso, o conjunto de recursos se torna um conjunto unitário $\mathcal{R} = \{1\}$ e cada usuário $i \in \mathcal{N}$ possui uma alocação $o_{i1}^{(t)}$ no instante t. O problema é definido como:

$$\begin{aligned} \max_{x} \quad & x \\ \text{sujeito a} \quad & \sum_{i\in\mathcal{N}} o_{i1}^{(t)} \leq 1, \\ & o_{i1}^{(t)} = \max\left\{0, \min\left\{\hat{\theta}_{i1}^{(t)}, (x - c_{i1}^{(t)})\right\}\right\}. \end{aligned} \tag{4}$$

O valor de x encontrado ao resolver a Eq. 4 equivale ao nível do líquido na analogia da Figura 2a. Dado o valor de x, cada usuário recebe uma alocação $o_{i1}^{(t)} =$

$\max\{0, \min\{\hat{\theta}_{i1}^{(t)}, (x - c_{i1}^{(t)})\}\}$, isso garante que as alocações são não-negativas e nunca ultrapassam as demandas. Quando os comprometimentos são nulos, a alocação max-min com estado equivale à max-min. A Figura 2b mostra como é a alocação para o mesmo exemplo da Figura 1b.[2] O usuário B recebe mais recursos e completa suas tarefas mais rapidamente, liberando o sistema de volta para o usuário A. Embora a alocação max-min com estado considere o passado, ela ainda está limitada a um único tipo de recurso.

4.4. Mecanismo SDRF

O SDRF expande a alocação max-min com estado para múltiplos recursos. Para isso, é usado o conceito de recurso dominante do DRF [Ghodsi et al., 2011]. Recursos dominantes são os recursos de maior demanda para um usuário, em relação à quantidade total do recurso no sistema. Como no modelo a quantidade total de todos os recursos do sistema foi normalizada em 1, um recurso $\tilde{r}_i^{(t)}$ é dito dominante para o usuário i no instante t, se $\tilde{r}_i^{(t)} \in \arg\max_{r \in \mathcal{R}} \theta_{ir}^{(t)}$. A partir do recurso dominante é possível definir para cada usuário um vetor de demandas normalizadas, em que as demandas para os recursos dominantes se tornam 1. O vetor de demandas normalizadas para o usuário i no instante t é denotado por $\tilde{\boldsymbol{\theta}}_i^{(t)} = \langle \tilde{\theta}_{i1}^{(t)}, \ldots, \tilde{\theta}_{im}^{(t)} \rangle$, onde $\tilde{\theta}_{ir}^{(t)} = \hat{\theta}_{ir}^{(t)} / \hat{\theta}_{i\tilde{r}_i^{(t)}}^{(t)}$, $\forall i \in \mathcal{N}, r \in \mathcal{R}$.

O DRF expande a alocação max-min para múltiplos recursos, fazendo com que todos os usuários recebam a mesma parcela de seus recursos dominantes. Isso é feito aumentando a parcela de recurso dominante de todos os usuários ao mesmo tempo até que pelo menos um dos recursos do sistema se esgote (o recurso gargalo). Os demais recursos dos usuários aumentam de forma proporcional de acordo com o vetor de demandas normalizadas. Assim, se um recurso possui demanda normalizada de $0{,}5$, ele aumenta com a metade do ritmo do recurso dominante. Da mesma forma que o max-min, se um dos usuários tiver suas demandas satisfeitas ele para de receber recursos.

O SDRF (DRF com estado) generaliza a alocação max-min com estado do mesmo modo que o DRF generaliza a alocação max-min para múltiplos recursos. Como os usuários podem ter comprometimentos diferentes para cada recurso, é definido o conceito de comprometimento dominante. O comprometimento dominante do usuário i no instante t é o maior comprometimento do usuário em relação ao total de recursos no sistema. Como a quantidade total dos recursos do sistema foi normalizada, o comprometimento dominante é simplesmente o maior comprometimento do usuário, $\tilde{c}_i^{(t)} = \max_{r \in \mathcal{R}}\{c_{ir}^{(t)}\}$. Usando o comprometimento dominante é possível definir o SDRF usando ideias do DRF e do max-min com estado. Assim como o DRF, o SDRF aumenta a parcela de recurso dominante para cada usuário até que pelo menos um dos recursos se esgote. Assim como o max-min com estado, os usuários só começam a receber recursos quando x está acima de seus comprometimentos (dominantes). O SDRF é formalmente definido como

$$\begin{aligned} \max_{x} \quad & x \\ \text{sujeito a} \quad & \sum_{i \in \mathcal{N}} o_{ir}^{(t)} \leq 1, \, \forall r \in \mathcal{R}\,, \\ & o_{ir}^{(t)} = \max\left\{0, \min\left\{\hat{\theta}_{ir}^{(t)}, (x - \tilde{c}_i^{(t)}) \cdot \tilde{\theta}_{ir}^{(t)}\right\}\right\}\,. \end{aligned} \tag{5}$$

[2] O exemplo calcula os comprometimentos usando $\delta = 1 - 10^{-6}$, o melhor valor encontrado na Seção 7.

Vale notar as semelhanças entre a Eq. 5 e a Eq. 4. A partir do valor de x encontrado, é possível calcular a alocação de cada usuário e recurso fazendo $o_{ir}^{(t)} = \max\{0, \min\{\hat{\theta}_{ir}^{(t)}, (x - \tilde{c}_i^{(t)}) \cdot \tilde{\theta}_{ir}^{(t)}\}\}$. A seguir as propriedades do SDRF são analisadas.

5. Propriedades do SDRF

Esta seção mostra que, mesmo considerando o passado, o SDRF satisfaz as propriedades fundamentais do DRF. Por limitações de espaço, as demonstrações de todas as proposições foram deixadas para o relatório técnico [Sadok et al., 2017].

O SDRF aumenta as parcelas de recurso dominante de todos os usuários até que um recurso se esgote (recurso gargalo). A proposição a seguir mostra isso.

Proposição 1 (Gargalo)**.** *A alocação obtida pelo SDRF garante que todos os usuários tenham suas demandas atendidas ou exista pelo menos um recurso gargalo. Formalmente,* $\boldsymbol{o}_i^{(t)} = \hat{\boldsymbol{\theta}}_i^{(t)}, \forall i \in \mathcal{N}$ *ou* $\exists r \in \mathcal{R}$ *tal que* $\sum_{i \in \mathcal{N}} o_{ir}^{(t)} = 1$.

Apesar de simples, a Proposição 1 é útil para a demonstração das propriedades seguintes. Uma das propriedades fundamentais do DRF é a verdade (ou à prova de estratégia). Ela garante que os usuários não se beneficiam ao reportar demandas falsas para o mecanismo. As Proposições 2 e 3 mostram que o SDRF também é à prova de estratégia.

Proposição 2 (Verdade no estágio)**.** *Quando os usuários consideram apenas a utilidade no estágio (Eq. 1), o SDRF é à prova de estratégia. Formalmente, se a alocação retornada pelo mecanismo quando o usuário* i *declara a verdade (*$\hat{\boldsymbol{\theta}}_i^{(t)} = \boldsymbol{\theta}_i^{(t)}$*) é denotada por* $\boldsymbol{o}_i^{(t)}$ *e quando ele mente (*$\hat{\boldsymbol{\theta}}_i^{(t)} \neq \boldsymbol{\theta}_i^{(t)}$*) por* $\boldsymbol{o}_i'^{(t)}$*, então* $u_i^{(t)}(\boldsymbol{o}_i^{(t)}) \geq u_i^{(t)}(\boldsymbol{o}_i'^{(t)})$.

Isso mostra que, quando os usuários consideram apenas as utilidades nos estágios, o SDRF é à prova de estratégia. No entanto, um possível receio em considerar as alocações passadas é que isso crie outros incentivos para os usuários manipularem suas demandas declaradas. Existe a possibilidade de que os usuários deixem de usar o sistema quando precisam, na esperança de que isso melhore suas alocações futuras, o que causaria ineficiências. A proposição a seguir mostra que isso não é possível.

Proposição 3 (Verdade no jogo repetido)**.** *Quando os usuários avaliam suas utilidades usando a utilidade de longo prazo esperada (Eq. 2), o SDRF é à prova de estratégia, independentemente dos fatores de desconto usados pelos usuários. Formalmente, se a utilidade de longo prazo esperada quando o usuário* i *declara a verdade (*$\hat{\boldsymbol{\theta}}_i^{(t)} = \boldsymbol{\theta}_i^{(t)}$*) é denotada por* $u_i^{[t,\infty)}$ *e quando ele mente (*$\hat{\boldsymbol{\theta}}_i^{(t)} \neq \boldsymbol{\theta}_i^{(t)}$*) por* $u_i'^{[t,\infty)}$*, então* $u_i^{[t,\infty)} \geq u_i'^{[t,\infty)}$.

É importante também que a alocação do SDRF seja eficiente, o que é demonstrado nas duas proposições a seguir. A Proposição 4 mostra que nenhum recurso é desperdiçado enquanto que a Proposição 5 mostra que a alocação possui eficiência estrita de Pareto. Isso garante que não é possível melhorar a alocação de algum usuário sem piorar a de outro.

Proposição 4 (Sem desperdício)**.** *A alocação* $\boldsymbol{O}^{(t)}$ *obtida pelo SDRF é tal que, se houver uma alocação diferente* $\boldsymbol{O'}^{(t)}$ *em que* $o'^{(t)}_{ir} \leq o_{ir}^{(t)}, \forall i \in \mathcal{N}, r \in \mathcal{R}$ *e para um usuário* $i^* \in \mathcal{N}$ *e recurso* $r^* \in \mathcal{R}$*,* $o'^{(t)}_{i^*r^*} < o_{i^*r^*}^{(t)}$*, então obrigatoriamente* $u_{i^*}^{(t)}(\boldsymbol{o}_{i^*}^{(t)}) > u_{i^*}^{(t)}(\boldsymbol{o'}_{i^*}^{(t)})$*. Em outras palavras, o SDRF não desperdiça recursos.*

Proposição 5 (Eficiência estrita de Pareto)**.** *A alocação obtida pelo SDRF possui eficiência estrita de Pareto, ou seja, o SDRF retorna uma alocação* $\boldsymbol{O}^{(t)}$ *tal que para qualquer outra alocação possível* $\boldsymbol{O'}^{(t)}$*, se existir um usuário* $i \in \mathcal{N}$ *em que* $u_i^{(t)}(\boldsymbol{o}_i'^{(t)}) > u_i^{(t)}(\boldsymbol{o}_i^{(t)})$,

então deve existir um outro usuário $j \in \mathcal{N}$ tal que $u_j^{(t)}(\boldsymbol{o}_j'^{(t)}) < u_j^{(t)}(\boldsymbol{o}_j^{(t)})$.

A última propriedade indica que é vantajoso para os usuários participar do sistema. Isso é feito mostrando que a utilidade que eles recebem é pelo menos tão boa quanto a que eles receberiam sozinhos, caso tivessem acesso a $1/n$ dos recursos do sistema.

Proposição 6 (Racionalidade individual)**.** *A alocação obtida pelo SDRF satisfaz a racionalidade individual. Formalmente, qualquer usuário $i \in \mathcal{N}$ pode usar uma estratégia que faz com que o sistema retorne uma alocação $\boldsymbol{o}_i^{(t)}$ tal que $u_i^{(t)}(\boldsymbol{o}_i^{(t)}) \geq u_i^{(t)}(\langle 1/n, \ldots, 1/n \rangle)$.*

6. Implementação

Esta seção se preocupa com a aplicabilidade do SDRF. Primeiro considera-se o efeito que o tempo contínuo e tarefas indivisíveis têm no modelo definido na Seção 3. Depois é apresentado um algoritmo para alocar tarefas usando o SDRF. No entanto, o algoritmo requer que as prioridades dos usuários sejam reordenadas antes de cada tarefa escalonada. Para mitigar esse problema é introduzida a árvore viva, uma nova estrutura de dados que mantém ordenados elementos com prioridades dinâmicas.

6.1. Tempo Contínuo

O modelo definido na Seção 3 assume que o tempo avança com uma sequência de jogos repetidos, o que sugere um tempo discreto. A definição de comprometimento na Eq. 3 é compatível com essa noção. No entanto, em um sistema real, tarefas podem chegar e terminar a qualquer momento. Logo, é necessária uma expressão para calcular os comprometimentos em tempo contínuo. Primeiro, a Eq. 3 é redefinida de forma recursiva usando uma equação a diferenças, $c_{ir}^{(t)} = (1-\delta)\bar{o}_{ir}^{(t)} + \delta c_{ir}^{(t-\Delta t)}$ onde os comprometimentos no tempo t podem ser calculados a partir dos comprometimentos no tempo $t - \Delta t$. Isso assume que $\bar{o}_{ir}^{(t)}$ permanece constante no intervalo $(t - \Delta t, t]$. Feito isso, o comprometimento pode ser aproximado em tempo contínuo [Oppenheim et al., 1999], resultando em

$$c_{ir}^{(t)} = (1-\hat{\delta})\bar{o}_{ir}^{(t)} + \hat{\delta} c_{ir}^{(t_0)}, \quad \hat{\delta} = e^{-(t-t_0)/\tau}, \quad \tau = -\Delta t / \ln(\delta) \tag{6}$$

onde é possível calcular $c_{ir}^{(t)}$ a partir de qualquer $c_{ir}^{(t_0)}$ contanto que $\bar{o}_{ir}^{(t)}$ permaneça constante de t_0 até t. Como o_{ir} só muda quando uma tarefa para o usuário i que usa o recurso r começa ou termina, a expressão é útil na prática.

6.2. Tarefas Não Divisíveis

Até agora foi assumido que as tarefas são divisíveis de forma arbitrária. Isso permite fornecer quantidades de recursos arbitrariamente pequenas a essas tarefas. Contudo, na prática as tarefas costumam não ser divisíveis [Hindman et al., 2011]. Para escalonar tarefas não divisíveis é usada a mesma abordagem de trabalhos anteriores [Ghodsi et al., 2011, Wang et al., 2014a]: aplicar o preenchimento progressivo na alocação de tarefas. O Algoritmo 1 resume o procedimento de alocação de tarefas. É definido um conjunto A de usuários ativos (usuários com pelo menos uma tarefa esperando ser escalonada) e guarda-se a quantidade total de cada tipo de recurso fornecida a cada usuário. Se o sistema não está cheio e existe pelo menos uma tarefa com escalonamento pendente, escalona-se a próxima tarefa do usuário com a menor parcela de recurso dominante, compensada pelos comprometimentos.

Algoritmo 1 Escalonamento de Tarefas com o SDRF

$A = \{1, \ldots, k\}$ ▷ conjunto de usuários ativos
$\boldsymbol{o}_i = \langle o_{i1}, \ldots, o_{im} \rangle, \forall i \in A$ ▷ total de recursos fornecidos para o usuário i
$\boldsymbol{c}_i = \langle c_{i1}, \ldots, c_{im} \rangle, \forall i \in A$ ▷ comprometimentos do usuário i
enquanto $A \neq \emptyset$ **faça**
  $i \leftarrow \arg\min_{i \in A} \left\{ \max_{r \in \mathcal{R}} \{o_{ir} + c_{ir}\} \right\}$ ▷ escolhe o usuário
  $\forall r, d_{ir} \leftarrow$ demanda por r na próxima tarefa do usuário i
  se $\forall r, \left(d_{ir} + \sum_{j \in \mathcal{N}} o_{jr} \right) \leq 1$ **então** ▷ há espaço para alocar a tarefa
    $\forall r, o_{ir} \leftarrow o_{ir} + d_{ir}$
    se não existem mais tarefas pendentes para o usuário i **então**
      remova i de A
  senão
    retorne ▷ o sistema está cheio

Sempre que uma tarefa chega ou termina, o Algoritmo 1 é executado novamente usando o conjunto A e os vetores $\boldsymbol{o}_i$, $\boldsymbol{c}_i$, $\forall i \in A$, atualizados. Quanto menores forem as tarefas, mais o Algoritmo 1 se aproxima do problema de otimização da Eq. 5.

A alocação rápida é essencial em um escalonador de tarefas. Em horários de pico, um escalonador pode precisar tomar centenas de decisões por segundo [Reiss et al., 2012]. A parte mais custosa do Algoritmo 1 é a escolha do usuário. Enquanto o DRF puro pode ser implementado usando uma fila de prioridades, que guarda a parcela de recurso dominante para cada usuário[3], isso não é possível para o SDRF. No DRF as prioridades só mudam quando $\boldsymbol{o}_i$ muda, no SDRF os comprometimentos mudam a cada instante e consequentemente as prioridades também. Recalcular as prioridades para cada usuário e recurso em todas as decisões de escalonamento de tarefas seria muito custoso. A próxima subseção mostra como resolver esse problema.

6.3. Árvore Viva

Ao escalonar tarefas, é necessário saber qual usuário tem a *maior prioridade*, o valor exato dos comprometimentos pouco importa. A árvore viva é uma estrutura de dados que mantém ordenados elementos com prioridades que variam de forma previsível. A ideia é focar em eventos de troca de posição, ao invés das prioridades em si. Quando as prioridades variam conforme uma função contínua, os elementos trocam de posição sempre que suas prioridades se interceptam. Uma árvore viva sempre tem um *tempo atual* associado e, para esse tempo, ela garante que os elementos estão ordenados. Sempre que o tempo atual é atualizado, ao invés de recalcular as prioridades de todos os elementos, a árvore viva verifica se houve algum evento de troca de posição desde a última atualização.

A árvore viva pode ser vista como a combinação de um vetor e duas árvores rubro-negras [Guibas e Sedgewick, 1978] (ver Figura 3). Uma das árvores é chamada de árvore de elementos, já que ela mantém os elementos ordenados por suas prioridades, enquanto a outra é chamada de árvore de eventos, já que ela guarda os eventos de troca de posição em ordem de tempo. O vetor é usado para a busca de elementos. Por simplicidade, assume-se que cada elemento possui um identificador inteiro único que pode ser usado como

[3] A implementação do DRF no Mesos [Hindman et al., 2011] usa um `std::set` da biblioteca padrão do C++, o qual é geralmente implementado usando uma árvore binária.

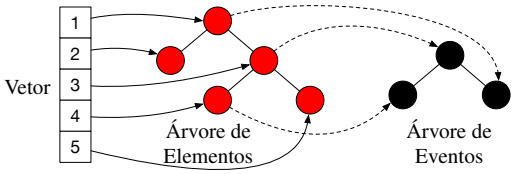


Figura 3. Ilustração de uma árvore viva. Posições no vetor apontam para elementos na árvore de elementos. Elementos que trocarão de ordem com o sucessor apontam para eventos na árvore de eventos.

índice do vetor[4]. Cada posição do vetor tem um ponteiro para um elemento na árvore de elementos (ou NIL, se não houver um elemento associado ao índice). Isso permite acessar elementos pelo identificador com complexidade $O(1)$. Quando dois elementos vizinhos devem trocar de posição no futuro, o elemento que vem antes possui um ponteiro para um evento de troca de posição na árvore de eventos.

Para determinar se dois elementos vizinhos trocarão de posição no futuro basta calcular a interseção entre suas funções de prioridade. Caso a interseção aconteça em um tempo futuro, um evento é adicionado à árvore de eventos com o identificador do elemento e o instante de interseção (usado como prioridade na árvore). No caso do SDRF, em que a árvore viva é usada na etapa de escolha do usuário do Algoritmo 1, os elementos são ordenados em ordem crescente de $\max_{r \in \mathcal{R}} \{o_{ir} + c_{ir}\}$, onde c_{ir} é calculado usando a Eq. 6.

7. Resultados

Esta seção avalia o desempenho do SDRF e da árvore viva usando simulações baseadas em traços do *cluster* do Google [Reiss et al., 2012]. Os traços contêm informação sobre a carga de trabalho de engenheiros e serviços do Google, compartilhando o mesmo *cluster* ao longo de um mês. Os usuários (ou serviços), submetem conjuntos de tarefas chamados de trabalhos. Os traços contêm eventos para todos os instantes em que uma tarefa é submetida, escalonada, ou termina. A partir desses eventos pode-se extrair as informações de demanda de CPU e memória, assim como os tempos de submissão e execução das tarefas. Esses dados são usados como entrada de uma simulação de eventos discretos[5]. Tarefas com demanda nula ou que foram canceladas pelo sistema do Google foram removidas, mas tarefas que falharam devido a erros dos usuários foram mantidas. Depois disso sobram 32 milhões de tarefas de 627 usuários.

As simulações são feitas para diferentes valores de δ e carga do sistema. Os valores de δ são relativos a um Δt de 1s (ver Eq. 6). O δ é variado de forma a deixá-lo exponencialmente mais perto de 1, $\delta = 1 - 10^{-1}, \ldots, 1 - 10^{-7}$. Isso equivale a aumentar τ exponencialmente. Para verificar o desempenho do SDRF para diferentes níveis de carga do sistema, varia-se a quantidade total de recursos no sistema. Como base para esse valor calcula-se a média de utilização do sistema nos traços do Google, chamando esse valor de R. A partir disso são feitas simulações variando o valor total de recursos no sistema de 50% de R até 100% de R, em passos de 10%.

[4]Quando os elementos não possuem identificadores inteiros, ou esses são muito esparsos, o vetor pode ser substituído por uma tabela *hash* e ainda apresentar buscas com complexidade $O(1)$ amortizada.

[5]Os códigos do simulador de eventos discretos, do SDRF e da árvore viva podem ser obtidos em `https://www.gta.ufrj.br/~sadok/codes/sdrf/`.

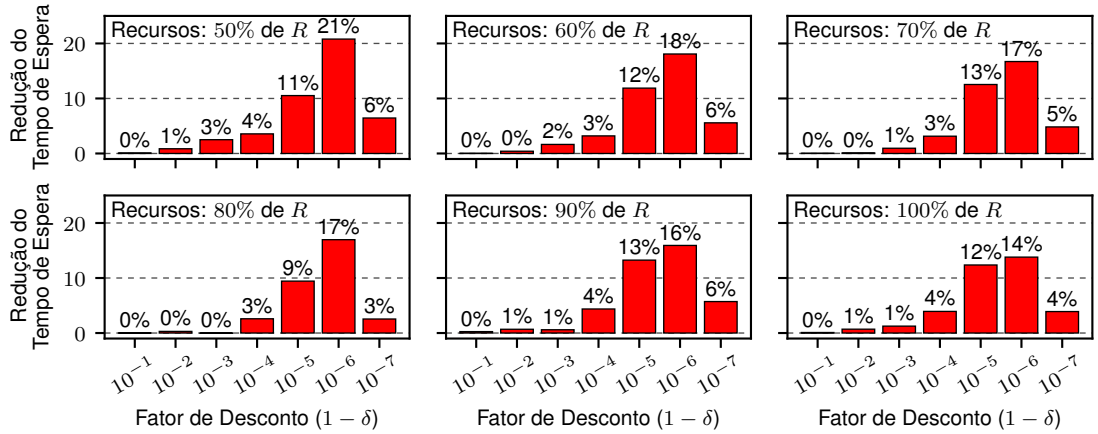


Figura 4. Redução no tempo médio de espera dos usuário em relação ao DRF.

A Figura 4 mostra a média de redução de espera dos usuários em relação ao DRF. São mostrados os resultados para simulações com diferentes valores de δ e carga do sistema. Em geral, diferenças de justiça são mais evidentes quando há menos recursos no sistema [Joe-Wong et al., 2013], o que também pode ser verificado na Figura 4. Quando δ é pequeno o suficiente, o SDRF tem desempenho equivalente ao DRF. Além disso, quando δ é suficientemente próximo de 1, o SDRF se aproxima do DRF. Isso é esperado ao se inspecionar a Eq. 6: para δ suficientemente próximo de 1, os comprometimentos nunca acumulam, já para δ suficientemente próximo de 0, os comprometimentos passam a ser simplesmente as últimas alocações, fazendo com que as tarefas sejam alocadas como no DRF. A maior redução no tempo de espera acontece quando $\delta = 1 - 10^{-6}$, para todos os níveis de carga do sistema avaliados. Para esse caso, o SDRF é superior ao DRF em mais de 10% em todos os cenários avaliados.

Além do tempo de espera, também avalia-se a quantidade de tarefas que cada usuário consegue completar. Para isso, calcula-se a razão de tarefas completadas para cada usuário (o número de tarefas completadas dividido pelo número de tarefas submetidas) tanto para o DRF quanto para o SDRF. A Figura 5 mostra os resultados para a simulação com $\delta = 1 - 10^{-6}$ e o total de recursos 50% de R. Cada bolha representa um usuário diferente sendo o tamanho logarítmico em relação ao número de tarefas submetidas pelo usuário. Quando a bolha está acima da reta $y = x$, o usuário consegue completar mais tarefas quando se usa o SDRF do que quando se usa o DRF. A maioria dos usuários completa mais tarefas com o SDRF, com apenas 9 usuários completando menos. Além disso, mesmo esses usuários tendo completado menos tarefas, suas razões de tarefas completadas tiveram impacto pequeno.

A seguir, a árvore viva é avaliada quando usada para implementar o SDRF nas mesmas simulações. A cada tarefa alocada, a árvore viva é atualizada para o instante atual. Quando isso ocorre alguns elementos são trocados de posição. Quando não há eventos a serem executados, a atualização acontece em $O(1)$. No extremo oposto, quando há eventos de troca para todos os elementos, todos devem ser reinseridos na árvore e o tempo é $O(n \log n)$. A Figura 6 mostra o número de eventos executados pela árvore viva durante o período completo de simulação, para todas as simulações. Cada curva representa um valor diferente de quantidade de recursos no sistema, 50% de R até 100% de R, de cima para baixo. O número de eventos aumenta quando a quantidade de recursos

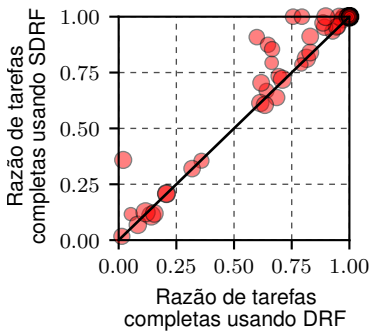


Figura 5. Razão de tarefas completadas usando o DRF e o SRDF. Cada bolha é um usuário diferente. Usuários acima da reta $y = x$ estão melhores com o SDRF.

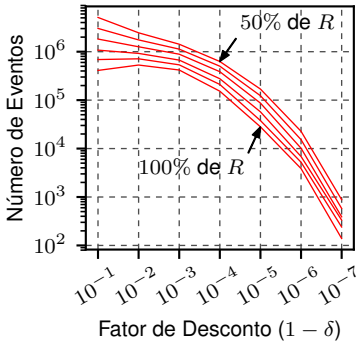


Figura 6. Eventos na árvore viva para valores diferentes de fator de desconto e recursos do sistema (50% até 100% de R de cima para baixo).

no sistema diminui. Além disso, quanto mais δ se aproxima de 1, menos eventos são observados. Isso é consistente já que os comprometimentos variam mais devagar quando δ se aproxima de 1. Quando $\delta = 1 - 10^{-6}$ e o total de recursos 50% de R, há um total de $22\,718$ eventos, o que corresponde a aproximadamente 7 eventos para cada 1000 tarefas escalonadas. Mesmo para o pior cenário ($\delta = 1 - 10^{-1}$, 50% de R) o número total de eventos é $5\,094\,167$ o que corresponde a aproximadamente 2 eventos para cada 10 tarefas escalonadas. Se o desempenho da árvore viva fosse próximo ao pior caso, ela ofereceria pouca vantagem em relação a reordenar todos os usuários a cada atualização. No entanto, com o número de atualizações observadas, as operações da árvore viva têm desempenho próximo aos de uma árvore rubro-negra onde os pesos dos elementos são estáticos.

8. Conclusão e Trabalhos Futuros

Este trabalho introduziu o SDRF, um mecanismo de alocação de múltiplos recursos que considera as alocações passadas e dá preferência aos usuários com menor uso médio. Primeiro foi introduzida a alocação max-min com estado, a qual foi expandida para múltiplos recursos usando o conceito de recurso dominante. Foi mostrado que o SDRF satisfaz as propriedades fundamentais do DRF além de garantir justiça de longo prazo. Para implementar o SDRF de forma eficiente foi introduzida a árvore viva, uma nova estrutura de dados que mantém ordenados elementos com prioridades que variam de forma previsível. Simulações com o SDRF mostraram que o SDRF reduz o tempo médio que os usuários esperam para as suas tarefas serem escalonadas. Além disso, ele aumenta o número de tarefas completadas para usuários de demanda menor com baixo impacto nos usuários de demanda maior. As mesmas simulações foram usadas para avaliar a árvore viva, mostrando que o SDRF pode ser implementado de forma eficiente.

Existem várias direções para investigação futura. O SDRF foi avaliado para a alocação de tarefas mas é possível que ele seja aplicável a outros cenários. Além disso, embora a possibilidade de usar pesos diferentes para cada usuário tenha sido mencionada, ela não foi avaliada formalmente. Uma outra investigação possível é o uso de funções diferentes para medir os comprometimentos. Além do SDRF, acredita-se que a árvore viva possa beneficiar outras aplicações, um exemplo é uma possível adaptação do algoritmo de Dijkstra para grafos com pesos que variam com o tempo.

Referências

Bonald, T. e Roberts, J. (2015). Multi-resource fairness: Objectives, algorithms and performance. Em *Proc. ACM SIGMETRICS*.

Chen, C., Wang, W., Zhang, S. e Li, B. (2017). Cluster fair queueing: Speeding up data-parallel jobs with delay guarantees. Em *Proc. IEEE INFOCOM*.

Dolev, D., Feitelson, D. G., Halpern, J. Y., Kupferman, R. e Linial, N. (2012). No justified complaints. Em *Proc. Conference on Innovations in Theoretical Computer Science*.

Ghodsi, A., Sekar, V., Zaharia, M. e Stoica, I. (2012). Multi-resource fair queueing for packet processing. Em *Proc. ACM SIGCOMM*.

Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S. e Stoica, I. (2011). Dominant resource fairness: Fair allocation of multiple resource types. Em *Proc. USENIX NSDI*.

Grandl, R., Chowdhury, M., Akella, A. e Ananthanarayanan, G. (2016). Altruistic scheduling in multi-resource clusters. Em *Proc. USENIX OSDI*.

Guibas, L. J. e Sedgewick, R. (1978). A dichromatic framework for balanced trees. Em *Proc. IEEE Symposium on Foundations of Computer Science*.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S. e Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. Em *Proc. USENIX NSDI*.

Jaffe, J. (1981). Bottleneck flow control. *IEEE Trans. on Communic.*, 29(7):954–962.

Joe-Wong, C., Sen, S., Lan, T. e Chiang, M. (2013). Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework. *IEEE/ACM ToN*, 21(6):1785–1798.

Kash, I., Procaccia, A. D. e Shah, N. (2014). No agent left behind: Dynamic fair division of multiple resources. *Journal of Artificial Intelligence Research*, 51(1):579–603.

Oppenheim, A. V., Schafer, R. W. e Buck, J. R. (1999). *Discrete-Time Signal Processing*. Prentice-Hall, 2 edição.

Parkes, D. C., Procaccia, A. D. e Shah, N. (2015). Beyond dominant resource fairness. *ACM Transactions on Economics and Computation*, 3(1):3:1–3:22.

Radunovic, B. e Le Boudec, J.-Y. (2007). A unified framework for max-min and min-max fairness with applications. *IEEE/ACM ToN*, 15(5):1073–1083.

Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H. e Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. Em *Proc. ACM SoCC*.

Sadok, H., Campista, M. E. M. e Costa, L. H. M. K. (2017). O passado também importa: Um mecanismo de alocação justa de múltiplos tipos de recursos ao longo do tempo. Relatório Técnico GTA-17-30, GTA/PEE/UFRJ. `https://www.gta.ufrj.br/ftp/gta/TechReports/SCC17a.pdf`.

Wang, W., Li, B. e Liang, B. (2014a). Dominant resource fairness in cloud computing systems with heterogeneous servers. Em *Proc. IEEE INFOCOM*.

Wang, W., Liang, B. e Li, B. (2014b). Low complexity multi-resource fair queueing with bounded delay. Em *Proc. IEEE INFOCOM*.