Escalonamento justo em infraestruturas de nuvem com múltiplas classes de serviço

Giovanni Farias[1], Raquel Lopes[1], Francisco Brasileiro[1], Marcus Carvalho[2]
Fabio Morais[2], João Mafra[1], Daniel Turull[3]

[1]Universidade Federal de Campina Grande (UFCG), Campina Grande, PB, Brazil

{giovanni, jvmafra}@lsd.ufcg.edu.br, {raquel, fubica}@dsc.ufcg.edu.br

[2]Universidade Federal da Paraiba (UFPB), Rio Tinto, PB, Brazil

{marcuswac, fabio}@dcx.ufpb.br

[3]Ericsson Research, Stockholm, Sweden

daniel.turull@ericsson.com

Abstract. *Cloud computing providers offer multiple service classes to deal with workload heterogeneity. Classes are distinguished by their expected Quality of Service (QoS), which is defined in terms of Service Level Objectives (SLO). A priority-based scheduling policy is commonly used to guarantee that requests submitted to the different service classes achieve the desired QoS. However, the QoS delivered during resource contention periods may be unfair to certain users. In this paper, we present an SLO-driven scheduling policy which takes the SLOs and actual QoS delivered for each request into account when making decisions. We used simulation experiments fed with traces from a production system to compare the SLO-driven policy with a priority-based one. In general, the SLO-driven policy delivered a better service than the priority-based one.*

Resumo. *Provedores de computação na nuvem oferecem múltiplas classes de serviço para lidar com a heterogeneidade da carga de trabalho. Essas classes são diferenciadas pela Qualidade de Serviço (QoS) esperada, que é definida em termos de Objetivos de Níveis de Serviço (SLO). Um Escalonamento Baseado em Prioridade (EBP) é uma forma comum de permitir que requisições por recursos de diferentes classes alcancem a QoS desejada. No entanto, em períodos de contenção de recursos, a QoS oferecida pode ser injusta para certos usuários. Neste trabalho, apresenta-se o Escalonamento Dirigido por SLO (EDSLO), que escalona com base nos SLOs e na QoS de cada requisição em um dado momento. Experimentos de simulação usando dados de um sistema real em produção foram executados para comparar as diferentes políticas. Em geral, o EDSLO entrega um serviço melhor que o EBP.*

1. Introdução

Um dos principais desafios para provedores de Computação na Nuvem é lidar com complexas infraestruturas de computação de larga escala e cargas de trabalho heterogêneas que variam ao longo do tempo (Verma et al. 2014). Nesse cenário, os provedores podem oferecer múltiplas classes de serviço para satisfazer maior quantidade de

usuários (Carvalho et al. 2015). Cada uma dessas classes possui um esquema diferente de precificação e uma qualidade de serviço (QoS, do inglês *Quality of Service*) mínima esperada. Geralmente, a QoS da classe é definida através de metas estabelecidas pelo provedor através dos objetivos de níveis de serviço (SLO, do inglês *Service Level Objectives*). Os provedores definem SLOs para várias métricas de QoS e se esforçam para cumpri-las. Assim sendo, classes de serviço podem ser determinadas de acordo com diferentes combinações de SLOs. O gerenciamento eficiente de recursos permite que o provedor possa satisfazer os SLOs para as diferentes classes.

As atividades relacionadas ao gerenciamento de recursos de nuvem podem ser divididas em três etapas (Carvalho et al. 2015): *i) Planejamento de Capacidade*: define a quantidade de recursos adequada para que o provedor consiga atender a demanda esperada; *ii) Controle de Admissão*: decide quais requisições devem ser rejeitadas para que as requisições admitidas tenham maiores chances de satisfazer seus SLOs; e *iii) Escalonamento*: escolhe quais requisições devem estar executando ao longo do tempo e quais servidores devem prover recursos para as mesmas. Este trabalho concentra-se na etapa de escalonamento. Ademais, os recursos requisitados por usuários são empacotados usando abstrações de isolamento como máquinas virtuais ou *containers*. Daqui em diante, o conjunto de recursos solicitados em cada requisição será referenciado como *instância*, independente de como são empacotados pelo provedor. Portanto, o escalonador tentará atender cada requisição alocando a instância a ela associada.

Alguns escalonadores diferenciam suas classes de serviço através da associação de uma prioridade para cada uma delas (Boutin et al. 2014; Delimitrou et al. 2015; Isard et al. 2009; Karanasos et al. 2015; Schwarzkopf et al. 2013; Vavilapalli et al. 2013; Verma et al. 2015). Nesses casos, classes que oferecem uma QoS melhor são mais importantes que classes que oferecem uma QoS menor. Quanto maior for a QoS prometida para a classe, maior será a prioridade a ela atribuída. Quando os recursos disponíveis não são suficientes para atender todas as requisições, o escalonador pode preemptar instâncias de menor prioridade para alocar instâncias de maior prioridade. Dessa forma, instâncias com prioridades mais altas têm menos chances de serem preemptadas e, consequentemente, têm maior probabilidade de obter QoS melhor que aquela obtida pelas instâncias com menor prioridade. Pelo fato desses escalonadores levarem em consideração a prioridade das requisições para decidir quais delas terão recursos alocados a si em um determinado instante de tempo, eles são classificados como "Baseados em Prioridade".

Apesar de amplamente considerado como um tipo de escalonador propício para sistemas que oferecem múltiplas classes de serviço, o Escalonamento Baseado em Prioridade (EBP) pode ser ineficiente em situações onde há contenção de recursos. Primeiramente, as instâncias com prioridades mais baixas têm maiores chances de não receberem a QoS esperada. Isso se deve à possibilidade delas sempre serem preemptadas para liberar recursos para instâncias com prioridades mais altas, mesmo quando essas de maior prioridade já estiverem recebendo QoS acima da prometida. Além disso, dado que uma instância não preempta outra de mesma classe, a QoS entregue para diferentes instâncias de mesma prioridade pode apresentar uma alta variabilidade.

Neste trabalho, uma nova abordagem de escalonamento é proposta: *Escalonamento Dirigido por SLO* (EDSLO). Essa abordagem toma decisões com base nos SLOs definidos para as classes e na QoS entregue para cada instância em um dado momento.

Seu principal objetivo é preemptar instâncias que estejam experimentando uma QoS além do que lhes foi prometida em benefício de outras que estejam experimentando uma QoS abaixo de suas respectivas metas. Isso deve ocorrer independente das classes das requisições. Além disso, o EDSLO promove justiça entre instâncias de mesma classe minimizando a variância da QoS oferecida em momentos com contenção de recursos — minimizando a diferença entre a QoS entregue para uma instância e sua respectiva meta.

Neste estudo, a disponibilidade da instância — o percentual de tempo que uma instância está em execução desde a admissão de sua respectiva requisição até o tempo atual — é considerada como a métrica de QoS de interesse. Esta escolha é dada pelo fato de a disponibilidade ser uma das principais preocupações de consumidores de computação na nuvem (Shahrad and Wentzlaff 2016), como também estar presente em 73% dos Acordos de Nível de Serviço (SLA, do inglês *Service Level Agreements*) negociados entre consumidores e provedores (Pan et al. 2013). Portanto, este trabalho considera que existem SLOs de disponibilidade associados às requisições e a meta de disponibilidade muda de acordo com a classe de serviço solicitada.

A avaliação do EDSLO se deu comparando-o com um EBP. Para isso, foram utilizados modelos de simulação alimentados com rastros de execução de um *cluster* em produção do Google (Wilkes 2011; Reiss et al. 2012). Os resultados mostram que, comparado com o EBP, a política proposta é capaz de entregar QoS satisfazendo os SLOs de todas as requisições em uma parcela significantemente maior dos cenários. Além disso, considerando os cenários onde houve violações de SLO para os dois escalonadores, o EDSLO entrega QoS mais próxima da pretendida em quase todos os cenários.

O resto do artigo está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. Em seguida, a Seção 3 descreve a política de escalonamento dirigido por SLO. A Seção 4 detalha a metodologia adotada para avaliação da política proposta. Na Seção 5, os resultados obtidos dos experimentos de simulação e medição são apresentados e discutidos, e, por último, as conclusões deste estudo são apresentadas na Seção 6.

2. Trabalhos Relacionados

A heterogeneidade da carga de trabalho torna a tarefa de escalonamento bastante desafiadora em ambientes de provedores de computação na nuvem. Os algoritmos de escalonamento propostos para *clusters* de larga escala podem ser classificados de acordo com suas arquiteturas: (i) *Escalonadores monolíticos*, que tomam decisões de forma centralizada, considerando o *cluster* como um todo (Gog et al. 2016; Isard et al. 2009; Vavilapalli et al. 2013); (ii) *Escalonadores em dois níveis*, onde um único gerenciador de recursos com informações sobre o estado do *cluster* oferece recursos para múltiplos escalonadores (Hindman et al. 2011); (iii) *Escalonadores distribuídos*, que permite a existência de diferentes escalonadores executando de forma independente sem informações sobre o *cluster* (Ousterhout et al. 2013) ou permite o compartilhamento dessas informações entre os múltiplos escalonadores (Boutin et al. 2014; Schwarzkopf et al. 2013; Verma et al. 2015); e, por último, (iv) *Escalonadores híbridos*, onde existe um escalonador centralizado que toma decisões para uma parte da carga e escalonadores distribuídos que lidam com o restante da carga (Delgado et al. 2015; Karanasos et al. 2015). O EDSLO apresentado neste trabalho pode ser classificado como monolítico, embora ele também possa ser implementado de forma distribuída.

Para oferecer múltiplas classes de serviço, algumas políticas de escalonamento associam prioridades distintas para cada uma das classes e permitem preempções de instâncias. Alguns dos escalonadores analisados consideram uma abordagem preemptiva (Boutin et al. 2014; Gog et al. 2016; Isard et al. 2009; Karanasos et al. 2015; Schwarzkopf et al. 2013; Vavilapalli et al. 2013; Verma et al. 2015). Na política de escalonamento proposta neste artigo, preempções também podem ocorrer. Porém, decisões sobre preempções são tomadas tendo como base os SLOs e a QoS realmente entregue para as requisições em um dado momento (ao invés de considerar apenas uma medida indireta dessa QoS, i.e. prioridades).

Algumas políticas de escalonamento também consideram SLOs na tomada de decisões (Delimitrou and Kozyrakis 2013; Delimitrou and Kozyrakis 2014; Goiri et al. 2010; Kong et al. 2011). Entretanto, essas políticas consideram os SLOs definidos para as aplicações que executam nas instâncias, enquanto que a política do EDSLO considera o SLO solicitado pelos usuários ao provedor, independente do tipo de aplicação que executa nas instâncias. Já o trabalho de Shahrad e Wentzlaff (Shahrad and Wentzlaff 2016) propõe um modelo onde os consumidores podem expressar um SLO de disponibilidade ao invés de ter que escolher um SLO dentre alguns pré-estabelecidos pelo provedor. A satisfação dos SLOs é considerada em intervalos de tempo que coincidem com os ciclos de bilhetagem. Uma instância é alocada após o início do ciclo até o tempo necessário para que tenha seu SLO satisfeito durante esse ciclo. Nesse momento a instância é preemptada (liberando recursos para outras), podendo voltar a executar apenas no início do próximo ciclo. Esse modelo é mais restrito e pode levar a alta variabilidade na QoS entregue para requisições com os mesmos SLOs. Por outro lado, o EDSLO busca provisionar recursos de forma justa ao longo de todo o tempo, e não apenas dentro de uma janela pré-definida. Assim sendo, preempções podem ocorrer a qualquer momento com base nos SLOs ou na QoS entregue para as instâncias.

3. Escalonamento Dirigido por SLO - EDSLO

Essencialmente, a missão do escalonador de um provedor é decidir, ao longo do tempo, quais das requisições devem ter suas instâncias alocadas e em quais dos servidores. Caso não haja recursos suficientes para alocar todas as instâncias, então algumas das requisições devem ser mantidas em uma fila de requisições pendentes. Nesse cenário, o escalonador pode precisar preemptar uma instância que já esteja alocada para liberar recursos em benefício de uma (ou mais) das requisições pendentes na fila.

O EDSLO é ciente da classe de serviço das instâncias e da QoS entregue para cada uma delas a cada instante de tempo. Dessa forma, o escalonador toma decisões considerando, além da QoS entregue para cada instância, os SLOs das diferentes classes. Assim, durante momentos de contenção de recursos, instâncias com QoS acima de suas metas são preemptadas em benefício de outras com QoS abaixo do SLO ou mais próximas de violá-lo. Isso deve ocorrer mesmo que essas últimas sejam de classes com metas de QoS mais baixas que as primeiras. Assim, o principal objetivo da política proposta é trazer mais justiça ao escalonamento (instâncias de mesma classe recebem QoS semelhantes), sobretudo quando a QoS prometida não pode ser garantida para todas as instâncias.

Neste trabalho considera-se a disponibilidade da instância como métrica de QoS de interesse. Todavia, considerar apenas a disponibilidade no momento da decisão sobre

preempção não é adequado. Uma vez que uma instância é preemptada, o tempo que levará para a disponibilidade desta instância ficar abaixo do seu SLO depende não só da sua disponibilidade atual, mas também do tempo em que ela esteve alocada. Por exemplo, uma instância que tem 95% de disponibilidade e um tempo de alocação acumulado de 1 hora pode ficar pendente por até 3,5 minutos antes que sua disponibilidade fique abaixo de um SLO de 90%, enquanto uma outra com a mesma disponibilidade, porém com apenas 10 minutos de tempo de alocação, violaria o mesmo SLO de 90% de disponibilidade apenas 35 segundos após ser preemptada. Por esta razão, o EDSLO toma decisões considerando uma métrica chamada de tempo-para-violar (TTV, do inglês *Time-To-Violate*) de uma instância. Esta métrica é computada para cada instância (em execução ou pendente), e indica por quanto tempo uma requisição poderia ficar pendente até que seu SLO de disponibilidade fosse violado. Formalmente, o TTV da instância da requisição j no tempo t é calculado com base no SLO de disponibilidade o_i da classe de serviço i solicitada por j, no tempo acumulado em que a instância de j esteve alocada a_j^t, e no tempo acumulado em que j esteve pendente na fila p_j^t, conforme Equação 1:

$$\tau_j^t = \frac{a_j^t}{o_i} - (a_j^t + p_j^t). \tag{1}$$

Muitos eventos podem desencadear a execução do escalonador. Quando uma nova requisição é admitida, o escalonador inicia seu TTV com zero e a insere na fila de pendentes. Da mesma forma, quando um servidor falha ou é retirado da infraestrutura, todas as instâncias alocadas no mesmo terão suas requisições adicionadas na fila de pendentes. O escalonador também é chamado quando: (a) uma requisição completa sua execução; e (b) quando um servidor recupera-se de falha ou novas máquinas são adicionadas à infraestrutura. Por fim, como o TTV das instâncias sofre alteração ao longo do tempo, o escalonador também deve ser executado periodicamente independente desses eventos. Caso uma dessas situações desencadeie a execução do escalonador, este recalcula o TTV de todas as instâncias e ordena a fila de pendentes por ordem crescente dos TTVs. Em seguida, o escalonador processa a fila completa, uma requisição por vez.

O escalonador escolhe o servidor onde alocar uma instância em duas etapas: *verificação de viabilidade* e *classificação*. Na primeira, o escalonador filtra os servidores que estão habilitados para alocar a instância associada a uma requisição j, levando em consideração a quantidade de recursos solicitada e as possíveis restrições de alocação especificadas em j. Alguns servidores podem se tornar habilitados apenas se uma ou mais das instâncias já alocadas forem preemptadas. No tempo t quando a viabilidade de um servidor é verificada, uma instância associada a uma requisição k só pode ser preemptada em benefício de j se $\tau_j^t < \tau_k^t$, com isso, a instância mais próxima de violar seu SLO será a que tem prioridade para executar; além disso, o escalonador considera uma margem de segurança para o TTV (σ_i), específica para cada classe de serviço i. Essa margem faz com que uma instância seja preemptada quando tenha disponibilidade maior do que sua meta. Assim sendo, instâncias com o TTV muito pequeno — isto é, com $\tau_k^t < \sigma_i$, onde i é a classe requisitada por k — não são consideradas como passíveis de preempção. Instâncias passíveis de preempção são consideradas seguindo a ordem do maior para o menor TTV. Se há mais de uma instância com o maior TTV, então uma delas é escolhida aleatoriamente. No cenário onde nenhum dos servidores esteja habilitado para alocar a instância,

sua requisição permanece na fila. Caso contrário, os servidores elegíveis são analisados na etapa seguinte de classificação.

Na etapa de classificação, o escalonador usa uma função de pontuação para escolher o servidor mais adequado dentre os filtrados na etapa anterior. O servidor com maior pontuação é selecionado para alocar a instância. Inicialmente os servidores elegíveis são classificados em dois conjuntos: servidores que possuem recursos para alocar sem a necessidade de preempção e servidores que necessitam de pelo menos uma preempção para alocar a requisição pendente. Se o primeiro conjunto não for vazio, o servidor selecionado será deste conjunto. Diferentes funções de pontuação podem ser usadas nessa seleção.

Por outro lado, se todos os servidores habilitados requerem preempção, uma pontuação para cada servidor é computada. No tempo t, a pontuação do servidor s para alocar a instância da requisição j da classe i, $\zeta^t_{s,j}$, é computada levando em conta o TTV de todas as instâncias do conjunto P_s (composto pelas instâncias a serem preemptadas em s para que a instância de j possa ser alocada neste servidor). A pontuação $\zeta^t_{s,j}$ é dada por:

$$\zeta^t_{s,j} = \sum_{k \in P_s} (\tau^t_k - \sigma_i). \tag{2}$$

Visto que o servidor com maior pontuação é o selecionado, a Equação 2 favorece a preempção de instâncias com maiores TTVs. Quanto maiores os TTVs das instâncias a serem preemptadas, maior será a pontuação do servidor. Se houver empate para a maior pontuação, então a mesma função de classificação utilizada quando não há necessidade de preempções é usada como forma de desempate.

É importante destacar que o EDSLO, como foi descrito até então, estabelece que todas as classes de serviço são igualmente importantes. Dessa forma, durante momentos de alta contenção de recursos — devido a erros cometidos no planejamento de capacidade e/ou controle de admissão —, a QoS entregue para requisições de todas as classes tendem a ser impactadas igualmente. Porém, este comportamento pode ser indesejado visto que instâncias de classes com metas de disponibilidade mais altas precisarão de mais tempo para se recuperar após o período de contenção. Por essa razão, o conceito de importância entre classes foi incluído no EDSLO. Dessa forma, torna-se possível prover um serviço melhor para as requisições mais importantes e ainda preservar o tratamento justo entre requisições de mesma classe. Para isso, atribui-se uma prioridade π_i para cada classe de serviço i, e, no tempo t, permite-se a preempção de uma instância k da classe i em benefício da instância j da classe i' se: *i)* $\tau^t_k < \sigma_i$, $\tau^t_j < \sigma_{i'}$ e $\pi_{i'} > \pi_i$; ou *ii)* $\tau^t_k < \sigma_i$, $\tau^t_j < \sigma_{i'}$, $\pi_{i'} = \pi_i$ e $\tau^t_j < \tau^t_k$. Isso permite que, as instâncias mais importantes sejam priorizadas em momentos de alta contenção de recursos, impossibilitando que uma instância "menos importante" prejudique outras "mais importantes". Além disso, o uso de prioridades no EDSLO não requer que as classes mais importantes sejam necessariamente aquelas que buscam atingir os maiores SLOs, como é o caso no EBP. Isso torna a ordenação de importância entre as classes completamente flexível e é mais um diferencial do EDSLO.

4. Materiais e Método

A avaliação da política de escalonamento proposta se deu através de experimentos de simulação e de medição. O modelo de simulação, as cargas de trabalhos utilizadas, o processo experimental e as métricas de avaliação são apresentados a seguir

4.1. Modelo de Simulação

O modelo de simulação desenvolvido para a avaliação do EDSLO foi implementado em Erlang e executa sob a plataforma Sim-Diasca[1]. O simulador realiza o escalonamento como descrito na Seção 3, com base na carga a ser escalonada e na infraestrutura do provedor. A função de pontuação implementada no modelo de simulação, usada na etapa de classificação, baseia-se no Kubernetes (Burns et al. 2016). Este sistema calcula a pontuação de um servidor como uma combinação de funções de prioridade. Duas funções padrões disponíveis no kubernetes foram implementadas no simulador (*Least Requested Priority* e *Balanced Resource Allocation*) e combinadas com o mesmo peso. Além disso, neste estudo, cada classe de serviço tem como prioridade associada o valor de seu SLO de disponibilidade. Portanto, as classes "mais importantes" são aquelas com maiores SLOs.

Para fins de comparação, um EBP também foi modelado no simulador. Neste escalonador, uma prioridade é atribuída para cada classe, com prioridades mais altas sendo atribuídas as classes cuja QoS prometida é maior. A estrutura dos modelos é essencialmente a mesma, com diferenças apenas na: (i) ordenação das requisições da fila de pendentes (em ordem decrescente de prioridades); (ii) execução de preempções (sempre preempta apenas instâncias de prioridades mais baixas); e, (iii) classificação dos servidores habilitados, com a necessidade de preempção(ões) para alocar uma instância (favorece aqueles com menor número de preempções de instâncias das classes mais importantes).

4.2. Carga de trabalho

A carga de trabalho recebida pelo simulador indica o tempo de admissão de cada uma das requisições, as quantidades de CPU e RAM solicitadas e quanto tempo leva para executar a requisição. Os escalonadores simulados garantem que as instâncias irão ter recursos alocados a si durante todo esse tempo, mas não necessariamente de forma ininterrupta. Após a conclusão de uma requisição, é possível calcular a disponibilidade entregue para a mesma e avaliar se ela teve seu SLO atendido.

As cargas usadas nos experimentos de simulação foram obtidas de rastros de execução de um *cluster* em produção do Google (Wilkes 2011). Esses rastros abrangem um período de 29 dias e seus dados estão ofuscados (e.g a capacidade dos servidores é dada como um percentual do maior servidor do *cluster*, portanto, a capacidade 1 para um dado recurso representa a capacidade do maior servidor para o mesmo recurso). Esses rastros são compostos por mais de 25 milhões de requisições submetidas por mais de 900 usuários do *cluster*. Essas requisições podem ser classificadas em 12 diferentes prioridades (0 a 11). Baseando-se na descrição dos rastros, é possível destacar duas classes: (i) *production* (*prod*) com prioridade > 8, são as mais importantes visto que supõe-se que nunca são preemptadas; e, (ii) *free* com prioridade < 2, são aquelas que são frequentemente preemptadas para liberar recursos para instâncias com prioridades mais altas. As requisições com prioridades intermediárias foram classificadas como *batch*. Portanto, *prod*, *batch* e *free* são as classes de serviço consideradas e suas metas de disponibilidade são arbritadas, nesse estudo, em 100%, 90% e 50%, respectivamente.

As cargas de trabalho utilizadas neste trabalho são subconjuntos dos rastros do Google gerados como segue. Inicialmente, uma análise de agrupamento dos usuários

[1]Informações sobre o Sim-diasca estão disponíveis em `http://sim-diasca.com`.

do *cluster* foi realizada, aplicando o modelo k-means levando em conta o número e as propriedades das requisições. Esse agrupamento resultou em seis grupos de usuários, de onde foram selecionados aleatoriamente 10% dos usuários de cada um dos grupos. As requisições submetidas pelos usuários selecionados formam a carga de trabalho para ser usada no experimento; 12 cargas de trabalho distintas foram geradas usando esse método.

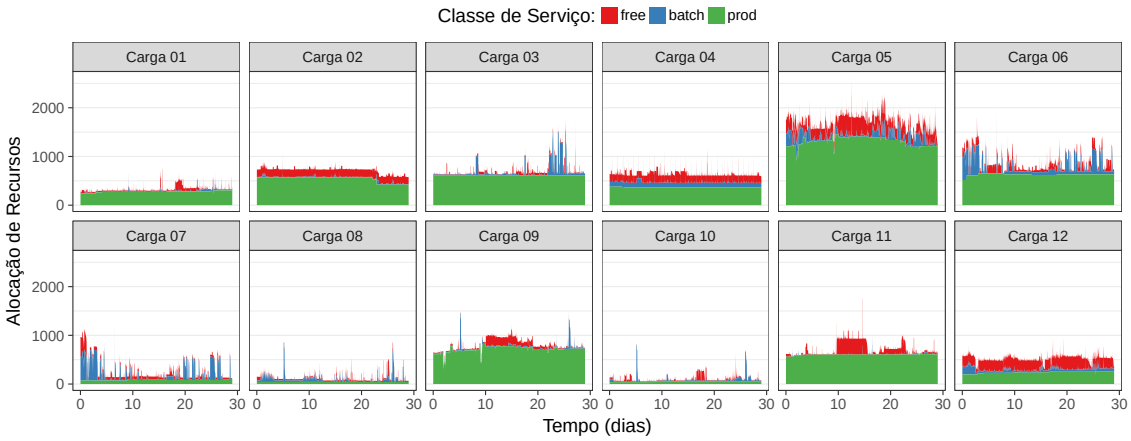

Figura 1. Alocação de recursos ao longo do tempo por carga de trabalho

A Figura 1 apresenta a quantidade de recursos alocados[2] ao longo do tempo para cada uma das cargas quando submetidas para uma infraestrutura composta por um único servidor de capacidade infinita para CPU e RAM. Mesmo visualmente, é possível atestar que as cargas geradas diferem substancialmente entre si; suas formas, a combinação de requisições por classe de serviço, as demandas dos picos de requisições e suas intensidades são diferentes. Essa heterogeneidade das cargas se deve ao fato dos diferentes subconjuntos de usuários reais levarem a diferentes agrupamentos de requisições. Considerando que apenas uma carga real é utilizada para avaliação, esta diversidade é importante para analisar o escalonador sob diferentes (e ainda verossímeis) cargas de trabalho.

4.3. Infraestrutura

Os experimentos de simulação para avaliação da política proposta seguem um projeto aninhado de um fator, que é o tamanho da infraestrutura do provedor. Este fator varia em 4 níveis diferentes. Segue o processo de definição desses níveis para cada carga:

1. Geração da carga a partir dos rastros do Google (descrito na Seção 4.2);
2. Simulação da alocação da carga gerada considerando que o provedor é composto por um único servidor com capacidade infinita para CPU e RAM;
3. Análise dos resultados do passo anterior para identificar a quantidade máxima de CPU e RAM utilizadas ao longo do tempo. Uma vez que todas as capacidades dos recursos são normalizadas nos rastros do Google, e, considerando que todos os servidores na infraestrutura terão a mesma capacidade de CPU (resp. RAM) que o maior servidor da Google considerando CPU (resp. RAM) — i.e. 1 para a capacidade de ambos os recursos —, o número de servidores (N) é definido como sendo o teto entre o maior valor dos valores máximos de CPU e RAM usados;

[2]Os valores são medidos em intervalos de 10 minutos e correspondem ao recurso mais usado na carga.

4. Definição dos 4 cenários para a capacidade da infraestrutura do provedor. A capacidade da infraestrutura define o nível de contenção de recursos do provedor. Os cenários analisados são: (a) baixa contenção — N servidores; (b) sem contenção — $N + 0.1N$ servidores; (c) média contenção — $N - 0.1N$ servidores; e (d) alta contenção — $N - 0.2N$ servidores.

Note que o cenário com baixa contenção é provisionado com base na máxima utilização, mas não necessariamente garante que todas as requisições poderão ser alocadas pelo tempo necessário para satisfazê-las. Isso ocorre porque, na prática, existe um *cluster* (ao invés de apenas um servidor) e o escalonamento naturalmente causa fragmentação e encalhamento de recursos (Verma et al. 2015). O cenário sem contenção, por sua vez, visa mitigar esses efeitos, porém, o custo de operar uma infraestrutura super provisionada com milhares de servidores pode ser proibitivo, dependendo da taxa de utilização dos recursos. O cenário com alta contenção representa um erro no planejamento de capacidade, enquanto que o cenário com média contenção reflete um balanceamento entre a busca por uma redução do custo de provisionamento e a garantia da QoS prometida.

Nesse estudo, o EDSLO está configurado para processar a fila de espera sempre que acontece um intervalo de tempo de 10 segundos em que nenhuma requisição é recebida ou concluída. Este intervalo foi definido empiricamente através da observação do Kubernetes em cenários onde há requisições na fila de pendentes e o escalonador tenta alocá-las periodicamente. Para efeito de simplicidade, a margem de segurança para o TTV também é definida como 10 segundos em todos os cenários experimentados.

4.4. Métricas de Avaliação

Cada uma das 12 cargas geradas (ver Seção 4.2) foi submetida para 4 infraestruturas com capacidades diferentes, definidas como detalhado na Seção 4.3. Essas cargas de trabalho foram submetidas para escalonamento duas vezes para cada infraestrutura, uma vez para cada tipo de escalonador: EBP e EDSLO. Replicações não são necessárias visto que os resultados são determinísticos quando a mesma carga e infraestrutura são consideradas.

Ao longo dos experimentos de simulação, métricas de desempenho são coletadas com o objetivo de comparar as abordagens de EDSLO e EBP. A principal métrica de comparação usada é o *Deficit de QoS* das requisições concluídas. Essa métrica é computada como a diferença entre a meta de QoS e a QoS realmente entregue para as requisições e é medida apenas para requisições que tiveram seus SLOs violados. Uma outra métrica avaliada neste estudo é a *Satisfação do SLO* da carga de trabalho. Essa métrica indica o percentual de requisições da carga que tiveram seus SLOs satisfeitos.

5. Resultados e Discussões

5.1. Análise de cenários com baixa contenção

Analisando os cenários com baixa contenção de recursos, o EDSLO foi capaz de satisfazer os SLOs para quase todas as cargas, com exceção para 6 e 7. Por outro lado, o EBP não satisfaz os SLOs em quase todos os casos, com exceção para 5 e 9. Com isso, verifica-se que o EDSLO é capaz de usar os recursos de forma mais eficiente do que o EBP. Como discutido anteriormente, a fragmentação de recursos é suficiente para causar violações de SLO. Uma vez que as cargas 5 e 9 estão entre as cargas com picos mais intensos (veja

Figura 1) e a capacidade é definida de acordo com a utilização máxima, elas estão entre as cargas com infraestrutura com mais recursos extras. Isso explica o fato do EBP conseguir atender todos os SLOs para as mesmas. Todavia, analisando detalhadamente as cargas 6 e 7, observa-se um pico de requisições *batch* no 3º dia em ambas, fazendo com que os dois escalonadores não consigam satisfazer os SLOs de todas as requisições desse pico.

Nos cenários sem contenção de recursos, todos os SLOs de todas as cargas foram satisfeitos com ambos os escalonadores. Portanto, um incremento de 10% no tamanho N do *cluster* foi suficiente para mitigar os efeitos da fragmentação e encalhamento de recursos e permitir que todos os SLOs das classes requisitadas fossem atendidos.

5.2. Análise de cenários média contenção

Através da análise desses cenários é possível investigar a eficiência dos escalonadores quando a etapa de planejamento de capacidade decide por uma infraestrutura 10% menor que aquela baseada na carga máxima. A Figura 2 apresenta os *deficit* de QoS obtidos pelas requisições que tiveram seu SLO violado. Esses *deficit* são apresentados em box-plots para dar uma ideia da dispersão dos valores. É importante destacar que não houve violações em alguns cenários, dessa forma, nenhum box-plot foi plotado para estes casos.

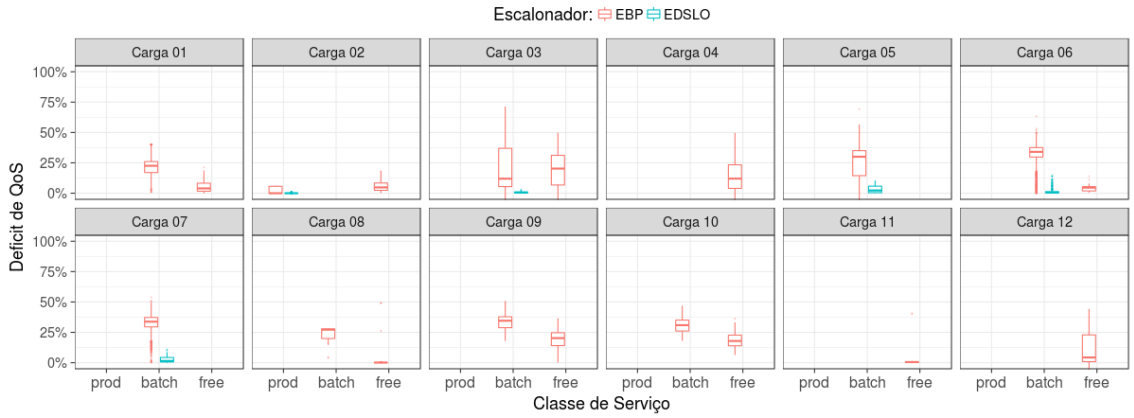

Figura 2. *Deficit* de QoS para infraestruturas no cenário com média contenção

Mesmo reduzindo 10% da capacidade N definida com base na utilização máxima, o EDSLO conseguiu satisfazer todos os SLOs em 7 das 12 cargas avaliadas (cargas 1, 4, e 8 a 12). Essas cargas estão entre aquelas com picos mais altos de requisições que não são da classe *prod* ou possuem uma quantidade significativa de requisições da classe *free*. Por utilizar o TTV para tomar decisões, o EDSLO foi capaz de selecionar de forma mais eficiente quais as instâncias devem executar, alcançando, geralmente, QoS que são mais próximas de suas respectivas metas. Este escalonador pode alternar quais das requisições devem ter suas instâncias executando ao longo do tempo. Esta habilidade não existe quando o EBP executa. Este considera todas as instâncias da mesma classe como iguais, não permitindo que requisições de mesma classe preemptem recursos uma da outra. Portanto, durante períodos de contenção de recursos, o EBP mantém algumas instâncias sempre executando e outras sempre pendentes. Esse comportamento não garante tratamento justo entre requisições de mesma classe, podendo levar a uma QoS bem ruim para algumas instâncias (as pendentes) enquanto outras têm QoS acima do esperado. Isso explica a razão pela qual a dispersão dos *deficit* de QoS é maior quando o EBP é executado.

No entanto, violações de SLO ocorrem para os outros casos (cargas 2, 3, 5, 6 e 7), independente do escalonador usado. Para as cargas 3, 5, 6 e 7, as violações ocorrem devido a picos de requisições *batch* (no dia 27 para a carga 3 e no dia 3 para as outras). Nesses casos, o EDSLO violou os SLOs em menos requisições que o EBP. Para a carga 2, ambos os escalonadores causaram *deficit* de QoS por causa do pico de requisições *prod* no dia 5 da simulação. Nesse momento a infraestrutura não teve capacidade para acomodar todas elas. A partir da Figura 2 está claro que os *deficit* para as requisições *prod* foram menores quando o EDSLO foi executado. Neste caso, o intervalo dos *deficit* de QoS é [0,00004%;1,01%] contra [0,003%;6,24%] quando o EBP foi usado.

No geral, quando violações dos SLOs ocorrem, os *deficit* de QoS medidos com o EDSLO são menores e variam menos que os obtidos com o EBP. O novo escalonador opta por decrementar vagarosamente a QoS de tantas instâncias de uma classe quanto possível, ao invés de degradar substancialmente a QoS de um número reduzido de instâncias. Como consequência, quando a infraestrutura não é suficiente para conseguir 100% de satisfação de SLOs da carga, o escalonador proposto prejudica as instâncias de cada classe igualmente, podendo levar a uma redução da taxa de satisfação de SLO. Nos cenários com média contenção, apenas as requisições *prod* da carga 2 obtiveram uma taxa de satisfação menor quando comparados os EBP e EDSLO (96,24% x 71,48%).

Ainda analisando a Figura 2, nota-se que há *deficit* de QoS para requisições *prod* (carga 2) enquanto não há para outras classes de serviço. Da mesma forma, existem *deficit* de QoS para requisições *batch* e não há para classe *free* (carga 3, 5, 6 e 7). Essas situações podem, erroneamente, dar a impressão de que o EDSLO favoreceu instâncias das classes *batch* e/ou *free* em detrimento de instâncias da classe *prod*. No entanto, isso ocorreu devido a alguns picos de requisições de mesma classe nessas cargas (*prod* em 2, e *batch* em 3, 5 6, e 7) onde a infraestrutura não foi capaz de alocar as instâncias desses picos de forma a satisfazer seus SLOs. Durante esses períodos, todos os recursos foram utilizados para alocar instâncias de mesma classe ou classes mais importantes. Após esses períodos, o EDSLO foi mais eficiente na escolha da instância certa a ser alocada. Assim, o EDSLO consegue oferecer QoS de acordo com a esperada para instâncias "menos importantes" não porque ele as prioriza, mas porque ele é mais eficiente em determinar quais delas devem estar executando e esperando ao longo do tempo.

5.3. Análise de cenários com alta contenção

Nesses cenários a capacidade é ainda menor. A Figura 3 apresenta os box-plots dos *deficit* de QoS medidos. Mesmo nesses cenários, o EDSLO conseguiu satisfazer os SLOs de todas as requisições para as cargas 4, 11 e 12. Essas cargas têm em comum o fato de nenhuma delas ter picos intensos de requisições *prod* (nem mesmo *batch*). Além disso, essas cargas possuem uma quantidade significativa de recursos solicitados para classes *batch* e *free* (principalmente para a última) que podem ser preemptados pelo escalonador, que faz uso disso para acomodar toda a carga. Considerando as cargas com violações de SLOs, a partir da Figura 3 fica claro que os *deficit* de QoS obtidos quando o escalonador proposto executa são menores e variam bem menos que os medidos quando o EBP é usado. No entanto, há duas exceções: cargas 2 e 8. Na carga 2, por volta de 1000 requisições *batch* tiveram SLO violado para beneficiar requisições *prod* que estavam executando no mesmo período. Essas instâncias *batch* (que são tipicamente menores que as *prod*) podem ser alocadas pelo EBP usando os recursos que não estão sendo usados pelas *prod* que, neste

caso, ficam pendentes. Na carga 8, apenas 4 requisições *free* tiveram seus SLOs violados com o EDSLO. Elas ficam pendentes para beneficiar instâncias *batch*. Essas mesmas requisições também têm seus SLOs violados usando o EBP, porém, neste caso, o número total de violações para a classe *free* foi aproximadamente 3500.

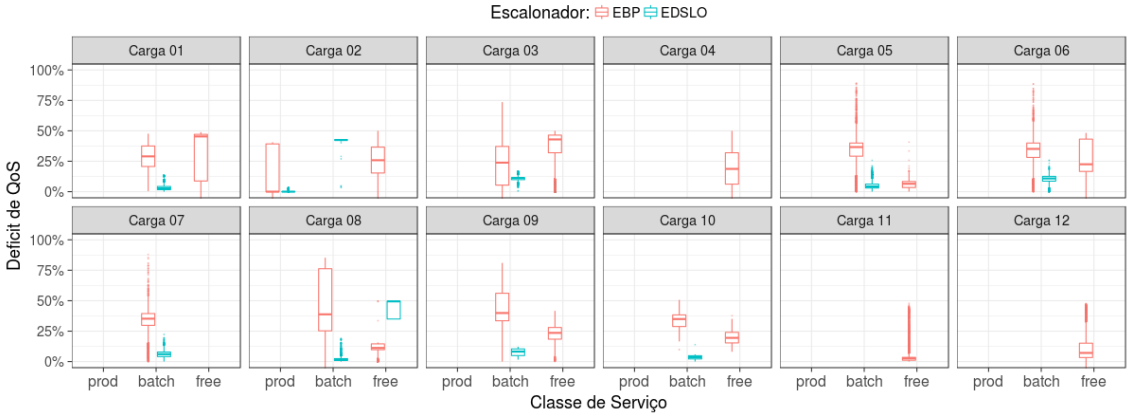

Figura 3. *Deficit* de QoS para infraestruturas no cenário com alta contenção

O uso do EDSLO tem como desvantagem a possibilidade do provedor reduzir as taxas de satisfação dos SLOs em comparação com o uso do EBP. Os intervalos de confiança (IC), com 95% de confiança, das taxas de satisfação dos SLOs observadas para as cargas no cenário com alta contenção são apresentados na Tabela 1. Novamente, a possibilidade de taxas menores é justificada pelos *deficit* de QoS mais baixos e pelo tratamento mais justo entre requisições da mesma classe alcançados pelo EDSLO.

Tabela 1. IC para a satisfação dos SLOs para os cenários com alta contenção

Escalonamento	prod	batch	free
EBP	[97.98%,100.0%]	[98.84%,99.75%]	[96.12%, 100%]
EDSLO	[79.00%,100%]	[98.33%,99.5%]	[98.33%,99.5%]

5.4. Validação dos modelos de simulação

Os modelos de simulação do EDSLO e do EBP foram validados através de experimentos de medições executados no kubernetes. O escalonador padrão do kubernetes foi usado para validar o EBP e um novo foi desenvolvido como prova de conceito do EDSLO.

Uma carga sintética foi gerada e submetida para: o simulador e o sistema real. Em ambos os cenários, a infraestrutura foi composta por 20 servidores, cada um deles com capacidade para alocar 10 requisições simultaneamente. A carga possui 256 requisições e foi projetada para ter o número exato de requisições para preencher toda a infraestrutura e ainda satisfazer todos os SLOs. O intervalo entre duas requisições sequenciais é de 1 segundo. A Figura 4 apresenta as disponibilidades para as instâncias associadas a cada uma das requisições, usando os dois escalonadores no kubernetes e nos modelos de simulação.

Como esperado, quando o EBP é usado, todas as instâncias *prod* e *batch* e algumas *free* têm disponibilidade de 100%. O restante das instâncias *free* possuem disponibilidade abaixo da meta. Essas foram submetidas quando a infraestrutura já estava completamente

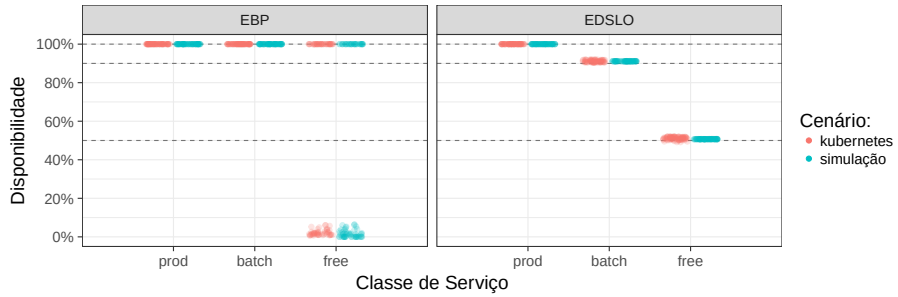

Figura 4. Disponibilidades usando os escalonadores em diferentes cenários

ocupada ou foram preemptadas quando outras requisições com prioridades mais altas foram submetidas. Por sua vez, o EDSLO entrega para cada instância uma disponibilidade que é bem próxima da que foi prometida. Comparando-se as execuções dos experimentos de medição e simulação, observa-se que não há diferença significativa entre seus resultados. Esse mesmo procedimento foi realizado para outras 10 cargas de trabalho geradas com características similares e os resultados foram todos semelhantes.

6. Conclusões

Este estudo apresenta uma nova política de EDSLO. Esta toma decisões com base nos SLOs e na QoS entregue para cada requisição no momento da decisão. O objetivo desta política é reduzir a variância de QoS entregue para requisições de mesma classe, promovendo um justo provisionamento. A política proposta foi avaliada comparando seu desempenho com o de uma política de EBP, ambas alocando as mesmas cargas de trabalho em infraestruturas com as mesmas capacidades. Os resultados mostram que o EDSLO consegue gerenciar as instâncias a serem alocadas de forma mais eficiente, obtendo a satisfação de todos os SLOs em mais cenários que o EBP. Além disso, quando não é possível garantir que todas as requisições recebam a QoS prometida, o escalonador proposto produz *deficit* de QoS que são menos variáveis, e, em média, menores que os produzidos pelo EBP para os mesmos cenários. Análises sobre outras métricas de desempenho (e.g. sobrecarga de preempções realizadas) e outros cenários (e.g. heterogeneidade das infraestruturas) serão deixadas como trabalhos futuros.

Agradecimentos

Os autores agradecem o apoio financeiro recebido pelo Centro de Inovações da Ericsson Telecomunicações S.A./Brasil e EMBRAPII-CEEI.

Referências

[Boutin et al. 2014] Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M., and Zhou, L. (2014). Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *OSDI*.

[Burns et al. 2016] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Commun. ACM*.

[Carvalho et al. 2015] Carvalho, M., Menascé, D., and Brasileiro, F. (2015). Prediction-based admission control for iaas clouds with multiple service classes. In *IEEE 7th Int. Conf. on Cloud Computing*. IEEE.

[Delgado et al. 2015] Delgado, P., Dinu, F., Kermarrec, A.-M., and Zwaenepoel, W. (2015). Hawk: Hybrid datacenter scheduling. In *Proc. of the 2015 USENIX Annual Technical Conf.*, number EPFL-CONF-208856. USENIX Association.

[Delimitrou and Kozyrakis 2013] Delimitrou, C. and Kozyrakis, C. (2013). Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*. ACM.

[Delimitrou and Kozyrakis 2014] Delimitrou, C. and Kozyrakis, C. (2014). Quasar: resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*.

[Delimitrou et al. 2015] Delimitrou, C., Sanchez, D., and Kozyrakis, C. (2015). Tarcil: reconciling scheduling speed and quality in large shared clusters. In *Proc. of the 6th Symp. on Cloud Computing*. ACM.

[Gog et al. 2016] Gog, I., Schwarzkopf, M., Gleave, A., Watson, R. N., and Hand, S. (2016). Firmament: Fast, centralized cluster scheduling at scale. Usenix.

[Goiri et al. 2010] Goiri, I., Julia, F., Nou, R., Berral, J. L., Guitart, J., and Torres, J. (2010). Energy-aware scheduling in virtualized datacenters. In *Cluster Computing (CLUSTER), 2010 IEEE Int. Conf. on*.

[Hindman et al. 2011] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S., and Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*.

[Isard et al. 2009] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. (2009). Quincy: fair scheduling for distributed computing clusters. In *Proc. of the ACM SIGOPS 22nd Symp. on Operating systems principles*.

[Karanasos et al. 2015] Karanasos, K., Rao, S., Curino, C., Douglas, C., Chaliparambil, K., Fumarola, G. M., Heddaya, S., Ramakrishnan, R., and Sakalanaga, S. (2015). Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *USENIX ATC*.

[Kong et al. 2011] Kong, X., Lin, C., Jiang, Y., Yan, W., and Chu, X. (2011). Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction. *Journal of network and Computer Applications*.

[Ousterhout et al. 2013] Ousterhout, K., Wendell, P., Zaharia, M., and Stoica, I. (2013). Sparrow: Distributed, low latency scheduling. In *Proc. of the 24th ACM SOSP*.

[Pan et al. 2013] Pan, W., Rowe, J., and Barlaoura, G. (2013). Records in the cloud (ric) user survey report. Technical report.

[Reiss et al. 2012] Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., and Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Symp. on Cloud Computing*.

[Schwarzkopf et al. 2013] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. (2013). Omega: Flexible, scalable schedulers for large compute clusters. In *Proc. of the 8th ACM European Conf. on Computer Systems*.

[Shahrad and Wentzlaff 2016] Shahrad, M. and Wentzlaff, D. (2016). Availability knob: Flexible user-defined availability in the cloud. In *Proc. of the 7th ACM Symp. on Cloud Computing*.

[Vavilapalli et al. 2013] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. (2013). Apache hadoop yarn: Yet another resource negotiator. In *Proc. of the 4th Symp. on Cloud Computing*. ACM.

[Verma et al. 2014] Verma, A., Korupolu, M., and Wilkes, J. (2014). Evaluating job packing in warehouse-scale computing. In *2014 IEEE Int'l Conf. on Cluster Computing, CLUSTER*.

[Verma et al. 2015] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at google with borg. In *Proc. of the 10th European Conf. on Computer Systems*.

[Wilkes 2011] Wilkes, J. (2011). More Google cluster data. Google research blog.