Reduzindo o Custo de Explorar Configurações para Execução de Aplicações na Nuvem

Maria Casimiro, Diego Didona, Paolo Romano,
Luís Rodrigues, Willy Zwanepoel
{maria.casimiro,paolo.romano,ler}@tecnico.ulisboa.pt,
{diego.didona,willy.zwanepoel}@epfl.ch

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa e EPFL

Resumo A tarefa de descobrir qual a configuração ideal para executar uma aplicação distribuída na nuvem é extremamente complexa devido ao vasto número de configurações alternativas que tipicamente estão disponíveis: os fornecedores de serviços na nuvem oferecem uma vasta gama de máquinas, com características e dimensões diversas e, para além disso, a maioria das aplicações possui inúmeros parâmetros de configuração. Estamos interessados em construir modelos que permitam automatizar esta escolha, garantindo níveis pré-determinados de Qualidade de Serviço, com custo operacional mínimo. Não é possível, no entanto, construir estes modelos sem experimentar efetivamente várias configurações, um processo não só moroso mas também dispendioso. Neste artigo propomos técnicas que permitem reduzir o custo da exploração através de uma criteriosa escolha das configurações a experimentar, que tem em conta o custo de cada experiência e a contribuição esperada da mesma para a exatidão do modelo. Os resultados da nossa avaliação mostram que o nosso algoritmo é capaz de encontrar configurações ótimas.

1 Introdução

O paradigma de computação na nuvem afirmou-se como uma realidade incontornável e são cada vez mais frequentes os casos de empresas que recorrem a servidores na nuvem para executar os mais diversos tipos de trabalhos. Apesar da utilização de serviços do tipo "plataforma como serviço" poder representar poupanças significativas quando comparada com os custos de manter uma infraestrutura própria, o seu uso ainda acarreta custos significativos.

A tarefa de descobrir qual a configuração ideal para executar uma aplicação distribuída na nuvem é extremamente complexa devido ao vasto número de configurações alternativas que tipicamente estão disponíveis: os fornecedores de serviços na nuvem oferecem uma vasta gama de máquinas, com características e custos diversos, e, para além disso, a maioria das aplicações possui inúmeros parâmetros de configuração. Ademais, uma grande parte das aplicações é demasiado complexa para que seja possível construir manualmente um modelo analítico que permita prever com precisão o seu desempenho para todos os perfis de carga e configurações. Desta forma, têm vindo a ganhar popularidade abordagens baseadas em técnicas de aprendizagem automática, que constroem modelos com base

em evidências empíricas que resultam da exploração de diferentes configurações. Por conseguinte, é bastante relevante desenvolver técnicas que permitam identificar não só as configurações ideais para executar uma certa aplicação na nuvem mas também, em particular, qual a configuração menos dispendiosa que satisfaz os requisitos temporais definidos para essa aplicação.

Neste trabalho, estamos interessados em estudar metodologias que possibilitem a criação, de forma automática, de modelos que permitam escolher a melhor configuração para executar uma certa aplicação com um dado perfil de carga. Uma vez que, para construir estes modelos, é necessário experimentar várias configurações, a própria construção do modelo acarreta também custos. Desta forma, no processo de construção do modelo, é preciso ponderar o custo em que se incorre na sua construção com as poupanças que se esperam obter ao descobrir configurações mais adequadas. Assim, neste artigo propomos um algoritmo, Otimização Bayesiana com Orçamento Limitado (OBOL), que permite reduzir o custo da exploração, através de uma criteriosa escolha das configurações a experimentar, e que tem em conta não só o custo de cada experiência mas também a contribuição esperada da mesma para a exatidão do modelo.

2 Trabalho Relacionado

Vários trabalhos têm abordado o problema de automatizar a procura da melhor configuração para executar aplicações na nuvem. Alguns destes sistemas focam-se na escolha dos parâmetros de configuração da aplicação, outros na escolha dos recursos fornecidos pelos operadores de infraestrutura na nuvem. Além desses, podemos ainda distinguir os trabalhos que tentam otimizar as configurações do ponto de vista dos utilizadores e os que tentam maximizar a utilização dos recursos, beneficiando assim os fornecedores de serviços na nuvem.

Um exemplo de um sistema que permite automatizar a escolha dos parâmetros de uma aplicação é o sistema iTuned [9], que aborda este problema no contexto de sistemas de bases de dados. O iTuned representa o desempenho estimado do sistema como um processo gaussiano que é usado em conjunto com um processo de inferência Bayesiana para estimar qual o benefício de explorar o espaço dos parâmetros de forma a maximizar a exatidão do modelo. Um outro sistema, cujo objetivo é afinar implementações de Memória Transacional para conjuntos de trabalho específicos, é o ProteusTM [8]. Este sistema também recorre a Optimização Bayesiana para guiar a exploração do espaço. Mais recentemente, o sistema CherryPick [3] propôs a utilização de uma abordagem semelhante para descobrir qual o conjunto e tipo de máquinas, numa plataforma de computação na nuvem, que permite atingir níveis pré-determinados de desempenho (ex., tempo de resposta ou débito) com um custo operacional mínimo.

Por sua vez, o Quasar [6] e o HCloud [7] são exemplos de sistemas que pretendem otimizar a operação de quem fornece máquinas virtuais de forma a rentabilizar o equipamento. Para guiar a escolha das configurações, o Quasar utiliza técnicas de recomendação como *filtragem colaborativa* [10, 18] de forma a estimar o desempenho de uma aplicação para diferentes combinações de recursos. O HCloud, por sua vez, recorre ao Quasar para realizar as previsões necessárias.

Algorithm 1 Seleção da melhor configuração

```
1: function ESCOLHER_CONFIG
2:     S ← initS()                          ▷ Amostrar configs iniciais e atualizar variáveis de estado
3:     while O ≥ 0 ∧ I ≠ ∅ do
4:         (c, U_c) ← PROXIMA_CONFIG(S, MaxProfundidade)
5:         if (c == null ∨ U_c ≤ ε) then                                  ▷ Exploração termina
6:             return argmin_{custo(c)}{S}
7:         else                                                           ▷ Atualizar modelo e estado
8:             custo(c) ← sample(c)
9:             A ← A ∪ {(c, custo(c)}; O ← O − custo(c); I ← I \{c}
10:        end if
11:    end while
12: end function
```

Neste artigo exploramos ferramentas que ajudem quem pretende executar aplicações na nuvem e que, na linha dos trabalhos anteriores, utilizem processos de inferência Bayesiana [5, 17] para guiar a exploração das várias configurações que são testadas na construção do modelo. No entanto, ao contrário de trabalhos anteriores para a otimização da execução de aplicações na nuvem, pretendemos tomar em consideração o custo da fase de exploração, necessária para a construção do modelo de previsão. De facto, sistemas como o iTuned, o ProteusTM ou o CherryPick, acima referidos, ao longo do processo de refinamento do modelo, experimentam configurações tendo apenas em conta, a cada iteração, a contribuição esperada dessa experiência para melhorar a precisão do modelo, independentemente do custo da mesma. Isto leva a que o processo de construção do modelo possa ser muito dispendioso e, inclusive, anular os potenciais benefícios económicos resultantes da construção de um modelo muito preciso.

Neste contexto, são também relevantes para o nosso objetivo trabalhos recentes que estendem os algoritmos de otimização Bayesiana para ter em consideração um limite do número de explorações que são realizadas [15, 16]. Estes trabalhos propõem a utilização de mecanismos de procura em profundidade para estimar o custo de diversos percursos de exploração. Um dos nossos objetivos é adaptar estas abordagens de forma a reduzir o custo da procura de configurações.

3 Otimização Bayesiana com Orçamento Limitado

Nesta secção é proposta uma nova metodologia para encontrar uma configuração adequada para executar uma aplicação na nuvem respeitando um orçamento máximo pré-definido que limita o processo de procura.

O objetivo é encontrar a configuração que respeita um conjunto de restrições de qualidade de serviço conhecidas à partida (p. ex., assegurando que o trabalho termina a sua execução num intervalo de tempo pré-definido) e que permite executar o trabalho com o menor custo, sem exceder o orçamento definido.

Em seguida, fornece-se uma perspetiva geral do processo de procura (Algoritmo 1) e, posteriormente, detalham-se alguns dos seus mecanismos.

3.1 Panorâmica

O objetivo do processo de exploração é construir um modelo do custo de uma aplicação quando executada na nuvem em diferentes configurações. Assume-se

que a aplicação, por exemplo um processo de análise de um grafo ou de treino de modelos baseados em técnicas de aprendizagem automática, executa durante um tempo finito mas desconhecido e que depende da configuração escolhida (em termos de recursos computacionais e de parâmetros internos da aplicação).

Neste trabalho assumimos que o custo é diretamente proporcional à duração de utilização dos recursos (máquinas virtuais) numa dada configuração, e que os custos por unidade temporal (normalmente segundos) de utilização dos recursos são conhecidos à partida. Portanto, ao prever, através do modelo, o custo de execução de uma aplicação numa dada configuração, obtém-se também uma previsão do tempo de execução da mesma.

O modelo, numa dada iteração i, é construído a partir do conjunto de amostras $\mathcal{A}^i$ disponíveis nessa iteração, obtido executando a aplicação num dado conjunto de configurações. Isto é, o conjunto das amostras corresponde ao conjunto das configurações que já foram exploradas. Cada amostra consiste num tuplo $\langle c_x, \mathit{custo}(c_x)\rangle$, em que c_x é uma dada configuração e $\mathit{custo}(c_x)$ é o custo incorrido ao executar a aplicação nessa configuração.

A cada iteração, o modelo é refinado, acrescentando-se um novo tuplo ao conjunto de amostras. Seja $\mathcal{I}^{i-1}$ o conjunto das configurações que ainda não foram exploradas no final da iteração $i-1$. O algoritmo deve escolher, na iteração i, uma configuração $c_i \in \mathcal{I}^{i-1}$ que, juntamente com o resultado dessa exploração, será adicionada ao conjunto de amostras, i.e., $\mathcal{A}^i = \mathcal{A}^{i-1} \cup \langle c_i, \mathit{custo}(c_i)\rangle$.

Como referido anteriormente, assume-se também que o algoritmo possui um orçamento O^0 para a criação do modelo e que a sua execução deve terminar ou quando este orçamento é esgotado ou se a redução de custo atingível ao explorar mais configurações, indicada com U_c, é prevista ser marginal (i.e., menor que uma constante ϵ configurável pelo utilizador). A configuração selecionada no final será a melhor configuração (menor custo e tempo de execução a respeitar a restrição) pertencente ao conjunto de amostras. A cada iteração, o custo de exploração da configuração c_i é deduzido do orçamento disponível, i.e., $O^i = O^{i-1} - \mathit{custo}(c_i)$. Assim, em cada iteração i, o estado S_i do algoritmo é um quarteto $S_i = \langle O_i, c_i, \mathcal{A}_i, \mathcal{I}\rangle$ em que O_i corresponde ao orçamento disponível, c_i denota a configuração atual, $\mathcal{A}_i$ é o conjunto das amostras e $\mathcal{I}$ é o conjunto das configurações por explorar. Considera-se, de seguida, o algoritmo. Por facilidade de exposição, descreve-se primeiro uma versão simplificada que na prática não é exequível devido à vastidão do espaço de procura. Depois, descrevem-se as alterações necessárias para tornar o algoritmo utilizável na prática.

Inicialização do algoritmo. O algoritmo é iniciado construindo um modelo base através da exploração de k configurações escolhidas aleatoriamente entre todas as configurações possíveis. A partir deste modelo base, o algoritmo executa um processo de afinação iterativo (Algoritmo 1, linha 3).

Escolha da próxima configuração. Um dos maiores desafios do algoritmo é escolher a configuração $c_i \in \mathcal{I}^{i-1}$ que deve ser explorada em cada iteração. Esta escolha deve contemplar não só o custo de experimentar essa configuração mas também a contribuição esperada dessa experiência para melhorar a qualidade do modelo. Ao aferir a contribuição esperada de uma dada experiência para o

modelo, o algoritmo tem em conta tanto a contribuição do tuplo $\langle c_i, custo(c_i)\rangle$ como a contribuição esperada das iterações seguintes $i+1$, $i+2$, ..., $i+d$ considerando que $\mathcal{I}^i = \mathcal{I}^{i-1} \setminus \{c_i\}$. Ou seja, ao estimar a contribuição de uma dada configuração, é feita uma estimativa realizando uma busca em profundidade, aferindo o efeito dessa escolha até d iterações no futuro. Denotamos a contribuição esperada de um dado percurso de exploração de profundidade d por *contrib_caminho*(c_i, c_{i+1}, ..., c_{i+d}). De forma idêntica, quando se avalia o impacto de uma determinada escolha no orçamento de exploração, considera-se o custo total do caminho *custo_caminho*(c_i, c_{i+1}, ..., c_{i+1}) = $\sum_{j=i}^{i+d} custo(c_j)$.

Idealmente, em cada iteração, o algoritmo calcularia todos os caminhos de exploração de profundidade d, c_i, c_{i+1}, ..., c_{i+d}, considerando todas as configurações que ainda não foram exploradas. Usando esta abordagem, o número de caminhos a analisar é da ordem de $|\mathcal{I}|^d$. Para cada um, o algoritmo calcula a contribuição esperada do caminho para o modelo *contrib_caminho*(c_i, c_{i+1}, ..., c_{i+d}). Posteriormente, aplica a cada caminho uma função que pesa a sua contribuição esperada com o seu custo *custo_caminho*(c_i, c_{i+1}, ..., c_{i+d}). Neste momento, é utilizado o rácio entre a contribuição esperada e o custo. Finalmente, é escolhida como próxima configuração a configuração c_i do caminho que maximiza este rácio e são atualizados os parâmetros do quarteto.

Nos parágrafos seguintes descrevem-se, em pormenor, alguns dos passos fundamentais do algoritmo. Em particular, descreve-se a técnica usada para computar a contribuição esperada de um caminho e quais as heurísticas usadas para evitar o cálculo, a cada iteração, da contribuição esperada de todos os caminhos possíveis (dado que o seu número é exponencial).

3.2 Otimização Bayesiana Sequencial baseada em Modelos

A solução proposta estende a técnica de Otimização Bayesiana Sequencial baseada em Modelos (OBSM) [13], sobre a qual se fornece uma breve explicação.

OBSM é uma estratégia de otimização de funções desconhecidas $f : D \to \mathbb{R}$, cuja estimação só é possível através de observações, possivelmente ruidosas, de valores amostrados. Esta estratégia funciona da seguinte forma: *(i)* avaliar a função alvo f em n pontos iniciais $x_1, \ldots, x_n$ e criar um conjunto de treino inicial, T, com os pares $\langle x_i, f(x_i)\rangle$; *(ii)* construir um modelo probabilístico M a partir de T; *(iii)* escolher o próximo ponto x_m de acordo com uma *função de aquisição* $(M, T) \to D$; *(iv)* avaliar a função f no ponto x_m e atualizar M com a amostra obtida ($\langle x_m, f(x_m)\rangle$); *(v)* repetir os passos *(ii)* a *(iv)* até que um critério de paragem seja satisfeito.

Função de aquisição. Na literatura, uma função de aquisição tipicamente utilizada é o *Expected Improvement* (EI) [14], que seleciona a próxima configuração a explorar de acordo com a melhoria que se espera obter face à melhor configuração descoberta até ao momento. Analogamente ao sistema CherryPick [3], pretende-se ter em conta restrições no tempo de execução do algoritmo, as quais podem ser captadas utilizando o *constrained Expected Improvement* (EIc) [11,15]. O EIc pode ser expresso em forma fechada como o produto do EI(x) e a probabilidade de x satisfazer uma dada restrição (Equação 1). No nosso caso, esta restrição

Algorithm 2 Escolha da próxima configuração para explorar

```
1: function PROXIMA_CONFIG(S,profundidade)
2:     M ← CAD(S)                          ▷ Treina um novo Conjunto de Árvores de Decisão (CAD)
3:     V ← {c ∈ I : P(custo(c) ≤ O_i) ≥ β}          ▷ Configs que obedecem ao orçamento
4:     if V = ∅ then return (null, 0)                ▷ Parar exploração
5:     else
6:         ∀c ∈ V, (U_c, C_c) = ESTIMAR_UTILIDADE(S, c, profundidade)
7:         return (c, U_c) : argmax_{c∈V}{U_c/C_c}
8:     end if
9: end function
```

corresponde ao tempo máximo de execução, T_{max}, definido pelo utilizador para o trabalho completar a sua execução.

$$EI_c(x) = EI(x) \cdot P[T(x) \leq T_{max}] \tag{1}$$

Note-se que o tempo T_c necessário para executar o trabalho numa dada configuração c é desconhecido *a priori*. Todavia, este pode ser calculado a partir do custo estimado de execução da configuração, $custo(c)$, e do seu preço por unidade de tempo, $preco(c)$, i.e., $T_c = \frac{custo(c)}{preco(c)}$. O preço por unidade de tempo é conhecido, uma vez que corresponde ao preço cobrado pelos fornecedores de serviços multiplicado pelo número de máquinas utilizadas.

Modelo probabilístico. De forma a que seja possível utilizar o EI, é necessário um modelo M capaz de prever a distribuição de probabilidade do custo esperado de cada configuração. Vários modelos podem ser selecionados para cumprir este objetivo. Neste trabalho, é utilizada uma abordagem baseada na técnica de *bagging ensemble* [4]. Treinamos 10 árvores de decisão a partir de subconjuntos diferentes dos dados de treino, selecionados aleatoriamente.

Assumindo que as previsões do conjunto de árvores seguem uma distribuição gaussiana $\sim \mathcal{N}(\mu_x, \sigma_x^2)$, é possível calcular o EI em forma fechada de acordo com a expressão $EI(x) = \sigma_x[u\Phi(u) + \phi(u)]$, em que $u = \frac{y(x_{min}) - \mu_x}{\sigma_x}$, ϕ representa a função densidade de probabilidade e Φ a função cumulativa de probabilidade de uma distribuição gaussiana. μ_x corresponde à média dos valores computados pelas árvores e σ_x^2 à sua variância.

3.3 Seleção dos Caminhos a Explorar

Tal como antecipado, ao contrário de sistemas [3, 8, 9] baseados em OBSM, pretende-se analisar não apenas a próxima configuração mas sim as d seguintes, de modo a melhorar a estratégia de exploração. Nos parágrafos seguintes, descreve-se em pormenor a técnica proposta para alcançar este objetivo.

Escolha da primeira configuração de um caminho. Como se pode observar no Algoritmo 2, sempre que é necessário selecionar a próxima configuração a explorar, a primeira tarefa consiste em construir o modelo M (Algoritmo 2, linha 2). Devido à vastidão do espaço de procura, estimar a qualidade de todas as configurações por explorar seria muito ineficiente. Como tal, a primeira heurística utilizada para evitar o cálculo da qualidade de todas essas configurações consiste na criação do conjunto das configurações viáveis, $\mathcal{V}$ (Algoritmo 2, linha 3). Este é o conjunto das configurações que são analisadas e inclui apenas as que,

Algorithm 3 Estimação da utilidade

```
1: function ESTIMAR_UTILIDADE(S, c, profundidade)
2:     M ← CAD(S)                                   ▷ Criar novo Conjunto de Árvores de Decisão (CAD)
3:     U ← EI_c(c)                                  ▷ Calcular melhoria esperada com restrição
4:     C ← custo(c)
5:     if profundidade = 0 then return (U, C)
6:     else
7:         (a_j, w_j) ← GH(f_x), j = 1, ..., N      ▷ Coeficientes da quadratura G-H
8:         for j = 1, ..., N do
9:             S' ← ⟨c ; A' ← A ∪ (c, w_j) ; O' ← O − w_j ; I' ← I \{c}⟩
10:            ESCOLHER_PROXIMO(S')
11:            if c' = null then continue           ▷ Não há c' adequado
12:            else
13:                (u, c) ← ESTIMAR_UTILIDADE(S', c', profundidade − 1)
14:                U ← U + γa_j u; C ← C + γa_j c
15:            end if
16:        end for
17:    end if
18:    return (U,C)
19: end function
```

com probabilidade β, se prevê terem um custo inferior ou igual ao orçamento disponível, i.e, $\mathcal{V} = \{c \in \mathcal{I} : P(custo(c) \leq O) \geq \beta\}$.

Redução da complexidade da procura. Generalizando, e considerando que se pretendia efetuar uma procura com profundidade d, o número total de caminhos a explorar seria $|\mathcal{V}|(|\mathcal{V}|-1)\ldots(|\mathcal{V}|-d)$. Para dimensões realistas de $\mathcal{V}$, i.e., de ordem superior a 50 configurações, a complexidade de explorar todos os caminhos é incomportável. Trabalhos recentes [11,15] exploram diferentes abordagens que visam reduzir a complexidade deste problema. Neste trabalho, é utilizada uma heurística conceptualmente semelhante às propostas que explora em largura, isto é, exaustivamente, todas as V configurações de $\mathcal{V}$ no primeiro passo da exploração mas que, para evitar um crescimento exponencial da procura, explora apenas em profundidade nas subsequentes $d-1$ fases de cada caminho.

A procura em largura, (Algoritmo 2, linha 6), associada ao primeiro passo da exploração, solicita a avaliação da utilidade de todas as configurações de $\mathcal{V}$. A procura em profundidade, por sua vez, é descrita pelo Algoritmo 3, e funciona recursivamente. Começa-se por construir o modelo M e estimar a utilidade e o custo da configuração atual. Em seguida, é feito um clone do estado atual para que seja possível retornar das chamadas recursivas sem corromper o estado, mantendo os conjuntos de configurações exploradas e por explorar atualizados. Posteriormente, é selecionada a configuração que se prevê oferecer mais melhorias face ao ótimo naquele momento (Algoritmo 3, linha 10) para que a sua utilidade seja estimada. Este processo é repetido até se atingir a profundidade d desejada. Neste altura, pode retornar-se para a profundidade inicial já com a contribuição de cada caminho estimada.

Contribuição de um Caminho. A contribuição de um caminho corresponde à soma das utilidades de todas as configurações desse caminho. A utilidade de uma configuração corresponde ao seu EIc, calculado de acordo com a Equação 1. Desta maneira, tem-se que $contrib\_caminho(c_i, c_{i+1}, \ldots, c_{i+d}) = U$.

Para avançar em profundidade, a partir de uma configuração de profundidade $i \geq 1$, é necessário estimar a utilidade das configurações de profundidade i',

Algorithm 4 Política para escolha da próxima melhor configuração simulada

```
1: function ESCOLHER_PROXIMO(S)
2:     M ← CAD(S)                          ▷ Criar novo Conjunto de Árvores de Decisão (CAD)
3:     V ← {c ∈ I : P(custo(c) ≤ O) ≥ β}   ▷ Configs que obedecem ao orçamento
4:     return c : argmax_{c∈I}{EIc(c)}
5: end function
```

com $i < i' \leq d$, o que envolve calcular expectativas encadeadas para as quais não há nenhuma expressão em forma fechada. Por conseguinte, estes valores são aproximados utilizando a quadratura *Gauss-Hermite* (G-H), uma forma de quadratura gaussiana para aproximar integrais em que N pontos são amostrados, cada um com um peso distinto. No caso do nosso algoritmo são utilizados 3 pontos para a aproximação. Ao utilizar a quadratura G-H sobre a distribuição Gaussiana associada à previsão gerada pelo modelo, para a configuração atual c a uma profundidade i, obtêm-se N pares (a_j, w_j). w_j corresponde a um possível custo para c e a_j é o peso associado a esse custo. O peso a_j está, de facto, relacionado com a probabilidade do custo ser w_j e é usado para determinar quanto w_j contribui para estimar a utilidade total do caminho. O parâmetro γ (Algoritmo 3, linha 14) é um fator de desconto [19] que permite calibrar o peso da utilidade prevista de uma configuração com base na distância futura (ou seja, o nível de profundidade na procura) em que esta previsão é calculada. Um valor de $\gamma = 0$ anula a contribuição de configurações num caminho de profundidade superior a um. Por sua vez, um valor de $\gamma = 1$ atribui o mesmo peso às utilidades previstas para todas as configurações de um caminho, independentemente da profundidade a que se encontram.

Para cada custo estimado w_j da atual configuração c, ou seja, para cada um dos N pontos, pretende-se prever qual seria a próxima configuração prevista pelo modelo, após ter sido atualizado com a amostra "simulada" (c, w_j). Este processo é realizado de acordo com a política descrita no Algoritmo 4, que, após atualizar o modelo, define o conjunto das configurações viáveis (i.e. que têm probabilidade β de o seu custo não exceder o orçamento) e escolhe, neste conjunto, a configuração que maximiza o EIc.

4 Avaliação

Nesta secção apresentam-se os resultados de um estudo experimental que visa avaliar o algoritmo proposto e quantificar os seus benefícios em comparação com o algoritmo usado no CherryPick [3]. Este foi escolhido não só por ser um sistema que representa o estado da arte e que se baseia na utilização de OB para a nuvem, mas também por ter como objetivo minimizar o custo final incorrido pelo utilizador e por estar sujeito a restrições por ele definidas. Começamos por descrever brevemente a concretização do algoritmo, passando em seguida para a enumeração dos casos de uso e introdução das métricas de comparação utilizadas. Finalmente, avaliamos o desempenho do algoritmo e discutimos os resultados.

Foi desenvolvido um protótipo do algoritmo em Java, recorrendo ao *software Weka* [12] para concretizar as árvores de decisão utilizadas na modelação dos custos das várias configurações. Em particular, são utilizadas árvores de de-

cisão baseadas no algoritmo *RandomTree*. Para a implementação da quadratura *Gauss-Hermite*, foi utilizada a biblioteca *FastGaussQuadrature.jl* [2].

Conjunto de dados. A avaliação do algoritmo foi realizada utilizando um conjunto de dados construído por nós, ao qual se chamou TF. Este conjunto engloba cerca de 300 configurações e foi obtido utilizando a biblioteca *Tensorflow* [1] para o treino de uma rede neuronal convolucional no reconhecimento do conjunto de imagens MNIST. Foram consideradas configurações definidas pelas seguintes características: o tipo (*small*, *medium*, *xlarge* e *2xlarge* da família *t2*) de máquina virtual adquirida na plataforma *Amazon Web Services* (AWS); o número de máquinas, que foi variado de modo a perfazer um número de núcleos pertencente ao intervalo {8, 16, 32, 48, 64, 80, 96, 112}; três hiper-parâmetros de configuração do algoritmo de treino da rede neuronal, nomeadamente, a taxa de aprendizagem ({1e-3, 1e-4, 1e-5}), o tamanho do lote ({16, 256}), e o modo de treino ({síncrono, assíncrono}).

O conjunto de dados é bastante desafiante para ambos os sistemas devido à sua vastidão. Para espaços de procura de dimensões menores, as diferenças entre os dois sistemas são atenuadas, uma vez que, havendo menos configurações para explorar, a tarefa de encontrar soluções ótimas é facilitada.

Métricas de avaliação. Para comparar o nosso algoritmo com o algoritmo utilizado pelo sistema CherryPick [3], recorremos a duas métricas: distância do ótimo (DOPT) e número de explorações (NEX) efetuadas.

A distância do ótimo é obtida calculando a diferença entre o custo da configuração ótima e o custo da configuração escolhida, normalizada com o custo da configuração ótima, i.e., $DOPT = \frac{custo(c_{escolhida}) - custo(c_{opt})}{custo(c_{opt})}$. Assim, quanto menor for a distância do ótimo, melhor a configuração obtida.

Ao comparar o número de configurações exploradas para os dois algoritmos equipados com o mesmo orçamento[1], conseguimos avalia-los de uma perspetiva diferente: a capacidade de maximizar o espaço de procura explorado face a um orçamento limitado, que, como veremos, está normalmente correlacionada com a probabilidade de encontrar uma solução mais próxima da ótima.

Implicações do modelo inicial. Os modelos iniciais dos sistemas são construídos com apenas 5 amostras, correspondente a cerca de 2% do espaço de procura. A escassez de pontos iniciais aumenta a dificuldade da tarefa, visto que o conhecimento existente para guiar a procura é significativamente reduzido.

Para além disto, a seleção das configurações iniciais é de extrema importância, visto que más configurações podem implicar uma grande redução inicial do orçamento, o que limita o número de possíveis explorações futuras. Neste caso, existe a possibilidade da solução encontrada pelo CherryPick ser melhor do que a encontrada pelo OBOL. Pretende-se melhorar este aspeto, através de heurísticas para seleção das configurações iniciais.

[1] Note-se que, na versão original, o CherryPick não possui a noção de orçamento, ou seja, se o valor do orçamento for ultrapassado, a procura não é interrompida. Para uma comparação mais justa, equipámos o CherryPick com esta noção.

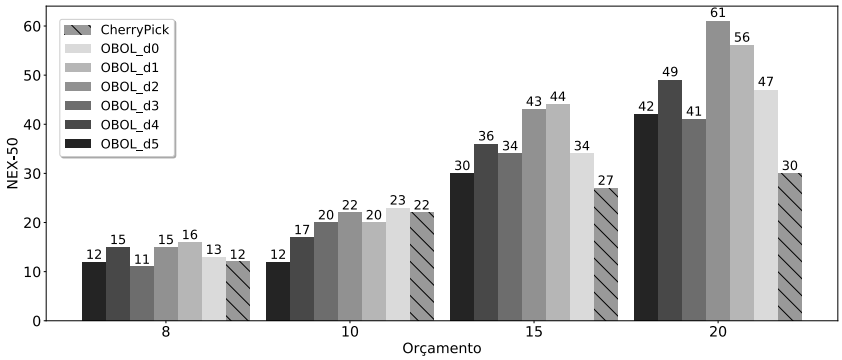

Figura 1: Percentil 50 (mediana) do número de explorações (NEX) para ambos os sistemas e para todas as profundidades testadas do OBOL

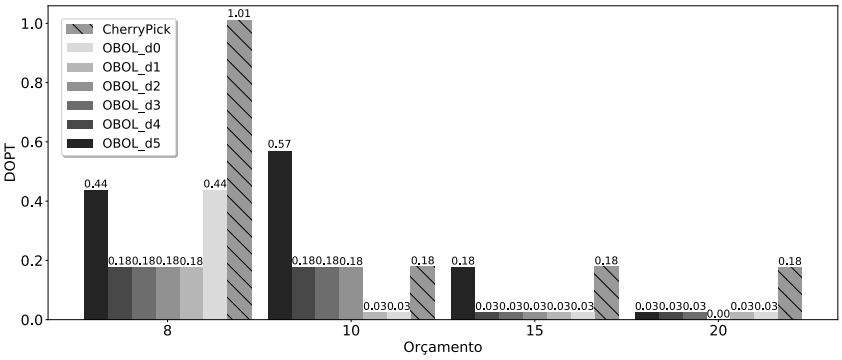

Figura 2: Percentil 50 (mediana) da distância do ótimo (DOPT) para ambos os sistemas e para todas as profundidades testadas do OBOL

Configuração das experiências. Cada experiência é repetida 50 vezes, usando, em cada repetição, 5 configurações iniciais selecionadas aleatoriamente. Foram atribuídos aos parâmetros β e γ valores de 0.99 e de 0.9, respetivamente, como proposto por Remi et.al [15]. Em relação à profundidade, consideraram-se os valores d = {0, 1, 2, 3, 4, 5}. Um valor de profundidade 0 retira ao nosso algoritmo a capacidade de prever o futuro, permitindo estabelecer um patamar para comparação com o CherryPick. Os orçamentos utilizados para as experiências correspondem a {8, 10, 15, 20} vezes o orçamento médio de uma configuração.

Resultados. Podemos observar na Figura 1 que o número de explorações (NEX) tende a aumentar com o incremento do orçamento disponível, sendo este efeito mais acentuado na nossa solução. De facto, enquanto que com orçamento 8, o NEX é equiparável para ambos os sistemas, quando este aumenta para 20, a solução proposta chega a explorar, no melhor caso, o dobro do CherryPick e, no pior, 35% a mais. Visto que a solução proposta consegue explorar um número

maior de configurações, é expectável que consiga identificar uma configuração final mais próxima da ótima. Esta hipótese é confirmada pelos resultados apresentados na Figura 2, em que se reporta o percentil 50 da distância do ótimo (DOPT) para os dois algoritmos. Observa-se que, no caso do OBOL, à medida que o orçamento aumenta, a DOPT diminui, enquanto que, para o CherryPick, esta estagna nos 18%. Verifica-se, também, que para o orçamento 20 e profundidade 2, o OBOL encontra a melhor solução em 50% dos casos, estando apenas a 0.03% do ótimo para as restantes profundidades desse orçamento e para todas as profundidades com orçamento 15.

Cruzando a informação de ambas as figuras, é visível que para a profundidade 5, com orçamentos mais baixos, os ganhos relativos às restantes profundidades são inexistentes. Conclui-se também que, na maior parte dos casos, se obtêm as melhores configurações utilizando profundidade 2.

Discussão. Em geral, a solução encontrada pelo OBOL é melhor do que a encontrada pelo CherryPick devido a dois fatores: à procura em profundidade, que permite estimar a qualidade de conjuntos de configurações e perceber que uma configuração que, num dado momento, não apresenta o melhor EIc (Equação 1) pode levar à ótima; à métrica utilizada para selecionar a próxima configuração a ser explorada. Para selecionar a próxima configuração, o CherryPick apenas analisa o EIc e escolhe explorar a configuração que o maximiza. O OBOL, por sua vez, analisa o rácio entre o EIc e o custo. Assim, escolhe configurações com melhor relação custo/qualidade o que se traduz em poupanças que permitem aumentar o número de explorações e o conhecimento do espaço, aproximando-se mais da solução ótima. Isto justifica que, para orçamentos maiores, o CherryPick estagne, pois, mesmo tendo mais orçamento, as configurações que escolhe explorar são mais caras do que as escolhidas pelo OBOL, levando a que gaste mais rapidamente o seu orçamento, explore menos e encontre piores soluções. Nomeadamente, verifica-se que o método do OBOL é melhor do que o do CherryPick, visto que o OBOL com profundidade 0, que só difere do CherryPick no método de seleção da próxima configuração a explorar, encontra melhores configurações.

5 Conclusões

Neste trabalho abordamos o problema de automatizar a escolha da configuração de uma aplicação para a nuvem, propondo uma solução baseada em técnicas de otimização Bayesiana que visa minimizar o custo de execução da aplicação, tendo em conta restrições predefinidas sobre o tempo máximo de execução da mesma. A característica mais inovadora da solução proposta consiste em incorporar explicitamente, pela primeira vez, as noções de orçamento disponível para a exploração e de planeamento das configurações a explorar de forma a maximizar a rentabilidade do orçamento. Através de uma avaliação experimental baseada num conjunto de dados realista, mostramos que a solução proposta consegue identificar a configuração ótima em 50% dos casos para o orçamento de 20 e explora, com o mesmo orçamento, até 100% mais configurações do que a solução do estado da arte.

Agradecimentos O conjunto de dados usado na avaliação deste trabalho foi obtido com a preciosa ajuda de Diogo Barradas, Manuel Reis e Pedro Joaquim. Este trabalho foi suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projetos com as referências UID/CEC/50021/2013 e PTDC/EEIS-CR/1743/2014.

Referências

1. Abadi, M., et al.: Tensorflow: A system for large-scale machine learning. In: Proc. of OSDI. pp. 265–283 (2016)
2. Alex Townsend: FastGaussQuadrature.jl. `https://github.com/ajt60gaibb/FastGaussQuadrature.jl` (2014)
3. Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: Proc. of NSDI. pp. 469–482 (2017)
4. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
5. Brochu, E., Cora, V.M., de Freitas, N.: A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. CoRR abs/1012.2599 (2010)
6. Delimitrou, C., Kozyrakis, C.: Quasar: resource-efficient and qos-aware cluster management. ACM SIGPLAN Notices 49(4), 127–144 (2014)
7. Delimitrou, C., Kozyrakis, C.: Hcloud: Resource-efficient provisioning in shared cloud systems. In: Proc. of ASPLOS. pp. 473–488 (2016)
8. Didona, D., Diegues, N., Kermarrec, A.M., Guerraoui, R., Neves, R., Romano, P.: Proteustm: Abstraction meets performance in transactional memory. In: Proc. of ASPLOS. pp. 757–771 (2016)
9. Duan, S., Thummala, V., Babu, S.: Tuning database configuration parameters with ituned. Proc. of VLDB 2(1), 1246–1257 (2009)
10. Ekstrand, M., Riedl, J., Konstan, J.: Collaborative filtering recommender systems. Foundations and Trends® in Human–Computer Interaction 4(2), 81–173 (2011)
11. Gardner, J., Kusner, M., Xu, Z., Weinberger, K., Cunningham, J.: Bayesian optimization with inequality constraints. In: Proc. of ICML. pp. 937–945 (2014)
12. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. 11(1), 10–18 (2009)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION'05. pp. 507–523 (2011)
14. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (1998)
15. Lam, R., Willcox, K.: Lookahead bayesian optimization with inequality constraints. In: Proc. of NIPS. pp. 1890–1900 (2017)
16. Lam, R., Willcox, K., Wolpert, D.: Bayesian optimization with a finite budget: An approximate dynamic programming approach. In: Proc. of NIPS. pp. 883–891 (2016)
17. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: Boa: The bayesian optimization algorithm. In: Proc. of GECCO. pp. 525–532 (1999)
18. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative Filtering Recommender Systems, pp. 291–324. Springer Berlin Heidelberg (2007)
19. Sutton, R.S., Barto, A.G., et al.: Reinforcement learning: An introduction. MIT press (1998)