Modelação de Sistemas Não-Deterministas Usando Aprendizagem Automática

Francisco Duarte[1], Richard Gil[1], Paolo Romano[1], Antónia Lopes[2] e Luís Rodrigues[1]

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{francisco.c.duarte,richard.martinez,paolo.romano,ler}@tecnico.ulisboa.pt
[2] Universidade de Lisboa, Faculdade de Ciências, LaSIGE mal@ciencias.ulisboa.pt

Resumo Entre as abordagens que suportam adaptação dinâmica, é possível encontrar duas técnicas distintas para realizar a modelação do sistema: as baseadas no conhecimento de peritos e as baseadas em aprendizagem automática, ambas com a suas desvantagens. As primeiras geram modelos incompletos, imprecisos e que se tornam desatualizados à medida que o sistema evolui. As segundas, requerem longos períodos de treino para conseguir bons resultados. Para além disso, estas últimas em geral são aplicáveis apenas a sistemas deterministas, não sendo capazes de capturar, por exemplo, que uma ação de adaptação pode ter dois ou mais resultados distintos devido a características que não são diretamente observáveis. Neste artigo apresentamos uma abordagem que combina as abordagens acima referidas, de forma a criar modelos de sistemas não-deterministas, que são simultaneamente precisos e compreensíveis, e que podem ser obtidos num curto período de tempo. Esta abordagem foi experimentalmente validada num sistema que realiza o re-dimensionamento dinâmico de uma instalação do RUBiS, uma conhecida aplicação de gestão de leilões na rede.

1 Introdução

À medida que os sistemas informáticos, como por exemplo os sistemas de informação web, se tornam mais complexos, estes também ficam mais difíceis de gerir por operadores humanos. Os sistemas atuais são compostos por muitos componentes, cada um deles com múltiplas opções de configuração e instalação. Esta complexidade dificulta a identificação da configuração que maximiza a utilidade do sistema. Para além disso, estes sistemas operam em ambientes dinâmicos sujeitos a cargas variáveis, onde ocorrem faltas e os componentes têm de ser atualizados frequentemente. Lidar com este dinamismo requer que se realizem múltiplas adaptações do sistema, o que torna a tarefa de gerir um sistema complexo penosa e propensa a erros. Neste contexto, a ideia de automatizar, mesmo que parcialmente, o processo de adaptação torna-se extremamente apelativa. A adaptação automática permite reagir mais rapidamente e de forma mais precisa a eventos que possam afetar negativamente o comportamento do sistema. Para além disso, a adaptação automática também pode contribuir para reduzir os

custos operacionais associados à manutenção do sistema, ao permitir que este seja gerido por equipas de menor dimensão.

Entre as diversas abordagens que têm sido propostas para suportar adaptação dinâmica encontramos dois tipos de técnicas distintas: as que assentam em modelos de adaptação definidos por operadores humanos [3,2] e as que usam modelos construídos através de aprendizagem automática [7,14,18,19]. Por um lado, a utilização de modelos definidos por operadores para guiar a adaptação do sistema permite tirar partido do conhecimento que os peritos têm sobre o comportamento do sistema. O conhecimento dos peritos relativamente ao impacto que diferentes ações de adaptação têm nas propriedades do sistema é particularmente relevante, uma vez que o processo de decisão, presente em sistemas de adaptação automática, envolve frequentemente a comparação de adaptações [2]. Infelizmente, os modelos definidos por peritos são muitas vezes incompletos e imprecisos [5]. Para além disso, é difícil manter estes modelos atualizados à medida que o sistema evolui, por exemplo, quando são feitas atualizações de software ou mudanças na infraestrutura do sistema. Por outro lado, abordagens completamente automáticas que usam aprendizagem "online" conseguem manter os modelos do sistema atualizados, com base na história do sistema, e são capazes de lidar bem com novas situações. Assim, as técnicas automáticas têm potencial para conseguir uma previsão mais precisa do verdadeiro comportamento do sistema. Porém, as abordagens que usam aprendizagem automática tendem a necessitar de um grande conjunto de observações, normalmente recolhidas num longo e exaustivo período de treino [16]. Em qualquer dos casos, uma propriedade desejável nestes modelos é a sua inteligibilidade, a qual facilita o envolvimento humano, fator que consideramos essencial para haver confiança no sistema de adaptação e na sua capacidade de adaptar o sistema corretamente.

Os sistemas adaptativos estão sujeitos a não-determinismo [8]. Neste trabalho, focamo-nos no não-determinismo associado aos efeitos das ações de adaptação. Por exemplo, a ação de ativar um servidor pode não ter sempre o mesmo impacto no sistema, devido a fatores que não são explicitamente modelados, como por exemplo a presença de outros processos em execução na mesma máquina. A informação relacionada com este tipo de incerteza pode ser útil em muitos casos, e.g., quando é necessário planear tendo em conta o pior caso possível. Para que se possa considerar os efeitos do não-determinismo, o modelo deve representar explicitamente esta informação. Pretendemos obter modelos de adaptação com estas características combinando a utilização de modelos especificados por peritos e a utilização de aprendizagem automática. Mais precisamente, a ideia é usar o modelo fornecido pelos peritos no arranque do sistema e usar aprendizagem automática para rever e atualizar estes modelos em tempo de execução, tirando partido da informação entretanto recolhida. O resultado do processo é uma representação textual de um modelo formal atualizado, legível por um operador. Este modelo pode ser usado por um sistema de adaptação para prever o resultado da respetiva ação de adaptação. A capacidade de lidar de forma explícita com o não-determinismo de um sistema no processo de aprendizagem do seu modelo é a principal contribuição deste trabalho.

Neste contexto, propomos uma abordagem nova para refinar modelos de adaptação de sistemas não-deterministas. Primeiro, é recolhido um modelo de impacto para cada ação de adaptação. Estes modelos são fornecidos por peritos usando, por exemplo, o formalismo proposto por Cámara et al. [2]. Estes modelos são usados para criar um conjunto de amostras sintéticas que será usado no processo de aprendizagem. De seguida, novas amostras, baseadas no funcionamento do sistema, são adicionadas ao conjunto. O conjunto de dados com todas as amostras, sintéticas e reais, é analisado pelo algoritmo K-Plane [1]. O resultado desta análise é depois usado para extrair um modelo linear por troços, que é usado para melhorar o modelo original. Esta solução foi experimentalmente validada num sistema que realiza re-dimensionamento dinâmico de uma aplicação web, concretamente o RUBiS.

2 Trabalho Relacionado

A adaptação dinâmica de sistemas é frequentemente gerida por operadores humanos que, ao observarem alterações nas condições de operação, adaptam o sistema executando uma sequência de ações de adaptação. Isto significa que os operadores têm um modelo mental do comportamento do sistema e de como este pode ser adaptado em determinadas situações. Este conhecimento pode ser usado para construir um modelo que permita suportar a adaptação automática do sistema. Este modelo, que tem de ser formal para poder ser processado computacionalmente, deve também ser compreensível para poder ser facilmente validado por outros peritos, aumentando o grau de confiança que os operadores têm no sistema [12]. Existem na literatura vários tipos de modelos concebidos especificamente com este propósito [3,2]. Por outro lado, a aprendizagem automática de modelos de adaptação recorre a dados recolhidos de um sistema em funcionamento, durante um período de treino e, em resultado disso, tem potencial para produzir modelos mais completos e precisos do que os fornecidos por humanos, desde que o conjunto de treino seja suficientemente grande e representativo.

Existem diversas abordagens para a construção de modelos de adaptação baseadas em aprendizagem automática. Podemos dividi-las naquelas que aprendem qual a adaptação que devem realizar num dado estado [19] e as que aprendem qual o efeito de realizar uma ação de adaptação numa determinada circunstância, ou seja, um modelo de impacto [7,14].

No primeiro caso, é aprendido um modelo que descreve como deve ser adaptado o sistema com base no estado em que se encontra, e.g., se o tempo de resposta de um serviço web estiver entre os 2 e 3 segundos, então adiciona dois servidores. No entanto, estes modelos de adaptação dependem dos objetivos de negócio, que podem mudar a qualquer altura. Por exemplo, se um serviço web quiser descer o tempo de resposta médio que os seus utilizadores observam de $500ms$ para $300ms$ então o modelo atual deixa de ser válido.

Por outro lado, se o modelo representar o impacto das ações de adaptação, e.g., ao ligar um novo servidor o tempo de resposta médio diminui $200ms$, al-

terações nos objetivos de negócio não afetam a validade do modelo. O modelo é usado por um sistema de adaptação que decide que ação realizar com base nos objetivos de negócio e no impacto de cada ação (fornecida pelo modelo de impacto). Por este motivo iremos focar-nos em aprender modelos de impacto de ações de adaptação.

Neste trabalho, pretendemos que a técnica de aprendizagem produza modelos com as seguintes caraterísticas: (i) precisos, para evitar adaptações desajustadas que piorem em vez de melhorarem o comportamento do sistema; (ii) legíveis e fáceis de compreender; (iii) capazes de representar explicitamente não-determinismo, ou seja, de capturar mais do que um efeito para uma dada ação.

Existem diversas técnicas de aprendizagem que podem ser aplicadas a este problema, capazes de inferir um modelo preciso com base num conjunto de pontos, como é o caso de "Redes Neuronais" [6] ou "Sistema de Inferência *Fuzzy*" [4]. Apesar destas abordagens serem capazes de aproximar funções não lineares com grande precisão, não são apropriadas para o nosso problema, uma vez que algumas inferem modelos que não são fáceis de traduzir para informação legível ou a informação que fornecem é limitada. Existem algumas abordagens que evitam este problema recorrendo a técnicas de aprendizagem que inferem tipos de modelos legíveis, tais como regras do tipo Evento-Condição-Ação (ECA) [11] ou árvores de modelação [15]. No entanto, nenhuma destas abordagens é capaz de capturar explicitamente efeitos não-deterministas das ações (o impacto estimado seria antes uma média de todos os resultados possíveis). Sykes et al. [17] apresentou uma alternativa capaz de considerar não-determinismo no resultado de ações. Porém, esta solução, assim como outras que aprendem modelos probabilísticos [10], considera apenas espaços de estados discretos, não sendo aplicável quando se quer prever o valor de propriedades com valores contínuos, como por exemplo o tempo de resposta.

Para inferir modelos com estas características, optámos por adotar uma abordagem de "subspace clustering", chamada K-Plane [1]. Este algoritmo identifica os K (hiper-)planos que melhor aproximam um conjunto de pontos num espaço multi-dimensional. Uma vez que estes planos se podem sobrepor no espaço de entrada, para uma determinada região pode existir mais do que uma saída, ou seja, haver mais do que um plano. Considerando que cada plano modela o efeito de uma adaptação, com vários planos é possível capturar os efeitos de adaptações não-deterministas. Para além disso, as funções inferidas são lineares, o que pode tornar a compreensão do modelo, por parte de operadores, mais fácil.

3 Aprender e Atualizar Modelos de Impacto

Para ilustrar os conceitos e técnicas envolvidas no desenho da solução desenvolvida iremos usar como exemplo uma aplicação web. Consideramos que o objetivo da adaptação é manter uma latência de resposta baixa, mas minimizando os custos relacionados com o número de servidores ativos. Assumimos uma configuração padrão, onde os pedidos são feitos a um proxy, que os distribui por um

Listagem 1.1. Estrutura de uma Regra

```
rsp > 0: // condição ou região do espaço de entrada
        [0.9] rsp' = (2/3) * rsp // função de impacto 1
        [0.1] rsp' = rsp // função de impacto 2
```

conjunto de servidores ativos. Nesta aplicação, as ações de adaptação disponíveis são: (i) ligar um novo servidor, para reduzir o tempo de resposta a pedidos de utilizadores (visto que distribui a carga por mais servidores), mas também aumenta os custos de operação; (ii) desligar um servidor ativo, para diminuir o custo, mas que, dependendo da carga, pode aumentar o tempo de resposta aos utilizadores.

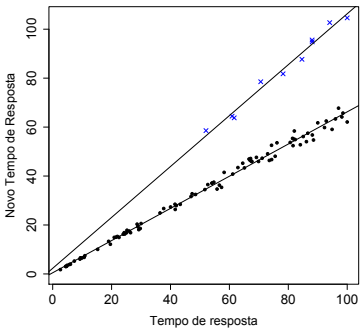
A adaptação da aplicação é controlada por um ciclo MAPE-K [13], responsável por decidir quando servidores devem ser ativados ou desativados, com base no tempo de resposta observado e um modelo do sistema que define o impacto previsto de cada uma das ações de adaptação no tempo de resposta.

Consideramos que os modelos de impacto são expressos numa linguagem semelhante à apresentada por Cámara et al. [2] (aqui simplificada de forma a apresentar apenas os pormenores que interessam para este trabalho). O modelo será constituído por um conjunto de regras, cada uma com uma condição definindo a região do espaço de entrada onde os impactos são válidos, e um conjunto de funções de impacto com a respetiva probabilidade associada. Na Listagem 1.1 apresentamos um exemplo de uma regra. As condições das várias regras do modelo devem ser mutuamente exclusivas, no máximo uma pode ser verdadeira. Para além disso, a soma das probabilidades de cada função de impacto numa regra deve ser igual a 1. Esta linguagem permite representar o facto de que uma ação de adaptação pode ter mais do que um efeito numa propriedade do sistema e ainda a definir a probabilidade de cada um dos casos; cada efeito é uma função do estado do sistema. Neste trabalho consideram-se que estas funções são lineares por troços.

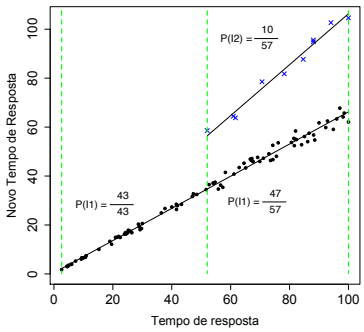
Apesar de termos como objetivo final ter a capacidade de partir de um modelo inicial, fornecido por um perito, e atualizar esse modelo, por falta de espaço iremos apenas descrever o processo de aprender um modelo de impacto sem conhecimento prévio, com base num único conjunto de amostras. Numa instalação real do sistema o conjunto de amostras seria atualizado periodicamente e o mesmo processo repetido de forma a manter o modelo atualizado.

Por amostra, entende-se um tuplo com o estado do sistema antes da ação de adaptação e o valor de uma propriedade alvo, ou seja, aquela que queremos ser capazes de prever. Considerando, por exemplo, que o tempo de resposta médio do sistema gerido decresce de $70ms$ para $50ms$ quando um servidor é ativado, a amostra correspondente seria $\langle 70, 50 \rangle$.

Nas próximas subsecções iremos descrever os passos para aprender um novo modelo, ou seja aprender os diferentes componentes do modelo de impacto, nomeadamente: (i) as funções lineares definidas para cada impacto; (ii) as proba-



(a) Funções de impacto (representadas por retas pretas) em conjunto com os pontos.



(b) Cálculo da probabilidade para cada função de impacto.

Figura 1. Processo de aprendizagem e atualização de Modelos de Impacto

bilidades associadas a cada caso e (iii) o conjunto de casos considerados (e.g., podem ser encontrados novos casos).

3.1 Inferir Funções de Impacto

No primeiro passo do processo usamos o conjunto de amostras, para uma dada ação de adaptação, e encontramos os planos que melhor os representam. Para encontrar estes planos usamos o algoritmo K-Plane [1], que retorna um conjunto de K planos que se adequam a um conjunto de pontos, no nosso caso, amostras, tal como o conjunto de amostras que pertence a cada plano. No entanto, o número de funções de impacto não é conhecido *a priori* na medida em que o não-determinismo surge da ausência de informação suficiente para discernir os comportamentos diferentes. Por este motivo K não é conhecido.

O nosso objetivo é encontrar o menor valor de K que capture os impactos relevantes, sem que o utilizador seja exposto a um grande número de funções de impacto irrelevantes. Para determinar o valor de K, o algoritmo é executado para diferentes valores, começando com $K = 1$, e incrementando K em cada iteração. Para determinar quando parar, usamos "cross-validation" para calcular o erro esperado do modelo, com base num sub-conjunto de amostras (conjunto de teste), para os diferentes valores de K e paramos quando for inferior a um limiar pré-definido, que iremos denominar T.

Para cada amostra do conjunto de teste é calculado o erro, ou seja, o desvio em relação à previsão. Com base nestes valores o erro do modelo é dado pela média do desvio de cada amostra. O erro para uma amostra é dado por:

$$Erro = \frac{abs(real - previsto)}{previsto}$$

Na fórmula *real* é o valor de saída da amostra e *previsto* o valor dado pela função de impacto mais próxima.

Depois de determinar o valor de K, o algoritmo K-Plane é aplicado à totalidade do conjunto de pontos, de forma a obter um modelo mais adequado aos dados. O resultado é um conjunto de planos, nos quais iremos basear as funções do modelo de impacto. Na Figura 1(a) é possível ver um exemplo do resultado.

3.2 Intervalos de Validade

No passo anterior descobrimos K planos, que capturam K funções de um modelo de impacto para uma dada ação. Os planos estendem-se infinitamente, ou seja, não têm limite. No entanto, para que seja possível traduzir o modelo aprendido para a linguagem apresentada é necessário definir uma sub-região do espaço de entrada no qual a função, ou plano, é válida. Por exemplo, ao observar o resultado da Figura 1(a) conseguimos identificar duas funções, mas uma delas só tem amostras para valores no intervalo $]50, 100]$ do espaço de entrada. Por este motivo, não existe razão para assumir que a função ainda é válida quando a entrada não se insere neste intervalo. Por isso, depois de determinar as funções de impacto, identificamos as sub-regiões do espaço de entrada onde as funções são válidas. Este passo permite determinar que a ação tem diferentes resultados quando o tempo de resposta pré-ação está no intervalo $[0, 50]$ ou $]50, 100]$.

Para cada plano, iremos usar os pontos mais próximos a esse plano para calcular o intervalo para cada dimensão de entrada. Considerando uma dimensão, de cada vez, o intervalo será entre o valor mínimo e máximo destes pontos, para essa dimensão. Após este passo temos um intervalo para cada dimensão e para cada plano – um hiper-cubo que engloba o plano. Uma vez que queremos representar explicitamente não-determinismo, ou seja, para uma região podem existir múltiplas funções de impacto, é necessário determinar se diferentes regiões se intersectam e nesse caso dividi-las de forma a mostrar o espaço de validade de cada função e também as regiões onde existe não-determinismo.

Se apenas considerarmos uma dimensão no espaço de entrada, e dois intervalos que se intersectam, então a divisão é feita juntando os limites dos intervalos numa lista, removendo valores duplicados. No nosso exemplo isto iria resultar em $[0, 50, 100]$. De seguida, extraímos os pares sequenciais existentes na lista, ou seja, $(0, 50)$ e $(50, 100)$.

Para um número de dimensões superior, apenas existe uma intersecção se existir uma intersecção em todas as dimensões do espaço de entrada. Como exemplo, iremos considerar duas regiões representadas por $(0 - 100, 0 - 100)$ e $(50 - 150, 50 - 150)$. Para dividir as regiões que se intersectam, consideramos apenas uma dimensão de cada vez, e se para a dimensão considerada o intervalo não for igual, então dividimos o intervalo tal como foi feito para o caso em que temos apenas uma dimensão. Com os intervalos que do processo resultam, criamos novas regiões. No exemplo proposto, o resultado seria: $(0 - 50, 0 - 100)$, $(50 - 100, 0 - 100)$, $(50 - 100, 50 - 150)$ e $(100 - 150, 50 - 150)$.

Listagem 1.2. Modelo Resultante

```
modelo de impacto(ligar servidor)
rsp > 2.6145 & rsp < 52.0478:
        [1] rsp' = rsp
rsp > 52.0478 & rsp < 100.0:
        [0.179] rsp' = 1.04 * rsp + 2.2965
        [0.821] rsp' = 0.6575 * rsp + 0.4741
```

3.3 Cálculo das Probabilidades

O último passo no processo de captura do modelo consiste em calcular a probabilidade associada a cada função, em regiões do espaço de entrada onde foi detetado não-determinismo, i.e., onde existe mais do que uma função de impacto. No nosso exemplo, foi detetado não-determinismo apenas numa das regiões, mais precisamente, no intervalo $]50, 100]$. A probabilidade para cada função de impacto será calculada segundo a seguinte fórmula:

$$P(impacto_i|região) = \frac{\#(região, impacto_i)}{\#(região)}$$

Onde $\#(região)$ é o número de amostras que estão na região a ser considerada, e $\#(região, impacto_i)$ o número de amostras, nessa região, que pertencem à função de impacto i. Isto está representado na Figura 1(b), onde as frações estão ao lado de cada plano.

O modelo que resulta deste processo está representado na Listagem 1.2.

4 Avaliação Experimental

Nesta secção apresentamos a avaliação da nossa solução. Para a avaliação fizemos as experiências com uma instância concreta do problema abstrato apresentado na secção anterior. Concretamente, aplicámos esta solução para atualizar o modelo de impacto usado para suportar o re-dimensionamento dinâmico do RUBiS.[3] O RUBiS é um website de leilões, bastante utilizado, semelhante ao eBay.

4.1 Bancada Experimental

Usámos o RUBiS, versão 1.4.3, numa máquina virtual com um CPU virtual, 2GB de RAM, a correr Ubuntu 14.04. Esta foi instalada num "cluster" de "workstations", cada uma com um processador Quad-Core Intel(R) Xeon(R) de 2.13GHz e 32GB de RAM, ligados por uma rede Ethernet privada Gigabit. Para gerar a carga de trabalho, adaptámos o cliente fornecido com o RUBiS. Para distribuir

[3] Rice University Bidding System: http://rubis.ow2.org

a carga entre os servidores ativos, usámos o HAProxy[4] 1.6, em funcionamento numa máquina virtual distinta.

Tal como no exemplo, a ação de adaptação que tem de ser modelada é a ativação de um servidor. Como seria de esperar, o caso real é mais complexo que o exemplo simplificado. Neste, monitorizámos as seguintes métricas: número de servidores ativos, número de pedidos por segundo e tempo de resposta a pedidos. Tal como no exemplo, o modelo estima o impacto das ações no tempo de resposta. A escolha de métricas a monitorizar podia ter sido mais completa, e.g., considerando ocupação do CPU dos servidores, no entanto, as que foram selecionadas provaram ser suficientes para mostrar os resultados pretendidos. Para além disso, ao usarmos menos variáveis obtemos um modelo mais simples e fácil de compreender. Para introduzir incerteza, considerámos que os servidores podem ter duas configurações de software: o servidor corre exclusivamente o serviço RUBiS, ou também está a correr outros serviços que prejudicam o seu desempenho. Estes serviços são simulados por um "*daemon*" que usa permanentemente 80% do CPU virtual. Assumimos que, quando um servidor é ativado, a sua configuração está fora do controlo do sistema de adaptação. Logo, mesmo que a probabilidade de ativação de cada configuração possa ser conhecida pelo fornecedor da infraestrutura, o modelo do sistema tem de considerar o resultado da ação como sendo não-determinista. Nas nossas experiências, a probabilidade de um novo servidor ser partilhado é de 40%.

Nas próximas subsecções iremos usar *active*, *req* e *rsp* como variáveis, que representam, respetivamente: o número de servidores ativos, o número médio de pedidos feitos por segundo e o tempo de resposta médio.

4.2 Recolha de Dados

Todas as experiências e comparações na avaliação usaram a seguinte metodologia. Recolhemos dados experimentais, usando o caso de estudo descrito nas secções anteriores, com diferentes configurações. Concretamente, com diferentes números de servidores ativos, entre 1 e 4, e diferentes cargas de trabalho, gerado por diferentes números de clientes, de 500 a 5000. Para cada configuração, recolhemos dados ao correr o sistema durante 3 minutos e descartando os primeiros e últimos 30 segundos, que correspondem aos períodos de "warm-up" e "cool-down", respetivamente. Repetimos cada uma das execuções cinco vezes, e descartámos as execuções que estavam abaixo e acima dos percentis 20 e 80, respetivamente. Isto garantiu que, em 90% dos casos o coeficiente de variação [9] do nosso conjunto de dados esteja abaixo dos 9%.

Para comparar a nossa solução com alternativas, o conjunto de dados fornecido foi o mesmo, para que as diferenças nos resultados sejam apenas consequência das abordagens e não flutuações nos dados recolhidos.

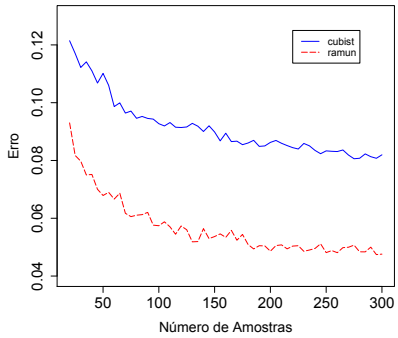


Figura 2. Precisão: Valor do Erro para a Árvore de Regressão (cubist) e solução.

4.3 Precisão

Começámos por comparar a precisão do modelo aprendido com a nossa solução, com a precisão do modelo aprendido usando *cubist* [5]. Optámos por usar *cubist* uma vez que o modelo aprendido se baseia em regras e considera estados contínuos, ou seja, o resultado é semelhante ao da nossa solução – se o sistema estiver num estado a previsão do impacto é dado por uma função linear dependente do estado do sistema.

Os resultados na Figura 2 mostram que a nossa solução consegue uma precisão superior (menor erro) que a alternativa. A diferença entre o erro do *cubist* e o da nossa solução resulta da existência de não-determinismo, que a nossa solução é capaz de capturar. Naturalmente, quanto mais díspar for o comportamento do sistema em função do não-determinismo, maior será o erro do *cubist* em relação à nossa solução.

4.4 Ruído

Um parâmetro crítico para a correção da solução é o limiar do erro (T) que é usado para decidir quando parar de incrementar K, ou seja, quando se espera que o modelo atinja precisão suficiente. A precisão do modelo depende, não só, mas também, da precisão dos valores medidos que são usados para criar o conjunto de pontos. O objetivo desta experiência é mostrar o impacto que o erro nos valores medidos tem na precisão do modelo aprendido, em particular no valor escolhido para K, ou seja no número de funções de impacto.

Todos os dados usados nesta secção foram obtidos experimentalmente e por isso já estão sujeitos a erro de leitura. No entanto, observámos que este erro é relativamente pequeno. Para demonstrar o impacto de erros mais elevados,

[4] Haproxy: the reliable, high performance tcp/http load balance: http://www.haproxy.org/

[5] J. R. Quinlan, "Rulequest Cubist," http://www.rulequest.com, 2016.

Erro \Erro Adicionado	0.00	0.05	0.10	0.15
0.05	2	3	3	4
0.10	1	1	1	2
0.15	1	1	1	1

Tabela 1. Número de funções de impacto para diferentes níveis de "poluição".

usámos os dados para criar um conjunto de pontos "poluído", onde um erro extra é adicionado. Os conjuntos foram criados com erros de 5%, 10% e 15%. Todas as amostras usadas correspondem a adicionar um servidor dedicado. Por isso, para o conjunto de pontos não poluído, não deve existir não-determinismo. Como podemos ver na Tabela 4.4, quando o limiar pré-definido do erro T é menor que o erro das leituras, a nossa solução assume que se encontra perante um caso de não-determinismo, o que indica que o limiar pré-definido do erro deve ser, pelo menos, tão grande como o erro de leitura.

5 Conclusões e Trabalho Futuro

Neste artigo propusemos uma nova abordagem que permite aprender e atualizar os impactos esperados de ações de adaptação de forma dinâmica. A nossa solução difere das existentes em dois aspetos chave: captura e aprende de forma explicita efeitos não-deterministas associados a ações de adaptação, e infere modelos que podem ser facilmente interpretados por operadores humanos. A primeira propriedade permite tornar o modelo mais robusto, mesmo na presença de fatores externos, que não podem ser facilmente medidos ou capturados no modelo. A segunda, permite que o operador humano se possa envolver no ciclo de adaptação; uma propriedade que consideramos essencial para que haja confiança no sistema de adaptação. Avaliámos a solução usando o RUBiS e demonstrámos que conseguimos ganhos significativos na precisão do modelo ao considerar explicitamente o não-determinismo dos efeitos das ações.

Neste artigo foi descrito o processo de aprendizagem de um novo modelo de impacto com base num conjunto de amostras. Numa instalação final, este processo deve ser repetido periodicamente. A arquitetura completa deve também incluir um passo extra de cura do conjunto de amostras, ou seja, uma forma de evitar que o conjunto de pontos cresça sem limites e também de dar menos peso a amostras muito antigas, que podem já não corresponder ao verdadeiro comportamento do sistema. Em trabalhos futuros também devem ser exploradas heurísticas que permitam encontrar o valor ideal de K, na inferência das funções de impacto, de forma mais eficiente.

Agradecimentos Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) e pelo PIDDAC através dos projetos com as referências PTDC/EEI-SCR/1741/2014 (Abyss) e UID/CEC/50021/2013.

Referências

1. Bradley, P.S., Mangasarian, O.L.: k-Plane Clustering. Journal of Global Optimization 16(1) (2000)
2. Cámara, J., Lopes, A., Garlan, D., Schmerl, B.: Adaptation impact and environment models for architecture-based self-adaptive systems. Science of Computer Programming 127(C) (2015)
3. Cheng, S.W., Garlan, D.: Stitch: A language for architecture-based self-adaptation. Journal of Systems and Software 85(12) (2012)
4. Chiu, S.L.: Fuzzy Model Identification Based on Cluster Estimation. Journal of Intelligent and Fuzzy Systems 2(3) (1994)
5. Didona, D., Quaglia, F., Romano, P., Torre, E.: Enhancing performance prediction robustness by combining analytical modeling and machine learning. In: 6th ACM/SPEC International Conference on Performance Engineering (2015)
6. Dietterich, T.G.: Machine learning in ecosystem informatics and sustainability. McGraw Hill series in computer science, McGraw-Hill (2009)
7. Duan, S., Thummala, V., Babu, S.: Tuning Database Configuration Parameters with iTuned. ReCALL 2(1) (2009)
8. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Lecture Notes in Computer Science. vol. 7475 (2013)
9. Everitt, B.S.: The Cambridge Dictionary of Statistics, Cambridge University Press. Cambridge, UK (1998)
10. Filieri, A., Grunske, L., Leva, A.: Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In: International Conference on Software Engineering. vol. 1 (2015)
11. Frömmgen, A., Rehner, R., Lehn, M., Buchmann, A.: Fossa: Learning ECA rules for adaptive distributed systems. In: IEEE International Conference on Autonomic Computing. No. 1, Grenoble, France (2015)
12. Huebscher, M.C., McCann, J.a.: A survey of autonomic computing-degrees, models, and applications. ACM Computing Surveys 40(3) (2008)
13. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1) (2003)
14. Lama, P., Zhou, X.: Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In: 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Miami, Florida, USA (2010)
15. Quinlan, J.R.: C4. 5: programs for machine learning. Elsevier (2014)
16. Sanzo, P.D., Re, F.D., Rughetti, D., Ciciani, B., Quaglia, F.: Regulating concurrency in Software transactional memory: An effective model-based approach. In: International Conference on Self-Adaptive and Self-Organizing Systems (2013)
17. Sykes, D., Corapi, D., Magee, J., Kramer, J., Russo, A., Inoue, K.: Learning revised models for planning in adaptive systems. In: International Conference on Software Engineering. San Francisco, CA, USA (2013)
18. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N., Heights, Y.: IBM Research Report A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In: 3rd International Conference on Autonomic Computing. Dublin, Ireland (2006)
19. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. Cluster Computing 11(3) (2008)