Auto-Configuração de Bases de dados NoSQL Multi-Dimensionais

Nuno Diegues, Muhammet Orazov, João Paiva, Luís Rodrigues, Paolo Romano

INESC-ID / Instituto Superior Técnico, Universidade Técnica de Lisboa, Portugal
{nmld, muhammet.orazov, joao.paiva, ler, paolo.romano}@ist.utl.pt

Resumo Os sistemas de armazenamento chave-valor caracterizam-se por exibirem elevado desempenho e escalabilidade. No entanto, a sua interface é bastante limitada, só permitindo aceder aos objectos através da sua chave primária. Trabalhos recentes corrigem esta limitação, permitindo o acesso a dados por atributos secundários, em particular através da projecção dos objectos em espaços multi-dimensionais. Estas soluções pecam, no entanto, por exigirem um complexo trabalho de configuração. Este artigo contribui para o desenvolvimento de técnicas de configuração automática para este tipo de sistemas, estudando este problema para o caso concreto do HyperDex. Através de uma análise pormenorizado do funcionamento do HyperDex, derivamos um modelo analítico que captura o seu desempenho. Com base neste modelo, desenvolvemos uma metodologia que permite fazer a configuração automática do HyperDex.

1 Introdução

Os sistemas de armazenamento por chave-valor, frequentemente designados por NoSQL (por oposição às bases de dados clássicas), são hoje muito usados em ambientes distribuídos com o intuito primário de garantir escalabilidade e desempenho. Estes sistemas tipicamente adoptam interfaces simples, em que os objectos são acessíveis apenas pela chave com que foram inseridos. A BigTable [2], o Dynamo [4], ou o Cassandra [12], são exemplos destes sistemas.

No entanto, aceder a um objecto apenas pela sua chave é bastante restritivo. Considere-se o exemplo de um website de reservas de quartos de hotel: é fácil perceber que o sistema necessita de suportar pesquisas de hotéis por localização ou por preço. É portanto importante que se possam fazer pesquisas por objectos, que neste caso representam hotéis, por outros atributos que não a chave. Recentemente, surgiram propostas que usam projecções multi-dimensionais entre esquemas chave-valor e redes sobrepostas de servidores "entre-pares" [6,9]. Destes, o HyperDex [6] reúne um conjunto de características que o torna muito interessante para suportar pesquisas complexas. O funcionamento do HyperDex baseia-se no conceito de dispersão hiperespacial (*hyperspace hashing*), que tem semelhanças com a dispersão coerente (*consistent hashing*) usada nos sistema entre-pares clássicos [10]. Resumidamente, um objecto com um conjunto de atributos $\mathcal{A}$ é projectado num espaço Euclideano com $|\mathcal{A}|$ dimensões. Desta forma, usando uma função de dispersão sobre cada valor dos atributos de um objecto, é possível projectar cada resultado para um ponto no espaço interpretando-o como

um vector de coordenadas. A ideia é atribuir conjuntos de pontos do espaço (ou seja, regiões) a servidores, distribuindo assim os dados.

Assente nesta ideia, o HyperDex disponibiliza uma interface rica com suporte para pesquisas parciais sobre quaisquer atributos. O objectivo do sistema é suportar com bom desempenho as pesquisas parciais, sem exacerbar o custo das inserções e modificações. O HyperDex permite que o programador configure o espaço Euclideano da forma que melhor serve a sua aplicação. Como veremos, diferentes configurações afectam drasticamente o desempenho do sistema, com disparidades que vão até 47× (medidas nos nossos testes). No entanto, encontrar a configuração óptima está longe de ser uma tarefa trivial de executar manualmente. Em primeiro lugar, o número de possíveis configurações cresce exponencialmente com o número de atributos considerados. Em segundo lugar, as técnicas e mecanismos usados pelo HyperDex são complexas, o que dificulta a tarefa de identificar qual a configuração adequada a cada cenário. Isto motiva o principal objectivo deste artigo, que consiste em desenvolver técnicas que permitam a configuração automática do HyperDex.

Neste artigo estudamos a técnica de dispersão hiperespacial em pormenor e em particular o funcionamento do HyperDex, tanto de uma perspectiva analítica como experimental. Apresentamos duas contribuições com o objectivo de maximizar o desempenho do Hyperdex para um dado padrão de carga e uma configuração da rede de servidores: 1) um modelo preditivo do desempenho do HyperDex para uma dada configuração, obtendo uma precisão média de 92%; e 2) uma arquitectura genérica que tira partido da contribuição anterior e que permite configurar o HyperDex de forma automática.

O artigo está organizado da seguinte forma. A Secção 2 apresenta uma descrição do HyperDex. De seguida, na Secção 3, derivamos um modelo analítico do HyperDex que é seguidamente validado na Secção 4. Na Secção 5, apresentamos a arquitectura da nossa solução para configurar automaticamente o HyperDex e avaliamo-la contra um conjunto de heurísticas diversas. Na Secção 6, revemos o trabalho relacionado, e concluímos o artigo na Secção 7.

2 Descrição do HyperDex

Um dos objectivos principais do HyperDex é suportar a pesquisa eficiente usando atributos secundários. Distingue-se de outras soluções (menos eficientes) que mantêm índices secundários ou requerem que a pesquisa seja feita envolvendo a maioria dos servidores. A ideia passa por usar dispersão hiperespacial, em que o sistema calcula deterministicamente o conjunto mínimo de servidores que podem ter resultados para a pesquisa.

2.1 Dispersão Hiperespacial

Um hiperespaço é um espaço Euclideano com N dimensões. Cada ponto do hiperespaço contém um valor produzido pela função de dispersão usada. A dispersão hiperespacial é uma técnica que usa um ou vários hiperespaços e projecta um atributo de um objecto numa dada dimensão (de um hiperespaço).

Consideremos um qualquer conjunto de atributos $\mathcal{A}$ tal que $|\mathcal{A}| = N$. Consideremos também um hiperespaço com N dimensões, tal que cada dimensão i

está associada a um atributo $\mathcal{A}_i \in \{\mathcal{A}_1, ..., \mathcal{A}_N\}$. A dispersão hiperespacial permite localizar um objecto no hiperespaço aplicando uma função de dispersão ao valor de cada atributo $\mathcal{A}_i$ do objecto. Este valor funciona como uma coordenada nessa dimensão i; assim obtém-se um conjunto de N coordenadas que correspondem a um ponto no hiperespaço. Para ser usada num sistema distribuído, esta técnica pressupõe ainda o particionamento do espaço em regiões disjuntas que são atribuídas a diferentes servidores. Desta forma, um objecto é armazenado num dado servidor apenas quando as suas coordenadas pertencem a uma região controlada por esse servidor.

Até aqui considerámos apenas a existência de um único hiperespaço, com tantas dimensões quantos os atributos dos objectos. No entanto, o volume de um hiperespaço aumenta exponencialmente com cada atributo adicional. Desta forma, os dados vão ficando cada vez mais rarefeitos no volume, o que dificulta o particionamento do espaço em regiões com a mesma massa, impedindo uma distribuição equilibrada dos dados nos servidores.

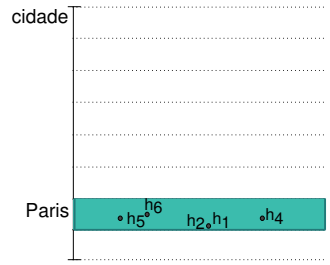
A solução apresentada no HyperDex é de usar vários sub-espaços, cada um com menor número de dimensões, ao invés de um único hiperespaço. Esta estratégia aumenta no entanto a complexidade da configuração do sistema: o programador tem que definir o conjunto de sub-espaços (denotado por $\mathcal{S}$), e para cada $\mathcal{S}_i \in \mathcal{S}$, quais os atributos $\langle \mathcal{A}_1, ..., \mathcal{A}_k \rangle$ usados, em que $\mathcal{A}_i$ é um qualquer atributo do objecto. De notar que o número de regiões num hiperespaço é proporcional a $O(2^{|\mathcal{A}|})$, enquanto nos sub-espaços é proporcional a $O(|\mathcal{S}| \times 2^{|\mathcal{S}_i|})$.

Ao longo do artigo usaremos como exemplo o serviço de hotéis já mencionado brevemente. Cada hotel é um objecto com vários atributos, entre os quais chave primária (o nome), categoria, preço, campos da morada, entre outros. Nas Figs. 1a e 1c ilustram-se dois possíveis sub-espaços com as suas regiões (distribuídas pelos servidores) e alguns pontos a representar hotéis. Considerando uma pesquisa por hotéis em Paris: usando o sub-espaço da Fig. 1a é necessário contactar apenas 1 região, enquanto que na Fig. 1c é necessário contactar 3. Se a pesquisa também especificar um preço de 120, será contactada apenas uma região em ambos os casos. De notar que, independentemente do número de dimensões de um sub-espaço, é importante que o número de regiões em que este é dividido seja sempre razoável. A estratégia adoptada no HyperDex é de dividir cada dimensão de um sub-espaço de forma a que o número de regiões total seja um valor pre-definido $\mathcal{R}$ (ou o mais perto disso possível).

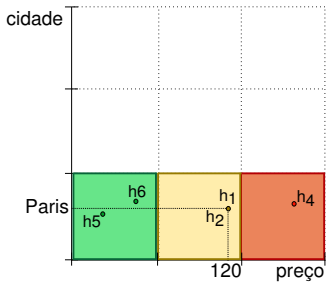
Na Fig. 1b apresentamos uma experiência bastante elucidativa do impacto da configuração do HyperDex no seu desempenho: ao configurar o HyperDex com um hiperespaço, ou com a melhor combinação possível de sub-espaços, o desempenho pode variar 8× a 47× dependendo rácio entre pesquisas e actualizações. Infelizmente, como veremos, não é trivial escolher a melhor configuração manualmente.

2.2 Operação de Pesquisa

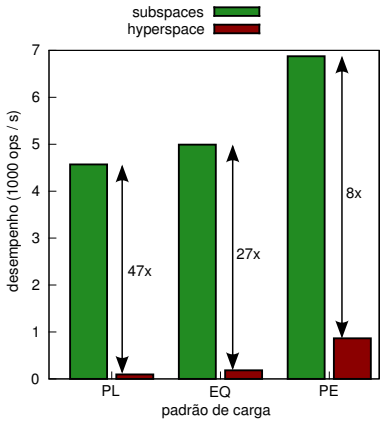
Define-se uma pesquisa $\mathcal{Q}_i$ sobre um conjunto de atributos (e respectivos valores). Usando como exemplo a Fig. 1c, uma pesquisa $\mathcal{Q}_i = \langle \textit{cidade}, \textit{preço} \rangle$ leva a contactar apenas uma região (com um preço de 120, seria a região do meio inferior). Para tal, o cliente escolhe um dos sub-espaços existentes na configuração, que lhe é fornecido por um coordenador central tolerante a falhas.



(a) sub-espaço $\langle cidade \rangle$.



(c) sub-espaço $\langle cidade, preço \rangle$.



(b) Desempenho do HyperDex com apenas um hiperespaço ou usando uma configuração mais complexa com sub-espaços. Estes resultados correspondem a três variantes do mesmo padrão de carga, mudando a percentagem de pesquisas.

Figura 1: Na esquerda, um exemplo de duas diferentes configurações. Na direita, resultados de desempenho que motivam o trabalho aqui apresentado.

Para executar uma pesquisa é necessário enviar uma mensagem para os servidores responsáveis pelas regiões cobertas. Cada servidor, executa a pesquisa localmente e retorna apenas os resultados relevantes. O número de servidores contactados varia conforme o sub-espaço escolhido e a definição completa ou parcial da pesquisa em relação a este; por exemplo, ao pesquisar apenas por Paris no sub-espaço $\langle cidade, preço \rangle$, todas as três regiões são contactadas. De notar que o HyperDex mantém uma cópia integral dos objectos em cada sub-espaço, e não apenas os atributos projectados nas respectivas dimensões dos sub-espaços.

2.3 Operação de Modificação

Nesta secção descrevemos como se processa a modificação de objectos no HyperDex. Assumimos que um objecto é acedido através da sua chave primária, e de seguida inserido com alguns atributos modificados. A operação de acesso recorre a um sub-espaço da chave primária, com uma única dimensão, e presente em todas as configurações por omissão. Enquanto que uma pesquisa utiliza apenas um sub-espaço $\mathcal{S}_i$, de entre os disponíveis, i.e., aquele que é mais propício à pesquisa, nas modificações todos os sub-espaços necessitam de ser actualizados dado que têm cópias integrais dos objectos. Para mais, o HyperDex pode ser configurado com um parâmetro $\mathcal{K} = f + 1$ para tolerar f falhas. Isto garante que um sub-espaço está sempre disponível já que as $\mathcal{K}$ réplicas são distribuídas para diferentes servidores. Logo cada objecto está replicado $(|\mathcal{S}| + 1) \times \mathcal{K}$ vezes.

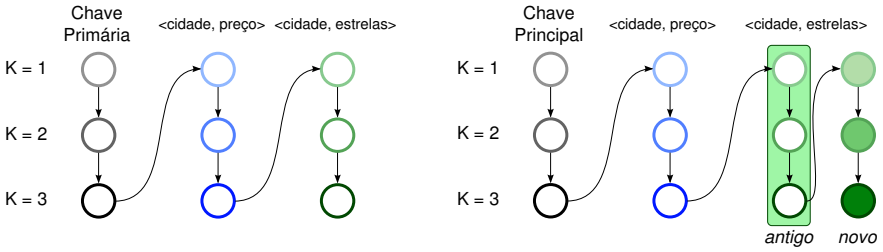
O HyperDex recorre à técnica de replicação em cadeia [16] para manter a coerência das várias cópias dos objectos que são armazenadas nos diferentes sub-

espaços. Dado que as coordenadas dos objectos nos espaços dependem do seu conteúdo, o HyperDex organiza as cadeiras de replicação utilizando uma técnica chamada "encadeamento dependente do valor", em que a cadeia de um objecto depende dos valores dos seus atributos. Quando um atributo é modificado, a posição do objecto pode mudar, o que leva a que novos servidores tenham que participar na cadeia. A Fig. 2 apresenta dois exemplos de cadeias de replicação para diferentes modificações usando a mesma configuração.

Consideremos uma modificação $\mathcal{Q} = \langle telefone \rangle$ em que alteramos o telefone de um dado hotel (omitimos os atributos que não mudam), como ilustrado na Fig. 2a. Neste caso são contactadas 3 réplicas em cada sub-espaço para actualizar o telefone do hotel. Na Fig. 2b a modificação $\mathcal{Q} = \langle estrelas, telefone \rangle$ altera adicionalmente a categoria do hotel. Isto leva a uma cadeia mais complexa porque o atributo *estrelas* está presente na dimensão de um dos sub-espaços. Ao mudar o seu valor, o hotel muda de posição no sub-espaço $\langle cidade, estrelas \rangle$, o que com elevada probabilidade o leva a mudar de região. É por essa razão que apresentamos duas sub-cadeias etiquetadas como *antiga* e *nova*: ao mover o hotel para a nova região, o servidor *antigo* apaga-o dos seus dados enquanto que o *novo* insere o hotel. Já os servidores na sub-cadeia da chave primária e no sub-espaço $\langle cidade, preço \rangle$ actualizam meramente os valores dos atributos do hotel. De notar que em operações subsequentes a cadeia de replicação já não envolve os servidores presentes na parte *antiga*.

3 Modelação Analítica do Desempenho do HyperDex

Nas secções que se seguem usamos a análise feita ao funcionamento interno do HyperDex para derivar um modelo analítico que captura o seu desempenho. Para tal, estudamos individualmente as operações de pesquisa e modificação. Para este modelo assumimos que o sistema é limitado pela capacidade de processamento dos servidores e não pela largura de banda da rede (caso contrário, não existem vantagens em usar uma solução distribuída). O modelo assume também que se pretende configurar o sistema para o débito máximo. Assume-se finalmente que



(a) Modificação $\langle telefone \rangle$. (b) Modificação $\langle estrelas, telefone \rangle$.

Figura 2: Cadeia de replicação resultante de duas modificações diferentes para uma mesma configuração do HyperDex.

os objectos estão uniformemente distribuídos pelas regiões, o que é razoável em aplicações com dados diversos recorrendo a uma função de dispersão uniforme.

3.1 Modelação de Pesquisas

Na situação de carga máxima, o custo de pesquisar no HyperDex é função do número de regiões contactadas. Considere-se uma determinada pesquisa $\mathcal{Q}_i$.

Hipótese 1 *Uma pesquisa $\mathcal{Q}_i$ tem o maior custo quando $\nexists_{\mathcal{S}_i \in \mathcal{S}} : \mathcal{Q}_i \cap \mathcal{S}_i \neq \emptyset$.*

Intuição. Dado que nenhum sub-espaço contém (pelo menos) um dos atributos pesquisados, então a pesquisa tem que contactar $\mathcal{R}$ regiões (i.e., todas) num qualquer dos sub-espaços. A escolha de sub-espaço é irrelevante dado que todas as regiões deverão estar uniformemente carregadas e distribuídas. □

Hipótese 2 *Todas as configurações em que $\exists_{\mathcal{S}_i \in \mathcal{S}} : \mathcal{S}_i \subseteq \mathcal{Q}_i$ levam ao desempenho óptimo quando se pesquisa por $\mathcal{Q}_i$.*

Intuição. Dado que existem $\mathcal{O}$ objectos espalhados uniformemente em $\mathcal{R}$ regiões, então cada região contém $\frac{\mathcal{O}}{\mathcal{R}}$ objectos. Cada atributo em $\mathcal{S}_i$ está também contido em $\mathcal{Q}_i$, ou seja a pesquisa define valores para todas as coordenadas em $\mathcal{S}_i$. Logo a pesquisa contacta apenas uma região, cujo servidor processa $\frac{\mathcal{O}}{\mathcal{R}}$ objectos. □

Hipótese 3 *Para qualquer $\mathcal{S}_i \in \mathcal{S}$ e $\mathcal{Q}_i$, o número de regiões contactado é:*

$$\mathcal{CR}^{exp}(\mathcal{Q}_i) = \sqrt[|\mathcal{S}_i|]{\mathcal{R}}^{|E|} \quad \text{em que: } E = \mathcal{S}_i \setminus \mathcal{Q}_i \tag{1}$$

Intuição. O conjunto E representa os atributos presentes em $\mathcal{S}_i$ que não estão definidos pela pesquisa parcial $\mathcal{Q}_i$. Para cada um destes atributos não definidos, todas as regiões ao longo da dimensão, respectivamente não definida, têm que ser contactadas. Regra geral, para assegurar um número de regiões total $\mathcal{R}$ por sub-espaço, cada dimensão é dividida em $\sqrt[|\mathcal{S}_i|]{\mathcal{R}}$ partições. Como resultado, o número de regiões contactado é o produto deste número de partições $|E|$ vezes, dado que este é o número de dimensões não definidas pela pesquisa. □

Podemos agora estimar o custo de uma pesquisa. Este é proporcional ao produto do número de regiões contactado (dado pela Equação (1)) pelo número de objectos em cada região. Para obter uma estimativa absoluta de desempenho, consideramos um factor β, que é um custo constante associado a processar um objecto e que é dependente do hardware do sistema. Desta forma, o desempenho expectável para uma pesquisa $\mathcal{Q}_i$ que usa um sub-espaço $\mathcal{S}_i$ é obtido por:

$$T^{exp}(\mathcal{Q}_i) = \frac{1}{custo(\mathcal{Q}_i)} \quad \text{em que: } custo(\mathcal{Q}_i) = \sqrt[|\mathcal{S}_i|]{\mathcal{R}}^{|E|} \times \frac{\mathcal{O}}{\mathcal{R}} \times \beta \tag{2}$$

Consideremos agora padrões de carga em que existem várias pesquisas $\mathcal{Q}$, e cada pesquisa $\mathcal{Q}_i$ ocorre com uma probabilidade p_i. Naturalmente, a soma destas probabilidades é 1. Definimos então um conjunto de pesquisas $\mathcal{Q}^S$ composto por todas as $\mathcal{Q}_i$. Desta forma podemos estimar o desempenho do sistema usando uma combinação linear dos custos de cada pesquisa (usando a Equação (2)):

$$T^{exp}(\mathcal{Q}_i) = \frac{1}{\sum_{i=0}^{|\mathcal{Q}^S|} (custo(\mathcal{Q}_i) \times p_i)} \tag{3}$$

3.2 Modelação de Modificações

A partir da análise do funcionamento das operações de modifição no HyperDex, prevemos um custo proporcional ao comprimento da cadeia de replicação.

Hipótese 4 *O custo de uma modificação é proporcional ao comprimento da cadeia de replicação envolvida:* $comprimento(\mathcal{Q}_i) = \mathcal{K}(1 + |\mathcal{N}| + 2|\mathcal{M}|)$.

Intuição. Existe sempre uma parte da cadeia proporcional ao produto do número de sub-espaços ($|\mathcal{S}|$) pelo grau de replicação ($\mathcal{K}$). Para além disso, é necessário ter em conta o sub-espaço da chave primária (não incluído em $\mathcal{S}$). O comprimento da cadeia representada na Fig. 2a é $(1 + |\mathcal{S}|) \times \mathcal{K} = (1 + 2) \times 3 = 9$. No caso geral, temos de admitir que os atributos dos sub-espaços podem ser modificados, como representado na Fig. 2b. Nesta situação, existem servidores adicionais na cadeia — os sub-espaços que são modificados levam a duas sub-cadeias em vez de apenas uma. Assim, definimos $\mathcal{S} = \mathcal{N} \cup \mathcal{M}$, em que $\mathcal{N} = \{\forall_{\mathcal{S}_i \in \mathcal{S}} : \mathcal{Q}_i \cap \mathcal{S}_i = \emptyset\}$ e $\mathcal{M} = \{\forall_{\mathcal{S}_i \in \mathcal{S}} : \mathcal{Q}_i \cap \mathcal{S}_i \neq \emptyset\}$. □

A abordagem anterior considera que cada servidor executa um esforço similar. Para obter uma estimativa mais precisa, é necessário avaliar cuidadosamente a quantidade de processamento associada a uma operação de modificação.

Hipótese 5 *O custo de uma modificação tem de ser corrigido por um factor* α.

Intuição. De facto, podem existir diferenças de processamento conforme uma modificação $\mathcal{Q}_i$ muda um atributo de um sub-espaço ou não. Usando o exemplo anterior na Fig. 2b, um sub-espaço que não é modificado necessita apenas de actualizar a cópia local do objecto, usando a operação local SUBSTITUIR. Por outro lado, um sub-espaço que é modificado gera duas sub-cadeias, onde os servidores *antigos* devem invocar localmente uma operação de APAGAR e os servidores *novos* devem invocar uma operação de ESCREVER. Acontece que SUBSTITUIR é uma operação de concretização menos onerosa. Consequentemente, introduzimos o factor de correcção α para ter em conta esta diferença. Este factor é proporcional ao número de sub-espaços que são modificados, $|\mathcal{M}|$. Tal como para β, o factor α está dependente da configuração do hardware e da implementação do HyperDex, e deve ser estimado nessas circunstâncias. □

Como apontado anteriormente, uma modificação implica um acesso (pela chave primária) o que corresponde a ter sempre um servidor extra no comprimento da cadeia. Finalmente, consideramos um parâmetro T_{max} que captura o máximo desempenho possível nas condições em estudo. Este parâmetro depende apenas da configuração de hardware e pode ser obtido recorrendo a um cenário simples, como quando $comprimento(\mathcal{Q}_i) = 1$, por exemplo modificando um objecto num hiperespaço configurado apenas com o sub-espaço chave. Concluindo:

$$T^{exp}(\mathcal{Q}_i) = \frac{T_{max}}{1 + \mathcal{K}(1 + |\mathcal{N}| + 2\alpha|\mathcal{M}|)} \tag{4}$$

3.3 Modelação de Cargas Híbridas

Quando o HyperDex é submetido a uma carga que inclui diversos tipos de operações, o seu desempenho pode ser estimado com uma combinação linear dos custos de cada operação ponderada pela probabilidade de ocorrência (analogamente à Equação (3)).

4 Avaliação da Precisão do Modelo

Para avaliar a viabilidade do nosso modelo, usámos um conjunto de dados com informações sobre hotéis nos EUA. A carga do sistema segue dois padrões sintéticos que representam a operação de um sistema real. Cada padrão é composto por pesquisas e modificações com três variantes: principalmente leituras (PL), com 90% pesquisas e 10% modificações; uma variante equilibrada (EQ); e principalmente escritas (PE), com 90% de modificações.

Padrão de carga A: Simula situações onde os utilizadores realizam frequentemente pesquisas muito específicas. As pesquisas são compostas por 4 conjuntos de atributos, com probabilidade e número de atributos especificados crescente. As modificações afectam 2 atributos com igual probabilidade; não são escolhidos nem o mais nem o menos frequentemente pesquisados.

Padrão de carga B: Simula situações em que a maioria dos utilizadores executam pesquisas muito amplas. As pesquisas são idênticas ao anterior, mas com probabilidades invertidas, de modo a que a consulta com um único atributo seja a mais comum. As modificações simulam um cenário em que um dos atributos é actualizado frequentemente (por exemplo, o preço de um hotel), e um conjunto de outros atributos é actualizado menos frequentemente.

4.1 Estimativa de Parâmetros e Ambiente de Execução

Todos os testes foram executados numa configuração de hardware com 9 servidores num cluster privado, ligados através de Gigabit Ethernet. O coordenador foi executado numa máquina dedicada e os restantes 8 servidores apenas serviam operações. Utilizámos um ambiente de teste semelhante ao de [6]: um processo cliente em cada um dos 8 servidores, e cada cliente a executar 32 *threads*.

Relembramos que o nosso modelo inclui três parâmetros que dependem da configuração de hardware. Após estimados, estes podem ser instanciados no nosso modelo, e não mais alterados, independentemente do padrão de carga. Por limitações de espaço, apenas explicamos sucintamente o processo de parametrização. Usámos cenários simples, facilmente sintetizados, para estimar os parâmetros para o nosso ambiente de testes. Naturalmente, tal simplicidade leva a um eventual erro na estimação, ao aplicar o modelo a padrões complexos. No entanto, verificámos que a utilização de apenas 24 execuções (em parte repetidas), de 2 minutos cada, foi suficiente para estimar α com apenas 6.5% de erro face a uma estimativa perfeita.

4.2 Precisão do Modelo

Para avaliar a precisão do nosso modelo, testámos os padrões de carga A e B, com uma amostra de todas as configurações possíveis dados 4 atributos pesquisados e modificados. Seleccionámos esta amostra gerando estimativas de desempenho para todas as configurações possíveis aplicando o nosso modelo, e selecionando as 5 configurações com melhor desempenho, assim como 5 outras escolhidas aleatóriamente entre as restantes. Na Fig. 3, apresentamos o desempenho estimado e medido para as configurações amostradas e para todos os padrões de carga. Idealmente, se os desempenhos fossem estimados sem erro, todos os pontos no gráfico estariam sobre a linha diagonal. Como tal, estes resultados mostram que o nosso modelo prevê com bastante precisão o desempenho real do sistema, uma vez que o erro médio é de apenas 9% com um desvio padrão de 7%.

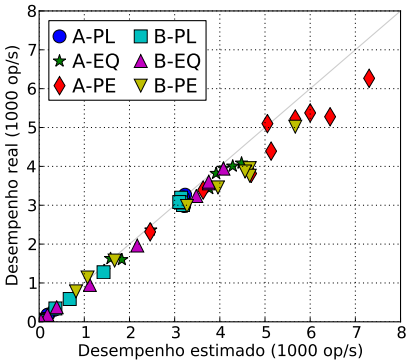


Figura 3: Desempenho estimado e medido para as diferentes configurações em cada padrão de carga.

P. Carga	coef τ	dist $\overline{\tau}$ média	dist $\overline{\tau}$ max
A-PL	0.83	1.2%	11.2%
A-EQ	0.94	2.0%	1.7%
A-PE	0.88	1.7%	13.9%
B-PL	0.72	1.6%	10.5%
B-EQ	0.94	0.1%	1.2%
B-PE	0.88	0.8%	4.9%

Figura 4: Coeficiente τ, média e máxima distancia $\overline{\tau}$ entre a ordem estimada e a verificada para as configurações.

5 Auto-Configuração do HyperDex

Nesta secção, recorremos ao modelo para estimar a melhor configuração dado um padrão de carga. O objectivo é configurar automaticamente o HyperDex sem intervenção do programador ou do gestor do sistema.

5.1 Arquitectura do sistema

A nossa solução é composta por três módulos principais: o *Analisador*, o *Oráculo*, e o *Configurador*. O *Analisador* monitoriza o sistema para gerar um perfil das operações $\mathcal{Q}_i$ realizadas e a sua frequência p_i. Os vários perfis são comprimidos e fornecidos ao *Oráculo* que utiliza o modelo preditivo para determinar qual a configuração que melhor se adequa ao perfil. Para isso, gera todas as combinações de configurações possíveis, consulta o modelo para cada uma delas, e classifica-as por ordem de desempenho. Finalmente, o *Oráculo* activa o *Configurador* com a decisão, e este aplica as mudanças ao HyperDex.

5.2 Avaliação

A melhor escolha: Independentemente da precisão da estimativa de desempenho, o mais importante é que a configuração prevista como sendo a melhor seja efectivamente a óptima. Isto significa que a solução deverá ordenar correctamente as diferentes configurações de acordo com o desempenho de cada uma. A Fig. 4 apresenta o coeficiente de Kendall (τ) [11] como uma métrica que avalia a ordem estimada contra a verificada experimentalmente. O coeficiente τ é uma indicação de como duas ordenações diferem: varia no intervalo $[0, 1]$, onde a precisão da ordem estimada é melhor quando se aproxima de 1. Os resultados apresentados mostram uma correlação alta entre a ordem estimada e a verificada.

O coeficiente de Kendall não é suficientemente expressivo para captar a subtileza da diferença entre a ordem medida e a resultante da aplicação da nossa solução. Apesar de ocasionalmente obtermos ordens diferentes das reais, nessas situações os valores de desempenho previstos e medidos são muito próximos. Assim, este erro tem um efeito reduzido no desempenho final do sistema. Como tal introduzimos a distância de Kendall, que mede o número de pares que têm uma posição relativa diferente em ambas as ordens, e multiplicamos esta distância pelo erro de desempenho resultante, para transmitir a importância relativa dos erros de classificação. Representamos esta distância por $\overline{\tau}$, em que se pretende obter o mais perto possível de 0 (ou seja, a ordem prevista corretamente).

A Tabela 4 apresenta a média e o máximo de $\overline{\tau}$ para cada padrão de carga. A maioria dos pontos tem distância 0; das 60 configurações classificadas, apenas 4 têm $\overline{\tau}$ superior a 2%; e nenhuma é superior a 14%. Logo, apesar da nossa solução não ser perfeita a estimar desempenhos, esses erros não afetam significativamente o desempenho do sistema. Destacamos também que em 5 das 6 cargas, a nossa solução seleccionou a melhor configuração, enquanto que no (B-LP) escolheu a segunda melhor configuração, gerando uma perda de desempenho de apenas 6%.

Comparação com heurísticas: Comparámos o resultado resultante da aplicação da nossa solução contra configurações sugeridas pelas seguintes heurísticas: i) *sem-sub-espaços*, equivalente a um sistema chave-valor comum; ii) *hyperspace*, um único sub-espaço contendo todos os atributos das operações; iii) *todos-sub-espaços*, um sub-espaço de uma dimensão para cada atributo; e iv) *dominante*, onde um sub-espaço é utilizado com o atributo mais comum nas pesquisas.

Na Fig. 5 apresentamos os resultados da nossa solução (denotado por *Automático*) e das heurísticas. Os resultados mostram como a configuração seleccionada pelo nosso modelo resulta sempre no melhor desempenho. Mesmo comparando com a heurística que alcança os melhores resultados, a estratégia *automático* tem um desempenho até 31% superior. De facto, a estratégia *todos-sub-espaços* alcança resultados próximos aos da nossa solução para alguns padrões de carga; no entanto, a sua previsão é afectada pelo aumento da predominância de modificações. Esta é uma consequência directa desta heurística favorecer as operações de pesquisa: utilizando um sub-espaço por atributo, todas as operações de pesquisa contactam apenas uma região. Logo essa escolha prejudica fortemente as modificações, dado que gera um grau de replicação considerável. Finalmente, destacamos também que, ao contrário das heurísticas apresentadas, a nossa estratégia pode adaptar-se às mudanças de padrão de carga, a fim de maximizar o desempenho, enquanto que as restantes são abordagens estáticas.

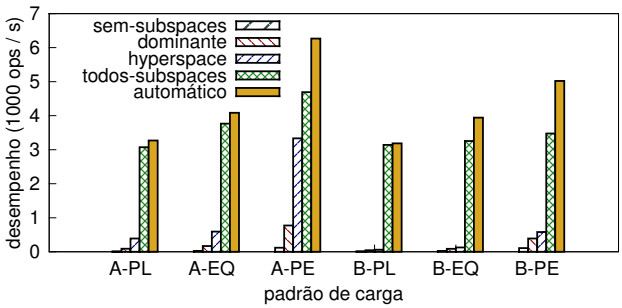


Figura 5: Desempenho da nossa previsão automática contra as heurísticas.

6 Trabalho Relacionado

Os sistemas de armazenamento chave-valor [2,4,12] proporcionam alternativas escaláveis [8,2] e de grande desempenho para armazenar dados. Para tal, estes sistemas são tipicamente baseados em dispersão coerente [10], que particiona com eficácia os dados pelos servidores, simplesmente aplicando uma função de dispersão às chaves associadas aos objectos e aos identificadores dos servidores.

Para fornecer funcionalidade mais rica do que simples operações baseadas na chave do objecto, algumas abordagens recorrem a mecanismos de inundação da rede com mensagens de pesquisa [3], ou inserem o objecto várias vezes no sistema, uma para cada atributo associado ao objecto [13,1]. Ambas as estratégias são ineficientes devido à redundância gerada. Para reduzir o número de servidores contactados, outras abordagens recorrem a curvas de preenchimento de espaço [14]. Esta técnica consiste em projectar um espaço multi-dimensional numa linha uni-dimensional, usada como um anel de dispersão coerente, e preservando a localidade do espaço multi-dimensional. No entanto, ao contrário do HyperDex [6], estas abordagens não escalam com o número de dimensões: a curva faz cada vez menos sentido (preserva cada vez menos localidade) quantos mais atributos o espaço tem. O HyperDex, por outro lado, evita esse problema criando vários sub-espaços, que, como argumentamos neste trabalho, devem ser configurados correctamente para serem bem aproveitados.

A ideia de recorrer a um modelo preditivo de um sistema NoSQL a fim de decidir sobre a sua melhor configuração não é nova. Trabalhos como [15,7,5] aplicam este conceito para controlar o redimensionamento de forma elástica, adaptando o sistema dinamicamente aos padrões de carga e evitando configuração manual. Na realidade, de forma semelhante à nossa solução, o trabalho de Cruz *et.al.* [7] tem também em conta a forma como o particionamento dos dados pelos nós afecta o desempenho do sistema. Estes trabalhos são, porém, orientados à auto-configuração do número de nós de sistemas chave-valor "tradicionais", enquanto o nosso se foca em configurar as dimensões de um sistema multi-dimensional.

7 Conclusões e Trabalho Futuro

Neste trabalho, apresentámos uma solução para a auto-configuração de um sistema de armazenamento de dados NoSQL multi-dimensional. Recorremos a um modelo preditivo para estimar o desempenho do sistema e, usando esta informação, obter a configuração óptima em função do padrão de carga. Mostrámos que a nossa abordagem prevê o desempenho do sistema com 92% de precisão média, e que seleciona a melhor configuração na maioria dos casos. Mostrámos também que os erros na selecção de configuração têm um impacto reduzido, e a nossa auto-configuração supera claramente heurísticas estáticas.

Como trabalho futuro, pretendemos melhorar a precisão das nossas estimativas de desepenho empregando técnicas mais complexas tais como teoria de filas para modelar o efeito de operações simultâneas nos servidores, assim como alterar dinamicamente o HyperDex de acordo com mudanças no padrão de carga.

Agradecimentos Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia através do financiamento pluri-anual do INESC-ID pelo programa PIDDAC, projectos PEst-OE/EEI/LA0021/2013 e CMU-PT/ELE/0030/2009.

Referências

1. A. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04*.
2. F. Chang et. al. Bigtable: a distributed storage system for structured data. In *OSDI '06*.
3. Y. Chawathe et. al. Making gnutella-like p2p systems scalable. In *SIGCOMM '03*.
4. G. DeCandia et. al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Oper. Syst. Rev.*, 41(6), 2007.
5. D. Didona et. al. Transactional auto scaler: elastic scaling of in-memory transactional data grids. In *ICAC '12*.
6. R. Escriva, B. Wong, and E. Sirer. Hyperdex: a distributed, searchable key-value store. In *SIGCOMM '12*.
7. F. Cruz et. al. Met: workload aware elasticity for nosql. In *EuroSys'11*.
8. S. Peluso et. al. When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In *ICDCS '12*.
9. P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04*.
10. D. Karger et. al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97*.
11. M. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2), 1938.
12. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.*, 44(2), 2010.
13. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware '03*.
14. C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8(3), 2004.
15. B. Trushkowsky et. al. The scads director: scaling a distributed storage system under stringent performance requirements. In *FAST'11*.
16. R. van Renesse and F. Schneider. Chain replication for supporting high throughput and availability. In *OSDI'04*.