vCube-CD: Armazenamento Distribuído e Elástico de Dados no vCube com Ajuste Autonômico de Dimensões

Jheferson Renan Borges
Elias P. Duarte Jr.

[1]Departamento de Informática, Universidade Federal do Paraná (UFPR)
Caixa Postal 19018 – 81531-990 – Curitiba/PR

{jrborges,elias}@inf.ufpr.br

Resumo. *O vCube é uma topologia virtual que se constitui a partir de um conjunto de n processos, que formam um hipercubo quando n é uma potência de 2 e todos os processos são corretos. Na medida em que processos falham e se recuperam, o vCube se reorganiza mantendo diversas propriedades logarítmicas. Este trabalho apresenta nossos esforços para definir um vCube capaz de ajustar dinâmica e autonomicamente suas dimensões. A ideia é que os processos mantêm individualmente um repositório de dados, mas com limite de capacidade pré-definido. Os dados são registros (chave,valor) mapeados para um processo existente do vCube. Quando a quantidade de dados armazenados em um processo chega ao limite (ou próximo ao limite), o vCube proposto expande, criando processos e possivelmente aumentando seu número de dimensões. Chamamos a versão de vCube-CD, de Carga Dinâmica. No trabalho, o algoritmo de expansão do vCube-CD é descrito, bem uma implementação através de simulação no PeerSim.*

1. Introdução

Sistemas de armazenamento distribuído de dados (*distributed data store*) são fundamentais e omnipresentes, sendo amplamennte utilizados nos mais diversos *datacenters* [Saadoon et al. 2022]. Um sistema de armazenamento distribuído consiste de uma rede, na qual os dados são mantidos por diversos nodos, podendo inclusive estar replicados [Siddiqa et al. 2017]. O desempenho destes sistemas é crítico, mas não mais que sua confiabilidade e escalabilidade [Wang et al. 2014], bem como segurança [Augustine et al. 2022].

Escalabilidade, neste contexto, se refere à capacidade do sistema de atender demandas das aplicações, mantendo o desempenho e a utilização de recursos em níveis aceitáveis. A elasticidade permite que o sistema obtenha mais recursos para atender à demanda, sempre que o necessário [Galante and de Bona 2012]. Há duas abordagens para elasticidade, horizontal e vertical. Na abordagem horizontal, se acrescenta mais nodos (máquinas) para atender à demanda. Enquanto isso, na abordagem vertical, os recursos da própria máquina (e.g. CPU, memória, etc.) é aumentado. O presente trabalho apresenta uma estratégia para introduzir elasticidade horizontal no vCube, topologia virtual descrita a seguir.

Considere um sistema distribuído síncrono que consiste de n processos que podem falhar por parada (*crash*). O vCube [Duarte et al. 2014] é uma topologia virtual

que permite interconectar os processos de forma escalável. Quando todos os processos estão corretos e n é uma potência de 2, o vCube é um hipercubo. Entretanto, quando processos falham, os processos corretos se reorganizam mantendo diversas propriedades logarítmicas, como a distância máxima entre os processos.

Neste trabalho apresentamos o vCube-CD (Carga Dinâmica), um sistema de armazenamento distribuído baseado no vCube. Os dados são do formato (chave, valor). Cada processo mantém dados armazenados localmente, tendo um limite superior que indica a capacidade de dados máxima que consegue armazenar. O espaço de chaves sempre é mapeado de forma balanceada para os n processos do vCube. A primeira diferença do vCube-CD para o vCube tradicional é que os 2^d processos de um vCube de dimensão d podem ou não estar instanciados. Quando recebe uma solicitação para armazenar dados acima do limite, o processo pode instanciar um vizinho, caso haja algum que resolve o problema. Alternativamente pode causar uma expansão da dimensão da dimensão atual. A expansão é feita até que o processo encontre outro para o qual chaves são mapeadas adequadamente. Observa-se que, nas dimensões superiores, apenas o processo para o qual as chaves são mapeadas é efetivamente instanciado.

Um protótipo foi implementado como prova de conceito no simulador PeerSim[1]. São apresentados resultados que avaliam a dimensão final e o número de processos instanciados no vCube-CD para diferentes espaços de chaves.

O restante do artigo está organizado da seguinte maneira. A próxima seção apresenta o vCube. Em seguida, na Seção 3, o vCube-CD é descrito. A Seção 4 apresenta a implementação e os resultados obtidos. Finalmente a conclusão segue na Seção 5.

2. Background: vCube, um Algoritmo de Diagnóstico Escalável

O vCube foi proposto originalmente como o algoritmo Hi-ADSD (Hierarchical Adaptive Distributed System-level Diagnosis) [Duarte and Nanya 1998]. Um algoritmo de diagnóstico distribuído assume que os processos podem estar em um de dois estados: correto ou falho. Apesar de que o modelo de diagnóstico do vCube assumir falhas por parada (*crash*), há uma outra abordagem denominada diagnóstico baseado em comparações que considera "falhas por computação incorreta" [Ziwich et al. 2005]. O vCube assume que todos os processos podem comunicar diretamente entre si em uma topologia subjacente *fully-connected*. Vale destacar que há estratégias de diagnóstico que assumem topologia arbitrária, isto é, há pares de processos que precisam de intermediários para comunicarem entre si [Duarte Jr and de Oliveira Mattos 2000].

O algoritmo Hi-ADSD permite que os processos se organizem em uma estrutura virtual escalável, que é um hipercubo completo quando o número de processos n é uma potência de 2, e todos os processos estão sem falhas. Entretanto, a grande vantagem da topologia é que os processos se reorganizam na medida em que há falhas, mantendo diversas propriedades logarítmicas. Os processos têm identificadores de $i = 0..(n-1)$, e realizam testes em clusters cada vez maiores, denominados $c_{i,s}$, $s = 1..\log(n)$. Os testes correspondem a arestas direcionadas que saem do processo i são definidas pela função $c_{i,s}$ na expressão (1) em que $\oplus$ é o operador ou exclusivo:

[1]https://peersim.sourceforge.net/

$$c_{i,s} = \{i \oplus 2^{s-1},\ c_{i\oplus 2^{s-1},\,1}, \cdots,\ c_{i\oplus 2^{s-1},s-1}\} \quad (1)$$

No vCube uma aresta direcionada (i, j) corresponde a um teste executado por um testador (processo i) no processo j. Os processos se testam para determinar os estados uns dos outros. Quando um processo correto é testado, o testador obtém informações sobre o estado de outros processos. No algoritmo Hi-ADSD havia situações (raras) que levavam a um número quadrático de testes. Com a motivação de garantir que o número de testes total não superasse $n\log(n)$, foi desenvolvido o algoritmo *Hi-ADSD with Detours* [Duarte Jr and Weber 2003], que permanece com a prova de complexidade em aberto, apesar de diversos esforços realizados, inclusive simulações exaustivas. Mais tarde, a versão que foi denominada VCube [Duarte et al. 2014] é apresentada com a prova de que cada processo é testado por no máximo $\log(n)$ testadores.

Além do número de testes, a latência é outra métrica usada para avaliar os algoritmos de diagnóstico. Os processos executam testes periodicamente, em um intervalo de testes determinado pelos seus relógios locais. Não há sincronização entre todos os relógios. Uma rodada de testes ocorre quando todos os processos corretos terminam a execução de seus testes assinalados. A latência de um algoritmo é determinada como o número máximo de rodadas de testes necessárias para que todos os processos corretos obtenham informações sobre um novo evento. Um evento corresponde à mudança de estado de um processo, de correto para falho ou vice-versa. O vCube apresenta latência de até $\log^2(n)$ rodadas. Em [Brawerman and Duarte 2001] foi proposta uma estratégia para sincronizar o índice s dos clusters $c_{i,s}$ sendo testados por todos os processos. Mais recentemente, o vCube tem sido implementado simplesmente com todos os processos executando todos os testes assinalados em todos os intervalos de testes, o que também garante uma latência de $\log(n)$ rodadas de testes.

Recentemente [Duarte Jr et al. 2022], foi desenvolvido um arcabouço teórico para unificar o diagnóstico distribuído com os detectores de falhas [Chandra and Toueg 1996]. A principal diferença é que o modelo de sistema neste caso é assíncrono, ou seja, não há limites de tempo conhecidos nem para a execução de tarefas, nem para a transmissão de mensagens. Desta forma, um processo correto pode cometer um engano ao testar outro processo correto, mas lento, e suspeitar de que está falho. No presente trabalho, o vCube-CD é proposto para o modelo síncrono, mas deve-se destacar que é possível fazer a transição para o modelo assíncrono utilizando o arcabouço teórico definido em [Duarte Jr et al. 2022].

O vCube já foi aplicado em diversos contextos, inciando pelo monitoramento de falhas de redes locais [Duarte and De Bona 2002]. A topologia também já foi utilizada para a construção de uma rede de sobreposição com suporte à execução de algoritmos paralelos [Bona et al. 2006]. Diversos algoritmos distribuídos já foram definidos sobre o vCube, incluindo estratégias para difusão confiável e ordenada de mensagens [de Araujo et al. 2019].

3. vCube Carga Dinâmica

Nesta seção é descrito o algoritmo distribuído vCube-CD (Carga Dinâmica). O vCube-CD é construído e mantido sobre um sistema distribuído síncrono, no qual há limites

conhecidos para o tempo de execução de tarefas, bem como para a transmissão de mensagens. O número de processos é sempre uma potência de 2. Assim $n = 2^d$, sendo d a dimensão do vCube-CD. Entretanto, vale já ressaltar que nem todos os processos estão instanciados. No vCube-CD os processos instanciados mantém dados, até um limite. Quando um processo recebe uma requisição de armazenamento que excede o limite faz o seguinte. Em primeiro lugar verifica se há processo não instanciado que "resolve o problema", isto é, para o qual são mapeados parte dos dados do processo no limite. Caso houver, tal processo é instanciado. Se não há processo não instanciado que resolva o problema, então o vCube-CD expande, aumenta de dimensão, até que seja encontrado pelo menos um processo para o qual são mapeados parte dos dados do processo no limite. Na nova dimensão, apenas tais processos são instanciados.

Para incrementar a dimensão do vCube-CD basta multiplicar por 2 os identificadores dos processos da dimensão d, criando (sem instanciar, a princípio) os demais processos faltantes na dimensão $d + 1$. Mais a frente, vamos ilustrar a expansão de dimensões, mas antes é necessário destacar que cada processo deve ser identificado pela sequência de duplas (i,d): i é o identificador do processo para o vCube de dimensão d. A estratégia de identificação dos processos é ilustrada e exemplificada abaixo, mas é importante compreender que cada processo mantém a lista de seus identificadores em todas as dimensões das quais participou.

O vCube-CD armazena dados do tipo (chave,valor). Cada chave é mapeada de forma única para o processo no vCube que armazena o dado. Considere que o espaço de chaves vai de $[0..(k-1)]$. O processo que armazena uma chave é determinado de forma única da seguinte maneira. São k/n chaves alocadas para cada processo. O mapeamento da chave é feito de forma muito simples: basta obter o resto da divisão inteira da chave por n: chave MOD n. Entretanto, há uma situação que demanda atenção especial: é o caso do processo mapeado não estar instanciado. Neste caso, a chave é mapeada para o vizinho instanciado no menor cluster. Assim, se chave é mapeada para o processo i que não está instanciado, ela é mapeada para um vizinho instanciado no cluster $c(i, 1)$, se existir algum. Caso contrário, se não há processo instanciado em $c(i, 1)$, mas há em $c(i, 2)$, ele é utilizado. Se houver mais de um vizinho instanciado em um cluster, é usada a ordem estabelecida pela função $c(i, s)$ para definir.

A Figura 1 mostra um exemplo do vCube-CD, iniciando com dimensão 0, portanto um único processo. Para simplificar, só são mostrados os identificadores da última dimensão. O espaço de chaves é de $[0..15]$, e cada processo pode armazenar localmente até 2 unidades de dados no máximo. Inicialmente todas as chaves estão mapeadas para o processo 0. São geradas as chaves 4, 5 que são armazenadas pelo processo 0. Quando é gerada a chave 6, surge a necessidade de expansão para dimensão 1, ficando o processo 0 com as chaves 2, 6; e o processo 1 com a chave 5.

Em seguida é gerada a chave 10, que causa a expansão para a dimensão 2. É instanciado o processo 2, que fica com chaves 6, 10; o processo 3 não é instanciado. Agora são geradas as chaves 7 e 11, que causam o instanciamento do Processo 3, ficando o processo 0 com a chave 4, o processo 1 com a chave 5, o processo 2 com as chaves 6, 10, e o processo 3 com as chaves 7, 11. No passo final do exemplo, é gerada a chave 15, que causa a expansão para a dimensão 3. Só é instanciado o processo 7, que armazena a nova chave 15.

A Figura 1 não mostra, mas é preciso que cada processo mantenha informações sobre a dimensão do sistema no seu identificador. Na expansão do vCube-CD os processos mudam de identificador e assim é necessário usar a dimensão d para identificar de forma única cada processo. Cada identificador é multiplicado por 2, assim o processo (0,1) continua sendo processo (0,2). Entretanto o processo (1,1) passa a ser processo (2,2) na dimensão 2. Os outros processos desta dimensão são: (1,2) e (3,2).

Na estratégia de expansão, o processo (i, d) verifica se é possível instanciar algum processo para distribuir os dados. Na verificação percorre todos os clusters, do menor para o maior. Não basta haver processos ainda não instanciados: é preciso confirmar se as chaves serão mapeadas para aquele processo, caso seja instanciado. Caso não exista tal processo, é preciso expandir a dimensão. O processo que faz a expansão envia uma mensagem por broadcast no vCube informando aos demais a nova dimensão. Cada processo que recebe a mensagem ajusta seu próprio identificador. Observe que dois ou mais processos podem disparar o processo sem prejuízo para a consistência do vCube-CD. Veja que a expansão deve continuar por mais dimensões até que seja encontrado um processo para o qual os dados necessários são mapeados.

4. Experimentos

Nessa seção são apresentados os resultados obtidos da simulação do vCube-CD. A implementação foi feita em Java usando a biblioteca PeerSim. As simulações foram executadas para demonstrar a capacidade de dispersão de dados entre os processos. São definidos o espaço de chaves e os limites de armazenamento local individual de cada processo. Para obter esses resultados, para cada combinação de espaço de chaves e capacidade os experimentos foram executados por 30 iterações. Os espaços de chaves consideram os espaços de chaves de $0..127$, $0..255$, $0..511$ e $0..1023$, com limites de capacidade em 2, 4, 8 e 16. Cada experimento inicia com 0 dados e um único processo (0). As chaves vão sendo incluídas aleatoriamente, são incluídas 50% do total de chaves possíveis.

A Tabela 1 mostra as dimensões máximas, mínimas e médias que resultaram do experimento. É possível ver claramente uma correlação entre o limite da capacidade de armazenamento e as dimensões a que o vCube-CD chega. Quando a capacidade é muito pequena, vão ser necessárias muitas expansões. Quanto maior a capacidade, mais próximo fica da dimensão ótima.

Na Tabela 2 destacamos a comparação entre a quantidade de processos efetivamente instanciados, na média, no mínimo e no máximo para os espaços de chaves e capacidades. É possível ver uma clara correlação entre a dispersão dentro do vCube-CD e o limite da capacidade de armazenamento dos processos individuais.

Agora, a Figura 2 é apresentada com o objetivo de comparar os valores obtidos nos experimentos com o caso ótimo, no qual cada processo tem toda a sua capacidade preenchida, portanto necessitando do número mínimo de processos instanciados. As curvas coloridas representam o valor ótimo para cada capacidade indicada na legenda. O eixo y representa a quantidade de processos instanciados e o eixo x, por sua vez, mostra o valor máximo dos espaços de chaves. Destaca-se que quanto maior a capacidade e o espaço de chaves mais próximo o resultado fica da alocação ótima.

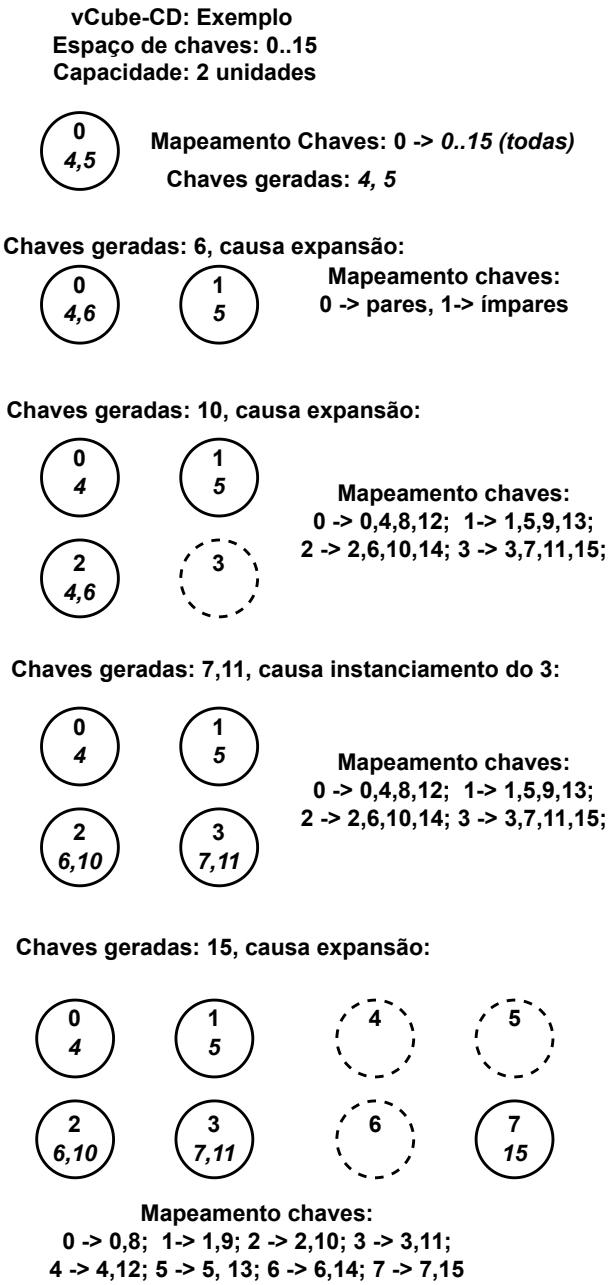

Figura 1. vCube-CD: um exemplo.

Espaço Chaves	Capacidade	Dim. Media	Dim. Máxima	Dim. Miníma
128	2	6,23	7,00	6,00
	4	5,00	5,00	5,00
	8	4,00	4,00	4,00
	16	2,97	3,00	2,00
256	2	7,27	8,00	7,00
	4	6,23	7,00	6,00
	8	5,00	5,00	5,00
	16	4,00	4,00	4,00
512	2	8,90	10,00	8,00
	4	7,33	9,00	7,00
	8	6,03	7,00	6,00
	16	5,00	5,00	5,00
1024	2	10,10	12,00	9,00
	4	8,70	10,00	8,00
	8	7,07	8,00	7,00
	16	6,03	7,00	6,00

Tabela 1. Experimentos vCube-CD: dimensões mínima, máxima e média para diferentes capacidades e espaços de chave.

5. Conclusão

Neste trabalho foi apresentado o algoritmo vCube-CD, que permite que os processos do vCube armazenem dados do tipo (chave,valor). Há um limite na quantidade de dados que um processo consegue armazenar localmente. Uma função faz o mapeamento das chaves para um processo do vCube. O espaço de chaves é mapeado de forma igualitária entre todos os processos do vCube-CD. A principal característica do vCube-CD é que é capaz de expandir caso seja necessário o armazenamento de mais dados. Na expansão, o vCube-CD dobra de tamanho, mas só são instanciados os processos necessários para a distribuição adequada de dados. O vCube-CD foi implementado no simulador PeerSim, e foram realizados experimentos para medir a dimensão resultante, bem como o número de processos instanciados para diversos espaços de chaves e capacidades máximas de armazenamento por cada processo individualmente.

O principal trabalho futuro é a definição de uma estratégia para redução de dimensões, isto é, quando os dados de processos são removidos, pode ser possível desinstanciar processos, e até mesmo encolher as dimensões do vCube-CD. Outro trabalho futuro é a investação funções hash para o mapeamento de chaves para os processos do vCube-CD, efetivamente propondo o vCube-CD como uma DHT (*Distributed Hash Table*). Deve ser feita também a comparação com outras DHTs existentes.

Referências

Augustine, J., Chatterjee, S., and Pandurangan, G. (2022). A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 87–98.

Espaço Chaves	Capacidade	Instanc. Media	Instanc. Máxima	Instanc. Miníma
128	2	37,87	41,00	35,00
	4	20,87	22,00	19,00
	8	11,13	12,00	9,00
	16	5,27	6,00	4,00
256	2	77,23	81,00	73,00
	4	42,67	46,00	38,00
	8	22,17	25,00	20,00
	16	11,27	13,00	9,00
512	2	157,10	164,00	152,00
	4	86,03	92,00	81,00
	8	45,12	48,00	43,00
	16	22,67	25,00	20,00
1024	2	314,50	323,00	303,00
	4	173,20	179,00	164,00
	8	89,47	95,00	85,00
	16	45,63	50,00	42,00

Tabela 2. Experimentos vCube-CD: número mínimo, máximo e médio de processos efetivamente instanciados para diferentes capacidades e espaços de chave.

Bona, L. C., Duarte Jr, E. P., Mello, S. L., and Fonseca, K. V. (2006). Hyperbone: Uma rede overlay baseada em hipercubo virtual sobre a internet. *XXIV Simpósio Brasileiro de Redes de Computadores*.

Brawerman, A. and Duarte, E. P. (2001). An isochronous testing strategy for hierarchical adaptive distributed system-level diagnosis. *Journal of Electronic Testing*, 17:185–195.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267.

de Araujo, J. P., Arantes, L., Duarte Jr, E. P., Rodrigues, L. A., and Sens, P. (2019). Vcube-ps: A causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, 125:18–30.

Duarte, E. P., Bona, L. C., and Ruoso, V. K. (2014). Vcube: A provably scalable distributed diagnosis algorithm. In *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 17–22. IEEE.

Duarte, E. P. and De Bona, L. E. (2002). A dependable snmp-based tool for distributed network management. In *Proceedings International Conference on Dependable Systems and Networks*, pages 279–284. IEEE.

Duarte, E. P. and Nanya, T. (1998). A hierarchical adaptive distributed system-level diagnosis algorithm. *IEEE Transactions on computers*, 47(1):34–45.

Duarte Jr, E. P. and de Oliveira Mattos, G. (2000). Diagnóstico em redes de topologia arbitrária: Um algoritmo baseado em inundaçao de mensagens. In *Anais do II Workshop de Testes e Tolerância a Falhas*, pages 82–87. SBC.

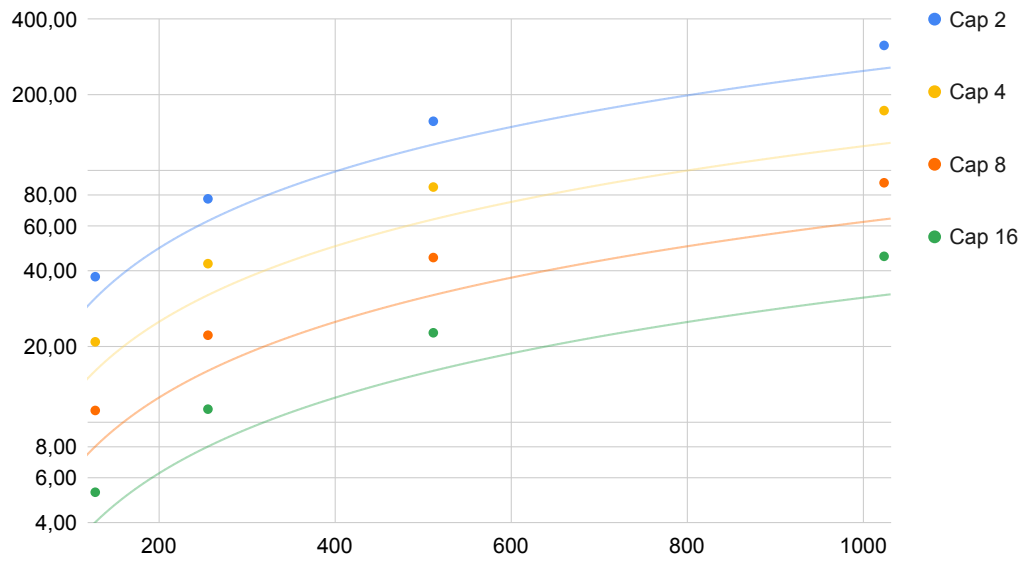

Figura 2. vCube-CD: comparação entre processos ótimos e processos obtidos no experimento dimensão.

Duarte Jr, E. P., Rodrigues, L. A., Camargo, E. T., and Turchetti, R. (2022). A distributed system-level diagnosis model for the implementation of unreliable failure detectors. *arXiv preprint arXiv:2210.02847*.

Duarte Jr, E. P. and Weber, A. (2003). A distributed network connectivity algorithm. In *The Sixth International Symposium on Autonomous Decentralized Systems, 2003. ISADS 2003.*, pages 285–292. IEEE.

Galante, G. and de Bona, L. C. E. (2012). A survey on cloud computing elasticity. In *2012 IEEE fifth international conference on utility and cloud computing*, pages 263–270. IEEE.

Saadoon, M., Hamid, S. H. A., Sofian, H., Altarturi, H. H., Azizul, Z. H., and Nasuha, N. (2022). Fault tolerance in big data storage and processing systems: A review on challenges and solutions. *Ain Shams Engineering Journal*, 13(2):101538.

Siddiqa, A., Karim, A., and Gani, A. (2017). Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18:1040–1070.

Wang, Y., Kapritsos, M., Schmidt, L., Alvisi, L., and Dahlin, M. (2014). Exalt: Empowering researchers to evaluate large-scale storage systems. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 129–141.

Ziwich, R. P., Duarte, E., and Albini, L. C. P. (2005). Distributed integrity checking for systems with replicated data. In *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 1, pages 363–369. IEEE.