Processamento Incremental de Grafos em Sistemas Heterogêneos

Pedro Joaquim, Manuel Bravo[1], Rodrigo Rodrigues, Miguel Matos, and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{pedro.joaquim, rodrigo.miragaia.rodrigues, miguel.marques.matos, ler}@tecnico.ulisboa.pt [1]angel.bravo@uclouvain.be

Resumo O processamento incremental de grafos tem hoje aplicações nas mais diversas áreas. Tipicamente, este tipo de processamento é feito em infraestruturas de computação na nuvem que oferecem um conjunto bastante diversificado de serviços. A heterogeneidade dos recursos existentes e os diferentes serviços de aluguer de máquinas, reservadas ou efémeras, acarretam custos distintos consoante as características pretendidas. Neste artigo apresentamos um sistema de processamento incremental de grafos que pretende usar esta heterogeneidade para reduzir os custos de exploração. A solução proposta faz uma atribuição dos nós do grafo às máquinas usando uma política ciente da heterogeneidade.

1 Introdução

O crescimento recente da quantidade de dados armazenados que são representados como grafos (incluindo redes sociais, redes de páginas web e redes biológicas), motivou o aparecimento de várias plataformas dedicadas ao processamento deste tipo de estruturas. Este tipo de sistemas simplifica o desenvolvimento de algoritmos que fazem a análise computacional do grafo, extraindo informação com significado do ponto de vista cientifíco, social ou comercial. Os domínios de interesse, tais como o comércio, a publicidade e os relacionamentos sociais capturam realidades que são bastante dinâmicas e, consequentemente, grafos que modelam estes domínios apresentam uma estrutura que se altera rapidamente e que necessita de ser reanalisada frequentemente. Os modelos de processamento incremental de grafos [1,2,3] foram apresentados para suportar este tipo de dinamismo. Quando o grafo é particionado de forma a ser processado por diferentes máquinas, o trabalho realizado por cada uma está fortemente relacionado com o número de atualizações que cada partição recebe. Grafos que representam modelos de dados reais apresentam desequilíbrios no número de actualizações recebidas por cada nó [4], impondo assim diferentes cargas computacionais em diferentes nós. Infelizmente, sistemas recentes de processamento incremental de grafos [1,2,3] ignoram a heterogeneidade de recursos nestas infraestruturas, perdendo assim oportunidades para reduzir os custos de exploração.

Neste trabalho propomos um sistema de processamento incremental de grafos, Hourglass, que explora oportunidades para reduzir os custos operacionais ao

criar partições de vértices com necessidades computacionais semelhantes e atribuir essas partições a máquinas com os recursos mais adequados. O Hourglass analisa as características da execução do algoritmo de processamento do grafo para identificar e criar essas partições. O sistema explora ainda a utilização de recursos efémeros, um serviço fornecido pela maioria das plataformas, que permite a utilização de recursos que, sendo de baixo custo, não fornecem garantias de elevada disponibilidade.

2 Trabalho Relacionado

Dividimos o trabalho relacionado em duas categorias, nomeadamente: processamento de grafos e utilização de recursos efémeros.

Sistemas de processamento estático de grafos, como o Pregel [5] da Google ou o Giraph [6], são concebidos para correr algoritmos de exploração sobre uma estrutura estática do grafo. Para suportar estruturas dinâmicas, estes sistemas necessitam de executar o algoritmo desde o seu início cada vez que o grafo sofre alterações, o que é extremamente ineficiente [2]. Modelos de processamento incremental de grafos, como o Kineograph [3], o GraphIn [2] e o iGraph [1], permitem que a estrutura do grafo seja modificada durante a computação e atualizam os valores resultantes para refletirem essas modificações. A maior parte deste tipo de sistemas transforma um fluxo de operações de atualização numa série de versões do grafo que são posteriormente processadas pelo algoritmo incremental de exploração. Curiosamente, em diferentes domínios, o fluxo de atualizações apresenta um padrão comum em que um grupo de vértices recebe a maior parte das operações de atualização. Por exemplo, Easley e Kleinberg[4] observaram este comportamento tanto em círculos sociais como em ligações entre páginas web. Nestes domínios, é mais provável que novos vértices sigam outros vértices populares (redes sociais) ou criem ligações a outras páginas mais populares (redes de páginas web). Esta informação já tinha sido utilizada pelo iGraph [1], um sistema de processamento incremental de grafos, que mede o número de vezes que cada vértice recebe atualizações para mover dinamicamente vértices entre partições. Isto procura balancear o trabalho efetuado em cada uma das partições. Esta estratégia é eficaz para equilibrar o trabalho em recursos homogéneos, mas não é claro se será também eficaz em ambientes heterogéneos.

Recursos efémeros, com condições de disponibilidade incertas, são oferecidos por provedores de infraestrutura a um preço bastante reduzido. Assim sendo, a utilização deste tipo de recursos é uma maneira eficaz de reduzir os custos de exploração quando é possível tolerar a indisponibilidade transitória de recursos. Exemplos relevantes de sistemas que utilizam este tipo de recursos são o Pado [7] e o Proteus [8]. O Pado [7] considera um conjunto de computações estáticas e com um fluxo de execução bem conhecido, algo que inviabiliza a sua utilização no processamento de grafos dinâmicos, cujo fluxo de execução é dependente de fatores externos que modificam o grafo. Esse conhecimento é utilizado para fazer uma divisão eficiente das computações entre recursos reservados e efémeros, baseada na sua importância no fluxo de execução. O Proteus [8] pressupõe uma separação

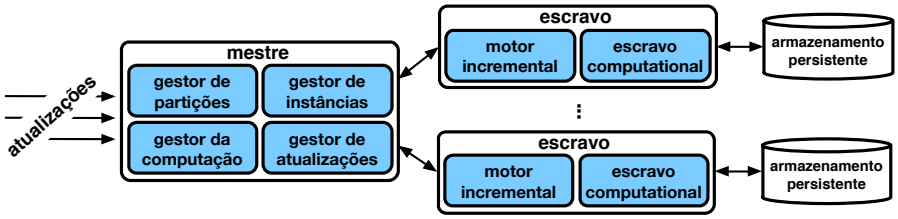

Figura 1: Visão geral da arquitetura do Hourglass.

clara do trabalho a ser processado por cada trabalhador, não existente no nosso contexto devido à necessidade de troca de mensagens entre trabalhadores.

3 Hourglass

Nesta secção descrevemos o Hourglass, um sistema incremental de processamento de grafos. O sistema tira partido da heterogeneidade dos recursos oferecidos pelos fornecedores de infraestrutura na nuvem para reduzir significativamente o custo de exploração com o menor impacto possível na performance do sistema.

3.1 Visão Geral

O Hourglass adota o mesmo modelo computacional de outros sistemas de processamento incremental de grafos como [1,2,3]. Para tirar partido da heterogeneidade, dos recursos existentes nas plataformas de infraestrutura como serviço (em termos de poder computacional, custo e confiabilidade), o Hourglass tem duas características fundamentais: (i) uma política de atribuição de vértices; e (ii) uma técnica de tolerância a faltas adequada às características do sistema alvo. A primeira tem em conta a carga imposta em cada vértice para fazer uma atribuição de recursos apropriada. A segunda permite lidar com as falhas dos recursos efémeros, que são muito prováveis, mitigando o impacto no desempenho do sistema.

O sistema tem uma arquitetura mestre/escravo típica neste tipo de sistemas [5]. Os principais componentes – ilustrados na Figura 1 – são:

- Um *nó mestre*, que gere o sistema e é o ponto de entrada para as operações de atualização para o grafo. Este é composto pelos seguintes subcomponentes:
  - Um *gestor de partições*, que divide o grafo em partições. Estas são classificadas e atribuídas aos recursos apropriados de acordo com a política de atribuição de vértices (§3.2).
  - Um *gestor de instâncias*, que funciona como uma camada de ligação entre o Hourglass e a plataforma de infraestrutura como serviço. Este é responsável por adquirir e libertar recursos. Existem recursos efémeros

e reservados. Para o primeiro tipo de recursos não existem garantias quanto à disponibilidade dos mesmos nem ao preço a que são adquiridos (normalmente muito mais baixo que os recursos reservados). O segundo tipo de recursos tem um preço fixo e existem garantias quanto à sua disponibilidade.
    - Um *gestor de atualizações*, responsável pela distribuição deste tipo de operações para as máquinas escravas que têm a partição respetiva. Este componente é também responsável por determinar quando é necessário começar o processamento de um novo conjunto de atualizações.
    - Um *gestor da computação*, que coordena a execução da computação incremental, define o começo e o final de cada iteração, verifica condições de paragem e efetua operações de agregação [5].
- Um conjunto de *nós escravos*, cada um responsável por manter em memória uma partição do grafo. Os escravos são compostos por dois subcomponentes:
    - Um *motor incremental* que é responsável por atualizar a estrutura do grafo, tendo em conta as operações de atualização recebidas, e identificar os vértices que foram modificados e cujos valores ficaram incoerentes para iniciar a computação incremental.
    - Um *escravo computacional* que é responsável pela coordenação local da computação em cada iteração.
- Um mecanismo externo de armazenamento persistente – acessível por qualquer nó escravo – fundamental para concretizar a tolerância a falhas.

3.2 Política de Atribuição de Vértices

O objetivo da política de atribuição é distribuir os vértices do grafo pelas máquinas disponíveis, tendo em conta os padrões de atualização específicos de cada aplicação [1]. Esta estratégia permite-nos criar partições com diferentes necessidades computacionais e atribuir os recursos de acordo com as suas características. Classificamos os vértices em duas categorias: *quentes* e *frios*. Vértices quentes são vértices que estão constantemente a receber operações de atualização e/ou participam frequentemente na computação incremental. Vértices frios são aqueles que são raramente modificados e executados na computação incremental. A política atual consiste em distinguir os vértices quentes dos frios recorrendo a um limiar de temperatura, definido pelo utilizador, e criar partições de vértices quentes e outras de vértices frios. As temperaturas dos vértices são atribuídas em função do número de vezes que cada vértice recebeu uma atualização ou foi executado durante a computação incremental. As partições frias são atribuídas a máquinas com menos recursos computacionais (mais baratas) e as partições quentes a máquinas com maior poder computacional (mais caras). Apesar de simples, esta estratégia mostra bons resultados, como mostramos na Secção 4.

Esta separação entre vértices frios e quentes permite ao Hourglass explorar a utilização de recursos efémeros. O sistema faz a atribuição das partições frias a máquinas efémeras uma vez que, desta forma, o impacto das falhas destas máquina é menor, pois afeta vértices pouco utilizados. Nalguns casos, a máquina

efémera pode falhar e recuperar sem que os seus vértices sejam usados, fazendo com que a falha não afete o desempenho do sistema.

Neste tipo de sistemas o volume de informação que tem de ser transmitido entre máquinas no decorrer do processamento é um fator predominante no seu desempenho. Nestes modelos computacionais, vértices vizinhos trocam mensagens entre si durante a computação. Mensagens entre vértices que estejam na mesma partição é feita localmente e como tal é mais eficiente. Estratégias para reduzir a quantidade de informação trocada procuram colocar vértices vizinhos nas mesmas partições ao mesmo tempo que são criadas partições balanceadas (frequentemente formulado como *k-balanced partitioning problem*). Trabalhos anteriores como [9,10] focam-se na criação deste tipo de partições em ambientes dinâmicos como aqueles onde o nosso trabalho se insere. Consideramos estas estratégias como algo complementar ao nosso trabalho. Como trabalho futuro pretendemos procurar formas de aliar estas estratégias ao nosso método de particionamento, nomeadamente aplica-las às partições quentes, onde acontece a maior parte da comunicação, de forma a melhorar o desempenho do sistema.

3.3 Tolerância a Falhas

Sistemas que utilizem recursos efémeros tem de lidar eficazmente com a falha frequente de máquinas. Tendo isto em consideração, o nosso objetivo é lidar com falhas deste tipo de recursos sem adicionar um custo computacional ou de memória adicional muito significativo em regime estável, quando as falhas não ocorrem. Para conseguir isto, nós propomos uma mecanismo de tolerância a falhas que combina salvaguarda de estado e replicação das operações de atualização. Os nós escravos guardam periodicamente o estado atual das suas partições recorrendo ao serviço de armazenamento persistente. De forma a evitar que as operações de atualização, que são recebidas pelo mestre e direcionadas para os escravos correspondentes, sejam perdidas, o Hourglass replica estes pedidos. Estas réplicas podem ser descartadas quando uma salvaguarda de estado acontece. A replicação das operações de atualização aliada à salvaguarda de estado é uma forma eficaz de evitar a replicação de partições [3] o que se traduz em maiores custos de exploração. Devido às suas características, apenas recursos reservados recebem réplicas das operações de atualização. Quando ocorrem falhas, as máquinas que substituem os nós falhados apenas têm de ler a última salvaguarda e pedir as operações de atualização em falta para restaurar o estado anterior.

4 Avaliação

Devido a questões de espaço limitamos a avaliação apresentada a um dos objetivos principais do sistema: pode o Hourglass reduzir os custos de exploração sem prejudicar significativamente o desempenho do sistema. Uma avaliação mais completa pode ser consultada em [11].

4.1 Configuração do Ambiente de Testes

Avaliamos o nosso protótipo do Hourglass no ambiente da Amazon[1] onde exploramos a utilização de *spot-instances*, recursos efémeros disponibilizados nesta plataforma. Nos testes realizados foram utilizados dois tipos de máquinas que denominamos: (i) máquinas M correspondentes a EC2 c4.xlarge com quatro vCPUs e memórias de 7.5GB; e (ii) máquinas S correspondentes a EC2 c4.large com dois vCPUs e memórias de 3.75GB. Foram utilizados volumes *EBS* como mecanismo externo de armazenamento persistente. Todas as máquinas corriam Ubuntu Server 14.04 LTS. Todas as configurações de testes seguintes utilizavam uma máquina M para o nó mestre que será omitida por questões de simplicidade.

Nos testes foram utilizados dois grafos. Um grafo real que representa as relações de amizade entre utilizadores da rede social *orkut* [12] com $3,072,441$ vértices e $117,185,083$ ligações. Foi também gerado um grafo sintético, utilizando o gerador R-MAT [13] com os mesmos parâmetros utilizados pelo benchmark graph500[2] gerando um grafo com $5,000,000$ vértices e $450,000,000$ ligações.

Foram utilizadas duas aplicações. A primeira destas aplicações calcula caminhos mais curtos com fonte única (do inglês single source shortest paths (SSSP)). Este algoritmo é frequentemente utilizado para calcular a distância de todos os vértices a um conjunto de vértices referência. Isto permite depois fazer aproximações entre as distâncias de todos os pares de vértices. A segunda aplicação foi o TunkRank [14], um algoritmo semelhante ao PageRank para calcular a influência de utilizadores em redes sociais.

No processo de geração de atualizações foi considerada a informação fornecida pelo Twitter[3], onde reportam que cerca de 40% dos utilizadores nunca fazem tweets. Desta forma, os quinze milhões de operações de atualização foram gerados aleatoriamente com destino a um conjunto de 60% do número total de vértices, também escolhidos aleatoriamente.

4.2 Comparação de Políticas de Atribuição de Vértices

Para aferir a eficácia do Hourglass, comparámos o desempenho de diferentes políticas de atribuição de vértices em dois clusters diferentes. Ambos os clusters com o mesmo número de máquinas, mas num deles apenas são utilizadas máquinas do tipo M (cluster C1) e no outro uma utilização proporcional de 60% de máquinas do tipo M e o restante do tipo S (cluster C2). O nosso objetivo é mostrar que no cluster C2 (mais barato) o Hourglass consegue apresentar resultados de desempenho comparáveis ao melhor caso possível, que acontece quando existe uma distribuição uniforme do trabalho por todas as partições no cluster C1 ($Uniform_{C1}$), equivalente à estratégia utilizada pelo iGraph [1]. Pretendemos também perceber o impacto da utilização da estratégia seguida pelo iGraph num cluster heterogéneo ($Uniform_{C2}$). Avaliamos ainda a distribuição uniforme

[1] https://aws.amazon.com/
[2] http://graph500.org/?page_id=12
[3] http://www.statisticbrain.com/twitter-statistics/

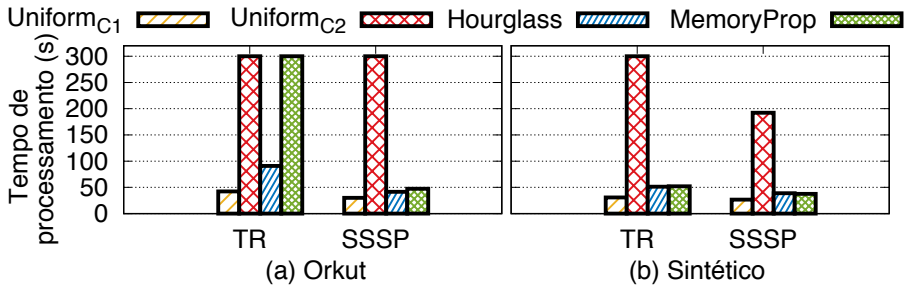

Figura 2: Tempo de execução com diferentes políticas de atribuição de vértices em diferentes grafos, aplicações e máquinas

do trabalho no cluster C2 aliada à criação de partições com um tamanho proporcional à memória disponível em cada máquina (*MemoryProp*).

A Figura 2 mostra os resultados obtidos. Os casos marcados com 300 segundos de execução não terminavam a computação, tendo acabado por abortar por falta de memória. No grafo do orkut foram utilizadas um total de quatro máquinas (C1 = 4M e C2 = 2M e 2S) e no grafo sintético um total de 12 máquinas (C1 = 12 M e C2 = 7M e 5S). Nesta configuração específica, em que consideramos a existência de um conjunto de vértices quentes de 60%, com a utilização de *spot-instances* para as máquinas do tipo S a configuração C2 representa uma redução do custo/hora entre 35% e 45%.

Como seria de esperar a atribuição uniforme de temperaturas no cluster C1 ($Uniform_{C1}$) oferece sempre o melhor desempenho e no cluster C2 sempre o pior desempenho ($Uniform_{C2}$). Isto acontece porque não é feito um balanceamento do trabalho adequado às características de cada máquina. O Hourglass no cluster C2 (*Hourglass*), apesar de utilizar partições com o mesmo tamanho para todas as máquinas, consegue apresentar tempos de execução comparáveis aos tempos obtidos no cluster C1 pela estratégia do iGraph. A distribuição uniforme de temperaturas em partições proporcionais à memória disponível em cada máquina (*MemoryProp*) consegue também apresentar bons resultados para a maioria dos casos. Esta estratégia tem no entanto algumas desvantagens face à estratégia do Hourglass. Em primeiro lugar, a atribuição proporcional pode em algumas situações sobrecarregar as máquinas que recebem partições maiores e levar o sistema a falhar, o que acontece no grafo do orkut. Em segundo lugar, a atribuição de vértices pode continuar a colocar vértices quentes em qualquer uma das máquinas, criando um impacto maior no desempenho do sistema em caso de falha deste tipo de recursos.

5 Conclusão

Neste artigo apresentamos o Hourglass, um sistema de processamento incremental de grafos que explora a heterogeneidade dos recursos existentes no

ambiente de exploração e a utilização de recursos efémeros para reduzir os custos do sistema. O sistema faz uma separação entre vértices quentes e frios para fazer uma atribuição mais eficiente de recursos às partições criadas. A avaliação do sistema mostra que este é capaz de reduzir os custos de exploração com base na proporção de vértices quentes e frios sem prejudicar o desempenho do sistema.

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

Referências

1. Ju, W., Li, J., Yu, W., Zhang, R.: igraph: An incremental data processing system for dynamic graph. Front. Comput. Sci. **10**(3) (June 2016) 462–476
2. Sengupta, D., Sundaram, N., Zhu, X., Willke, T., Young, J., Wolf, M., Schwan, K.: Graphin: An online high performance incremental graph processing framework. In Dutot, P.F., Trystram, D., eds.: Proceedings of the 22nd International Conference on Parallel and Distributed Computing (Euro-Par), Grenoble, Springer International Publishing (2016) 319–333
3. Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., Chen, E.: Kineograph: Taking the pulse of a fast-changing and connected world. In: Proceedings of the EuroSys. (2012)
4. Easley, D., Kleinberg, J.: Networks , Crowds , and Markets : Reasoning about a Highly Connected World. Science **81** (2010) 744
5. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A system for large-scale graph processing. In: Proceedings of the SIGMOD. (2010)
6. The Apache Software Foundation: Apache Giraph. http://giraph.apache.org/
7. Yang, Y., Kim, G.W., Song, W.W., Lee, Y., Chung, A., Qian, Z., Cho, B., Chun, B.G.: Pado: A data processing engine for harnessing transient resources in datacenters. In: Proceedings of the EuroSys. (2017)
8. Harlap, A., Tumanov, A., Chung, A., Ganger, G.R., Gibbons, P.B.: Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets. In: Proceedings of the EuroSys. (2017)
9. Huang, J., Abadi, D.J.: Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs. Proc. VLDB Endow. **9**(7) (March 2016) 540–551
10. Mayer, C., Tariq, M.A., Li, C., Rothermel, K.: Graph: Heterogeneity-aware graph computation with adaptive partitioning. In: Proceedings of the 36th International Conference on Distributed Computing Systems (ICDCS). (2016) 118–128
11. Joaquim, P.: Incremental graph processing on heterogeneous infrastructures. Master's thesis, Instituto Superior Técnico, Universiade de Lisboa (October 2017)
12. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. CoRR **abs/1205.6233** (2012)
13. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: A recursive model for graph mining. In Berry, M.W., Dayal, U., Kamath, C., Skillicorn, D.B., eds.: Proceedings of the SIAM International Conference on Data Mining. (2004) 442–446
14. Tunkelang, D.: A twitter analog to pagerank. Retrieved from http://thenoisychannel.com/2009/01/13/a-twitter-analog-topagerank (2009)