Desafios algorítmicos no processamento de grandes volumes de dados

Marcus Ritt[1], **Luciana S. Buriol**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 –Porto Alegre – RS – Brazil

Resumo. *O volume de dados produzidos e disponíveis em organizações, empresas e em domínio público está crescendo exponencialmente. Entretanto, a capacidade de processar esses dados não tem crescido da mesma forma. O IDC estimou que em 2007 a quantidade de informação produzida mundialmente (*255 *EB) ultrapassou, pela primeira vez, o espaço disponível para seu armazenamento (*246 *EB). Como esses dados não somente precisam ser armazenados, mas também processados e analisados de forma eficiente, o interesse em algoritmos para processamento de dados massivos cresceu significadamente nos últimos anos.*

Este artigo discute os modelos de algoritmos propostos para o processamento de grandes volumes de dados, apresenta os desafios nesta área, identifica aplicações no cenário Brasileiro e discute soluções possíveis para que tais desafios sejam transpostos.

1. Introdução

Numa era digital, informações antigamente armazenadas em diversas mídias se tornam gradualmente digitais. Alguns exemplos são livros eletrônicos, registros públicos, ou dados de pacientes em hospitais. Em paralelo, novas tecnologias produzem um volume cada vez maior de informações. Redes de sensores permitem medir um grande número de parâmetros ambientais, por exemplo para o controle de processos de produção. A Internet armazena informações sobre milhares de usuários, além de manter repositórios de vídeo, áudio e outros tipos de dados. Satélites geram imagens da terra no volume de petabytes [Meyer et al. 2003]. Gera-se um volume de dados produzidos e disponíveis em organizações, empresas e em domínio público que cresce exponencialmente. O IDC estimou que em 2007 o volume de informação produzida mundialmente (255 EB) ultrapassou, pela primeira vez, o espaço disponível para seu armazenamento (246 EB) [IDC 2007].

A disponibilidade de tais dados não implica que os mesmos são eficientemente utilizados. O processamento de dados massivos é um desafio para a ciência da computação, e está diretamente relacionado ao desafio *Gestão da Informação em grandes volumes de dados multimídia distribuídos* da lista dos *Grandes Desafios da Pesquisa em Computação no Brasil nos Próximos 10 Anos*, anunciados em 2006 pela Sociedade Brasileira de Computação.

A maioria dos algoritmos conhecidos não podem ser aplicados a quantidades massivas de dados por razões diversas, tais como capacidade de armazenamento ou tempo de

processamento. Com o objetivo de transpor tais barreiras, modelos algorítmicos para dados massivos foram recentemente propostos [Prokop 1999, Aggarwal 2007, Vitter 2007].

Os algoritmos tradicionais foram projetados em um modelo no qual a memória RAM é a única memória com acesso uniforme. Um acesso diferente aos dados de entrada frequentemente os torna inviáveis na prática. Isso motiva o estudo de novos modelos computacionais que consideram essa realidade, bem como novas abordagens algorítmicas para tornar o processamento de dados massivos viável.

A seguir alguns cenários nacionais são apresentados, demonstrando a necessidade de pesquisa nesses desafios no Brasil.

- O conselho nacional de trânsito (CONTRAN) decidiu em novembro de 2006 introduzir no Brasil um sistema de controle de veículos – Sistema de Identificação Automática de Veículos (SINIAV) – até o final do ano 2011 [Conselho nacional de trânsito 2006]. O sistema prevê a identificação dos veículos através de chips de radiofrequência (RFID), com leitores correspondentes instalados em todo país. Aplicações possíveis desse sistema são o controle do registro de veículos, o rastreamento de veículos roubados e até o controle de tráfego.
  Atualmente existem 43 milhões de veículos no Brasil. Se supormos uma média de somente 10 eventos por dia por veículo, temos que processar diariamente um volume de dados de aproximadamente 1.6 GB (supondo uma informação de 32 bits por evento), de forma distribuída sobre toda área do Brasil.
- A biologia e a biotecnologia visam entender melhor organismos e plantas. Com esse conhecimento busca-se melhorar produtos agronômicos, reduzir danos ambientais, ou desenvolver novos medicamentos, entre outros objetivos. Entre os problemas que esses estudos buscam resolver estão a união de fragmentos obtidos pelo sequenciamento, o armazenamento, a compressão e a busca em bancos de dados biológicos, a predição da estrutura de proteínas, o estudo de vias biológicas e a classificação genética de características. Uma das técnicas que mais avançou a capacidade de entender sistemas biológicos são Microarrays [Lander 1999], que permitem analisar a expressão genética de milhares de genes em um único experimento.
  A dimensão dos dados biológicos facilmente ultrapassa o tamanho da memória principal da maioria dos computadores atuais. Por exemplo, o `GenBank`, um banco de dados de sequências genéticas, em 2008 dispunha de quase 99 milhões de sequências genéticas [NCBI], que correspondem a mais de 99 bilhões de pares de bases de DNA. A quantidade de dados produzidos pela análise de um experimento com Microarrays pode chegar a mesma quantidade dos dados originais.
- Em telecomunicações, informações sobre chamadas telefônicas, pontos de transferência de tráfego de Internet, páginas web visitadas, bem como outros tipos de transações que possuem origem-destino, são armazenadas por períodos de até dois anos, dependendo das leis de cada país. A finalidade deste armazenamento é para utilização comercial, podendo também ser governamental (dependendo do país). Estes dados podem ser estudados para melhorar os serviços prestados pelas companhias telefônicas. Exemplo de análises incluem estimativas da quantidade

de dados passantes em determinadas conexões com destino em um intervalo de endereços IP, ou quantos endereços IP distintos usaram uma certa conexão durante o dia, quais os k maiores fluxos passantes da rede a cada instante, entre outras questões, devem ser respondidas diariamente. Com informações como estas é possível detectar anomalias na rede, bem como determinar que tipo de serviços podem ser oferecidos aos clientes. A análise precisa de tais dados se torna impraticável considerando algoritmos tradicionais, visto que muitas estimativas têm que ser fornecidas em um curto período de tempo.
Uma pesquisa sobre o uso da Internet, patrocinada pela agência de publicidade F/Nazca Saatchi & Saatchi[1], divulgou que em agosto de 2008 o número de internautas no Brasil era de 64,5 milhões. Um número ainda maior de brasileiros possui telefones. Processar informações sobre o tráfego Internet diário desses usuários certamente demanda processamento massivo.

O número de aplicações com demanda por processamento massivo é grande e, em muitas situações, os dados não são devidamente explorados pela falta de algoritmos adequados. Apesar do crescente interesse da comunidade acadêmica por este tópico nos últimos anos, ainda não é evidente, para cada tipo de problema, qual é a técnica mais adequada. Um desafio ainda maior é aprimorar as técnicas já existentes para que novos problemas sejam resolvidos. Observando um aspecto prático, apesar da disponibilidade de tais técnicas e de algumas soluções já propostas na literatura, a transferência desses conhecimentos para aplicações práticas em empresas é outro desafio.

Com o objetivo de abordar esta questão, além de citar aplicações no cenário nacional, este artigo apresenta e discute os modelos de algoritmos propostos para o processamento de grandes volumes de dados, identifica desafios nesta área, bem como discute soluções possíveis para que tais desafios sejam transpostos. O artigo encontra-se organizado da seguinte forma. Na seção 2 são analisados os modelos computacionais atuais, e apresentados os modelos propostos para projetar algoritmos eficientes para dados massivos. A seção 3 discute os desafios algorítmicos vinculados a esses novos modelos. Ainda nesta seção, apresentamos o que consideramos os principais desafios nesta área. O artigo é concluído com algumas considerações finais e uma bibliografia atualizada no assunto.

2. Modelos algorítmicos para processamento de dados massivos

Modelos tradicionais de computação consistem em um processador com acesso à memória. Os modelos atualmente usados para análise de algoritmos assumem um tempo fixo para a execução de cada operação ou logarítmico no tamanho dos dados, assim como acesso livre à memória (com tempo de acesso uniforme).

Esses modelos não refletem a realidade em pelo menos dois pontos. Uma arquitetura atual consiste em diversos níveis de memória como, por exemplo, os caches primário e secundário. O desempenho atual de um algoritmo depende significadamente do aproveitamento ótimo da hierarquia de memórias, já que o tempo de acesso entre elas pode ser até um fator 1000 diferente (veja Tabela 1). Portanto, no projeto de algoritmos eficientes, o melhor uso das memórias tem que ser considerado.

[1] www.fnazca.com.br

Tabela 1. Principais características de diferentes níveis da hierarquia de memórias [Hennessy and Patterson 2002].

Nível	Tamanho	Tempo de acesso [ns]	Taxa de transmissão [MB/s]
Registro	$\leq$ 1 KB	0.25-0.5	20000-100000
Cache	$\leq$ 16 MB	0.5-25	5000-10000
Memória principal	$\leq$ 16 GB	80-250	1000-5000
Disco	$\geq$ 100 GB	5000000	20-150

Outro limite dos modelos atuais é o processamento de dados que ultrapassam o tamanho da memória principal. Estima-se que a diferença entre o tempo de acesso à memória é até um milhão de vezes mais rápido que o acesso a disco [Vitter 2007]. Como essa diferença é muito grande, o tempo de execução na prática é dominado pelo número de acessos (entrada/saída) aos blocos do disco. O tipo de processamento possível nesse caso também depende da forma em que os dados estão organizados. Se os dados estão armazenados em disco, a aplicação pode aproveitar do acesso livre e estático aos blocos. Em outros cenários os dados estão disponíveis somente em forma transiente. Exemplos importantes são páginas web obtidas pelos programas que percorrem sistematicamente todas páginas (*crawlers*), ou dados fornecidos de grandes números de sensores em redes de sensores. Em ambos os casos, o volume de dados é muito grande para que estes sejam armazenados e processados posteriormente, e um algoritmo tem que processar de forma *online*, i.e. elemento por elemento na sequência apresentada, e com memória local muito menor que o volume total de dados. Algoritmos desse tipo também são chamados *algoritmos de fluxo de dados* (*data stream algorithms*). Dependendo da aplicação, múltiplas leituras dos dados são permitidas.

Os limites mencionados dos modelos tradicionais necessitam do projeto de novos modelos computacionais que capturem de forma adequada as características de algoritmos que processam grandes volumes de dados. Como os limites desses modelos são de natureza diferente, não existe um único modelo que reflete todas as necessidades para análise dos algoritmos com uma complexidade aceitável na prática. Portanto, atualmente existem três modelos diferentes difundidos, que refletem características diferentes da hierarquia de memória, e que são aplicados a diferentes cenários: algoritmos universalmente cache-eficientes, algoritmos em memória externa e algoritmos de fluxo de dados. As próximas seções detalham esses novos modelos.

2.1. Algoritmos universalmente cache-eficientes

Para utilizar o *cache* de forma mais eficiente, o projetista de algoritmos enfrenta várias dificuldades, pois a hierarquia de memórias nos computadores atuais é complexa. Existem vários níveis de *cache*, como os registros, ou os *caches* primário e secundário. Os *caches* têm tamanhos diferentes que dependem da máquina. Eles são organizados em blocos (ou linhas) cujo tamanho tem que ser considerado no projeto de algoritmos, visto que eles têm organizações diferentes. Por exemplo, a associatividade do *cache*, que determina quais blocos da memória principal podem permanecer ao mesmo tempo no *cache*. Essa complexidade, bem como a variação entre arquiteturas, complicam o modelo computacional e dificultam o projeto e a análise de algoritmos que levam todos esses parâmetros

em consideração. Pela dificuldade de projetar algoritmos cache-eficientes, são usados por alguns algoritmos como otimizadores automáticos, com o objetivo de obter uma melhor implementação. Um exemplo é a ferramenta ATLAS (*Automatically Tuned Linear Algebra Software*) [Whaley et al. 2001], que otimiza algoritmos fundamentais da álgebra linear, tais como a multiplicação de matrizes.

O modelo mais comum para a análise e projeto de algoritmos cache-eficientes é o modelo (Z, L) de um *cache* ideal, que foi proposto por [Prokop 1999]. O modelo consiste em um processador que se comunica com a memória principal através de um *cache* (Figura 1). O *cache* consiste em Z palavras organizadas em linhas de *cache* com L palavras. Uma linha de *cache* é lida ou escrita integralmente na memória principal. O modelo assume que o *cache* é totalmente associativo, isto é, uma linha pode ser armazenada em qualquer posição do *cache*. Ainda, a estratégia de substituição é ideal, ou seja, caso for necessário, a linha que seria referenciada mais tarde será removida do *cache*[2].

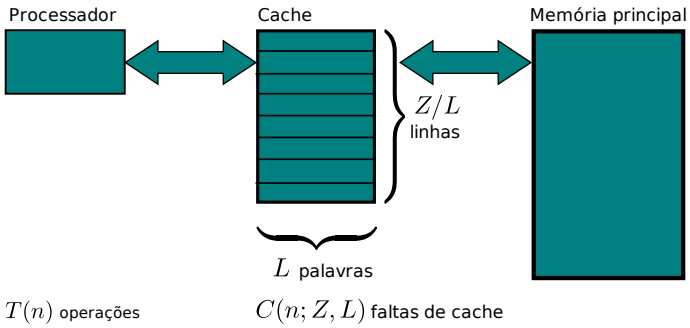

Figura 1. Modelo ideal de cache. O processador acessa a memória principal através de um cache. O cache armazena Z palavras, organizadas em linhas (ou blocos) de tamanho L. O cache é totalmente associativo e a estratégia de substituição é ideal.

Uma referência do processador para um endereço cuja linha correspondente não pertence ao *cache* gera uma *falta de cache* (*cache miss*). A análise comum de algoritmos conta as operações $T(n)$ em função do tamanho da entrada n. No modelo do *cache* ideal, uma segunda medida de eficiência é o número de faltas de *cache* $C(n; Z, L)$ que depende ainda dos parâmetros do *cache*. O projeto de algoritmos universalmente cache-eficientes tem como objetivo minimizar essa segunda medida, usando o mesmo número de operações que algoritmos conhecidos, e ainda sem depender explicitamente dos parâmetros do *cache*.

Em [Frigo et al. 1999] mostra-se que é possível atingir esse objetivo. Eles apresentam, além de outros, um algoritmo de multiplicação de matrizes, que gera assintoticamente o menor número de faltas de *cache* possíveis, com o mesmo número de operações (complexidade de tempo) de algoritmos tradicionais. Além disso, esses resultados podem ser generalizados para um número arbitrário de níveis de *caches*, um fato que os torna ainda mais úteis na prática.

A Tabela 2 apresenta alguns dos principais algoritmos propostos no modelo do

[2]Observa-se que a estratégia ótima não pode ser implementada. A complexidade de cache de outras estratégias de substituição usadas na prática é assintoticamente equivalente.

cache ideal.

Tabela 2. Alguns dos principais resultados obtidos no modelo universalmente cache-eficiente.

Problema	Complexidade	
	$T(n)$	$C(n; Z, L)$
Transposição de matrizes $m \times n$ [Frigo et al. 1999]	$O(mn)$	$O(1 + mn/L)$
Transformação Fourier [Frigo et al. 1999]	$O(n \log n)$	$O(1 + n/L(1 + \log_Z n))$
Ordenação [Frigo et al. 1999]	$O(n \log n)$	$O(1 + n/L(1 + \log_Z n))$
Multiplicação de matrizes $n \times n$ [Frigo et al. 1999]	$O(n^3)$	$O(n + n^2/L + n^3/L\sqrt{Z})$
Subsequência comum mais longa [Chowdhury and Ramachandran 2006a]	$O(n^2)$	$O(n^2/ZL)$
Problem do gap [Chowdhury and Ramachandran 2006a]	$O(n^3)$	$O(n^3/L\sqrt{Z})$
Eliminação Gaussiana generalizada [Chowdhury and Ramachandran 2006a]	$O(n^3)$	$O(n^3/L\sqrt{Z})$

2.2. Algoritmos de fluxo de dados

Os modelos de algoritmos tradicionais consideram que os dados estão disponíveis de forma persistente. Mas muitas aplicações recebem ou produzem um fluxo contínuo de dados que, em muitos casos, não pode ser armazenado e deve ser processado na mesma velocidade em que é observado. Exemplos são as redes de sensores que geram dados continuamente, a análise do fluxo de chamadas telefônicas, a análise de dados de *clickstreams* (visitas a páginas web), o monitoramento de operações financeiras, o monitoramento médico, entre outros.

No modelo original de fluxo de dados os dados são recebidos continuamente, em uma ordem arbitrária. A característica principal destes algoritmos é que devem processar os dados usando recursos de memória limitados, podendo ou não ser dependentes da estrutura dos dados. O espaço de memória disponível para o processamento geralmente é sublinear no tamanho de dados. Neste modelo o algoritmo perde o acesso livre aos dados. Estes algoritmos são conhecidos na literatura como algoritmos de fluxo de dados [Aggarwal 2007, Muthukrishnan 2005], ou algoritmos sublineares. Aplicações típicas são o monitoramento de dados, a análise de fluxos de comunicação ou de grafos massivos.

Os modelos de fluxo de dados envolve três variáveis:

- Um fluxo $S = s_1, s_2, s_3, ...$ de dados, que não necessariamente seja finito.
- Um espaço de memória M que pode ser sublinear ou dependente da estrutura dos dados, representado o total de memória que o algoritmo pode usar.
- O número de leituras (P que podem ser efetuadas sobre os dados.

A Figura 2.2 apresenta o modelo de fluxo de dados. Um algoritmo implementado sob o modelo de fluxo de dados S lê os itens de dados, os processa, e adiciona ou atualiza a informação na memória M. A qualquer momento uma consulta à memória pode ser realizada, que retorna a estimativa sobre os dados naquela unidade de tempo.

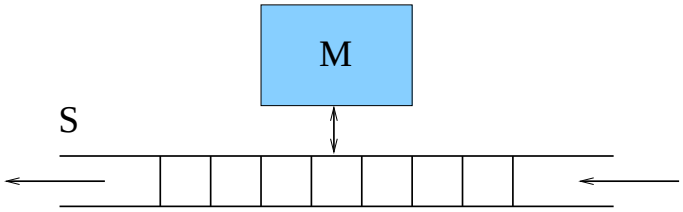

Figura 2. O modelo de fluxo de dados possui uma memória M que armazena informações processadas a partir da leitura de S. S é um fluxo de dados composto por n itens (n é infinito se for um fluxo contínuo). M possui tamanho sublinear em relação a S.

De forma geral, o objetivo do projeto desses algoritmos é estimar uma dada propriedade sobre os dados. O desempenho dos algoritmos é medido pelos seguintes fatores:

- o número de leituras P dos dados (para valores de $P > 1$, em geral tem-se melhores resultados),
- o tempo de processamento de cada elemento $s_i \in S$,
- o tempo de resposta do algoritmo, considerando a informação disponível em memória,
- e a quantidade de memória utilizada.

A Tabela 3 apresenta a complexidade de espaço (memória) e de tempo necessários pelo melhor algoritmo que resolve cada um dos problemas listados na primeira coluna da tabela. Em todos os casos os algoritmos consideram uma única leitura aos dados ($P = 1$). Os algoritmos utilizam memória M para fornecer com probabilidade $1 - \gamma$ uma aproximação $1 \pm \epsilon$ da solução ótima do problema. O tempo de consulta em todos os casos é linear no tamanho da memória. Nas fórmulas apresentadas, f representa o valor no qual se quer estimar a frequência de itens; C_G representa o coeficiente de aglomeração (*clustering coefficient*) (probabilidade de existência de um arco entre dois nós vizinhos de um dado nó) do grafo G; $|S|$ representa a quantidade total de itens do fluxo de dados; e w são quantidades de itens que ocorrem em sequência (a estimativa é fornecida sobre janelas com w itens).

Tabela 3. Alguns dos principais resultados obtidos através do uso do modelo de fluxo de dados. Os algoritmos resolvem os problemas indicados, usando espaço de memória M e tempo T para processamento de cada item do fluxo de dados. Os algoritmos utilizam memória M para fornecer com probabilidade $1-\gamma$ uma aproximação $1 \pm \epsilon$ do resultado, com tempo de consulta ao resultado $O(M)$. Todos leem os dados apenas uma vez ($P = 1$).

Problema	Memória M	Tempo T
Norma de Hamming [Cormode et al. 2003]	$O(\frac{1}{\epsilon^2} \cdot \log(\frac{1}{\gamma}))$	O(1)
Frequência de itens [Manku and Motwani 2002]	$O(\frac{2}{\epsilon} \cdot \log(f^{-1} \cdot \gamma^{-1}))$	O(1)
Coeficiente de aglomeração [Buriol et al. 2007]	$O(\log \frac{1}{\delta} \cdot \frac{1}{\epsilon^2 \cdot C_G})$	O(1)
Similaridade de Jacard [Datar and Muthukrishnan 2002]	$O(\log \lvert S \rvert \cdot \log w)$	$O(\log \log w)$

Este modelo difere do modelo clássico de algoritmos *online* por possuir limite do uso dos recursos, como a memória utilizada, fazendo com que elementos de S não sejam explicitamente armazenados na memória em uso pelo algoritmo.

Os primeiros modelos propostos consideram que os itens de S são recebidos em uma ordem arbitrária. Mas é comum o projeto de algoritmos de fluxo de dados considerando que os mesmos ocorram de forma estruturada. A razão é que ter os dados estruturados possibilita o projeto de algoritmos mais acurados, que fornecem melhores resultados usando os mesmos recursos computacionais.

Por exemplo, considere o algoritmo de fluxo de dados proposto em [Buriol et al. 2006] que estima o número de triângulos de um grafo. Este algoritmo tem três passos ($P = 3$), ou seja, três leituras dos dados. O primeiro passo objetiva contar o número de arcos do grafo; o segundo passo escolhe s tuplas de nós u, v, w, sendo que um par corresponde aos nós de um arco $(u, v) \in G$; e o terceiro passo verifica a existência dos arcos (u, w) e (v, w) (que completam o triângulo). Em qualquer grafo não direcionado, o valor esperado do número de triângulos é $E[t] = 3|T_3|/|T_1| + 2|T_2| + 3|T_3|$, sendo T_i o conjunto de tuplas com i arcos entre os nós da tupla. O denominador corresponde à soma de todas as tuplas, enquanto que T_3 é o número de tuplas que formam um triângulo. Para este algoritmo, considerou-se que o grafo encontra-se armazenado como uma lista de adjacência. Visto que o grafo é não direcionado, cada arco (u, v) é armazenado duas vezes: uma na lista de adjacência de u, e outra na lista de adjacência de v. Com isso, um mesmo triângulo pode ser contado até três vezes (e por isso é multiplicado por três), uma para cada combinação de arco e nó. Considerando este algoritmo, mostrou-se que para um número de execuções paralelas $s \geq 3/\epsilon \cdot |T_1| + 2|T_2| + 3|T_3|/3|T_3| \cdot ln2/\gamma$ é possível estimar com probabilidade mínima de $(1 - \gamma)$ o número de triângulos de um grafo, considerando um erro de $\pm\epsilon$.

Os modelos não incluem P no modelo quando os dados são recebidos em fluxo contínuo. Trabalhos recentes consideram o caso em que algoritmos de fluxo de dados também sejam aplicados a dados massivos armazenados em disco e, neste caso, pode-se considerar mais de uma leitura destes dados. Exemplos deste caso são os algoritmos para estimar propriedades de grafos massivos, em aplicações cujo objetivo é calcular propriedades sobre o grafo. Pode-se destacar o algoritmo para o cálculo do número de triângulos em grafos [Z. Bar-Yosseff 2002], sendo tais resultados melhorados (de cúbico para linear) numa versão de algoritmo de amostragem [Buriol et al. 2006]. Recentemente também foram propostos por [Buriol et al. 2007] algoritmos para o cálculo do número de cliques bipartidos de pequena dimensão, assim como para o cálculo do coeficiente de aglomeração de grafos massivos. Estes algoritmos foram propostos para $P = 1$ e $P = 3$, usando dados estruturados e dados não estruturados. Quanto maior o número de passos, e quando estrutura for considerada, melhores são os resultados teóricos e, por consequência, melhores serão os resultados experimentais.

2.3. Algoritmos em memória externa

Frequentemente os dados massivos são armazenados em memória secundária (principalmente discos rígidos), e nem sempre é possível manipular o conjunto das entradas inteiramente na memória principal.

O tempo de acesso à memória secundária é três ordens de magnitude maior, e a taxa de transmissão duas ordens de magnitude menor, em comparação com a memória principal. A unidade de transferência tipicamente é um bloco de dados (de 512, 1024 ou

mais bytes). Diferente de algoritmos de fluxo de dados, algoritmos em memória externa podem aproveitar do acesso livre aos dados. Mesmo assim, a localidade dos acessos é desejável, já que o deslocamento da cabeça de leitura de um disco rígido custa muito tempo.

Além do número de operações do processador, o desempenho de um algoritmo em memória secundária depende do tempo da transferência (leitura ou escrita) dos dados entre o disco e a memória principal. Existem diversos modelos de discos, que consideram características como latência de acesso, e a banda de transferência. Pela sua simplicidade o modelo mais utilizado para a análise de algoritmos de memória secundária é o modelo proposto em [Vitter 2007] que captura as principais características dos algoritmos em memória secundária.

Este modelo, denominado *Parallel Disk Model* (PDM), apresenta as seguintes propriedades (veja figura 3):

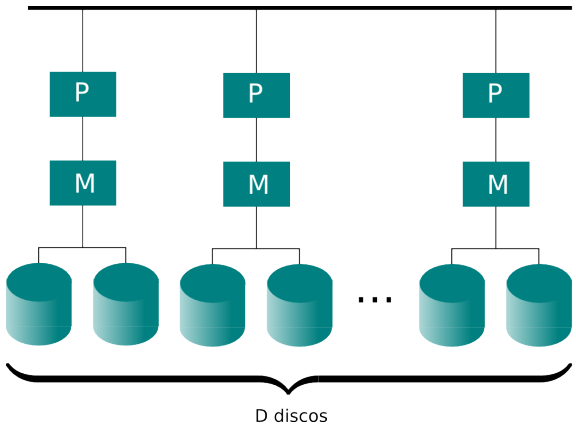

Figura 3. Modelo de discos paralelos. Um número P de processadores, cada um com memória principal de tamanho M, tem acesso a D discos independentes. Cada processador controla cerca D/P discos. No exemplo temos $D = 2P$.

- N = tamanho do problema (em unidades de itens de dados)
- M = tamanho da memória interna (em unidades de itens de dados)
- B = tamanho do bloco de transferência (em unidades de itens de dados)
- D = número de discos independentes
- P = número de CPUs
- Q = número de consultas de entrada
- Z = tamanho da saída (em unidades de itens de dados).

O número de processadores e o número de discos são independentes, e o modelo assume que um processador controla cerca de D/P discos. Os discos operam de forma paralela, tal que D blocos de dados podem ser transferidos em uma única operação. Os parâmetros Q e Z modelam algoritmos que processam consultas. Cada uma das Q consultas pode gerar um número diferente de resultados. Portanto o desempenho do algoritmo depende do tamanho da saída Z.

A duas principais medidas de desempenho de algoritmos de memória externa no modelo PDM são: o número de transferências efetuadas e o espaço usado. Como o tempo

Tabela 4. Resultados de alguns algoritmos determinísticos de memória externa em grafos considerando $D = P = 1$

Problema	Limitante Inferior	Limitante Superior
Scan(N) [Vitter 2007]	$\Omega(\frac{N}{B})$	$\Omega(\frac{N}{B})$
Sort(N) [Y.J.Chiang et al. 1995]	$\Omega(\frac{N}{B} log_{\frac{M}{B}} \frac{N}{B})$	$O(\frac{N}{B})$
Search(N) [Abello and Vitter 1999]	$\Omega(\log_B N)$	$O(\log_B N)$
Componentes conexos [Abello and Vitter 1999]	$\Omega(sort(V))$	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$
Árvore geradora mínima [Dementiev et al. 2004]		$O(sort(E) \cdot \log\log B)$
Caminhos mínimos de única fonte [Arge et al. 2000]		$O(sort(N))$

de transferência geralmente domina o tempo total da execução, o número de operações na memória principal não é considerado nessa análise. Essa hipótese nem sempre é justificada, podendo ser necessária a análise das três medidas.

Desta forma, o projeto de algoritmos em memória secundária visa minimizar o número de acessos à memória externa. Com isso, abre-se mão das facilidades de memória virtual tipicamente disponíveis nos sistemas operacionais, para indicar em que momentos devem ser realizadas as transferências, assim como quais informações devem estar na memória principal e quais devem permanecer na memória secundária.

Uma classificação possível de tais algoritmos compara o tamanho da entrada do algoritmo em relação ao tamanho da memória principal disponível. Através desta observação é possível classificar os algoritmos como internos, se toda informação couber na memória; semi-externos, se algum atributo dos dados pode ser armazenado na memória principal (por exemplo todos os nós de um grafo, mas não os arcos); e externo, se nenhum atributo dos dados couber completamente em memória.

A Tabela 4 apresenta os melhores limitantes inferiores e superiores conhecidos para algoritmos básicos de leitura dos dados, ordenação, busca de um elemento, componentes conexos (CC), cálculo da árvore geradora mínima, e cálculo de caminhos mínimos de fonte única. Os primeiros três problemas consideram uma sequência de N elementos, enquanto que os três últimos têm como entrada um grafo $G = (V, E)$, com $|V|$ nós e $|E|$ arcos. A título de exemplo, vamos detalhar um pouco mais a análise do algoritmo `Scan(N)`. A leitura de N itens (sendo que N é maior que o tamanho de memória disponível) pode ser feita da seguinte forma: leia $\frac{N}{B}$ conjuntos de itens (blocos) de tamanho B. Com isso, o número de I/Os que serão executados será exatamente $\frac{N}{B}$, que corresponde ao número de acessos ao disco para efetuar as leituras dos blocos.

3. Desafios algorítmicos

Algoritmos de memória externa estão sendo estudados há bastante tempo na ciência da computação. A necessidade do seu estudo surgiu porque a memória principal era pequena, e o principal meio para armazenar dados era através das fitas magnéticas. A

disponibilidade de discos rígidos baratos e a memória principal crescente diminuiu o interesse nesses algoritmos nos anos oitenta e noventa. Esse interesse ressurgiu no final dos anos noventa com o crescimento dos dados e o crescente *gap* entre o tamanho da memória principal e a capacidade dos discos. Devido a isso, a área de algoritmos em memória externa está bem avançada, quando comparada com as outras duas abordadas neste artigo. Nos últimos anos houve um grande avanço nessa área. Por exemplo, máquinas de busca como o Google ou Yahoo só funcionam com seu algoritmos projetados em memória externa. Mesmo assim os resultados em diversas áreas ainda são preliminares. Ainda, o projeto em memória externa de algoritmos básicos de grafos, como árvores geradoras mínimas e caminhos mínimos de fonte única, foram recentemente propostos [Dementiev et al. 2004, Arge et al. 2000]. No caso de árvores geradoras mínimas para grafos não direcionados, o melhor algoritmo conhecido em memória externa não é baseado no algoritmo de Prim ou Kruskal, e sim no algoritmo de Boruvska [Dementiev et al. 2004].

Já os outros dois modelos de algoritmos para grandes volumes de dados apresentados nas seções anteriores encontram-se num estágio anterior de desenvolvimento. Os primeiros artigos usando esses modelos foram publicados há menos de 10 anos. Quando compara-se o avanço dos estudos e aplicações entre os modelos apresentados, observa-se uma grande diferença entre eles.

Os algoritmos cache-eficientes possuem uma estrutura bem definida, e resultados teóricos recentes apontam um avanço significativo nos últimos anos [Frigo 1999, Chowdhury 2007]. Mas ainda são pouco utilizados em situações práticas. Uma razão disso é a falta de implementações e estudos experimentais, que verificam o desempenho das propostas na prática.

Para vários problemas não se sabe, atualmente, se é possível projetar um algoritmo universalmente cache-eficiente, e qual o limite inferior para o número de *cache misses*. Entre eles estão todos os problemas em grafos com estrutura arbitrária, cujo acesso à memória é menos previsível. Um outro desafio para a pesquisa é explorar a aplicabilidade de paradigmas tradicionais de projeto de algoritmos. Vários algoritmos conhecidos usam estratégias gulosas, programação dinâmica ou divisão e conquista, por exemplo. É uma pergunta em aberto quais desses paradigmas, sobre quais condições permitem uma implementação cache-eficiente. Um primeiro estudo mostra que é possível projetar algoritmos eficientes para certas classes da programação dinâmica [Chowdhury 2007].

Ao contrário, a área de algoritmos de fluxo de dados se salientou por suas aplicações práticas (em telecomunicações), seguido de uma fundamentação teórica ainda recente. Tais algoritmos ainda encontram-se distribuídos em uma variedade de modificações do modelo, que muitas vezes permeiam outros modelos algoritmos. Dos poucos resultados teóricos, poucos se aplicam na prática. Ainda é um desafio explorar os limites teóricos desse modelo. Como o modelo só permite um acesso restrito aos dados, ainda nem é claro quais problemas permitem um algoritmo nesse modelo. Em compensação, a necessidade de processamento de dados de fluxo contínuo fez com que alguns algoritmos fossem desenvolvidos para casos específicos.

De forma geral, em todos os modelos existem poucos algoritmos eficientes e

pouca experiência prática em implementações. Portanto, a verificação desses modelos através de estudos experimentais é importante. Ainda existem poucas implementações disponíveis, e ainda menos estudos sistemáticos do desempenho deles. Como os modelos são recentes, o seu valor preditivo ainda tem que ser confirmado. Experimentos inicias com algoritmos universalmente cache-eficientes, por exemplo, mostram que só alguns resultados teóricos refletem a prática [Chowdhury and Ramachandran 2006b], e os limites dos modelos, bem como técnicas para adaptar as implementações a situações práticas, ainda têm que ser explorados.

Um outro desafio atual e futuro é o projeto de algoritmos sublineares. Algoritmos podem ser sublineares em termos de tempo ou memória. Os algoritmos de tempo sublinear em geral consideram uma amostra dos dados. Para estes algoritmos o desafio é definir qual tipo, bem como a quantidade de amostras, que deverão ser processadas de forma que seja possível estimar uma propriedade sobre os dados. Para os algoritmos de espaço sublinear, os dados podem ser lidos uma ou mais vezes, mas a memória disponível é sublinear em relação ao tamanho da entrada.

Ainda, um outro desafio é o estudo de novos modelos que combinem os modelos existentes. Um exemplo são modelos híbridos que permitem analisar algoritmos de fluxo de dados que usam memória externa no processamento local. Esse modelo híbrido é bem plausível, visto que muitos algoritmos usam memória que cresce sublinearmente com o tamanho da entrada. Supondo um grande volume de dados, a memória principal pode ser insuficiente para armazenar os dados do algoritmo.

Em resumo, consideramos como os principais desafios para processamento de dados massivos:

- Estabelecer limites inferiores e superiores para problemas em todos os modelos discutidos.
- Projetar algoritmos em memória externa sobre estruturas de dados irregulares. Exemplos são a busca em profundidade em grafos direcionados, bem como a computação de componentes fortemente conectados.
- Projetar algoritmos cache-eficientes para problemas fundamentais em grafos, na álgebra linear e processamento de *strings*.
- Projetar algoritmos de fluxo de dados para problemas fundamentais em grafos.
- Estudar equivalências entre os diferentes modelos de fluxo de dados propostos.
- Propor algoritmos híbridos para problemas, tal como resolver o algoritmo `PageRank` para classificação de páginas web [Page et al. 1998] usando mais que um dos modelos abordados neste artigo.

A seguir, apresentamos medidas que naturalmente difundem o conhecimento e uso de tais algoritmos:

1. Ensinar as técnicas fundamentais de projeto desses algoritmos.
   As técnicas devem ser adaptadas, ou têm que ser modificadas. Por exemplo, algoritmos de divisão e conquista, tal como o algoritmo de Strassen para multiplicação de matrizes [Strassen 1969], são resolvidos eficientemente quando projetados sob o modelo universalmente cache-eficiente. A questão é entender quais são os

projetos via programação dinâmica que podem ser processados de forma cache-eficiente. No caso de memória externa, novas técnicas como *time-forward-processing* [Arge 2003, Y.J.Chiang et al. 1995] são importantes.

2. Projetar algoritmos básicos usando modelos de algoritmos para dados massivos, tais como algoritmos de busca, ordenação, algoritmos fundamentais em grafos – busca por largura, busca por profundidade, caminhos mais curtos – álgebra linear, solução de sistemas lineares, entre outros.
3. Estudo de probabilidade. Com o aumento do volume dos dados, o estudo avançado de probabilidade tem se tornado um conhecimento cada vez mais importante para os pesquisadores da área de algoritmos. Futuramente será essencial. Desta forma, desde agora deve-se difundir mais este conhecimento.
4. Criação de centros e grupos de estudo sobre processamento de dados massivos. Por exemplo, em 2007 foi criado na Europa (Dinamarca) um centro de estudos de algoritmos para dados massivos, com foco nos três modelos de algoritmos discutidos neste artigo. O Centro MADALGO (Center for Massive Data Algorithmics)[3] tem suporte para prover Escolas de Verão e Workshops no assunto, contratar doutorandos, pós-doutorandos e pesquisadores convidados.

4. Conclusões

O volume de dados cresce exponencialmente, e gera novos desafios para diversas áreas, dentre elas está a ciência de computação. São necessários novos modelos de processamento e novas abordagens algorítmicas para resolver esses problemas eficientemente. A pesquisa nessa área ainda é recente, e atualmente está em fase de definição e validação de novos modelos em que resultados preliminares foram obtidos.

Três modelos algoritmos para processamento massivo já foram propostos e estão sendo usados. Cada um deles possui aplicação em situações específicas:

- Algoritmos cache-eficientes: utilizados em situações nas quais os dados são alocados completamente na memória principal, mas o processamento em cache pode ser explorado eficientemente, diminuindo o tempo de execução do algoritmo.
- Algoritmos de memória externa: indicados em situações que não é possível alocar os dados em memória principal.
- Algoritmos de fluxo de dados: indicados em duas situações i) os dados são processados conforme são recebidos, dispondo de memória limitada. ii) uma estimativa sobre os dados é suficiente, dispondo de espaço em disco para os dados, mas usando memória sublinear.

Tais modelos de algoritmos massivos devem ser mais conhecidos e mais desenvolvidos de forma que se torne um conhecimento comum poder identificar quando a resolução de um problema pode ser beneficiada pelo uso da técnica algorítmica adequada. Finalmente, é preciso identificar que problemas podem ser melhor resolvidos se usar os modelos de algoritmos para dados massivos. O processamento com dados massivos não é apenas útil, é necessário e fundamental na realidade atual.

[3] www.madalgo.au.dk

Referências

Abello, J. M. and Vitter, J. S., editors (1999). *A Functional approach to external graph algorithms*. Dimacs Series In Discrete Mathematics And Theoretical Computer Science. American Mathematical Society. 306p.

Aggarwal, C. C. (2007). *Data Streams: Models and Algorithms*. Springer.

Arge, L. (2003). The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24.

Arge, L., Brodal, G. S., and Toma, L. (2000). On external-memory MST, SSSP, and Multi-way Planar Graph Separation. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory, Lecture Notes In Computer Science*, pages 433–447.

Buriol, L., Frahling, G., Leonardi, S., Sohler, C., and Marchetti-Spaccamela, A. (2006). Counting triangles in data streams. In *25th ACM Symposium on Principles of Database Systems (PODS2006), Chicago, Illinois*, pages 253–262.

Buriol, L. S., Frahling, G., Leonardi, S., and Sohler, C. (2007). Estimating clustering indexes in data streams. In *15th Annual European Symposium on Algorithms (ESA), Springer Series Lecture Notes in Computer Science*.

Chowdhury, R. A. (2007). *Cache-efficient Algorithms and Data Structures: Theory and Experimental Evaluation*. PhD thesis, University of Texas, Austin.

Chowdhury, R. A. and Ramachandran, V. (2006a). Cache-oblivious dynamic programming. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 591–600, New York, NY, USA. ACM.

Chowdhury, R. A. and Ramachandran, V. (2006b). Cache-oblivious dynamic programming. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 591–600, New York, NY, USA. ACM.

Conselho nacional de trânsito (2006). Resolução no. 212, Dispõe sobre a implantação do Sistema de Identificação Automática de Veículos – SINIAV em todo o território nacional.

Cormode, G., Muthukrishnan, S., Datar, M., and Indyk, P. (2003). Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, 15(3):529–540.

Cortes, C., Fisher, K., Pregibon, D., and Rogers, A. (2000). Hancock: A language for extracting signatures from data streams. *Proceedings of the ACM SIGKDD International Conference on Knoledge Discovery and Data Mining, ACM*, pages 9–17.

Cranor, C., Johnson, T., Spataschek, O., and Shkapenyuk, V. (2003). Gigascope: A stream database for network applications. *Proceedings of ACM SIGMOD*, pages 647–651.

Datar, M. and Muthukrishnan, S. (2002). Estimating rarity and similarity over data stream windows. *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 323–334.

Dementiev, R., Sanders, P., Schultes, D., and Sibeyn, J. F. (2004). Engineering an external memory minimum spanning tree algorithms. In *3rd IFIP International Conference on Theoretical Computer Science (TSC2004)*, pages 195–208, Toulouse, France.

Frigo, M. (1999). *Portable High-Performance Programs*. PhD thesis, MIT.

Frigo, M., Leiserson, C. E., Prokop, H., and Ramachandran, S. (1999). Cache-oblivious algorithms. In *Proc. 40th IEE Sympos. Found. Comp. Sci.*, pages 285–297.

Hennessy, J. L. and Patterson, D. A. (2002). *Computer architecture: A quantitative approach*. Morgan Kauffmann Publishers Inc., 3rd edition.

IDC (2007). The expanding digital universe: A forecast of the worldwide information growth through 2010. White paper sponsored by EMC-IDC.

Lander, E. S. (1999). Array of hope. *Nature Genetics Supplement*, 21:3–4.

Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. *In Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–369.

Meyer, U., Sanders, P., and Sibeyn, J., editors (2003). *Algorithms for Memory Hierarchies: Advanced Lectures*. Springer, 1st edition.

Muthukrishnan, S. (2005). *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers Inc.

NCBI. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html. Online. Acessado em 22/09/2007.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Technical Report Technical report, Stanford University.

Prokop, H. (1999). *Cache-oblivious algorithms*. PhD thesis, MIT.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356.

Szalay, A. and Gray, J. (2002). 2020 computing: Science in an exponential world. *Nature*, 440:413–414.

Vitter, J. S. (2007). External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2).

Whaley, R. C., Petitet, A., and Dongarra, J. (2001). Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27(1-2):3–35.

Y.J.Chiang, M.T.Goodrich, Grove, E., Tamassia, R., Vengroff, D., and Vitter, J. (1995). External-memory graph algorithms. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149.

Z. Bar-Yosseff, R. Kumar, D. S. (2002). Reductions in streaming algorithms, with an application to counting triangles in graphs. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632.