**Processamento de Dados Massivos**

A evolução dos sistemas computacionais e a difusão do acesso à Internet em diversas formas estão dando origem ao que muitos denominam “a Idade da Informação” [[1]](https://pt.wikibooks.org/wiki/Processamento_de_Dados_Massivos/Introdu%C3%A7%C3%A3o#cite_note-1). Os volumes massivos de dados produzidos pelas mais diversas fontes, como medições coletadas por sensores dos mais diversos tipos, registros (*logs*) dos serviços oferecidos pela Web, diversos tipos de conteúdo produzidos pelos usuários no que se convencionou chamar de Web 2.0, tudo isso apresenta novos desafios e possibilidades. As possibilidades são inúmeras, sempre baseadas na ideia de se derivar novo conhecimento a partir da análise de tais volumes de dados. Os desafios, por sua vez, envolvem questões como definir formas eficientes de coletar e armazenar toda informação, garantir sua preservação e acesso eficientes, bem como extrair informação útil de tais volumes de dados. Dado o interesse geral devido ao seu potencial, essa área vem recebendo grande atenção na imprensa técnica e mesmo na imprensa em geral, sendo denominada “*Big-data*”. Neste trabalho, pretendemos discutir o problema de como processar esses grandes volumes de dados.

Exatamente o que se entende por *big-data* depende bastante do contexto Apesar de normalmente se associar o conceito apenas a volumes extremamente grandes de dados, na verdade a definição abrange três dimensões, que devem ainda ser consideradas em perspectiva para cada usuário: volume, velocidade e variedade

Certamente, **volume** é uma dimensão claramente associada a dados massivos. Um infográfico produzido por GOOD, Oliver Munday e IBM []](https://pt.wikibooks.org/wiki/Processamento_de_Dados_Massivos/Introdu%C3%A7%C3%A3o#cite_note-4)representa alguns desses volumes. Entre eles, pode-se ver que a cada minuto são carregados no Youtube o equivalente a 20 horas de vídeo, que há em média 50 milhões de tweets por dia e que 2,9 milhões de mensagens de e-mail são enviados por segundo. Entretanto, a definição de volume massivo deve ser também ajustada em função dos recursos disponíveis para seu processamento. Nem todas as organizações possuem os recursos computacionais de uma empresa como Google ou Facebook; em muitos casos, dados na casa de centenas de Gigabytes já apresentam um desafio para serem processados, considerando-se os recursos disponíveis.

Uma segunda dimensão é a **velocidade** com que os dados são gerados e com que precisam ser processados em diversos casos. Por exemplo, o Observatório da Web coleta um grande volume de informações em tempo real de diversas fontes, como sítios de notícias, *blogs* e *Twitter*, para gerar diversas análises atualizadas a cada minuto durante eventos transmitidos ao vivo. Nesse caso, o desafio é processar o volume de dados gerado ao longo do tempo e em tempo hábil, o que é denominado processamento de *streams*.

Finalmente, a **variedade** dos dados e dos resultados esperados também são determinantes para a definição de *big-data*. A possibilidade de se coletar informações textuais, fotos, áudio e vídeo tornam muitas vezes inviável o uso de sistemas de gerência de bancos de dados tradicionais. A exploração de informações de redes complexas, representando relacionamentos entre pessoas e/ou eventos dá origem a grafos complexos, que também não são facilmente armazenados em sistemas convencionais.

Enfim, o processamento de dados massivos de forma eficiente exige o uso de paralelismo, tanto para o armazenamento dos dados quanto para seu processamento. Dessa forma, o acesso aos dados é acelerado, já que leituras em paralelo se tornam possíveis, e o processamento é dividido entre diversas unidades de processamento, acelerando a geração de respostas. Esse modelo de paralelismo, usualmente conhecido como o padrão *dividir-para-conquistar*, ou *divisão-e-conquista*.

Apesar desse modelo de processamento já ser largamente conhecido da comunidade de processamento paralelo, ele também vem ganhando larga aceitação nas tarefas de *big-data* pelo surgimento de ambientes de processamento desenvolvidos especificamente para esse tipo de atividade. Além disso, esses grandes volumes de dados surgem normalmente no contexto de aplicações em nuvem, que executam em grandes *datacenters*, onde recursos para armazenamento e processamento distribuído já existem na forma de um grande número de máquinas convencionais interligadas por redes de alta velocidade.

**O ambiente de execução**

Dados massivos estão frequentemente relacionados ao modelo de computação em nuvem. Nesse caso, serviços são oferecidos de forma transparente de localização e adaptando-se às demandas dos usuários. O processamento real e, de forma relacionada, o armazenamento dos grandes volumes de dados envolvidos, se dá em *datacenters* instalados em algum lugar da rede. Para melhor entendermos os modelos de processamento de dados massivos devemos, então, entender como esse ambiente se organiza e como os dados são armazenados.

**Armazenamento de dados**

Considerando-se essa arquitetura, para garantir o melhor desempenho durante o processamento, dados devem ser distribuídos entre os discos das diversas máquinas de forma que possam ser lidos em paralelo, para serem processados também em paralelo. Apesar de algumas soluções mais simples deixarem a tarefa de organizar os dados dessa forma a cabo do usuário, idealmente essa funcionalidade deve ser oferecida por um sistema de armazenamento distribuído. Historicamente, sistemas de arquivo paralelos []](https://pt.wikibooks.org/wiki/Processamento_de_Dados_Massivos/Aspectos_gerais_do_ambiente_de_Big_Data#cite_note-3)foram desenvolvidos para esse fim no contexto de sistema de processamento de alto desempenho (HPC). Entretanto, no contexto de dados massivos, observou-se que os padrões de acesso aos dados tendem a ser mais simples que nos ambientes de HPC e soluções particulares têm se mostrado mais adequadas nesse contexto. O modelo mais adotado para esse fim tem sido o modelo proposto pelo Google File System (GFS) e também implementado pelo Hadoop File System (HDFS).

No Google File System, arquivos são escritos sequencialmente quando criados, ou dados são acrescentados sequencialmente ao final de um arquivo já existente (*append*). Alterações não são possíveis depois que um dado é escrito e o objetivo é tornar eficiente padrões de acesso que percorram todo o arquivo de cada vez. Para isso, arquivos são dividos em blocos grandes (usualmente 64 MB), que são distribuídos pelos discos de diversas máquinas do sistema e replicados para aumentar a disponibilidade dos mesmos.

No GFS (e HDFS), cada sistema de arquivos tem uma máquina escolhida como servidor do espaço de nomes (*namenode*), responsável também por armazenar os metadados de cada arquivo. A estrutura de arquivos do GFS é isolada e independente da árvore de diretórios usual acessada por cada máquina. Os acessos se dão através de uma biblioteca especial e não através da interface de chamadas de sistema.

Ao criar um arquivo, um cliente registra seu nome com o *namenode* e começa a escrever os blocos em máquinas escolhidas no mesmo *rack* em que executa o processo escritor (para aproveitar a melhor banda entre máquinas no mesmo *rack*). Uma vez escrita essa primeira cópia o cliente continua seu processamento, enquanto o servidor de armazenamento do GFS que recebeu o bloco (denominado *datanode*) passa a replicar o mesmo em um outro *datanode*, escolhido normalmente em um outro *rack* (para evitar problemas com falhas que afetem um *rack* inteiro). Usualmente cada bloco é replicado três vezes (apesar desse número pode ser configurado pelo usuário para cada arquivo) e uma terceira cópia é feita então pelo segundo datanode para um outro *datanode* no mesmo *rack* (de novo, aproveitando a banda local).

Ao abrir um arquivo para leitura, o cliente se comunica como *namenode* e recebe dele a lista de blocos do arquivo, com a identificação dos namenodes que armazenam cada bloco. O cliente pode então escolher de qual *namenode* requisitar os dados de que necessita baseado em critérios de proximidade. Dessa forma, vários clientes podem escolher ler partes diferentes do arquivo a partir de *datanodes* diferentes, aumentando a banda de leitura dos dados.

Periodicamente, processos administrativos verificam a disponibilidade de cada *datanode* e os blocos armazenados em cada um, comandando novas cópias ou movendo blocos entre nós para garantir o balanceamento entre eles. Esses processos também cuidam de comandar a redistribuição de blocos quando novos *datanodes* são acrescentados ao sistema, ou comandar novas replicações quando algum nó falha.

Esses são elementos importantes do ambiente normalmente utilizado para processamento de dados massivos. A seção a seguir discute os principais aspectos de escalabilidade e eficiência que precisam ser considerados nesse caso, para então discutirmos nas seções seguintes os modelos de processamento disponíveis.

**O modelo MapReduce**

Conforme discutido anteriormente, um dos aspectos essenciais do processamento de dados massivos é exprimir o processamento de forma que o maior volume possível de dados seja acessado (lido) e processado em paralelo, aumentando a velocidade final do processamento. Um modelo que se mostrou particularmente bem sucedido nesse aspecto foi o MapReduce, proposto pela Google em 2004

O modelo opera sobre o Google File System, de onde os dados podem ser lidos de forma eficiente, em paralelo. Os arquivos são vistos como listas de registros simples, como linhas, ou registros formatados segundo algum padrão definido pelo usuário. Cada registro lido do arquivo é representado por um par *(chave,valor)* (p.ex., o número de cada linha e o texto nela encontrado).

O processamento a ser realizado deve ser então descrito por duas funções simples que dão nome ao modelo e operam sobre chaves e valores:

* map (), que recebe como entrada pares chave/valor e os processa, gerando como saída novos pares chave/valor, potencialmente de tipos diferentes dos anteriores,
* reduce (), que recebe como entrada chaves produzidas como saída da função anterior, junto com uma lista de todos os valores associados a cada chave, e produz novamente como saída outro(s) par(es) chave/valor.

Conceitualmente, o modelo se reduz a uma versão particular do divisão-e-conquista já discutido anteriormente. Em particular, cada registro (par chave/valor) extraído do arquivo de entrada pode ser potencialmente processado em paralelo, de forma independente de todos os demais. A sincronização durante o processamento ocorre ao final da fase do map, quando todos os pares com chaves iguais devem ser agrupados para serem processados juntos na fase de redução. Novamente há potencialmente oportunidade para

um alto grau de paralelismo, já que cada chave (com sua lista de valores) pode ser processada separadamente.

Apesar de sua relativa simplicidade, a grande aceitação do modelo se deveu em parte à facilidade com que questões como o aproveitamento de paralelismo em diversos níveis (p.ex., máquinas *multicore* em um *cluster*), balanceamento de carga e tolerância a falha são facilmente tratados.

Para melhor aproveitar o poder computacional de cada nó de processamento, o modelo permite ainda a definição de uma função combine (), que serve para realizar o pré-processamento das listas de valores gerados por cada nó que executa a função map(), quando esse processamento pode reduzir o volume de comunicação entre nós. Essa função pode ser usada para já gerar um valor agregado para todos os valores associados por cada chave em um nó, de forma que apenas um valor precise ser enviado para os nós redutores para cada chave. No caso do programa ContaPalavras, o mesmo código da função reduce() poderia ser usado: todas as ocorrências de uma dada palavra em um nó seriam combinadas e ao invés de enviar uma lista de uns para o redutor, cada nó já enviaria o resultado da contagem (soma) da lista local para o redutor, que continuaria somando os valores recebidos dos vários nós de map() para obter a contagem final.

O balanceamento de carga e a tolerância a falhas são obtidos por um mesmo mecanismo: o sistema acompanha o término das tarefas no sistema e quando começa a ter recursos desocupados re-edita tarefas que ainda não foram completadas, aproveitando o primeiro resultado que se torne disponível.