UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JOÃO GABRIEL RODRIGUES DE OLIVEIRA LIMA

STORMSOM – CLUSTERIZAÇÃO EM TEMPO-REAL DE FLUXOS DE DADOS DISTRIBUÍDOS NO CONTEXTO DE BIGDATA

BELÉM
2015

‘

JOÃO GABRIEL RODRIGUES DE OLIVEIRA LIMA

STORMSOM –CLUSTERIZAÇÃO EM TEMPO-REAL DE FLUXOS DE DADOS DISTRIBUÍDOS NO CONTEXTO DE BIGDATA

Dissertação apresentada ao Programa de Pós Graduação em Engenharia Elétrica, Área de Concentração em Computação Aplicada, Universidade Federal do Pará, como requisito parcial para a obtenção do título de Mestre.

Orientador: Prof. Dr. Ádamo Lima de Santana
Coorientador: Prof. Dr. Diego Lisboa Cardoso
Coordenador: Prof. Dr. Evaldo Gonçalves Pelaes

BELÉM

2015

‘

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, meu ponto de equilíbrio e fonte da força, minha grande inspiração e meu protetor que me conduziu durante toda esta caminhada.

À minha família, por sempre estarem comigo, pelo apoio e preocupação, em especial a meus pais, que sempre acreditaram no meu potencial e a meu irmão, por tudo que venceu e construiu em sua vida, sendo hoje a pessoas que mais admiro e respeito.

Ao professor e orientador Ádamo Santana pela extrema confiança que sempre depositou no meu trabalho, pelo apoio nos momentos mais difíceis e pelo seu conhecimento e críticas que foram de extrema importância para meu crescimento pessoal e profissional. Agradeço aos companheiros integrantes do Laboratório de Planejamento de Redes de Alto Desempenho – LPRAD e do Laboratório de Inteligência Computacional e Pesquisa Operacional – LINC, pelo companheirismo, brincadeiras e distrações que tornaram o ambiente mais estimulante e produtivo.

Agradeço ao Ederlon Barbosa, meu grande amigo e irmão, pelo suporte emocional, por todas as brincadeiras e por estar sempre presente em todos os momentos, principalmente os mais difíceis.

Agradeço a Alana Cavalcante, pelo apoio em todos momentos, pela amizade, companheirismo, carinho, compreensão e principalmente paciência.

Enfim, a todos vocês que compartilharam comigo seus conhecimentos e amizade, sou eternamente grato por viver no mesmo tempo em que vocês, e pelo privilégio de tê-los conhecido. Meus mais sinceros agradecimentos. Obrigado.

‘

FICHA CATALOGRÁFICA

Rodrigues de Oliveira Lima, João Gabriel

Stormsom –Clusterização em tempo-real de Fluxos de Dados Distribuídos no Contexto de Big Data– Belém, 2015.

Nº de páginas: 61

Área de concentração: Computação Aplicada.

Orientador: Prof. Dr. Ádamo Lima de Santana.
Coorientador: Prof. Dr. Diego Lisboa Cardoso.

Dissertação de Mestrado–Universidade Federal do Pará (UFPA), Instituto de Tecnologia (ITEC), Programa de Pós-Graduação em Engenharia Elétrica (PPGEE).

1.Neural Network; 2. Real Time Computing; 3.BigData

‘

SUMÁRIO

‘

‘

LISTA DE ILUSTRAÇÕES

‘

LISTA DE TABELAS

‘

LISTA DE ABREVIATURAS

API	Application Programming Interface
CRM	Customer Relationship Management
DBMS	Database Management System
HDFS	Hadoop Distributed File System
NoSQL	Not Only SQL
SQL	Structured Query Language
XML	Extensible Markup Language
NAS	Network Attached Storage

‘

RESUMO

STORMSOM – CLUSTERIZAÇÃO EM TEMPO-REAL DE FLUXOS DE DADOS DISTRIBUÍDOS NO CONTEXTO DE BIGDATA

Cresce cada vez mais a quantidade de cenários e aplicações que algoritmo necessitam de processamento e respostas em tempo real e que se utilizam de modelos estatísticos e de mineração de dados a fim de garantir um melhor suporte à tomada de decisão. As ferramentas disponíveis no mercado carecem de processos computacionais mais refinados que sejam capazes de extrair padrões de forma mais eficiente a partir de gandes volumes de dados. Além disso, há a grande necessidade, em diversos cenários, que o os resultados sejam providos em tempo real, tão logo inicie o processo, uma resposta imediata já deve estar sendo produzida. A partir dessas necessidades identificadas, neste trabalho propomos um processo autoral, chamado StormSOM, que consiste em um modelo de processamento, baseado em topologia distribuída, para a clusterização de grandes volumes de fluxos, contínuos e ilimitados, de dados, através do uso de redes neurais artificiais conhecidas como mapas auto-organizáveis, produzindo resultados em tempo real. Os experimentos foram realizados em um ambiente de computação em nuvem e os resultados comprovam a eficiência da proposta ao garantir que o modelo neural utilizado possa gerar respostas em tempo real para o processamento de *Big Data*.

Descritores: Stream Computing, Neural Network, BigData.

1 INTRODUÇÃO

Analisar crescentes quantidades de dados provenientes de diversas fontes, não é uma tarefa trivial. Para tal, faz-se necessária a utilização de técnicas computacionais avançadas a fim de descobrir correlações potencialmente úteis entre os diversos dados e encontrar regras quantitativas e qualitativas associadas aos mesmos.

O primeiro desafio deste escopo trata-se de processar uma massiva quantidade de dados, sendo que, para diversos domínios, é primordial que essas informações sejam coletadas e processadas em tempo real, para tal, faz-se necessário o estudo de novas abordagens e ferramentas. Atualmente, a ferramenta mais utilizada para lidar com grande volume de dados é o Apache Hadoop (Tom White,2012). Para trabalhar sobre lotes de dados estáticos, o Hadoop opera muito bem, no entanto, quando há a necessidade de um processamento em tempo real, onde a necessidade gira em torno de um fluxo de informações, o mesmo, certamente, não é uma boa opção (J. Kornacker, 2012).

É cada vez mais comum a existência de cenários que requerem o processamento de fluxos de dados em tempo real, onde os dados chegam e uma resposta é executada para reagir à essa nova informação contribuindo imediatamente para a tomada de decisão. Este tipo de computação é chamada de *Stream Computing* ou processamento de fluxo de dados (Andrade, 2013).

No modelo de data mining tradicional, os dados são coletados e o processo de extração do conhecimento é iniciado (Joey, 1997). Na prática faz-se a garimpagem em cima de dados estáticos que não refletem o momento atual, mas sim o contexto passado. Com *Stream Computing* este processo de mineração de dados pode ser efetuado em tempo real. Em vez de disparar consultas sobre uma base de dados estática, coloca-se uma corrente contínua de dados (*streaming*) atravessando um conjunto de consultas e operações. Nesse contexto, podemos pensar em inúmeras aplicações, sejam estas em finanças, saúde, manufatura, dentre outras (IBM, 2014).

Em processamento de fluxo de dados, ao invés da noção de volume, é determinante a noção de velocidade e uma nova ferramenta chamada Apache Storm tem se tornado cada vez mais popular nesse contexto, devido seu uso por parte de diversas empresas, onde mostrou-se realmente uma ferramenta promissora para a

computação de fluxos de dados. Em linhas gerais, o Apache Storm faz para processamento em tempo real o mesmo que o Hadoop para faz para o processamento em lote.

O cenário atual demanda soluções computacionais inteligentes que gerem resultados eficazes e sejam capazes de identificar padrões em cenários críticos como de *Big Data*, além da necessidade de respostas cada vez mais imediatas. A todo momento surgem novas ferramentas que tentam agregar mais valor, de forma mais facilitada, ao processo de extração do conhecimento. Cursos, treinamento e especializações têm se inserido nas empresas para buscar a formação técnica de seus recursos para serem capazes de competir neste novo cenário.

Por esse motivo, há grande abertura para a proposição de novas soluções que venham resolver ou contribuir para a resolução das problemáticas que envolvem mineração de dados sobre *big data* em tempo real. Apesar das diversas soluções de mercado, são raras as que buscam modelos computacionais avançados de inteligência e mineração para compor seus produtos, devido a complexidade desses modelos e a necessidade de conhecimento técnico-científico para aplica-los e interpretá-los, além do grande desafio em implementá-los nesses cenários, haja vista que a grande maioria dos modelos não foi concebido para lidas com grandes volumes de dados, menos ainda provendo respostas em tempo real.

Neste trabalho é proposto um modelo de processamento autoral que utiliza o modelo neural conhecido como Mapas Auto-Organizáveis para clusterizar grandes volumes de dados. Chamado de StormSOM, foi concebido para trabalhar sobre um fluxo contínuo de volume distribuídos de dados, retornando respostas em tempo real. Neste processo há a junção dos conceitos de processamento de fluxo de dados em tempo real, mineração de dados e processamento distribuído.

Esta proposta vem de encontro com a problemática supracitada, pois se propõe a resolver um cenário crítico, comum no mercado atual, aplicando conceitos e modelos computacionais mais refinados para a extração de padrões sobre grandes volumes de dados, o que a torna diferenciada. Além disso, garante o tratamento tanto em relação ao volume de dados quanto à velocidade do processamento, garantindo também que os resultados do processo de *clusterização* sejam apresentados em tempo real para tomada de decisão imediata ou integrado a outras soluções e tecnologias que se beneficiem desses resultados.

Será apresentada toda a arquitetura de processamento para a execução do processo de *clusterização*, utilizando banco de dados não-relacional – NoSQL - executada em um ambiente de *Cloud Computing.* As simulações executadas sobre o cenário de estudo, comprovaram a eficiência da solução em termos quantitativos ao obter, analisar e processar os dados em tempo real.

No Capítulo 2, todo o referencial teórico deste trabalho será contextualizado, no Capítulo 3 serão apresentados os trabalhos correlatos que dão embasamento e justificam o desenvolvimento desta nova metodologia. No Capítulo 4 a arquitetura da metodologia será detalhada, com todas as suas características e particularidades. No Capítulo 5, as simulações de desempenho para diversos cenários serão apresentadas. Por fim, no Capítulo 6 é feita a conclusão deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as tecnologias e técnicas utilizadas para a que seja possível compreender a implementação da solução proposta neste trabalho. Muitos conceitos ainda são muito recentes e estão sendo maturados a cada dia pelo mercado tecnológico, outros, entretanto, já estão bem consolidados como é o caso modelo neural de mapas auto-organizáveis.

2.1 BIG DATA

O termo *Big Data* foi usado pela primeira vez em dezembro de 2008 como *Big Data Computing* em uma publicação feita por Randon Bryant, Randy Katz e Edward (Bakshi, 2012). Com o crescimento do uso de ferramentas sociais na WEB o número de dados gerados por usuários tem aumentado de tal modo que ficou difícil a coleta, o processamento e analise destes por empresas web (Bakshi, 2012). Muitos desses dados são encontrados em forma de texto, ou seja, são dados não estruturados ou semiestruturados. Esses arquivos de textos representam aproximadamente 80% dos dados de uma empresa.

O *Big Data* usa tecnologias subjacentes como NAS (Network AttachedStorage), sistemas de arquivos distribuídos, banco de dados orientados a objetos e bancos de dados orientados a documentos (WEC, 2012). Conjuntamente com o NAS, são utilizados sistemas de arquivos distribuídos. A maioria dos sistemas de arquivos distribuídos possuem uma arquitetura cliente-servidor, mas existem soluções completamente descentralizadas, como os sistemas baseados em cluster. O objetivo principal de um sistema de arquivos distribuídos é lidar com o grande crescimento das informações principalmente devido ao crescimento da WEB com a evolução do uso de redes sociais. Porém os dados que normalmente compõem o que podemos chamar de Big data são semiestruturados.

As características de um dado semiestruturado são as seguintes (WEC, 2012):

- Modelos de Dados que são definidos após a existência dos dados propriamente ditos;

- Não existe uma estrutura padrão, os dados semanticamente similares podem ter mais ou menos informações em relação aos outros;
- A estrutura fica implícita na forma como os dados são representados;
- Quando existe um conjunto de atributos onde cada um pode ser de um tipo diferente, faz com que haja uma estrutura extensa;
- A estrutura e os valores dos dados mudam frequentemente;

O *BigData* tem como função descobrir padrões de negócios através de dados não estruturados ou semiestruturados.

Na próxima subseção será apresentada uma estratégia para o armazenamento de dados não estruturados para se trabalhar com *BigData*, os chamados bancos de dados não relacionais.

2.2 NOSQL - BANCOS DE DADOS NÃO RELACIONAIS

NoSQL (Not Only SQL) é um termo usado para designar os sistemas de gerenciamento de banco de dados que diferem, de alguma forma, do clássico sistema de gerenciamento de banco de dados relacional (RDBMS). Os bancos de dados não relacionais não exigem estruturas fixas de tabelas e geralmente evitam operações de associação (a junção de tabelas dos bancos relacionais). Não fornecem propriedades ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) e tipicamente foram concebidos para escalar horizontalmente (Perroud, 2014).

Para usuários especialistas em bancos de dados relacionais, pode ser perturbador entender alguns princípios de bancos de dados não relacionais. Os sistemas NoSQL podem ser mais complexos que os bancos de dados tradicionais, assim como, em alguns aspectos, são mais simples. Estes sistemas podem se adaptar de modo prático a um conjunto de dados volumoso, mas, em contrapartida, algumas operações devem ser feitas de forma manual e específica para cada cenário.

Um aspecto importante da definição dos bancos de dados não relacionais é a noção de escalabilidade. A escalabilidade pode ser descrita como uma propriedade desejável de um sistema, uma rede, ou um processo, que indica a sua capacidade de manipular uma quantidade crescente de trabalho de uma forma normal ou a ser facilmente expandida (Perroud, 2014). A escalabilidade pode ser feita de duas dimensões:

- Escalabilidade Vertical: aumento dos recursos de hardware em um nó de processamento;
- Escalabilidade Horizontal: adicionar nós de processamento ao cluster;

Bancos de dados NoSQL geralmente escalam horizontalmente: se o banco de dados precisa de mais recursos (armazenamento, poder computacional, memória, etc.), um ou mais nós devem ser adicionados ao cluster. No entanto, a adição de mais máquinas aumenta o risco de falhas (problemas de rede, espaço em disco ou memória insuficiente, problemas de hardware, etc.). Por isso, uma propriedade essencial de um sistema distribuído é a tolerância a falhas, que permite que um sistema continue funcionando corretamente em caso de falha de alguns dos seus componentes (Abd-El-Barr, 2013).

Os bancos de dados NoSQL apresentam algumas características fundamentais que os diferenciam dos tradicionais sistemas de bancos de dados relacionais, tornando-os adequados para armazenamento de grandes volumes de dados não estruturados ou semi- estruturados. Na Tabela1, serão mostradas essas características:

	Relacional	Não-Relacional
Escalonamento	Possível, porém, complexo. Devido a natureza da estruturada do modelo, a adição de forma dinâmica e transparente de novos nós no grid não é realizada de modo natural.	Uma das principais vantagens desse modelo. Por não possuir nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos.
Consistência	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam um maior grau de rigor quanto à consistência dasinformações.	Realizada de modo eventual no modelo: só garante que, se nenhuma atualização for realizada sobre o item de dados, todos os acessos a

		esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	Outro fator fundamental do sucesso desse modelo. O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados, seja atendida por parte do sistema e que o sistema fique menos tempo não-disponível.

Tabela 1– Comparativo entre as principais características entre Bancos de dados relacionais e não-relacionais

Apesar dos aspectos destacados, existem particularidades para o uso de bancos não relacionais que envolvem uma decisão mais estratégica, esse conceito é conhecido comoTeorema CAP e será apresentado na próxima subseção.

2.2.1 O Teorema CAP

O teorema CAP (*Consistency, availability, Network Partition Tolerance*), proposto pelo cientista da computração Eric Brewer, em 2000 no Simpósio sobre Princípios de Computação Distribuída (Greiner, 2014), afirma que é impossível para um sistema fornecer simultaneamente todas as três propriedades a seguir:

- **Consistência**: todos os nós têm acesso ao mesmo conjunto de dados simultâneamente;
- **Disponibilidade**: garantia de que cada solicitação receberá uma resposta;
- **Tolerância a particionamento de dados na rede**: O sistema não poderá responder incorretamente ao particionamento de dados no cluster, mesmo diante de uma falha de rede;

Todas estas propriedades são desejáveis e esperadas, mas dependendo da necessidade de cada cenário, podemos aplicar no máximo dois deles. Teremos,

portanto, sistemas que proporcionam consistência e disponibilidade, consistência e tolerância ao particionamento e, finalmente, disponibilidade e tolerância ao particionamento. Vamos definir mais detalhadamente essas três propriedades, pois são de muita importância para este trabalho, onde tivemos que optar por algumas dessas propriedades para alcançarmos os resultados esperados para o processamento correto da clusterização pelo modelo neural.

O Teorema de CAP vem equilibrar as operações e transações de dados de acordo com cada aplicação. Em um sistema distribuído, não é possível evitar as partições de rede. Portanto, deve-se encontrar o equilíbrio certo entre a consistência e a disponibilidade. Para ilustrar isso, vamos considerar duas grandes plataformas web muito conhecidas atualmente: Facebook e Ebay. Se uma atualização de status é feita no Facebook não é realmente um problema se ele não fizer esta atualização ao mesmo tempo para todos os usuários no mundo. Usuários preferem ter uma plataforma que está sempre disponível do que totalmente consistente. O Facebook deve, portanto, usar sistemas que fornecem tolerância ao particionamento e disponibilidade. Por outro lado, vamos imaginar que alguém quer comprar um produto no Ebay e há apenas uma unidade em estoque. Pouco antes de pagar, o site mostra um erro dizendo que não há nenhum outro produto disponível. Para este caso, o problema da consistência dos dados tornar-se bastante expressivo. Portanto, uma plataforma como o Ebay deve usar sistemas que proporcionam consistência e tolerância ao particionamento.

No âmbito dos bancos de dados não relacionais, sistemas que proporcionam consistência e disponibilidade têm problemas com partições e, normalmente, lidam com isso através do uso de replicação. Exemplos desses sistemas incluem RDBMSs tradicionais como MySQL ou PostgreSQL, Vertica (orientada coluna), Aster dados (relacional) e Greenplum (relacional). Por outro lado, o Cassandra (orientado a coluna/tabular), Amazon Dynamo (valor-chave) ou CouchDB (orientado a documentos) são sistemas que são eventualmente consistentes, estão sempre disponíveis e tolerante a partições (AP). Finalmente, sistemas como o Google BigTable (orientada a coluna / tabular), HBase (orientada a coluna / tabular), MongoDB (orientado a documentos) e BerkeleyDB (keyvalue) são consistentes e tolerantes a partições (CP), mas podem ter problemas com a disponibilidade.

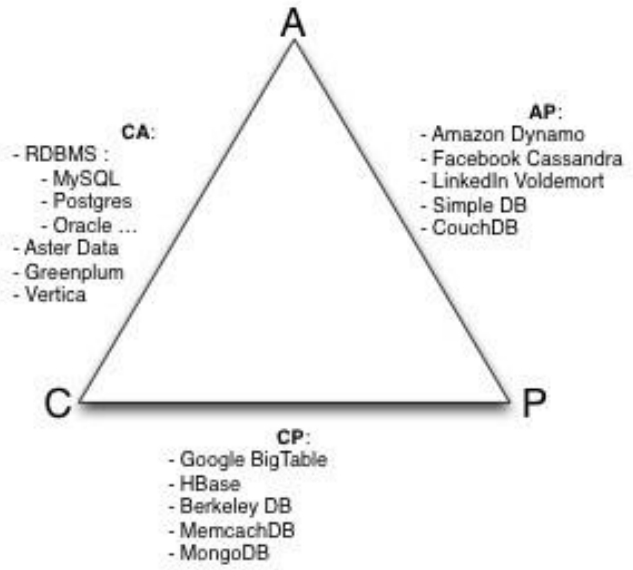

Figura 1 - Guia dos Bancos NoSQL segundo o Teorema CAP (Hurst, 2014)

Pelo que foi explanado nesta subseção, fica claro que a decisão de utilizar um determinado banco de dados depende de cada aplicação. No caso específico deste trabalho, foi decidido utilizar um banco disponível e tolerante a partições para que o objetivo de clusterizar dados em tempo real, ou com baixa latência, fosse alcançado com êxito.

2.2.2 Redis DB

Redis (Seguin, 2014) é banco de dados não relacional open-source, chave-valor, baseado em memória e, principalmentepor esse motivo, possui um alto desempenho na leitura e escrita de grande volume de dados, além de total flexibilidade em sua estrutura. Devido suas características que serão explanadas nesta subseção, foi escolhido para ser o banco de dados utilizado neste trabalho.

2.2.2.1 Modelo de dados

No Redis, uma base de dados é identificada por um número, o banco de dados padrão é o número 0. O número de bases de dados pode ser configurado, mas o padrão é 16 bancos de dados. Basicamente, um banco de dados Redis é um

dicionário de pares de chave e valor. No entanto, para além da estrutura de valor chave clássica, onde o valor é uma *string* e os usuários são responsáveis por analisá-la a nível de aplicação, o Redis oferece mais opções de estruturas de dados, onde valores e estruturas de dados podem ser armazenadas como:

- **String** – estrutura mais básica e padrão de armazenamento. Ex.: nomes, lugares, identificadores, variáveis de sessão e cache.
- **Lista de strings**: Suporta inserções no início ou fim de uma lista. Além disso, é capaz de realizar consultas em alta velocidade.
- **Um conjunto de strings**: Uma coleção não duplicada de *Strings,* ao invés de adicionar a mesma sequência repetidamente produz apenas uma única cópia. As operações de adição e remoção levam um tempo constante para serem executadas (O (1)).
- **Um conjunto ordenado de strings**: Semelhante à definição anterior, mas em um conjunto ordenado, cada String está associada com um valor/resultado especificado. Esta contagem é usada como critérios de classificação e pode ser a mesma entre os vários membros do conjunto.
- **Um hash**: Neste caso, cada valor em si é um mapa de campos e valores. Este tipo de dados é muito útil para representar objetos. Ex.: um objeto “Aluno” terá vários campos, por exemplo, nome, idade e email.

2.2.2.2 Consultas

Cada tipo de estruturas de dados tem seu próprio conjunto de comandos disponíveis (Seguin, 2014). O Redis não suporta índice secundário e todas as consultas baseiam-se nas chaves, isso significa que, por exemplo, é impossível fazer uma consulta por alunos que estejam na idade de 20 anos (isto é, estudantes cujo valor do campo idade é 20). Existem estratégias que resolvem esse tipo de limitação, mas que são ineretes as características principais deste tipo de armazenamento de dados.

2.2.2.3 Persistência

Para atingir alto desempenho, bancos Redis são estabelecidos em memória. No entanto, uma desvantagem óbvia é que isso faz com que ele dependa fortemente da disponibilidade de memória, limitando assim a capacidade de armazenamento de dados. Por outro lado, o Redis também tem a capacidade de amazenamento em disco, assim o conjunto de dados pode ser recarregado na memória na inicialização do servidor e posteriormente armazenada no sistema de arquivos. A metodologia de processamento e clusterização apresentada neste trabalho utiliza esta estrutura de armazenamento híbrida (memória-disco) garantindo alto desempenho para a solução.

2.2.2.4 Escalabilidade

Bancos de dados Redis podem ser replicados usando o modelo mestre-escravo. No entanto, ele não suporta *failover* automático, o que significa que se houverem falhas no mestre, um escravo tem que ser promovido manualmente para substituí-lo. Um escravo pode ter outros escravos, por isso também pode aceitar solicitações de gravação, haja vsta que por padrão um escravo está em modo somente leitura. Nas implementações deste trabalho, todos os escravos tinham permissões de leitura e escrita.

2.2.2.5 Conclusão

Por tudo que foi apresentado nas subseções anteriores, o banco de dados não relacional RedisDB se mostrou o canditato ideral para a implementação da solução proposta neste trabalho: o StormSOM. É importante ser esclarecido que qualquer outro banco de dados que siga as características e requisitos apresentados seria viável para a implementação. Com o uso do Redis, posicionado e implementado de forma estratégica, foi possivel agregar desempenho para o processamento de fluxos de dados, garantindo respostas com baixa latência.

Nas próximas subseções, serão abordados os conceitos de clusterização de dados e ferramentas de processamento em tempo real.

2.3 APRENDIZAGEM DE MÁQUINA

A aprendizagem automática ou aprendizado de máquina é um sub-campo da inteligência artificial dedicado ao desenvolvimento de algoritmos e técnicas que permitam ao computador aprender, isto é, que permitam ao computador aperfeiçoar seu desempenho em alguma tarefa. Enquanto que na inteligência artificial existem dois tipos de raciocínio - o indutivo, que extrai regras e padrões de grandes conjuntos de dados, e o dedutivo - o aprendizado de máquina só se preocupa com o indutivo. Algumas partes da aprendizagem automática estão intimamente ligadas à mineração de dados e estatística e sua pesquisa foca nas propriedades dos métodos estatísticos, assim como sua complexidade computacional.

Neste trabalho o foco é sobre o agrupamento de dados através de suas características e similaridades. Nas próximas sub-seções serão apresentados esses conceitos e de que forma foram aplicados na solução proposta.

2.3.1 Clusterização de Dados

O termo ‘Análise de Agrupamentos’, também conhecido por clusterização, está relacionado a diferentes algoritmos de classificação, todos com o objetivo de organizar (categorizar) dados em estruturas (grupos) que façam sentido, sendo estas estruturas baseadas nas características comuns de cada informação. Para a formação dos grupos (clusters) são utilizadas técnicas estatísticas multivariadas, com conotação exploratória, que verificam a similaridade dos objetos, através de coeficientes específicos para cada tipo de variável. É importante destacar que o clusterizador não possui informações a priori dos grupos em que os dados devem ser categorizados. Esta é a diferença entre clusterização e classificação: na classificação as regras que definem os grupos são conhecidas a priori.

Segundo (Pakhira, 2014), o resultado obtido a partir da clusterização é um conjunto de grupos com coesão interna e isolamento externo, ou seja, elementos dentro de um mesmo grupo são tão similares quanto possível e são, ao mesmo tempo, tão dissimilares quanto possível dos elementos presentes nos demais grupos.

2.3.2 Redes Neurais Artificiais

O método de redes neurais possui uma maior capacidade para a descoberta deconhecimento em bases de dados, porém sua implementação é mais complexa. Visto que é baseado em aprendizagem não supervisionada, a rede é alimentada sem que haja comparações com modelos pré-definidos, tendo como base a competição entre os elementos. Este método mostra-se adequado para ser aplicado na tarefa de clusterização, pois possui a propriedade de utilizar a descrição de dados, sendo utilizado paraanálise de dados que possuem relações desconhecidas (Simon, 2014).

2.3.2.1 Mapas auto-organizáveis

Self-Organizing Maps (SOM), ou rede de Kohonen (Kohonen, 2001), faz parte de um grupo específico de redes neurais utilizadas para a aprendizagem não supervisionada. Considerando-se a analogia com redes neurais biológicas, os diferentes grupos/ clusters, que por sua vez, representam padrões diferentes, são modelados em termos de neurónios. Em contraste com o neurônio tradicional encontrado em redes neurais artificiais (RNAs) para aprendizado supervisionado – perceptron – o SOM diferencia-se em sua definição pela forma que os pesos sinápticos são representados; considerando que, em uma abordagem supervisionada os pesos modelam a amplitude do impacto que cada neurônio tem na sua seguinte e conexão neurônio, SOM os usa como as coordenadas para os centróides dos clusters candidatos. Quando comparado com outros algoritmos de agrupamento, a utilização do SOM fornece as seguintes vantagens:

- Não há necessidade de definir o número esperado de clusters a priori. Ao contrário de algoritmos como k-médias, k-medoides, fuzzy c-means, etc., SOM usa uma grade/mapa de neurônios (Figura 2), que pode representar agrupamentos individuais ou, se for grande o suficiente, irá criar grupos de neurônios semelhantes que juntos representam um padrão de cluster;
- Ele pode identificar grupos com padrões de formas complexas. De uma forma semelhante as RNAs, são bons para a modelagem de funções complexas. O SOM não está limitado a localização de padrões para a construção de *clusters* com formas circulares ou elípticas. Eles podem fornecer um algoritmo baseado

em densidade simplificada quando os seus parâmetros de treinamento são bem ajustados.

Consideremos um vetor de variáveis de entrada X = [x_1 , x_2 , x_3 , ..., x_k], e um mapa de neurónios definidas C = [C_1 , C_2 ,..., C_m], onde cada sinapse conecta um xi variável de entrada para um neurónio cj com um peso wij mapeamento. Por sua vez, como cada entrada de xi é ligado a todos os neurónios na rede, cada cj neurónio tem um conjunto de sinapses de entrada e, por conseguinte, um vector de associado de pesos wj = [W_{1j} , W_{2j} , W_{3j} ,..., W_{kj}], que representa um possível centroid cluster.

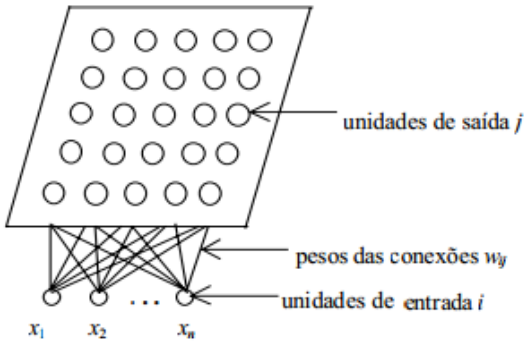

Figura 2 - Modelo de programação do mapa auto-organizável

Durante o passo de formação do algoritmo (Greiner, 2014), cada amostra /tupla S_i, a partir da base de dados é mapeada para um neurónio c_j mais próximo, seguindo a métrica de similaridade aplicada, normalmente a distância euclidiana; isto é, encontrar o neurónio vencedor que minimiza a distância d_{ij}, como se segue:

$$d_{ij}=\sqrt{\left(s_{i1}-w_{1j}\right)^2+\left(s_{i2}-w_{2j}\right)^2+\cdots+\left(s_{ik}-w_{kj}\right)^2} \qquad (1)$$

Após o treinamento, para ajustar os clusters, os pesos w_j do neurônio vencedor e seus vizinhos (neurônios com pesos semelhantes) são atualizados de acordo com (2). Assim, deslocando o neurônio para o padrão avaliado e, a cada nova atualização e iteração, os conjuntos de centróides estarão sintonizados.

$$w_{ij} = w_{ij} + \eta h_j[s_i - w_j] \qquad (2)$$

Onde η representa a taxa de aprendizagem, ou seja, a amplitude do ajuste do peso em relação a amostra; h_j é um valor binário que indica se o neurônio dado é vizinho do neurônio vencedor (h_j = 1) que também deve ser atualizado.

Como as vantagens e aplicabilidades do SOMs foram brevemente destacadas e podem ser amplamente visto na literatura, neste trabalho não avaliaremos os clusters obtidos como resultado do uso do SOM. Vamos nos concentrar especificamente sobre a análise quantitativa e desempenho da abordagem proposta, como um novo processo para implementar um cluster SOM sobre BigData, em computação de fluxo de dados em tempo real.

2.4 PROCESSAMENTO DE DADOS EM TEMPO REAL

Primeiramente, é importante esclarecer que o objetivo de um sistema de processamento em tempo real é dado uma quantidade massiva de dados, obter uma resposta eficiente com baixa latência de resposta. Processamento de dados em tempo real é geralmente chamado de "computação de fluxo". Um fluxo é uma fonte ilimitada de eventos. O objetivo de um sistema de computação de fluxo é fazer a transformação desses fluxos de dados em tempo real e para tornar os dados processados diretamentedisponíveis para o usuário final.

Para a construção deste trabalho foi necessário fazer um levantamento completo das principais ferramentas para processamento de fluxos de dados, analisando-se diversos pontos como arquitetura, escalabilidade, flexibilidade, curva de aprendizagem, implementação, dentre outros parâmetros inerentes ao processamento de dados. Na próxima subseção serão descritos os frameworks que mais se destacaram neste contexto, dentre eles, será apresentado o Apache Storm, que foi o framework selecionado para ser a base arquitetural de processamento deste trabalho.

2.4.1 Medusa

O Medusa é um sistema de processamento de fluxo distribuído construído a partir do Aurora (Bifet, 2010) como um mecanismo único de processamento. Seu

sistema de consultas pode ser distribuido em vários nós de processamento. Esses nós podem estar sob o controle de uma entidade ou podem ser organizados como um cluster de baixo acoplamento. Dentre suas principais vantagens e características, destacam-se:

- Permite escalabilidade incremental ao longo de vários nós de processamento.
- Possui alta disponibilidade, pois os nós de processamento podem monitorar e assumir funções de outro quando ocorrem falhas.
- Permite a composição do fluxo de diferentes origens, para tirar proveito da distribuição inerente em muitas aplicações de processamento de fluxo, por exemplo, monitoramento de clima, análise financeira, etc.

Em contrapartida a todas as vantagens oferecidas pelo Medusa, está a pouca documentação e a arquitetura complexa que dificulta sua alteração para fins diversos, para ser utilizado tem-se que seguir seus protocolos de utilização, o que o torna tecnologicamente um produto de muita qualidade, mas inviável para o uso neste trabalho.

2.4.2 Borealis

O Borealis é um motor de processamento de fluxo distribuído que herda funcionalidade de processamento de fluxo de núcleo do Aurora (Bifet, 2010) e funcionalidade de comunicação de entrenós do Medusa (Marz, 2014). O projeto Borealis é impulsionado pelas deficiências na utilização dos frameworks Aurora e Medusa, para o desenvolvimento de várias aplicações de computação de fluxos de dados. Dentre suas principais características, o Borealis estende o sistema básico do Aurora com a capacidade de (1) modificar vários dados e atributos de consulta em tempo de execução, e (2) operam de forma distribuída. O Borealis tem um distribuidor de carga baseado em correlação, que distribui a carga de trabalho dinamicamente entre os servidores. O objetivo básico dessa arquitetura é minimizar a latência média de processamento. É um sistema tolerante a falhas, que não perde nenhuma tupla, mas poderá produzir tuplas redundantes caso identifique alguma possível falha no sistema.

Apesar de robusto em sua concepção arquitetural e servir de inspiração para novas ferramentas, o Borealis em si possui uma comunidade de desenvolvedores

fechada, recebendo poucas atualizações e com fraca documentação, não se adequando para o uso neste trabalho.

2.4.3 MapReduce Online

MapReduce on-line propõe uma arquitetura de MapReduce modificada que permite que dados sejam canalizados entre os operadores. Isso amplia o modelo de programação MapReducepara além do processamento em lote e suporta a agregação on-line, que permite aos usuários ver retornos enquanto um *job* (processo único de MapPeduce) ainda está sendo computado (Condie, 2014).

A maior diferença entre o algoritmo MapReduce original e MapReduce on-line é que os dados intermediáriossão materializados não mais em um arquivo local temporário (em cada nó), mas é canalizado entre operadores, preservando as interfaces de programação e modelos de tolerância a falhas ddo framework MapReduceoriginal.

O StormSOM, que será apresentado nos próximos capítulos, utiliza a técnica de MapReduce em sua estrutura base, entretanto o uso do MapReduce online não se adequou para o objetivo final desta proposta devido sua forma de tratar os fluxos não deixando tanta flexibilidade para a inserção de um modelo de clusterização neural.

2.4.4 Apache Drill

Apache Drill é um sistema distribuído para análise iterativa de conjuntos de dados em grande escala, com base no framework Dremel, do Google. Seu principal objetivo é prover uma forma alternativa de processar eficientemente conjuntos volumosos de dados. É eficiente em escalar 10.000 servidores e capaz de processar petabytes de dados, assim como trilhões de registros em segundos (Apache, 2014). ODrillnão se limita em ser uma ferramenta para processar um fluxo de dados em tempo real, seu objetivo se estende em obter respostas de consultas sobre grandes volumes de dados, quase em tempo real.

O framework é composto essencialmente por três partes (Apache, 2014):

- **Usuário**: Permitea interação direta com interfaces, como a interface de linha de comando (CLI) e uma interface REST ou permite a aplicação interagir diretamente com o núcleo do Drill.
- **Processamento**: composta pelo núcleo do framework Drille linguagens de consulta.
- **Fonte de Dados**: configuração da fonte dos conjuntos de dados que serão processados.

O fluxo de execução básico do Apache Drillé o seguinte: um usuário gera uma consulta que será analisada pelo verificador de consulta. Em seguida, esta é convertida num plano lógico que descreve o fluxo de dados de base em uma operação de consulta. O plano lógico é convertido em um plano físico (execução), através de um otimizador e um mecanismo responsável pela execução do plano.

O Apache Drill é uma solução completa, porém com uma alta curva de aprendizagem e uma arquitetura complexa de ser modificada, o que foi impeditivo para seu uso neste trabalho. Apesar disso, sua arquitetura de consulta foi inspiração para a construção da topologia de processamento do StormSOM.

2.4.5 IBM InfoSphere

IBM InfoSphere é um produto comercial que consiste em um conjunto de ferramentas para integração e gestão de informações (IBM, 2013). As principais ferramentas incluem:

- **IBM InfoSphereInformation**: Servidor de integração de dados.
- **IBM InfoSphereStreams**: é uma plataforma de computação avançada que permite que aplicativos coletem, analisem e correlacionem informações a partir de milhares de fontes de dados em tempo real. A solução pode lidar com taxas de transferência de dados muito elevadas.
- **IBM InfoSphereBigInsights**: é uma ferramenta executada sobre o Hadoop. Ele simplifica o uso do Hadoop através do *Big SQL*, uma interface SQL para Hadoop.
- **IBM InfoSphereWarehouse**: é uma plataforma de armazenamento e análise de dados baseado em IBM DB2, um banco de dados relacional escalável.

A IBM oferece no InfoSphere uma solução totalmente integrada que contém um data warehouse clássico, uma plataforma Hadoop e um mecanismo de processamento de fluxo. Apesar das diversas vantagens é uma ferramenta fechada impossibilitando sua adaptação dos outros fins que estejam além de seu escopo inicial.

2.4.6 Yahoo S4

O Yahoo S4 é uma ferramenta para processamento de fluxos em tempo real. Foi inicialmente lançado pela Yahoo! em outubro de 2010 e é um projeto *Apache Incubator* desde setembro de 2011, portanto, está licenciado sob a licença de código-aberto Apache 2.0.

O S4 é de propósito geral, para processamento em baixa latência, distribuído, orientado a eventos, descentralizado, escalável e modular que permite aos programadores uma fácil implementação de aplicativos para o processamento de fluxos contínuos e ilimitados de dados. O fluxo de trabalho do Yahoo S4 é: os eventos de dados com chaves são encaminhados para os*ProcessingElements* (PEs), que consomem os eventos e fazem uma das seguintes ações: (1) emitir um ou mais eventos que podem ser consumidos por outros PES, (2) publicar os resultados. O S4 é inspirado no modelo MapReduce projetado para atuar em problemas reais dentro do contexto de aplicativos de busca que utilizam algoritmos de mineração de dados e aprendizado de máquina.

O S4 é um candidato perfeito para a evolução do trabalho que será apresentado nesta dissertação, possui boa documentação e comunidade ativa.

2.4.7 Spark

O Spark é um ambiente de computação em cluster, que possui código aberto, assim como o Hadoop (J. Kornacker, 2012), mas com algumas diferenças que o tornam superior em determinados cenários de cargas de trabalho principalmente por trabalhar diretamente com o processamento de fluxos de dados em tempo real, ou seja, o Spark permite distribuir em memória os conjuntos de dados, otimizando assim, as cargas de trabalho e consultas iterativas. É implementado na linguagem Scala e a

utiliza como framework de aplicação, proporcionando grande facilidade na manipulação de conjuntos de dados distribuídos.

Foi desenvolvido na Universidade da Califórnia, em Berkeley, e seu principal objetivo inicial era servir de base para a construção de sistemas de análise de dados em grande escala e com baixa latência. O framework é uma adição interessante para a crescente família de soluções de análise de grande volume de dados e fornece não só um conjunto de recursos para o processamento de dados distribuídos, mas possui uma pilha tecnológica completa contanto inclusive com um microframewrok de aprendizagem de máquina chamado Mlib. A figura 3 mostra como a pilha tecnológica do Spark está estruturada.

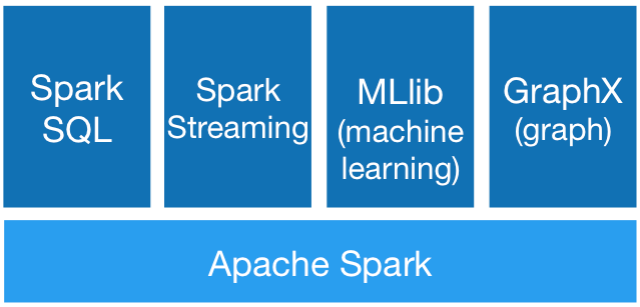

Figura 3 - Arquitetura da pilha de componentes do spark framework

2.4.8 Apache Storm

É o framework utilizado neste trabalho, e será apresentado em sua forma geral nas próximas subseções, destacando as principais vantagens de sua utilização e a motivação de criar uma nova metodologia para aprendizagem de máquina sobre sua arquitetura.

2.4.8.1 Introdução

O Apache Storm é um sistema de processamento de fluxo de dados distribuído e em tempo real com tolerância a falhas, o que garante o processamento eficiente de um fluxo contínuo e ilimitado de dados. Cada uma dessas palavras-chave serão utilizadas no modelo arquitetural proposto neste trabalho. O Apache Storm é um projeto *open-source* licenciado sob a licença EPL – Eclipse Public License (Simon,

2014). A EPL é uma licença muito permissiva, possibilitando que o projeto seja utilizado para qualquer fim, sendo proprietário ou *open-source*.

Com a capacidade de processar fluxos ilimitados de dados de maneira facilitada e confiável, o Apache Storm faz para o processamento em tempo real o que Hadoop fez para processamento em lote, além de possuir uma arquitetura simplificada, podendo ser usado com qualquer linguagem de programação (Marz, 2014).

O framework implementa um modelo no qual os dados fluem continuamente através de uma rede de estruturas de transformação. A abstração de um fluxo de dados é chamada de *Stream* que é uma sequência ilimitada de tuplas. Uma tupla é uma estrutura que pode representar tipos padronizados de dados (como inteiros, pontos flutuantes, array de bytes) ou tipos definidos pelo usuário através do uso de algum código de serialização. Cada fluxo é definido por uma identificação única que pode ser usada para construir topologias de fontes de dados. Os *streams* originam-se de estruturas chamadas de *Spouts.* Aestrutura responsável por consumir e produzir os dados do fluxo é chamado de *Bolt*.

Destacando esses componentes, o Apache Storm, possui a seguinte estrutura:

- **Tuplas**: Conjunto unitário de elementos de dados que serão processados.
- **Streams**: representa o fluxo das informações, sendo basicamente, uma sequência de tuplas.
- **Spouts**: fonte de Streams que serão processadas.
- **Bolts**: Unidades de processamento, podendo executar funções, junções, agregações ou comunicação com outras fontes de dados, de acordo com sua programação.
- **Topologias**: consiste na modelagem visual da rede de Spouts e Bolts.

O modelo conceitual desta topologia de processamento de fluxo de dados do ApacheStorm é apresentado na Figura 4.

Uma topologia é distribuída entre processos de trabalho, onde para cada componente - um *Spout*ou *Bolt* - é atribuído um número definido de tarefas a serem distribuídos entre os processos de trabalho.

O framework Storm possui uma extensão chamada de Trident que consiste em ser uma abstração de alto nível, permitindo a execução de consultas, de baixa latência, durante o processamento do fluxo (Marz, 2014). Possui um funcionamento semelhante ao Apache Pig (Apache, 2013) e ao Cascading (Nathan, 2013), com recursos para fazer junções, agregações, agrupamentos, funções e filtros no fluxo de dados. O Trident é um recurso muito utilizado neste trabalho para monitorar o agrupamento dos dados nos nós de processamento.

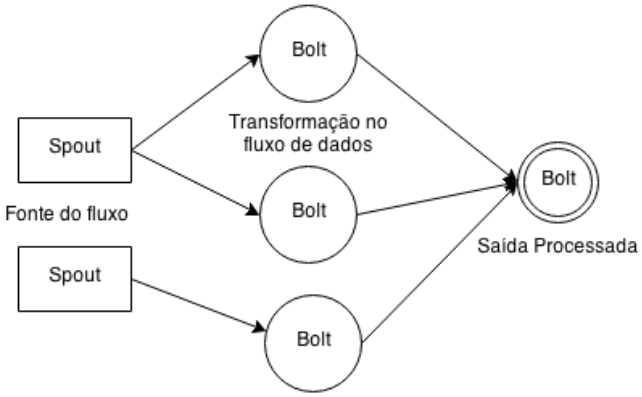

Figura 4 - Modelo de programação baseado em topologias

Atualmente, o Apache Storm é utilizado como solução para o processamento de fluxo de BigData por grandes empresas que se destacam por investimentos em pesquisas de inovação tecnológica.

2.4.8.2 Componentes de um Cluster Storm

Um cluster Storm pode ser comparado a um cluster Hadoop, considerando que, em Hadoop são executados "*Works MapReduce*", e em Storm executam-se "Topologias". A principal diferença entre "work" e "Topologias" é que um *work MapReduce* eventualmente processa um conjunto de dados até um ponto pré-definido, ao passo que uma topologia processa infinitamente e em tempo real (Joey, 1997).

Existem dois tipos de nós em um cluster Storm, o mestre e os nós operadores. O nó mestre executa um *daemon* chamado "Nimbus". O Nimbus pode ser comparado ao *JobTracker* do Hadoop e é responsável pela distribuição de código em torno do cluster, a atribuição de tarefas às máquinas e monitoramento de falhas.

Cada nó operadorexecuta um *daemon* chamado "Supervisor". O *supervisor* gerencia os processos que serão executados, com base no que Nimbusatribui a ele. Cada processo de trabalho executa um subconjunto de uma topologia. Uma topologia em execução consiste de muitos processos de trabalho distribuídos em várias máquinas.Toda coordenação entre os supervisores e o Nimbus é feita através do Zookeeper. Zookeeper é um serviço centralizado para manter as informações de configuração, fornecendo serviços de sincronização e de grupos distribuídos.

2.4.8.3 Agrupamento de Fluxos

Para introduzir a noção de agrupamentode fluxos de dados, vamos exemplificar sua aplicação: poderíamos imaginar que um nó envia tuplas com os seguintes campos: timestamp, webpage. Esta poderia ser uma seqüência de cliques simples emuma página web. O próximo nó é responsável por contar os cliques por página. Por conseguinte, é obrigatório que as mesmas páginas sejam encaminhadas para a mesma tarefa. Caso contrário, teremos de mesclar os resultados em um novo processo e este não é o objetivo. Portanto, precisamos agrupar o fluxo pelo campo ‘webpage’.

Os diferentes tipos de agrupamento de fluxos encontram-se descritos na Tabela 2. Além desses, também é possível implementarumagrupamento fluxo personalizado através da interface de implementação chamada "CustomStreamGrouping".

Agrupamento	Descrição
Shufflegrouping	Tuplas são distribuídos aleatoriamente em tarefas do nó de uma forma tal que cada nó está garantido para obter um número igual de tuplas.
Fieldsgrouping	O fluxo é repartido pelos campos especificados no agrupamento.
Allgrouping	O fluxo é replicado em todas as tarefas do nó.
Global grouping	Todo o fluxo vai para um único de tarefas do nó. Especificamente, ele vai para a tarefa com o menor id.
Nonegrouping	Este agrupamento especifica que não há uma política explícita de agrupamento.

Directgrouping	Este é um tipo especial de agrupamento. Um fluxo agrupado com este padrão diz que o produtor da tupla decide qual tarefa o consumidor receberá e executará nesta. Esse agrupametosó pode ser declarado em fluxos diretos.
Local grouping	Se o nó de destino tem um ou mais tarefas no mesmo processo de trabalho, tuplas serão embaralhadas.Caso contrário, essa age como um agrupamento normal do tipo*shufflegrouping*.

Tabela2– Lista de agrupamentos de fluxos de dados do Apache Storm

2.4.8.4 Tolerância a Falhas

A Tolerância a falhas é uma propriedade que permite que um sistema continue funcionamento corretamente mesmo diante da falha de alguns dos seus componentes (Abd-El-Barr, 2013). Especialmente no caso de sistemas distribuídos, onde grandes aglomerados podem ser compostos por centenas ou mesmo milhares de nós, é essencial que o sistema possa lidar com falhas. A lista de coisas que podem dar errado é quase ilimitada. Podem ser problemas de hardware, por exemplo, um disco rígido pode ter quebrado ou o ventilador não funciona mais, etc. Os problemas também podem vir da rede, o software, código de utilizador, etc.

O Storm é equipado de rotinas rápidas e completas de tolerância à falhas, o que significa que os processos irão parar sempre que um erro inesperado for encontrado. Foi concebido de modo que possa interromper seus processos de forma segura, a qualquer momento, e recuperar corretamente quando o processo é reiniciado.

2.4.8.5 Topologia Transacional

O Storm garante que uma tupla em um nó específico será totalmente processada, mas apenas uma única vez.Vamos supor que, eventualmente, apenas uma parte da tuplafoi processada. Logo, esta entrará em um processo de repetição até que seja completamente computada pelo nó de processamento (tolerância à falhas). Esta situação pode ser um problema se, por exemplo, quisermos contar as ocorrências de palavras em uma frase. Algumas palavras serão contadas mais de

uma vez. A topologia transacional do Stormgarante que as tuplas serão processadasapenas uma vez.

2.4.8.6 Interface de Gerenciamento

O StormUI é uma interface web que mostra informações sobre as topologias do cluster de funcionamento da Apache Storm. É utilizado para analisar o desempenho das topologias e o processamento do fluxo de dados, assim como a detecção de falhas no processamento.

2.5 O PARADIGMA DE APRENDIZAGEM DE MÁQUINA DISTRIBUÍDA SOBRE FLUXO DE DADOS

Estruturas de aprendizagem de máquina de fluxos de dados, são incapazes de escalar o processamento para atender a um fluxo intenso, contínuo e infinito de dados. A fim de resolver este problema de escalabilidade, há diversas soluções para torná-los distribuídos e para fazer chamados os algoritmos on-line, serem executados em paralelo. Este modelo de processamento será referenciado neste trabalho como DMLDS – *Distributed Machine Learning on Data Stream* – e sua concepção parte do princípio da junção de outras áreas do conhecimento, como apresentado na Figura 5.

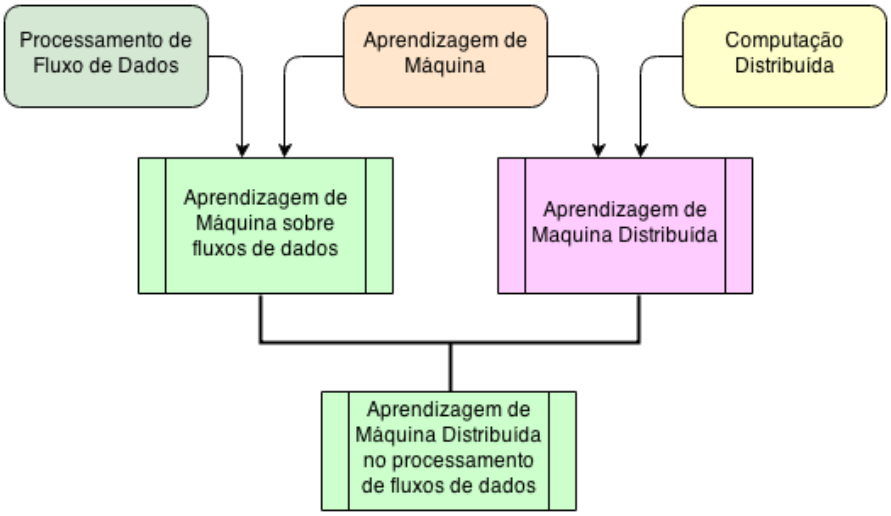

Figura 5 - Aglomeração dos conceitos para o surgimento do paradigma de Aprendizagem de Máquina Distribuída no processamento de fluxos de dados

Os principais frameworks para a aprendizagem de máquina distribuída sobre fluxos de dados, que foram relevantes em sua contribuição, serão apresentados na próxima seção.

3 TRABALHOS CORRELATOS

A concepção do StormSOM foi pautada pelas necessidades identificadas tanto no mercado de soluções quanto na literatura científica em geral, onde foi detectada a carência de soluções que envolvessem o processamento, em tempo real, tolerante a falhas, de volumes distribuídos de dados juntamente com a utilização de um modelo neural de clusterização. Este capítulo apresentará os trabalhos correlatos ao StormSOM e de que forma a solução proposta esta posicionada em relação às demais metodologias.

3.1 FRAMEWORKS PARA APRENDIZAGEM DE MÁQUINA SOBRE FLUXO DE DADOS

Um dos principais frameworks para MLDS existentes é o Massive Online Analysis (MOA) (Marz, 2014). O MOA consiste em um conjunto de algoritmos on-line para: classificação, clusterização e detecção de padrões que foram adaptados para trabalhar com fluxos de dados. Contém ferramentas de visualização de informações de clusters de dados.

O Vowpal Wabbit (Ran, 2014) é outro exemplo de estrutura de MLDS, e utiliza o algoritmo perceptron como base. Os autores otimizaram o VowpalWabbit para trabalhar com a entrada de dados em formato texto, além disso, este framework é composto por vários algoritmos online, tais como gradiente descendente estocástico e os métodos de sub-gradiente adaptativos, além de fornecer algoritmos de aprendizagem linear.

O Debellor (Wojnarski, 2008) utiliza uma arquitetura baseada em componentes, ou seja, consiste de uma rede de componentes, onde cada componente se comunica com outros através de dados transmitidos. Essa arquitetura permite que os desenvolvedores de algoritmos para ML, possam estender facilmente a estrutura do Debellor, adaptando-o em uma implementação alternativa. Os autores afirmam que o Debellor é "escalável" em termos do tamanho da entrada (Wojnarski, 2008) ou seja, é capaz de lidar com o aumento no tamanho dos dados de entrada, isso significa dizer que o Debellor suporta escalabilidade vertical. No entanto, ele não suporta escalabilidade horizontal, uma vez que só trabalha sobre um único nó de processamento.

Outra direção para mineração de fluxos de dados é utilizar motores de processamento de fluxo (Stonebraker, 2005) (SPE – *StreamProcessingEngine*) existentes no mercado para a implementação de algoritmos online. A SPE é um mecanismo de cálculo especializado em processar grande volume de fluxo de dados de entrada, com baixa latência. Stonebraker (Abadi, 2003) define uma SPE como um sistema que se funciona ao estilo SQL sobre um fluxo de dados, mas sem armazenar dados.

Há SPEs de código aberto, como Aurora (Abadi, 2003), o Medusa, o STREAM, o Borealis. E há também SPEs comerciais, como IBM InfoSphereStreams (IBM Redbooks, 2014), Microsoft StreamInsight (Siegel, 2014), (Andrade, 2014) e StreamBase. As SPEs estão evoluindo e tendem a utilizar modelos de programação como o Map-Reduce em vez de modelos de programação estilo SQL para processar fluxos de dados. Exemplos proeminentes de SPE que estão em destaque no mercado e representam o estado-da-arte são: S4 (Neumeyer, 2014) e Apache Storm, framework SPE utilizado neste trabalho.

O projeto StormPattern (Goetz, 2014) é uma adaptação dos modelos para aprendizagem de máquina para o processamento em fluxo de dados, utilizando uma abstração do Storm, chamada de Trident. Na versão atual, o StormPattern não permite aprendizagem on-line, só permite pontuação on-line usando o modelo PMML (Predictive Model Markup Language) (Zhu, 2010). StormPattern suporta modelo PMML para estes seguintes algoritmos: randomforest, regressão linear, agrupamento hierárquico e regressão logística.

Uma das adaptações mais relevantes que temos disponível no mercado atualmente é o projeto Trident-ML (Nalya, 2014). Consiste em uma biblioteca on-line para aprendizagem de máquina em tempo real, construído sobre o Apache Storm. Semelhante ao StormPattern, o Trident-ML também utiliza a abstração Trident na sua implementação e suporta algoritmos on-line para ML e análises estatísticas como: classificação linear, regressão linear, k-means, padronização e normalização, média e variância.

Apesar de apresentarem grandes contribuições no âmbito do processamento de fluxos de dados e serem ideais para certos tipos de cenários e aplicações, uma das características dos sistemas citados, nesta seção, é a execução sequencial dos algoritmos sobre uma única instância (máquina), processamento conhecido como

single-node, o que ocasiona uma escalabilidade limitada em relação a um fluxo de grande volume de dados.

O StormSOM não possue limitações quanto ao processamento, haja vista que é escalável sobre uma arquitetura em cluster para prover distribuição de processamento, execução paralela e tolerância a falhas.

Na próxima seção serão exploradas soluções distribuídas para a mineração de fluxos de grande volume de dados.

3.2 FRAMEWORKS PARA APRENDIZAGEM DE MÁQUINA DISTRIBUÍDA SOBRE FLUXO DE DADOS

Em termos de estruturas, identificamos alguns *frameworks* que se destacam e pertencem a categoria de DMLDS, são eles: JubatusAcinonyx, StormMOA, SAMOA.

O Jubatus (Tokui, 2014) é um exemplo de framework para DMLDS que possui uma biblioteca para algoritmos para ML que operam de forma distribuída para regressão, classificação, recomendação e detecção de anomalias. Ele introduz o conceito de modelo de aprendizagem de máquina local, o que significa que podem haver vários modelos em execução ao simultaneamente processando diferentes conjuntos de dados. Usando esta técnica, o Jubatus alcança escalabilidade horizontal via paralelismo horizontal e particionamento de dados, para isso, o Jubatus adapta o paradigma de MapReduce em três conceitos fundamentais:

- *Update*: processa um dado, atualizando o modelo local.
- *Analyze*: processa um dado através da aplicação do modelo local e obtém o resultado, tal como a qual classe pertence em uma operação de classificação.
- *Mix*: é a junção do modelo local em um modelo misto que será usado para atualizar todos os modelos locais do Jubatus.

O Jubatus estabelece forte ligação entre a implementação dos algoritmos para aprendizagem de máquina e alguma SPE distribuída subjacente. A razão disso é para que os desenvolvedores que utilizam o Jubatus possam construir e implementar suas próprias SPEs distribuídas de forma personalizada. O Jubatus alcança a tolerância a falhas usandoumcomponente chamado *jubakeeper* (Pop, 2014) que utiliza ZooKeeper (Junqueira, 2014) como inspiração.

O StormMOA (Bifet, 2011) é um projeto que combina o MOA (Bifet, 2010) com o Apache Storm - framework utilizado neste trabalho - para satisfazer a necessidade de implementação escalável dos algoritmos de ML para lidar com um fluxo ilimitado de dados. Os desenvolvedores do StormMOA aplicam duas variantes de algoritmos de ML:

Implementação em memória, onde StormMOA armazena o modelo apenas na memória e não o persiste em um armazenamento local. Esta implementação precisa reconstruir o modelo desde o início quando alguma falha acontece.
Implementação com tolerância a falhas, onde StormMOA persiste modelo, portanto, é capaz de recuperá-lo em caso de falha.

Assim como o Jubatus, o StormMOA também estabelece forte ligação entre MOA e o Apache Storm, este acoplamento impede a extensão do StormMOA para a utilização com outras SPEs.

Por fim, analisamos o SAMOA, um framework para aprendizagem e máquina distribuída com alta abstração. O SAMOA permite o desenvolvimento de novos algoritmos de ML sem lidar com a complexidade dos mecanismos subjacentes de processamento de fluxo, tais como Apache Storm e Apache S4.

Nesta seção apresentamos algumas soluções para o processamento e utilização de algoritmos de aprendizagem de máquina sobre volumes massivos de dados, tanto em lote quanto sobre fluxo de dados.

Neste trabalho optamos por não utilizar nenhum dos frameworks de ML disponíveis no mercado, primeiramente porque estes frameworks ainda estão em processo de amadurecimento como já foi mostrado anteriormente, outro ponto relevante é que nenhum dos produtos apresentados possui uma solução neural para a clusterização de grandes volumes de fluxos de dados contínuos e distribuídos. Faz parte do escopo deste trabalho, propor uma arquitetura robusta e eficiente para esta problemática utilizando mapas auto-organizáveis como modelo neural de clusterização dos fluxos de dados.

4 CLUSTERIZAÇÃO DE FLUXOS DE DADOS DISTRIBUÍDOS EM TEMPO REAL

Nesta seção serão descritos os detalhes da solução proposta, sua visão conceitual, arquitetural e implementação.

4.1 STORMSOM - CLUSTERIZAÇÃO EM TEMPO-REAL DE FLUXOS DE DADOS DISTRIBUÍDOS NO CONTEXTO DE BIGDATA

Neste trabalho propomos um algoritmo que chamamos de StormSOM. O nome é proveniente da combinação de Apache Storm (SPE base da solução) e SOM – *self-organization map* – rede neural para a clusterização de dados, apresentado no Capítulo 2, subseção 2.3.2.1.

Antes de iniciarmos a apresentação do modelo em termos específicos, é importante destacar os motivos que nos leravam a escolhermos o SOM como modelo de clusterização. Como já foi citado anteriormente, grande parte dos modelos computacionais foram desenvolvidos para propósito geral e não estão prontos para serem utilizados sobre grandes volumes de dados retornando respostas em tempo real. Em grande parte, dependem de processos iterativos que levam tempo para serem finalizados. Então, neste contexto, o primeiro desafio para a composição da proposta foi adaptar o modelo neural de clusterização para que seja capaz de ser aplicado em cenários de fluxos contínuos e ilimitados de dados. O SOM se destacou pela facilidade de adaptar seu modelo de processamento para interagir com a arquitetura baseada em topologia exposta pelo Apache Storm. Além disso, esta abordagem possui um alto grau de inovação, haja vista que em nenhuma das soluções disponíveis, até o momento, no mercado, aplicam mapas auto-organizáveis para a clusterização de fluxos de dados com respostas em tempo real.

O StormSOM foi construído para ser uma solução genérica, podendo ser aplicada, em teoria, sobre qualquer domínio de dados, outro ponto relevante é que sua concepção foi pensada para que o algoritmo seja executado em um ambiente distribuído de *cloudcomputing,* haja vista que todos os experimentos foram feitos utilizando esse tipo de ambiente. As tuplas de dados consumidas podem ser de naturezas diversas e as fontes de dados (*Spouts*) podem variar de acordo com a necessidade do domínio de dados em questão.

Construir uma extensão de um SPE pode ser um grande desafio, pois é necessário lidar, não apenas com seus recursos básicos, mas também com as utilizações mais complexas. Neste trabalho, um grande desafio foi lidar com o processamento distribuído, ou seja, além de adaptarmos o algoritmo de clusterização para que sua execução seja possível em um ambiente de fluxos contínuos de dados, também temos que garantir que esse processamento possa ser realizado simultaneamente por diversos nós (computação distribuídas) sem que haja perda de integridade nos dados. Para tal, foram feitas adaptações na arquitetura para a utilização do Redis (subseção 2.2.2) de modo que os pesos da rede neural fossem guardados e consultados em tempo real, mantendo todos os nós de processamento atualizados utilizando uma só configuração de rede para a clusterização dos dados.

Nos itens subsequentes desta seção, serão apresentados os detalhes da arquitetura e funcionamento do processo.

4.1.1 Modelagem da Solução

Como já foi apresentado em seções anteriores, o Apache Storm, possui uma estrutura chamada de *Spout*, que representa a fonte do fluxo de dados, o primeiro passo para a modelagem da solução é definir uma fonte de dados contínua. Para tal, utilizamos o banco de dados, e sobre ele usamos uma abstração de estrutura de dados de filas FIFO (first-input/first-output), onde o primeiro que entra é o primeiro a sair, criando assim, uma fila de dados que são produzidos por uma instância (responsável apenas por produzir o fluxo de dados) e consumidos por outra instância (ou conjunto distribuído de instâncias) com o StormSOM em tempo real (Figura 6).

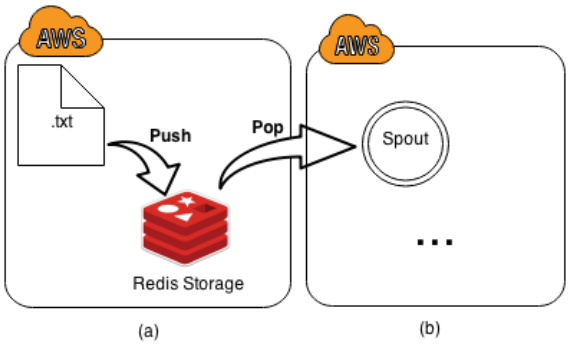

Figura 6 - Arquitetura conceitual da produção e consumo de tuplas pelo StormSOM. Uma instância, lê os arquivos de dados e armazena em fila dentro do Redis (a), por sua vez (b) uma instância StormSOM consome esses dados e inicia o processamento.

O objetivo de separar as instâncias é simular um ambiente de sensores climáticos, que coletariam os dados e os enviariam para um ambiente de *cloudcomputing* a fim de processá-los obtendo-se repostas em tempo real ou com baixa latência (Bifet, 2011).

De posse da compreensão do funcionamento da fonte de dados, agora é necessário entendermos o funcionamento da topologia de processamento do StormSOM. Esta topologia conta com 3 *Bolts* de processamento, sendo eles (foram dadas nomenclaturas para cada *bolt,* a fim de facilitar a especificação da arquitetura):

- ***MissingBolt***: nó de processamento responsável por verificar a existência de dados faltosos. Por exemplo: vamos supor que em um processo de coleta de um sensor, fossem enviados dados com a seguinte tupla: [' ','48.000','9.7810','-2.4740'], onde os valores em cada posição do vetor representariam respectivamente: latitude, longitude, velocidade e direção. Percebemos que a primeira posição do vetor da tupla, correspondente ao valor da latitude, está vazio, ou seja, provavelmente, ocorreu alguma falha na obtenção desta informação, no sensor de coleta de dados. Caso fosse enviado para a clusterização, além de ocasionar uma falha interna no modelo de dados, prejudicaria o processo de clusterização. Lembramos que, neste trabalho, não cuidamos do pré-processamento de dados de modo a corrigir ou eliminar inconsistências, pois a topologia do StormSOM já garante a execução desta tarefa em tempo real, dentro de um fluxo contínuo de dados. O ***MissingBolt***, trata esta tupla, evitando que ela seja enviada para clusterização. Para fins de controle e consistência de dados, esta dupla “defeituosa” é armazenada em um banco Redis para análise, registrando, além das informações originais da tupla (latitude, longitude, velocidade e direção), informações temporais de quando o dado foi coletado e qual fonte de dados foi responsável por ele, possibilitando assim, identificar falhas no sistema de sensores que coletou o dado defeituoso em questão.
- ***CounterBolt***: nó de processamento responsável por realizar uma operação de contagem por frequência, utilizando map-reduce (Tom White,2012). Este nó tem grande importância nesta topologia, pois garante que sejam possíveis outras análises estatísticas que tenham como base a frequência de ocorrência

dos dados. Ele é executado após os dados passarem pelo ***MissingBolt***, de modo que mantenha a consistência no processo de contagem dos dados.

- ***SOMBolt***: nó de processamento responsável pela clusterização dos dados propriamente dita. Obtém a tupla já validada pelo ***MissingBolt***e registrada para contagem no ***CounterBolt*** e inicia o fluxo de clusterização em 3 etapas:

Atualização dos Pesos (etapa inicial): nesta etapa, o algoritmo consulta um banco Redis (que possui grande capacidade para escrita e leitura de dados), a fim de consultar os pesos atualizados do modelo neural. Caso não haja registros de pesos no banco, esses serão gerados randomicamente segundo o fluxo padrão do SOM. Na figura 7, temos 3 nós de processamento do StormSOM distribuídos, consultando a mesma base de pesos Redis, um processo concorrente e assíncrono que garante que a rede neural esteja sempre atualizada.

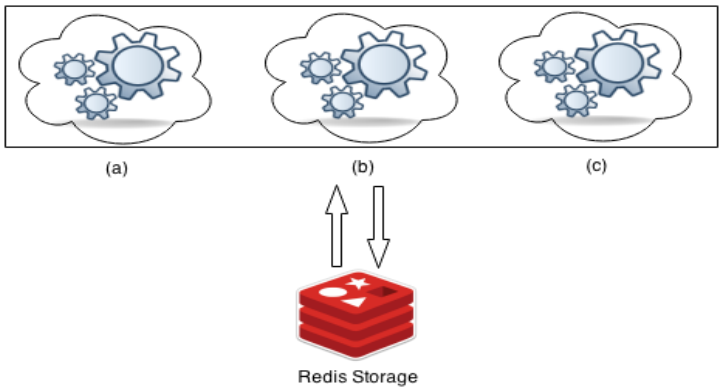

Figura 7 - Os nós de processamento StormSOM: (a), (b) e (c) recuperam a estrutura da rede neural no início de cada etapa de processamento e persistem os novos pesos no fim do processo. Isso é possível haja vista que o Redis apresentagrande capacidade

Execução: nesta etapa, o clusterizador é executado. Na próxima subseção, serão apresentados alguns requisitos inerentes desta etapa como o processo de normalização dos dados e o treinamento da rede neural.

Atualização de Pesos (etapa final): nesta etapa, ocorre a coleta das informações da rede neural, e a persistência desses dados no Redis.

Estas etapas compõe o processo de clusterização pelo SOM sobre um fluxo de dados e em todos os casos estamos avaliando uma tupla de cada vez. Uma analogia interessante é imaginarmos uma base de dados, que será clusterizada, sendo iterada. Esse processo iterativo é feito sobre uma base onde o escopo dos dados já é conhecido. Em nosso caso, esse processo iterativo é substituído pelo fluxo, onde os dados são enviados de forma contínua e ilimitada, onde não se tem noção do volume e qualidade dos dados que serão enviados.

- ***OutBolt****:* por fim, o último nó da topologia é executado. Este *bolt* recebe as tuplas já com a clusterização, representada por um campo a mais que corresponde ao cluster a qual a tupla pertence. Além disso, este *bolt* é responsável por enviar esses dados processados, não somente do **SOMBolt,** mas também dos demais bolts que produzem informações estatísticas, para outras bases de dados externas. Para garantir análises futuras dos dados, OutBolt, poderia armazenar os dados no Apache Cassandra (banco de dados não relacional, escalável, que trabalha muito bem com dados numéricos) e no ElasticSearch (indexador de dados, que armazena e cria índices sobre os dados para consultas rápidas e semânticas). A adição desses dois recursos se dá pela possibilidade de expansão deste trabalho, em direção à uma plataforma de visualização de informação.

4.1.2 Arquitetura da Solução

No Capítulo 2, subseção 2.4.8, apresentamos os conceitos da topologia do Apache Storm. Nesta seção vamos aprofundar esses conceitos aplicados ao StormSOM, de modo a apresentar como o mesmo está distribuído arquiteturalmente e como interage com os demais recursos utilizados neste trabalho (banco de dados e ambiente de simulação). Na Figura 8, destacamos a topologia do StormSOM, onde em (1) temos a fonte do fluxo de dados (Figura 6), em (2) instâncias do *MissingBolt* recebem as tuplas e verificam sua consistência, em (3) o *CounterBolt*, recebe as tuplas já validadas e executa o mapreduce, em (4) o SOMBolt *clusteriza* os dados e em (5) o *Outbolt*, cuida do armazenamento de resultados e os dispõe para análise e suporte à decisão.

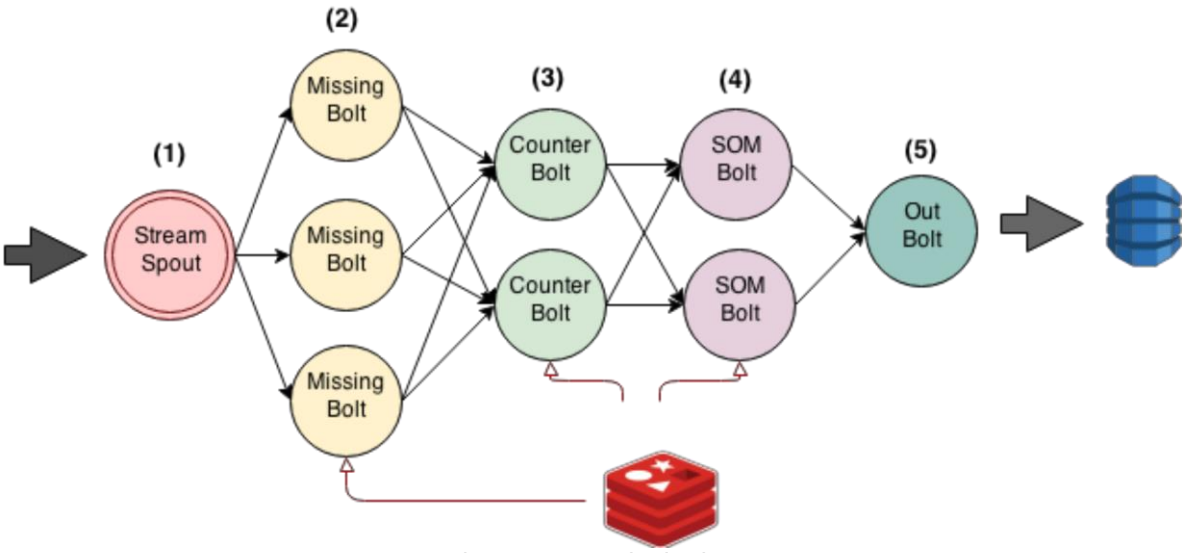

Figura 8 - Topologia do StormSOM

Como pode ser observado na Figura 8, o banco Redis está presente em quase todas as etapas de processamento, pois essas etapas necessitam realizar transições intensivas e o Redis possui a vantagem de ser executado em memória, garantindo assim, um ganho no poder de processamento. Apenas não está relacionado com o último nó - *OutBolt*– pois este já executa outro tipo de armazenamento, enviando os dados já processados para bancos de dados não-relacionais e indexadores, para análise e suporte à decisão.

Como já foi explorado na subseção 2.3.2.1, o algoritmo SOM, é um método não-supervisionado e iterativo, ou seja, existirá um processo de treinamento da rede neural, para que a mesma seja capaz de extrair os padrões do conjunto de dados de entrada. Consideremos, portanto, uma base de 1 bilhão de dados, sendo processados em forma de fluxo contínuo, e a rede neural foi configurada para iterar na fase de treinamento até uma determinada condição de parada ser atingida, que pode ser com a diminuição do erro de treinamento ou um número de épocas pré-definido. Se aplicarmos a forma iterativa tradicional, teríamos que percorrer a mesma base diversas vezes até a condição de parada ser validada, desta forma estaríamos corrompendo o conceito de processamento de fluxo de dados, pois o mesmo não teria resposta em baixa latência (Bifet, 2010). Portando, para adequar a fase de treinamento do algoritmo SOM, de modo que seu treinamento esteja de acordo com

os conceitosdeprocessamento de fluxo contínuo, foi criada uma adaptação da topologia, como apresentado na figura 7.

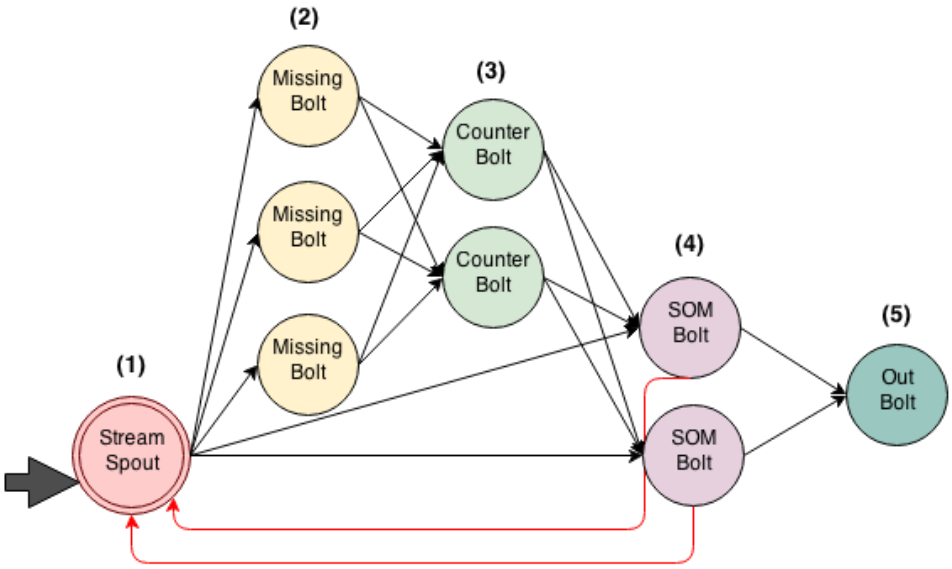

Figura 9 - Topologia do StormSOM adaptada ao processo de treinamento do algoritmo SOM

Desta forma, a topologia do StormSOM adaptada no nó *SOMBolt* para que itere cada tupla, pelo *n* vezes (informação parametrizada), de modo a transformar esta dupla em uma nova entrada, até que sua condição de parada *n* , seja atingida. A tupla, além dos campos de dados, conterá, adicionalmente, os campos: iteração e cluster, onde o campo iteração conterá o valor da iteração corrente e o cluster, conterá a informação do cluster (saída da rede) ao qual a determinada tupla pertence. Esta solução faz uma alteração no processo de treinamento, capaz de adaptá-lo ao processamento de fluxo contínuo de dados sobre bigdata do StormSOM. Ao ser enviado para o *StreamSpout*, a tupla será verificada e caso tenha o campo chamado “iteração”, o mesmo será enviado diretamente para o *SOMBolt*, caso não tenha, seguirá o processo normal a partir do *MissingBolt*, desta forma garantimos que a contabilização da frequência de ocorrência dos dados seja feita de forma adequada, e que este processo de treinamento não venha onerar o processamento da rede com redundâncias desnecessárias.

Na próxima seção, serão apresentados os ambientes de simulação e os resultados dos testes realizados no StormSOM.

5 SIMULAÇÕES E RESULTADOS

Neste capítulo, serão apresentadas as simulações executadas nos ambientes de cloud computing distribuídos sobre a arquitetura de servidores da Amazon Web Services. Com as análises realizadas pretende-se comprovar a eficiência da solução proposta sendo executada em ambientes reais de operação.

5.1 AMBIENTE DE SIMULAÇÃO

As análises foram feitas utilizando servidores na AWS - Amazon Web Services, com o serviço de *Elastic Cloud Compute* (Amazon EC2), que permite ao usuário criar instâncias de máquinas com uma variedade de sistemas operacionais de acordo com cada necessidade. Foram utilizadas instâncias que oferecem equilíbrio de recursos de computação, memória e rede. Deste modo, o ambiente utilizado para as análises segue a configuração:

- Instância: m1.medium – 1,7 GiB de memória, 1 unidade de processamento EC2 (1 núcleo virtual com 1 unidade de processamento EC2), 160 GB de armazenamento de instância local, plataforma de 64 bits. Processadores Intel Xeon.
- Sistema: Amazon Linux AMI 2013.09 (ESB) Linux 3.4;

A versão do Apache Storm utilizada foi a 0.8.1 e o algoritmo StormSOM, proposto neste trabalho, foi implementado sobre a plataforma Java 8 update 20. A decisão de usar a plataforma citada foi pautada, primeiramente, devido a implementação original do Storm ser feita em Java, e a versão utilizada se dá pela maior eficiência no gerenciamento de memória e maior interoperabilidade com com outras plataformas, caso isto venha a ser um fator interessante em implementações futuras nas etapas evolutivas deste projeto. A construção do ambiente de execução, no que concerne à preparação do ambiente, foi feita segundo (Anderson, 2014) e de acordo com as especificações da documentação oficial do Apache Storm.

5.2 ANÁLISES DE RESULTADOS

Nesta seção serão apresentados os resultados provenientes das simulações realizadas no ambiente de *cloud computing*, onde pretendemos mostrar a eficiência do StormSOM para a clusterização de fluxo contínuo de dados em tempo real. O objetivo principal deste trabalho é a análise quantitativa, pois pretendemos mostrar o desempenho da solução diante de diversos cenários, unificados e distribuídos. Em todos os experimentos, foi utilizado um conjunto de 27.280*10^9 registros, enviados em fluxo contínuo e concorrente.

5.2.1 Avaliação Quantitativa

As avaliações quantitativas são o enfoque principal deste trabalho, em detalhes, queremos comprovar que independente da intensidade do fluxo e do volume de dados, o StormSOM é capaz de gerar bons resultados. Entende-se como um bom resultado, um processamento com baixa-latência (Bifet, 2010), ou seja, o tempo entre entrada dos dados na topologia StormSOM e sua respectiva saída, deve ser mínimo e se manter estável, mesmo diante de variações no fluxo de dados. Em todos os experimentos realizados, tínhamos uma instância exclusiva para ser fonte de dados (Figura 3).

5.2.1.1 Processamento Single-node

Este experimento foi realizado utilizando uma única instância (máquina) como nó de processamento StormSOM, nele pretende-se mostrar o comportamento do algoritmo em relação ao processamento do fluxo de dados, em apenas um nó de processamento (Figura 10). Este é um tipo de experimento inicial, que visa apresentar o funcionamento do sistema sem a complexidade do processamento paralelo e distribuído.

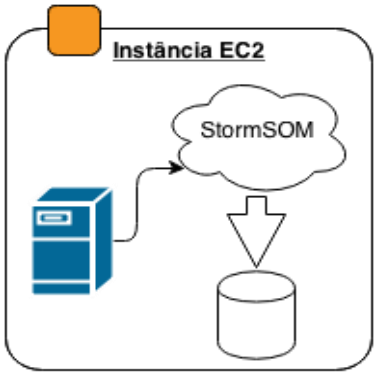

Figura 10 - Organização conceitual do nó de processamento single-node do StormSOM

Avaliamos a relação entre a quantidade de dados envidados no fluxo e o tempo de processamento. Os resultados mostram que o StormSOM se mantém estável, em relação ao volume de dados a serem processados, como apresentado na Figura 9. Além disso, é importante verificar a demanda de processamento na instância, para analisar o consumo dos recursos da máquina em relação ao uso de CPU (figura 10).

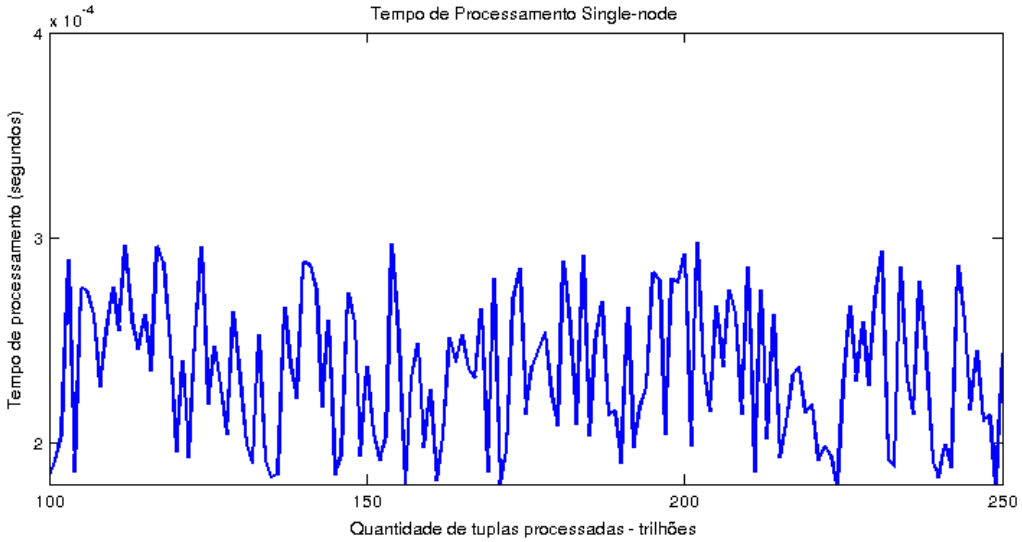

Figura 11 - Tempo de Processamento em um único nó StormSOM (single-node)

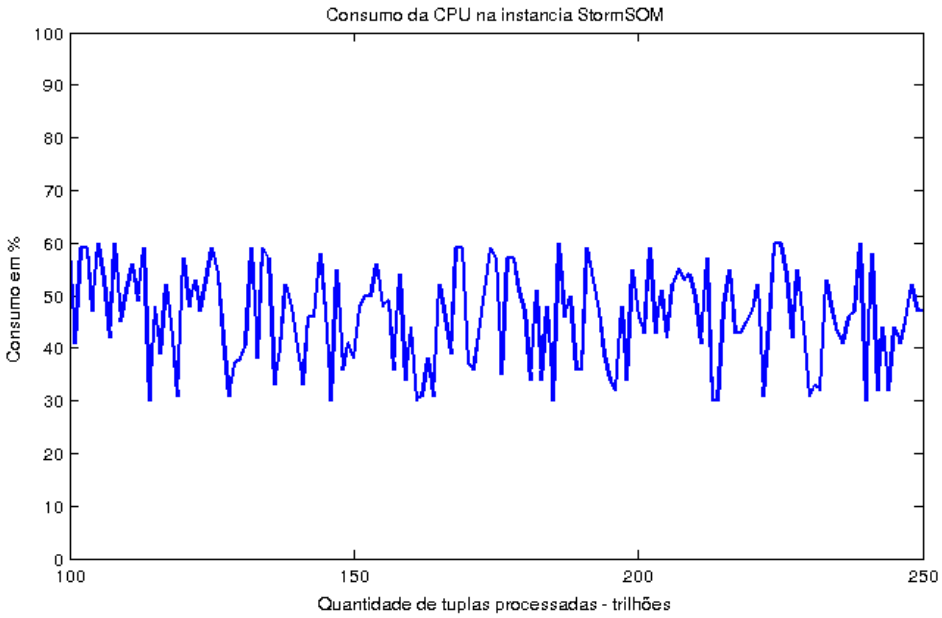

Figura 12 - Consumo de recursos da instância para o processamento do StormSOM em modo *single-node*

O tempo de processamento total varia entre 0.0000148 e 0.000176 segundos e é possível perceber variações homogêneas durante processamento das tuplas, ocorrendo alguns padrões de picos de processamento. Observou-se que esta variação é ocasionada devido à etapa de leitura e escrita do Redis, pois como seus dados são guardados em memória, e a máquina possui memória limitada, o desempenho pode variar quando blocos volumosos de dados são persistidos concorrentemente com as demais operações feitas no banco. Persistir em blocos, consome menos recursos do banco de dados se compararmos com a persistência de dados unitários, entretanto, pode prejudicar o tempo de processamento, portanto, é importante atentar para o momento correto que a persistência será unitária ou em bloco.

Em relação ao uso da CPU a análise gráfica mostra que o uso do processador está sempre abaixo de 65%, mesmo diante dos demais serviços básicos do sistema operacional.

Este resultado, garante que não haverá sobrecarga na instância, independente do volume de dados processados. A variação do uso da CPU, é compreensível pela noção de que, para a instância, sempre será processado uma tupla por vez, mesmo que hajam diversos nodos sendo executados paralelamente.

5.2.1.2 Processamento de fluxos distribuídos

Nesta análise, vamos incluir mais de uma fonte de dados, ou seja, vamos analisar o desempenho do StormSOM para o caso de se ter diversas fontes de dados distribuídas processando em um único nó, como ilustrado na figura 11. A figura 12, apresenta os resultados para 10, 20 e 50 fontes de dados.

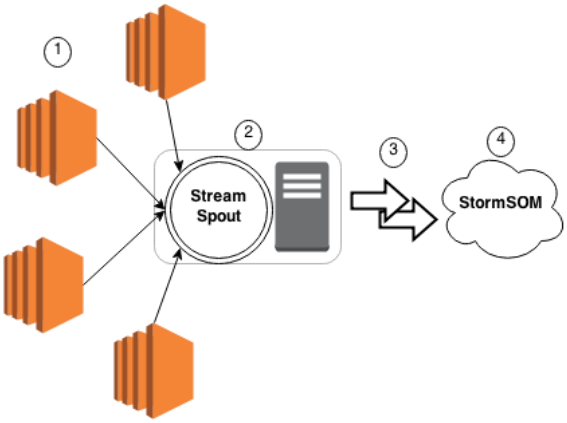

Figura 13 - Arquitetura conceitual da execução do StormSOM para diversas fontes de dados e com o processamento em um único nó. (1) representa as fontes de dados, em (2) uma instância Redis aglomera os dados e envia em fluxo (3) para o StormSOM (4).

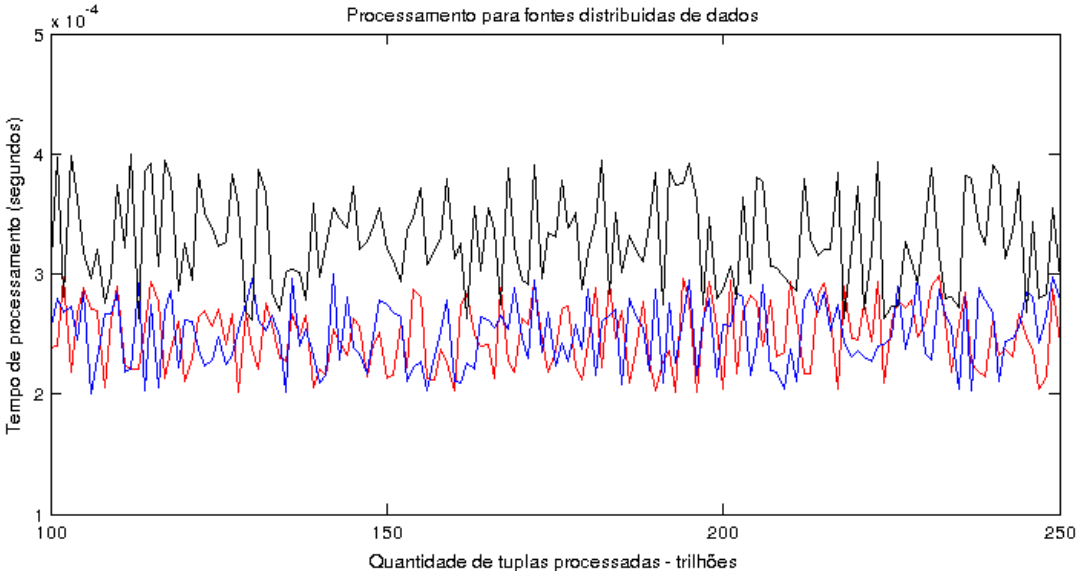

Figura 14 - Resultados do processamento para fontes distribuídas de dados. (a) resultado para (a)10, (b) 20 e (c) 50 fontes

Aumentou-se o fluxo de dados, e manteve-se o tempo de coleta desses dados, com isso ocorreu maior sobrecarga nos nós de processamento da topologia

do StormSOM. Percebemos, pelos resultados apresentados, que houve um aumento substancial no tempo de processamento das tuplas. Poderíamos garantir um tempo inferior, como na figura 9, caso aumentássemos o intervalo de coleta, no entanto o algoritmo levaria mais tempo para processar todas as tuplas do fluxo de dados.O nó *StreamSpout*tem a função de aglomerar os dados das diversas fontes e disponibilizá-los para processamento, o mesmo apresentou um consumo de CPU como destacado na figura 13.

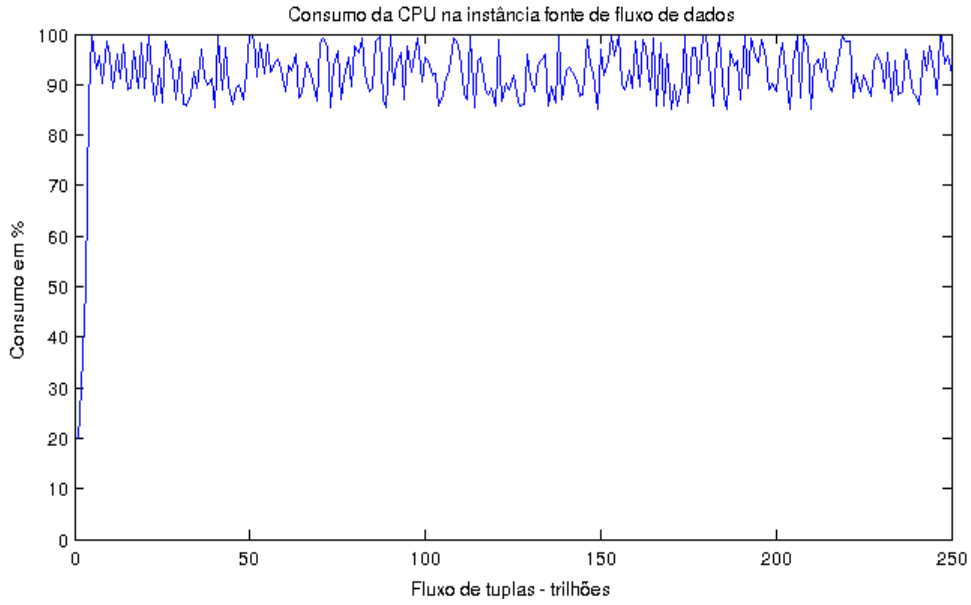

Figura 15 - Carga de processamento na instância fonte do fluxo de dados

A partir desta análise, verificamos que o StormSOM, mantém-se estável mesmo diante do aumento significativo das fontes de dados. Por outro lado, é observado maior sobrecarga no *StreamSpout*, pois o mesmo irá coletar e armazenar esses dados (escrita) e os disponibilizará para serem consumidos e processados pelo StormSOM (leitura). Mesmo utilizando um banco de dados com alta escalabilidadehá esta sobrecarga, uma solução seria adequar as configurações dos recursos desta instância para lidar com este volume de dados, outra solução seria escaloná-lo horizontalmente, aumentando a quantidade de nós de armazenamento Redis.

5.2.1.3 Processamento de topologia distribuída

Como no exemplo da subseção anterior, vamos analisar o desempenho do StormSOM para processar diversas fontes de dados, entretanto, neste experimento, vamos analisar o StormSOM em um ambiente paralelo e distribuído, onde cada nó da topologia é um cluster de processamento (figura 13). A figura 17, mostra os resultados desta simulação para 10, 20 e 50 fontes de dados.

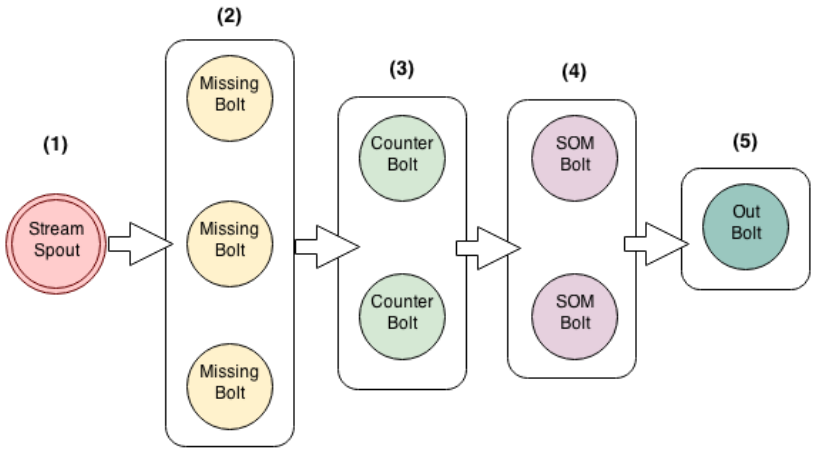

Figura 16 - Arquitetura conceitual do processamento paralelo e distribuído do StormSOM, onde cada componente é executado em uma instância.

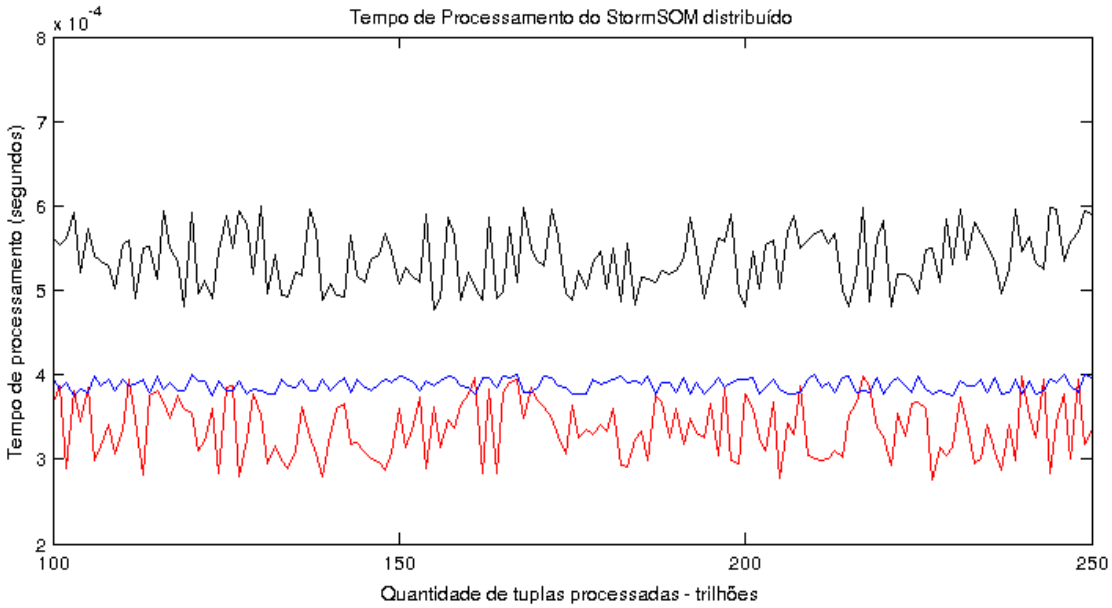

Figura 17 - Resultados do processamento paralelo e distribuído do StormSOM. (a) resultado para 10, (b) 20 e (c) 50 fontes

Na figura 17 fica clara a diferença da estabilização do processamento para 10, 20 e 50 fontes de tuplas. A figura mostra que em 20 tuplas há a presença de uma estabilização na variação do tempo de processamento das tuplas: isto não se dá apenas pela quantidade de tuplas, haja vista que para 10 fontes distribuídas de dados não há este tipo de comportamento, no entanto, ocorre devido a vazão da entrada e saída da tupla do StreamSpout (responsável por receber e organizar as fontes de dados), o tempo de persistência de cada nós de processamento (ler/escrever os pesos da rede neural em tempo de execução), e saída da topologia. Logo, nesta análise, para o cenário de software e hardware utilizados na simulação, 20 fontes é o ideal para o processamento de fluxos de dados distribuídos. Para alterar essa condição, basta modificar o ambiente de execução.

Existem cenários, onde distribuir os nós de processamento pode vir a ser uma decisão correta, entretanto é importante perceber que existem fatores que influenciam neste tipo de resultado, como: interface de rede, conectividade, protocolos de transferência de dados, haja vista que não há mais memória compartilhada nesses casos, a tramitação de dados e feita pela interface de rede, podendo ocorrer falhas ou ruídos. O Apache Storm possui um controle de tolerância à falhas, onde o mesmo continua tentando enviar seus resultados para o próximo nó, até que este envio seja efetivado com sucesso. Entretanto, o fato de existir este controle de falhas, ainda há um impacto no tempo de processamento.

Esta análise destaca a flexibilidade da solução proposta, haja vista que a mesma possui a capacidade de ser implementada em diversos cenários, tento bons resultados em cada um deles, sendo escalável de acordo com cada necessidade.

5.3 CONSIDERAÇÕES FINAIS

É importante destacar que os cenários analisados não são de uso genérico, o primeiro passo é entender a demanda de processamento para, posteriormente, decidir a forma como o StormSOM deverá ser configurado, seja para processamento *single-node* com uma única fonte de dados, seja para diversas fontes de dados, para processamento paralelo e distribuído, ou até mesmo, se é realmente necessário o uso de uma estrutura para processamento de fluxos de dados, pois em muitos caso, é mais adequado a aplicação do processamento em lote. Caso a escolha seja feita de

forma equivocada é provável que haja uma perda de desempenho e ineficiência do modelo, portanto, a escolha coerente da forma de aplicação é um passo fundamental para o funcionamento correto do StormSOM.

A decisão por utilizar um ambiente de *cloud computing* para a realização das simulações dá-se devido a intenção de aproximarmos, ao máximo, a proposta apresentada neste trabalho com os domínios de aplicações reais. Atualmente, quase a totalidade dos serviços corporativos estão sendo executados nesses ambientes. Empresas especializadas nesses serviços como: Amazon, Microsoft, Oracle, IBM, RackSpace, dentre outras, oferecem pacotes de serviços cada vez mais especializados e incorporam soluções que venham a solucionar problemas relacionados ao armazenamento e à análise de dados. Por esse motivo, optamos por apresentar e comprovar que nossa proposta é factível e consistente para a utilização em ambientes reais de produção, sobre diversas áreas e domínios do conhecimento.

6 CONCLUSÃO

O campo da mineração de dados constitui uma alternativa para processar grandes volumes de dados, provenientes de diversas fontes, dada a sua capacidade para descobrir padrões, possibilitando o apoio em análises complexas sobre dados e tomada de decisão. Com cenários cada vez mais complexos surge a necessidade do uso de modelos computacionais mais refinados, entretanto, em domínios com conjuntos de dados mais complexos e muitos deles necessitando gerar respostas em tempo real

Neste trabalho foi proposto o StormSOM, um processo para clusterização distribuída de grandes volumes de fluxos de dados. Dentre as soluções disponíveis no mercado, o StormSOM se destaca nos seguintes pontos:

- Propõe uma adaptação ao modelo neural de clusterização capacitando sua utilização sobre grandes volumes contínuos e ilimitados de dados, gerando respostas em tempo real
- Modelo baseado em topologia, contruído sobre um dos frameworks mais inovadores do mercado.
- Flexível e escalável, adequando-se a qualquer volume de dados.
- Capacidade de coletar e processar dados de forma distribuída.
- Arquitetura programável e fácil de adaptar para as especificidades de cada domínio de aplicação

Teve-se a preocupação de detalhar o estado da arte em relação as técnicas, modelos e ferramentas para tratar fluxos de dados, a fim de destacar o fator de inovação deste trabalho, pois, até o momento, é o primeiro a propor uma arquitetura baseada em topologia para computação de fluxo de dados, que faça a clusterização de grandes volumes de dados de forma distribuída, utilizando um algoritmo neural.

O StormSOM tem como objetivo principal a clusterização de dados, no entanto ele também trata dados faltosos e faz coleta estatística dos dados, abrindo a possibilidade da aplicação de outras técnicas para extração do conhecimento, os experimentos realizados mostram a eficiência do processo no que o mesmo se propõe.

Este trabalho foi publicado no XXXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (LIMA, 2015) e foi base para o desenvolvimento de

um processo, para o qual foi solicitado um depósito de patente de invenção junto ao Setor de Propriedade Intelectual da Universidade Federal do Pará, com protocolo Nº BR 10 2014 033141 7.

Como trabalhos futuros, pretendemos expandir as análises de modo a trazer um foco maior para a qualidade dos resultados comparando os mesmos com resultados obtidos através de implementações tradicionais do algoritmo SOM. Pretende-se também, utilizar outros conjuntos de dados e ampliar o parque de servidores para a utilização da arquitetura proposta em grande escala. Outra análise futura é utilizar sensores como fontes de dados, ao invés de uma máquina que simule essa função. Como o StormSOM já possui uma coleta estatística de dados, também é uma expectativa futura, a inclusão de outros métodos estatísticos e modelos de aprendizagem de máquina ainda mais refinados para extração do conhecimento. Por fim, pretendemos refinar a arquitetura proposta com as inovações tecnológicas que surgiram ao longo desta pesquisa.

Alguns problemas foram encontrados na elaboração deste trabalho, pois como estamos tratando de um tema recente, é normal o surgimento trabalhos relacionados que estiveram sendo desenvolvidos paralelamente por outros grupos de pesquisa, o que ocasionou uma análise retroativa sobre o escopo, a cada novo trabalho analisado. Além disso, destacamos que a maior dificuldade no processo de elaboração, está diretamente relacionada ao custo para a realização do trabalho, em termos financeiros. Os ambientes de *cloud computing* são cobrados por tempo de máquina ligada, armazenamento, pacotes de dados transmitidos, dentre outros custos. No processo de elaboração desta pesquisa, incluindo principalmente a etapa de simulações, foram necessárias muitas horas de uso dos servidores, desde a criação do ambiente, aprendizagem dos recursos, testes de desempenho, testes de configuração, instalação dos sistemas operacionais e ferramentas, criação de novas instâncias, armazenamento de dados, melhorias nas interfaces de rede e comunicação, dentre outras atividades que demandaram tempo e consequentemente custos elevados para o projeto, que foi custeado de forma independente.

REFERÊNCIAS

TOM WHITE. Hadoop: The definitive guide, 4th Edition, O'Reilly, 2012. 7

JUSTIN ERICKSON MARCEL KORNACKER. Cloudera impala: Real-time queries in apache hadoop, for real 2012.

HENRIQUE C. M. ANDRADE, Fundamentals of Stream Processing: Application Design, Systems, and Analytics, 1st Edition, Cambridge, 2013.

IBM. A new paradigm: Stream computing and IBM's InfoSphere Streams. 2014.

JOEY R. Object-oriented neural networks in C++, London: Academic Press; 1997.

BAKSHI, K. 2012. Considerations for big data: Architecture and approach. Aerospace Conference, 2012 IEEE, vol., no., pp.1,7, 3-10. doi: 10.1109/AERO.2012.6187357

WEC. WORLD ECONOMIC FORUM. Big Data, Big Impact: New Possibilities for International Development, 2012.

BENOÎT PERROUD. Apache Cassandra, http://goo.gl/YLMt0f, 2013. iii, 13, 19

MOSTAFA ABD-EL-BARR. Design and Analysis of Reliable and Fault-Tolerant Computer Systems. Kuwait University. 2013.

ROBERT GREINER. CAP Theorem: Explained. http://goo.gl/Alchur. 2014

N. HURST. Visual Guide to Nosql Systems. http://goo.gl/2kNXVc. vii, 18. 2014

SEGUIN K. The Little Redis Book. 2014.

PAKHIRA, M. K., BANDYOPADHYAY, S., MAULIK, U. Validity index for crisp and fuzzy clusters, Pattern Recognition, June 2004.

SIMON H. Redes neurais: princípios e técnicas, Porto Alegre: Bookman; 2001.

A. BIFET, G. HOLMES, R. KIRKBY, AND B. PFAHRINGER, "MOA: Massive Online Analysis," J. Mach. Learn. Res., vol. 11, pp. 1601–1604, Aug. 2010.

NATHAN MARZ. Storm Trident, https://goo.gl/cTBCwP, 2014

NATHAN MARZ. Storm Ix, https://goo.gl/cTBCwP, 2014.

PETER ALVARO TYSON CONDIE, NEIL CONWAY. Mapreduce online. Technical report, UCBerkely, Yahoo! Research, 2009. 35

THE APACHE FOUNDATION. Apache Drill. Edição 1, O'Reilly 2014

IBM. IBM Infosphere, 2013. 37

EPL. Eclipse Public License. 2013

APACHE. Welcome to apache pig! 2013

PACO NATHAN. Enterprise Data Workflows with Cascading. Edição 1. O'Reilly. 2013

NATHAN MARZ. A storm is coming: more details and plans for release. TwitterEngineering, 2011. 42

RAN BI. Vowpal Wabbit: Fast Learning on Big Data. 2014

M. WOJNARSKI, “Debellor: A Data Mining Platform with Stream Architecture,” in Transactions on Rough Sets IX, pp. 405–427, Springer, 2008

M. STONEBRAKER, U. C¸ETINTEMEL, AND S. ZDONIK, The 8 Requirements of Real-Time Stream Processing, SIGMOD Rec., vol. 34, pp. 42–47, Dec. 2005.

D. J. ABADI, D. CARNEY, U. C ¸ETINTEMEL, M. CHERNIACK, C. CONVEY, S. LEE, M. STONE-BRAKER, N. TATBUL, AND S. ZDONIK, “Aurora: a new model and architecture for data stream management,” The VLDB Journal, vol. 12, pp. 120–139, Aug. 2003.

IBM REDBOOKS. Addressing Data Volume,Velocity, and Variety withIBM InfoSphere Streams V3.0. 2014.

ERIC SIEGEL. Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die. 2014

HENRIQUE C. M. ANDRADE . Fundamentals of Stream Processing: Application Design, Systems, and Analytics. 2014

L. NEUMEYER, B. ROBBINS, A. NAIR, AND A. KESARI, “S4: Distributed Stream Computing Platform,” in 2010 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 170–177, 2010.

P. TAYLOR GOETZ, BRIAN O'NEILL . Storm Blueprints: Patterns for Distributed Real-time Computation. 2014

XIAODONG ZHU, JIANZHENG YANG. An Extended Predictive Model Markup Language for Data Mining, 2010.

ANAND NALYA, ANKIT JAIN . Learning Storm, 2014.

SHOHEI HIDO, SEIYA TOKUI Jubatus: An Open Source Platformfor Distributed Online Machine Learning. 2014

DANIEL POP. Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions. 2014

FLAVIO JUNQUEIRA, BENJAMIN REED. ZooKeeper: Distributed Process Coordination. 1ª Edição, 2014.

ALBERT BIFET, GEOFF HOLMES, RICHARD KIRKBY AND BERNHARD PFAHRINGER. Data Stream Mining: A Practical Approach. 2011.

A. BIFET, G. HOLMES, R. KIRKBY, AND B. PFAHRINGER, “MOA: Massive Online Analysis,” J. Mach. Learn. Res., vol. 11, pp. 1601–1604, Aug. 2010

RAUL CASTRO FERNANDEZ, Towards Low-Latency and In-Memory Large-Scale DataProcessing. DEBS 2013. The 7th ACM International Conference on Distributed Event-Based Systems. 2013

P. TAYLOR GOETZ, BRIAN O'NEILL, Deploying Storm on Hadoop for Advertising Analysis, 2014.

QUINTON ANDERSON. Storm Real-Time Processing Cookbook, 2014.

LIMA, J. G. R. O. ; SANTANA, A. L. ; VIJAYKUMAR, N. L. ; FRANCES, C. R. L. ; G. Jeremy ; D. Robert . Metodologia para Clusterização em Tempo Real de Fluxos de Dados Climáticos Distribuídos no Contexto de Big Data. In: XXXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2015, Recife.