**NÍCOLAS HENRIQUE VIEIRA TOLEDO**

**SEBASTIÃO BATISTA DE ANDRADE NETO**

**ESTATÍSTICAS DE REDES SOCIAIS**

**COM BIG DATA**

**UNIVERSIDADE DO VALE DO SAPUCAI**

**POUSO ALEGRE**

**2015**

# DISCUSSÃO DE RESULTADOS

Neste capítulo são apresentados e discutidos os resultados obtidos pela pesquisa por meio do desenvolvimento das tecnologias empregadas no trabalho. Esta discussão foi realizada sob diferentes perspectivas, com o intuito de fazer a demonstração dos pontos fortes e fracos de cada uma delas.

Inicialmente, antes mesmo de ser feita a escolha do tema, o objetivo dos participantes era desenvolver um sistema de acesso público que abrangesse grande parte da população brasileira. Além disso, era necessário fazer a escolha das ferramentas e tecnologias que, preferencialmente, não fossem abordadas pela grade do curso de Sistemas de Informação.

Após serem feitas as pesquisas sobre as áreas de grande influência no Brasil, decidiu-se que o sistema deveria abranger o cenário político. A ideia inicial era ajudar a população brasileira a fazer uma análise dos papéis desempenhados pelos políticos brasileiros, por meio da demonstração da opinião pública dos mesmos. Para que isso fosse realizado, foi necessário pesquisar sobre as tecnologias a serem empregadas.

Inicialmente, foi constatado que um sistema web atenderia a necessidade do acesso público do sistema. O sistema web também tem a vantagem de poder atingir uma proporção maior da população brasileira. Seguindo esta direção, chegou-se a conclusão que as tecnologias relacionadas à *Big Data* sanariam as necessidades do projeto, já que o volume de dados seria alto e as informações deveriam estar disponíveis para os usuários em tempo integral.

Além disso, foi necessário definir de onde os dados seriam minerados. Depois de algumas análises, foi decidido que o melhor caminho seria a mineração de dados de redes sociais. Dentre todas, a rede social *Twitter* foi a escolhida, pois por meio dela foi possível ter acesso às opiniões públicas de maneira mais direta e clara, além dos usuários estarem em constante expressão política em seus *tweets*. Para isso, foi utilizado o *framework* *Apache Storm*, o qual faz a mineração dos *tweets* através do *Twitter Stream* API, retornando os mesmos no tempo de sua ocorrência.

O *Storm* implementado juntamente com a biblioteca *Twitter4j*, especificamente designada para a mineração de *tweets*, além de resolver a necessidade de busca dos *tweets*, forneceu também a base necessária para a implementação do tratamento e armazenamento das informações necessárias para o projeto.

Para ser feito o armazenamento das informações, foi necessária a escolha de um banco de dados especificamente designado para grandes volumes de dados, não podendo assim, ser um banco de dados relacional. Para tal atividade, foi utilizado o banco NoSQL MongoDB, pois ele é um banco de dados NoSQL que tem um ótimo desempenho relacionado a volume de dados, além de ser designado à aplicações web.

Assim que foram decididas as tecnologias de mineração e armazenamento de dados, foi preciso decidir quais tecnologias seriam utilizadas para ser feito o desenvolvimento da aplicação web, definida como um dos objetivos específicos do projeto. Desse modo, foi utilizado o Node.JS juntamente com o *framework* *Express*, os quais auxiliaram na criação da infraestrutura da aplicação, rotinas *server-side* e persistência dos dados.

Para facilitar a comunicação, foi utilizada a linguagem *JavaScript* no *front-end*, devido ao servidor Node.JS ser inteiramente construído na linguagem *JavaScript*. Desse modo, foi também utilizada a linguagem de marcação HTML5 juntamente com a folha de estilos CSS3 e o *framework Bootstrap* para a apresentação das informações para o usuário da aplicação, além do EJS *tamplate engine*.

Após estas etapas, os objetivos específicos e as ferramentas da pesquisa estavam definidos, e os prazos estipulados. As tarefas foram posteriormente divididas entre os autores do projeto, sendo periodicamente realizadas reuniões para o acerto dos detalhes.

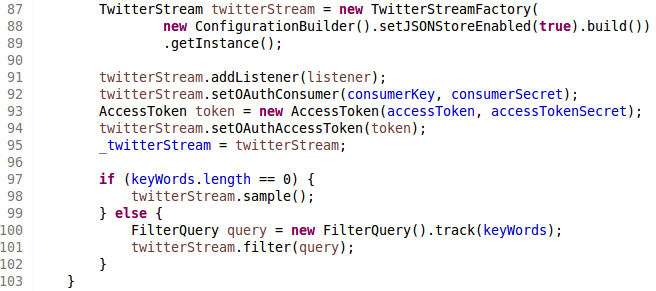
O resultado de todas estas etapas foi o sistema de estatísticas políticas Politistatus. O sistema foi desenvolvido para ser acessado por qualquer cidadão brasileiro com acesso à internet que deseja saber como está a aprovação popular de seu representante na rede social *Twitter*. O sistema foi desenvolvido para ser executado no servidor Node.JS, totalmente escrito na linguagem de programação *JavaScript*, em contraste com o *framework* de mineração de dados, *Apache Storm*, inteiramente escrito na linguagem Java.

Desse modo, foram alcançados os objetivos específicos do projeto, com um produto capaz de apresentar as informações políticas da rede social *Twitter* em seu tempo de ocorrência. As tecnologias-chave para a construção do sistema foram o *framework* *Apache Storm* em conjunto com a biblioteca *Twitter4j*, pois através delas, foi possível fazer a conexão com a *Twitter Stream* API, além de ser feita a mineração e armazenamento da massa de dados.

Conforme mencionado por Leibiusky, Eisbruch e Simonassi (2012), o *framework Apache Storm* foi confiável ao ponto de não serem encontradas falhas em sua implementação. O fluxo de entrada dos dados é tratado por um *spout* conectado diretamente à *Twitter Stream* API.

Conforme mencionado por Twitter4j (2015), foi adicionada a biblioteca *Twitter4j* na implementação do *Storm*, proporcionando assim uma conexão sem maiores dificuldades à *Twitter Stream* API. Desse modo, foi possível focar mais no desenvolvimento da lógica de tratamento dos dados recebidos pelo *Storm*.

A Figura 1 demonstra no código o momento onde é feita a conexão com a *Twitter Stream* API, a qual é setada no momento da criação da topologia do *Storm*.



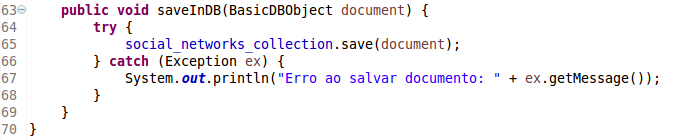
**Figura 1** - Conexão com a *Twitter Stream* API. **Fonte:** Elaborado pelos autores.

Continuando com a confirmação feita por Leibiusky, Eisbruch e Simonassi (2012), após a leitura dos dados minerados, as tuplas são encaminhas para a o *bolt* responsável pelo tratamento dos *tweets*, verificando se os dados das tuplas recebidas contém as informações de importância necessária para a aplicação. A Figura 2 demostra o método da classe *PolitistatusTwitterBolt*, recebendo como parâmetro uma tupla e suas informações.



**Figura 2** - Classe *PolitistatusTwitterBolt*, o *bolt* do *Storm*. **Fonte:** Elaborado pelos autores.

Após as informações serem tratadas pelo *bolt*, é gerado um objeto que será enviado para a classe responsável pelas operações relacionadas com o banco de dados. Nessa classe, os dados que estão no objeto são salvos. A partir da repetição desse fluxo, é gerada a massa de dados, que posteriormente estará disponível para o consumo da aplicação web. A Figura 3 demonstra o método da classe *PolitistatusDatabase* responsável por salvar os *tweets* no MongoDB.

****

**Figura 3** - Método onde ocorre o armazenamento dos *tweets* minerados. **Fonte:** Elaborado pelos autores.

Paralelamente a toda esta lógica, foi implementada uma *Task* no momento em que a topologia do *Storm* é criada, com a finalidade de verificar se as palavras-chave informadas pelo usuário não foram alteradas. A *Task* executa em um intervalo de tempo de 5 segundos, continuamente. A Figura 4 demonstra no código o momento em que a *Task* é criada, ou seja, logo após a criação da topologia do *Storm*.



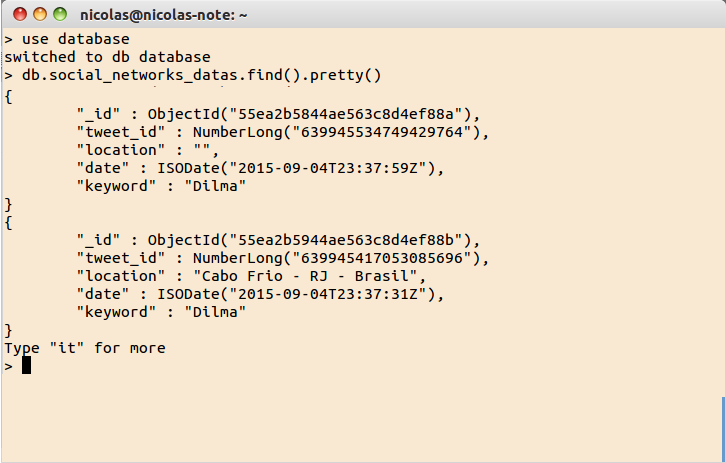
**Figura 4** - Criação da *Task*. **Fonte:** Elaborado pelos autores.

Assim, quando a topologia é criada, é chamada uma classe responsável pela lógica da *Task*. Essa lógica executa uma busca nos documentos do banco de dados, mais especificamente no campo de *usersKeywords*, retornando uma lista contendo todas as palavras-chave cadastradas pelos usuários da aplicação web. Caso essa lista de palavras-chave seja diferente das que estão sendo usadas para a mineração dos *tweets*, destrói-se a topologia antiga, criando uma nova topologia para a mineração dos dados atualizados, iniciando novamente o ciclo de vida do *Storm*. A Figura 5 demonstra a lógica da *Task*.



**Figura 5** - Lógica da *Task*. **Fonte:** Elaborado pelos autores.

Confirmando o que foi mencionado por Chodorow (2013), o banco de dados MongoDB foi flexível o bastante para armazenar a massa de dados gerados pelo *Storm*. Por ser um banco de dados orientado a documentos, foi de extrema facilidade realizar buscas com relações hierárquicas complexas, sanando assim uma das necessidades do sistema. A Figura 6 exemplifica o formato de documento dos *tweets* no MongoDB. Estes dados foram previamente minerados pelo *Storm*.



**Figura 6** - *Tweets* no MongoDB, no formato de documentos. **Fonte:** Elaborado pelos autores.

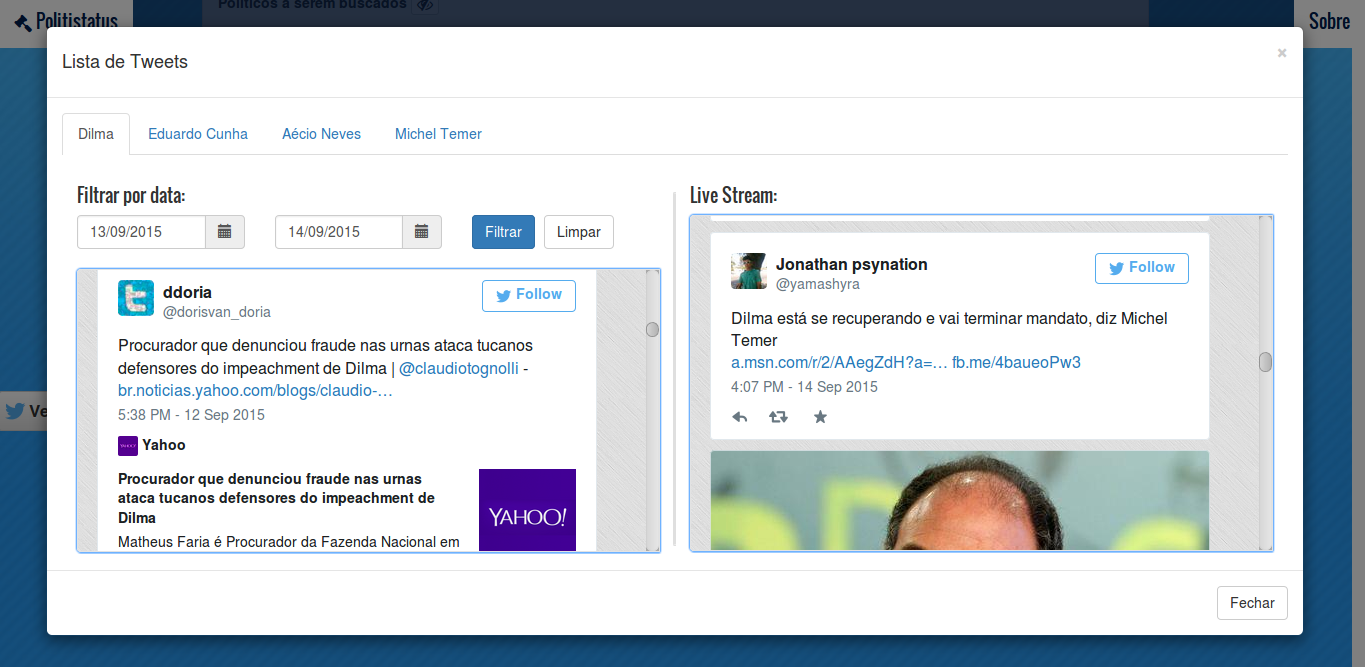
Para ser feito o envio dessas informações para o usuário, foi necessária a implementação de um servidor com arquitetura não-bloqueante. Realmente, a tecnologia Node.JS foi a escolha perfeita para o desenvolvimento do *back-end* da aplicação web. Assim, confirmando o que foi mencionado por Pereira (2013), a implementação do Node.JS proporcionou à aplicação boa performance e baixo consumo de memória por causa de seu modelo não-bloqueante.

Desse modo, foi possível implementar um mecanismo de busca dos *tweets* mais atualizados do banco de dados. Esse mecanismo, com o funcionamento semelhante à *Task* criada no *Storm*, busca os *tweets* mais recentes no banco em um intervalo de tempo de 1 segundo. Foi necessária também a implementação do *framework* Socket.IO no servidor, para que uma conexão *socket* estivesse sempre aberta para o envio desses *tweets* para o usuário. A Figura 7 exemplifica no código a lógica do mecanismo de busca dos *tweets* atualizados, sendo posteriormente enviados para a aplicação web via Socket.IO na linha 35.

****

**Figura 7** - Método da busca atualizada e envio dos *tweets*. **Fonte:** Elaborado pelos autores.

Após a implementação de todas essas tecnologias, os *tweets* estão prontos para serem mostrados para o usuário da aplicação web. Para tal, foi criada uma modal do *Bootstrap* responsável por gerar e mostrar esses *tweets*. A geração dos *tweets* é feita através de uma chamada em um arquivo *JavaScript* do próprio *Twitter*, o qual recebe o *id* do *tweet* como parâmetro. Posteriormente, o *tweet* é carregado e retornando para o *JavaScript* responsável por mostrá-lo. Esse mesmo arquivo também é responsável por mostrar para os usuários os *tweets* filtrados por data. A Figura 8 demonstra a modal de demonstração dos *tweets*. Conforme demonstrado na imagem, do lado esquerdo da modal são apresentados os *tweets* fitrados por data, e, no lado direito, são mostrados os *tweets* em seu tempo de ocorrência.

****

**Figura 8** - Modal de demonstração dos *tweets*. **Fonte:** Elaborado pelos autores.

Paralelamente à demonstração dos *tweets* para o usuário, é feita a marcação dos locais no mapa. Primeiramente, é setado no mapa a latitude, longitude e zoom corretos, para que o mapa fique devidamente centralizado na tela. Quando os dados são recebidos via Socket.IO, é chamada a função *getLocationByAddress* do *JavaScript* controlador do mapa, sendo passado como parâmetro a cidade do usuário responsável pelo *tweet*. Essa função fica responsável por passar essa cidade para uma URL do *Google Maps* API, a qual irá retornar um objeto do tipo JSON contendo a latitude e longitude da cidade.

Após esses procedimentos, é chamada a função *setMaker* do controlador do mapa, o qual fica responsável por criar um marcador no mapa indicando qual a cidade do usuário que gerou aquele *tweet*.

Desse modo, confirmando assim o que foi dito por Svennerberg (2010), todas as configurações e procedimentos para o funcionamento dinâmico do mapa foram feitos através de arquivos *JavaScript*. A Figura 9 exemplifica o mapa da aplicação com algumas localidades marcadas.

****

**Figura 9** - Mapa da aplicação, com algumas marcações. **Fonte:** Elaborado pelos autores.

Depois que os testes foram realizados verificou-se que a aplicação já se encontrava apta a demonstrar de forma eficiente e concisa a opinião política dos usuários do *Twiter*. Desse modo, comprovou-se que o objetivo geral da pesquisa foi alcançado.

# REFERÊNCIAS

CHORODOW, K. **MongoDB The Definitive Guide**. 2. Ed. Sebastopol: O’Reilly, 2013.

LEIBIUSKY, J.; EISBRUCH, G.; SIMONASSI, D. **Getting Started Whith Storm.** Sebastopol: O’Reilly, 2012.

PEREIRA, C. R. **Aplicações web real-time com Node.js.** São Paulo: Casa do Código, 2013.

TWITTER4J. **Twitter4j - A Java library for the Twitter API**. Disponível em: <http://twitter4j.org/en/index.html>. Acessado em 30/05/2015.