Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

UMA FERRAMENTA PARA PROCESSAMENTO DE CONSULTAS MULTIDIMENSIONAIS PARA DADOS DE REDES SOCIAIS

Aluno: Milton Stiilpen Júnior
Matricula: 07.1.4171

Orientador: Álvaro Rodrigues Pereira Júnior

Ouro Preto
19 de dezembro de 2011

Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

UMA FERRAMENTA PARA PROCESSAMENTO DE CONSULTAS MULTIDIMENSIONAIS PARA DADOS DE REDES SOCIAIS

Relatório de atividades desenvolvidas apresentado ao curso de Bacharelado em Ciência da Computação, Universidade Federal de Ouro Preto, como requisito parcial para a conclusão da disciplina Monografia I (BCC390).

Aluno: Milton Stiilpen Júnior
Matricula: 07.1.4171

Orientador: Álvaro Rodrigues Pereira Júnior

Ouro Preto
19 de dezembro de 2011

Resumo

As redes sociais sempre foram alvo de pesquisa, principalmente por parte dos sociólogos. Hoje, com a eminência do poder computacional, a *Web* tornou-se capaz de conectar as diversas partes do globo, possibilitando assim a coleta e análise de um rico universo de dados da massas de usuários que nela navegam. Assim, o estudo de redes sociais tem, dia após dia, se mostrado factível e valoroso para os pesquisadores da Ciência da Computação. Dessa forma, o que será proposto nesse trabalho de conclusão de curso é uma ferramenta para o processamento de consultas multidimensionais para dados de redes sociais.

Palavras-chave: Recuperação de Informação na *Web*. Redes Sociais. Engenharia de Software. Sistemas Distribuídos.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Encontra-se, na atualidade, uma explosão de informações graças à existência de diversas redes em escala mundial e de "gigantes" que trabalham os dados oriundos destas, como, a título de exemplo, a *Google* e o *Facebook*. Pode-se dizer que as pessoas acrescentam diariamente conteúdo na Internet, seja alguma notícia, seja um comentário ou opinião sobre política, religião, seus gostos e interesses, humor etc[5]. Sem contar a interação que existe entre as pessoas, principalmente em redes sociais. E o melhor de tudo: quase todos esses dados são públicos e relativamente fáceis de se coletar! [12]

Nem sempre, no entanto, foi assim. Outrora, a dificuldade de se obter dados era assustadora se comparada com o agora. Obstáculos geográficos, pessoais, culturais, entre outros, atrapalhavam uma análise mais abrangente de uma rede social. A coleta de informações normalmente era feita por questionários e demandava tempo. Seu fim principal, contudo, não deixou de ser o mesmo: possibilitar o estudo aprofundado do comportamento humano. No caso das redes sociais *online*, o estudo dos dados de uso de um usuário podem trazer diversas oportunidades. Ferramentas de tomada de decisão, melhorias no design de interação de um sistema, otimização em sistemas distribuídos e redes, ricos estudos de interação social [4] etc.

Logo, nós, cientistas da computação, através de algoritmos e estruturas de dados, temos a capacidade de desenvolver uma ferramenta capaz de coletar informações do provedor *web* e, a partir dessa, estruturar e analisar tais dados . É o que propomos nesse projeto. Desenvolver uma ferramenta extensível para a coleta e análise de redes sociais.

2 Justificativa

Por que os dados de redes sociais são importantes? Consoante Nielsen Online[10], a cada cinco usuários ativos da Internet, quatro visitam redes sociais. Os norte-americanos gastam mais tempo no *Facebook* do que em qualquer outro site dos Estados Unidos. Lá, aplicativos de redes sociais estão entre os três mais utilizados em *smartphones*. E podem-se ver esses dados repetirem no dia a dia. O mundo encontra-se interligado através de várias redes sociais online[2]. Dois terços da população *online* global visita ou participa de tais redes. O crescimento dessa área trouxe consigo o interesse dos cientistas, que começaram a pesquisar sobre o tema. Não obstante, o estudo em si ainda está na sua "infância" [3].

Por que criar uma ferramenta? O foco do estudo ainda não é específico. Todavia uma ferramenta desse porte abre portas para uma série de estudos mais aprofundados, como, por exemplo: análise de sentimentos [6]; propagação de mensagens [11]; "previsão" de notícias [9] etc. Sem contar com a própria engenharia por trás da ferramenta, que envolve Áreas como Recuperação de Informação na *Web*(coletores, índices invertidos, similaridade de documentos), Sistemas Distribuídos e Engenharia de Software(padrões de projeto).

A ideia de desenvolver uma ferramenta que tenha funcionalidades como consulta textuais, consultas estruturadas, agregador de valores, gráficos e mapas sobre dados de redes sociais se mostra, portanto, siginificativamente interessante.

3 Objetivos

3.1 Objetivo geral

O objetivo, em linhas gerais, é chegar ao final da Monografia II (BCC 391) com um sistema *web* sólido de análise dos dados de redes sociais, que seja capaz de armazenar e consultar algumas milhões de mensagens, em tempo real. A ferramenta é, de forma genérica, um sistema no padrão de arquitetura Model-View-Presenter[1] (MVP), mais especificamente uma variante acrescida de dois módulos auxiliares. Os componentes que conterá vão da coleta até a visualização de gráficos e mapas, conforme ilustra o Diagrama de Componentes na Figura 1. A solução pode e deve receber alterações, sempre com a intenção de melhorias. Na próxima seção, será explicada cada parte da arquitetura proposta.

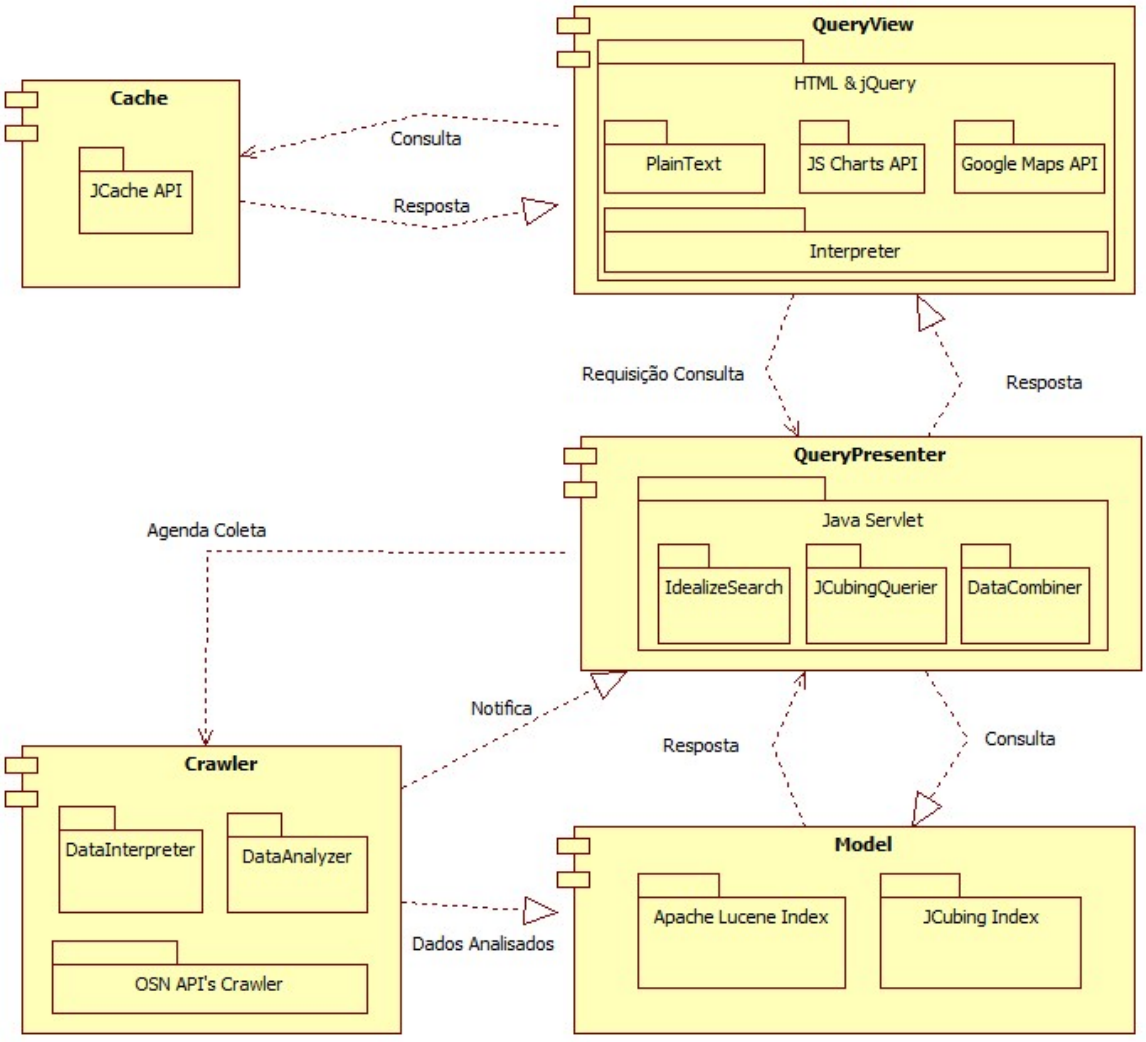

Figura 1: Diagrama de Componentes atual da Ferramenta

3.2 Objetivos específicos

- Model: O módulo de base de dados, nomeado *Model*, é o responsável por definir os dados a serem apresentados. Possui dois tipos de bancos de dados: Estruturados e Não Estruturados. Para a parte Estruturada, utilizamos o *JCubing*. Pela segunda, por sua vez, utilizamos o *Apache Lucene*. Ambos serão explanados à frente.

- QueryPresenter: Módulo nomeado *QueryPresenter* atua como mediador de requições/respostas. Ele recebe requição da camada *View*, é responsável por recuperar os dados da camada *Model* e formatá-los para serem visualizados. Ele possui pacotes de pesquisa de dados, *searcher* (pela parte não estruturada) e *cube* (pela parte estruturada) e um pacote, *dataCombiner*, que faz a combinação dos resultados das pesquisas e formatação dos dados para a *View*. E tudo é encapsulado em um Servlet (componente servidor do sistema).

- QueryView: Módulo nomeado *QueryView* é a interface que gera a visualização dos dados e monitora comando de usuários (eventos) para a camada *Presenter* atuar buscando os dados a serem apresentados. Desenvolvido no ambiente *Web*, baseado em bibliotecas *JavaScript: jQuery, JS Charts e Google Maps*.

- Cache: Levando em consideração que o processamento das pequisas pode ser custoso, devido à sua natureza, o módulo denomidado *Cache* é um componente para evitar que consultas comumente realizadas sejam sempre computadas. Assim, o resultado da pesquisa feita por um usuário é instantaneamente mostrado. Esse componente é baseado na biblioteca *Java Caching System*.

- Crawler: O módulo de coleta de dados, nomeado *Crawler*, será responsável por prover dados ao *Model* sempre que for requisitado. A existência desse módulo é possível graças às Aplicações de Programação de Aplicativos (API), providas pelas Redes Sociais, como *FacebookAPI* e *TwitterAPI*. Além desse pacote de coleta via API, existe um pacote de intepretador dos dados coletados e um pacote que irá análisa-los e classificá-los, para que possam ser adicionados ao nosso modelo de dados.

4 Metodologia

A metodologia utilizada nesse projeto para o desenvolvimento da ferramenta vem crescendo constantemente na Engenharia de Software: o *Scrum*. Consiste em uma metodologia de gerenciamento e desenvolvimento de software interativa e incremental. Nós o adaptamos para melhor nos atender nesse projeto. Primeiramente foi desenvolvido um *Backlog* e, a partir deste, ocorreram alguns *Sprints* ao longo do período. Cada um destes tinha suas tarefas predefinidas e vinha com um entregável ao fim do mesmo, como podemos ver no cronograma de atividadades (Tabela 2).

Requisitos	
1	Usuário pode fazer busca por termos; busca com operadores booleanos; busca por proximidade e busca por frase
2	Resultados estariam agregados inicialmente por estado, polaridade e anos. Cabe o usuário decidir como agregar
3	Busca avançada: Usuário especifíca como ele quer agregar os dados: Região, Cidade, Data, Ano, Semestre, Mês, Semana, Dia e até Hora do Dia
4	Resultados serão apresentados em forma de gráficos e mapas
5	Resultados serão salvos para outros usuários navegarem

Tabela 1: Requisitos

5 Cronograma de atividades

Na Tabela 2, o planejamento feito para o desenvolvimento da ferramenta.

Atividades	**Ago**	**Set**	**Out**	**Nov**	**Dez**
Definição do tema	X				
Sprint 0: Estudo sobre redes sociais	X	X			
Sprint 1: Desenvolvimento da arquitetura		X			
Sprint 2: Implementação Versão 1			X		
Sprint 3: Implementação Versão 2				X	
Sprint 4: Implementação Versão 3				X	
Sprint 5: Escrita do relatório final e Apresentação					X

Tabela 2: Cronograma de Atividades.

6 Desenvolvimento

Nesta seção iremos relatar todo o processo de desenvolvimento da ferramenta de consultas multidimensionais sobre dados de redes sociais. A ideia de se desenvolver essa ferramenta proveio das análises de conteúdo realizadas em [5]. Estas, feitas no artigo, foram executadas em uma base de dados através de um programa estático e sem interface gráfica. Ao perceber o valor daquelas, surgiu a possibilidade de gerar outras análises e, logo, desenvolver uma ferramenta que as fizesse de maneira dinâmica.

6.1 Base de dados

Para desenvolver a mencionada ferramenta, a Base de Dados utilizada é a mesma empregada em [5]. Essa base de dados é uma base da Rede Social *Twitter* com 29,013,854 de *tweets* de 198,714 usuários Brasileiros, desde a criação do Twitter em 2006 até Julho 2009. Cada *tweet* possui atributos como: Tweet Id, UserId, Mensagem, Estado, Data e Hora. A partir desses atributos, cada *tweet* passou por um algoritmo de indexação que formatou os dados em dois índices, um estruturado e um não estruturado. Como índice não estruturado, foi utilizada a *Apache Lucene*[8], que é uma biblioteca de máquina de pesquisa textual bem completa, *open source* e de alto *performance*. Com essa biblioteca, foi gerado um índice textual com duplas (Tabela 3) em que é possível fazer pesquisas, a partir de um aplicativo de pesquisa provido pela própria biblioteca.

Tweet Id	Mensagem
18019829	Quem NUNCA sonhou com um desses??? http://bit.ly/sOp5Wh *–*

Tabela 3: Exemplo de Dupla de Atributos-Índice Textual

Já como índice estruturado, foi utilizada o *JCubing*, que é uma biblioteca que provê o desenvolvimento de modelos e análise multidimensionais de dados, com um catálogo cheio de técnicas e métodos para computar, atualizar, representar e minerar cubos de dados multidimensionais [7]. Com o *JCubing*, foi gerado um índice com tuplas (Tabela 4)

Tweet Id	Pol...	Estado	Região	Ano	Semestre	Mês	Semana	Dia	Hora	Data
18019829	Pos	SP	Sudeste	2007	Sem1	03	Semana1	10	22	2007...

Tabela 4: Exemplo de Tupla de Atributos-Índice Estruturado

E como os atributos de um *tweet* originaram essas colunas?

- Quando se verificou que um *tweet* não possuia um valor para o atributo Estado, foi utilizada uma API do *Google Geocoding* para determinar a geolocalização a partir do campo texto de localização no cadastro do usuário que postou o *tweet* [5].
- A Polaridade de uma mensagem é a ideia de classificá-la como boa ou ruim, como pode ser visto com mais detalhes em [5]. Nesse caso, foram utilizados Emoticons para classificar as mensagens. Mensagens que continham Emoticons como

"=)" ou ":-)" foram classificados como "positivos". Já mensagens que continham Emoticons como "=(" ou ":-(" foram classificados como "negativos". Mensagens sem Emoticons, a seu turno, foram classificados na categoria "neutros". Essa, pois, é uma heurística simples para classificação de sentimentos, e mesmo assim a classificação de positivos/negativos chegou a 2,713,531 de *tweets*,10% do total.

- Atributos de Data e Hora em geral foram derivados para que combinações diferentes pudessem ser feitas e assim fosse possível aumentar o nível de especifidade de uma consulta no cubo multidimensional.

6.2 Implementação Versão 1

A partir da Base de dados e da suposta arquitetura, a implementação do sistema foi feita, módulo a módulo, no sentido *bottom-up*.

Primeiramente, foi desenvolvida a camada *Model* que encapsula os índices gerados pela indexação da Base de Dados. Em segundo lugar, foi desenvolvido o componente *QueryPresenter*, responsável por acessar os dados do modelo, formatá-los e enviá-los para a camada *View* do sistema. Nela foram implementados três pacotes.

O primeiro pacote, é o de consulta textual. Denominado *searcher*, esse pacote encapsula classes de Pesquisa de Índice, Analisadores Textuais, *Queries* de Pesquisa, Documento e Diretório da biblioteca[8], entre outros, com o intuito de facilitar a utilização da busca de Documentos em nosso índice textual. Assim, a pesquisa é executada de maneira simples. A classe principal é a *IdealizeSearch*, que deve ser instanciada e possui um método de pequisa, *search*, com parâmetros de entrada do texto consulta e campo a serem pesquisados (em nosso caso só existe o campo mensagem). Essa classe, ao ser instanciada, lê um arquivo de propriedades, definido, para iniciar as configurações de pesquisa desejada de maneira trivial, que são:

- Caminho do Índice que será pesquisado: O caminho de pesquisa pode ser passado tanto pelo arquivo de propriedades como também na própria instanciação do Objeto;
- Similaridade Textual: Ao executar uma consulta com distância de edição, qual o coeficiente de similaridade deve ser levado em consideração;
- Top N Documentos: Qual é a quantidade desejada de Documentos encontrados;
- Resultado Mínimo: Mínimo necessário para descartar o resultado com distância de edição, caso o processo ainda não tenha terminado;
- *Verbose*: Ativa ou desativa o log de inicialização do Objeto.

O algoritmo de pesquisa da *IdealizeSearch* funciona da seguinte maneira: Ao receber os parâmetros de pesquisa, são inicializadas duas *Threads*. Uma para pesquisa exata de termos e outra para pesquisa aproximada. Cada uma dessas *threads* passam por um interpretador que gera uma consulta entendida pelo *IndexSearch*, e é feito uma consulta em cada uma das *Threads*. A *Thread* principal fica aguardando o resultado das *Threads* de consulta. Quando a *Thread* de pesquisa exata retorna, é conferido se o Mínimo de resultado estabelecido é cumprido. Caso afirmativo, *IdealizeSearch*

retorna o resultado. Caso contrário, é aguardado o resultado da pesquisa por distância por edição, para que possam ser sugeridas possíveis consultas corretas ao usuário. Assim, após o resultado da *Thread* com pesquisa aproximada, há um algoritmo para identificar os termos resultantes de um documento que equivalem a consulta feita pelo usuário. Desse modo retornando esses termos dentro dos top 5 documentos da *Thread* de pesquisa aproximada, como uma possível sugestão para o usuário.

O segundo pacote, *cube*, foi desenvolvido e adaptado com a Base de Dados pelo próprio coordenador do projeto *JCubing*. Foi disponibilizada uma interface simples de consulta, chamada *Querier*, que recebe uma *String* de consulta e retorna um vetor de mapas<Classe,OpenLongHashSet> onde a Classe são os nome dos atributos e a OpenLongHashSet contém os Tweet Ids.

O terceiro e último pacote do componente *QueryPresenter* é responsável por fazer a combinação do resultado dos dois índices e formatá-los, para enviar ao componente *QueryView*. A classe principal por fazer essa interseção é a *IdealizeIntersection*, que recebe como parâmetros o resultado da pesquisa textual, mais o vetor de mapas que o *Querier* retornou, e qual o tipo de detalhe o usuário requisitou.

A partir desses parâmetros, acontece uma interseção e, por fim, há a geração de uma *String*, que possui um padrão, com separadores de campos, que será enviada para que a camada *View* possa interpretá-la e mostrá-la da maneira como bem entender.

Por motivos de entendimento, eis um exemplo de uma *String* para a camada superior: "1000%tweets;900;100", em que o "1000" significa quantidade de resultados encontrados na pesquisa textual e "tweets" como a classe de detalhamento requisitada pelo usuário, "900" sendo a quantidade de *tweets* positivos, a partir da pesquisa textual e "100" como a quantidade negativa de *tweets*, a partir da mesma.

Logo, tínhamos o ferramental completo para executar uma consulta. Entretanto ainda faltava um servidor que recebesse consultas e enviasse respostas. Assim, encapsulando as classes principais citadas, foi implementada uma Classe que estende a Classe *HttpServlet*, da biblioteca *Apache Tomcat*, para que fosse possível receber/enviar as requisições/respostas.

Para o componente *QueryView*, nessa primeira versão, apenas uma *HTML* simples foi implementada, com algumas funcionalidades simples de *script*. O usuário poderia consultar escrevendo somente um texto e era retornado para ele a quantidade de mensagens positivas e negativas, a partir de sua consulta.

A ferramenta tinha sua primeira versão, e logo apareceram problemas. Pela grandiosidade de dados trabalhados, consultas textuais demoravam cerca de 3 segundos, consultas multidimensionais, mais 10 segundos e a fase final de interseção, em torno de 2 minutos.

6.3 Implementação Versão 2

A versão dois viria a ser a versão um com uma interface gráfica mais amigável do que a primeira. O componente *QueryView* precisava ser mais que simplesmente uma página branca *web*. Assim, utilizando a biblioteca *JavaScript jQuery*, foram adicionadas várias funcionalidades à página, até então "estática". Baseado na Programação Orientada a Eventos, a implementação da camada *View* tem funcionalidades como exibição de gráficos e geração de uma página com os dados específicos de uma consulta, para, caso seja requerido, exportar as estatísticas para outro ambiente. Além do mais, foi

também implementado uma "Consulta Avançada" para o Usuário, que agora é capaz de especificar e combinar como "quiser" o resultado de sua pesquisa. Assim, de maneira mais intuitiva, o usuário pode fazer requisições de pesquisa dinâmica e seu resultado já vem disponibilizado em dois tipos de visões, gráficos e textos. Houve uma tentativa de se utilizar a biblioteca *Google Maps* porém, com alguns problemas e tempo, foi deixada para segundo plano.

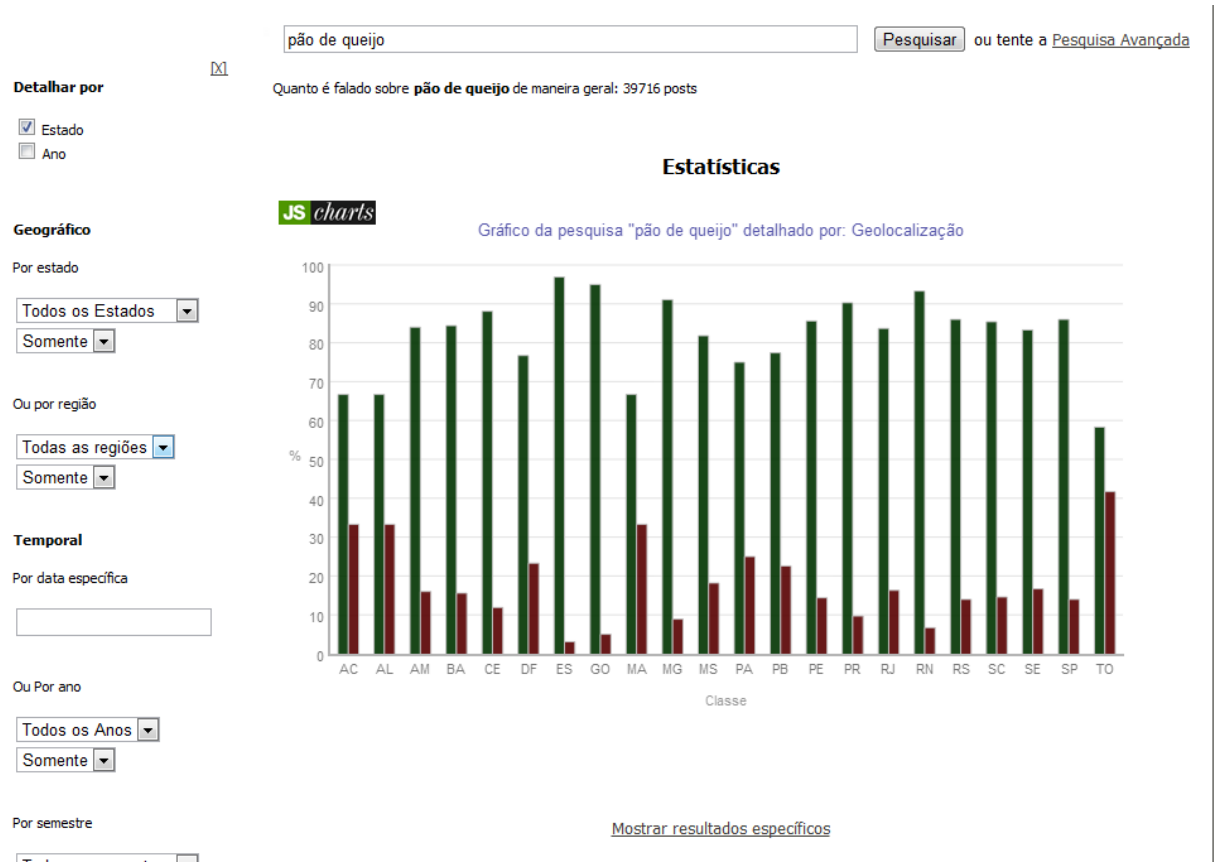

Figura 2: Interface Gráfica da Ferramenta

6.4 Implementação Versão 3

Por último, após a implementação da camada *View* do sistema, era hora de otimizar todo o processo. O primeiro passo a ser dado era descobrir o que mais atrasava o processamento de uma consulta. Ao perceber que a fase de interseção dos resultados estava fazendo acesso ao disco rígido para resgatar os Tweet IDs, foi tomada a decisão de que a classe *IdealizeSearch* seria provedora de uma *cache* com todos os Tweet IDs, em Memória Principal, já que o índice textual atualiza constantemente seus ids primários, por motivos de desempenho.[8]. O segundo passo foi paralelizar todo o processo de consultas. O *Servlet*, ao receber uma consulta, cria uma *Thread* para a consulta textual, outra para a consulta do cubo multidimensional e aguarda o resultado das duas. Assim que o resultado chega, é iniciado o processo final de interseção e formatação das estatísticas. Todo esse processo foi paralelizado. Assim, uma *Thread* é inicializada para cada mapa do vetor resultante da consulta do cubo. Dessa forma, cada *thread* faz sua

interseção, classifica e retorna seu resultado. Ao fim de todas as *threads*, simplesmente são agregados todos os resultados de cada *Thread* intersessora.

Foram obtidos, portanto, resultados satisfatórios com os tempos de boa parte das consultas. Estas, que antes demoravam em torno de 2 minutos (e até mais), agora giram em cerca de 10 segundos. Dessa maneira foi finalizada a terceira versão do sistema.

7 Trabalhos Futuros

Como Trabalhos Agendados, podemos citar o módulo ainda não implementado, *Cache*, e o módulo ainda não finalizado, *Crawler*. A *Cache* será um módulo de acesso rápido que terá algumas políticas de armazenamento de resultados as quais poderão ser configuráveis. Será desenvolvida a partir da biblioteca Java JCS, *Java Caching System* disponibilizada via Licença *Apache*. Já para o módulo *Crawler*, será desenvolvido um analisador de mensagens, com o fim de tornar possível classificá-las como positivas/negativas com uma heurística mais trabalhada, para poder classificar um maior número de mensagens e com mais precisão do que a que foi empregada em [5]. Após isso, a integração dos dados coletados e analisados é um importante passo a ser dado.

Podemos contar com a visualização de mapas, num futuro próximo, pois estudo de APIs Web para mapas se encontra em andamento.

Ademais, o último requisito da Tabela 1 é um trabalho futuro. Desenvolver um sistema de cadastro de usuários, que podem salvar pesquisas e mantê-las armazenadas, que funcionaria, analogamente, como o módulo de Cache.

Também podemos citar, como trabalhos futuros mais distantes, pesquisas na área de Análise de sentimentos para criação de melhores Analisadores de Polaridade das mensagens e "Predição" de Eventos.

Referências

[1] Micah Alles, David Crosby, Brian Harleton Carl Erickson from Atomic Object, Greg Pattison Michael Marsiglia from X-Rite, and Curt Stienstra from Burke Porter Machinery. Presenter first: Organizing complex gui applications for test-driven development. *Proceedings of Agile Conference*, 2006.

[2] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. *Article of New York Times November 22*, 2011.

[3] Fabrício Benevenuto, Jussara M. Almeida, and Altigran S. Silva. Explorando redes soociais online: Da coleta e análise de grandes bases de dados às aplicações, 2011.

[4] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. *IMC '09 Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 09, 2009.

[5] Fabrício Benevenuto, Diego Silveira, Thalisson Oliveira, Leonardo Bombonato, Reinaldo Fortes, and Álvaro Pereira Jr. Entendendo a twittesfera brasileira. *SBSC*, 2011.

[6] Johan Bollen, Alberto Pepe, and Huina Mao. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[7] Joubert de Castro Lima. Jcubing at www.joubertlima.com.br, November 2011.

[8] Erik Hatcher and Otis Gospodnetic. *Lucene in Action*. Manning, 2 edition, 2011.

[9] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[10] Nielsen Online. State of the media: The social media report q3 2011. Technical report, Nielsen/McKinsey Company, 2011.

[11] Tiago Rodrigues, Fabrício Benevenuto, Meeyoung Cha, Krishna P. Gummadi, and Virgílio Almeida. On word-of-mouth based discovery of the web. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2009.

[12] Matthew A. Russel. *Mining the Social Web*. O'Reilly, 1 edition, 2011.