University of Glasgow | School of Computing Science

# Pulse: A Social Media Analysis Toolkit

Paul Holmes

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 25th 2011

# Abstract

An explosion in the usage of social media applications such as Twitter has created a wealth of data well-suited to mining and analysis. With the correct tools, this data can be analysed and interpreted in a meaningful way — identifying trends, establishing opinions and understanding usage.

This dissertation presents *Pulse*, a social media analysis toolkit. It is capable of processing tweets streamed directly from Twitter, managing their storage and indexing, analysing each for aggregated statistics and presenting the results in a browser-based application. Users can search using temporal or subject-based queries, and are presented with components that display information relevant to a particular time, country or subject. The results themselves are presented through a number of means: as text, graphically, utilising chart visualisations and through use of maps.

Many social analysis tools already exist, as desktop applications and websites. However, few provide such comprehensive analysis. Existing tools are limited by the volume of data they utilise, the domains in which they operate, and the features which they implement.

Pulse encapsulates millions of tweet per day as subject trends, geographic hotspots, topic opinions and more. The user application presents this data as components; units of functionality that provide specific analysis of a given property, each of which can be modified to suit user requirements and extended. Should new requirements arise, additional components can be created from scratch and easily integrated into the existing application.

## Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: ____________________ Signature: ______________________

## Acknowledgements

I would like to thank my level 4 project advisors, Dr Iadh Ounis and Dr Craig Macdonald, for their continued support, advice and endless patience.

# Contents

# Chapter 1

# Introduction

This document describes Pulse, a toolkit for streaming and analysing large volumes of data from Twitter, a social networking website. This chapter describes the problem definition, motivations for solving such a problem, and the aims of the system to be developed.

## 1.1 Problem Definition

The volume of data being created by Twitter users is incredibly large, and access to archived data is limited. Additionally, few utilities exist that provide the ability to analyse Twitter posts *en masse* in great depth over multiple variables. Being able to study this data, mining it to understand trends and topics of note at a particular time, would be useful in many commercial and academic fields.

Several systems already exist which provide some level of insight into tweet trends. However, as discussed in Section 2.2, most of these systems consider only one particular aspect of the Twitter domain, provide superficial statistics, or do not give any indication of how information being used is gathered and processed.

The project consists of two major parts: constructing an ‘always on’ application which can stream and process the required data, and constructing a user application which can display and manipulate the analysis results in a meaningful way.

## 1.2 Motivation

Creating a system such as Pulse is interesting for several reasons. The sheer volume of data generated on a daily basis by Twitter is huge (around 100 million tweets per day). Developing a robust, reliable and stable application which is able to handle this amount of data is an exciting technical challenge.

Meaningful Twitter analysis also has important practical applications, in areas such as marketing, sales and support. As its popularity grows, companies are becoming more prolific in their use of Twitter for customer engagement and brand management. One example of this is Microsoft, who have utilised Twitter as a tier 1 customer support mechanism. Multiple staff operate the username '@XboxSupport' [9] , which provides proactive and reactive customer support to users having technical issues.

By monitoring mentions of their brand names, responding to user complaints directly and tracking opinion within tweets mentioning their products, they have managed to significantly reduce the number of support calls to their contact centres, ensuring customer satisfaction is high as well as reducing support costs [2]. Providing a toolkit that would allow organisations to go more rapidly from a passive presence on Twitter to an active one would allow them to reap the same benefits.

Twitter is also a domain featuring unusual social relationships and ever-changing trends. Few tools exist that allow more than superficial analysis of the domain, therefore creating an innovative system able to explore and visualise the Twittersphere in a way that highlights changes, patterns and cycles through discovery is itself a motivating factor.

## 1.3 Pulse System Aims

The primary aim of Pulse is to allow for in-depth analysis of data gathered from Twitter through an engaging user application.

From a system perspective, Pulse should be capable of gathering a substantial volume of Twitter data for analysis. This involves streaming data from the relevant data sources, storing this data, indexing it for later retrieval and performing batch analysis of the data.

From a user persepective, Pulse should allow for extraction of meaningful insights from the data analysed. It should present results in a user-friendly and manageable way, and allow users to navigate results in both a directed and undirected fashion. The information presented to the user should give insight into a particular topic or time that otherwise would be difficult or impossible to discover.

Pulse should also be extensible. Adding new components to display information in a different way should be possible for advanced users. Similarly, existing components should be organised in such a way that modifying their functionality is trivial.

## 1.4 Dissertation Outline

The rest of this document describes the development process of Pulse. It is structured as follows:

- Chapter 2 explains the purpose of the system, as well as the aims to be achieved in creating Pulse.
- Chapter 3 describes the application requirements, expected functionality and desirable features, as well as explaining the overall design of the toolkit, describing the core technologies utilised and the application architecture.
- Chapter 4 discusses the data processing system powering Pulse.

- Chapter 5 describes the client-side aspect of the Pulse user application, including a detailed description of each component.
- Chapter 6 describes the server-side aspect of the Pulse user application.
- Chapter 7 explains the rationale, process and results of the Pulse system user evaluation.
- Chapter 8 summarises the successes and challenges of the project, as well as describing future development opportunities.

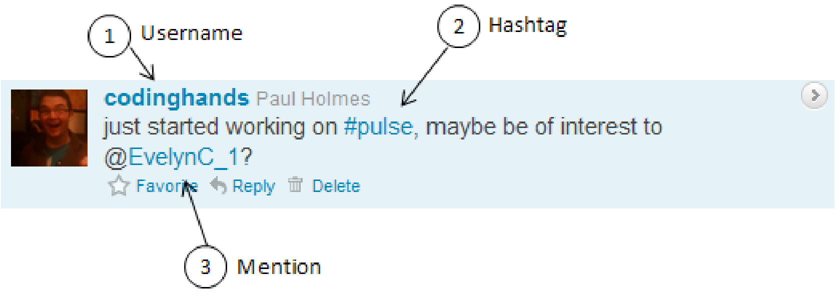
# Chapter 2

# Background

## 2.1 Twitter: A Microblogging Service

The primary data source of Pulse is Twitter. Twitter is a microblogging platform that allows users to post small updates — 'tweets' — typically answering the question *"what's happening?"*. Within each tweet users can post text as well as meta-information and links to external content. Having started operation as 'twttr' in 2006 [1], it has since grown to be ranked 10th worldwide in the Alexa Traffic Rankings [10] of most visited websites. Whilst Twitter has not disclosed the current number of users utilising the service, in April 2010 it was announced that there were 105,000,000 registered users. [8].

### 2.1.1 Tweets



codinghands Paul Holmes
just started working on #pulse, maybe be of interest to @EvelynC_1?

Figure 2.1: A typical tweet, taken from a user timeline

Tweets are the primary communication medium within Twitter. Consisting of a 140 character message, each tweet is listed on the tweeting user's profile page, as well as in the stream of each of that user's followers. The technical definition of a tweet as indexed and used by Pulse components is described in Section 4.4.

### 2.1.2 Social Relationships

Unlike most social media outlets, Twitter has two main relationship structures that allow users to share tweets. Rather than providing a reciprocal 'friend' mechanism that validates both parties as equals, users can be followers of and / or followed by any another user. As long as a user's tweets are public, that user can be followed by anyone. However, the user being followed is under no requirement to follow the requesting user in return.

Following another user automatically adds that user's tweets to your timeline. The timeline is the default page displayed once login to the Twitter site is complete. It lists the most recent tweets of a user, integrated with the most recent tweets of all that user follows. As new tweets are posted this list is automatically updated. Similarly being followed by a user means that your tweets are displayed in their timeline.

### 2.1.3 Tweet Terminology & Usage

**Hashtags**

Though a tweet can only contain text, users can mark particular words as 'hashtags'. Each hashtag is prefixed by a hash symbol, as shown in Figure 2.1. Hashtags allow users to categorise their tweets and signify to other users that tweet's content. Rather than a static hashtag list existing, a hashtag is implicitly created after its first use within a tweet. Futhermore a tweet can contain as many hashtags as allowed within the character limit. Twitter uses hashtag information as one variable in determining trends - that is, the most popular topics on Twitter.

**Mentions**

In addition to hashtags, tweets can also contain 'mentions'. These are explicit references to another user within the tweet. References can be identified by the @ symbol prefixing a user name. Mentions are another way of adding meta information to a tweet, signifying that the tweet in some way concerns the user being mentioned. The user being mentioned will see this tweet under the '@Mentions' tab of their profile. Any user can be mentioned in any tweet, regardless of the users' relationship.

**Retweeting**

'Retweeting' is the central mechanism within Twitter for allowing users to rebroadcast a tweet. Retweeting a tweet results in a copy of tweet being posted by the retweeting user. Tweets can be retweeted multiple times, with particularly popular retweets being factored into Twitter's list of trending topics.

## 2.2 Existing Analysis Systems

A large variety of tools are now available which offer a particular insight into the Twitter domain. A selection of these were reviewed to establish the capabilities of existing systems, and also to identify functionality which is poorly represented.

Twitter analysis tools can be broadly divided into five categories:

- *User tracking tools*: these tools provide individual users with statistics about their own account and Twitter usage. Simplistic capabilities such as tweets posted and typical Twitter usage patterns are supplemented with more advanced features, such as graphing of a user's connections network, or recommendations of who to follow on Twitter based on the user's current network.
- *Twitter-specific interactions analysis*: many tools offer some view of how prevalent particular Twitter-specific interactions are; for example, the most popular retweets being posted, daily user rankings, or commonly posted links.
- *Self-promotion tools*: certain tools provide specific functionality to aid users in attracting more followers. These vary from simple tools that highlight which of a user's followers are not reciprocating, to more advanced tools that automatically follow users with low follower counts, message users with promotional material and recommended users to follow based on their reciprocation rate.
- *Brand marketing tools*: commercial tool packages are available which let businesses use Twitter to promote their brands, analyse consumer opinions and track product mentions.
- *Trend tracking tools*: applications which allow users to track particular topics over time, displaying how usage changes and highlighting times where usage deviated from what would be typically expected.

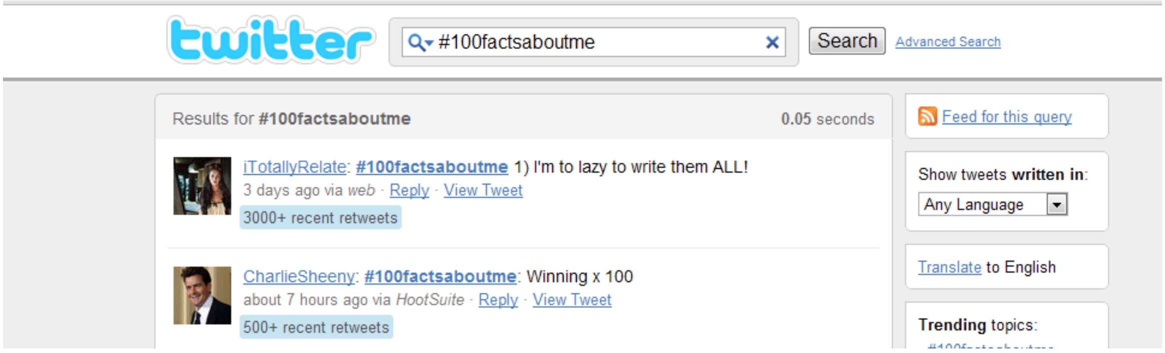
This project is not focused on user tracker or aiding user self-promotion, so analysis of sample applications in these categories has been omitted.

Of the tools categorised four were chosen for deeper analysis to understand Twitter analytical capabilities generally available, and also to identify features not currently common amongst tools.

### 2.2.1 Twitter Search

Twitter Search is shown in Figure 2.2. Twitter Search can be used to query what's happening on Twitter in real time, with the results page automatically updated as new tweets are posted.

Twitter Search is an interesting tool for casually exploring popular topics at a point in time. It does, however, have its limitations. Queries can only be issued for roughly the last month of Twitter activity — all older data is archived and inaccessible from the site. This limitation restricts the amount of trend tracking that can be performed to only the very recent past, which for some queries is not sufficient.



twitter #100factsaboutme Search Advanced Search

Results for **#100factsaboutme** 0.05 seconds

iTotallyRelate: **#100factsaboutme** 1) I'm to lazy to write them ALL!
3 days ago via *web* · Reply · View Tweet
3000+ recent retweets

CharlieSheeny: **#100factsaboutme**: Winning x 100
about 7 hours ago via *HootSuite* · Reply · View Tweet
500+ recent retweets

Feed for this query

Show tweets **written in**:
Any Language

Translate to English

**Trending** topics:

Figure 2.2: Twitter Search

Furthermore the Twitter Search application only provides a single, 'raw' view of Twitter activity. Though searches can be made more complex by using boolean operators or emoticons they still only return a list of tweets. No information is given on how ranking is performed, so the results themselves may be subject to internal Twitter filtering.

### 2.2.2 TrendsMap

TrendsMap [5], shown in Figure 2.4 is a geographically focused tool which shows trending tags in any location across the world. Tags prevalent in particular locations are plotted on the map by checking the user's location. Each tag is then sized relative to its popularity. Clicking on any tag generates a real-time update stream of tweets featuring the tag.

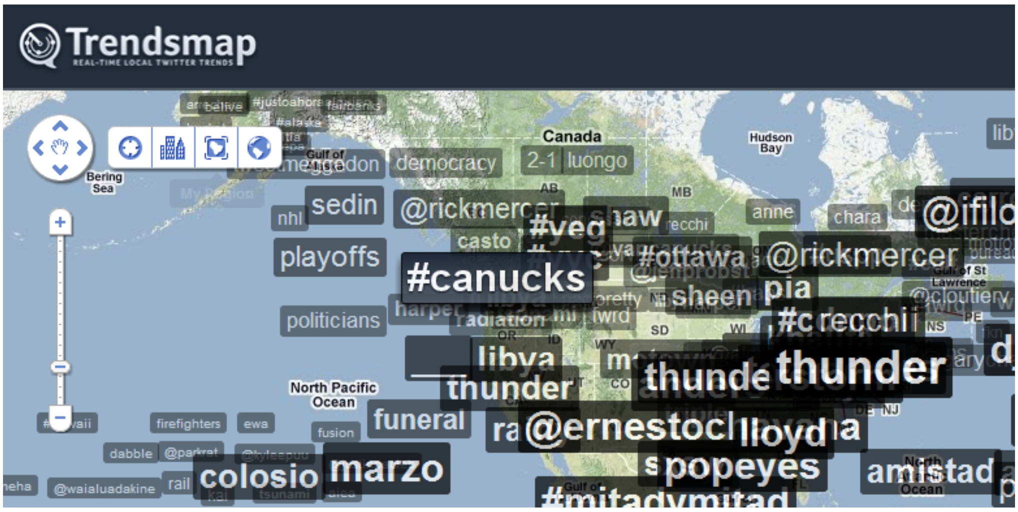


Figure 2.3: TrendsMap

TrendsMap is particularly good for allowing user exploration, as every tag is connected to a stream of tweets which contain clickable usernames and tags. However, it is not possible to see how trends changed over time, nor view any aggregations of trends from over a period of time. As an instantaneous snapshot of Twitter trends Trendsmap provides some excellent features, however in terms of historical analysis or trend comparison it provides very little.

### 2.2.3 Trendistic

Trendistic [4] provides more traditional analysis tools, charting tag usage over various time periods. The site can be operated in two ways: by typing a query tag in manually or by clicking on one of the suggested ‘trending tags’. A graph is then generated, visualising number of occurences over time.

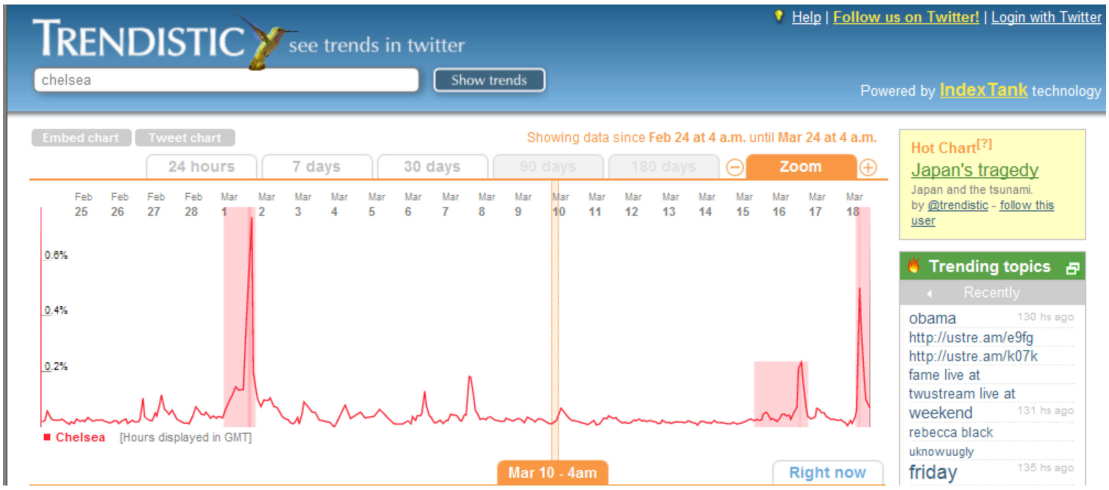


Figure 2.4: Trendistic

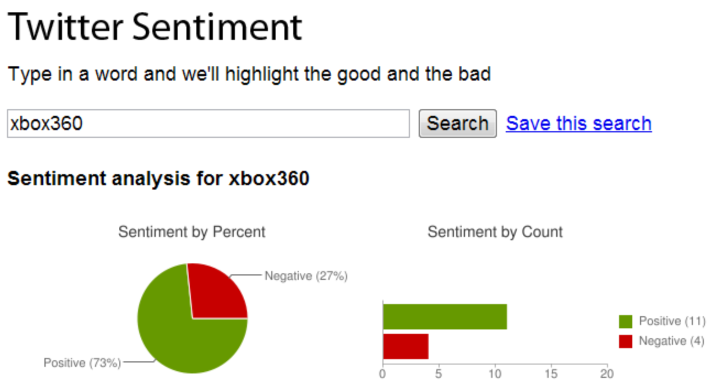
The functionality provided by Trendistic is certainly useful, but also fairly limited. The data presented cannot be interacted with or pivoted in any way — only the graph of usage over time is provided. Cursory testing with the Trendistic site also highlighted some functional issues. Often tweets failed to load and the interface does not provide any assistance or guidelines to users.

Tag usage over time is certainly an important measure of topic popularity, but without supplemental information to place this quantitative data into context, or even a way to compare multiple against each other, there are few practical uses for the site beyond general interest.

### 2.2.4 Twitter Sentiment

Twitter Sentiment [7] is a sentiment analysis tool developed at Stanford University, shown in Figure 2.5. It uses machine learning techniques and a maximum entropy classifier trained with a corpus of tweets featuring certain emoticons [49] to classify tweet text as either ‘negative’ or ‘positive’.

Twitter Sentiment is perhaps the most interesting tool available for analysing Twitter output. Understanding tweet sentiment has many practical uses, such as brand management, polling and assisting user decision making. Developing an effective sentiment analysis process would give Pulse a method of ranking topics (or brands, people, places. . . ) by establishing sentiment. This could then be used to inform decision making (“will I buy X or Y?”) or illustrate feeling towards a topic (“I hate / love X”).


Twitter Sentiment

Type in a word and we'll highlight the good and the bad

xbox360

Sentiment analysis for xbox360

Figure 2.5: Twitter Sentiment

### 2.2.5 Requirements Elicitation

From the tools reviewed it is clear that most consider only Twitter data from the recent past to be relevant. Almost all of the services reviewed use either live results instantaneously requested from Twitter, or a history of at most one month. This is potentially an area where Pulse could specialise, providing much longer-term trend tracking and analysis — especially for terms which are almost always trending (certain users, brand names, companies etc).

Most of these tools are also only focused on one particular aspect of the Twitter domain. Pulse should provide a more broad, comprehensive overview when queried by presenting data to the user in different ways, with each data view highlighting a particular facet of the result set. However, developing multiple visualisations should not be at the cost of accuracy or reliability each individual data view.

Sentiment analysis tools were also reviewed, and based on the satisfying and informative user experience had with the Twitter Sentiment tool Pulse should ideally implement some basic form of sentiment analysis.

## 2.3 Conclusion

In this chapter Twitter was introduced as a social networking platform with its own distinct terminology, communication paradigms and constraints. A series of tools which can be used to analyse output from Twitter were discussed, highlighting particularly useful features and identifying application weaknesses. These discoveries will be used to help form requirements for the system.

In the next chapter the problem domain is analysed, and a design for Pulse is proposed.

# Chapter 3

# Analysis & Architecture

Having reviewed available Twitter analysis tools the next stage in the project development process took place — problem analysis and system design. In this chapter we discuss the high-level requirements of the project, introduce the two major subsystems, describe the overall architecture of the toolkit, and evaluate numerous technology options available for each subsystem.

## 3.1 Application Requirements

Requirements for the application were elicited through discussion with the project supervisors and by analysis of existing tools performed in Chapter 2.

Each requirement was categorised using the MoSCoW prioritisation technique as proposed by Clegg [47]. The necessity of each requirement was decided by considering its impact on achieving the system aims. Those vital for success became 'must haves', those which are necessary but need not be fully implemented became 'should haves', those which would profoundly improve the system but were not absolutely necessary became 'could haves', and finally those which would be advantageous but not necessary became 'would like to haves'

The finalised MoSCoW categories are listed below.

- *Must Have*
  - A robust data gathering application capable of managing the work involved in streaming millions of tweets into a easily retrievable storage format.
  - An application accessible in the browser that allows users to query the stored data in a usable, engaging way.
  - A mechanism allowing users to search by topic or hashtag. The results returned should give a comprehensive overview of that tag's usage from geographical, temporal and magnitudinal perspectives.
  - A series of visualisations providing analysis of hashtag query results. Envisioned examples included opinion analysis, a map of tweets plotted by location and a cloud of co-occurring hashtags.

  - A significant amount of historical tweet data for long-term temporal range queries.

- *Should Have*
  - A mechanism allowing users to search across a temporal range. The results returned should give an overview of Twitter activity across the temporal range.
  - A series of visualisations providing analysis of temporal range queries results. Envisioned examples include a trending topics graph, the most popular links over a day, and aggregate daily Twitter sentiment.
  - Support for queries to be compared.

- *Could Have*
  - Pivot data visualisations, where a set of information can be 'pivoted' on different variables to highlight specific properties of the data.
  - Caching performed by the client, removing the need for users to make unnecessary query requests trying to search for the same data twice.
  - A mechanism allowing users to search for an individual user. The results returned should give a complete picture of that user's Twitter activities — social connections, level of engagement, influence and usage tendencies.

- *Would Like to Have*
  - An mobile version of the application for use on small form-factor devices.

*N.B.* — These requirements apply to the system as a whole, each directly related to the aims of the project. Subsystems have their own internal requirements, defined in their respective chapters.

## 3.2 Core Technologies

The requirements set out in Section 3.1 define what Pulse must be able to do. These were used to select particular technologies which would most easily and quickly support building the required functionality. Early in development it became clear that Pulse would require two main subsystems: one for data gathering (the 'data processing application'), and one for users allowing them to interact with the data (the 'user application'). This section describes the core technologies evaluated and gives justification for the technologies selected.

### 3.2.1 User Application Framework

The most significant engineering work undertaken during the project was the creation of a user application able to handle search queries and respond with appropriate data visualisations.

The following factors were considered whilst analysing candidate frameworks:

- *Maturity*: the framework should be well-documented and stable.

- *Ease of Use*: functionality in Pulse will be added at a rapid pace during the development process. The selected framework should aid this iterative process with debugging tools, useful libraries and a quick write-test-deploy cycle.

- *Portability*: the final application should be runnable across multiple platforms, and the ideal framework will innately support cross-platform development.

- *Framework Weight*: the framework should not require significant resources to operate, and should be flexible enough to allow integration of non-framework extensions if required.

- *Deployment*: deploying the application should be a simple process requiring little additional software or hardware beyond the framework itself.

- *Communication*: the user application needs some way to retrieve data for each query. The framework used should provide a thorough, robust communication mechanism for this purpose.

Several frameworks and technologies were initially reviewed to understand which would best suit development of Pulse. It quickly became clear that a web application, rather than a traditional desktop application, was the best way to proceed. Browser-based applications remove the need for users to install any software, provided direct access to a well-defined presentation markup language (Cascading Style Sheets, or CSS [40]), and naturally fit the data domain — Twitter itself a pioneering web application. Furthermore Twitter provides tweet data in JavaScript Object Notation format, a data interchange type which browsers are extremely fast at parsing. The data format and its implications are discussed fully in Section 4.4.

### jQuery

The first technologies considered were JavaScript libraries such as jQuery [25], which simplifies client-side HTML scripting in web applications, as well as providing a series of rich interaction widgets. jQuery is remarkably lightweight (a 22Kb file which can be included using one line of HTML) and has a large number of pre-defined, easily implemented complex interaction behaviours with are traditionally challenging to emulate in the browser. It also supports AJAX (Asynchronous JavaScript & XML) server requests with a series of useful abstractions masking cross-browser idiosyncrasies.

jQuery is a mature platform, used by 43% of the 10000 most frequently visited websites worldwide [26]. It has extensive documentation and an active support community which would be invaluable during development should implementation issues arise. jQuery is also easily deployed: the library needs only to be uploaded to a web server.

Whilst jQuery greatly simplifies JavaScript development, it does have its limitations. jQuery is effectively a library of helper methods which augment and support other frameworks, rather than providing an all-encompassing structure in which applications can be built. jQuery would have to be integrated into a larger development model (such as one using PHP on a server) to actually create a fully-fledged application.

Development in raw JavaScript is also troublesome regardless of libraries used. The only tools available for JavaScript development are browser-based debuggers such as Firebug [16] which inspect code at run time. Unfortunately there are very few satisfactory development aides when writing JavaScript that provide anything like the support which IDEs such as Eclipse or Visual Studio provide. Given the complexity of the proposed application it would be preferable to write in a language with more advanced development tools and utilities.

Other JavaScript libraries evaluated which were very similar to jQuery included YUI (Yahoo User Interface library) [42] and MooTools [29]. Each provides JavaScript abstractions and interface widgets, but neither has a overarching structure in which complex applications can be built.

### Google Web Toolkit

Google Web Toolkit (GWT) is a web development toolkit which lets developers quickly build complex web applications. The most interesting feature of GWT is the manner in which applications are written. GWT features a Java to JavaScript compiler that allows developers to write code in Java, which on deployment is compiled to highly-optimised JavaScript.

The compiler supports a significant subset of the Java language (with the obvious exclusion of multithreading and concurrency), and can create different application code permutations aimed specifically at particular browsers. This approach would significantly reduce the effort required in ensuring Pulse is portable across platforms and browsers.

Furthermore Google supplies Java wrappers for several of its most popular JavaScript APIs. These include the Maps API (for mapping and location translation), the Visualisations API (a suite of charts, graphs and other customisable data visualisations), and the Translation API (for language detection and translation). All of these would be immensely useful for representing retrieved data to the end user.

Although development in GWT is in Java it is still possible to include native JavaScript. GWT features a JavaScript Native Interface for calling code written in pure JavaScript (analogous to the Java Native Interface for calling code written in C/C++). This means both external libraries and handwritten optimised JavaScript can be arbitrarily mixed with Java.

Server communication is conducted using remote procedure calls to Java servlets. These are defined within GWT as ‘services’, compiled to conventional Java byte code instead of JavaScript. This means GWT applications are often deployed as servlet containers onto Java web servers [15], such as Apache Tomcat [14].

There were, however, certain drawbacks to using GWT. The framework itself is still in its infancy. This can be seen both in the level of documentation provided for certain features (often none, or at best a sparsely-populated Javadoc description), and the relatively low level of developer support compared to lightweight options such as jQuery.

Furthermore GWT has a very rigid structure that strictly defines the application. Though perhaps useful early in the project whilst developing prototypes there is a risk that tight structure could become a hindrance once more experienced with the framework.

### 3.2.2 Framework Selection

Ultimately GWT was selected for use as the user application framework. Being able to write web application code in Java (and all of the development advantages this entails) meant the prototype could be turned around very quickly. A solid communication structure is included, and production deployment is a trivial process. Full details of user application development are provided in Chapter 5.

### 3.2.3 Storage & Retrieval

Having decided on a framework to structure the user application it was necessary to consider how data would be streamed and stored from Twitter. This primarily involved reviewing available retrieval systems and comparing them against the following attributes:

- *Scalability*: Pulse requires millions of tweets each day to function, each of which must be indexed and stored. The platform chosen must be able to scale well to this volume of data, in storage, retrieval and indexing.
- *Retrieval Functionality*: Data must be accessible in a way that is conducive to useful search; that is, a well-defined query language should be available that provides the client application with all required data.
- *Retrieval Efficiency*: Accessing data from the client application must be fast enough for real-time search, even with thousands of results. The retrieval mechanism must also be resource-light.
- *Retrieval Effectiveness*: how balanced is the total resources required by the platform, relative to the platform's performance?
- *Required Infrastructure*: The chosen platform must be able to run on standard hardware available to the university.
- *Platform Maturity*: The chosen platform must be stable for regular intensive use, and have sufficient documentation for setup, configuration and troubleshooting.

Two options were considered for storage and retrieval of data in Pulse:

- *HBase* - an open-source, distributed, versioned, column-oriented store built to hold very large amounts of data over clusters of machines using standard hardware [12].
- *Terrier* - a flexible, efficient search engine developed at the University of Glasgow for indexing large-scale document collections [56].

To establish which was better suited to powering the Pulse data processing application each was compared against the above criteria.

### HBase

HBase is a “an open-source, distributed, versioned, column-oriented store” . It is *distributed* in that it is built upon a distributed file system, Apache Hadoop. It is *versioned* in that each table is three-dimensional, where the third dimension is time. HBase can be configured to store previous versions of a row for later retrieval.

Table 3.1 shows the analysis conducted for HBase.

| Criterion | Analysis |
|---|---|
| Scalability | HBase is built upon a distributed file system explicitly designed for very large data sets. Scalability is one of the strongest reasons for selecting HBase as the application platform. |
| Retrieval Functionality | The default HBase retrieval mechanism involves scanning each table row by row. The scanning process can be filtered to exclude certain rows based on columns or column families, but scanning large tables still takes a non-negligible amount of time – too long for use in real-time. A secondary layer then would need to either cache or index HBase results. |
| Retrieval Efficiency | Scanning tables takes a disproportionately large amount of computation to achieve fairy simplistic functionality. |
| Retrieval Effectiveness | Scanning each table is a laborious process not suited to real-time retrieval. A separate retrieval layer would be needed to implement real time data retrieval |
| Required Infrastructure | HBase requires the Hadoop distributed filesystem to operate. This in turn requires a cluster of machines to act as nodes. The university does have a cluster available which can be used which already has HBase, however the reliability of this infrastructure is unknown. |
| Platform Maturity | HBase has yet to reach a complete 1.0 release, and so is effectively still in beta development. That said there are several active support outlets (mailing lists, forums etc.) staffed by users with length experience. However as the potential platform for a central part of Pulse HBase is an unknown quantity. Incompatibilities, errors and other implementation concerns which apply to ‘1.0’-type applications are all possible challenges. The documentation for HBase however is fairly complete and comprehensive. |

Table 3.1: HBase Suitability Analysis

### Terrier

Terrier provides indexing and retrieval of large scale document collections in a highly-customisable manner.

Table 3.2 shows the analysis of Terrier against each criterion.

| Criterion | Analysis |
|---|---|
| Scalability | Terrier is highly scalable, designed to index millions of documents in a time-responsive manner. It can also be used in conjunction with the Apache MapReduce framework to perform distributed indexing |
| Retrieval Functionality | Terrier provides a simple query language (more than sufficient for the needs of Pulse) which can be used to query any Terrier index. Initial testing showed that even for large document sets response time was fast. However, testing with very large indices was not attempted. Terrier Collection and Document classes are easily extended to cover tweets. |
| Retrieval Effectiveness | Initial tests of Terrier running in conjunction with a simple streaming application showed satisfactory performance whilst not hindering the streaming process in any way. |
| Required Infrastructure | Aside from the Terrier application and available hard disk space, none. |
| Platform Maturity | Terrier is currently in version 3, with a planned further release this year. Terrier is widely used within the research community for non-trivial search, indexing and retrieval projects. Additionally the lead Terrier developer is also a project supervisor. Comprehensive documentation is provided online |

Table 3.2: Terrier Suitability Analysis

### 3.2.4 Retrieval Platform Selection

The initial version of the application used HBase as a data store for tweets, with Terrier providing indexing capabilities. However as feared the stability of the cluster was not sufficient for the 'always on' application and so HBase was discarded as a storage option.

From there Terrier was the obvious choice to power the data processing application in both storage and retrieval. In addition to being highly customisable and scalable (coping with millions of documents per index) it is also well-tested, stable and mature.

## 3.3 System Architecture

Pulse is comprised of 5 major components. The data processing application is responsible for obtaining, storing and enabling later tweet retrieval. It should be an 'always-on' application, downloading data without a human operator being required. The application should also be robust and fail gracefully on error, maintaining state and being able to pick up from where it failed. The full development process of this application is documented in Chapter 4.

The Application Data Store is the central repository for storing all of Pulse's data files. Whilst all Pulse components may freely access this folder, the user application must utilise a server-side service to retrieve results the Terrier indices. The structure of this directory, as well as an explanation of its contents, is also provided in Chapter 4.

The User Application is the front-end in which users can make queries, view results and manipulate

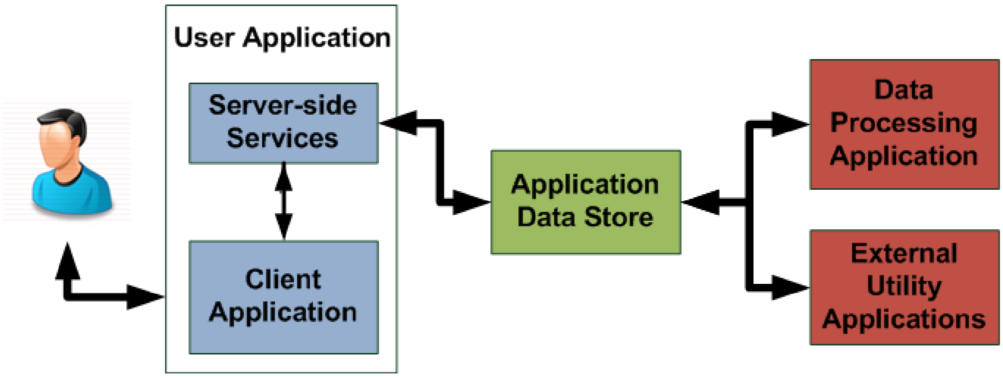


Figure 3.1: System Architecture

data, operating on a two tier client-server model. Requests for data made by the user are passed from the client application to a server-side service, which processes the request, retrieves the necessary data from the Application Data Store, and returns it to the client. A detailed description of the client application, from inception to testing, is given in Chapter 5. A detailed description of each server-side service is given in Chapter 6.

Other applications may be needed to amend, update or cleanse the streamed data, each requiring periodic access to the data store. An example application which aggregates daily data in this way is described in Section 4.9.

## 3.4 Conclusion

In this chapter a series of requirements was defined against which Pulse can be compared at the end of the project. These requirements encapsulated required functionality, as well as properties the final system should exhibit.

Each of the requirements can be broadly categorised as mainly involving user interaction or data processing. From this the need for two distinct subsystems was established. The first is a browser-based user application for querying and manipulating stored Twitter data. The second is a robust data gathering and processing application which can store tweets in an easily retrievable way.

Before a user application could be created a significant volume of tweet data was required. The data processing application, described in the next chapter, is responsible for the entire data gathering process — from initial download of each single tweet, to daily indexing of millions of tweets.

# Chapter 4

# Data Processing

A significant volume of data was required in order to perform meaningful data analysis in the user application. Developing a robust application to manage streaming, parsing, storing and indexing large volumes of tweets in real time was one of the key development challenges of Pulse. This section describes the approach taken, issues faced, and the final application.

## 4.1 Data Processing Requirements

As the primary data source of Pulse, the data processing application has ultimate responsibility for the quality, volume and integrity of all tweet data (and hence, all results derived from this data).

From a functional perspective the main requirement of the data processing application was to stream tweets from Twitter and store the data within a format that could be later accessed and queried by the user application. The process of loading, querying and retrieving data also needed to be fast enough for real time usage. Furthermore the format used must be organised in a way that naturally suits the various query types.

Whilst streaming tweets the application must stay synchronised with the rate at which data arrives and not build up a backlog. As there are around 120 tweets per second to be processed the application must perform all parsing and processing in an extremely small time window. This influenced the choice of libraries used, the methods in which data was handled, and the level of concurrency utilised.

Certain queries also require access to aggregated statistics of a given day. The application must be able to generate these statistics in some way and integrate the analysis results into the wider Pulse data directory.

From a non-functional perspective the application must be able to recover from failures and handle errors appropriately. The scope for unexpected errors is large, so ensuring the application can gracefully deal with dropped connections, Twitter outages, rate limiting, or any other eventuality is paramount.

With the application streaming tweets all day every day it should also operate with minimal user supervision. Logs with appropriate messages, warnings and errors should be provided to verify application stability, troubleshoot errors and preempt more serious failures.

Integrity checking the data streamed is also necessary at all points in the application. This includes validating format from the stream, catching malformed content during parsing and ensuring proper and correct storage within the data structures utilised.

## 4.2 Application Design

With the requirements above in mind the application went through two major design iterations:

1. *HBase-centric Store*: HBase used as the central data store for all tweets. Retrieval to be through HBase scans or Terrier indices.
2. *Finalised Terrier-centric Application*: Terrier used as both storage and retrieval mechanism, utilising a concurrent indexing pipeline.

Initial development of the application began in tandem with the client application. At this time the primary data store was HBase (see Section 3.2.3 for further discussion). As a rapidly developed prototype the highest priority objective was on storing tweets in a way that could be retrieved and manipulated later, rather than focusing on the concrete detail of how retrieval would be achieved. The earliest version of the application was structured as shown in Figure 4.1.

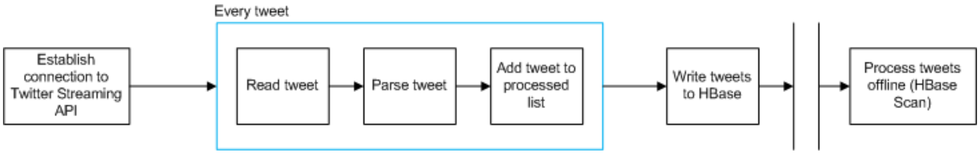

Every tweet
Establish connection to Twitter Streaming API
Read tweet
Parse tweet
Add tweet to processed list
Write tweets to HBase
Process tweets offline (HBase Scan)

Figure 4.1: Initial processing logic of application

The difficulties in using HBase as a data store eventually became too numerous for it to be considered a stable option for the data processing application. After deciding against HBase, Terrier was adopted as both the storage and retrieval mechanism because of its:

1. *Speed*: very large indices can be created within effective time limits
2. *Reliability*: given properly cleansed data Terrier is exceptionally reliable.

With this came large changes to the design of the application and the flow of data through the system. This new logic is shown in Figure 4.2.

The architecture of the application was then defined using this data flow diagram as a foundation.

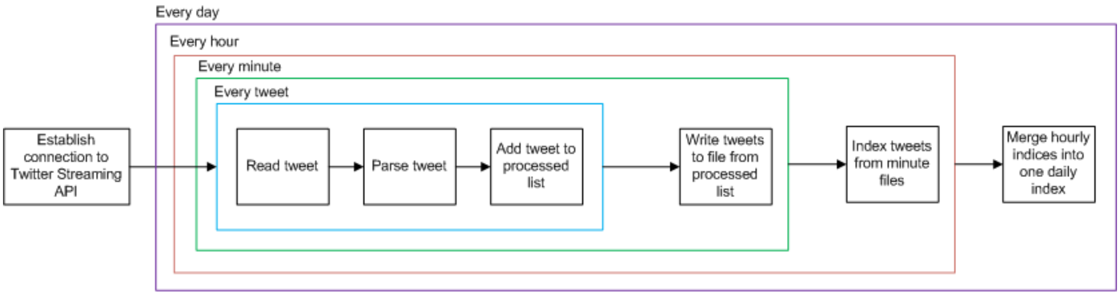

Every day
Every hour
Every minute
Every tweet
Establish connection to Twitter Streaming API
Read tweet
Parse tweet
Add tweet to processed list
Write tweets to file from processed list
Index tweets from minute files
Merge hourly indices into one daily index

Figure 4.2: Core processing logic of application

### 4.2.1 Architecture

The application architecture needed to provide a rigid structure organising each processing task. These tasks including streaming, parsing, writing, indexing and analysing tweets. To appropriately structure the application each task was categorised using the list below based on how resource-intensive it was:

- *Inline*: The task is not resource intensive, and can be performed within the main streaming thread.
- *Concurrent*: The task is resource intensive, but can safely be delegated to another application thread without affecting the main streaming thread.
- *Offline*: The task is very resource intensive and may cause the application to lag. It should be performed in a separate application.

Table 4.1 shows the final categories for the key tasks to be performed by the application.

| Task | Typical Time to Complete | Required Resources | Category |
|---|---|---|---|
| Tweet parsing | <1ms | Parsing library | Inline |
| Directory creation | <1ms | None | Inline |
| File writing | 3̃s per minute | None | Concurrent |
| Data cleansing | 2̃m per hour | None | Concurrent |
| Indexing | 1̃5m per hour | Terrier + dependencies | Concurrent |
| Statistical analysis | 5̃0m per day | None | Offline |

Table 4.1: Task Categorisation

Having categorised each task a high-level architecture diagram was constructed which described in broad terms the general application structure. This is shown in Figure 4.3.

The Streaming thread controls application startup and initialisation, as well as directory creation and tweet streaming itself. When appropriate, new threads are created to manage the period writing, indexing or cleaning of tweet data. This is all stored within a central data repository containing minutely, hourly and daily data files. This datastore can be accessed by the client application for further manipulation and presentation of the results. Finally, the statistic analysis required is

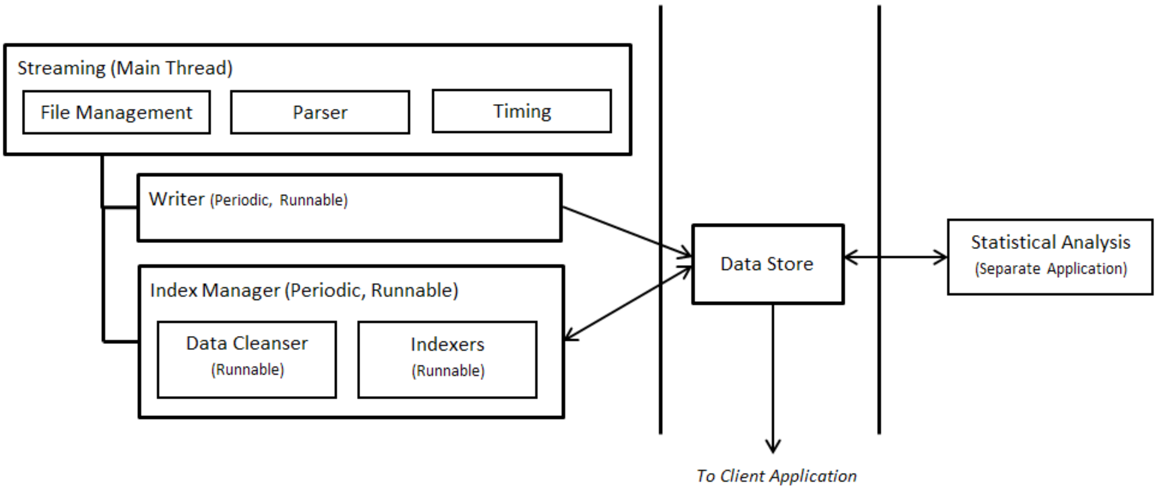


Figure 4.3: Data processing application architecture

conducted by another application entirely, though it relies on information placed in the datastore throughout the streaming process.

Throughout the rest of this chapter the individual architectures of each component are described in greater detail.

### 4.2.2 Storage

Having deciding on using Terrier as the primary retrieval mechanism a folder structure was required in which to organise Pulse data files, indices, logs and the tweets themselves. This data is stored in a hierarchical folder structure, shown in Figure 4.4.

At the top level is the Pulse data directory. This is the central data directory which is used by the user application for retrieving and displaying query results and analytics. Within this directory are the daily data directories (a day being from 00:00:00 - 23:59:59) containing tweets and indices for a given day. These day directories are named using the epoch timestamp at midnight on the day they represent (e.g. data from February 20th 2011 is in folder '1298164800000').

Aside from the core tweet and index directories each daily folder also contains a text file of hyperlinks extracted from tweets on the day, a log file detailing any errors in processing data for that day, and a file of daily aggregated statistics (see Section 4.9).

## 4.3 Data Source

Pulse utilises the Twitter Streaming API to obtain tweets. This API allows connecting applications to utilise an HTTP connection to stream tweets in real time as posted by users (this process is described further in Section 4.5).

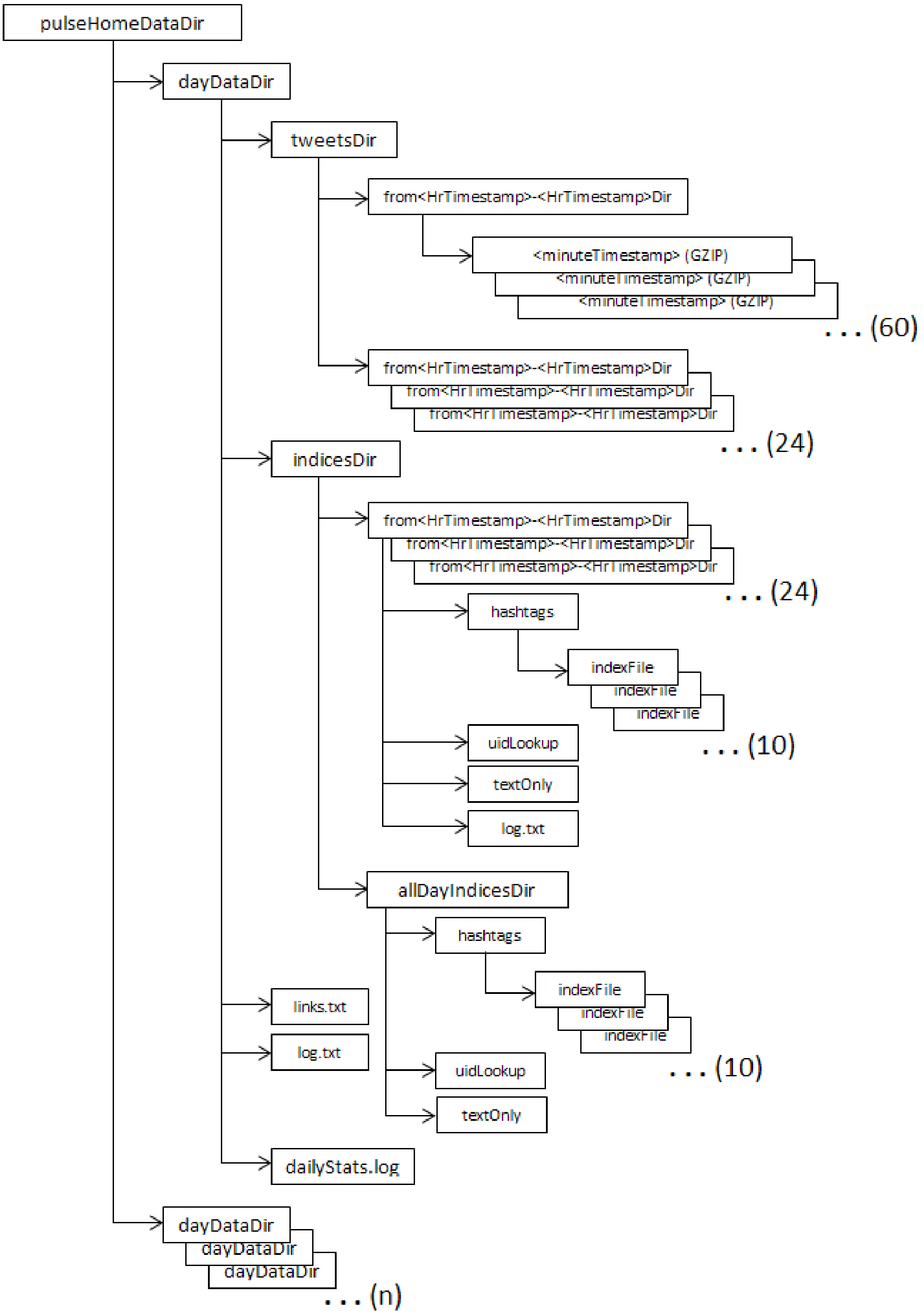


Figure 4.4: Data folder tree structure

Twitter has three access levels designated for the Streaming API: 'Firehose' (all public tweets), 'Gardenhose' (around 10% of all public tweets) and 'Sprinkler' (around 1% of all public tweets). For this project a request for Gardenhose access was granted by Twitter. This is the typical access level required for meaningful data analysis — with an average of 10 million tweets per day, the Gardenhose access level provides a significant data set to study.

Due to storage constraints, Pulse further reduces the volume tweets provided by the Streaming API to a random sample of 50% of the Gardenhose. The final volume stored by Pulse is roughly 5% of all public tweets. Once read from the input stream tweets are parsed and stored as discussed in Section 4.5.

## 4.4 Tweet Entities

### 4.4.1 Data Format

An individual tweet as streamed to Pulse can be requested as either a Javascript Object Notation (JSON) [27] object or as an XML response. JSON is a data interchange format easily parsed by browsers that is considerably more lightweight than alternatives such as XML. Ensuring parsing and processing tweets took as little time as possible was crucial in achieving the processing speeds required (given the number of tweets being dealt with each second), both in the data processing application and in the browser. As Twitter offers clients of the API either response, both data formats were tested with a sample tweet entity to establish which would be faster to stream, faster to parse by browsers and easier to parse cross-browser (see Table 4.2).

| Format | Response Size (Bytes) | Time to Parse (ms) | Cross-Browser Parsing Method |
|---|---|---|---|
| XML | 2534 | 1.38ms | Varies (ActiveX, DOMParser) |
| JSON | 1715 | 0.88ms | JavaScript eval() function |

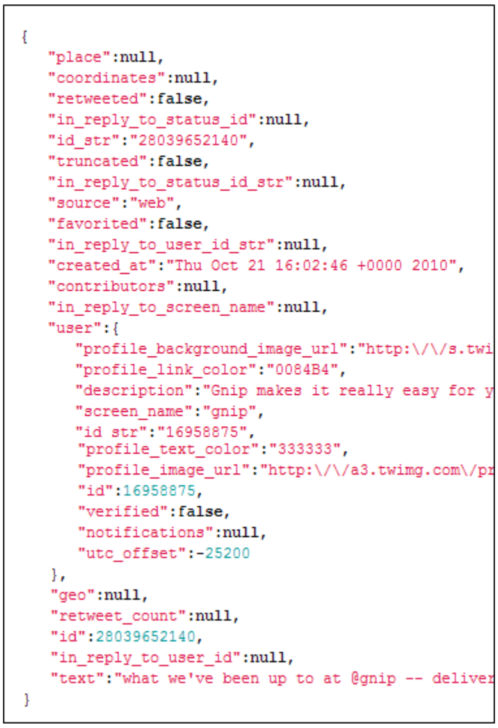
Table 4.2: JSON vs. XML: Attribute Comparison

Based on these results the parse time of each format was quite similar in Google Chrome, the browser used for testing. However, JSON's smaller footprint and the ability of browsers to natively parse the language meant that it was the logical format to use in Pulse.

One interesting caveat of this decision was the resulting differences observed between Google Web Toolkit hosted mode and deployed mode. Hosted mode in GWT does not fully translate client-side code to JavaScript. Instead it is run as standard Java bytecode on in the Java Virtual Machine. Hence it is not until deployment that speed gains from using particular JavaScript features are seen. For example, the true speed of JSON parsing is not revealed until the application is deployed and operating solely from the browser.

### 4.4.2 Tweet Structure

An open connection to the Twitter Streaming API supplies a constant stream of real-time JSON tweets which are parsed and processed by the application. Each contains a complete record of information about a given tweet: the tweet itself, location information, user information, and other relevant meta-data. A typical tweet entity is shown in Figure 4.5.



```
{
   "place":null,
   "coordinates":null,
   "retweeted":false,
   "in_reply_to_status_id":null,
   "id_str":"28039652140",
   "truncated":false,
   "in_reply_to_status_id_str":null,
   "source":"web",
   "favorited":false,
   "in_reply_to_user_id_str":null,
   "created_at":"Thu Oct 21 16:02:46 +0000 2010",
   "contributors":null,
   "in_reply_to_screen_name":null,
   "user":{
      "profile_background_image_url":"http:\/\/s.twi
      "profile_link_color":"0084B4",
      "description":"Gnip makes it really easy for y
      "screen_name":"gnip",
      "id str":"16958875",
      "profile_text_color":"333333",
      "profile_image_url":"http:\/\/a3.twimg.com\/pr
      "id":16958875,
      "verified":false,
      "notifications":null,
      "utc_offset":-25200
   },
   "geo":null,
   "retweet_count":null,
   "id":28039652140,
   "in_reply_to_user_id":null,
   "text":"what we've been up to at @gnip -- deliver
}
```

Figure 4.5: Typical JSON tweet entity

The entity itself is a JSON object, within which are multiple key-value pairs. These pairs can hold string and numerical values, as well as additional nested JSON objects. There are also JSON arrays that provide easy access to important ‘entities’ of the tweet — notably hashtags used, URLs featured and users mentioned. These can then be easily extracted and manipulated as required.

### 4.4.3 Tweet Parsing

Though tweets arrive as JSON entities from the Twitter stream they are immediately parsed using a JSON parser into objects of type *Tweet*. The *Tweet* class gives convenient access to the oft-used key-value pairs stored within each tweet entity, as well as convenience methods used by other Pulse components for processing, manipulation and presentation of Twitter data.

The JSON parser used by Pulse is built upon a Java implementation of the reference parser available from json.org [27]. This was chosen early in the project based on previous experience parsing JSON. Early benchmarks of parsing performance on incoming tweets (at double the current rate) showed no lag or backlog being caused by the parser. Based on this it was selected for use in Pulse. However, were the application to process considerably more tweets per second this may have to be reviewed.

The JSON tweet entity itself is also stored as part of the *Tweet* object. This has advantages in allowing future components to access currently unused values within the tweet (such as user data) — however, it also increases the amount of storage required for every *Tweet* object. Though this is certainly a non-negligible storage increase (a single tweet entity is often 5000 bytes or more) it provides future component developers with a full and complete record of each tweet. Given that Twitter only offers access to a very recent history of tweets on any of its non-commercial streams this was decided as vital, both for future development and also before the full suite of components to be provided in this release of Pulse was decided upon.

## 4.5 Streaming

One of the main requirements of the data processing application was to stream tweets from Twitter into manageable, logical data structures for later manipulation. The streaming process implemented was integral to completing this requirement, and encapsulates the end-to-end process of connection to storage on disk.

### 4.5.1 API Connection

**Connection Considerations**

Pulse connects to the Twitter Streaming API using an authenticated long-lived HTTP connection. Provided a valid username and password is supplied (in the case of Pulse, my own Twitter username and password with Gardenhose access granted) tweets will continue to stream until:

- Twitter performs network maintenance: in this case, the application will attempt to reconnect after a *linearly* increasing sleep time.
- The connection is closed application-side: in this case, the client will attempt reconnect after an *exponentially* increasing sleep time, capped at 360 seconds.
- The connection is closed server-side: in this case there is likely an error occurring within the application (such as duplicate connections, laggy connection, or invalid login details). These events almost never occur.

**Quality of Service**

Twitter makes the following three statements [38] regarding the quality of service to be expected of the Streaming API:

- *Best Effort*: On occasion statuses may be missing from the stream
- *Unordered*: Statuses are streamed in no particular order, “but will tend to be k-sorted, where k is the number of tweets received in about 3 seconds” [38].
- *Generally at-least-once*: Duplicates that do occur are likely due to “operational events” caused by Twitter.

Management of each of these constraints is discussed in Section 4.8.

**Sample Specification**

As previously mentioned, Pulse downloads around 5% of all public tweets at Gardenhose access level. However before arriving at the application this stream is pre-filtered by Twitter.

Firstly, all non-public statuses are excluded, such as statuses from protected accounts. Secondly, an unspecified ‘quality filter’ is applied to certain accounts whose tweets are removed from the public stream. Twitter does not specify what quality filters are, aside from that they catch accounts posting repetitious or ‘spam’ tweets — stating “Our search results will not include suspended accounts, or accounts that may jeopardize search quality. Material that degrades search relevancy or creates a bad search experience for people using Twitter may be permanently removed.” [38].

Whilst this has the potential to show bias towards results in particular application scenarios (such as the ratio of spam to non-spam tweets) I believe for Pulse’s purposes it is a negligible issue. Users of Pulse will want to understand what real users were discussing and in what way, rather than filtering through unimportant or spam results.

Once these are filtered Twitter performs a modulo 100 operation on each public tweet’s UID. Those which returns results in the 0-10 range are streamed to Gardenhose users. Though this itself is not random, Twitter states that this in combination with its status ID assignment algorithm (the details of which are undisclosed) ultimately tends towards a random sample in non-trivial streaming scenarios.

These conditions may lead to doubt surrounding the true randomness of the sample provided, and whether bias (particularly towards quality) is being introduced. However, for Pulse’s purpose this data set is still incredibly useful for two reasons:

1. Twitter offers a vast result set of millions of real-time tweets every day, along with large amounts of supplementary data. Few other social media data sources are as comprehensive, concise, popular and easily available.
2. Pulse end users will primarily focus on quality tweets which give insight into events in the Twittersphere on a given day or topic. Spam or accounts which abuse the platform are likely to bias Pulse results as much as quality filtering on the part of Twitter.

### 4.5.2 Connection Implementation

Pulse utilises the Apache HTTPClient library [13] to create and configure HTTP connections to the Streaming API. HTTPClient was selected primarily on Twitter recommendation: it provides the full suite of functionality required for Twitter connections (keep-alive headers, authentication, status code information), as well as allowing connections by proxy (required by the university network). It is also a lightweight, mature and well-documented library which helped quickly develop a working prototype of the application.

The Streaming API connection is formed, initiated and closed in four stages:

1. *Configuration*: Appropriate parameters are set for the HTTP request, and authentication details are added. If the application is behind a proxy, proxy settings are also configured.

2. *Request / Response*: The HTTP request is executed and a response is returned. If the response is OK (200) then streaming begins. If not, the application waits for an appropriate length of time.

3. *Stream*: Once a valid connection is established an *InputStream* and *BufferedReader* are created. These are used to read the response from Twitter and obtain each JSON entity.

4. *Closure / Reconnect*: Once the reader returns 'null' the connection has likely been closed. Assuming this is a temporary error the application will attempt to reconnect after an appropriate wait time.

Should the connection end for any reason, an appropriate time is waited (explained in Section 4.5.1) before reconnection is attempted, dependent on the disconnection reason.

### 4.5.3 Writing Streamed Tweets

The streaming process will continue as long as a valid connection remains open to Twitter. However, at certain 'trigger points' the application will send all tweets in memory to file.

The streaming trigger is initially configured for one minute after application startup. Once the current tweet publication date is equal to or greater than the trigger three actions are performed; a new *TweetWriter* thread is created, the current *TweetWriter* thread is passed the tweets it needs to write, and the streaming trigger is pushed forward one minute. At this point all tweets in memory will be written to a 'minute file' within the appropriate hourly and daily directories.

### 4.5.4 Streaming Issues

It was observed that almost exactly once per hour, every hour, the connection to Twitter was dropped (signified by a reconnect message within the application log files). Though this only resulted in a near negligible number of dropped tweets (roughly 1 or 2 per reconnect) it was still occurring even though the implementation meets the guidelines laid out by Twitter. As the effect on the application was marginal little time was spent investigating the root cause. However, the testing conducted revealed that the issue was impossible to recreate on a non-university machine.

Based on this, and considering the regularity and predictability of the dropped connections, I believe it was caused by the university proxy server resetting the connection after an hour. Whilst not ideal the limited effect on data quality meant this was an acceptable cost given the benefits derived from being able to stream tweets for almost 24 hours a day.

### 4.5.5 File Management

The organisation and structure of files is defined by key time periods; on startup, every day, every hour and every minute. When the specific trigger points for each of these periods is reached one or more actions are taken by the application. Figure 4.6 gives an overview of these events.

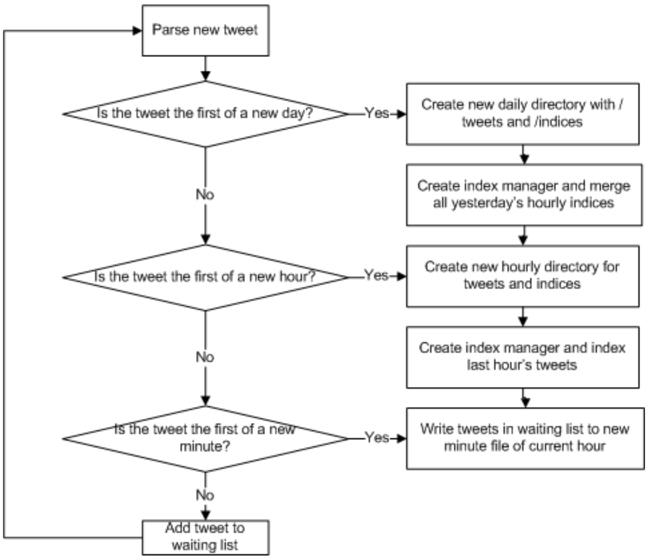

Parse new tweet
Is the tweet the first of a new day?
Yes
Create new daily directory with /tweets and /indices
Create index manager and merge all yesterday's hourly indices
No
Is the tweet the first of a new hour?
Yes
Create new hourly directory for tweets and indices
Create index manager and index last hour's tweets
No
Is the tweet the first of a new minute?
Yes
Write tweets in waiting list to new minute file of current hour
No
Add tweet to waiting list

Figure 4.6: File management logic

On startup the application performs the failure recovery checks detailed in Section 4.7. However, should the current instance be a fresh run, Pulse will create a series of directories in which to store tweet and index data. It will also configure all of the trigger points of the application (minutely, hourly, and at midnight). Once streaming begins directories for the current hour will also be created if they do not already exist.

Every minute, a new file is created within the current hour directory of the current day. This file contains the combined tweets for the previous minute, and is named using that minute's Unix timestamp (e.g *gzipTweets-<UnixTimestamp>*). Tweets are stored in compressed GZIP files as binary objects. Each hourly folder usually contains 60 of these files, corresponding to one per minute of streaming. However, this can vary if the connection is slowed or dropped for any significant period of time (> 1 minute).

Tweets were originally stored in hourly files, however the volume of data kept in memory meant that this option quickly became unfeasible. Electing to divide tweets into files by minute eradicates this issue. It also adds robustness to the system, in that if for any reason a writer thread fails to complete writing, only one minute's worth of tweets is lost.

Every hour new directories are created for that hour's tweets and indices. Like the daily directories and the minute files, these are named using the Unix timestamp of the hour the represent (e.g. *'from<UnixTimestamp>-60mins'*)

Every day a new daily data directory is created, along with corresponding 'tweets' and 'indices' folders.

## 4.6 Indexing

### 4.6.1 Indexing Triggers

Tweets are indexed using Terrier on a hourly and daily basis within the data processing application. Indexing within the data processing application is triggered on reaching specific timing values. Each time a tweet is parsed the publication date is inspected and compared to a 'hourly flag'. The hourly flag is initially set on application startup to one hour after tweet streaming has begun. When the publication date of a tweet is equal to or after the hourly flag, the hourly indexing process starts and the hourly flag is pushed forward one hour.

Should the application have started at some arbitrary point during the day (rather than having been running for multiple days) the hourly flag will automatically reset at midnight.

Before full indexing begins the appropriate folder structure for indices to be placed in is created. This structure is described in detail in Section 4.2.2. Additionally the individual index pipeline components are created and added to the index manager. It is at this point that each indexing pipeline component has its tweet source, index destination, and desired index type configured.

### 4.6.2 Indexing Manager

The indexing manager is responsible for maintaining order in the pipeline through which tweets pass before, during and after indexing. Every component in the pipeline runs within its own thread separate from the main streaming thread, including the indexing manager itself. Running the three application threads concurrently was necessary to ensure streaming ran as fast as possible.

The indexing manager is initialised containing each pipeline component it must execute. Though all of the components know the source and destination of the data they require, the indexing manager ensures that these are executed in the right order within their own threads.

Each pipeline component is extended from the abstract class PipelineComponent. This itself is subclassed by two further classes: IndexPipe and CleanerPipe. The index manager runs all pipeline components subclassing CleanerPipe first, then pipeline components subclassing IndexPipe, each within its own thread. The index manager waits for these processes to die before continuing with the next in the queue. This prevents concurrent modification errors whilst reading tweet files, though still allows the main tweet streaming thread to run independent of the indexing process. This sequence is shown in Figure 4.7.

The indexing manager also keeps track of successful and failed index processes, enabling overall success of cleaning and indexing to be monitored and, if necessary, debugged. This proved useful during the development stage of the application for establishing the root cause of problematic bugs (e.g. the aforementioned concurrent modification file I/O issue avoided by using thread joins).

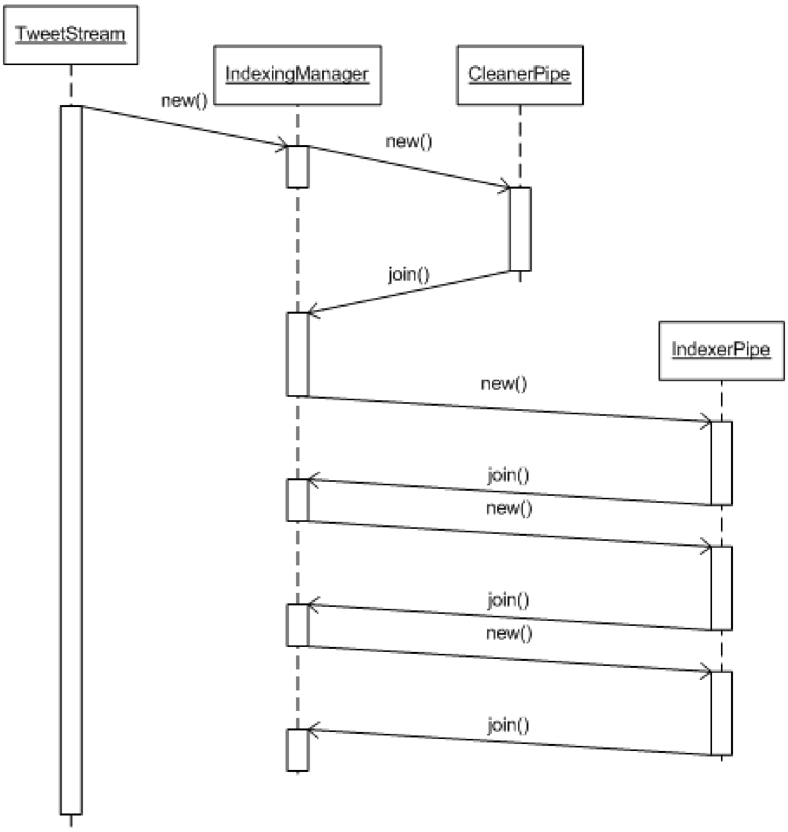


Figure 4.7: Index Manager Thread Management

### 4.6.3 Data Cleansing

Before indexing can begin each minute file must be cleansed to ensure the data it contains is valid. In addition the 60 minute files are then checked for duplicates. Any duplicate tweets across the files have the potential to cause indexing to fail (by adding duplicate meta-index keys for a given UID).

This risk is mitigated by adding a duplicate remover class to the indexing pipeline. Run before indexing as a separate thread, all tweets from each minute file are read from file, and the UID of each tweet is then stored along with the file read from as a hashset. Should a tweet UID already exist in the set then it is added to a duplicate list, along with the file in which it appears.

Once this is complete the tweets are read again. If the tweet UID and file name appear together in the duplicate list the tweet is discarded. Otherwise, the tweet object is written to a new file with an amended filename. If no duplicate tweets were found in a given file then only the filename of the file is changed.

Using the filename / tweet UID hashset means that there is no need to read in every tweet for an hour into memory. Avoiding this pitfall was essential to ensure future scalability. Holding even one hour's tweets in memory was unfeasible and during initial testing often resulted in the JVM running out of heap space. Though more resources could have been provided, this vertical approach to scalability would not be satisfactory should the application be deployed streaming a larger percentage of the Twitter public stream.

### 4.6.4 Indexers

Once the data files have been cleaned indexing can be begin. Each index pipeline component is responsible for indexing one particular type of index, as well as configuring relevant Terrier parameters. For Pulse's purposes, this means setting the name and length of the meta-index key-value pairs, and disabling Terrier term pipeline options. For example, given the scope for unusual tokens appearing in tweets, stemming and stopword removal are not utilised by the Pulse indexers.

#### Documents & Collections

Terrier represents document corpora using a Collection interface [34] — essentially an iterator over a series of individual documents (themselves represented by classes implementing the Document interface [35]). Whilst it provides concrete implementations of these for popular document types, Pulse required custom implementations in order to create its indices. Each custom Document implementation is responsible for parsing and term extraction of a particular input type, with the custom Collection class managing creation of and access to these documents.

The Collection implemented by Pulse, *TweetCollectionFromGZIP*, provides Terrier with documents created from the compressed tweet files produced during streaming. The tweet source for each index type during hourly indexing is always the same, however the documents created for indexing vary. Table 4.3 describes each of the three types of index created every hour.

The *HashtagDocument* class tweaks the standard FileDocument class to be more lenient in defining what a term is — necessary given the form hashtags take (e.g. they cannot be space separated, may

| Index Name | Document Class | Description | Meta Data |
|---|---|---|---|
| Hashtag | *HashtagDocument* | each document within the collection indexed is a comma-separated string of hashtags for a given tweet | Tweet UID |
| Unique Identifier Lookup Index | *DBRecordDocument* | each document within the collection indexed is a unique tweet ID | Tweet UID, JSON entity |
| Text Only Index | *TweetTextDocument* | each document within the collection indexed is the full text of a tweet. This index is not currently used by Pulse | Tweet UID |

Table 4.3: Index Types

not be 'traditional' words). For each string of tags indexed the corresponding tweet UID is added as document meta-data to the Terrier meta-index.

The UID lookup index acts as a pseudo-database for storage and retrieval of JSON tweets. Each *DBRecordDocument* added to the index is a UID string for a given tweet. The Terrier meta-index then stores the full JSON entity corresponding to that UID. This means the UID index can be queried for a particular UID, after which the meta-index can be searched to get the full JSON tweet for further processing.

There is a known issue at this stage involving the length of tweet entities. Cursory tests of 1000 sample tweets showed that the JSON entities were always under 5000 characters in length. Terrier must be informed of the length of each meta-index value through its `indexer.meta.forward.keylens` property, which in the case of JSON entities was set to 5000. However, later request handling occasionally failed to parse certain tweet entities retrieved. Investigation revealed that JSON entities larger than 5000 characters were simply truncated when stored in the meta-index. This has since been fixed, though older indices created before the bug was identified still contain malformed tweets. A workaround for these indices is implemented by the core retrieval service (described in Section 6.4.3).

The *TweetCollectionFromGZIP* class ensures the correct documents are created for each indexer. The class first checks the indexer mode to create the required document from the current tweet object. This is then returned to Terrier for indexing.

Should a file have no further tweets the *TweetCollectionFromGZIP* class attempts to load the next tweet file (if it exists). If it does, the next file of tweets is processed and indexed. If not, then all tweets have been indexed and the indexer joins the indexing manager. This process is shown in Figure 4.8.

### 4.6.5 Daily Indexing

At the end of each day all of the hourly indices created need to be merged into a single daily index so as to hold an entire day's tweets. It is these daily indices that are then later queried by the client application for tweet data. The following section details the approach taken in completing these mergers.

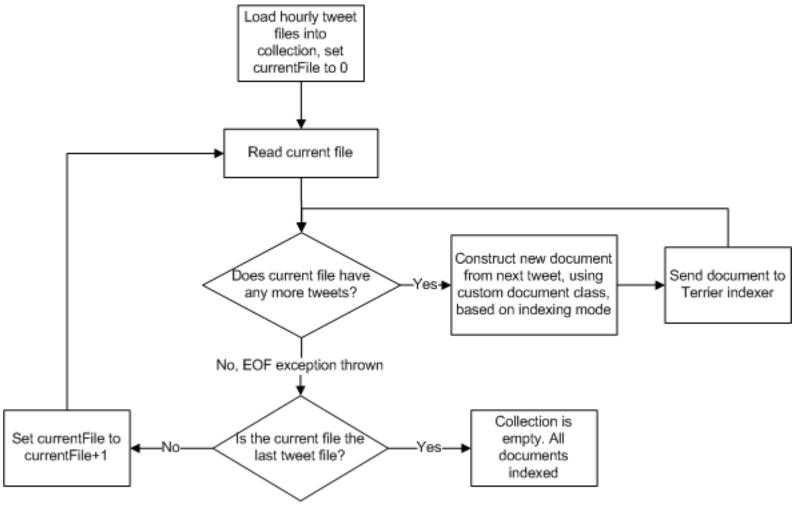

Load hourly tweet files into collection, set currentFile to 0
Read current file
Does current file have any more tweets?
Yes
Construct new document from next tweet, using custom document class, based on indexing mode
Send document to Terrier indexer
No, EOF exception thrown
Is the current file the last tweet file?
Yes
Collection is empty. All documents indexed
No
Set currentFile to currentFile+1

Figure 4.8: TweetCollection Logic

## Merging Approach

Pulse uses the mergeTwoIndices method provided by Terrier's *Indexer* class to merge indices in a pairwise fashion. During the first round of mergers the indices to be merged are loaded from the hourly indices folder. These are merged and output as 12 2-hourly indices within a new folder, /dailyIndices.

From this point onwards all mergers take place within the /dailyIndices folder. As the pairing off of indices progresses eventually only three indices are left. Pulse merges two of these, then merges the resultant index with the leftover index. This merger tree structure is shown in Figure 4.9. Interim indices are deleted at the end of each complete daily merge. The merging process is conducted three times for each day folder; once for each of the three index types.

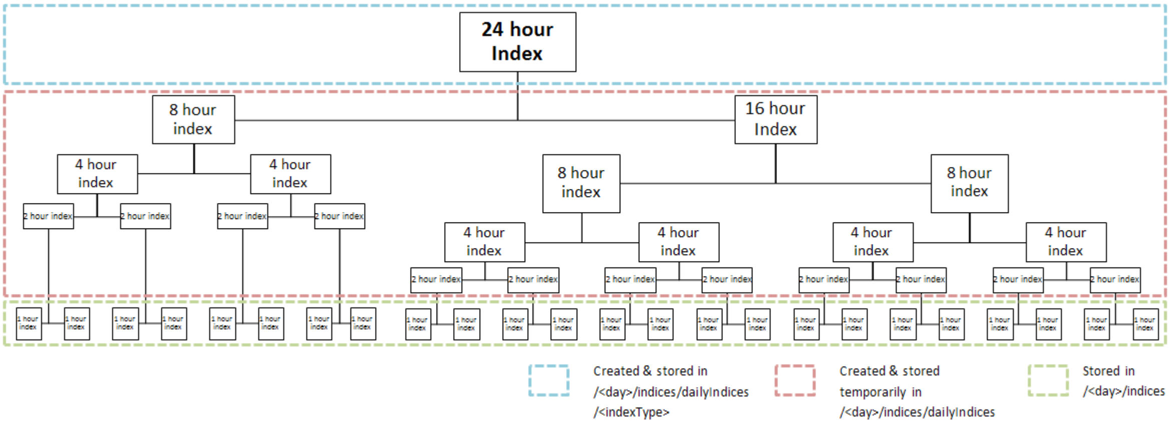


Figure 4.9: Pairwise Merge Tree

### Merging Implementation

The hourly indices folder is read to check how many indices of the desired type are available for merging, which are then stored as an array of directories. Each index in this array of 'clean' directories is then paired off with another into *TweetMergeWorker* threads. *TweetMergeWorker* is a class implementing Runnable that performs each individual two-index merger using the aforementioned Terrier method.

Once all indices inside the current clean directories array have been merged, $\frac{n}{2}$ new index directories will have been created (where *n* is the number of original directories). Each of these directories is named corresponding to the time range its index covers (e.g. *from<unixTimestamp>-<unixTimestamp>*) These new directories are considered 'clean' and added to the clean directory array, whilst those just merged are considered 'dirty' and ignored. This process is repeated until the number of clean directories is 1 (the final 24 hour index).

Once only 1 index directory remains all prior directories are deleted and the final index is renamed to its index type, rather than time range.

Table 4.4 details the time taken to fully merge 24 hours of tweets (the final index had 5,011,326 documents). These benchmarks were obtained using a Java heap size of 1500Mb on a dual-core, 1.8Ghz processor CentOS 5 machine.

| Index Type | Total Time (hh:mm:ss) | Total rIndex Size (Mb) |
|---|---|---|
| Hashtag | 00:35:23 | 271 |
| UID Lookup | 05:14:04 | 5289 |
| Text Only Index | 00:48:09 | 782 |

Table 4.4: Daily Merging Benchmarks

*N.B.* The vast majority of the UID lookup index size is the meta-index.

### Merging Challenges

The main difficulty in merging indices was trying to integrate the process into the existing indexing pipeline. Merging multiple indices, particularly of the size used by Pulse, takes a large amount time and resources. Running the merging process alone with anything less than 1500Mb of memory often led to heap space exhaustion, and during preliminary testing of a unified indexing pipeline similar failures occurred at midnight with even higher heap sizes due to the fact that both hourly and daily indices were being created. Ultimately it was decided to run the daily merging process as a seperate task, for both merging stability and robustness of the data processing application.

This process has been scripted to be as smooth as possible, but further work is required to fully integrate it into the indexing pipeline. Were it to be fully integrated a distributed structure would almost certainly be required to handle the work, more so were the application to be scaled further in terms of tweets processed.

Ensuring merging time was kept to a sane limit was also challenging. In order to merge the 24 hourly indices into a daily index in a reasonable time Terrier was configured not to use reverse key lookup for the meta-index. Reverse keys (as opposed to forward keys) are meta-index keys whose values can uniquely identify a document within a collection. As tweet UIDs are stored as meta-data within every index it would be useful to be able to lookup indices using this parameter, however full 24 hour merging was never successfully completed whilst this property was set.

Whilst omitting reverse keys considerably improved the speed of daily indexing (from an indeterminate amount of time to around 6:30 hours for all 3 mergers), it meant that reverse meta-data key lookups could not be performed. This had an impact on performance during retrieval from the server (described in Section 6.4.4).

## 4.7 Failure Recovery

Should the data processing application fail for any reason it was imperative that when restarted it could resume work starting from its failure point. To this end the application performs checks on start up to establish whether it was previously streaming into the data directory, or it has been tasked with a new streaming session.

At startup the application requires a directory path argument informing it where to create the Pulse data folder structure (see Figure 4.4). Before streaming begins it checks within this folder to ascertain if data from previous runs exists. If not, it will create new structures and begin streaming anew. However, if so the application will attempt to restart from where it failed.

Firstly, the folder is checked to ensure it is valid. A valid Pulse data directory will contain only directories with Unix timestamps as names - midnight on the day the data applies to. Finding a file or any other type of folder will result in the application exiting. Once validity is confirmed the latest 'day' folder will be checked.

If the day matches the current day the application will resume from the next hour of that day. So, if the application fails at 9:05pm on 24th February and is restarted immediately, it will resume streaming from 10:00pm. Similarly, if the application is restarted at any point during the original day it will start from the top of the next hour, up until midnight. If the day does not match the current day, the application will create a new day folder with the current day's Unix timestamp. It will then start streaming immediately into the new day directory.

Whilst these measures provide broad recovery solutions (streaming usually begins again within an hour of failure), they could be improved. Waiting until the hour means that potentially 59 minutes worth of data is ignored, and during testing 'holes' within the aggregate data could be seen from where the system was down. However, having now reached stability the data application downtime is low, and the additional effort of resuming streaming by the minute or even second was exponentially greater than the potential benefit given the time available for development. This also provided motivation to ensure the application was actually robust and correct, rather than just being very good at recovering from failure.

## 4.8 Adherence to API Usage Conditions

Ensuring Pulse used the Twitter streaming API correctly was a requirement of Gardenhose access, and as Pulse's core data provider adhering to these mandates was a critical requirement. The requirements laid out by Twitter can be divided into two types:

- Technical Requirements: mandates that specify how the application should interact with the API. These include use of appropriate connection types, handling of connection failures and dealing with non-200 HTTP codes. Inadherence to these requirements would result in temporary or permanent suspension of access to the API.
- Data Handling Requirements: implementation details that must be considered when using the API. These include tolerance of duplicate or out of order tweets, data integrity checking and graceful parse failure handling. Lack of consideration around these requirements would result in missing data, corrupt data and unexpected application failures.

The technical requirements were dealt with by considering points at which the processing application could stall, and if they occurred establishing which party was responsible for that failure. From there appropriate actions were defined that either rectified the failure, or waited for an appropriate event or amount of time before normal service resumed. These stallpoints are listed in Table 4.5.

| Stallpoint | Description | Effect | Resolution Strategy |
|---|---|---|---|
| HTTP Status Code != 200 | On making an HTTP connection to Twitter the server has responded with a non-OK HTTP code | Unable to connect to Twitter | Sleep for 20s, increase time exponentially for each consecutive failure |
| Object read from stream is null | Null read suggests the connection has been closed | Unable to stream tweets | Close existing connection and reconnect |

Table 4.5: Application Stallpoints & Resolution Strategies

Where the technical requirements above focused on correct use of the API, the data handling requirements were important for the functioning of Pulse as a client application. Twitter makes few guarantees about the order and cleanliness of data it provides, requiring clients to ensure they perform data cleansing on anything streamed. Pulse mitigates these risks as described in Section 4.6.

| Issue | Description | Effect | Resolution Strategy |
|---|---|---|---|
| Malformed or non-tweet entity | JSON object streamed is malformed or unparseable | Tweet unusable as data | Catch parsing exception, discard tweet, continue streaming |
| Duplicate tweet | Tweet is a duplicate of another tweet | UID lookup index for hour will fail | Eradicate tweets from minute files before hourly indexing using tweet cleaner (described in detail in Section 4.6) |
| Tweet out of sync | Tweet is not in order with other tweets received | Inaccurate data in Pulse application | Check divergence from current time. If over 60 seconds, discard tweet |

Table 4.6: Data Integrity Issues

## 4.9 Additional Batch Processing

A separate data processing utility was developed to aggregate Twitter data on a daily basis. The output is used to power temporal range queries within Pulse without needing to search tweet indices. 2 files are generated by this application, *dailyStats.log* and *dailyStatsTop10.log*. Each file contains a binary object of class *DailyStats*, containing:

- *Daily Hashtags*: each hashtag used throughout the day, how many unique countries tag was used in, number of unique users using tag and number of retweets tag featured in
- *Hourly Tweet Volume*: hourly volume of tweets across the day
- *Daily Sentiment*: summed daily Twitter sentiment. See Section 5.7.8 for a detailed description of the sentiment analysis process
- *Retweets*: each retweet posted during the day, the retweet count, the publication time, the location of the user posting the retweet and the usernames of each retweeter
- *Geographical Tweet Count*: list of countries tweeting, plus volume for each country
- *Source Count*: list of unique devices used to tweet from, plus volumes for each device
- *URLs Tweeted*: list of URLs posted on Twitter, plus occurence count of each
- *User Mentions*: list of all Twitter users mentioned, plus number of mentions.
- *User Follower / Following Counts*: list of all users, plus number of followers and users followed

The *dailyStats.log* output contains a comprehensive overview of Twitter activity for each day. However, its file size is prohibitively large for real time use (30Mb per day, and, as a complete record of a day on Twitter, contains data most users will find irrelevant (e.g. tags which appear only once, URLs which are posted rarely). Figure 4.10 shows the data aggregation process.

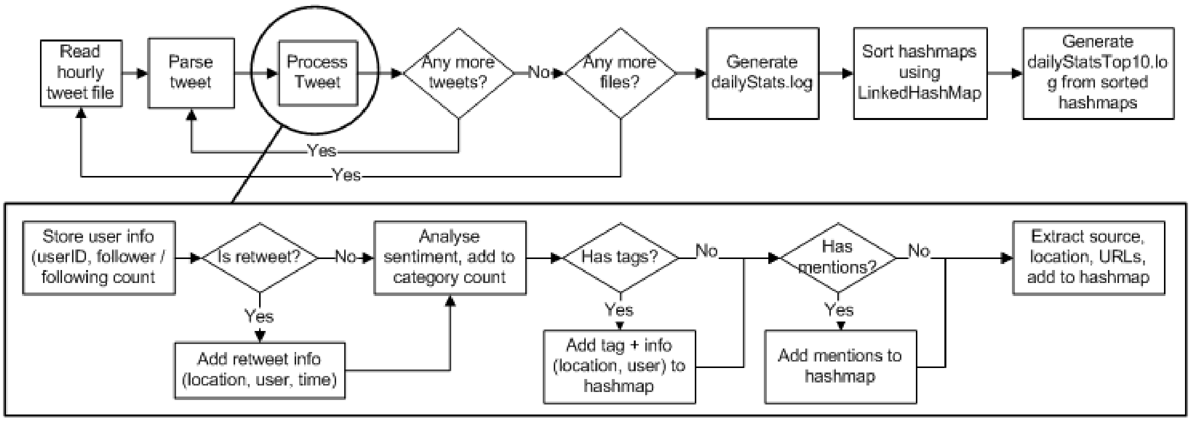

Read hourly tweet file
Parse tweet
Process Tweet
Any more tweets?
No
Any more files?
Generate dailyStats.log
Sort hashmaps using LinkedHashMap
Generate dailyStatsTop10.log from sorted hashmaps
Yes
Yes
Store user info (userID, follower / following count
Is retweet?
No
Analyse sentiment, add to category count
Has tags?
No
Has mentions?
No
Extract source, location, URLs, add to hashmap
Yes
Add retweet info (location, user, time)
Yes
Add tag + info (location, user) to hashmap
Yes
Add mentions to hashmap

Figure 4.10: Twitter Data Aggregation

| File | Tags | Tweet Vols. | Senti. Cat. | RTs | Countries | Source | URLs | Mentions |
|---|---|---|---|---|---|---|---|---|
| dailyStats | All | 24 hours | 6 | All | All | All | All | All |
| dailyStatsTop10 | 0.25% | 24 hours | 6 | 20 | All | 35 | 0.1% | None |

Table 4.7: Aggregate Twitter Data Comparison

As Pulse temporal range queries aim to present an overview of the most popular Twitter topics of the day a subset of the above is stored within a different file: *dailyStatsTop10.log*. A comparison of data in each file is presented in Table 4.7.

Both files store the volume of tweets per hour, per sentiment category and per country — these are simple arrays with negligible storage cost. The largest omission from *dailyStatsTop10* is user mention data. Pulse does not provide analysis of user activity, so mention data only slowed down file loading at query time. However, future components could use this information to rank users and map social connections.

The smaller file also excludes 99.25% of all hashtags posted, storing only the top 0.25% (roughly 700 per day). Usage of hashtags on Twitter follows a power-law distribution, whereby a very small percentage of hashtags are used very frequently each day, with a long tail of infrequently-used hashtags. The ability to freely create hashtags by preprending '#' to any word means there are limitless potential hashtags, the vast majority of which never build momentum. Those that do are stored for use in Pulse.

Retweets and sources are limited as well. The number of devices used to tweet also exhibits a power-law distribution, from the vastly popular official apps down to the most esoteric handbuilt programs. Of these only the top 35 are stored each day (chosen as this number were found to be used 10,000 times or more per day). Retweets are also limited to the top 20 (by number of occurences).

The number of URLs stored is also vastly reduced by number of occurences. 68% of all links posted occur only once, and a further 21% occur less than five times. For a link to be classified as popular Pulse requires at least 100 occurences in a single day — this equates to roughly 0.1% of all links posted.

With the amount of data reduced each *dailyStatsTop10.log* file is around 30Kb — much more manageable for real-time use. When a request is received the server-side application loads the appropriate *dailyStatsTop10.log* file and parses the *DailyStats* object into a serializable format. Usage of these files is discussed in Chapter 6.

## 4.10 Testing

Table 4.8 describes the test cases relevant to the streaming component of the application.

Test cases 1 – 9 focus on the robustness of the application, whilst cases 10 – 12 primarily concern synchronisation and propagation of triggers to other components within the application.

Table 4.9 shows the actual tests performed upon the system, based on the test cases, with the test results.

| Test Case ID | Description |
|---|---|
| 1 | Application is run with valid arguments (data folder path / Terrier home path) |
| 2 | Application is run with one or both arguments missing |
| 3 | Previous run of application ended unexpectedly, same data folder is provided |
| 4 | HTTP response code from Twitter API connection = 200 |
| 5 | HTTP response code from Twitter API connection $>$ 200 (error code) |
| 6 | Response entity read from live connection is non-null |
| 7 | Response entity read from live connection is null |
| 8 | Tweet parse is successful |
| 9 | Tweet parse is unsuccessful |
| 10 | Current midnight trigger is reached |
| 11 | Current hour trigger is reached |
| 12 | Current minute trigger is reached |

Table 4.8: Test Cases: Streaming

These tests were carried out numerous times over the course of development, with baseline results stored for comparison against results taken after new functionality had been added (for example, the failure recovery scope changing from daily to hourly-level recovery).

Table 4.10 describes the test cases relevant to the indexing component of the application.

Terrier was treated as a ‘black box’ component for purposes of testing, on the assumption that provided with valid data and appropriately configured parameters it would perform the function expected of it.

Table 4.11 shows the actual tests performed upon the system, based on the test cases, with the test results.

The failure of test 3 resulted in a major design change. Having all 3 indexers run simultaneously would have considerably increased the indexing time, however each requires access to the same tweet files — throwing an exception. Given that the hourly indexing process still completed in under 40 minutes when run sequentially it was deemed unnecessary to further investigate a possible solution, opting to wait for each indexing thread to die before continuing.

Table 4.12 describes the test cases relevant to the daily merging component of the application.

As discussed in Section 4.6.5 merging proved too troublesome to include as a component within the indexing pipeline. The results of testing index merging within this pipeline are shown in Table 4.13.

Some peculiar bugs relating to implementation details of the daily index process were discovered during testing. In test case 2, an odd number of initial candidate indices would result in an error. This is because the first round of mergers (from 24 to 12 usually) places the interim indices in a new directory, /dailyIndices. During the second round of merging all candidate indices are found in the /dailyIndices folder. Any original hourly indices not merged in the first round are then ignored.

Though now the application always generates 24 hourly indices, previous development versions occasionally failed to create an index. This failure was compounded through another index being

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Application run with valid args. | Streaming begins | Success: application runs as expected |
| 2 | Application run missing args. | Application usage error displayed | Success: message displayed |
| 3 | App. startup with previously used data folder | Resume streaming through app. recovery | Success: streaming resumes from correct hour / day |
| 4 | 200 OK connection response | Begin reading tweets from stream | Success: tweets streamed |
| 5 | Non-200 connection response | Exponential wait for every further non-200 response | Success: wait applied, error logged |
| 6 | Non-null response read from stream | Parse tweet | Success: tweet parsed |
| 7 | Null response read from stream | Linear wait applied until entities non-null | Success: wait applied until non-null response received |
| 8 | Valid tweet, parsed successfully | Date checked, relevant triggers fired | Success: triggers fired based on date |
| 9 | Invalid tweet, parsed unsuccessfully | Discarded, add to fail count | Success: tweet discarded |
| 10 | Tweet date equal to or after midnight trigger | All triggers reset, new dir. struct. created | Success: dir. struct. created, new triggers created |
| 12 | Tweet date equal to or after hour trigger | Indexing pipeline activated | Success: indexers added and manager by indexing manager |
| 13 | Tweet date equal to or after minute trigger | Writer created, tweets written to file | Success: last minute's tweets written to file |

Table 4.9: Tests: Streaming

missed at the merging stage.

Test case 2 describes the 'file in use' error received when trying to delete the interim indices from the /dailyIndices directory. Interestingly only 1 file from each index was in use, a Terrier index file that was somehow being held open by the application. After checking to ensure all files were being closed after indexing the error still appeared. Only by running the directory cleanup as a separate process did the error not occur.

For these reasons, and others discussed previously, daily merging was extracted and performed through a separate application.

*N.B.*: As a primarily autonomous application little user input is required except on start up. As such all filepaths, if initially configured correctly, will be valid through the various application stages (streaming, writing and indexing).

The nature of the application, designed to be autonomously run for extended periods of time, meant that testing was particularly challenging. New features and were regularly added as the design evolved, and ensuring that this had no impact on existing functionality meant not only feature

| Test Case ID | Description |
|---|---|
| 1 | Tweet source folder files <> 60 |
| 2 | EOF reached in minute file |
| 3 | Multiple indexers running concurrently |
| 4 | Terrier indexing fails (for any reason) |
| 5 | Tweet file is corrupt |

Table 4.10: Test Cases: Indexing

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Collection class supplied <> 60 files | Change in value logged and used as new total files | Success: file total adjusted |
| 2 | EOF reached in current tweet file | Load new tweet file, if any, or exit | Success: new file loaded if it exists |
| 3 | Multiple indexers initiated concurrently | Indices of types specified | Failure: concurrent modification error (of tweet files) |
| 4 | Terrier fails to complete indexing | Retry, attempt next index process, log | Partial: attempts next index process, logs error |
| 5 | Corrupted tweet file | Ignore file and continue indexing | Success: file ignored and error logged |

Table 4.11: Tests: Indexing

testing but also longer-term integration testing.

This testing sometimes occurred over a number of days, and occasionally errors in the code would only manifest themselves after a sustained period of operation. For example, the original trigger code that initiated writing and indexing would occasionally go out of synchronisation with the tweet stream. The issue only manifested itself after a number of days running, which meant during this time the integrity of the data streamed was unknown. Only by monitoring logs gathered over time did a slow network connection reveal itself, making the application go out of sync with tweets being streamed. Although a minor fix, the debugging process cost both time and data.

Similar to this situation were issues in the daily mergers of indices. In that scenario use of meta-index reverse keys was causing the UID lookup index merge process to run indefinitely. Before discovering this was the case extensive attempts were made to try and complete the merger with the property enabled. However, each attempt required several hours before failure could be safely declared and a new approach tried.

## 4.11 Performance Analysis

The performance of the data processing application was measured using two metrics:

- *Uptime*: how long was the application active during the run period?
- *Data Integrity*: how many indices were created succesfully vs. those that were corrupt?

| Test Case ID | Description |
|---|---|
| 1 | Directory for merging has 24 hourly indices |
| 2 | Directory for merging has <> 24 hourly indices |
| 3 | Clean up initiated on dailyIndices folder |
| 4 | 1+ supplied indices are corrupt |

Table 4.12: Test Cases: Merging

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Indices directory with 24 indices | 1 daily index directory | Success: directory with single daily index created |
| 2 | Indices directory with ¡¿ 24 indices | 1 daily index directory | Partial: as test case 1 if even number of index directories, failure if odd number |
| 3 | Directory cleanup configured on daily index directory | Removal of all interim indices | Failure: file in use error |
| 4 | 1+ corrupt indices | Ignore index and continue with remainder | Success: corrupt indices ignored and remainder indexed |
| 5 | Corrupted tweet file | Ignore file and continue indexing | Success: file ignored and error logged |

Table 4.13: Tests: Merging

- *Mean Time Between Failures (MTBF)*: how long does the application run between each crash / restart cycle?

Two major versions of the application were developed, one storing data in HBase, the other using Terrier indices.

The HBase application ran for periods of 2-4 days before failure. This often occurred due to an unexpected error within the HBase cluster. Unfortunately early versions of the application had no logging capability and so such failures went unmissed for hours or, in some cases, days. For this, and the slow scan speed whilst searching the database, HBase was discarded as a data store option (though tweets streamed during this time have been backed up for future use).

The Terrier-centric application was significantly more robust. After component testing the application was run from January 22nd to March 4th, during which logs were created detailing the success or failure of each function in the system on an hourly basis. Table 4.14 lists events that occured during this time.

The potential uptime for the system between 22/1/11 and 4/3/11 was 1008 hours. The system was inactive or offline for a total of 81 hours, giving a total uptime figure of 91.9%, with a MTBF of 359 hours ( 2 weeks).

Part of the reason each downtime period was so lengthy was that no-one was informed when a failure occurred. The application requires a notification system to alert a human operator each time there is a failure that requires attention.

| Date | Event Type | Reason | Downtime | Action Taken |
|---|---|---|---|---|
| 24/1/2011 | Software Failure | Java heap memory set too low, application failed | 28 hours | Restarted application with Java '-Xmx' option increased |
| 1/2/2011 | Software Failure | Unable to reconnect after connection dropped | 5 hours | Fixed reconnect bug, launched application |
| 25/2/2011 | Hardware Failure | Power disconnected to streaming machine | 48 hours | Restarted application on power reconnection |

Table 4.14: Robustness Analysis

Furthermore the machine running the application was not protected with an uninterruptible power supply. In a real-world scenario the application would be running on a machine or cluster which could guarantee certain physical properties (failover to UPS on power failure, or seamless transition to another server on hardware failure using virtualisation).

The application recovered well from each error. Where possible it restarted from a previous run, ensuring to check for existing data folders. If a day had been missed it created new directories and simply restarted streaming.

In terms of valid indices the application performed extremely well. It is testament to the stability of Terrier that every index created during the run period was valid, with no errors bar memory requirements occurring during the whole period. Furthermore each set of hourly indices were successfully merged into the appropriate daily index for every full day of stable operation.

## 4.12 Conclusion

The data processing application is a complex, timed workflow that was designed to be robust, stable, reliable, trustworthy and largely autonomous. Where errors do occur they are appropriately logged and usually gracefully recovered from — a fact confirmed by extensive and thorough testing.

The structures and indices created for holding data, both on the file system and using Terrier, allow for organised access to all the data required by client-facing sections of Pulse. These applications can rely upon the data processing application to provide them with tweets and statistics that are guaranteed to be correct and non-corrupt.

The next chapter discusses how the data gained and stored by the processing application is used to provide a rich user experience on the client-side.

# Chapter 5

# Pulse User Application: Client-Side

The client user application is the most important component within Pulse, providing a way to query all of the data stored, and interact with the results in a dynamic, exploratory and interesting manner. It supplies numerous views of the data retrieved, and permits searching with multiple configurable parameters. This section describes the approach taken in planning, structuring and building the user application.

## 5.1 Application Requirements

The application requirements were split into two categories: what the *user* must be able to do (user requirements), and what the *system* must be able to do to support the user (system requirements). Each of these is discussed further in this section.

### 5.1.1 User Requirements

Development of the Pulse client application was focused on three user interaction concepts, which represent the main actions users should be able to perform using the system:

- *Query*: users should be able to issue advanced queries, covering both topics and time, with advanced configurable options
- *Analyse*: users should be assisted in the analysis of Twitter data using visualisation techniques, filtering and aggregated statistics
- *Manipulate*: users should be able to manipulate data beyond what is initially presented, both in single units of functionality and across the whole system

From these interaction concepts concrete, evaluable user requirements were established. Most importantly users must be able to immediately query the system in an intuitive way. The interface should clearly indicate what is required of the user, and should model typical website user interaction where appropriate.

Multiple query types must be available to allow users to search based on both particular topics of interest and over periods of time, with results providing a comprehensive view of the query domain. Switching between query types should be implicit based on user input, rather than an explicit choice.

'Components' – logical units giving a particular perspective on the result set — should give a broad, informative overview on initial response, from which users can investigate further by combining queries or by 'drilling down' for more specific detail. Information displayed within each component should be unique to conserve screen space and eliminate redundancy. Components are described in Section 5.7.

The system must be easy to use by novices, but also provide a highly customisable experience for advanced users. The complexity associated with advanced use should be masked for those not requiring it, yet easily available should the need arise.

Users must be able to compare the results of different queries to observe differences in tag use and time periods. Components used for comparisons should provide additional functionality making clear the differences between each result set.

The analysis performed by the system must be accurate and aid users in identifiying trends within a single result set and across multiple result sets. The data used to perform such analyses must be available for the user to review.

Querying and navigating the system to find particular pieces of information should be intuitive. Given the large volume of data available for searching desirable data must not become obfuscated by poor structure or a cluttered user interace.

The application must encourage user exploration by providing organic paths to move from one query to the next. Where appropriate information related to a query should be provided in addition to the direct query result set.

Finally, all functionality should be clearly explained, either explicitly using online help, or implicitly through the design of each component.

### 5.1.2 System Requirements

The functional requirements of the system developed naturally once user requirements had been established. *N.B.* — these requirements applies to the system as whole. Each system layer and individual component has its own specific requirements discussed in the appropriate section.

- Given a topic-based query the system must generate a set of components containing relevant data and analysis of a single facet of the result set.
- Given multiple topic-based queries the system must allow for comparison of each result set.
- Given a temporal range query the system must display trending content from the Twitter domain across the applicable time range, highlighting changes and identifying patterns.
- The application must have some way of retrieving data from the Pulse data directory (created by the data processing application).

- Result sets should be cached once retrieved to allow quick regeneration of the appropriate components later in a Pulse session.

In addition to these absolute requirements the application should exhibit the following qualities:

- *Portable*: as a browser application the system must be able to run across all popular browsers. Fallbacks should be in place where a browser does not support certain functionality.
- *Usable*: the client application must be easy to use and learn. Functionality should be clearly labelled and help provided where necessary.
- *Extensible*: components must be developed in a uniform way to allow for future functionality additions. Developing new components must be a technically simple process.
- *Configurable*: a full suite of options should be available for users to specify exact parameters for system use and each query.
- *Quick Development Cycle*: given the time available for development the system must be structured in such a way as to allow for quick prototyping and development of new functionality.

## 5.2 Design Principles

The Pulse client application was designed in adherence to the following principles:

- *Application, not Website*: the finished application should take advantage of the fact it is running in a browser, but function like a fully-fledged desktop application.
- *Colour with Function*: though the application should be visually appealing, colour should also be used for emphasis, comparison and as a meaningful metaphor when displaying results.
- *Show Don't Tell*: data returned from queries should (at least initially) be displayed in a visually compelling way, rather than simply providing raw numerical data.
- *Clarity, not Brevity*: the comprehensiveness of results should not be overshadowed by poor application organisation or 'visual chaos'. Components should be distinct from each other, stay within their outer boundaries and be clearly delineated.

All UI elements are styled using Cascading Style Sheets (CSS). CSS is typically used by websites for uniformly applying style and structure to HTML elements. Though most of the structural positioning of Pulse was performed using GWT panels, individual elements were styled to maintain a system-wide appearance. A minimal colour palette of white, turquoise and navy blue was chosen for the user interface, providing a clean interface once the application loads.

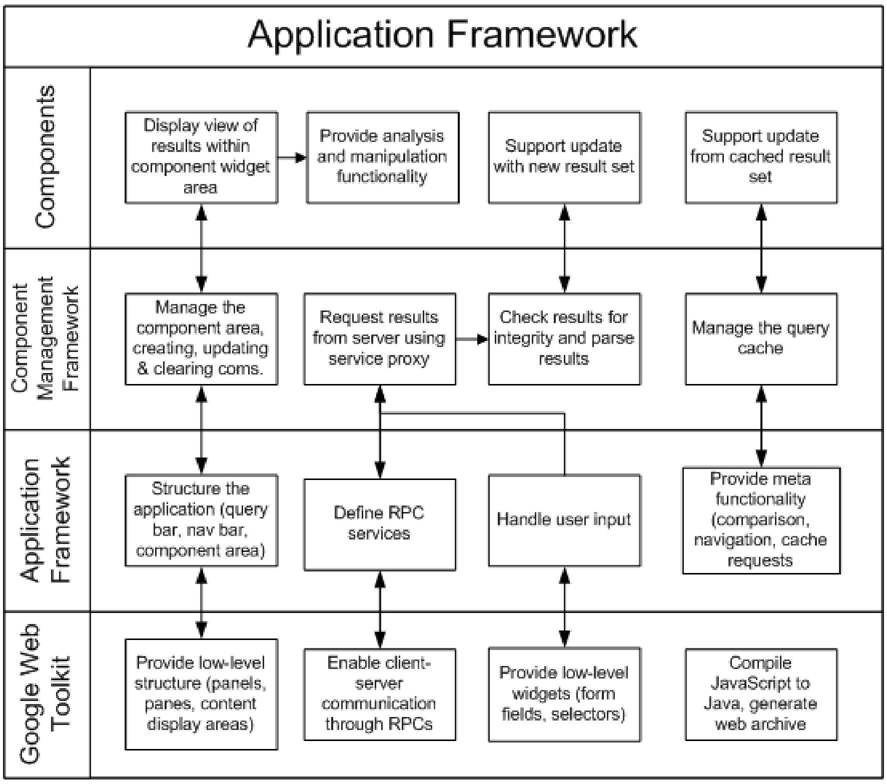

Application Framework
Components
Component Management Framework
Application Framework
Google Web Toolkit
Display view of results within component widget area
Provide analysis and manipulation functionality
Support update with new result set
Support update from cached result set
Manage the component area, creating, updating & clearing coms.
Request results from server using service proxy
Check results for integrity and parse results
Manage the query cache
Structure the application (query bar, nav bar, component area)
Define RPC services
Handle user input
Provide meta functionality (comparison, navigation, cache requests
Provide low-level structure (panels, panes, content display areas)
Enable client-server communication through RPCs
Provide low-level widgets (form fields, selectors)
Compile JavaScript to Java, generate web archive

Figure 5.1: Application Architecture

## 5.3 Architecture

Figure 5.1 illustrates the client application architecture, divided into four distinct layers.

Pulse has been designed so that system functionality is divided logically across these four layers. At the lowest level is the Google Web Toolkit (GWT) [20] application framework. As discussed earlier GWT provides a development toolkit for building complex web applications, where code is written in Java and compiled into JavaScript. Once compiled GWT applications can be deployed on a Java web server as a Web Archive (WAR) which conforms to the Java Servlet 2.5 specification [24].

A full analysis of GWT is given in Chapter 3, however in practical application terms GWT provides:

- A comprehensive library of standards-compliant widgets, structural ‘panels’ and application layouts for building applications.
- A mechanism for performing client-server communication: GWT RPC.
- Java development tools vastly superior to any JavaScript tools available
- Abstraction layers masking common JavaScript / AJAX application complexities such as XML HTTP request and cross-browser inconsistencies.
- Quick deployment of highly optimised code onto a compatible Java server.

GWT organises the application in the browser window using a *Layout Panel*. Layout Panels are standards-compliant content areas used to structure application layout at the highest level. More specifically, Pulse uses the *DockLayoutPanel* to position three content areas (shown in Figure 5.2). This is then attached to the *RootLayoutPanel* — a singleton implementation of *LayoutPanel* which itself is attached to the HTML document body.

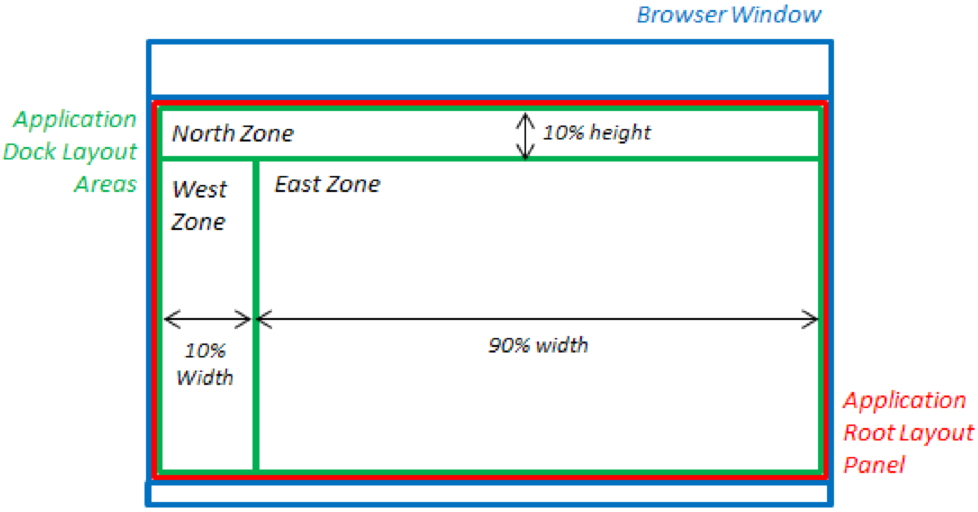


Figure 5.2: Dock Layout Panel

The North Area of the DockLayoutPanel is used by the Query Bar, and contains structure panels that hold the query form together. The West Area contains a *StackLayoutPanel* within which is the Navigation Bar. The StackLayoutPanel is tabbed so as to hide any content not currently selected. The East Area is managed entirely by the component manager, used for displaying components. For more specific details of how these areas are structured see Section 5.5.2.

Above GWT is the Pulse Application Manager. This layer combines simple widgets and content panels into more complex entities for placing in each of the GWT DockLayoutPanel areas. In addition the application manager handles almost all user input, from the Query Bar and the Navigation Bar.

Once requests are received by the Application Manager they are passed to the Component Manager. This layer is responsible for executing all queries and displaying the relevant results. It also manages the session query cache.

Components make up the final layer of the client application. These are content units which manipulate and present data retrieved by the component manager to the user in a visually appealing way.

## 5.4 Query Types

Pulse features three types of query which can be issued by the user. Each presents a particular view of Twitter activity in a unique way — by topic, by topic comparison, or by time.

### 5.4.1 Hashtag Queries

Hashtag queries are the simplest query type in Pulse, issued by typing a hashtag in the search box and selecting a date to search. The resulting components provide an overview of tag usage on the date selected, including temporal, geographical and sentiment analysis.

The results of hashtag queries are displayed using the following components:

- *Tagged With Component*: shows the query tag and its volume, along with all other tags that co-occur in each result tweet
- *Location Component*: marks each tweet with a marker on a world map, with the tweet text viewable by clicking on the marker
- *Sentiment Analysis Component*: provides sentiment analysis of tweets featuring the query tag
- *Tag Volume Component*: shows the distribution of tag usage over the course of the date selected
- *Geographical Heatmap Component*: shows the total number of tweets per country featuring the query tag
- *In Context Component*: provides additional contextual information about the query tag's usage.

Each of these components is explained in depth in Section 5.7.

### 5.4.2 Slashtag Queries

Slashtag queries allow for comparison of two tags on a given day. They can be issued by inputting a pair of tags in the search box, separated by a slash, and selecting a date to search. The resulting components provide comparison of results for the two tags specified, indicating differences and similarities in usage between each.

The results of slashtag queries are displayed using the following components, each a modified version of its hashtag query counterpart:

- *Tagged With Duo Component*: shows both query tags and their respective volumes, as well as all co-occurring tags. In addition highlights co-occurring tags common to both query tags, and in what volume.

- *Location Duo Component*: marks each tweet from the two result sets with markers on a world map, a different colour for each query tag.
- *Sentiment Analysis Duo Component*: provides sentiment analysis of tweets featuring the query tags, as well as sentiment comparison.
- *Tag Volume Component*: shows the distribution of each tag's usage over the course of the date selected
- *Geographical Heatmap Component*: shows the total number of tweets per country featuring the query tags as two maps, as well as a comparison map showing which tag was most popular in each country.
- *In Context Component*: provides additional contextual information for both query tags.

### 5.4.3 Temporal Range Queries

Temporal range queries in Pulse provide an overview of Twitter activity over a range of days, including top tags, most retweeted tweets and the most popular links. Users can select a range of days and compare differences over time, amongst other configurable options. Temporal range queries can be issued by leaving the search box empty and selecting a date. The Options panel can be used to amend more specific configuration settings.

The results of temporal range queries are displaying using the following components:

- *Trends Component*: a pivot chart which displays the most popular tags for multiple days in a highly customisable way. Tags can be ordered and tracked via numerous variables and displayed using many different visualisation techniques.
- *Place / Device Component*: shows which devices were used most often to tweet with, as well as the most actively tweeting countries across multiple days.
- *Sentiment / Volume Component*: highlights daily Twitter sentiment, in addition to daily tweet volume.
- *Retweet Paths Component*: displays the top retweets for a selected day, along with a map showing how they propagated across the world.
- *Popular Links Component*: lists the most popular links from Twitter on the selected day in a visually appealing way.

### 5.4.4 Cached Queries

Both hashtag and slashtag queries are cached so that they can restored later without making a server request. Because of this all tag-based queries can also be compared using the Compare Panel (see Section 5.5.4) — creating a slashtag view of two cached query tags.

## 5.5 Layer 1: Application Framework

The application framework structures the user interface into logical areas, handles most user input, defines the client-side elements of GWT services and provides meta functionality applicable to all queries.

### 5.5.1 GWT Services

A GWT RPC service ('service') allows for communication between the client application and a server using RPC ('Remote Procedure Calls'). This allows the JavaScript client application to asynchronously request data from a Java service, much like an XML HTTP Request in an AJAX web application.

Figure 5.3 illustrates the class structure of each service (adapted from the GWT Anatomy of Services diagram [21]).

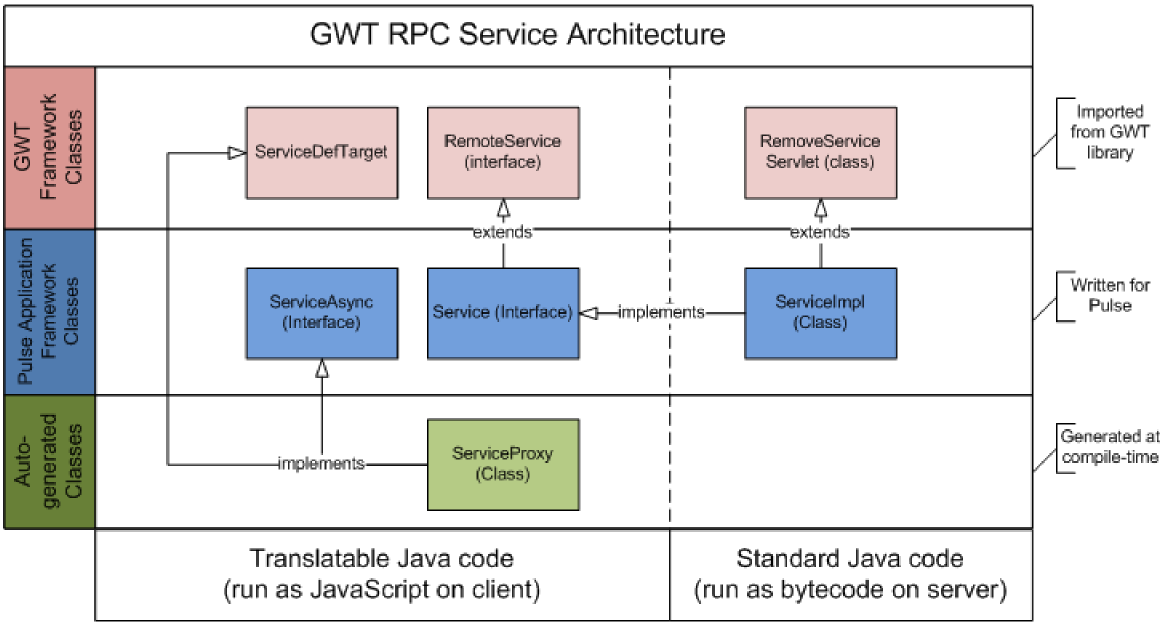

GWT RPC Service Architecture
GWT Framework Classes
Pulse Application Framework Classes
Auto-generated Classes
ServiceDefTarget
RemoteService (interface)
RemoveService Servlet (class)
extends
extends
ServiceAsync (Interface)
Service (Interface)
implements
ServiceImpl (Class)
implements
ServiceProxy (Class)
Translatable Java code (run as JavaScript on client)
Standard Java code (run as bytecode on server)
Imported from GWT library
Written for Pulse
Generated at compile-time

Figure 5.3: RPC Service Structure

Services consist of client-side code defining the service interface, and server-side code implementing that interface. A concrete implementation of each service interface is generated automatically by GWT at compile-time. Because no implementation exists until compile-time services are accessed via RPC service proxy stubs, created by the component manager (discussed in Section 5.6.3).

Each service requires three classes be created:

- *Service (Interface)*: this is definitive version of the service as implemented by the server-side class, and specifies the expected arguments and return type of the service.

- *ServiceAsync (Interface)*: based on the Service interface ServiceAsync is used for creating the client RPC proxy stub.
- *ServiceImpl (Class)*: a concrete implementation of the Service interface, providing the required functionality for the service.

A typical example of a Service interface is shown below:

```
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("birdcall")
public interface TwitterQueryService extends RemoteService {
InitialResponse getChirps(Request r);
}
```

This class imports the required classes, extending *RemoteService*. The function that the server-side class is expected to implement is defined, and a name is also given to the service ('birdcall') which identifies the service within the application manifest.

The related ServiceAsync interface is shown below:

```
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface TwitterQueryServiceAsync {
void getChirps(Request r, AsyncCallback<InitialResponse> callback);
}
```

Instantiations of the auto-generated implementation of this interface are used client-side to make service requests. Importantly it must be supplied an asynchronous callback function which executes on return — all services within GWT are non-blocking, given that the client application runs in a single-threaded environment. The callback handles the response from the server, or failure if the service throws an exception.

These two classes, plus their server-side implementations, are defined for each service used by Pulse. The server implementations for each service are discussed in Chapter 6.

### 5.5.2 Application UI Structure

Figure 5.4 shows the general structure of the application.

In line with the design goals of the application the basic structure is as minimalistic as possible. Core system functionality is immediately obvious in the Query Bar, whilst advanced options are hidden within the Options Panel of the Navigation Bar until selected.

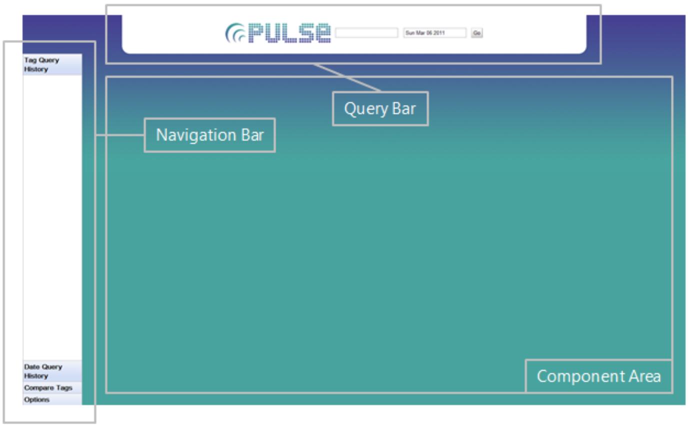


Figure 5.4: Application UI Structure

Most of the screen space is dedicated to the component area, only populated once a query has been issued. Given the number of options, queries and features Pulse offers it was important to ensure the user wasn't intimidated at first glance by the interface. Options are set to sensible defaults so that users get interesting results from their very first query, with focus placed on the component area by fading in components before they display results.

The remainder of this section explains each unit of the UI in detail.

### 5.5.3 Query Bar

The Query Bar, shown in Figure 5.5, is the top panel of the user interface, allowing the user to submit queries to Pulse for processing by the component manager. It is also responsible for forming requests based on user input and passing these to the component manager.

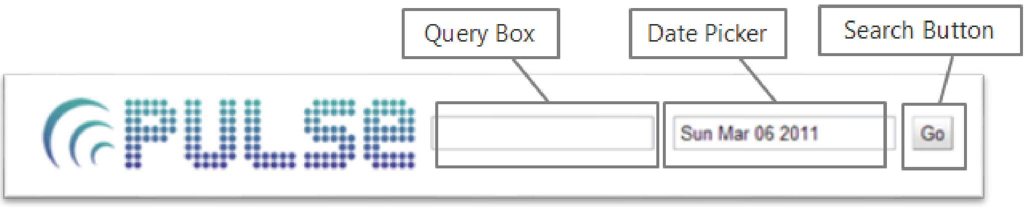


Figure 5.5: Query Bar

The Query Bar features three main input widgets:

- *Query Box*: a simple text box for entering hashtag queries. Users can enter tags with or without a '#' symbol — Pulse will automatically assume tags missing the symbol are hashtag queries. It also handles slashtag queries, of the form '#<tag1>/#<¿tag2>'. Pressing 'Enter' whilst focus is on the search box will initiate querying.

- *Date Picker*: a text field which when clicked displays a calendar widget for selecting a date. This field cannot be directly typed in; a date must be selected by using the popup widget. This provides a guarantee to the server-side service that the date selected is valid when looking for the correct index to load.
- *Search Button*: clicking this button will create a new request using the values in the search box and date picker, and starts the query process. It will also disabled the button until the query has completed (i.e. all components have been updated).

Once the 'Go' button has been clicked the Query Bar creates a new request to pass to the component manager. The type of this request is decided by checking the combination of values within each input field — if the search box has no value then the query is deemed to be for a temporal range. If the search box has any text then regular expressions are used to check whether it is a hashtag or slashtag search.

Once the request type has been established options are validated and added to the request. The option values are taken from the Navigation Bar Options Panel, discussed in Section 5.5.4.

### 5.5.4 Navigation Bar

The Navigation Bar, shown in Figure 5.6, occupies the left panel of the user interface. It is used to display query histories, allow comparison queries and configure Pulse options. To conserve space it uses a 'stack panel' widget, which displays only one panel at a time, the others accessible by clicking on the relevant tabs.

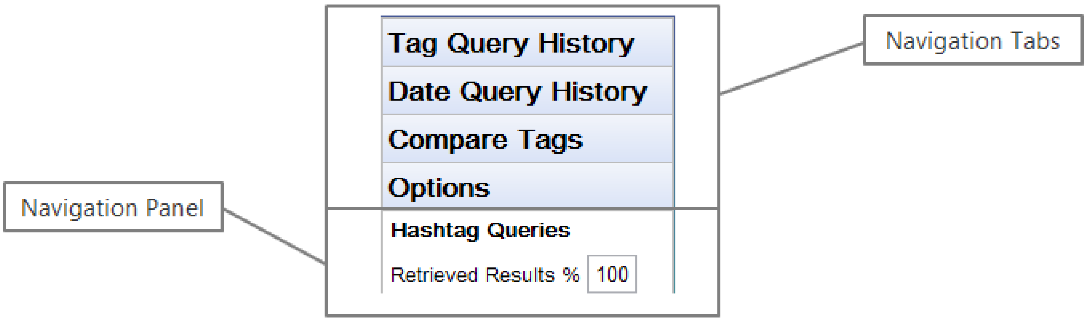


Figure 5.6: Navigation Bar

#### Tag Query History Panel

Figure 5.7 shows the Tag Query History Panel, used to show a list of all tags queried for during the current Pulse session.

Each tag within the panel corresponds to a single query. When clicked, the results from the selected query are loaded from the query cache into the component area. Furthermore hovering over a tag

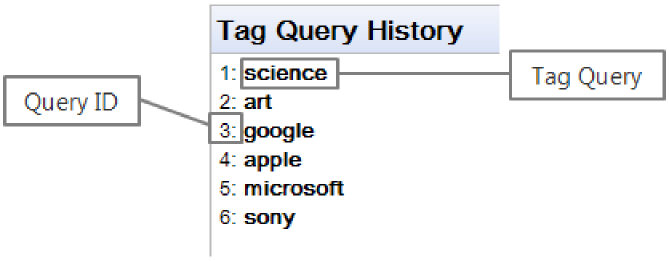


Figure 5.7: Tag Query History Panel

displays an info window giving more specific information about the query, such as for what day it was issued and what options were set.

Tags are added to this panel during the update workflow by the component manager. However, it is the Navigation Bar that constructs the relevant event handlers, builds a popup pane to show on hover, and positions the tag.

### Date Query History Panel

Figure 5.8 shows the Date Query History Panel, used to display a list of all dates queried for using temporal range queries during the current Pulse session.

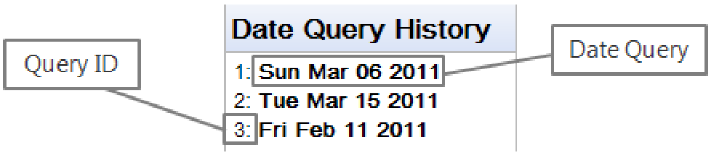


Figure 5.8: Date Query History Panel

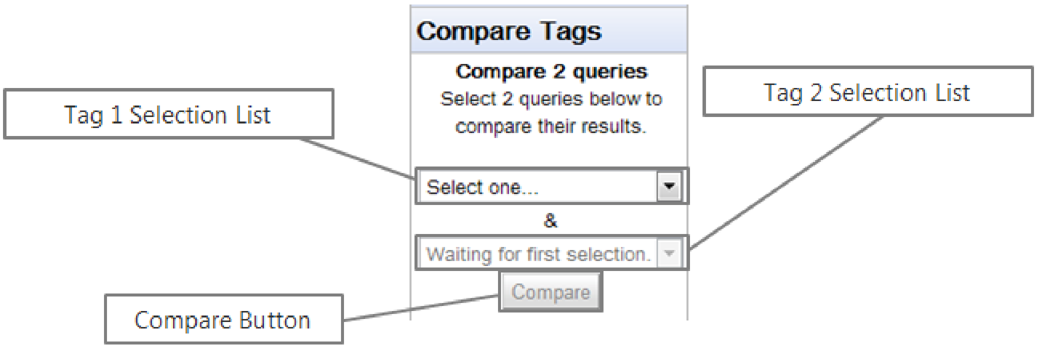
Each temporal range query is listed as an individual date entry. This panel serves only as a record of temporal range queries issued — this query type is not cached so no actions other than review can be performed.

Dates are added to this panel during update workflow by the component manager.

### Compare Panel

Figure 5.9 shows the Compare Panel, used to issue tag comparison queries.

All tag-based queries issued during the current Pulse session are listed within each tag selection list. Input validation is carried out to ensure that comparison is only available once at least two queries have been issued. Selecting two tags and clicking ‘Compare’ will initiate a new cached comparison query, sent to the component manager for handling.

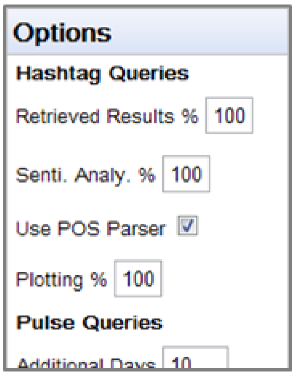

Compare Tags

**Compare 2 queries**
Select 2 queries below to compare their results.

Figure 5.9: Compare Panel

### Options Panel

Figure 5.10 shows the Options Panel (cropped for brevity), used to display configurable settings to the user.


Options

**Hashtag Queries**

Retrieved Results % 100

Senti. Analy. % 100

Use POS Parser ☑

Plotting % 100

**Pulse Queries**

Additional Days 10

Figure 5.10: Options Panel

The options available are split by query type — generally one set for tag-based queries, and another set for temporal range queries. There are several options relevant only to specific components, so each is discussed in detail within that component's description in Section 5.7.

## 5.5.5 Component Area

The component area, initially empty, is used to display all components for a given query. The component manager controls the component area, where during the update workflow components are generated, updated and cleared.

## 5.6 Layer 2: Component Management

The core content units of Pulse, components, are managed by a component manager. The component manager is responsible for accepting queries, requesting and parsing results, creating components and managing the process of updating each component. It is also responsible for tracking, storing and retrieving cached queries, passing this information to components when appropriate.

### 5.6.1 Purpose

Using a component manager to control each component, both in terms of structure and content, has a number of advantages. Primarily it allows for application modularisation and complexity masking; each component is unaffected by the success or failure of other components, is not required to handle any query input, and is unconcerned with its position relative to other components.

It also provides a single location for components to access the validated response to a query. Components can safely assume that the data being passed to it has been checked for integrity, and queries which return no results are caught before components are updated.

Structurally components need only be created with a relative width and height. This is then made absolute by the component manager within the bounds of the component area. Should application-level changes be made to Pulse's structure these are instantly propagated to each component.

Given the number of components and types of query it was also essential to have some entity which could oversee and operate an end-to-end update workflow, from user request to component presentation. This process involves communication between many different parts of the application, so it was necessary to have an intermediary which could orchestrate and implement each stage in the process.

Finally, cache storage and retrieval could become disorganised if each component had to individually determine whether it was required and if so, what cache was to be used. With a component manager Pulse can store, retrieve and propagate the appropriate cached results to the right components — with components only needing to know how they individually deal with those results.

### 5.6.2 Update Workflow

Figure 5.11 illustrates the workflow followed for each query, from request to completion.

The workflow is initiated by a request being formed by the Query Bar. This is then passed to the component manager. On receiving this request the *currentRequest* variable is set. This is available system-wide providing components with the current query tag, the query date, and the options configured. Then, a new query cache entry is created for storing each component's data cache. Existing components are cleared from the component area and new ones relevant to the current request type are generated.

Once all parties are prepared updating a request is made to the server via *TweetFetch*, the class which handles all interactions with the index query service (described in Section 6.4. Once finished the server sends a response back to the client, and subsequently the component manager.

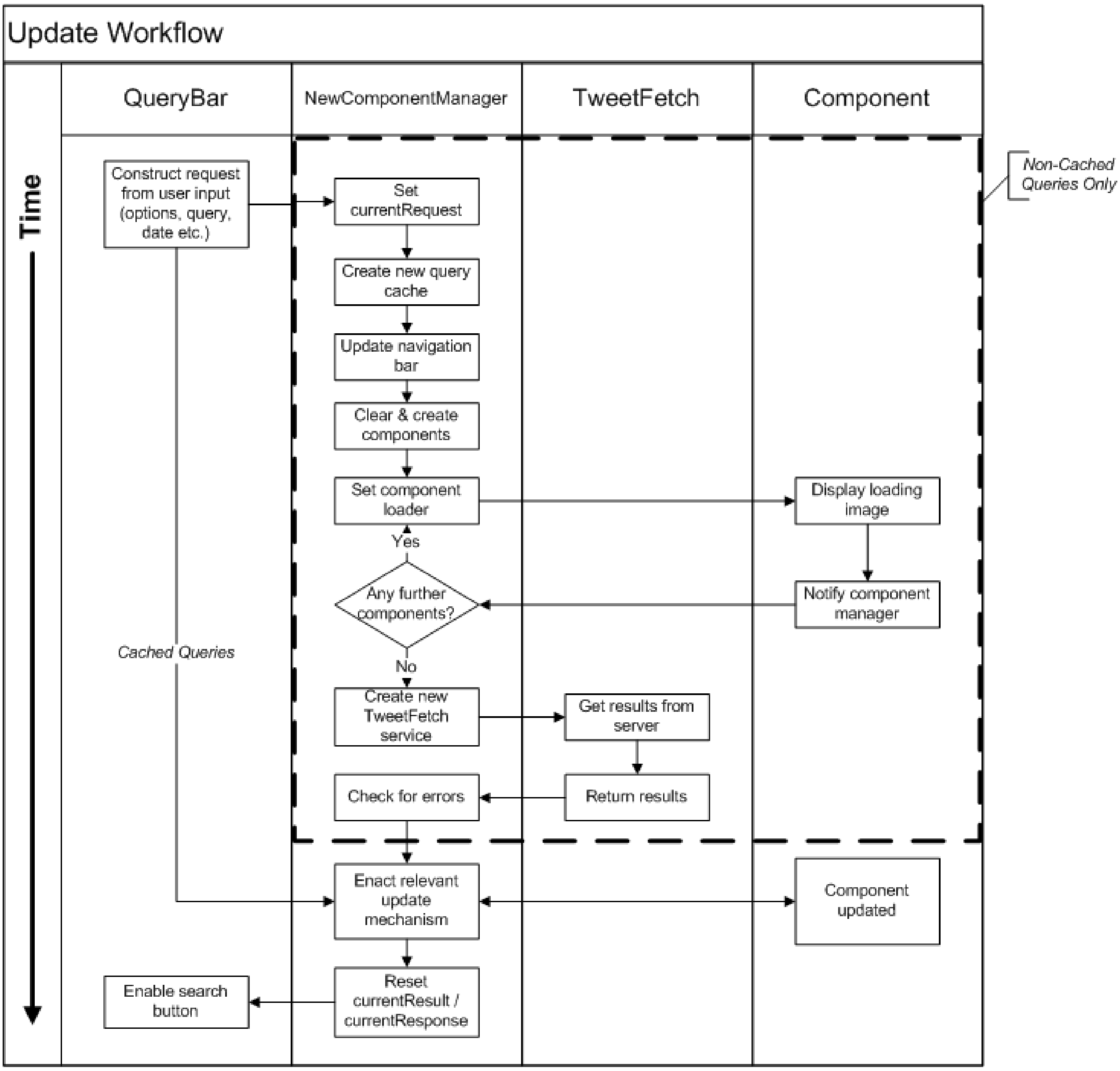


Figure 5.11: Update Workflow

The component manager first checks whether any results have been returned, and if so whether there are any errors in the results. If so, an error message is displayed to the user. If not, the results are parsed and each component is updated in turn using the appropriate update mechanism for that request (details in Section 5.6.8. Each component is obliged to explicitly inform the component manager when it has finished updating — this is to avoid issues with asynchronous callbacks returning after the component manager has completed the update workflow.

Once each component has been updated the Query Bar search button is re-enabled ready for the next query.

Each step in this workflow is explained in greater detail below.

### 5.6.3 Initialisation

Before Pulse can begin responding to queries various APIs and services must be initialised.

#### Internal Services

Pulse relies on server-side services to perform Java-language manipulation and data retrieval which would be otherwise impossible on the client. This is achieved using Remote Procedure Calls (RPC) between the client and server. To facilitate this RPC proxy stubs are configured for each service, and as these are reusable they are initialised on application startup by the component manager. Services can then be invoked by calling the appropriate method on the RPC proxy stub. Execution continues immediately after a service invocation, so a callback method must be supplied to handle the server response.

The actual implementation of these services is on the server. The implementation details are discussed in Chapter 6.

#### External APIs

Pulse uses several external APIs when responding to queries which are initialised by the component manager on application startup. A dialog will show within the component area until each has successfully initialised — until then, the system cannot be used. Initialisation requires an internet connection, as such Pulse can only be used on an internet-connected machine. The API initialised are as follows:

- *Google Maps API*: provides mapping and geocoding services to certain components
- *Google Visualisations API*: powers the majority of Pulse visualisations (graphs, charts, certain types of map)
- *Google Translate API*: performs natural language detection and translation.

Whilst GWT provides the JavaScript Native Interface (JSNI) for incorporating handwritten JavaScript (including external JS libraries) into Java code, it also provides Java wrappers for the most popular of its JS APIs. With the functionality provided by the GWT-wrapped and competitor's native libraries being essentially the same (mapping, charts and graphs etc.), the convenience of the GWT wrapped APIs was preferred.

### 5.6.4 Requests & Responses

Interaction with the server is conducted through requests and responses. Each time a new query is issued by the user a request is generated, of which there are three types: *Request*, an abstract class from which other requests are extended, *SingleRequest*, for single-tag queries, and *DuoRequest*, for dual-tag queries.

The *Request* abstract class stores information about a request common to all query types. For example, regardless of query type every request has some date filter. Similarly, every query type has some subset of options which apply.

The *SingleRequest* and *DuoRequest* classes are specific to queries which feature a hashtag. Each adds fields for hashtags; one in the case of *SingleRequest*, and two in the case of *DuoRequest*. These are used by hashtag and slashtag queries respectively.

Responses are generated by the server after processing a request. Different response are issued depending on the request type — further discussion of responses can be found in Chapter 6.

After each component has finished updating requests and responses are added to to the query cache. If a slashtag request was made it is split into two hashtag requests and stored as two separate cache entries. This means each query tag in a slashtag query can later be compared against any other query in the cache, rather than just the tag specified in the initial slashtag request. The query cache is explained fully in Section 5.6.6.

### 5.6.5 Component & Request Types

All components have a *component type*, which is included within one or more *request types*. Both component and request types are defined as Java enumerations, components in *ComponentType*, and requests within *RequestType*.

*RequestType* also has a method which returns the types of components required for a given type of request. This is used by the *NewComponentManager* when generating components in response to a query.

### 5.6.6 Query Cache

Each hashtag or slashtag query request and response is stored within the query cache, a store of all data retrieved and created throughout a Pulse session. The query cache allows users to quickly retrieve and display results generated previously, rather than re-querying for the same term.

Each query is stored as a *QueryCache*, a collection of all the data required to generate components for a query without making a server request. In order to use this data each component then specifies a means, through its *updateFromCache* method, through which this data can used to re-create the component.

During a query's initial update phase each component supplies the cache with a cache object specific to that component. Usually this will not contain the raw results of a query, rather the results after manipulation within the component. Though this is more storage-intensive (requiring each component to define its own cache requirements) it significantly reduces time to generate components from the cache — particularly where the component performs significant data transformation on the raw results of the query, or makes a server request. Individual component cache specifications are described in Section 5.7.

Access to the query cache is managed by the *QueryCacheManager*, providing safe access to the session cache. Components need not be aware of the wider state of the system, only that they must store their results in the cache supplied by the *QueryCacheManager*. In exceptional situations where a cache other than current query cache is needed by a component unsafe access is also available. This is primarily used by components which operate asynchronously, with results arriving after an update cycle has completed.

The *QueryCacheManager* consists of a list of *QueryCache*s (each an individual cache of data pertinent to a single query), and control methods for accessing specific cache entries. Components can send data to the cache manager via the component manager.

Temporal range queries are not cached by the query manager; rather the aggregate statistic files used by time-queries are preloaded on the server side — resulting in very quick response times.

### 5.6.7 Navigation Modification

Once a query has been accepted by the component manager it must notify the Navigation Bar so that it can be updated. This happens in two places:

- *Query History*: the appropriate history panel must be updated to include the new query, either 'Date Query History' for time queries, or 'Tag Query History' for hashtag queries
- *Compare Tab (tag queries only)*: a new entry in the 'Compare Tags' list is required, through which the tag can be selected for use in cached comparison queries

The Navigation Bar is responsible for making these changes — see Section 5.5.4 for further details.

### 5.6.8 Update Mechanism

The process followed to update components in Pulse varies based on the request type, and whether the update is from cache or not. The component manager deals with this complexity by selecting an update mechanism using the request type as the determining factor.

### Hashtag Queries

Figure 5.12 illustrates the update mechanism used during a single hashtag search.

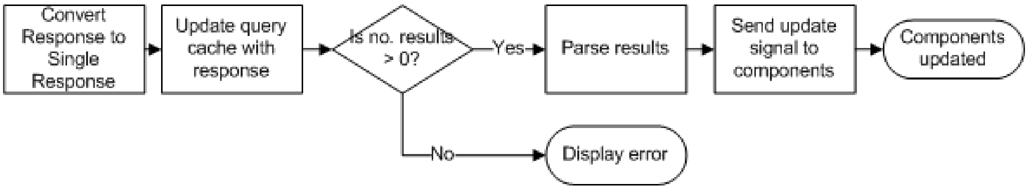

Convert Response to Single Response
Update query cache with response
Is no. results > 0?
Yes
Parse results
Send update signal to components
Components updated
No
Display error

Figure 5.12: Hashtag Search Update Mechanism

The response from the server is first cast to a *SingleResponse*, and set as the current response within the component manager. The query cache configured earlier is then updated with the server response. If the response has any results, these are then parsed into tweets from the JSON entity returned using *ClientTweetParser* — described in Section 5.6.9. Finally the update signal is sent to all components in turn, where each component then performs manipulation and presentation of the data as defined by its *update()* method.

### Slashtag Queries

Figure 5.13 illustrates the update mechanism used during a slashtag (dual hashtag) search.

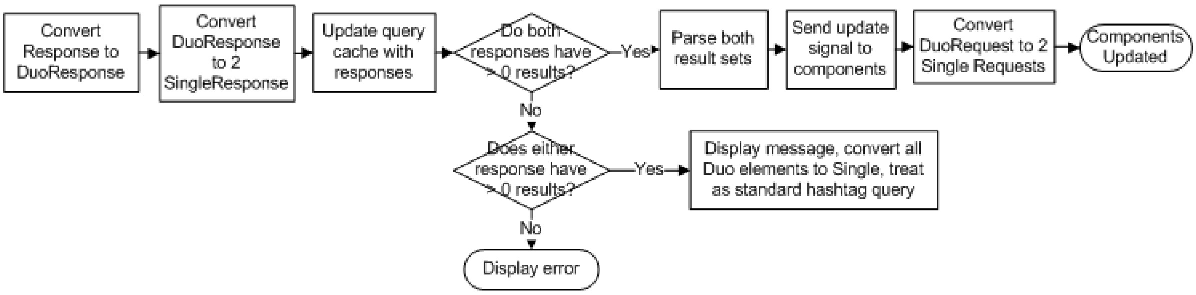

Convert Response to DuoResponse
Convert DuoResponse to 2 SingleResponse
Update query cache with responses
Do both responses have > 0 results?
Yes
Parse both result sets
Send update signal to components
Convert DuoRequest to 2 Single Requests
Components Updated
No
Does either response have > 0 results?
Yes
Display message, convert all Duo elements to Single, treat as standard hashtag query
No
Display error

Figure 5.13: Slashtag Search Update Mechanism

Slashtag requests require more management due to the possibility that both, one or neither query tag may return results. Furthermore each part of the query needs to be stored in the cache as an individual request / response pair to allow flexibility when selecting queries for comparison from the query cache.

To do this, this response from the server (a *DuoResponse*) needs to be split into two *SingleResponse* instances. Each of the two entries in the query cache is then updated with the appropriate response. Should either of the query tags return no results then an information dialog is displayed and the update mechanism changes to that of a single hashtag search. Similarly should neither query tag return any results then an error is displayed.

If both query tags have results then these are parsed, and the update signal is sent to each component in turn. Once all components have been updated the query cache for each tag is finalised, converting the *DuoRequest* into two *SingleRequests* and storing them appropriately.

### Cached Hashtag Queries

Figure 5.14 illustrates the update mechanism when restoring results from a cached hashtag query.

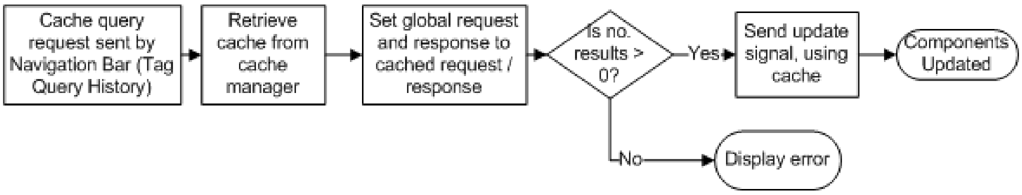

Cache query request sent by Navigation Bar (Tag Query History)
Retrieve cache from cache manager
Set global request and response to cached request / response
Is no. results > 0?
Yes
Send update signal, using cache
Components Updated
No
Display error

Figure 5.14: Cached Hashtag Search Update Mechanism

Cached update requests are always sent from the Navigation Bar: either by clicking on a tag in the Tag Query History, or by selecting two tags for comparison in the Compare Panel. These both store the cache ID associated with the query tag(s) for simple retrieval later. The component manager fetches the cache corresponding to the ID provided and checks to see if the query had any results. If not, an error is displayed informing the user. If so, all components in turn are updated from cache, using their *updateFromCache()* methods.

### Cached Comparison Queries

Figure 5.15 illustrates the update mechanism used when two tags from the cache are selected for comparison.

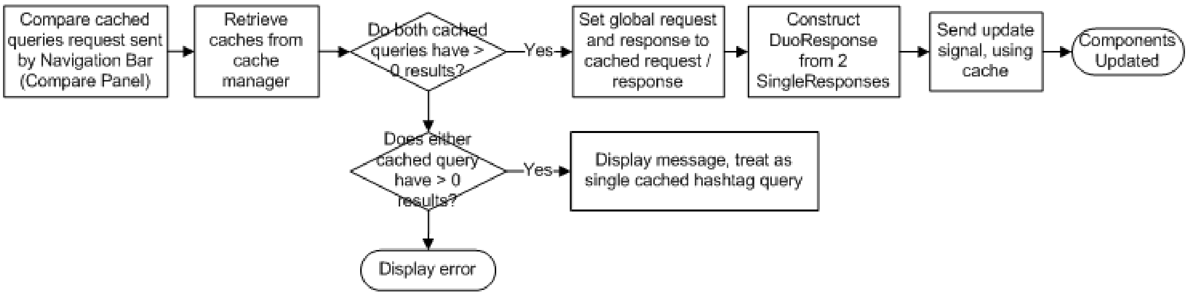

Compare cached queries request sent by Navigation Bar (Compare Panel)
Retrieve caches from cache manager
Do both cached queries have > 0 results?
Yes
Set global request and response to cached request / response
Construct DuoResponse from 2 SingleResponses
Send update signal, using cache
Components Updated
Does either cached query have > 0 results?
Yes
Display message, treat as single cached hashtag query
Display error

Figure 5.15: Cached Comparison Update Mechanism

Comparison queries effectively morph two hashtag queries into a single slashtag query for display within slashtag components. Like slashtag searches checks must be done to see if both, one or none of the queries had results. As with failed slashtag queries should only one query tag have results an error message is displayed and the update mechanism operates in cached hashtag mode.

However, if both query tags have results then a *DuoResponse* is created from the individual *SingleResponse*s. This is made available as the update signal propagates to each component. Once all components have been populated the system is ready for the next query.

### Temporal Range Queries

Figure 5.16 illustrates the update mechanism used for temporal range queries.

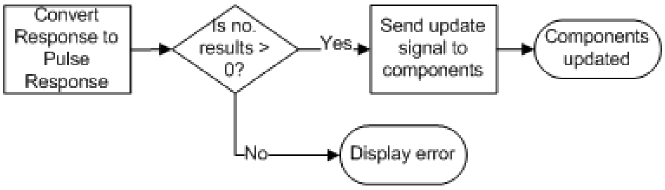

Convert Response to Pulse Response
Is no. results > 0?
Yes
Send update signal to components
Components updated
No
Display error

Figure 5.16: Time Query Update Mechanism

Temporal range queries are not cached client-side, which makes managing them considerably easier than tag-based queries. Furthermore the response returned from the server is constructed specifically for use by temporal range query components, with one hashmap of data for each. Though more structured than the respond from a tag-based query, the data provided is still general enough to be re-usable in custom components.

### Update Challenges

Implementing the slashtag query functionality meant a major rewrite of the way results were stored and components updated. This is reflected in the somewhat disjointed way that requests and responses are converted from single to duo and back again across the course of the update workflow.

Similarly, the addition of query caching later in development resulted in further modification of the main update process. This eventually lead to the update mechanisms described above, but each is only really suitable for its individual intended purpose.

The way this currently operates provides adequate functionality, but it would be difficult to extend it to multi-query comparisons or for different types of query. Given additional development time this would be restructured: further centralising the data required by each component, and reducing the number of transformations between request types.

## 5.6.9 Tweet Parsing

Tweets are parsed client-side by the *ClientTweetParser*. This class utilises JSNI methods to quickly parse the JSON results array into more manageable *Tweet* objects, later used to update each component.

The server-side tweet fetch service returns results as a string representation of a JSON array, with each element a single tweet entity. The *ClientTweetParser* uses the JS *JSON.parse()* function to convert the string into a *JavaScriptObject* — in GWT, a direct representation of a native JS object. The required data is then extracted using native JS methods, after which the newly-created *Tweet* object is added to the parsed list.

When parsing is complete the tweets are stored within the component manager for later use by individual components.

### 5.6.10 Component Area

The component area consists of all the space available within the user interface not taken by the Query or Navigation Bars. On creation components are added to the *component shell*, which fills the entire component area. Addition and removal of components in the component shell is also handled by the component manager.

## 5.7 Layer 3: Components

Components in Pulse are content units which perform manipulation and presentation of data retrieved from a query. Each component provides a unique view on the retrieved Twitter data, occasionally utilising services provided by server-side application. During execution components are generated depending on the query type made (hashtag, slashtag, comparison or temporal range), with each query type having a particular set of components utilising the returned information.

### 5.7.1 Component Specification

Each component is derived from the *NewComponent* abstract class. This class specifies basic functionality every component must have and provides functionality common to all components. It also defines the size and layout of each component. Concrete implementations must extend this class and define component-specific behaviour.

#### Component Structure

Each component within Pulse has the same generic structure, shown in Figure 5.17.

This structure is defined by *NewComponent*, which automatically configures the appropriate height and width given the size of the screen, as well as performing the required GWT initialisation and applying the appropriate CSS styles.

Components should display a name for the component, as well, as providing any additional toolbars that the component requires, inside the title bar. The widget area is space provided for presenting the results of that component's data manipulation.

#### Functionality

The *NewComponent* class defines functionality common to all components. This can be called by the component manager at any time with guaranteed results. These methods are used to control

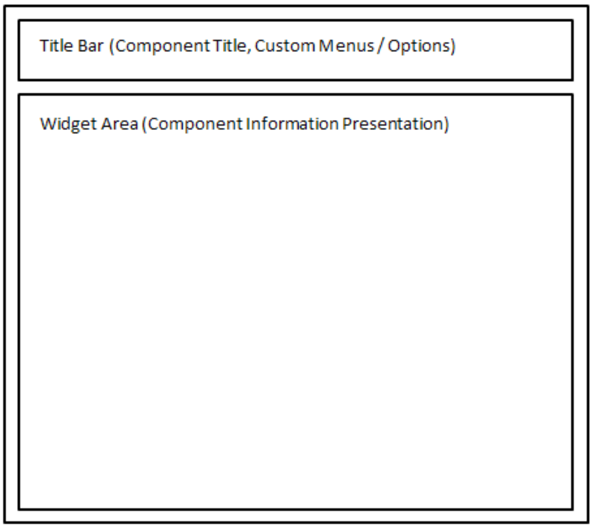

Title Bar (Component Title, Custom Menus / Options)
Widget Area (Component Information Presentation)

Figure 5.17: Component Structure

presentation of components in exceptional circumstances should component-specific handlers not be defined.

- *Loading*: a generic loader image is displayed whilst results are being retrieved or computed.
- *No Results Found*: displays a simple error message within the component when queries return no results.

Though these methods provide a fallback should they not be extended by a concrete component, certain functionality must be defined for components to function correctly within Pulse. These methods define what to do with result sets returned, how to handle cached updates, and how to clear the component.

- *Update*: The core function of a component is to manipulate and present data in some way after each query. The update method should implement the necessary transformations and visualisations.
- *Cached Update*: How to handle data returned not from a query, but from the query cache. Depending on what the component stores in the query cache this may differ greatly from a standard update.
- *Clear*: Components may have to reset certain properties or processes in addition to simply removing data from the widget area. The clear method should be implemented to bring that component to a ready state.

### 5.7.2 Creating New Components

Pulse was designed to be easy to extend and modify. As such the process of creating new components is simple.

- *Extend NewComponent*: components are based on the NewComponent class, which should be extended by all components. The NewComponent class is discussed in 5.7.1).
- *Create a component cache*: creating a component cache, and a method which can use this cache to build a new instance of the component, allows for fast display of previously-queried terms.
- *Define new ComponentType*: a new enumeration type should be created for the component, used as an internal name.
- *Assign ComponentType to RequestType*: the new component type should be added to one or more request type declarations, indicating what types of requests should generate the component.
- *Add rule to NewComponentManager*: the component manager needs to know what initialisation is required each time a component of that type is instantiated (e.g. supporting libraries or server-side services). To do this, a new rule should be added to the component manager *createComponents()* method.

### 5.7.3 Tagged With Component

The Tagged With Component, illustrated in Figure 5.18, shows hashtags that co-occur with the query tag in each of the tweets in a result set. It displays terms as a word cloud, with more frequently co-occurring tags drawn using a larger text size, with tags getting progressively smaller the less frequently they appears. These tags can be clicked on, initiating a new search for that tag.

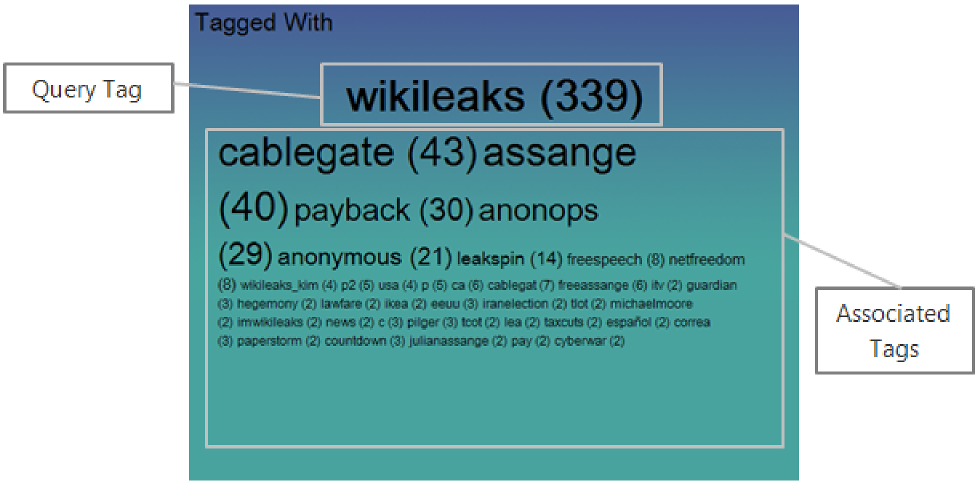

Tagged With
Query Tag
wikileaks (339)
cablegate (43) assange (40) payback (30) anonops (29) anonymous (21) leakspin (14) freespeech (8) netfreedom (8) wikileaks_kim (4) p2 (5) usa (4) p (5) ca (6) cablegat (7) freeassange (6) itv (2) guardian (3) hegemony (2) lawfare (2) ikea (2) eeuu (3) iranelection (2) tlot (2) michaelmoore (2) imwikileaks (2) news (2) c (3) pilger (3) tcot (2) lea (2) taxcuts (2) español (2) correa (3) paperstorm (2) countdown (3) julianassange (2) pay (2) cyberwar (2)
Associated Tags

Figure 5.18: Tagged With Component

**Purpose**

The component should provide context about the query tag, informing users of tags used in tandem with their query and in what proportions. This should be done in a visually compelling way that makes clear the most and least popular tags used in conjunction with the query tag.

It should also aid results exploration, allowing users to organically move from one query to the next by investigating links between particular hashtags. Allowing users to move from query to query as a natural process, instead of always using explicit searches, enables Pulse to act as a discovery tool as well as providing information on explicit queries.

Finally, it should clearly show associations between particular hashtags. By providing these associations users should be able to expand their query to hashtags they may not have otherwise searched for.

#### Motivation

Visualising co-occurring tags is useful for understanding tag usage context. This is especially applicable in fields such as brand management — for example, advertisers may wish to know what tags Twitter users are associating with a brand name. If users are tweeting negative tags (for example, '#broken' or '#fail') in conjunction with a brand then action can be taken to identify the reasons for user dissatisfaction and target those users. Usage of this technique is mentioned by Higham et al [50], achieved using Skyttle [32], a brand management tool which identifies influential web users across many social mediums in specific topic areas.

#### Data Requirements

The component requires the full hashtag list for each tweet in the result set.

The query cache used by this component stores the generated tag cloud as a *FlowPanel* (equivalent to an HTML <div> section), as well as a hashmap of tag/occurence pairs. This is used to generate the comparison table used by the Duo version of the component.

#### Cloud Generation

The algorithm used to generate the tag cloud is shown below in pseudocode.

```
for all tweets
    if tweet has tags
        if tweet has unique tag
            store new tag entry, set occurences to 1
        else
            update tag entry, set occurences to occurences+1
get min, max occuring tags
diff = max - min
if diff < 10
    numSizeSteps = 1 // the number of font sizes to use
else
    numSizeSteps = diff / 10
add query tag with num. occurences
set initial font size
```

```
for each size step // starts at max-size step, reduced by size step on loop
    for each tag
        if tag occurences >= sizeStep
            add tag & num. occurences, mouse handlers
            remove tag from tag list
    reduce font size
```

All tags that feature with the query tag are counted and stored as key value pairs. The difference is calculated, and if the difference ¿ 10 a number of size steps are iterated over. If the tag has more occurences than *(max occurring tag - size step)* it is added to the cloud with the current tag font style and removed from the tag list — several font styles are defined in CSS for each iteration, and each iteration the style is changed to a smaller font size. To ensure clarity tags that feature only once across all tweets are ignored. The final tag cloud is stored as a FlowPanel.

### Component Testing

Table 5.1 shows the test cases defined for the Tagged With Component:

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned |
| 2 | Hashtag query results have no associated hashtags |
| 3 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 4 | Cached query is selected |
| 5 | Tag from the result cloud is clicked |
| 6 | Tag from the result cloud is hovered on |

Table 5.1: Test Cases: Tagged With Component

Table 5.2 shows the tests derived from each test case, and the results of running each test.

### Duo Equivalent

The Duo version of the Tagged With Component allows for comparison of two word clouds from separate queries, shown in Figure 5.19.

As well as providing all of the functionality of the standard Tagged With Component, the Duo version provides a list of tags which appear in both result sets, and in what volumes (this can be generated from cache using the tag / occurences hashmap). The tag cloud container also features a vertical bar which can be dragged left or right to focus fully on either cloud.

This additional functionality resulted in four new test cases, shown in Table 5.3

These were implemented by the tests in Table 5.4.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | A hashtag query with results | Component displays query tag at top of component, volume in brackets, associated tags with volumes as a cloud | Success: component displays query tag, volumes and tag cloud |
| 2 | A hashtag query with results, none of which have any other hashtags apart from the query tag | 'Component displays query tag at top of component, along with volume in brackets | Success: component displays query tag alone |
| 3 | A slashtag query where only 1 tag returns results | Output from one of test cases 1 - 2 | Success: output is one of test cases 1 - 2 |
| 4 | A cached hashtag query | Quick display of output from test cases 1 - 3 | Success: data loaded from cache and displayed within component |
| 5 | A click on any tag within the result cloud | Tag is sent to search box, and 'Go' button is programatically clicked | Success: tag sent to search box and button clicked, initiating new query |
| 6 | Hover mouse over any tag within the result cloud | Tag colour changes to white until hover out | Success: on hover tag turns white |

Table 5.2: Tests: Tagged With Component

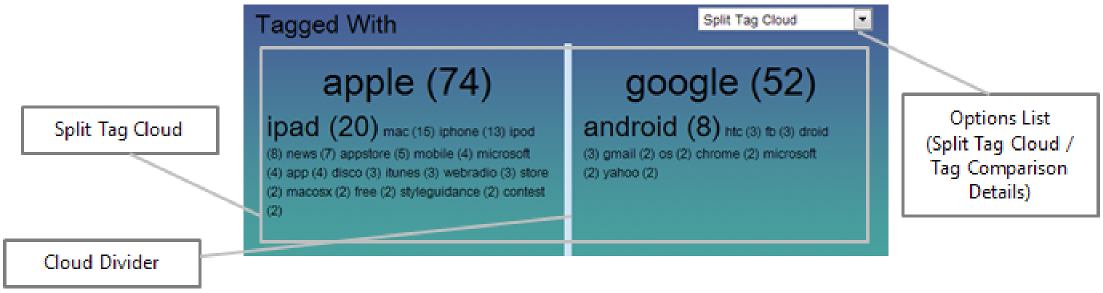

Tagged With
Split Tag Cloud
apple (74)
ipad (20) mac (15) iphone (13) ipod (8) news (7) appstore (5) mobile (4) microsoft (4) app (4) disco (3) itunes (3) webradio (3) store (2) macosx (2) free (2) styleguidance (2) contest (2)
google (52)
android (8) htc (3) fb (3) droid (3) gmail (2) os (2) chrome (2) microsoft (2) yahoo (2)
Split Tag Cloud
Cloud Divider
Options List (Split Tag Cloud / Tag Comparison Details)

Figure 5.19: Tagged With Duo Component

### 5.7.4 Location Component

The Location Component, illustrated in Figure 5.20, marks each tweet on a world map using longitude / latitude data. As very few tweets provide this natively the Google Maps API (initialised by the component manager) is used to 'geocode' (translate textual location data into a specific latitude / longitude pair) each tweet based on whatever location information is supplied.

| Test Case ID | Description |
|---|---|
| 1 | Slashtag query results are returned for both tags |
| 2 | 1 of 2 slashtag query tag results have no associated hashtags |
| 3 | Both slashtag query tag results have no associated hashtags |
| 4 | Duo cached query is selected |

Table 5.3: Test Cases: Tagged With Duo Component

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | A slashtag query with results for both tags | Split pane with 2 sets of a results displayed, query tag at top of component, volume in brackets, & tag cloud | Success: component displays split pane and 2 clouds |
| 2 | Slashtag query, only 1 tag has associated hashtags | Tag with results - as test case 1. Tag without - query tag at top of component, volume in brackets | Success: component split pane, 1 tag with cloud, 1 tag without |
| 3 | Slashtag query, neither tag has associated hashtags | Output from test case 2 | Success: component displays query tags and volumes |
| 4 | Duo cached query from the ’Compare’ navigation tab | Quick display of output from test case 1-3 | Success: data loaded from cache and displayed |

Table 5.4: Tests: Tagged With Duo Component

**Purpose**

The component has two main purposes. The first is to illustrate where exactly tweets for a particular topic are most popular in the world, indicated by large clusters of markers over locations with heavy Twitter activity featuring the query tag. It also allows users to investigate the individual text of tweets from a specific location.

**Motivation**

Twitter provides several geographic meta-data fields which can be used to map tweets. Several tools exist already which provide this functionality, such as TwitterVision [39], however as with many of the analysis tools evaluated in Section 2.2 most focus only on real-time mapping of all public tweets. Other web mashups exist combining Twitter data and Google Map functionality, usually for a very specific purpose. GeoChirp [17] is one such example, a location-aware utility showing only tweets posted by Twitter users geographically close by.

The Location Component implemented in Pulse primarily indicates popularity of tags by geographic location. Concrete examples of the questions it can answer include: ‘what are people saying about #google in China, and how does this compare to users in the US?’, or ‘which country featured most chatter about the #facup final?’. No existing tool was found that plots tweets *dynamically* retrieved by topic, where dynamically means in response to a search query.

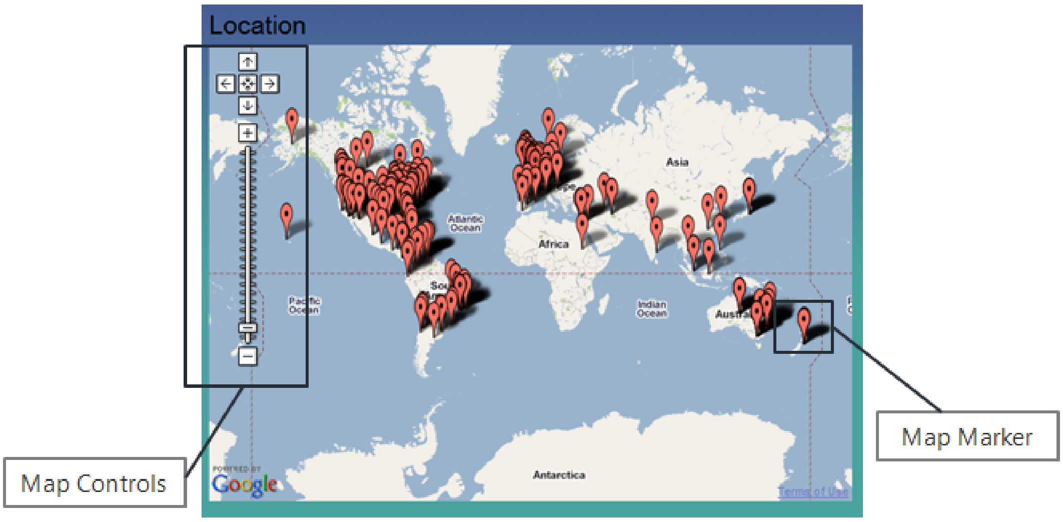


Figure 5.20: Location Component

### Data Requirements

This component uses the location field from each tweet for geocoding and the tweet text (displayed in a popup window after clicking a marker).

The query cache used by this component stores each marker (i.e. latitude / longitude pair) geocoded as an array. These can be instantly added to a map when retrieved form cache.

### Geocoding

As most tweets do not include explicit latitude / longitude data the location field of each must be geocoded. Helpfully the Maps API provides a *Geocoder* class which asynchronously attempts to geocode a given string. However, the API has both rate and frequency limits, which if exceeded will cause geocodes to fail. Typically in situations like these a scheduler or even *Thread.sleep()* could be used to stagger requests. However, with the final application compiled to JavaScript this was impossible.

A solution to this issue to use GWT's *Timer* class, a mechanism for implementing delayed logic. *Timer*s can be set to run on a schedule a specific number of times, with actions for perform specified in the *Timer*'s *run()* method.

The Location Component needs to throttle geocoding to roughly one request every 100ms. As geocoding is asynchronous a callback was also defined to handle the response from each geocode. The *ExtendedLocationCallback* stores the tweet text

With the complexity of managing the plotting and geocoding process increasing a manager class, *ScheduledMapFetch*, was devised. This class is responsible for holding the geocoding queue, running the geocoding timer, creating callbacks for each geocode, and plotting the results.

The final geocoding process is described in Table 5.5

| Action | Class |
|---|---|
| Extract tweet locations | MapComponent |
| Create new ScheduledMapFetch with reference to map | MapComponent |
| Add location list to ScheduledMapFetch queue | MapComponent |
| Execute ScheduledMapFetch fetcher | MapComponent |
| Create & schedule new Timer | ScheduledMapFetch |
| Start timer | ScheduledMapFetch |
| (Within Timer run() method) If queue $>$ 0, create new ExtendedLocationCallback with reference to map, tweet text and marker to use (if Duo) | ScheduledMapFetch |
| Make asynchronous call to geocoding service | ScheduledMapFetch |
| (On geocoder return) Pick most relevant geocode result, create info window with tweet text, plot on map | ExtendedLocationCallback |

Table 5.5: Geocode Process

### Options

The only Location Component option is *Plotting %*. This is a value between 0 and 100 representing how many tweets from the retrieved result set should be geocoded. This can be adjusted for queries which are likely to return a large number of results, where not all need to be mapped.

### Component Testing

Table 5.6 shows the test cases defined for the Location Component:

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned, map plot % = 100 |
| 2 | Hashtag query results are returned, map plot % = 0 |
| 3 | Hashtag query results are returned, map plot % $>$ 0 and $<$ 100 |
| 4 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 5 | Cached query is selected, with map markers |
| 6 | Cached query is selected, with no map markers |
| 7 | Map marker is clicked on |
| 8 | Component is geocoding existing query results when new query is issued |

Table 5.6: Test Cases: Location Component

Table 5.7 shows the tests derived from each test case, and the results of running each test.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | A hashtag query with results, map plot % option set to 100 | Map with markers as geocoded for each tweet | Success: map shown with markers for each tweet |
| 2 | A hashtag query with results, map plot % option set to 0 | Map with no markers | Success: map shown with no markers |
| 3 | A hashtag query with results, map plot % option set to between 1 and 99 | Output from one of test cases 1 - 2 | Success: map is shown with appropriate markers |
| 4 | Slashtag query where only 1 tag returns results | Output from one of test cases 1 - 3 | Success: component displays as if it were a hashtag query |
| 5 | A cached hashtag query with map markers | Output from one of tests cases 1, 3 | Success: map is shown with cached markers |
| 6 | A cached hashtag query with no map markers | Output from test case 2 | Success: map is shown with no markers |
| 7 | A click on a map marker | Info window displayed above marker with tweet text, any existing open info windows closed | Success: an info window is generated with tweet text, existing windows are closed |
| 8 | A new query issued whilst geocoding existing query | Existing geocoding is cancelled, and output is from one of test cases 1-4 | Success: geocoding is cancelled and restarted with new data |

Table 5.7: Tests: Location Component

### Duo Equivalent

The Duo version of this component shows markers for both queries in different colours. Additionally markers can be toggled on or off by clicking icons in the component title bar.

This additional functionality resulted in two new test cases, shown in Table 5.8

| Test Case ID | Description |
|---|---|
| 1 | Both maps have geocoded results to show |
| 2 | Only one map has geocoded results to show |
| 3 | Neither map has geocoded results to show |

Table 5.8: Test Cases: Location Duo Component

These tests are important — geocoding is a process separate to retrieval of results, so it may be that only one or neither result set has any markers to show. The Location Duo Component needs to handle each of these situations.

Checks for these conditions were implemented by the tests in Table 5.9.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Slashtag query where both result sets have 1 or more geocoded tweets | Map with differently-coloured markers for each result set, clickable markers to toggle visibility of result sets | Success: map shown with markers, toggles included in title bar |
| 2 | Slashtag query where one result set has 1 or more geocoded tweets | Map with one set of markers, no markers in title bar | Partial: map shows relevant markers, but title bar markers still appear |
| 3 | Slashtag query where neither result set has any geocoded tweets | Empty map, nothing in title bar | Partial: map has no markers, but title bar markers still appear |

Table 5.9: Tests: Location Duo Component

These tests highlight one of the difficulties of handling asynchronous requests. As the component doesn't know whether any tweet can be geocoded until it tries markers are added at the start of the update to the title bar. If all geocodes fail, or the plotting % option is set too low, no markers will appear on the map but the title bar markers are not removed. Whilst only a visual error it suggests the map has markers when it does not, potentially leading to user confusion.

### Future Extensions

The GWT wrapper of the Google Maps API provides only the official maps functionality. The native JavaScript library has unofficial extensions which allow for more advanced map overlays, such as clustering of markers over locations and customisable information windows. These could be includes via JSNI to add additional functionality to the marker popup windows such as user information, or highlight areas with intense Twitter activity.

### 5.7.5 Tag Volume Component

The Tag Volume component, shown in Figure 5.21, shows how tweet usage for a term was spread across the query date.

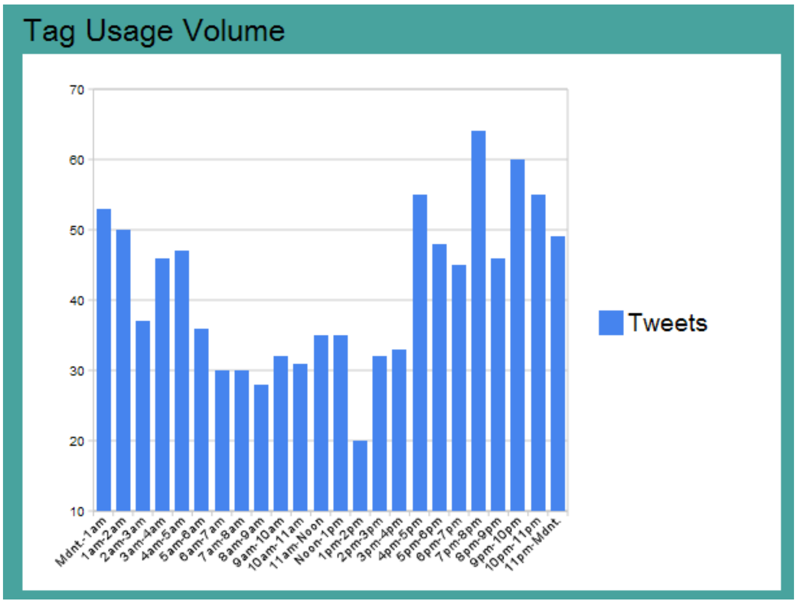


Figure 5.21: Tag Volume Component

#### Purpose

The functionality provided by this component is fairly rudimentary: a graph detailing how tweet usage varied, hour-by-hour, over the course of the query date. Envisioned uses include detecting how long after a breaking new story or event that Twitter chatter began, or judging when users of a particular hashtag are most likely to be active on Twitter.

#### Data Requirements

This component requires the publication date of each tweet in the result set.

The query cache used by this component stores a list of period names (for the X axis legend) and an array of numerical tweet volumes.

#### Data Preparation & Visualisation

The total number of tweets per hour is determined by iterating over the result set and checking the publication time against each period (by default there are 24 periods of 1 hour). This is then transformed into a data table, supplied to the Visualisations API constructor, which generates a new column graph.

#### Component Testing

Table 5.10 shows the test cases defined for the Tag Volume Component:

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned |
| 2 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 3 | Cached query is selected |
| 4 | Bar in the result graph is clicked |

Table 5.10: Test Cases: Tag Volume Component

Table 5.11 shows the tests derived from each test case, and the results of running each test.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | A hashtag query with results | Graph of hourly tweet occurences for given tag over day. | Success: component displays volume graph |
| 2 | A slashtag query where only 1 tag returns results | Output of test case 1 | Success: component displays volume graph |
| 3 | A cached hashtag query | Output of test case 1 | Success: component displays volume graph |
| 4 | Click on any bar within the result graph | Info window detailing time period and number of tweets | Success: info window displays time period information |

Table 5.11: Tests: Tag Volume Component

#### Duo Equivalent

The Duo version of the component displays the aggregated periodic volumes of each result set as pairs of columns for each time period.

### 5.7.6 Geographical Heatmap Component

The Geographical Heatmap Component, shown in Figure 5.22, enables users to quickly identify which countries most frequently tweeted a particular tag — without waiting for geocoding. The Duo version of this component also provides comparison functionality, highlighting which tag was more popular in a country, and by how much.

#### Purpose

This component was designed to avoid the wait time associated with the Location Component. Balancing processing time against result accuracy is a common challenge when updating components,

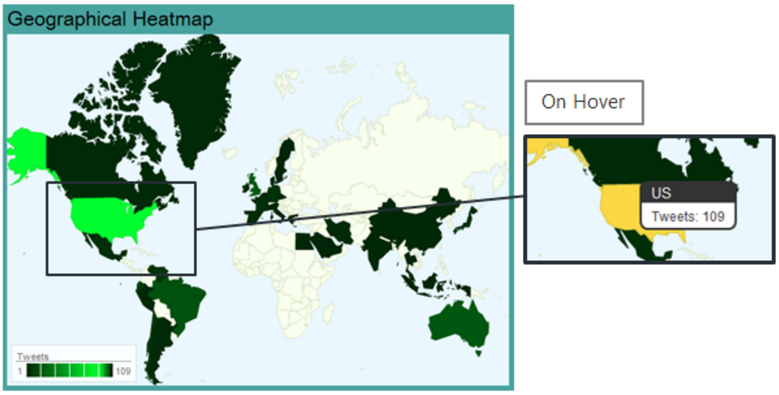


Figure 5.22: Geographical Heatmap Component

and the Geographical Heatmap component aimed to alleviate the inability to geocode instantaneously, whilst still giving a general overview of country-level Twitter activity.

Countries are coloured on a sliding gradient scale proportional to the number of tweets attributed to that country. Hovering over any country activates a popup panel detailing the country and number of tweets.

### Motivation

This component complements the Location component by aggregating tweet volume by country using a more appropriate visualisation. Users can very quickly establish where in the world a tag was most popular, and also in the Duo version of this component they can compare two tags' popularity using the comparison heatmap. Concrete questions that can be answered by this component include 'where did most tweets contaning #wikileaks appear?', or comparative questions such as 'which is more talked about in the UK - #labour or #conservative?'.

The only existing tool that was found featuring similar functionality was TweetVolume [6], however this has since been disabled for public use.

### Data Requirements

The component requires the timezone field of each tweet in the result set.

The query cache used by this component stores the calculated country / occurrence pairs as a hashmap.

### Timezone Lookup & Visualisation

As discussed previously real geocoding requires both external HTTP requests and time. However, tweets do provide an estimation of location via the tweet user timezone field. To quickly approximate the volume of tweets from a particular country this component compares the timezone field against a pre-defined list of common Twitter timezones and their respective countries. Although this is not as accurate as geocoding it does occur instantaneously, whereas geocoding 1000 results takes in the region of 100 seconds.

The country / volume list is then used to construct a data table powering the map visualisation, constructed by the Visualisation API.

### Component Testing

Table 5.12 shows the test cases defined for the Geographical Heatmap Component:

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned, timezone lookup returns at least 1 match |
| 2 | Hashtag query results are returned, timezone lookup returns no matches |
| 3 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 4 | Cached hashtag query with geo. heatmap results |
| 5 | Cached hashtag query with no geo. heatmap results |
| 6 | Country is hovered over |

Table 5.12: Test Cases: Geographical Heatmap Component

Table 5.13 shows the tests derived from each test case, and the results of running each test.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Hashtag query with results, geocoding returns at least 1 match | Geo heatmap with countries highlighted by total tweets | Success: heatmap displays with countries highlighted |
| 2 | Hashtag query with results, timezone lookup returns no matches | No results found error message | Success: error message displayed |
| 3 | Slashtag query where only 1 tag returns results | Output from one of test cases 1 - 2 | Success: output is one of test cases 1 - 2 |
| 4 | Cached hashtag query with geo heatmap results | Quick display of output from test case 1 | Success: heatmap displays with countries highlighted |
| 5 | Cached hashtag query with no geo heatmap results | Quick display of output from test case 2 | Success: error message displayed |
| 6 | Hover over country | Info window above country with country name, number of tweets, nothing if country has no tweets | Success: info window displayed |

Table 5.13: Tests: Geographical Heatmap Component

### Duo Equivalent

The Duo version of this component features geographical heatmaps for both result sets, as well as a comparison map highlighting country-by-country which of the two query tags was most popular. These can be switched using a drop-down list in the title bar of the component.

As with the Location Component the timezone lookup process may return results for only one or neither result set — both scenarios should be handled gracefully. The test cases in Table 5.14 confirm this is checked by the component.

| Test Case ID | Description |
|---|---|
| 1 | Both result sets have 1 or more countries identified from tweet timezones |
| 2 | Only one result sets has 1 or more countries identified from tweet timezones |
| 3 | Neither result set has any countries identified |

Table 5.14: Test Cases: Geographical Heatmap Duo Component

These were implemented by the tests in Table 5.15.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Slashtag query where both result sets have 1 or more countries identified | Geographical heatmaps with tweeting countries highlighted, comparison map generated | Success: geo. heatmap displayed, comparison map generated |
| 2 | Slashtag query where only 1 result set has 1 or more countries identified | Geographical heatmap with tweeting countries highlighted, no comparison map | Sucess: geo. heatmap displayed |
| 3 | Slashtag query where neither result set has any countries identified | Error message within component | Success: component displays error message |

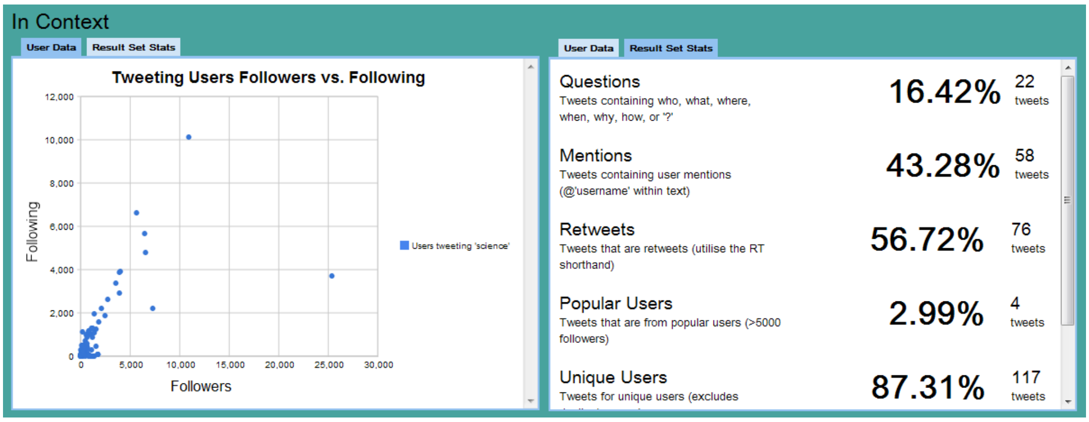
Table 5.15: Tests: Location Duo Component

### Future Development

Whilst the Geographical Heatmap Duo Component comparison map is useful, the single hashtag map provides little functionality over and above the Location Component. With additional development time this component would be integrated into the Location Component, with a toggle to switch between the two types of map.

## 5.7.7 In Context Component

The In Context component, shown in Figure 5.23, categorises tweets in the results set based on several factors: Twitter functions used, popularity of tweeting users and type of content posted. From a development perspective it is used to quickly prototype new Pulse component functionality.



| | | |
|---|---|---|
| Questions<br>Tweets containing who, what, where, when, why, how, or '?' | 16.42% | 22 tweets |
| Mentions<br>Tweets containing user mentions (@'username' within text) | 43.28% | 58 tweets |
| Retweets<br>Tweets that are retweets (utilise the RT shorthand) | 56.72% | 76 tweets |
| Popular Users<br>Tweets that are from popular users (>5000 followers) | 2.99% | 4 tweets |
| Unique Users<br>Tweets for unique users (excludes | 87.31% | 117 tweets |

Figure 5.23: In Context Component

### Purpose

The component also aims to help users identify whether Twitter is used primarily as a social network or a news source for certain topics. Kwak et al [52] found that Twitter displays characteristics unlike any other human social network, particularly low following / follower reciprocity. The In Context component graphs each user's followers and total users followed to provide a simplistic assessment of whether this applies to individual topics as well as across the Twittersphere.

In addition it filters each tweet to establish certain characteristics occurring across a result set. For example - are users asking questions or making statements when using a certain tag? Are most posts retweets or original content? How often are 'popular' users tweeting about a topic?

It was also used as a testbed for developing component prototypes. Creating a new component takes time, so to enable quick feasibility testing of proposed functionality tabs can simply be added to this component showcasing new features. Any data processing or cache requirements can be added as required, and should the features be deemed worthy they can later be moved into a new component.

### Approach

In the tweet categorisation view each tweet's content is scanned to check whether any of its content matches certain filters. These are:

- *Questions*: if the tweet text contains any 'question word' (who, what, where, why, when) or a question mark
- *Mentions*: if the tweet text contains '@<username>'
- *Retweets*: if the tweet text contains 'RT'
- *Popular Users*: if the tweeting user has $>$ 5000 followers
- *Unique Users*: how many tweets in the result set were from unique users

- *Facebook Tweets*: how many tweets originated from Facebook

The results are then displayed as absolute totals and as a percentage of all tweets.

In the follower / following view each tweeting user is plotted on a scatter graph, using the user's followers and users followed counts.

The component utilises a *TabPanel* to enable switching between multiple content panels. Components that feature multiple data views, each needing to be distinct, can use *TabPanel*s to show only one view at a time. The remaining views are listed horizontally as clickable tabs, hidden until required.

### Data Requirements

The component requires the text of each tweet in the result set, as well as the follower / following totals of users responsible for each tweet.

The query cache used by this component stores a comma-separated string of follower / following total pairs, as well as a *TweetSetStats* object. This holds the total

### Component Testing

The In Context Component has no secondary filtering (e.g. geocoding or timezone lookup) so will always show data as long as there are results. Table 5.16 shows the test cases defined for the Tag Volume Component:

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned |
| 2 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 3 | Cached query is selected |

Table 5.16: Test Cases: In Context Component

Table 5.17 shows the tests derived from each test case, and the results of running each test.

### Duo Equivalent

The Duo equivalent of this component adds another tab for a second result set in which to display the tweet characteristic statistics, and plots the follower / following data for both sets as two colours on a scatter chart.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | A hashtag query with results | Scatter chart of follower / following pairs, aggregated tweet characteristics in new tab | Success: scatter chart / characteristics panel displayed |
| 2 | A slashtag query where only 1 tag returns results | Output of test case 1 | Success: scatter chart / characteristics panel displayed |
| 3 | A cached hashtag query | Output of test case 1 | Success: scatter chart / characteristics panel displayed |

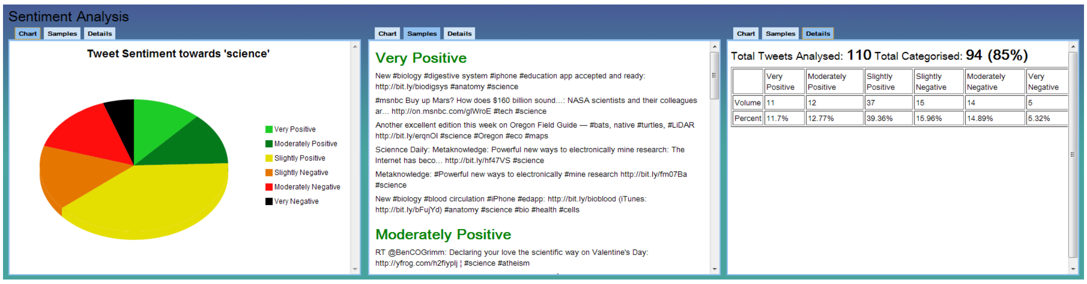
Table 5.17: Tests: In Context Component

### Future Development

At the moment Pulse does not utilise individual user information to any great degree. This component could be used to display a social network connection graph for each tag query, with links between each tweeting users visualising the clusters of connected tag usage.

## 5.7.8 Sentiment Analysis Component

The Sentiment Analysis Component, shown in Figure 5.24, processes tweet text and tries to categorise it into one of six sentiment groups, from ‘very positive’ to ‘very negative’.



Very Positive

New #biology #digestive system #iphone #education app accepted and ready: http://bit.ly/biodigsys #anatomy #science

#msnbc Buy up Mars? How does $160 billion sound...: NASA scientists and their colleagues ar... http://on.msnbc.com/glWroE #tech #science

Another excellent edition this week on Oregon Field Guide — #bats, native #turtles, #LiDAR http://bit.ly/erqnOl #science #Oregon #eco #maps

Sciennce Daily: Metaknowledge: Powerful new ways to electronically mine research: The Internet has beco... http://bit.ly/hf47VS #science

Metaknowledge: #Powerful new ways to electronically #mine research http://bit.ly/fm07Ba #science

New #biology #blood circulation #iPhone #edapp: http://bit.ly/bioblood (iTunes: http://bit.ly/bFujYd) #anatomy #science #bio #health #cells

Moderately Positive

RT @BenCOGrimm: Declaring your love the scientific way on Valentine's Day: http://yfrog.com/h2fiyplj ¦ #science #atheism

Total Tweets Analysed: 110 Total Categorised: 94 (85%)

| | Very Positive | Moderately Positive | Slightly Positive | Slightly Negative | Moderately Negative | Very Negative |
|---|---|---|---|---|---|---|
| Volume | 11 | 12 | 37 | 15 | 14 | 5 |
| Percent | 11.7% | 12.77% | 39.36% | 15.96% | 14.89% | 5.32% |

Figure 5.24: Sentiment Analysis Component

### Motivation

Analysis of tweent sentiment was identified during the existing tools evaluation in Section 2 as a desirable feature for the Pulse user application. Understanding tweet sentiment has a wide variety of applications. These include searching results to inform decision making, discovery of new interests and opinion polling. The Sentiment Analysis component should be able to answer questions related to these topics. For example, a user may be considering purchase of a CD by a band they’ve recently heard of, say ‘#royksopp’. To help make their decision they opt to search Pulse and see what other users are saying about the CD. The Sentiment Analysis Component will categorise each tweet returned as either positive — “The new #royksopp album is great!” — or negative “very disappointed by #royksopp album :(”. The aggregated results will then be displayed to the user, along with some samples from each category, to help the user make a decision.

Another example usage is in opinion polling. Take two political parties, the ‘#blues’ and the ‘#reds’. How does public opinion towards each differ? The Sentiment Analysis Component can evaluate statements featuring each tag and provide a comparison of each, as well as sample tweets for each category.

### Approach

Numerous approaches were attempted before a final process for sentiment analysis was decided upon. The following factors were considered for each approach:

- Performance: individual tweets are analysed for sentiment in real time on retrieval from the Terrier index. As such, the time and processing power required to perform sentiment analysis must be bounded to a reasonable limit.
- Accuracy: the method used to analyse tweets for sentiment must give a reasonable level of accuracy, and also be able to analyse as many tweets as possible.
- Usefulness: the level of analysis must not be as broad as to be meaningless, but should provide users with an accessible and comprehensive overview of sentiment towards a tag on a given day. Furthermore, it should justify results with samples and statistics.

In addition to these factors, the challenges described in categorising tweets by Bifet et al [45] were also considered:

- *Categorisation*: having a human operator categorise each tweet would be a laborious task — with over 6 million tweets each day this becomes virtually impossible.
- *Time*: the volume of tweets returned in each result set means the analysis process must be fast enough so that results are displayed in real time
- *Text Content*: text posted on Twitter may contain irony, sarcasm, or other products of natural language that could lead an analyser to incorrect conclusions. Furthermore the text content is at most 140 characters, meaning one-word analysis misses could change the final sentiment category drastically.

Each of these challenges needed to be tackled by whatever approach was implemented.

The most basic approach attempted was use emoticon detection. Emoticons (‘emotion icons’) express feelings by replicating simple facial expressions using keyboard characters. For example, ‘:-)’ signifies happiness, whilst ‘D:’ signifies severe dismay or anger. The initial sentiment analysis approach checked each tweet for a number of emoticons, and applied a ‘sentiment score’ based on which emoticons were used.

This technique had numerous benefits. Computationally it is extremely simple, and could even be performed by the client application without any server calls. Additionally it is accurate: no actual classification is performed by the application, as the user has already imbued sentiment through the use of an emoticon.

However, there are several issues with this technique. Firstly, the variety of emoticons used by people is huge, and new ones are created all the time. Furthermore certain countries use very particular emoticons which deviate from the standard horizontal style common in the West — for example Japanese emoticons are vertically-orientated and considerably more expressive [60], such as `( ^_^)' expressing happiness. Finally, only a subset of all tweets feature emoticons of any kind. Any that don't would be ignored by the component, resulting in a greatly reduced sample size.

The next approach considered was OpinionFinder [59], a system of utilities and tools which can be be used to identify opinions within text. However, organising the pipeline of components required for analysis, to perform in real-time, was a laborious task. At the time of writing a complete Java re-write of OpinionFinder is being undertaken to encapsulate the setup and configuration process as a single application. In the mean time the system as-is was deemed overly-complex to use on a query-by-query basis.

The final approach combines SentiWordNet [43], the Stanford Part-Of-Speech (POS) tagger [58] and emoticon filters.

SentiWordNet is a lexical resource containing scored sets of English words. Scores are applied in three categories: positivity, negativity, and objectivity. The word sets are each a group of synonyms, defined by WordNet [54] — a "lexical resource of English nouns, verbs, adjectives, and adverbs organized into sets of synonyms, each representing a lexicalized concept".

The SentiWordNet wordset list was used to create a sentiment dictionary. Each dictionary key is a single word annotated with it's part of speech (for example, 'house#n' represents the word 'house', which is of type 'n' – a noun). Keys are associated with a final sentiment value calculated from each of the three individual SentiWordNet scores.

The Stanford POS tagger identifies the part of speech of each token within tweet text. This is appended to the word, after which the final 'word#pos' token is looked up in the sentiment dictionary. Should a match be found the base tweet score of 0 is adjusted appropriately — adding up to 1 for positive words, and subtracting up to 1 for negative words. The final score is derived by dividing the intermin score by the number of words analysed.

The last step involves applying score modifiers based on emoticons — tweet score is increased by 0.5 if any positive emoticons are found, and reduced by 0.5 if any negative emoticons are found.

The final score is then banded into one of six categories: 'very positive' (score $>$ 0.5), 'moderately positive' (0.25 $<$ score $<=$ 0.5), 'slightly positive' (0 $<$ score $<=$ 0.25), 'slightly negative' (-0.25 $<$ score $<$ 0), 'moderately negative' (-0.5 $<$ score $<=$ -0.25) or 'very negative' (score $<=$ 0.5). This sum of tweets within each band is returned to the client, along with a small sample of tweets within each category.

This process is performed in real-time, however the POS tagger takes 50ms per tweet to identify parts of speech. For this reason a less accurate option was introduced. Users can opt to bypass the POS stage, and instead each word is looked up with every possible POS type appended to it (noun, verb, adjective and adverb). If there is more than 1 hit in the dictionary an average score is taken — whilst less accurate it gives users the option to analyse sentiment faster.

### Data Requirements

The Sentiment Analysis component uses the text of each tweet for analysis.

The query cache used by this component stores the total number of tweets in each sentiment category, as well as a small sample of tweets from each category.

### Data Preparation & Visualisation

Sentiment analysis requires several external files and libraries to operate. To achieve this a service is invoked by the component, *SentimentAnalysisServiceImpl*, which accepts a *SentimentAnalysisServiceRequest* containing the tweets to be analysed and the sentiment analysis options. These are analysed as described in Section 5.7.8, and the results are returned to the client callback function as a *SentimentAnalysisServiceResponse*. The response contains the total number of tweets per category and a sample of tweets from each (if any).

Once analysis is complete the category totals are used to create a pie chart of overall tweet sentiment, as well as an explicit data table. Both of these elements are displayed within a *TabPanel*, along with the sample tweet category table.

### Options

The Sentiment Analysis Component has two options: *Use POS Tagger* and *Analysis %*. Selecting *Use POS Tagger* will enable the POS tagger during sentiment analysis, rather than just checking all occurences of each word in the dictionary. For large results sets this may slow down the application.

The *Analysis %* option allows users to specify what percentage of the result set should be analysed. This can be reduced for queries with very large result sets. *Math.random* is used to ensure the sampled results are selected randomly from all available results.

### Component Testing

Table 5.18 shows the test cases defined for the Sentiment Analysis Component:

There are some specific edge cases applicable to this component. Even if the general result set returned to Pulse has one or more results, the component options may result in 0 results being analysed. In this case, the component cache still needs to be created to avoid errors when regenerating components from cache.

Table 5.19 shows the tests derived from each test case, and the results of running each test.

| Test Case ID | Description |
|---|---|
| 1 | Hashtag query results are returned, sentiment analysis % = 100 |
| 2 | Hashtag query results are returned, sentiment analysis % = 0 |
| 3 | Hashtag query results are returned, sentiment analysis % $>$ 0 and $<$ 100 |
| 4 | Slashtag query is converted to hashtag query due to 1 query tag having no results |
| 5 | Hashtag query results are returned, POS tagger is in use |
| 6 | Cached query is selected, with sentiment analysis results |
| 7 | Cached query is selected, with no sentiment analysis results |

Table 5.18: Test Cases: Sentiment Analysis Component

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Hashtag query with results, sentiment analysis % option set to 100 | A tabbed component displaying a pie chart, samples and statistics. Component cache is added to query cache | Success: component displays each tab with correct data |
| 2 | Hashtag query with results, sentiment analysis % option set to 0 | A message is displayed inside the component, all other components display. Component cache is created but empty | Success: a ”No results found” error message displayed inside the component |
| 3 | Hashtag query with results, sentiment analysis % option set between 1 and 99 | Output from one of test cases 1 - 2 | Success: component displays relevant tags or error message, depending on % analysed |
| 4 | Hashtag query with results, ’Use POS tagger’ option selected | Slower retrieval of results, output from test case 1 / 2 | Success: POS tagger option read by component and passed to servlet |
| 5 | Slashtag query where only 1 tag returns results | Output from one of test cases 1,2,3 or 4 | Success: component displays as if query were a hashtag query |
| 6 | Cached hashtag query with sentiment analysis results | Quick display of output from one of test cases 1 - 5 | Success: data loaded from cache and displayed within component |
| 7 | Cached hashtag query with no sentiment analysis results | A message is displayed inside the component, all other components display | Success: message displayed |

Table 5.19: Tests: Sentiment Analysis Component

### Duo Equivalent

The Duo version of this component provides analysis of both result sets, displaying the results within 2 tab panels. These can be switched using a drop-down list in the title bar of the component.

There are additional test cases to be considered in the Duo component - these are listed in Table 5.20. Consider a slashtag search where one result set is large and the other comparatively small. If the sentiment analysis % is set to a low figure, then it may be that only the larger set will actually have any sentiment analysis results. In this case, the component must show the results for that set, and handle zero results for the other set gracefully.

| Test ID | Input | Expected Output | Result |
|---|---|---|---|
| 1 | Slashtag query returns 2 result sets, one large, one small, and sentiment analysis % has low value | If only 1 result set is analysed, display within single tab panel | Success: component displays only 1 tab panel |

Table 5.20: Tests: Sentiment Analysis Duo Component

### Future Development

Currently the component will try and analyse any tweet returned from each query, which includes non-English tweets. There were several ways to approach this problem, each with advantages and disadvantages.

The first approach is by far the simplest: ignoring tweets where the `lang` atrribute is not 'en'. This key included in every tweet entity as an indicator of the user's native language. What it does not indicate is the language of the tweet text. Table 5.21 shows the accuracy level of this information for a small set of 100 tweets obtained after searching for the tag '#news'.

| Tweets Classified | Language Match | Language Mismatch | Accuracy |
|---|---|---|---|
| 100 | 76 | 24 | 76% |

Table 5.21: Language Classification - 'lang' attribute

As shown the level of accuracy is fairly poor — 24% of tweets sampled were misidentified as English.

The second approach tried was to detect if any non-ASCII characters had been used in the tweet, limiting the allowable alphabet to Latin characters only. The same sample of 100 tweets was checked to see how accurate this method was (shown in Table 5.22)

| Tweets Classified | English, Non-ASCII | English, ASCII | Non-English, ASCII | Non-English, Non-ASCII |
|---|---|---|---|---|
| 100 | 19 | 58 | 7 | 16 |

Table 5.22: Language Classification - ASCII / Non-ASCII English

25% of all English tweets contained non-ASCII characters, whilst 31% of non-English tweets contained only ASCII valid characters. This method of filtering would remove a large proportion of English content, as well as continued analysis of much non-English content. Based on these results this method was quickly discounted.

Another approach tried was to use a text classification library. The Java Text Categorizing Library (JTCL) REF! was used to try and 'guess' the language of each tweet. JTCL is based on the algorithm proposed by Cavnar et al [46], whereby an N-gram frequency profile is computed for each tweet, which is compared to known profiles for various languages using a distance measure.

Whilst very accurate for documents as small as 300 bytes, tweets pose a challenge in being a maximum of 140 characters — even shorter after removal of links, user mentions and retweet markers. Table 5.23 shows the results from using the JCTL library.

| Tweets Classified | Language Match | Language Mismatch | Accuracy |
|---|---|---|---|
| 100 | 84 | 16 | 84% |

Table 5.23: Language Classification - JTCL

The JCTL generally managed to identify language correctly, with the exception of tweets which featured several hashtags. Though all punctuation is removed hashtags are still textually strange, often using abbreviations, slang or multiple words without spaces. This was the case in almost all of the mismatch scenarios. As 14% is still a large mismatch percentage JCTL categorisation was not implemented in the final system.

The Google Translate API was also tested to see how effective it was at detecting language. However, initial benchmarks showed the round trip time for each request was between 50 and 200ms — far too long for real-time usage with a non-trivial number of results.

As it stands foreign language tweets are still analysed and counted towards the overall total. Further work is required to find a more effective method for eliminating non-English tweets without removing English text in the process. A first step towards this would be an option to pick one of the three options presented, however due to time constraints this was not implemented.

### 5.7.9 Trends Component

The Trends Component contains a multifunction pivot chart, which allows for analysis of daily trend data in a number of formats, comparing multiple variables. An annotated version of the component is shown in Figure 5.25.

#### Diagram Key

The following key applies to Figure 5.25.

1. *Change Y-Axis Variable*: Allows the Y-axis variable to be changed. Available options are 'Countries', 'Occurrences' and 'Unique Users'

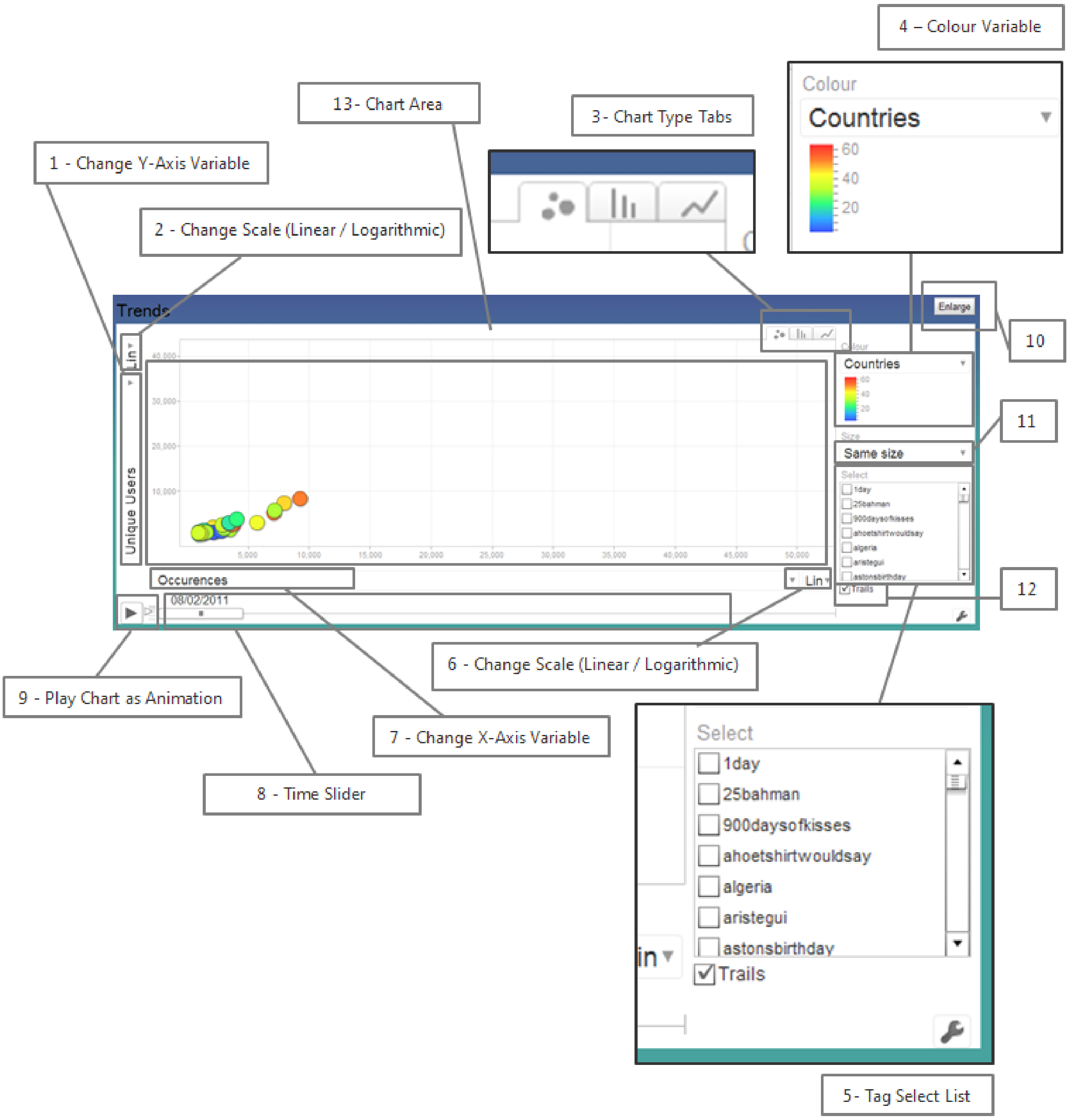


Figure 5.25: Trends Component

2. *Change Scale (Linear / Logarithmic)*: Change the scale type for the Y-axis. Can be either linear or logarithmic
3. *Chart Type Tabs*: Change the type of graph displayed. The 3 main chart types are discussed in detail later in this section 'Scatter', 'Bar' or 'Line'.
4. *Colour Variable*: Colouring can be used represent a third variable, along with the X and Y axises. Available options are 'Countries', 'Unique Users', 'Occurrences', 'Same Colour' (all points are same colour), or 'Unique Colour' (all points have different colours). The gradient below indicates what each colour represents
5. *Tag Select List*: Choose specify tags to show on each chart. Click 'Deselect All' to reset this list
6. *Change Scale (Linear / Logarithmic)*: Change the scale type for the X-axis. Can be either linear or logarithmic
7. *Change X-Axis Variable*: Allows the X-axis variable to be changed. Available options are 'Countries', 'Occurrences' and 'Unique Users', 'Time' and 'Order: Alphabetical'
8. *Time Slider*: Scroll to watch graph points change over time. Will only be visible if X-axis is set to anything other than 'Time'
9. *Play Chart As Animation*: Animate the graph points over time. Use the slider to the left of the play button to change playback speed
10. *Enlarge Trends Component*: The Trends Component contains much detail. In order to better use it, click this button for the component to occupy all space in the component area
11. *Size Variable*: The size of each node can be used to represent a fourth variable. Available options are 'Same Size', 'Countries', 'Unique Users' and 'Occurrences'
12. *Use Trails*: Show trails as each node is animated. Use in conjunction with the time slider to create ad-hoc line graphs
13. *Chart Area*: Area used to display results

### Purpose

The Trends Component provides multidimensional views of hashtag usage over a temporal range. Each hashtag can be pivoted on number of occurences, number of unique users tweeting the tag, number of unique countries in which the tag is being used, and by usage date. Colour and shape can also be used to represent any of these variables. Tags can be viewed as an animated scatter chart, a bar chart, or a line graph.

The dataset returned for each query can be filtered using the temporal query options in the application Navigation Bar. Hashtags can be whitelisted or blacklisted, and limits can be placed on the number of occurences a tag must have for it to be appear within the component.

Users can track topics over time, observe tag patterns of usage (does the tag spike in occurences every day, or week, or month?), identify changes in tag usage, and understand tag usage context (i.e. is the tag localised to one country, or used worldwide? Is the tag repeatedly used by a small number of users, or widely used?)

### Motivation

The Trends Component enables users to identify particular tags which were ‘trending’ on Twitter during a period of time. A Twitter ‘trend’ is a word or phrase which is being aggressively used by a large volume of users, usually in response to some event (e.g ‘#superbowl’), but also occasionally through concerted effort on the part of Twitter users (e.g. ‘#justsaying’).

Trendistic (discussed in Section 2.2) provides a basic implementation of the Trends Component, graphing tag usage over time. It does not provide any comparison functionality, nor does it feature variables other than the number of tag occurences.

Whilst number of occurences is certainly an important metric for judging tag popularity there are other factors that need to be considered. Is the tag trending in a single region, or worldwide? Is the tag being ‘spammed’ by a small group of users, or has it grown popular organically, used by a large group of unique users?

Unlike Trendistic the Trends Component also provides tag comparison. Each daily top tag can be manipulated independently of other tags, or in combination with any number of tags. When comparing several tags the Trends Component will automatically adjust to an appropriate zoom level so that the tags can be better compared and visualised over time. An excellent example of this would be to compare usage of tags such as ‘#egypt’, ‘#bahrain’, ‘#tunisia’, ‘#libya’ and ‘#yemen’ over a temporal range of two months. The Trends Component would show spikes in user, overlaps between each, when the tag first and last appeared and how widespread tag usage was. None of the tools evaluated provide the flexibility and customisation available with this component.

### Data Requirements

This component uses the top hashtags hashmap, returned by the server in response to temporal range queries. The tag hashmap contains string keys representing the Unix timestamp for each day returned, and a comma separated string value with each tag and its associated variables. The structure of each value is shown below:

```
#tag,<numOccurences>,<numUniqueUsers>,<numUniqueCountries>
```

### Data Preparation & Visualisation

The GWT MotionChart widget [18] powering the Trends Component visualisation must be supplied with data in a specific manner to enable pivoting functionality. This is shown in pseudocode below:

```
dataTable = new DataTable // GWT visualisation structure
dayTagLimit = request.getTagLimit // max. top tags to show per day
for each column required //each column is one variable (countries,
    dataTable.addColumn  //unique users, occurences etc.)
sort hashmap keys // each key is a Unix timestamp string
for each day key
    dayString = tagHashmap.get(key)
```

```
dayDetails = dayString.split(',') // array length 5, each entry
for each tag within dayLimit      // a single var.
    if tag in wList and not in bList and within occurence limit
        dataTable.addRow
        for each variable in dayDetails
            dataTable.setValue(current row, dayDetail[value])
```

Once constructed the data table is passed to the Visualisations API, which generates the visualisation. This visualisation is one of few implemented in Flash, so a Flash browser plugin must be installed on the client-side for this component to display correctly.

For easier viewing the component can also be enlarged by clicking the 'Enlarge' button in the component title bar.

#### Options

Multiple options can be set which customise the data displayed by the Trends Component. These include:

- *Daily Top Tag Limit*: the number of top hashtags per day to show in the component
- *Tag Minimum / Maximum Occurences*: upper and lower boundaries for the number of occurences a tag must have for it to show in the Trends Component
- *Trend Tag Whitelist / Blacklist*: a list of tags to either be explicitly included (whitelist) or excluded (blacklist) from the trends components. This can be used to filter the top hashtags returned into a more concise set

#### Data Views

Data within this component can be viewed in 3 different ways: as a scatter graph, as a bar chart, or as a line graph. These views are explained in detail in Appendix A.

### 5.7.10 Place / Device Component

The Place / Device Component, shown in Figure 5.26, displays a graph of devices used to access Twitter over a specific time range, as well as a map for each day in the time range highlighting country-level tweet volume.

#### Purpose

Surprisingly Twitter does not regularly provide detailed service usage statistics. This component was designed to address the lack of data by visualising specific information about Twitter usage, by country and by device.

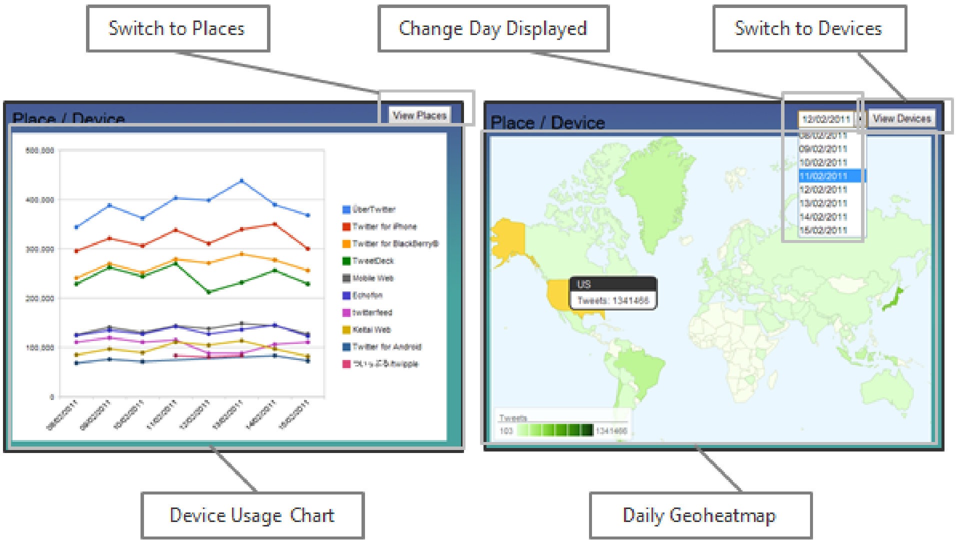


Figure 5.26: Place / Device Component

### Data Requirements

This component uses the country occurences hashmap and top devices hashmaps, returned by the server in response to temporal range queries. The country occurences hashmap contains string keys representing the Unix timestamp for each day returned, and a comma separated string value with country / occurrence pairs.

```
<country>,<numOccurences>,<country>,<numOccurences>...
```

The top sources hashmap is structured in the same way, with each hashmap value a comma-separated list of devices.

```
<device>,<numUses>,<device>,<numUses>...
```

### Data Preparation & Visualisation

The component contains two views: a line graph plotting device usage over time, and a series of maps showing Twitter usage by country. The two views can be switched by clicking the 'View' button in the component title bar.

The devices view consists of a line graph, with each device in the top devices hashmap plotted against the number of times it was used to post a tweet. This is done for every day in the temporal range, showing variance in device usage over time. The line graph is drawn using the Visualisation API.

The maps view shows a version of the Geographical Heatmap Component, but instead of plotting tag usage by country, this map shows overall daily tweet usage. The map can be changed to show a different day by picking a date from the title bar date list.

#### Options

‘Web’ is by far the most popular method chosen by users when they update Twitter. By selecting ‘Exclude web from devices’ in the Options Panel of the Navigation Bar ‘Wen’ will not feature in temporal range query results. This makes differentiating between devices in the Device graph much easier.

The number of devices shown can also be adjusted. The default value is set to 10, and though this can be increased it quickly clutters the devices graph. Developing a more robust version of this view is listed as a future development.

#### Future Work

Whilst the visualisations provided by this component are useful as they are, there is scope for integrating the two data sets so that device usage per country is also analysed. For example, which countries tend to use unofficial applications instead of the official Twitter applications? How prevalent is updating via iPhone in Japan compared to updates via Blackberry? These types of questions are not currently answered by the data shown in the component, but could be with additional development time.

The devices view also needs additional work to better handle large numbers of different devices. The graph is difficult to read if more than 15 devices are shown, and whilst options allow the end user to specify a limit they are happy with the component should be able to hide and show devices on demand.

### 5.7.11 Sentiment / Volume Component

The Sentiment / Volume Component, shown in Figure 5.27, displays daily Twitter sentiment and tweet volume.

#### Purpose

The Sentiment / Volume Component has 2 functions. The first is allows users to observe changes in sentiment across a temporal range. Whilst topic-based sentiment analysis gives a focused sentiment breakdown for a small result set, aggregated sentiment volumes allow users to identify events which are so pervasive that they have an overall effect on Twitter. An early example of this was noticed on Valentine’s Day, where there was a noticeable increase in positively-ranked tweets.

The second function of this component is an early attempt at judging total Twitter usage volume. This view, though functional, requires further work.

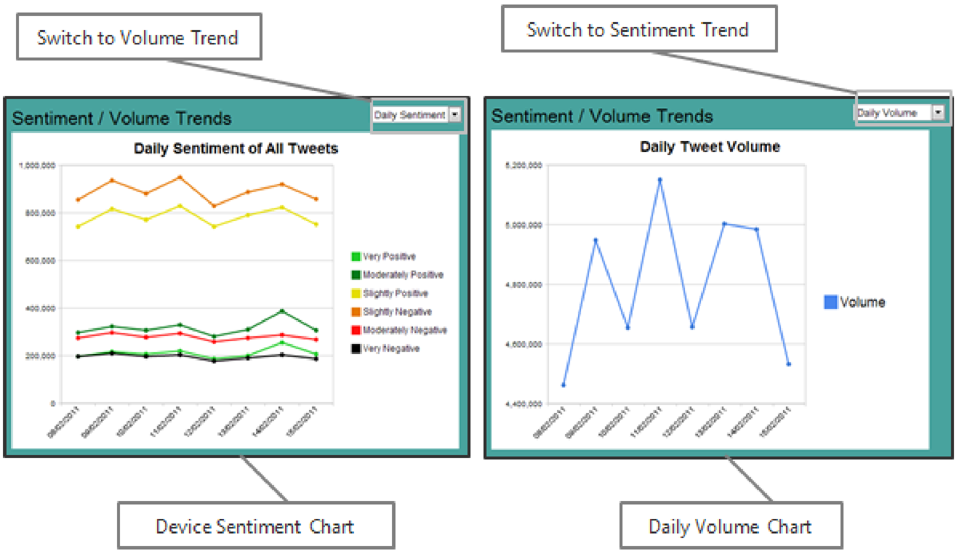


Figure 5.27: Sentiment / Volume Component

### Data Requirements

This component uses the daily sentiment and daily volume string, returned by the server in response to temporal range queries. The sentiment hashmap contains string keys representing the Unix timestamp for each day returned, and a comma separated string value structured as follows:

```
<category>,<numTweets>,<category>,<numTweets>...
```

The volume string is a comma-separated list of day / volume pairs.

```
<timestamp>,<volume>,<timestamp>,<volume>...
```

### Data Preparation & Visualisation

Each of the views contains a single line graph mapping either sentiment or volume over time. The views can be switched by clicking the 'View' button in the component title bar.

The sentiment view features of a line graph, with each category of sentiment (from 'very positive' to 'very negative') plotted against each day. The line graph is constructed using the Visualisation API.

The volume view also features a line graph, plotting volume against time for the temporal range specified.

*N.B.*: As discussed earlier Pulse stores only 5% of all public tweets. As such the volumes displayed are not representative of *total* Twitter usage.

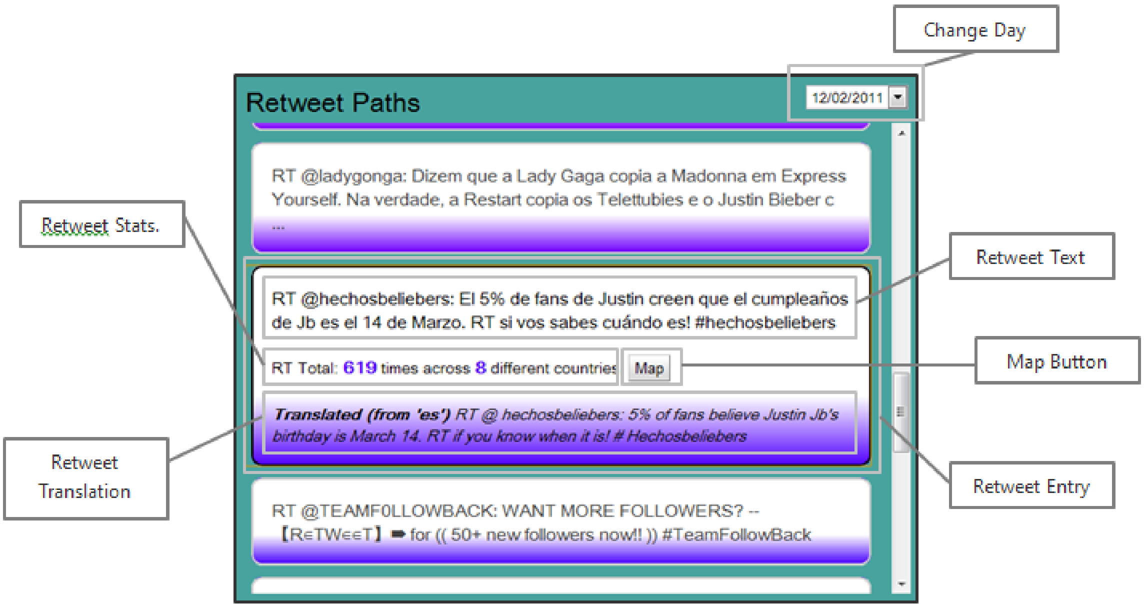
### Future Work

This component was the last to be implemented, and still needs further work to fully take advantage of the data available. The aggregate sentiment values shown on the sentiment line graph do allow for observation of unexpected changes over time.

The volume view should explain the caveats associated with only storing a subset of the data, and also give estimates as to total tweet usage. These are small changes that would significantly increase the usefulness of this component.

## 5.7.12 Retweet Paths Component

The Retweet Paths Component, shown in Figure 5.28, focuses on analysing retweets in terms of volume and propagation.



RT @ladygonga: Dizem que a Lady Gaga copia a Madonna em Express Yourself. Na verdade, a Restart copia os Telettubies e o Justin Bieber c ...

RT @hechosbeliebers: El 5% de fans de Justin creen que el cumpleaños de Jb es el 14 de Marzo. RT si vos sabes cuándo es! #hechosbeliebers

RT Total: 619 times across 8 different countries

***Translated (from 'es')*** *RT @ hechosbeliebers: 5% of fans believe Justin Jb's birthday is March 14. RT if you know when it is! # Hechosbeliebers*

RT @TEAMF0LLOWBACK: WANT MORE FOLLOWERS? -- 【R∈TW∈∈T】 ➡ for (( 50+ new followers now!! )) #TeamFollowBack

Figure 5.28: Retweet Paths Component

### Motivation

Retweeting is particularly interesting facet of Twitter. When a users retweets a tweet they are showing a level of agreement with the tweet content, and an inclination to share that content with their own follower group.

The mapping element of this component was motivated by an study conducted by Kwak et al of whether Twitter is a social network or news source. They found that no matter how poorly connected a user is any retweet is “likely to reach a certain number of audience, once the users tweet starts spreading via retweet” [52]. This process is visualised by plotting retweet occurences on a map.

Whilst ‘top retweet’ lists are commonplace on the internet (for example, The Retweetist News [3]) no retweet mapping function was seen amongst existing systems.

### Data Requirements

This component uses the top retweets hashmap, returned by the server in response to temporal range queries. The retweet hashmap contains string keys representing the Unix timestamp for each day returned. Each value contains a structured string with retweet texts, as well as list of users, user locations and publication dates for each retweet. This additional information is used to map retweet propagation over time.

### Data Preparation & Visualisation

This component features two views: an ordered list of retweets by number of occurences, and a map displaying location ‘waypoints’ indicating how the retweet propagated geographically.

The initial view lists the top retweets by number of occurences for the query date selected. A list box is also generated in the component title bar to switch between different dates. Each retweet is contained within a box which when clicked shows additional statistics. The gradient shown in each box is a visual indication of the retweet popularity, with solid red representing the most popular retweets, to solid blue representing the least popular.

The gradient is applied by generating a hexadecimal colour code, calculated as follows:

```
colours = 512 // number of colours available
difference = maxOccurences - minOccurences
colourIncrease = colours/difference
aboveMin = occurences - minOccurences
colourVal = aboveMin * colourIncrease
if colourVal < 256
    colourCode = colourVal.toHexString + '00FF'
    if colourCode.length < 6 // edge case
        colourCode = '0'+colourCode
else
    blueValue = 256+(256-colourVal).toHexString
    if blueValue.length < 2 // edge case
        blueValue = '0'+blueValue
    colourCode = 'FF00'+blueValue
return colourCode
```

The basic concept was to calculate a ‘colour increase value’ which is multiplied by the number of occurences over the minimum each retweet has. From this a hexadecimal colour code is created representative of a retweet’s position in the ordered list.

Clicking a box reveals further information about that retweet, including total occurences and number of unique countries. There is also a button to switch to the component mapping view (see ‘Mapping’ below).

Should Pulse detect the retweet is non-English it will attempt to translate it using the Google Language API [19]. If successful a translation is shown in italics below the original text.

### Mapping

The location information associated with each retweet can be used to map retweet propagation. Pulse implements this using the Maps API, geocoding retweet location information and plotting markers on a map (illustrated by Figure 5.29)

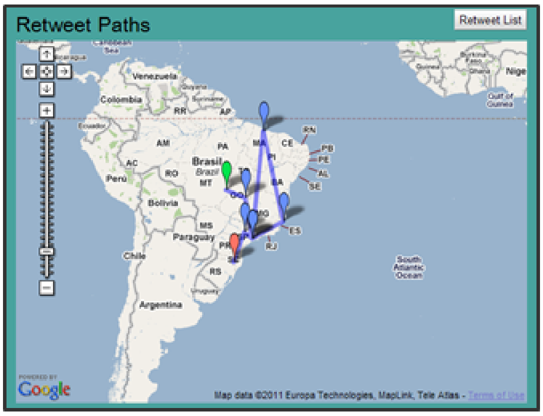


Figure 5.29: Retweet Paths Component Map

The green map marker signifies the first occurrence, with blue overlay lines showing connections between each subsequent retweet. The last retweet is indicated using a red marker.

Each retweet path mapping is limited by the fact that Pulse only collects 5% of public tweet data. The tweet will almost certainly have been retweeted in many more locations, the exact number being included as an attribute in the JSON tweet entity. However it serves its purpose well as a rough illustration of waypoints throughout the retweet’s propagation path.

### Options

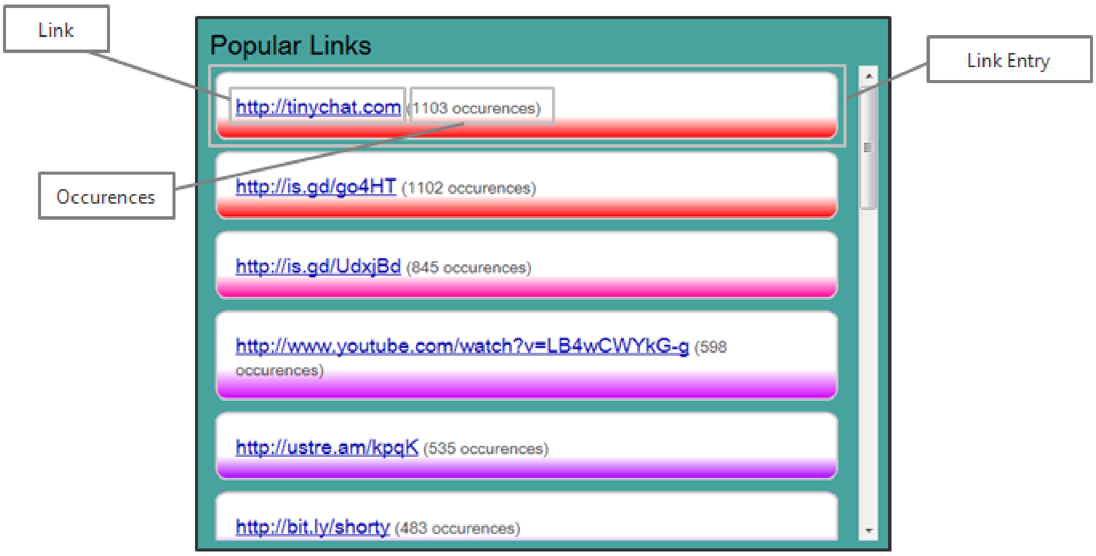
The number of retweets shown can be set from the Options Panel in the Navigation Bar. The default value is 20.

### Future Work

The main focus for improvement with this component concerns the propagation mapping functionality. Ideally a user would click a balloon and see context information surrounding that particular retweet — the user, the time of day and so on. With this information it would be possible to identify users responsible for particular retweets.

### 5.7.13 Popular Links Component

The Popular Links Component, shown in Figure 5.30, highlights URLs most featured in tweets on a specific day.



http://tinychat.com (1103 occurences)
http://is.gd/go4HT (1102 occurences)
http://is.gd/UdxjBd (845 occurences)
http://www.youtube.com/watch?v=LB4wCWYkG-g (598 occurences)
http://ustre.am/kpqK (535 occurences)
http://bit.ly/shorty (483 occurences)

Figure 5.30: Popular Links Component

#### Motivation

Links are the main method through which users on Twitter can share information outwith the 140 characters limit and offer the potential to connect Twitter trends to the wider social web. In conjunction with retweeting Twitter has provided an easy way for users to rapidly propagate content.

Many sites exist which are entirely devoted to Twitter link tracking. One of the most popular, TweetMeme [37], specialises in comprehensively reporting popular links by category, date and other filters. The Popular Links Component was developed late in the project timeline and so requires further work to match existing tools. Suggested ways of doing so are explained in the ‘Future Work’ section below.

As it stands the Popular Links Component gives a very high-level overview of link usage on Twitter, ordered by number of occurences.

### Data Requirements

This component uses the top links hashmap, returned by the server in response to temporal range queries. The links hashmap contains string keys representing the Unix timestamp for each day returned, and a comma separated string value containing each link and the number of times it occurs (shown below).

```
<link>,<numOccurences>,<link>,<numOccurences>...
```

### Data Preparation & Visualisation

This construction of this component is identical to the Retweet Paths Component, explained in Section 5.7.12.

### Options

The number of links shown can be set from the Options Panel in the Navigation Bar. The default value is 20.

### Future Work

The current implementation of this component is fairly basic in only providing an ordered list of links. It would be interesting to understand what motivates users to tweet links, and how the content of each link correlates with trending topics.

Links posted on Twitter could be used as a set of seeds for a web crawler, providing a data source separate but related to Twitter for topic comparison. A link index of pages crawled and standard tweet indices could be queried simultaneously, with tools provided to analyse whether any parallels can be drawn between content on Twitter and content on the wider web.

## 5.8 Conclusion

The Pulse client application was built to provide a powerful, flexible and usable interface through which users could easily perform queries in a number of ways. It provides both topic and temporal range analysis of Twitter data through several components, each tailored to provide a particular view of results.

The application can also be extended to incorporate new functionality, and is highly configurable through options available to the user. Data is also cached by the client for quick retrieval of past queries.

The next chapter discusses how data is managed and retrieved from the Pulse data store.

# Chapter 6

# Pulse User Application: Server-Side

Division of work between client and server in two-tier systems such as Pulse allows for logical separation of functionality. Traditionally with web applications presentation is handled by a browser (using technologies such as HTML and CSS), whilst data manipulation is handled by a server (database querying, dynamic page generation and use of scripting languages such as PHP). Pulse follows this pattern by defining *services* with which the client can communicate.

This chapter describes the services required by Pulse, conceptually and in implementation terms. These services are then tested and benchmarked against a specific set of requirements.

## 6.1 Services Overview

Services provide rigidly defined functionality to the client application, communicating via GWT RPC. Each service on the server is an implementation of a service interface specified by the application framework (see Section 5.5), and can be invoked by calling the defined service method through the client-side RPC service proxy (see Section 5.6.3).

Importantly each service is distinct from all other services. Though running on the same server they operate completely independently of each other. As such, there is no overall architecture unifying services, only the RPC mechanism unifying each client service proxy and its server implementation.

Services are also persistent. Some server-side processing systems, such as CGI [57], do not maintain state between operations. However, once invoked by the client Pulse services maintain state between future calls. This is a highly beneficial attribute, allowing for implementation of features such as data caching for faster retrieval.

In terms of the wider application architecture services as implemented by GWT are ultimately Java servlets. A servlet is simply a Java class which extends the capability of a server by means of a request-response programming model [41].

Servlets are typically extended from the standard Java HTTP servlet class, *HttpServlet* [23], however GWT includes an extended version of the standard class, *RemoteServiceServlet* class [31], which handles client-server data serialisation and ensures invocation of the correct service method [22] on receiving a request.

## 6.2 Purpose

Services are needed by the client application for two main reasons. The first of these is to act as a proxy between the data store and the application, handling requests, loading the correct data files, searching tweet indices, parsing query results and constructing appropriate responses.

The complexity involved in managing these tasks is abstracted by defining a simple request / response model. The client can make a request to a service for a particular type of data without having any knowledge of the process undertaken to actually generate or obtain that data. It only needs to know how to parse the response it receives. The service itself is built entirely in Java, making the organisation of such a process considerably easier than if implemented solely by the client.

The second purpose of services is to support client-side components in performing data analysis. Whilst elementary manipulation can be conducted by the browser, code-intensive tasks are naturally suited to server execution. For example, components may choose to interact with a service when they need to:

- Utilise an external library or API that is unavailable within the browser
- Perform large-scale manipulation which would take unreasonably long in the browser
- Access files on the web server

In these cases the component only needs to construct a request and define a callback function to handle the response, with the service handling the implementation detail in a language suited to the task.

## 6.3 Suitability of Services

Whilst services can be extremely useful in assisting components, certain factors must be considered before opting to implement functionality on the server rather than client-side.

- *Data Transfer*: the amount of data being transferred between the client and server should be minimised as much as possible to maintain low network latency. If a large amount of data needs to be transferred then the time taken to compute on the client side should be compared to the round-trip service request time to see which is preferable.
- *Task Type*: the task should be assessed in terms of how naturally it fits the client or server. For example, presentation-based work naturally fits the capabilities of web browsers, whereas intensive data handling will often be more suited to a seperate server application.

- *Effect on Responsiveness*: many web browsers (such as Mozilla Firefox [30]) feature an execution time limit for JavaScript code which, when reached, results in an error message being displayed to the user. If this happens users may refresh the page (ending any outstanding requests) or simply navigate away - both severe anti-goals of the system. Using services avoids this, allowing the UI and other work to perform asynchronously.

The client application requires the use of services at two points during each query: to retrieve the initial result set from the Pulse data store, and during the component update cycle. Most components manage processing on the client, however the Sentiment Analysis component does require a supporting service (to use an external Java library).

## 6.4 Tweet Fetch Service

The Tweet Fetch service provides the client application with the initial results needed to generate components for each query.

### 6.4.1 Requirements

The client application must have moderated access to the Pulse data store so that it can generate components in response to user queries. However, the structures used by the data store require Java libraries to open, special manipulation to use and resources to store in memory. These constraints mean that direct client interaction with the data store is near impossible. The Tweet Fetch service should mask this complexity, providing structured responses in reply to client requests.

The Tweet Fetch service must also have a short time to respond. Nielsen explains that after 10 seconds users being to lose focus on the task at hand if no response or feedback is given [55]. Queries are issued in real-time by users, so the delay between clicking 'Search' and seeing results should be as small as possible. This is the main performance factor on which the service will be tested.

### 6.4.2 Architecture

The Tweet Fetch service architecture is illustrated in Figure 6.1.

Requests from the client are initially received by the *TwitterQueryServiceImpl* class. This is an implementation of the service interfaces defined by the application framework, and also acts as the service entry point. Requests are then delegated to the appropriate query manager: TagQueryManager for hashtag-based query requests, and TemporalQueryManager for temporal range query requests.

Each manager is responsible for the end-to-end processing of particular query types. The TagQueryManager operates a pipeline retrieving tweets based on query tags, whereas the TemporalQueryManager retrieves daily statistics based on a query date.

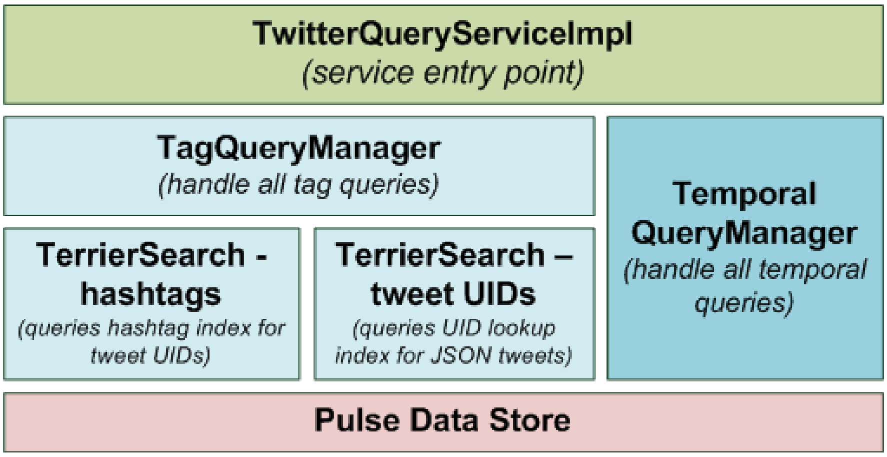


Figure 6.1: Tweet Fetch Service Architecture

The TagQueryManager needs to interact with Terrier indices to get tweet data, so a secondary management class, *TerrierSearch*, was created to control Terrier querying and index manipulation..

The lowest architectural level is the Pulse data store. All tweet data is stored here after processing by the application described in Chapter 4.

Once all processing is complete managers pass a finalised response to TwitterQueryServiceImpl for transmission back to the client.

Each of the stages, plus the data sent interchanged between each stage, is described in Section 6.4.3.

### 6.4.3 Process

Figure 6.2 gives a high-level overview of the process followed once a request has been received.

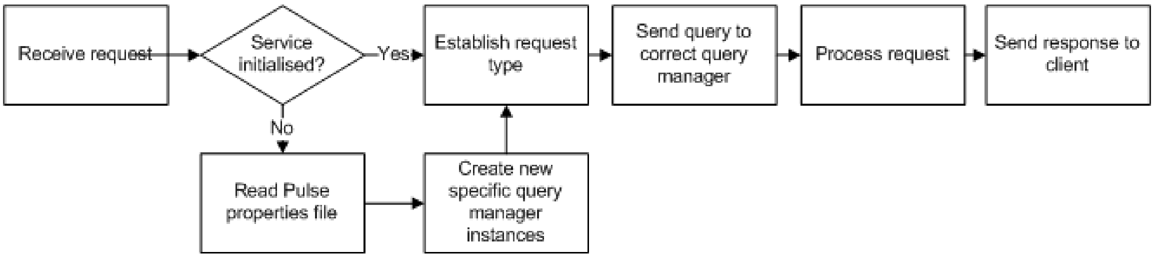

Receive request
Service initialised?
Yes
No
Establish request type
Send query to correct query manager
Process request
Send response to client
Read Pulse properties file
Create new specific query manager instances

Figure 6.2: Tweet Fetch Service Process

#### Initialisation & Request Delegation

On first invocation the Pulse properties file is read to locate the system data directory. Once complete, new instances of the *TagQueryManager* and *TemporalQueryManager* classes are created.

The TwitterQueryServiceImpl class then informs each manager of the data directory location. As the Tweet Fetch service is persistent this process happens only once.

Each request sent to the service has an explicit request type (detailed in Section 5.6.4). TwitterQueryServiceImpl uses this to delegate responsibility for the query to the correct manager class.

### Tag-Based Queries

The TagQueryManager handles interactions with the file system, selection, loading and querying of the correct Terrier indices, result parsing, and construction of the correct response type. This process is illustrated by Figure 6.3.

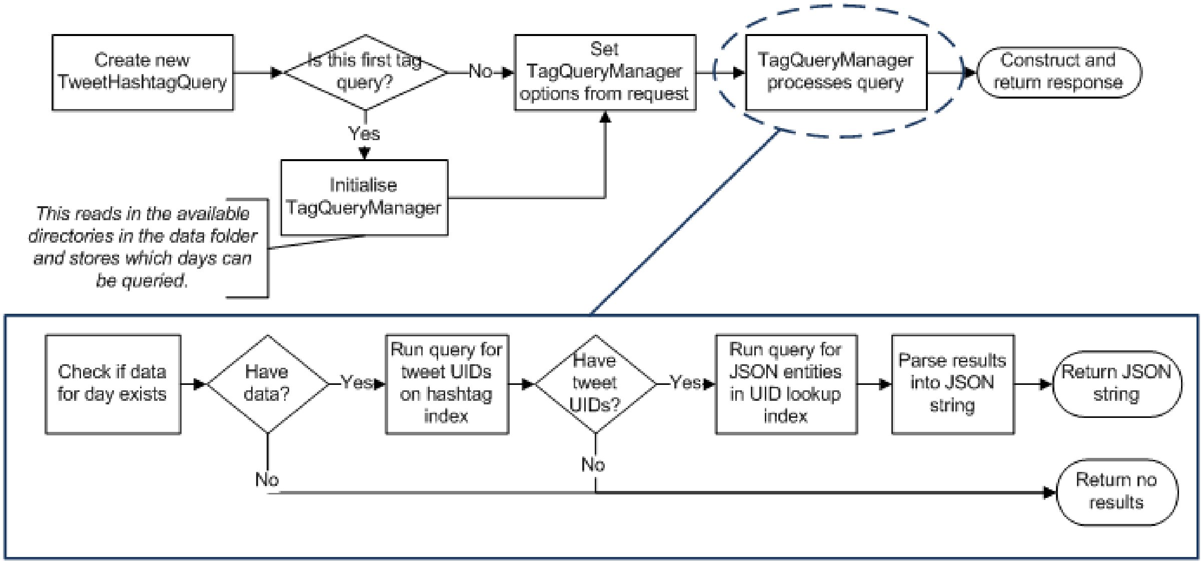

Create new TweetHashtagQuery
Is this first tag query?
No
Yes
Set TagQueryManager options from request
TagQueryManager processes query
Construct and return response
Initialise TagQueryManager
This reads in the available directories in the data folder and stores which days can be queried.
Check if data for day exists
Have data?
Yes
No
Run query for tweet UIDs on hashtag index
Have tweet UIDs?
Yes
No
Run query for JSON entities in UID lookup index
Parse results into JSON string
Return JSON string
Return no results

Figure 6.3: TagQueryManager Process Overview

Earlier versions of Pulse used a series of filters to reduce the result set size before returning to the client. This was deprecated once HBase had been dropped as the main data store — however, the class *TweetHashtagQuery* remains in use. It simply encapsulates the date and tag of the current query, and is passed to the TagQueryManager by the service entry point.

The manager first checks the data directory to see what days are available for querying. Should the manager receive a request for a day with no data it will immediately construct a ‘no results’ response and return this to TwitterQueryServiceImpl. Otherwise it will attempt to search for the tag supplied.

Tweet lookup is conducted in two stages:

1. The relevant hashtag index is loaded from disk, then searched for all occurences of the query tag. If there are any results the Terrier documents IDs returned are used to obtain tweet UIDs from the Terrier meta-index.

2. If there are any results from the first search the UID lookup index is loaded. Each tweet UID is added to a query string, which is then supplied to Terrier for querying. The UID lookup meta-index is then used to obtain the full JSON tweet string.

Stage 1 is illustrated in Figure 6.4.

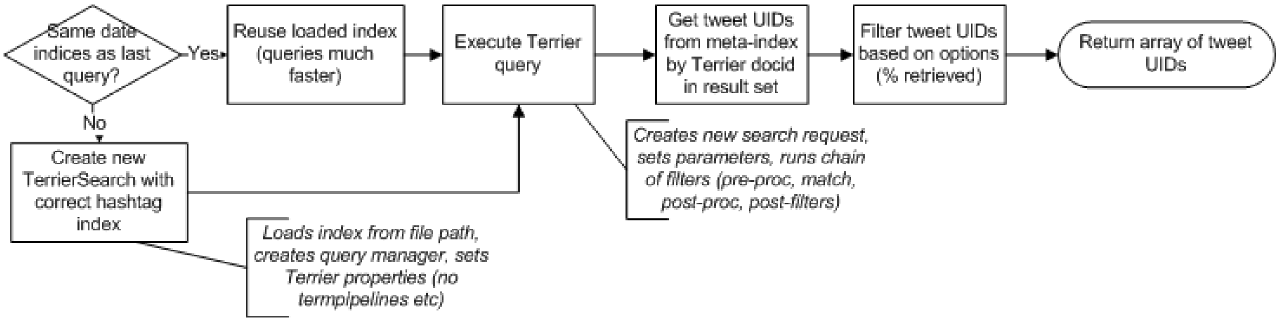

Same date indices as last query?
Yes
No
Reuse loaded index (queries much faster)
Execute Terrier query
Get tweet UIDs from meta-index by Terrier docid in result set
Filter tweet UIDs based on options (% retrieved)
Return array of tweet UIDs
Create new TerrierSearch with correct hashtag index
Creates new search request, sets parameters, runs chain of filters (pre-proc, match, post-proc, post-filters)
Loads index from file path, creates query manager, sets Terrier properties (no termpipelines etc)

Figure 6.4: TagQueryManager Hashtag Lookup

The *TerrierSearch* class, based on the *InteractiveQuerying* [36] class provided with Terrier to demonstrate command line search, is responsible for the end-to-end Terrier querying process. The TagQueryManager creates a two instances on initialisation, one each for hashtag indices and UID lookup indices. These are wholly responsible for:

- *Loading the required Terrier index file*: the path provided by the tag query manager.
- *Applying Terrier properties*: when retrieving results from the hashtag index the level or type of matching and ranking is not important — all that matters is that the result set includes every Terrier document which the tag appears in. The ‘term.pipelines’ property is left blank to ensure no stemming or stopword removal is applied.
- *Executing the query*: Terrier executes the query and provides a set of results, incorporating Terrier document IDs.
- *Extracting meta-data*: for each Terrier document ID returned the appropriate data (either tweet UID or JSON tweet) is extracted from the meta-index (explained in Table 4.3). This data is then returned to the tag query manager.

Each instance of *TerrierSearch* is kept for as long as the index it contains is relevant. Should a user query for multiple topics on the same date consecutively the index only needs to be loaded once. This speeds up round trip query response time quite significantly, as will be discussed in Section 6.4.4. If the query date changes the *TerrierSearch* instances are reset and new indices are loaded.

The result from the first Terrier query, if any, is a list of tweet UIDs. Should no results have been found the query manager will return an empty response to TwitterQueryServiceImpl. Otherwise, the tweet UIDs are filtered based on the ‘Retrieval %’ user option (which specifies the percentage of results to return).

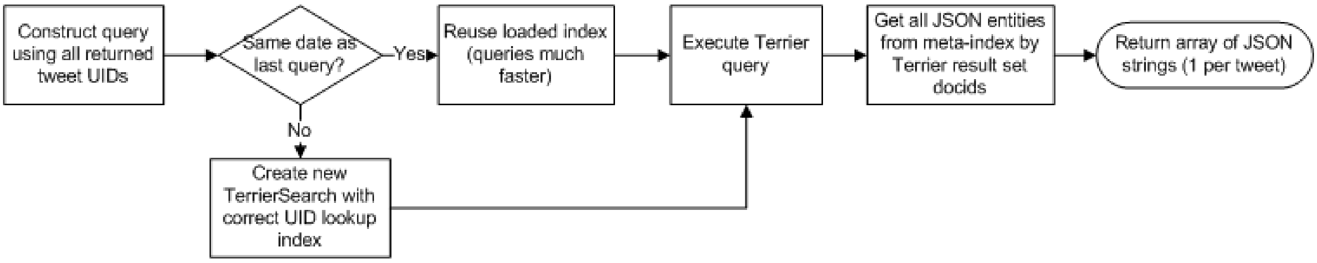

Construct query using all returned tweet UIDs
Same date as last query?
Yes
No
Reuse loaded index (queries much faster)
Create new TerrierSearch with correct UID lookup index
Execute Terrier query
Get all JSON entities from meta-index by Terrier result set docids
Return array of JSON strings (1 per tweet)

Figure 6.5: TagQueryManager UID Lookup

Stage 2, shown in Figure 6.5, uses the retrieved UIDs to lookup each tweet from the UID lookup index.

The UID lookup index stores every tweet for a single day, indexed by UID. This means that by supplying a query string containing space-separated [33] UIDs Terrier will return the complete set of tweet document IDs that those UIDs represent. The document IDs are then used to extract the actual JSON tweet strings from the meta-index, which are returned to the TagQueryManager as an array. The pseudocode for this process is shown below:

```
resultSet = Terrier query process results
metaIndex = index.getMetaIndex()
tweets = new String array
for all document ids returned
    tweet = metaIndex.getItem(jsonTweet, documentID)
    tweets.add(tweet)
return tweets
```

This query-based approach, though not the fastest, does return all tweets required to fulfil the client request. Terrier also provides ‘reverse key lookups’, where document IDs can be obtained by defining a meta-index key which is unique for every document. To enable this the index must be created with 1 or more reverse keys specified via Terrier’s `indexer.meta.reverse.keys` property, and the index merging process must include the appropriate index structures in each merge (enabled via the `merger.meta.reverse` property).

The initial approach then was to simply specify the tweet UID as a reverse meta key, so that rather than having to perform any queries each tweet could be looked up knowing only its UID. However, enabling reverse key lookup caused daily merging to effectively halt (described in Section 4.6.5). A fix was never found to remedy this problem, so reverse keys were not utilised by the final application.

Once the tweet strings have been returned to the TagQueryManager they are checked for integrity. This is a workaround to a bug described in Section 4.6.4. Any tweets obtained from an index created whilst the bug was still in place are checked to ensure they are under 5000 characters long. Those that aren’t (and hence have likely been truncated) are excluded from the final result set.

The final TagQueryManager task is to construct a JSON array of retrieved tweet entities. Testing showed that transfer of a single serialised string is considerably faster than transfer of array of serialised tweet objects, or even an array of strings. As JSON is plain text the tweet array is easily represented in string form. Full testing of each return type tried is described in Section 6.4.4.

The JSON string is returned to TwitterQueryServiceImpl, which in turn creates an *InitialSingleResponse*. InitialSingleResponse is a serialisable class used by hashtag queries to encapsulate the JSON string and a set of optional query statistics (not used by the final application). The response is then returned to the client.

Slashtag queries are also handled by the TagQueryManager, illustrated in Figure 6.6.

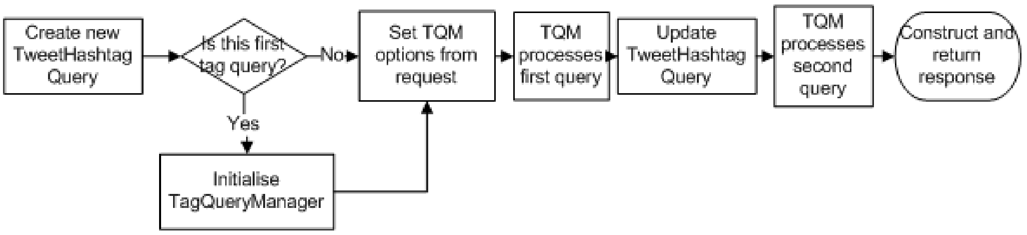

Create new TweetHashtag Query
Is this first tag query?
No
Yes
Initialise TagQueryManager
Set TQM options from request
TQM processes first query
Update TweetHashtag Query
TQM processes second query
Construct and return response

Figure 6.6: TagQueryManager Slashtag Process Overview

The retrieval process for each tag of a slashtag query is identical to the process followed for a single hashtag. However, there are three main differences in managing slashtag queries:

- The query manager performs the entire lookup process twice (once for each tag)
- The overall query process takes less time than if the two tags were queried for individually. Slashtag queries are always for the same date, so once Terrier has loaded that day's index the second tag query returns considerably faster.
- An *InitialDuoResponse* is used to return the results to the client. This encapsulates an additional JSON string and statistics object, one for each tag.

**Temporal Range Queries**

Temporal range query requests, described in Section 5.4.3, are sent to the TemporalQueryManager for processing. The TemporalQueryManager handles interactions with the file system, selection, loading and parsing of the correct daily statistics files (see Section 4.9) and construction of the correct response type. This process is illustrated by Figure 6.7.

Before the first temporal query is started by the manager it performs scans the Pulse data directory for daily statistics file and loads each file it finds into a cache. These files are fairly small ( 30Kb) and so can be held in memory for an entire application session. This is achieved using a hashmap of Unix timestamp / statistics object pairs. On receiving a temporal range query request the manager needs only look in the cache keyset to see if the data exists. If no matching key is found a 'no results' response is immediately returned to TwitterQueryServiceImpl.

Each day's Twitter activity is encapsulated by objects of the class *DailyStats*, with one object stored per statistics file. The TemporalQueryManager is responsible for reading the necessary objects and converting the data within to a format suitable for transmission. As we will see in Section 6.4.4

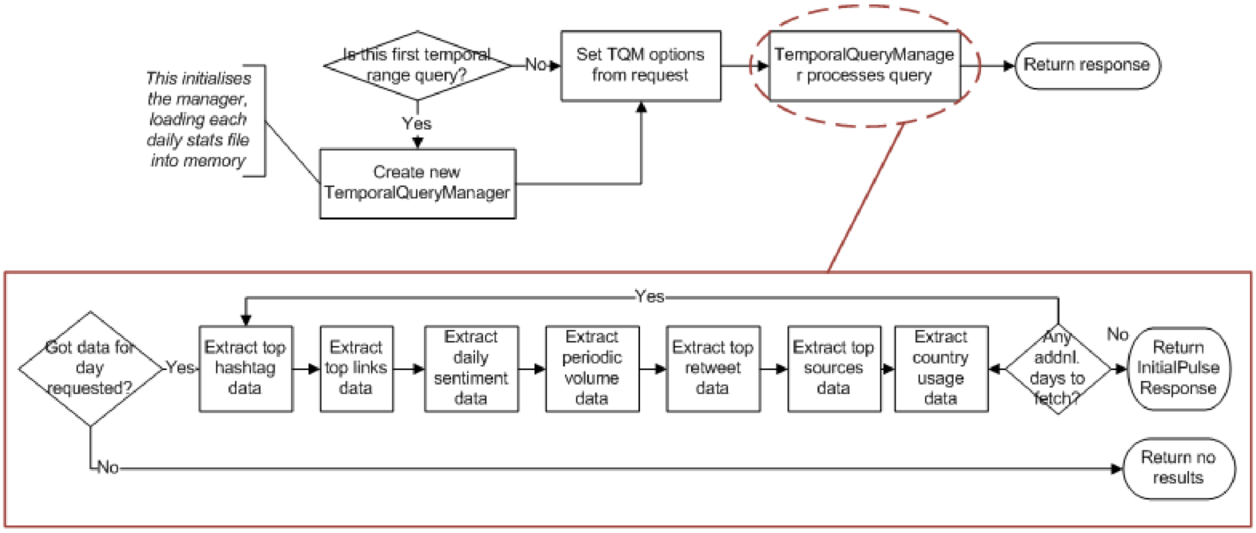

This initialises the manager, loading each daily stats file into memory
Is this first temporal range query?
No
Yes
Set TQM options from request
TemporalQueryManager processes query
Return response
Create new TemporalQueryManager
Got data for day requested?
Yes
No
Extract top hashtag data
Extract top links data
Extract daily sentiment data
Extract periodic volume data
Extract top retweet data
Extract top sources data
Extract country usage data
Any addnl. days to fetch?
Yes
No
Return InitialPulse Response
Return no results

Figure 6.7: TemporalQueryManager Process Overview

complex object serialisation results in very slow data transmission, so simply returning DailyStats objects to the client was not a viable option.

Furthermore users can specify various options configuring how much (or little) of daily activity they want to see. The TemporalQueryManager applies these constraints and filters the results as requested. Filtering options available to users that are applied by TemporalQueryManager include:

- *Additional days*: the number of additional days either side of the query date to return. For example, a temporal range query with a date of January 20th and ‘Additional days’ set to 10 will return data from January 10th to January 30th. The default value is 10.
- *Daily Top Tag Limit*: the number of popular tags by total uses to return. Whilst each file contains details of the top 0.25% of hashtags for that day this can be filtered further by setting a hard limit. The default value is 50.
- *Retweets to Show*: the number of top retweets by total occurences to return. The default value is 20.
- *Popular Links to Show*: the number of top links by total occurences to return. The default value is 20.
- *Num. Devices to Show*: the number of popular devices and corresponding usage volumes to return. The default is 10.

All other temporal range query options are applied by individual components client-side.

If data is available to fulfil a request then the required DailyStats objects are processed. The TemporalQueryManager has several methods which extract and transform data from each DailyStats object so that it can be more easily transmitted and processed by the client. The extraction and transformation process is shown below.

```
for each data type // tags, retweets, links etc.
    dateTimestamp = queryDate.toString
    dataHashmap = new Hashmap // stores daily data for this data type

    for each day before query date
        offset -= 86400000 // 1 day in ms.
        newTimestamp = (queryDate.toLong-offset)
        statsObject = cache.get(newTimestamp)
        stringRepStatsObject = parse(statsObject) // see below
        dataHashmap.put(newTimestamp, stringRepStatsObject)
    offset = 0;

    for each day after query date
        offset += 86400000
        newTimestamp = (queryDate.toLong+offset)
        statsObject = cache.get(newTimestamp)
        stringRepStatsObject = parse(statsObject) // see below
        dataHashmap.put(newTimestamp, stringRepStatsObject)

    return dataHashmap
construct response
```

The data types extracted are shown in Figure 6.7.

The parsing required to create a string representation of each data type varies greatly from type to type. For example, hashtags are stored as *HashtagInfo* objects within the DailyStats file, so these need to be manipulated differently from retweets (which are stored as *RetweetInfo* objects). As a rule the data for each type is transformed into a string, where each unit of data (a single tag or retweet) is comma separated, and the attributes of each data unit (number of tag occurrences, number of unique tag users etc.) is space separated. Every data type in the DailyStats object has an associated parsing method which returns a valid string representation of that data.

A caveat of this approach is that client-side components must know the string structure used so that they can deconstruct the response. Though the structure is fairly simple it would be more efficient for the component manager to parse temporal responses into a uniform format usable by all components, rather than delegating responsibility for parsing to individual components. This would be addressed in a future version of Pulse.

Each parsing method will eventually return a HashMap¡String,String¿ object. The hashmap keys are Unix timestamps, and the values are string representations of daily statistics for the specified data type.

Once all required days have been parsed the accumlated data hashmaps are encapsulated as an instance of the serializable *InitialTemporalResponse* class. This object is then returned to TwitterQueryServiceImpl for transmission.

### 6.4.4 Performance Analysis

As discussed in Section 6.4.1 the most important attribute of the TweetFetch service is responsiveness. Performance benchmarks were carried out using the application to establish response time for a range of queries.

Benchmarks were carried out under the following conditions:

- Network: The client and server were run on separate machines using different network connections to measure network latency.
- Client: Google Chrome 12, running with 2Gb of available RAM on a single-core Windows PC.
- Server: Apache Tomcat 7, running with a 2Gb heap size on a quad-core Windows PC.
- Index Size: the tag query indices used were 254Mb (hashtags) and 4.8Gb (UID lookup)

**Tag Based Queries**

Timing values were taken for each of the following operations:

1. Client - Server transmission
2. Hashtag index — tweet UID lookup
3. UID index — JSON tweet lookup
4. Meta-index data retrieval
5. Parsing
6. Server - client transmission

Certain operations, such as trivial checks and tasks found to have a negligible time footprint, have been excluded for clarity. Additionally the time taken to initialise the service was not included — this happens only once for as long as the application is running, even between multiple browser sessions and users.

Table 6.1 shows the different scenarios used for testing.

Table 6.2 shows timings taken for each operation (all times are in milliseconds).

Queries with a smaller number of results (such as in scenarios 1,2 and 5) or those where indices have already been loaded (such as scenarios 4 and 6) did return within the time limit aim of 10 seconds. As tag usage frequencies follow a power-law distribution this should be the case for the majority of hashtags in the data store.

However, those with over 1000 results (particularly from a non-cached index) return after a much longer delay than the suggested responsiveness limit.

| ID | Query Type | Index Previously Loaded? | Result Set Volume |
|---|---|---|---|
| 1 | Hashtag Query | FALSE | 143 |
| 2 | Hashtag Query | TRUE | 143 |
| 3 | Hashtag Query | FALSE | 1022 |
| 4 | Hashtag Query | TRUE | 1022 |
| 5 | Slashtag Query | FALSE | 143 / 124 |
| 6 | Slashtag Query | TRUE | 143 / 124 |
| 7 | Slashtag Query | FALSE | 1022 / 1061 |
| 8 | Slashtag Query | TRUE | 1022 / 1061 |

Table 6.1: Tag-Based Scenarios

| ID | Cli-Serv. Trans. | HT index lookup | UID index lookup | Parsing | Serv.-Cli. Trans. | Total |
|---|---|---|---|---|---|---|
| 1 | 7 | 1472 | 2845 | 3 | 51 | 4379 |
| 2 | 6 | 36 | 1364 | 2 | 55 | 1465 |
| 3 | 6 | 3751 | 19588 | 7 | 104 | 23459 |
| 4 | 9 | 121 | 9600 | 6 | 119 | 9859 |
| 5 | 11 | 2153 | 5116 | 6 | 82 | 7373 |
| 6 | 9 | 57 | 2610 | 5 | 89 | 2776 |
| 7 | 8 | 5917 | 34117 | 22 | 178 | 40249 |
| 8 | 8 | 381 | 19726 | 20 | 166 | 20309 |

Table 6.2: Tag-Based Query Performance

This is mainly due to the process of querying the UID lookup index, then retrieving the relevant meta-data for each tweet UID. As discussed earlier Terrier does offer reverse key lookup for obtaining document IDs using a unique meta-data value, but this caused issues at index merging stage described in Section 4.6.5. Indices featuring reverse lookup keys were not available for testing, however given future development time resolving the index merging / reverse key issue would be a high priority development task.

Query responsiveness could also be improved hy further harnessing the benefits of caching, extending the cache to include a larger number of data ranges. One potential way of achieving this would be to implement an *LRUMap* [28]. The LRU (Least Recently Used) Map is part of the Apache Commons Collection library [11], providing a map implementation with a fixed number of entries. Once the map has been filled to capacity adding any new entries automatically removes the least-recently used entry.

Currently the tag query manager loads a new index if the current query date is different from the previous query date. Instead, an LRUMap could be used to store multiple TerrierSearch instances for various dates. The benefits of using preloaded indices are obvious (compare preloaded scenario 8 with unloaded scenario 7, for example), and would over an application session greatly improve query responsiveness.

**Temporal Range Queries**

Timing values were taken for each of the following operations, using a sample of 36 daily statistic files.

1. Client - Server transmission
2. Response creation
3. Server - client transmission

Given that all files are cached on initialisation, and that the processing performed is mainly string manipulation, it was expected that temporal range queries would be very quickly processed and returned to the client.

Table 6.3 shows the different scenarios used for temporal query performance testing.

| ID | Additional Days |
|---|---|
| 1 | 2 (1 either side) |
| 2 | 20 (10 either side) |

Table 6.3: Temporal Query Scenarios

Table 6.4 shows timings taken for each operation (all times are in milliseconds).

| ID | Cli-Serv. Trans. | Response creation | Serv.-Cli. Trans. | Total |
|---|---|---|---|---|
| 1 | 3 | 14 | 121 | 139 |
| 2 | 2 | 53 | 271 | 328 |

Table 6.4: Temporal Query Performance

As expected each query returned in tenths of a second. With several hundred daily statistics files this may have taken longer, however the non-linear increase in retrieval time suggests that the process has a baseline ‘fixed cost’, on top of which additional statistics files add very little processing time. This is also well within the desirable response limit as stated in Section 6.4.1.

## 6.5 Sentiment Analysis Service

The Sentiment Analysis Service provides assistance in analysing the sentiment of each tweet in a result set. It is invoked by the Sentiment Analysis component once the initial result set from the Tweet Fetch service has been received. The approaches considered, analysis requirements and implementation detail are are discussed fully in Section 5.7.8.

## 6.6 Deployment

Deploying Pulse for production use is a straightforward process. GWT automatically generates a Web Application Archive (WAR) which can be easily launched on a Java webserver such as Apache Tomcat. The WAR directory contains both the client JavaScript and server classes / libraries required to run the application, as well as all images and CSS stylesheets.

Certain filepaths must be defined for Pulse so that it can locate the data directory and other required files. These are configured using a Pulse properties file, pulse.properties. This file is stored within the WAR directory and should be reviewed before deployment. On intiailisation each service will read the properties and configure itself based on the property values. The three properties currently used are listed below.

- `pulse.data.dir`: path to the Pulse data directory. Folder names in the path should not contain any spaces.
- `sentiment.dictionary.file`: path to the sentiment analysis dictionary required by the Sentiment Analysis service.
- `sentiment.pos.tagger.file`: path to the sentiment analysis POS tagger file.

Comments can be added to the file by prefixing a new line with the '#' character.

Practical instructions for deploying Pulse were consolidated into an Administrator Guide, This has been included as Appendix E.

## 6.7 Conclusion

This chapter described services, server-side applications that provide support to client-side components. The main tweet retrieval service, Tweet Fetch, was described in detail, explaining the development process and discussing the implementation details of the final system.

Performance analysis was also carried out to establish system responsiveness. Though certain queries take longer than the 10 second target to complete, the majority of those tested did return within the time limit. For those that did take longer several areas for improvement were identified that could be developed in a future version.

The next chapter introduces an empirical evaluation carried out with users to establish the usability and desirability of Pulse.

# Chapter 7

# User Evaluation

Ensuring that Pulse delivered a positive user experience was one of the key system criteria defined at the start of the project. The user evaluation described in this chapter provides an empirical process to establish how successful Pulse is in meeting this criterion, through both qualitative and quantitative means.

## 7.1 Evaluation Aims

The user evaluation provided an unprecedented opportunity to observe how different types of user interacted with Pulse, in directed and undirected scenarios. The evaluation was structured to measure the system against three criteria:

- *Usability*: all of the functionality in Pulse must be easily accessible by the end user enabling them to achieve their goals and be productive.
- *Desirability*: the application should be desirable; encouraging high user engagement, satisfaction, entertainment, and ideally, fun.
- *Usefulness*: the information, visualisations and analysis supplied by Pulse must result in the user understanding a particular facet of Twitter in a meaningful way after usage.

## 7.2 Evaluation Preparation

With the aims of the evaluation defined an empirical procedure was required to measure the success of the system against each aim. This involved selecting appropriate techniques to elicit information from participants, designing a task list which steadily introduced new concepts whilst maintaining user interest, and obtaining pertinent, comprehensive feedback from each participant detailing their opinions of the system.

### 7.2.1 Pre-Evaluation

The pre-evaluation questionnaire was designed to establish the prior experience of each participant in areas related to Pulse. This can then be used in conjunction with evaluatory results to understand usability and desirability relative to each participant's social media experience.

Three main areas were considered in the pre-evaluation questionnaire:

- *Social Networking Experience*: the most general category, to understand participant's confidence with basic social networking, their usage habits and preferred means of networking interaction. Understanding the 'casual user' response will help determine the accessibility of Pulse.
- *Twitter Experience*: Pulse requires a basic understanding of Twitter to fully utilise the features it offers. Understanding how much knowledge of the Twitter domain each participant has allows for comparison of results between novice and expert users, and whether prior experience affects usability or desirability.
- *Trend Analysis Experience*: the most specific category, aimed at those with experience in tools similar to Pulse. Understanding the opinions of experienced participants is critical for assessment of Pulse in the wider context of other social media analytical tools.

### 7.2.2 Evaluation Tasks

Creating evaluation tasks for Pulse was challenging due to the exploratory nature of the system. Pulse encourages exploration by providing natural information discovery paths between each query — as such, there is no 'ground truth' against which the system can be compared to objectively rank its success. However, participants must still be guided so that each has a uniform baseline experience to rank once the evaluation is complete.

To overcome these difficulties the evaluation task list incorporates three types of activity. The first, *walkthroughs*, introduces participants to the core functionality of Pulse through simple, step-by-step activities. Each clearly introduces a specific aspect of the system using exact instructions ensuring all participants are trained to the same level. In addition the sample queries selected for each walkthrough were chosen so that they very clearly and explicitly demonstrated the purpose of the walkthrough.

The following functionalities were highlighted by the evaluation walkthroughs:

- *Hashtag Search*, introducing the most basic type of query within Pulse.
- *Hashtag Comparison (By Date)*, introducing the ability to compare the result of two different queries by date.
- *Hashtag Comparison (By Tag)*, emphasising that two queries featuring different tags can also be compared.
- *Options*, providing a brief overview of how options can be used to configure the query process.

- *Slashtag Search*, introducing the slashtag query syntax for faster comparative searching.
- *Time Querying*, introducting the concept of time queries (without any query tag) to see an overview of daily Twitter activity.

Whilst walkthroughs are necessary for user training, they alone do not allow for task-orientated usage. To ensure participants had an opportunity to use Pulse with a goal in mind a second activity type, *scenarios*, was also included. Scenarios require the user to find a particular piece of information using experience gained from the walkthroughs and by utilising the user manual. No guidelines are given on how this should be achieved; rather the user must explore Pulse and find the required information in whatever way they deem most appropriate.

For every scenario there are a number of methods which can be used to obtain the required data — however, there is always one method which is simplest. Users then needed to be observed to understand whether the easiest method was chosen, and if not, why not. It could be because it was not immediately obvious, difficult to access or simply obfuscated. Should any of these be true it would have a measurable impact on user satisfaction.

Scenarios were also used to showcase the variety of components in Pulse. Each scenario focuses on one or more specific components, providing participants with further learning opportunities and experience to reflect on once the evaluation was complete. Ensuring the scenarios were varied was also crucial in maintaining user interest and establishing which components were most useful and engaging.

The scenarios selected for the evaluation are shown in Table 7.1.

| Scenario | Functionality | Component Focus |
|---|---|---|
| 1 | Hashtag Query | Geographical Heatmap |
| 2 | Hashtag Query | Tag Volume |
| 3 | Comparison (Date) | Tagged With / Tag Volume |
| 4 | Slashtag Query | Geographical Heatmap |
| 5 | Time Query | Trends |
| 6 | Time Query | Trends |
| 7 | Time Query | Retweet Paths |
| 8 | Time Query | Options + Trends |
| 9 | Time Query | Options + Place Device |

Table 7.1: User Evaluation Scenarios

Given the time constraints of the evaluation it was unfeasible to include every component for each scenario. However, once the main scenarios have been completed users are invited to undertake system *exploration*. Put simply participants were encouraged to query for any topic or period of interest, interacting with components freely. Whilst difficult to analyse quantatively the aim was to see how users, with experience, would use the system given no direction.

The final evaluation task list also included a short preamble explaining the evaluation procedure, points which the participant should consider as they completed the evaluation, and standard information concerning evaluation etiquette.

### 7.2.3 Post-Evaluation Review

Typically user evaluations are conducted to establish the usability of a system. Usability, defined as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” [51], is only one part of the user experience. Though an important aspect of system evaluation it is primarily focused on assessing the ease in which users can achieve certain goals.

As users turn to technology for entertainment and general interest the ‘desirability’ of a system must also be considered. Desirability, defined as “Worth having or seeking, as by being useful, advantageous, or pleasing” [44], concerns intangible, qualitative terms which can be applied to products: fun, engaging, boring, useful, complex, overwhelming and colourful, for example. However, assessing the desirability of a system through standard quantitative means is subject to various user biases.

The post-evaluation review is divided into two distinct sections: one aiming to assess desirability, and one assessing usability.

#### Desirability Analysis

A method was developed based on tools created by Microsoft to establish how desirable participants found the system. The process as described by Benedek et al [44] follows:

1. Participants are given a selection of ‘reaction cards’, each containing a single adjective.
2. Participants must then sort these cards and select which apply to the product being evaluated.
3. The cards are then filtered further, with the 5 most relevant cards remaining.
4. The evaluator then conducts a guided interview with participants to understand the reasoning behind selecting each of the 5 remaining cards.

This process had the potential to take too long given the 45 minute limit of the evaluation, so instead of reaction cards a word list was created. Participants were asked to initially check each word which they felt applied to Pulse, then circle the 5 most relevant. A short guided interview was then conducted to understand why those particular words had been highlighted.

Whilst this process is undoubtedly more time-consuming than a questionnaire, it provides a wealth of qualitative data which can be used to directly improve the user experience. It also helps to offset the inevitable positive and acquiesence biases inherent in almost all forms of questionnaire,

#### Usability Analysis

With a method for assessing desirability in place standard means were used to assess application usability. An oft-used quantitative method was utilised, developed by IBM [53], which incorporates a series of statements and associated Likert scales on which participants can state their level of agreement.

The scale used ranges from 1 (‘Strongly Agree’) to 7 (‘Strongly Disagree’), with each statement focusing on a particular aspect of usability: ease of learning, productivity, system help, clarity of information and so forth.

Though the IBM evaluation contains 19 statements several of these were better ranked through the desirability analysis. For conciseness these were removed, leaving 14 statements for participants to rate.

Though this questionnaire is subject to both positivity and acquiescence bias it still serves a useful purpose: giving quantitative data which can be compared to the qualitative results of the desirability analysis to establish whether there are significantly different conclusions that can be drawn from each result set, and if so, what those differences are.

### 7.2.4 Materials

The following materials were developed for the user evaluation:

- *User Guide*: a comprehensive user-focused document explaining the features available, as well as a ‘Quick Start’ guide allowing users to immediately become productive in Pulse.
- *Pre-Evaluation Questionnaire*: initial questions to establish participant familiarity with concepts closely linked to Pulse: social network experience, Twitter experience, and trend analysis experience.
- *Evaluation Task List*: instructions for the evaluation and a series of tasks designed to encourage learning and exploration of the system.
- *Post-Evaluation Review*: questions for users which encourage reflection on their use of Pulse, as well as a word list used to perform a guided interview with the participant.
- *Administrator Guide*: supplementary documentation for administrator’s detailing how to configure a new instance of Pulse.

These materials are included as appendices to this document.

### 7.2.5 Participant Recruitment

Participants from the following demographics were invited to evaluate Pulse:

- *Computing Science Undergraduates*: other students within the university with a solid technical background in computing science.
- *Computing Science Postgraduates*: graduate-level students with a particular focus on information retrieval
- *Non-Computing Science Participants*: those from other fields with varying levels of Twitter / computing knowledge.

The user evaluation was completed by 7 participants in total.

### 7.2.6 Evaluation Sessions

Evaluations were carried out in two environments: a computing science laboratory at the University of Glasgow, and the home of the author. In each case two monitors were configured on a single machine providing each user with the best possible visual experience. Though this is arguably misrepresentative of Pulse were it to be deployed as it stands the need for further low-resolution optimisation has already been documented as a necesary future development.

Before the evaluation began participants were briefed on the system and given the opportunity to ask any questions. The evaluation process was also explained.

Evaluations were estimated to take 45 minutes, though in practice this varied based on user engagement, post-evaluation discussion and interest in the system.

## 7.3 Pre-Evaluation Results Analysis

### 7.3.1 Social Networking Experience

The first series of questions were designed to establish each participants general social networking experience. Figure 7.3.1 illustrates the variety of social networks used by participants.

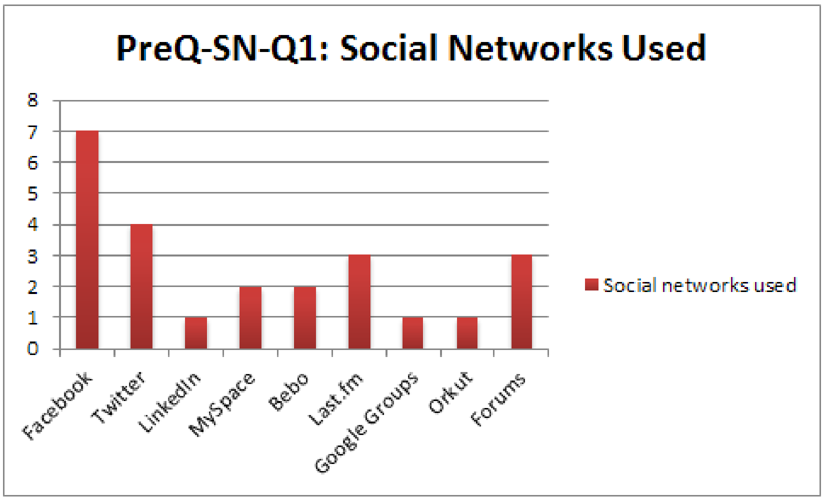


Figure 7.1: Social Networking Experience

All participants used Facebook, unsurprising given its ubiquity amongst university students. In addition 3 participants used 4 or more different social networks (including Twitter), suggesting each may have a particular role or function, or fulful a different need, within the participants usage of the social web. Most importantly all participants had an understanding of the core concepts involved in social networking: relationships, shared content, and shared interests.

Figure 7.3.1 illustrates the reasons participants selected for using social networking sites.

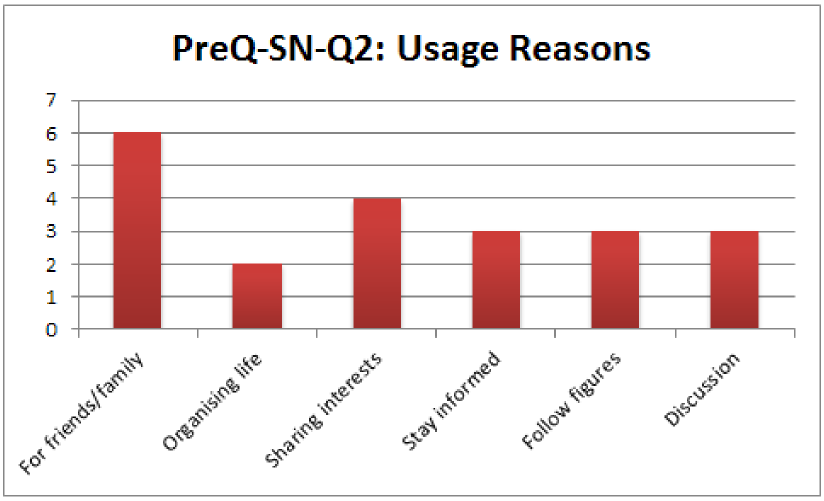


Figure 7.2: Social Networking Usage Reasons

The most common reason for social network usage was for keeping in touch with friends and family. Those participants that also selected ‘sharing interests’ all selected ‘stay informed’, ‘follow figures’ and ‘discussing’ as well. This suggests that those using social networks only to stay in touch used Facebook alone, and participants selecting other options had presences elsewhere on the web to satisfy those requirements.

Figure 7.3.1 illustrates the amount of time participants spent using social network sites.

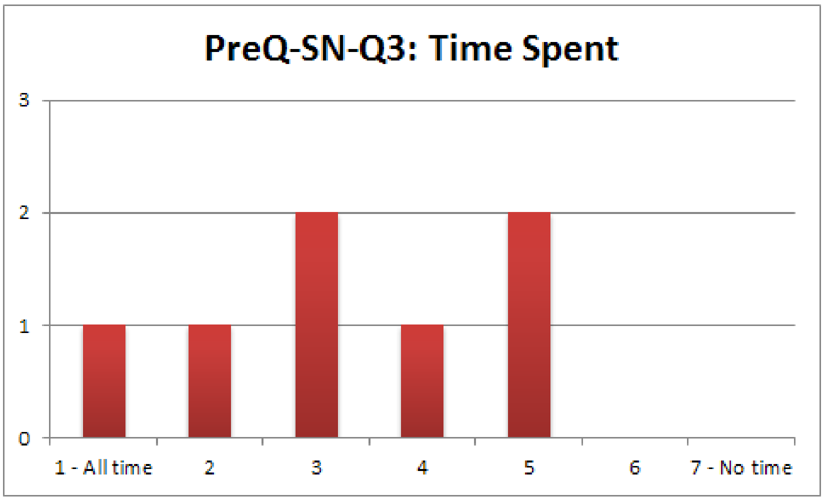


Figure 7.3: Social Networking Usage Level

Social networking usage was very common amongst participants, with the majority spending between half and all of their time online using them. For those participants who use Twitter this means understanding the functionality Pulse provides in Twitter terms (hashtag trends, top retweets etc.) should be considerably easier. For those uninitiated to Twitter it may prove challenging.

### 7.3.2 Trend Analysis Experience

The expectation of the majority of participants was that they would not have any prior trend analysis experience. Figure 7.3.2 illustrates those that had and had not.

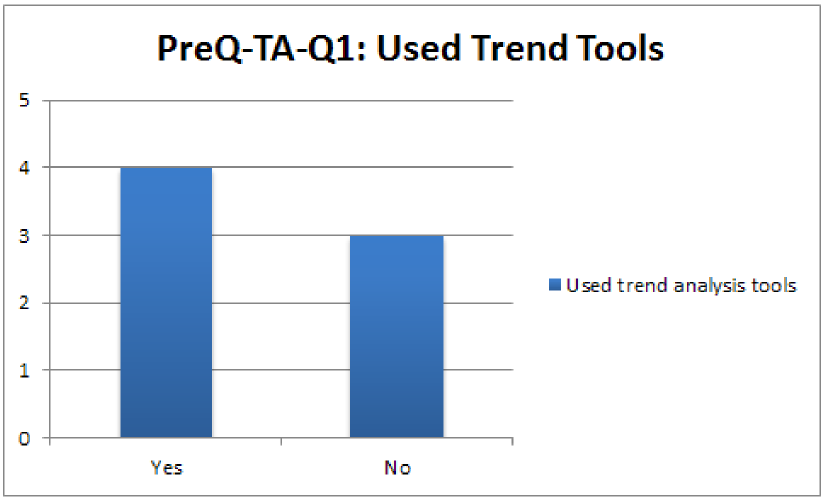


Figure 7.4: Trend Analysis Tools Used

Interestingly over half the the evaluation group had some trend analysis tools experience. Whilst this was positive in that these users could be expected to understand Pulse capabilities faster, this is a much larger proportion than one would expect in typical user group of the same age from outside of an academic setting, and has the potential to bias the results.

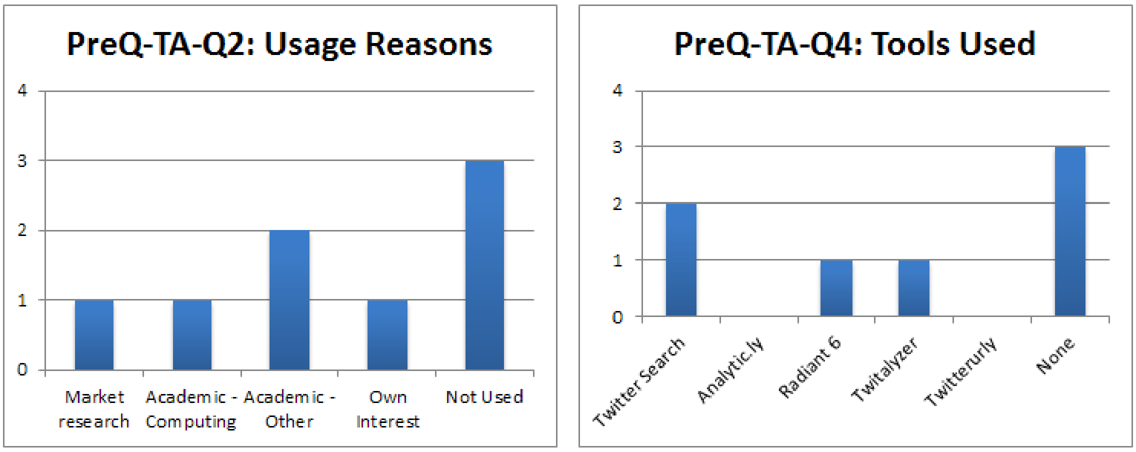


Figure 7.5: Reasons for Use and Tools Used

Figure 7.5(a) illustrates reasons participants had used trend analysis tools, whilst Figure 7.5(b) shows the tools used. Of those participants with analytical experience, one had previously worked as an analyst at a social media marketing business. When asked what features were most useful in tools they had used, it was "the ability to drill down on metrics for granular data". Most however had only used Twitter Search (a useful feature of which being the "dynamically updated results page"), or none at all.

### 7.3.3 Twitter Experience

The Twitter usage rate of each participant was also investigated.

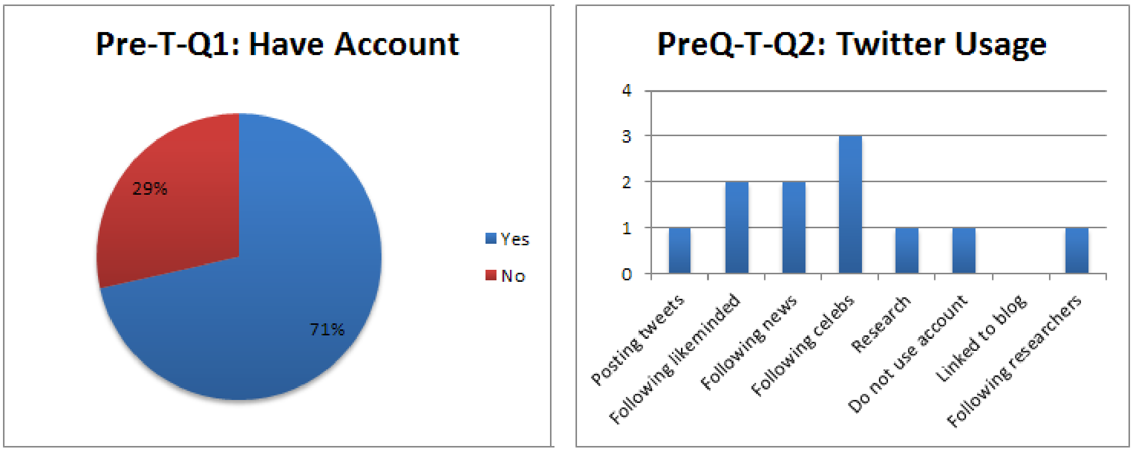


Figure 7.6: Users with Twitter Accounts / Frequency of Account Use

Figure 7.6(a) highlights those with Twitter accounts, and Figure 7.6(b) shows how often these are used by the participants. Those with accounts primarily used them to follow celebrities / media personalities or news. Interestingly only 1 participant actually posted tweets — consumption of content being more prolific than production in the Twitter domain for this user group.

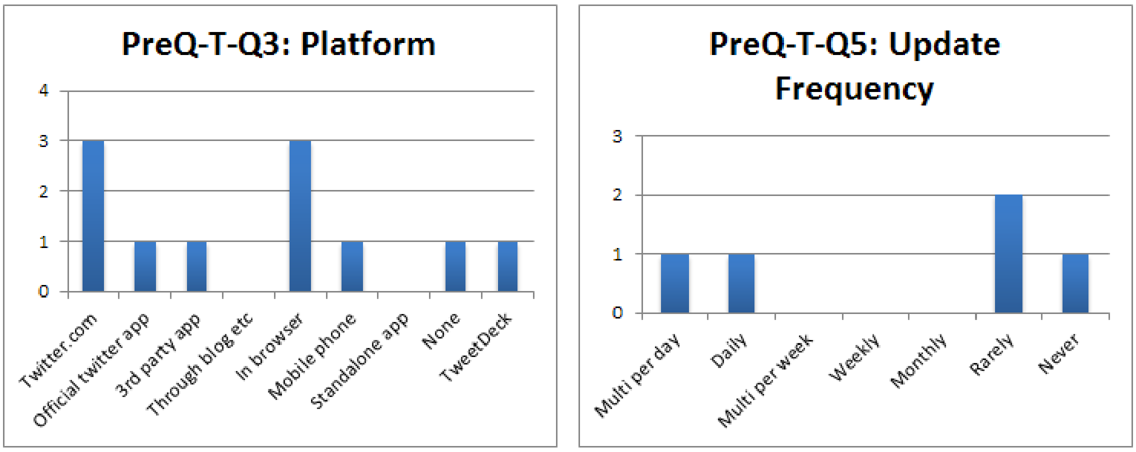


Figure 7.7: Twitter Access Method / Twitter Usage Frequency

Figure 7.7(a) shows how participants accessed Twitter, whereas Figure 7.7(b) shows the frequency of use. Unsurprisingly Twitter usage from the desktop in a browser is the most common access method across participants (this is also true for every day that Pulse has data for). In terms of update frequency participants are split between high-activity and very infrequent users, with little middleground — potentially signifying that users need to 'catch the Twitter bug' before it becomes a regular part of their social interaction online.

The apparent lack of interest by some participants in Twitter may also have an effect on user engagement with Pulse; if the source itself is not of interest to a user, will visualisations and analysis of data from that source be any more interesting?

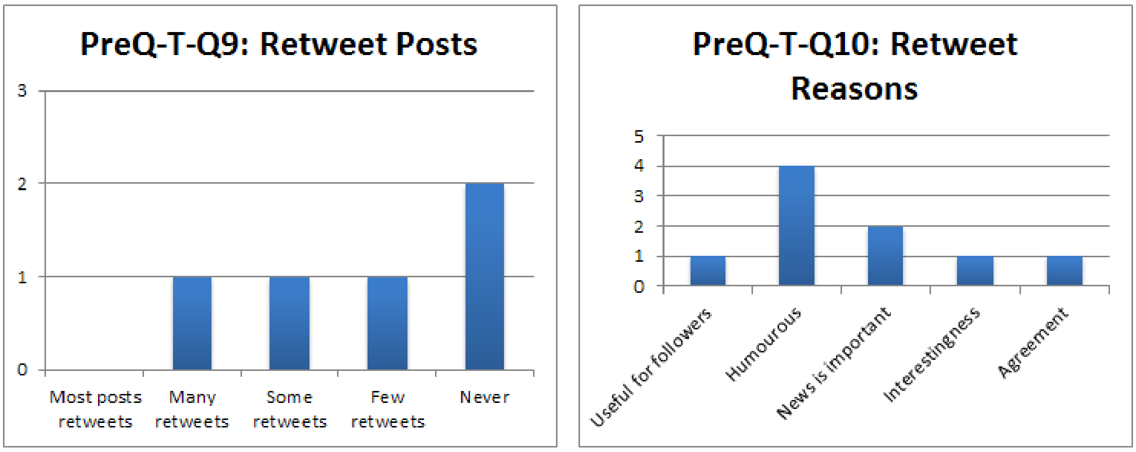


Figure 7.8: Retweet Frequency / Retweet Situation

Figures 7.8(a) and 7.8(b) focus on the participants retweeting frequency, and under what situations participants retweet. The majority of retweets are simply humourous comments that participants wish to share — though based on the number of participants and their posting frequency it is hard to draw further conclusions.

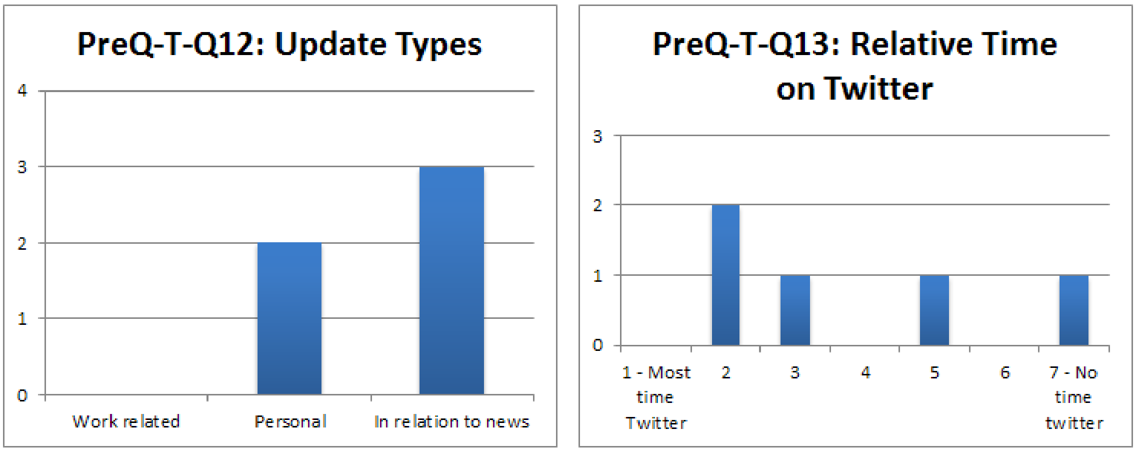


Figure 7.9: Update Type / Comparison with Other Networks

Finally Figure 7.9(a) shows the type of update participants tend to post, as well as the relative time compared to other social networks that participants spend on Twitter (Figure 7.9(b)). As with update frequency participants tend to either spend a significant amount of social networking time on Twitter, or very little. The functionality provided by Twitter is quite different to the relationship-based nature of alternatives such as Facebook, so for those wishing to keep in touch with friends and family using social networking Twitter may not be the preferred option.

### 7.3.4 Conclusions

The pre-evaluation questionnaire helped to unearth several properies of the participant group. Most importantly Twitter is generally seen as a source of content, rather than an outlet. Participants primarily follow users of interest, choosing to read those users tweets rather than post their own.

It also established that social networking is ubiquitous amongst the evaluation participants. Even participants with no deep technical experience are familiar with the interface paradigms of social networking sites and the web — all of which Pulse is based upon.

Finally the time invested by users on social networking sites, as a proportion of all time online, is fairly high. This level of user engagement with social networking itself will hopefully translate into a similar level of engagement with Pulse.

## 7.4 Post-Evaluation Results Analysis

As discussed in Section 7.2.3 the post-evaluation review was split into two discrete sections, each of which is discussed below.

### 7.4.1 Desirability

Users were first asked to rate Pulse by selecting one or more words from a list of 106 adjectives. Table 7.2 shows the results of this process:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Useful | 7 | Organised | 3 | Stable | 2 | Frustrating | 1 |
| Advanced | 6 | Reliable | 3 | Accessible | 1 | Inadequate | 1 |
| Bright | 6 | Responsive | 3 | Business-like | 1 | Intimidating | 1 |
| Impressive | 6 | Appealing | 2 | Clean | 1 | Obscure | 1 |
| Effective | 5 | Cluttered | 2 | Clear | 1 | Satisfying | 1 |
| Engaging | 5 | Credible | 2 | Comprehensive | 1 | Simple | 1 |
| Creative | 4 | Cutting edge | 2 | Confusing | 1 | Slow | 1 |
| Entertaining | 4 | Efficient | 2 | Consistent | 1 | Stimulating | 1 |
| Fast | 4 | Fresh | 2 | Controllable | 1 | Time-consuming | 1 |
| Meaningful | 4 | Friendly | 2 | Convenient | 1 | Too technical | 1 |
| Usable | 4 | High quality | 2 | Counter-intuitive | 1 | Trustworthy | 1 |
| Approachable | 3 | Innovative | 2 | Desirable | 1 | Unconventional | 1 |
| Attractive | 3 | Overwhelming | 2 | Easy to use | 1 | Understandable | 1 |
| Busy | 3 | Powerful | 2 | Effortless | 1 | Unpredictable | 1 |
| Complex | 3 | Professional | 2 | Energetic | 1 | | |
| Fun | 3 | Relevant | 2 | Exciting | 1 | ***Total Positive*** | 122 (85.9%) |
| New | 3 | Sophisticated | 2 | Flexible | 1 | ***Total Negative*** | 20 (14.1%) |

Table 7.2: Word Selection Frequency

Of all words selected nearly 86% were classified as ‘positive’, which itself is validation of the application’s user appeal. Every participant described Pulse as ‘useful’, whilst almost all described it as ‘impressive’ and ‘advanced’. Satisfyingly from a usability perspective it was explicitly deemed ‘usable’ by 4 participants, and even ‘desirable’ by 1 participant. The prevalence of these terms can be seen in Figure 7.11

Furthermore, 5 participants described the system as ‘engaging’ (with one remarking that ‘it would take hours to see everything’), 4 participants called it ‘entertaining’ and 3 described it as ‘fun’.

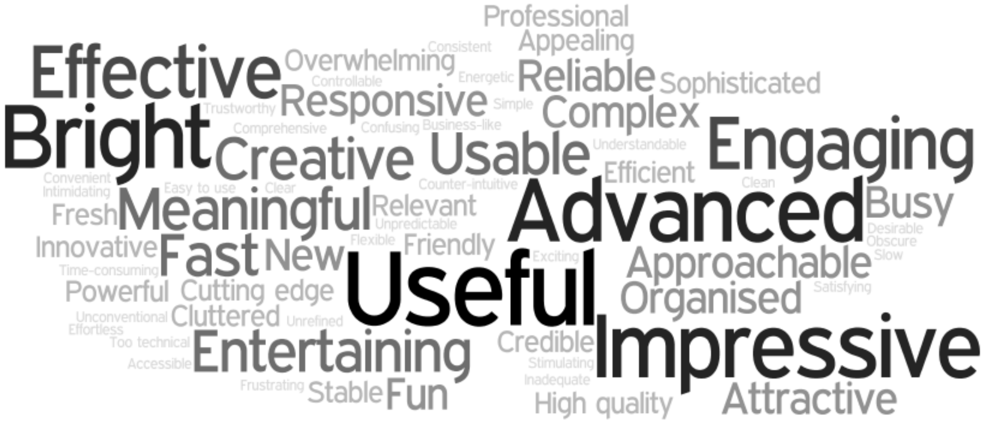


Figure 7.10: Adjective Cloud

This suggests that the application is more than a data analysis tool able to provide certain services, but also an enjoyable user experience that could be used for entertainment purposes as much as task-orientated search.

Other highlights include positive acknowledgement of the aesethetic design (‘attractive’ and ‘creative’ in particular), as well as recognition of the potential uses of the system (‘meaningful’ and ‘business-like’). Furthermore the innovative aspects of Pulse were noticed by 2 participants, labelling it ‘cutting-edge’.

The most frequently occuring negative terms, ‘busy’ and ‘cluttered’, highlight the difficult in presenting so much information within the confines of a browser window. This indicates further work needs to be performed in hiding or abstracting functionality until requested by the user.

It was suggested by one participant that only one component be shown full-size at any given time, with a series of preview windows along the bottom of the screen used to switch between components. This is an excellent suggestion, particular for low-resolution monitors, and would be worth investigating further given additional development time.

Other negative terms used, such as ‘overwhelming’ and ‘confusing’, emphasised that users need to be introduced to each component in a carefully managed way to prevent mental overload. The current online help in Pulse is functional but basic; adding a tutorial video or animation overlay on first use could help to alleviate the initial mental workload of understanding the function of each component.

Certain words chosen were applicable to very specific situations. For example, ‘counter-inituitive’ was picked by one participant due to the way time queries are filtered using a tag whitelist (rather than typing tags in the query box). Another example of this was the term ‘exciting’, which was specifically directed towards the Trends Component shown after a successful time query.

Though the word list is unmatched in understanding user sentiment there are inconsistencies which are difficult to resolve, particularly where participants interpret words differently. For example, 4 participants rated the the application as ‘fast’, whilst 1 judged it to be ‘slow’. Similarly 2 participants described the application as ‘friendly’, whereas one used the word ‘intimidating’. These scenarios are inevitable when using a subject evaluation technique, but given more time the interview would have included a request for clarification from participants when they used a term that several others had completely disagreed with.

Table 7.3 shows the results of the filtering process, narrowing down words selected by each participant to the 5 most relevant.

| | | | | | |
|---|---|---|---|---|---|
| Impressive | 3 | Innovative | 1 | Unconventional | 1 |
| Useful | 2 | Inadequate | 1 | Confusing | 1 |
| Bright | 2 | Overwhelming | 1 | Intimidating | 1 |
| Usable | 2 | Busy | 1 | Fresh | 1 |
| Advanced | 2 | Cutting edge | 1 | Organised | 1 |
| Engaging | 2 | Simple to Use | 1 | New | 1 |
| Cluttered | 2 | Powerful | 1 | Unrefined | 1 |
| Time-consuming | 1 | Complex | 1 | High Quality | 1 |
| Effective | 1 | Dated | 1 | Creative | 1 |
| | | ***Total Positive*** | 25 (71.4%) | ***Total Negative*** | 10 (28.6%) |

Table 7.3: Filtered Word Selection Frequency

Forcing participants to pick only 5 words resulted in some interesting changes to the frequency table. ‘Impressive’ still features as the one of the most common adjectives, with participants commenting that the system was “well designed, really responsive”, and that it had “a huge number of features, lots of interesting stuff”.

The participant with the most social media analysis experience made some interesing points during the guided interview. One of the filtered words they chose to use was ‘inadequate’, in that Twitter user information is not represented in any way by the system. Whilst this was initially part of the design of Pulse it became clear that trying to cover topics, time and users would simply be impossible given the project deadline. However, the fact that an experienced user of analysis tools has considered this an important omission suggests the triumvirate of users, topics and time is essential for a fully-fledged analysis application.

Those participants with no prior experience using Twitter made some useful comments regarding ease of use and UI consistency. In particular, when typing in the query box a slash is used to differentiate between two tags. However, the options tag whitelist uses a comma for this same purpose. Syntax for similar tasks should be consistent across the whole application, and only through evaluation can such errors be recognised and rectified.

As with the general word list ‘useful’ was used twice by participants in their filtered word lists to describe Pulse. One spoke about the “many possible uses” available, and that it allows for “deep investigation” of periods of time. Another user said that “discovery was important” to them, analogising it to Wikipedia: sometimes they are looking for a particular piece of information , other occasions it’s a tool to randomly discover new things.

The term ‘cluttered’ appeared again in 2 participants’ filtered word lists. One participant described it as having “lots of stuff everywhere, information overload”. Another remarked that “it’s hard

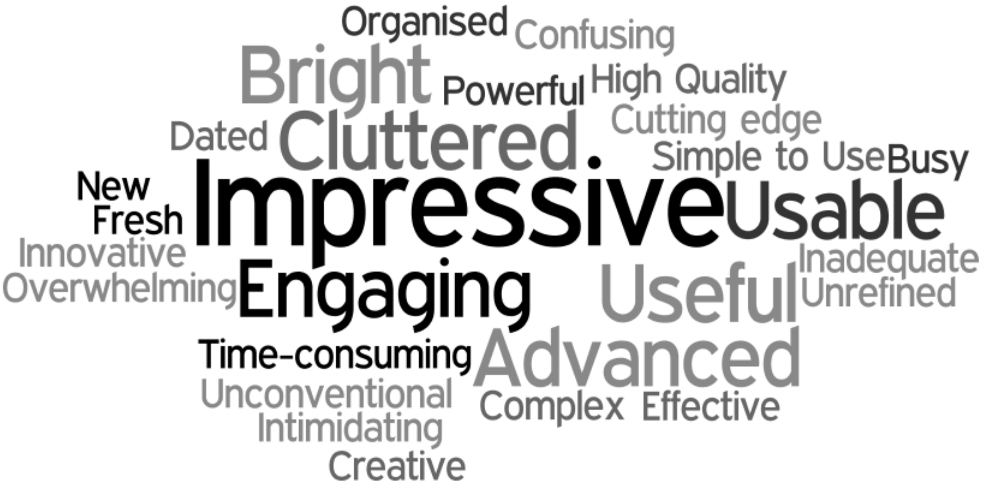


Figure 7.11: Filtered Adjective Cloud

to avoid with so many features, but still need to declutter". Redesigning the component area to shrink components not in active use would go some way to alleviating this issue, and is considered a priority future development.

### Verbal Protocol Analysis

A quantitative analysis of each guided interview was also collated by performing a retrospective verbal protocol analysis [48]. This involves coding each participants response during the guided interview as 'positive' or 'negative' to establish general sentiment towards their experience using Pulse. Table 7.4 shows the results of this analysis for each participant, as well as additional noteworthy comments.

From the results it can be seen that the percentage of positive filtered words correlates almost exactly to the aggregate positive sentiment of the verbal protocal analysis — both resulted in a score of 71%. This is particularly remarkable given that each was based on entirely different measurements and elicitation methods.

The remaining percentage of negative remarks generally concerned the busyness or cluttered nature of the user interface once components were displayed. These criticisms are understandable, as Pulse is currently an early stage product. Providing the required system functionality took precedence in development until late in the project. With this functionality in place, further development could focus on implementing changes suggested by participants in the user evaluation.

The difference in feedback from participants with very little or much experience was also interesting. The more experienced participants discussed the functionality of components in detail, some even asking how certain components were developed. They were also more focused on the scenarios set, and provided constant commentary on their actions throughout the evaluation.

Novice users acted in a different way, instead focusing on the potential possibilities that the system

| PID | P.Profile | Pos. | Neg. | Tot. | Comments |
|---|---|---|---|---|---|
| 1 | Twttr. Novice, No tools exp. | 3 (60%) | 2 (40%) | 5 | Query responsiveness could be faster, overwhelming at first, captures detail comprehensively |
| 2 | Twttr. Novice, No tools exp. | 3 (50%) | 3 (50%) | 6 | Good at what it does, very busy, discovery important, lots of options, cluttered |
| 3 | No Twttr. exp., No tools exp. | 6 (85%) | 1 (15%)) | 7 | Never seen anything like it, lots of stuff you can do, unique, interesting visualisations |
| 4 | Twttr. Active, High tools exp. | 4 (66%) | 2 (34%) | 6 | No user info., simple to use, trends component advanced, interface affords clicking |
| 5 | Twttr. Active, Low tools exp. | 6 (85%) | 1 (15%) | 7 | Powerful tool for business, market research, well designed, responsive, attractive |
| 6 | No Twttr, exp., No tools exp. | 5 (83%) | 1 (17%) | 6 | Appealing, draws attention, just "have a go at it", not like anything ever used, needed user guide |
| 7 | Twttr. Inexp., No tools exp. | 5 (63%) | 3 (35%) | 8 | Distinguishable, needs consistency, maps very useful, feels "like Google", slightly cluttered |
| | | **32 (71%)** | **13 (29%)** | **45** | |

Table 7.4: Verbal Protocol Analysis

as a whole offered. Typical comments would be surprise or intrigue at features as they were introduced throughout the evaluation. This can be seen in the type of words picked by each participant group. Novice users focused on more abstract terms ('creative', 'impressive', 'fresh') whilst more experienced users honed in on particular system features they liked or disliked ('simple to use', 'powerful', 'inadequate').

### 7.4.2 Usability

A standard usability questionnaire, based on the IBM Computer User Satisfaction Questionnaire [53], was used to quantatively judge the usability of the system. Each participant was presented with 14 statements, with an associated Likert scales, and asked to rank their agreement with each statement (these statements are provided inline with the analysis below).

Due to varying user interpretation of the distances between each value on the Likert scales, and hence the relative meaninglessness of 'average' values, the aggregated scores were transformed into box plots for analysis.

The aggregated results of questions 1 through 7 are shown in Figure 7.12.

Statement 1, *"Overall, I am satisfied with how easy it was to use Pulse"* had a median value of 2, with the majority of participants ranking easy of use between 2 and 3. This implies moderate agreement with the statement, and validates both the findings of the qualitative evaluation and par-

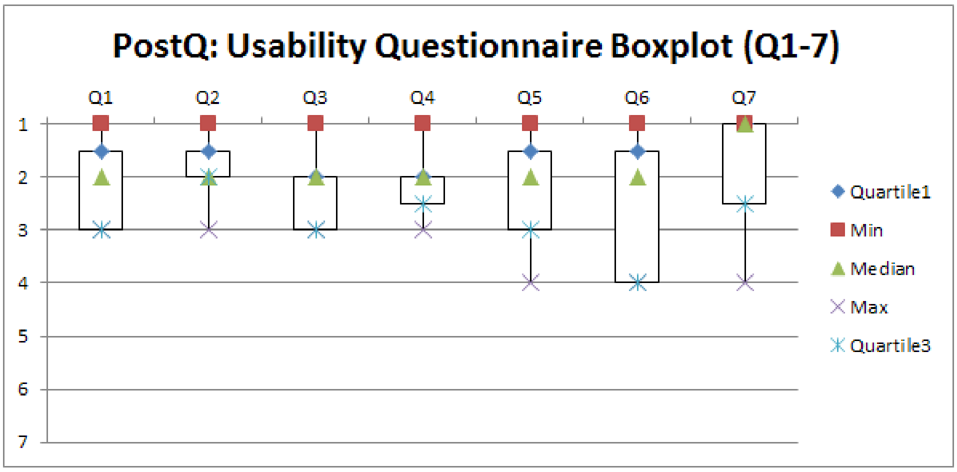


Figure 7.12: Result Boxplots - Questions 1 - 7

ticipant comments that work needs to be carried out to improve the application's display approach on lower-resolution devices.

Statement 2, *"I could effectively complete tasks set using Pulse"*, was included to measure how effectively the tasks supplied could be carried out using the system. The median response for this statement was also 2, however the majority of results lay between 1 and 2 with only 1 outlying value of 3. The participant who ranked this task 3 was also the most novice user in the participant group, mentioning that at first the component interface was confusing. Additional online help as discussed previously could be implemented to improve scores on this matric, through generally speaking participants were happy with their task completion effectiveness.

Statement 3, *"I could complete tasks set quickly using Pulse"*, measures user satisfaction with the time taken to complete each task. From observing participants the main blocker to faster task completion was simply inexperience with the system — again, further online help to hasten learning and simply more time with the application would likely improve scores for this statement. That said, the lowest level of agreement here was still a positive result of 3.

Statement 4, *"I felt comfortable using Pulse"*, was important for measuring how quickly participants felt at ease with the system. The user interface was designed to be as unintimidating as possible at startup, with the query bar inviting users to begin searching. This worked to a large extent: most users scored 2 for this statement, suggesting they were at ease operating the system.

Statement 5, *"It was easy to learn how to use Pulse"*, was expected to yield low scores given the large number of features and the limited time available within the evaluation. It is testament to the clarity of the application that most users scored this statement 2 or higher, with only the most novice users requiring verbal assistance during the evaluation. Once again, additional online help would almost certainly result in this score rising.

Statement 6, *"I believe I became productive quickly using Pulse"*, had an interesting spread of results. Whilst 4 participants ranked this 2 or higher, the remaining 3 all selected option 4 — neither agreeing not disagreeing. As a web-based application Pulse should enable users to immediately find what they require if they have a task in mind, so adding help that would improve the rank of this statement would be a future development priority. That said, Pulse does balance a fine line between immediacy and depth, and oversimplification of the system could result in users dismissing it as trivial or not fit for purpose.

Statement 7, *"The information (such as help and other documentation) provided with Pulse was clear"*, had a surprising result. Almost all users felt the documentation was appropriate (the statement had a median rank of 1), except the 2 most technically-advanced participants. This may be down to the attitudes displayed by these participants — rather than completing every walkthrough they both skipped certain ones to explore the system. This training stage was essential for other participants to fully understand the system's capabilities, but advanced users may require a simpler method that both allows freedom and educates simultaneously. A potential solution would be more elements with hover actions, explaining the function of an element, which can be quickly checked and either investigated or ignored depending on the user's usage requirements.

The aggregated results of questions 8 through 14 are shown in Figure 7.13.

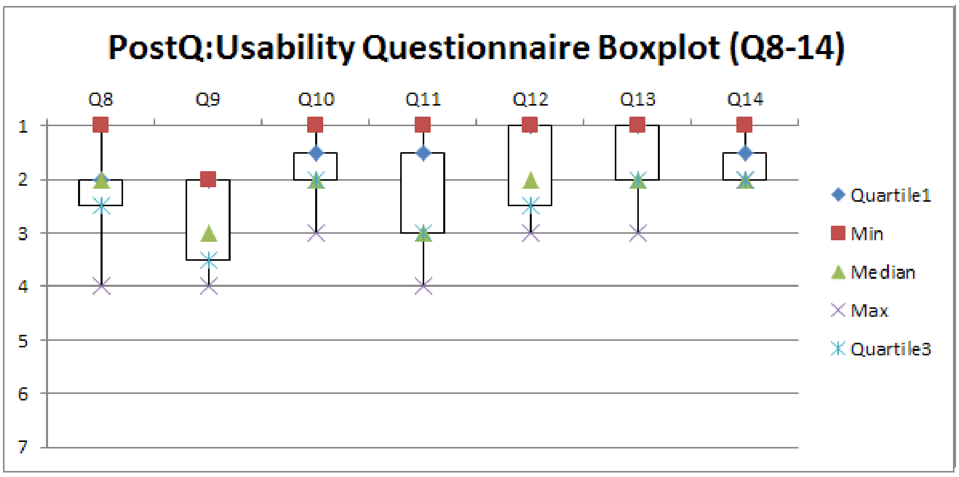


Figure 7.13: Result Boxplots - Questions 8 - 14

Statement 8, *"It was easy to find the information I needed"*, focused on whether the paths to information were clearly signposted. Participant agreement with this statement spread from 'strongly agree' to 'neither agree nor disagree', suggesting that some participants were not fully satisfied with the information finding process. Once again this is likely an effort in user education, through implicit or explicit means. Many users took only a cursory glance the user guide throughout the whole evaluation — as with many other of these statements online help would significantly aid users in information finding.

Statement 9, *The information provided by Pulse was easy to understand*, assesses the meaningfulness of results returned. Most participants expressed agreement with this statement, however verbal feedback from many sessions was that the Trends Component was initially quite daunting and not very well explained. The user guide provides a full breakdown of this component, however adding a short animation to cycle through each option, view and mode the first time the component loads would aid users in fully comprehending the analysis possible by using it correctly.

Statement 10, *The information was effective in helping me complete the set tasks*, had strong agreement from almost all participants.

Statement 11, *The organisation of information within Pulse is clear*, was important in verifying that the component-centric layout worked from a user perspective. This statement had a median value of 3 (weak agreement). Verbal feedback from participants highlighted that the lack of strong agreement wasn't about the layout of components, but more the placing of additional options within complex components (for example, the "Split Tag Cloud" option list in the Tagged With Component). Users did not expect these lists in comparison and slashtag queries as they did not appear when using single hashtag queries. This meant they were often overlooked or ignored whilst searching for information. Making this lists more prominent, either through font size or colour, would resolve this issue.

Statement 12, *The user interface of Pulse is pleasant*, was included to assess user enjoyment of the system interface. Every participant agreed with this statement, describing it as "bright", "clear", "vivid" and "professional".

Statement 13, *I liked using the interface of Pulse*, assesses how pleasant the UI was to interact with. As with statement 12 users tended towards strong agreement (median score 2). The application uses a number of interface widgets which all afford clicking, dragging, hovering and toggling — each a simple but satisfying user interaction.

Statement 14, *Overall I was satisfied with Pulse*, asks users to make a final decision on their opinion of Pulse. It is gratifying to see participants display strong agreement with this statement, and suggests success in achieving system usability.

In total, 92% of responses (90 answers) were of score 3 or higher — indicating agreement with each statement.

As illustrated by the result graphs no response higher than 4 was given for any of the 14 statements. This is likely the effect of positivity bias within each statement — the wording inclining users to either agree or, at worst, not say either way. However, the large volume of qualitative results from the desirability analysis meant weak agreement or nonchalance could be compared to words selected and comments made by the same participant. Together it was possible to infer improvements and opinions, much more so than by analysing the quantative usability data alone.

## 7.5 Conclusion

The user evaluation described in this chapter aimed to assess the usability, desirability and usefulness of Pulse with end users. An empirical evaluation method was defined in order to establish baseline participant knowledge, teach participants how to use the system, let participants explore the system, and finally understand participant reactions to the system.

The results show that Pulse has achieved each of these aims. In both parts of the post-evaluation review participants expressed positive sentiment towards the functionality, usability and desirability of Pulse. 71% of all qualitative feedback was praise for the features and design of the system, as well as over 92% of all quantative usability feedback.

It also confirmed that Pulse was of interest to users, and that it has something new to offer over and above existing tools. Furthermore, qualitative evaluation techniques showed that participants not only found the system usable, but also engaging, fun and entertaining to use.

Perhaps most crucially additional development work has been clearly identified which, if completed, would further elevate user perceptions of Pulse.

In the final chapter we conclude with a discussion of whether Pulse has achieved its aims, challenges and triumphes faced in developing Pulse, and future work that would improve functionality, usability and desirability.

# Chapter 8

# Conclusion

This final chapter provides a summary of aims achieved, and suggests future work which could be carried out to improve and extend Pulse.

## 8.1 Summary

The aim of this project was to develop a social media analysis toolkit which would allow in-depth analysis of Twitter data through an engaging user application. Accomplishing this aim involved the development of two distinct subsystems.

The first was an application to manage the process of gathering data from Twitter. In addition to streaming, parsing, writing and indexing tweets on a rigid schedule this application had to be robust and stable, able to run autonomously for long periods of time.

Several technologies were tried to determine which would best handle the storage and retrieval required, after which the Terrier information retrieval platform was selected. Every day the data processing application downloaded up to 10 million tweets, each of which was passed through a pipeline of processing stages before being indexed by Terrier.

In addition to data management Pulse also had to provide an engaging user application through which tweets could be queried, visualised and analysed. Evaluation of existing systems revealed that most focused on the immediate present, and they rarely incorporated more than one specific functionality.

Pulse is different in providing data for short-term, medium-term and long-term queries. Tweets can be observed on an hour by hour basis to identify first mention of a topic, or over the course of months to visualise usage patterns and identify trending topics. Few of the evaluated systems offer such range.

Pulse also acts as a dashboard for Twitter analysis. Components unify functionality which had previously only been available through standalone tools. Every query type results in a cohesive suite of components, each with a specific motivation and purpose. In many cases Pulse provides more features than the equivalent tool in the standalone systems evaluated — particular in discovering trends and understand global Twitter usage. In this respect Pulse is unique amongst Twitter analysis tools.

The user application is also extensible. New components can be rapidly developed owing to the highly modularised application structure, with the layered application architecture abstracting complexity between different each layer.

Having developed Pulse an empirical user evaluation was conducted with a mixed-ability participant group. The evaluation aimed to establish two things: whether the application was usable, and whether it was desirable.

Usability was evaluated through a set of formal evaluation methods. Timings to complete each task were taken, users were asked to 'think aloud' during the course of the evaluation and were issued with a usability questionnaire to rate their experience.

Desirability was assessed by encouraging participants to describe their experience using Pulse by selecting words from a wide selection of adjectives. Amongst the words chosen regularly were "useful". "engaging" and "advanced", but perhaps most satisfying for a data analysis application was the frequent use of the word "fun".

In both cases users overwhelmingly agreed that Pulse was both usable and desirable. However, as well as giving praise users were also honest in their feedback when the system failed to operate as they expected. This invaluable data is discussed further in Section 8.2.

In summary Pulse manages to achieve the aims defined on undertaking the project. A thorough analysis, development and testing process ensured Pulse contained a comprehensive suite of functionality. Several potential users validated its usability and desirability. Finally, reviewing the requirements confirms that Pulse is fit for relevant, fit for purpose and capable.

## 8.2 Future Development

The size of the Twitter domain and the virtually endless data manipulation possibilities that exist means that there is significant scope for Pulse to be extended and improved.

As it stands Pulse does not consider users — ranking, connections or usage patterns — as part of data analysis. This was a conscious decision based on the amount of time available to develop the software, instead opting to focus on comprehensiveness of tag and temporal range queries.

In future however Pulse could easily be extended to incorporate user analysis. Pulse already stores user information associated with every tweet it handles, and the modular nature of components means that defining a new query type and creating user analysis components would be a task in data analysis, not in application restructuring.

Specific component-level improvements are detailed in Chapter 5, however in terms of cross-component analysis there is much scope for improvement. Components in Pulse are currently self-contained units of functionality. The next step would be to establish how best to share processed data between components, enabling much more granular analysis. Developing a mechanism for components to disseminate then disclose the data they receive to fellow components would enable the creation of complex, manipulable data views. For example: Pulse can currently both detect sentiment of tweets and plot their location. But what is the best way to implement a feature which shows sentiment only for a specific location? Or highlights hourly popular links rather than daily links? Pulse struggles to answer these more specific questions as it stands, but with further development time cross-component integration could remedy these constraints.

Evaluation of the finished system also identified areas for improvement that would enhance the user experience. Pulse currently lacks any significant online help, and with so many components the results can initially be overwhelming. This initial mental workload could be alleviated by creating an animated walkthrough stepping users through their first query, explaining the application at a high level, then highlighting components and explaining their function.

More general future work needs to be done to improve application responsiveness with large result sets. Caching has been implemented wherever possible, but queries returning several thousand results taken unacceptably long to complete. The approach here would be twofold: enabling reverse key lookup whilst indexing, and resolving outstanding issues this is known to cause in the merging process. Without the intermediate UID lookup query stage application responsiveness would increase significantly.

# Bibliography

[1] TechCrunch. http://techcrunch.com/2006/07/15/is-twttr-interesting/, July 2006.

[2] Social Media Examiner: How Microsoft Uses Twitter To Reduce Support Costs. http://www.socialmediaexaminer.com/how-microsoft-xbox-uses-twitter-to-reduce-support-costs/, July 2010.

[3] The Retweetist News. http://retweetist.com/, December 2010.

[4] Trendistic. http://trendistic.com, December 2010.

[5] TrendsMap. http://trendsmap.com/, December 2010.

[6] TweetVolume. http://www.tweetvolume.com/, November 2010.

[7] Twitter Sentiment. http://twittersentiment.appspot.com, December 2010.

[8] Twitter.com ChloeS Status. http://twitter.com/#!/ChloeS/status/12172098017, April 2010.

[9] Twitter.com XboxSupport Stream. http://twitter.com/#!/xboxsupport, March 2010.

[10] Alexa Twitter Site Info. http://www.alexa.com/siteinfo/twitter.com, March 2011.

[11] Apache Commons Collection Library. http://commons.apache.org/collections/, March 2011.

[12] Apache HBase Home. http://hbase.apache.org/, March 2011.

[13] Apache HTTP Client. http://hc.apache.org/httpcomponents-client-ga/, March 2011.

[14] Apache Tomcat. http://tomcat.apache.org/, March 2011.

[15] Deploying on a servlet container using RPC. http://code.google.com/webtoolkit/doc/latest/DevGuideDeploying.html#DevGuideDeployingServletContainerUsingRPC, March 2011.

[16] Firebug — Web Development Evolved. http://getfirebug.com/, March 2011.

[17] GeoChirp. http://www.geochirp.com/, March 2011.

[18] Google Code Visualizations API: MotionChart. http://code.google.com/apis/visualization/documentation/gallery/motionchart.html, March 2011.

[19] Google Language API. http://code.google.com/p/gwt-google-apis/wiki/LanguageGettingStarted, March 2011.

[20] Google Web Toolkit. http://code.google.com/webtoolkit/, March 2011.

[21] GWT Anatomy of Services. http://code.google.com/webtoolkit/doc/2.1/images/AnatomyOfServices.png, March 2011.

[22] GWT Documentation: Communicating with a Server). http://code.google.com/webtoolkit/doc/1.6/DevGuideServerCommunication.html, March 2011.

[23] Java 2 Platform EE v1.3: Class HttpServlet). http://download.oracle.com/javaee/1.3/api/javax/servlet/http/HttpServlet.html, March 2011.

[24] Java Servlet 2.5 Specification. http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html, March 2011.

[25] jQuery. http://jquery.com/, March 2011.

[26] jQuery Usage Statistics. http://trends.builtwith.com/javascript/JQuery, March 2011.

[27] JSON.org. http://www.json.org, March 2011.

[28] LRUMap Javadoc, Apache Commons Collection Library. http://commons.apache.org/collections/apidocs/org/apache/commons/collections/map/LRUMap.html, March 2011.

[29] MooTools: a compact JavaScript framework. http://mootools.net/, March 2011.

[30] MozillaZine - Unresponsive Script Warning. http://kb.mozillazine.org/Unresponsive_Script_Warning, March 2011.

[31] RemoteServiceServlet (Google Web Toolkit Javadoc). http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/server/rpc/RemoteServiceServlet.html, March 2011.

[32] Skyttle: Market Sentinel. http://www.marketsentinel.com/tools/skyttle/, March 2011.

[33] Terrier 3.0 Documentation: Query Language. http://terrier.org/docs/v3.0/querylanguage.html, March 2011.

[34] Terrier 3.0 Javadoc: Collection. terrier.org/docs/v3.0/javadoc/org/terrier/indexing/Collection.html, March 2011.

[35] Terrier 3.0 Javadoc: Document. terrier.org/docs/v3.0/javadoc/org/terrier/indexing/Document.html, March 2011.

[36] Terrier 3.0 Javadoc: Interactive Querying. http://terrier.org/docs/v3.0/javadoc/index.html, March 2011.

[37] TweetMeme. http://tweetmeme.com/, March 2011.

[38] Twitter QOS Guidelines. `http://dev.twitter.com/pages/streaming_api_concepts#quality-of-service`, March 2011.

[39] Twitter Vision. `http://twittervision.com/`, March 2011.

[40] W3C CSS Homepage. `http://www.w3.org/Style/CSS/`, March 2011.

[41] What is a Servlet? `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html`, March 2011.

[42] YUI - Yahoo User Interface Library. `http://developer.yahoo.com/yui/`, March 2011.

[43] S. Baccianella, A. Esuli, and F. Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Seventh conference on International Language Resources and Evaluation, Malta. Retrieved May*, volume 25, page 2010, 2010.

[44] J. Benedek and T. Miner. Measuring Desirability: New methods for evaluating desirability in a usability lab setting. *Proceedings of Usability Professionals Association, Orlando, USA*, 2002.

[45] Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. In Bernhard Pfahringer, Geoff Holmes, and Achim Hoffmann, editors, *Discovery Science*, volume 6332 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-16184-1.

[46] W.B. Cavnar and J.M. Trenkle. N-gram-based text categorization. *Ann Arbor MI*, 48113:4001, 1994.

[47] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach (Computer Aided System Engineering)*. Addison Wesley Longman, 1994.

[48] K.A. Ericsson and H.A. Simon. *Protocol analysis*. MIT press Cambridge, MA, 1993.

[49] Alec Go, Richa Bhayani, and Lei Huang. *Twitter Sentiment Classification using Distant Supervision*, pages 1–6. 2009.

[50] D.J. Higham, P. Grindrod, and E. Estrada. Mathematics Faces up to Facebook. 2010.

[51] ISO/IEC 9241-14:1998. *Ergonomic requirements for office work with visual display terminals (VDT)s: Part 14: Menu dialogues*. ISO, Geneva, Switzerland, 1998.

[52] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[53] J.R. Lewis. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International journal of human computer interaction*, 7(1):57–78, 1995.

[54] George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38:39–41, November 1995.

[55] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[56] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[57] D. Robinson and K. Coar. The common gateway interface (CGI) version 1.1. RFC 3875, Internet Engineering Task Force, October 2004.

[58] K. Toutanova, D. Klein, C.D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[59] T. Wilson, P. Hoffmann, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff, and S. Patwardhan. OpinionFinder: A system for subjectivity analysis. In *Proceedings of HLT/EMNLP on Interactive Demonstrations*, pages 34–35. Association for Computational Linguistics, 2005.

[60] Masahide Yuasa, Keiichi Saito, and Naoki Mukawa. Emoticons convey emotions without cognition of faces: an fmri study. In *CHI '06 extended abstracts on Human factors in computing systems*, CHI EA '06, pages 1565–1570, New York, NY, USA, 2006. ACM.

## Appendix A

# User Guide

# PULSE

## User Guide

| *Version* | *Change log* |
|---|---|
| **1.0** | Fixed minor errors, added In Context component details |
| 0.1.1 | Amendments per CM recommendations: initial overview in Quick Start, branched Installation to Administrator's Guide. |
| 0.1 | Initial draft |

# Contents

# What is Pulse?

**Pulse** is a social media analysis toolkit.

More specifically, it allows for detailed analysis of the Twitter domain about any topic you might wish to investigate, across multiple time periods, in comparison to other topics.

You can compare opinions on a topic, follow tweets across the globe, find out people's attitudes to world news, see which of a set of products is more popular, observe Twitter usage volume over time, and more.

To quickly get started using **Pulse**, see the '*Quick Start*' section. To learn more about Twitter, see '*What is Twitter?*'.

***Pulse: a social media analysis toolkit***

# What is Twitter?

Twitter is a microblogging service that asks it's users: "*what's happening?*". Users have 140 characters in which to post status updates about anything they want – opinions on the world around them, conversations with other users, what was on TV last night…

Within each tweet users can specify *hashtags*. Hashtags are simply any word in a tweet prefixed with a '#'. This instantly becomes a form of inline categorisation, and you can search for tweets featuring any hashtag within **Pulse** – just prefix your query with '#'.

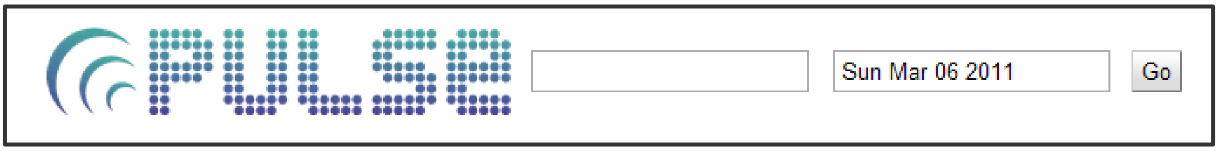
Users can also '*mention*' within a tweet. This allows users to reference another user inside their tweet by using their username and the '@' symbol.

Finally, users can broadcast another user's tweets if they like them - called '*retweeting*'. Retweets indicate some of the most popular content on Twitter as selected by users, so **Pulse** tracks the most retweeted tweets each day.

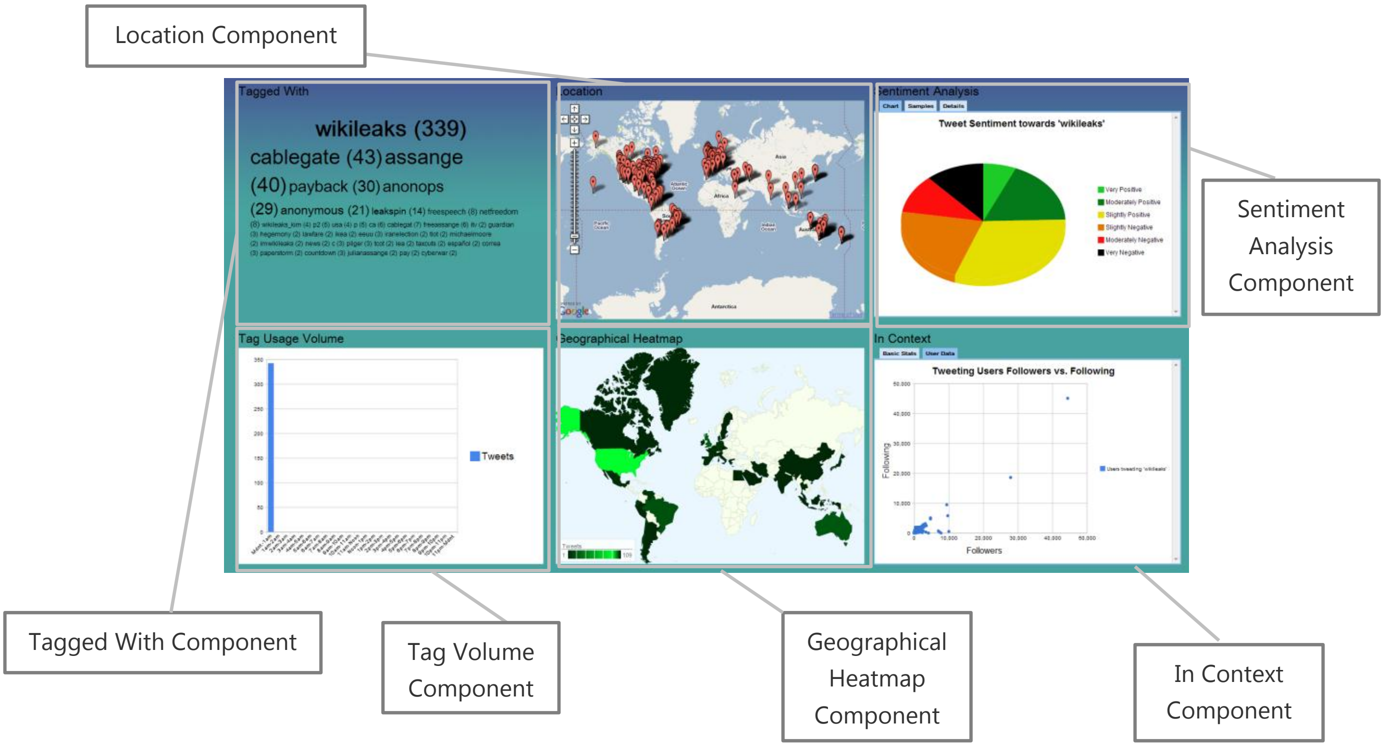
# Quick Start

Getting started is simple:

- Take a look at the Pulse Tips section for the most useful starting information.
- Give **Pulse** a *query* and click '*Go'*



Successful **hashtag queries** (e.g. '#glasgow') will result in the following:



Location Component

Sentiment Analysis Component

Tagged With Component

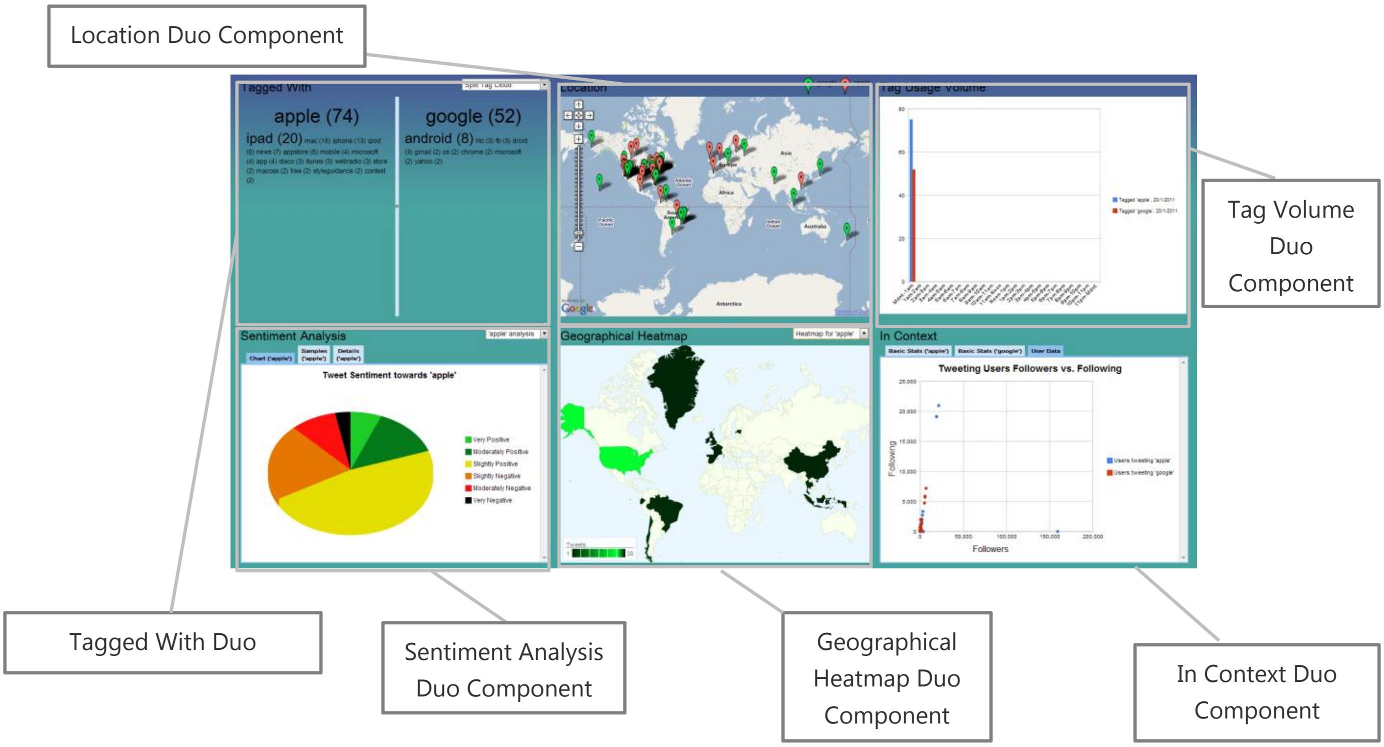
Tag Volume Component

Geographical Heatmap Component

In Context Component

- **Tagged With Component:** view tags that were used in the same tweet as the query tag, and in what volume.
- **Location Component**: see where tweets came from, and what people across the globe are saying
- **Sentiment Analysis Component**: visualises the opinions of people towards a given hashtag, with samples and statistics
- **Tag Usage Volume Component**: see how often a tag is used across the day
- **Geographical Heatmap Component**: find out which countries most tweeted a tag
- **In Context Component**: additional information about the query and the day's tweets.

Successful **slashtag queries** (e.g '#glasgow/#edinburgh) will display the following components:



Location Duo Component

Tag Volume Duo Component

Tagged With Duo

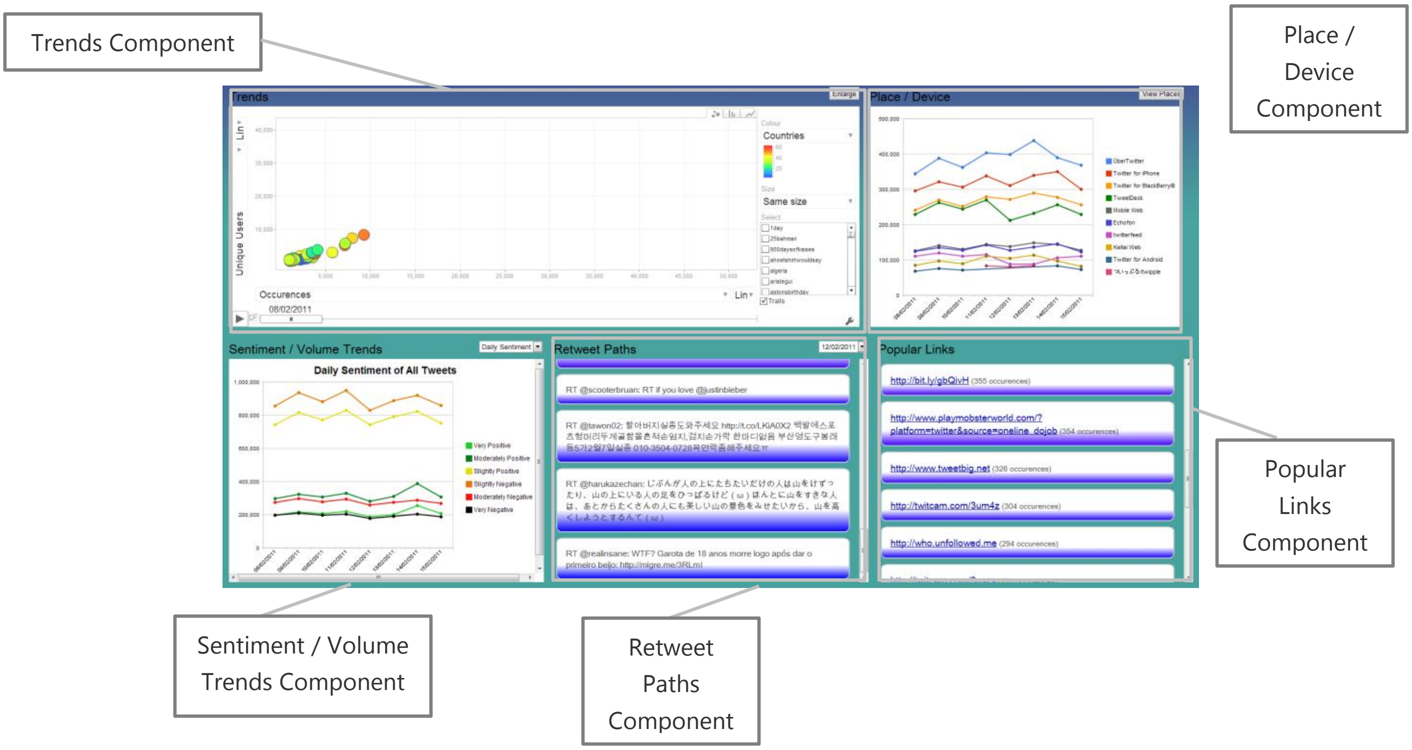
Sentiment Analysis Duo Component

Geographical Heatmap Duo Component

In Context Duo Component

Each component compares the results from both tags, and offers a number of ways to see the differences. See the Slashtag Query Components section for more detail.

Finally, **time queries** will display the following:



Trends Component

Place / Device Component

Popular Links Component

Sentiment / Volume Trends Component

Retweet Paths Component

RT @scooterbruan: RT if you love @justinbieber

RT @tawon02: 할아버지실종도와주세요 http://t.co/LKiA0X2 백발에스포츠형머리두게골함몰흔적손엄지,검지손가락 한마디없음 부산영도구봉래동5가2월7일실종 010-3504-0728꼭연락좀해주세요ㅠ

RT @harukazechan: じぶんが人の上にたちたいだけの人は山をけずったり、山の上にいる人の足をひっぱるけど(ω)ほんとに山をすきな人は、あとからたくさんの人にも美しい山の景色をみせたいから、山を高くしようとするんて(ω)

RT @realinsane: WTF? Garota de 18 anos morre logo após dar o primeiro beijo: http://migre.me/3RLml

http://bit.ly/gbQivH (355 occurences)

http://www.playmobsterworld.com/?platform=twitter&source=oneline_dojob (354 occurences)

http://www.tweetbig.net (326 occurences)

http://twitcam.com/3um4z (304 occurences)

http://who.unfollowed.me (294 occurences)

- **Trends Component**: follow trends, pivot data, zoom in on topics and points in time, and more.
- **Place / Device Component:** see how people are tweeting, and where from.
- **Sentiment / Volume Component**: find out the mood on Twitter
- **Retweet Paths Component**: follow retweets in any language as they propagate around the globe.
- **Popular Links Component**: see the most popular links posted to Twitter.

This is just a brief overview of the functionality and options within **Pulse.** The rest of this user guide explains the key features and modes of the system in greater detail.

# Pulse Tips

## Use the Search Box Efficiently

You can enter several different types of query in the main search box:

- Prefix your query with a '#' for a hashtag search
  - e.g. '#science'
- Add a slash between 2 hashtag queries to compare them simultaneously
  - e.g. '#science/#art'
- Leave the query box empty and see the top results across the whole day selected.

## Don't Search Twice!

Have you completed a search in the past whose results you'd like to see again? Select the search from the 'History' tab in the navigation bar! The results will load instantly and you can compare them with any other search. See the *'Compare Tags Panel'* for more details.

## Use Options to Narrow Search

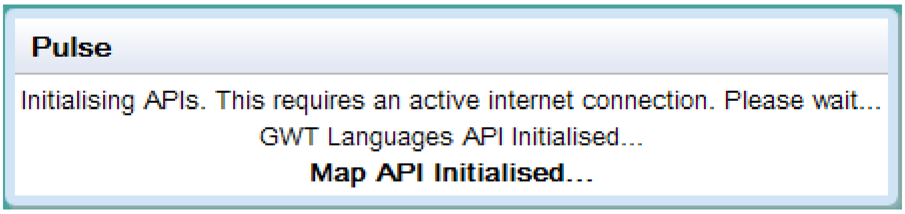
**Pulse** features many options to suit all types of queries. Don't need full map plotting? Turn it off. Want more accuracy in sentiment analysis? Use the part-of-speech tagger. Have a particular set of topics you'd like to analyse over a day? Set up a tag whitelist.

You can see all of these options within the *Options Panel* in the Navigation Bar.

# Pulse Basics

## Startup

**Pulse** requires an internet connection to start up. Before using it, please ensure you are connected to the internet. The below message will show until an internet connection is available.



Pulse

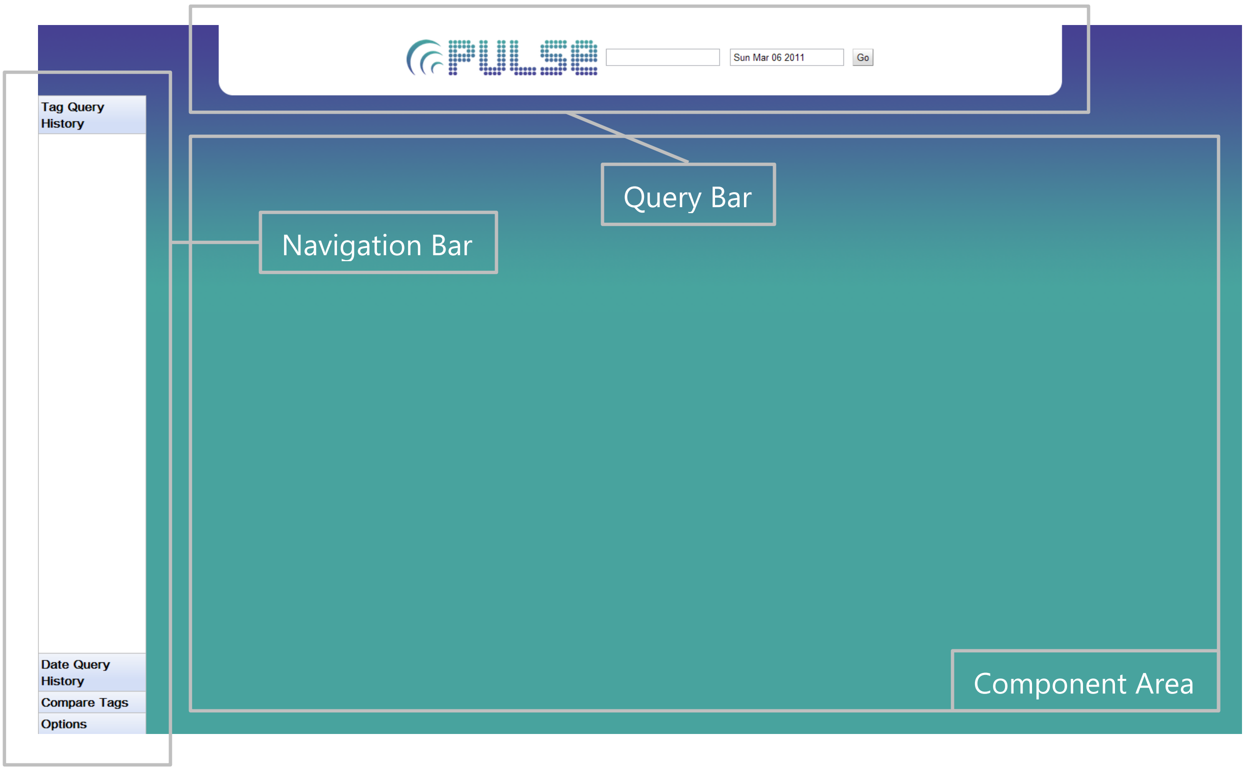
Initialising APIs. This requires an active internet connection. Please wait...

GWT Languages API Initialised...

**Map API Initialised...**

## Home

The Home screen is the first you'll see when **Pulse** has loaded. From here, you can make queries, change options, compare queries, view saved queries and more.



The **Query Bar** is where you can enter new queries to search for and select dates to search.

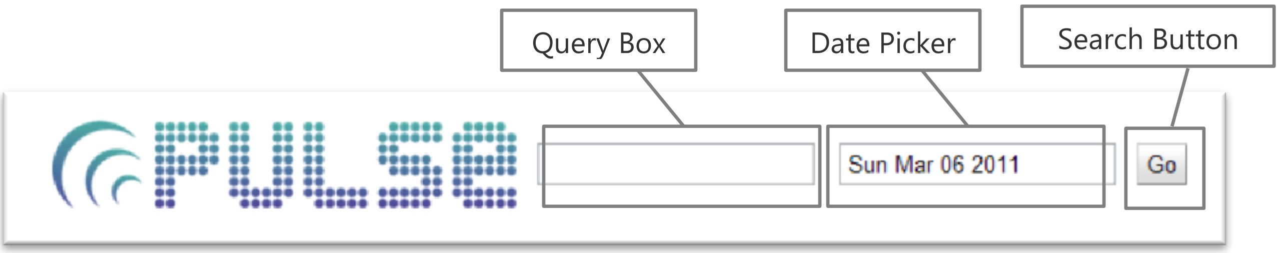
The **Navigation Bar** allows you review past queries, select queries to compare, and configure options for further queries.

The **Component Area** is where results will be displayed, in the form of components.

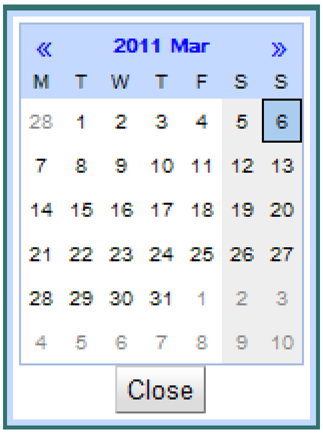
Understanding the purpose of each of these areas is vital for using **Pulse** to its fullest.

## Query Bar

The **Query Bar** is the main location querying **Pulse**.



You can enter several different types of query in the **Query Box**, each of which provides access to a different set of information about a given topic or query.



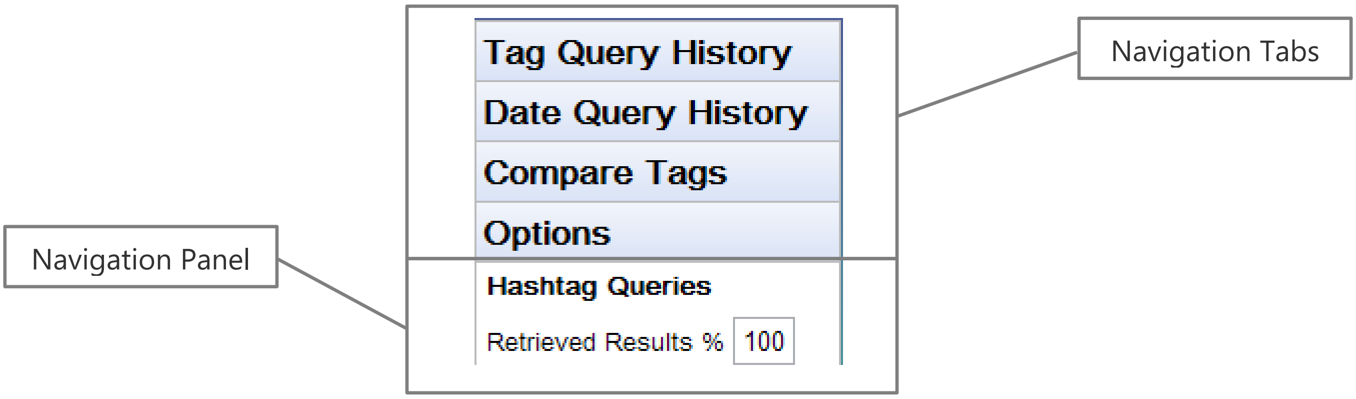
| M | T | W | T | F | S | S |
|---|---|---|---|---|---|---|
| 28 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |

You can also select a date by clicking on the **Date Picker**. This will restrict results to the date selected. Clicking the Date Picker will open the calendar view (right). You can browse months and select a particular date using this, or click '*Close*' to return to the **Pulse** Home screen.

Finally, the **Search Button** sends your query to **Pulse** for processing. Once a query has been started, you need to wait for it to finish before issuing another. The Search Button will be disabled during this time (left).

## Navigation Bar

The **Navigation Bar** provides options for queries, as well as allowing comparisons and restoring cached results.

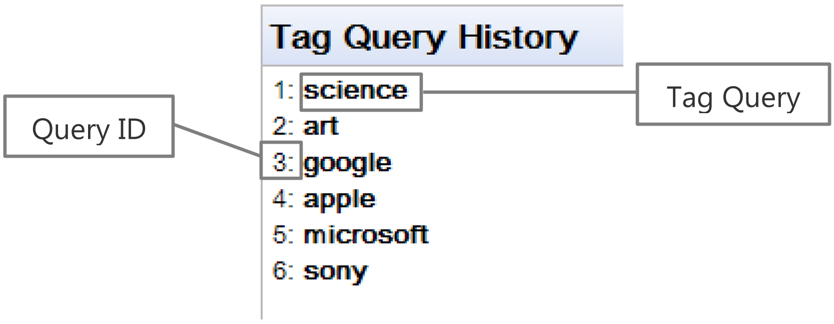


Retrieved Results % 100

Each of the **Navigation Tabs**, when clicked, opens a **Navigation Panel**.

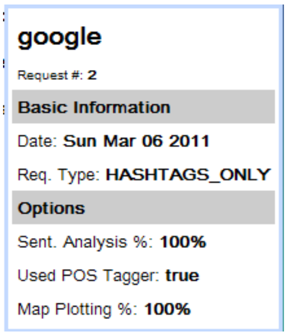
### Tag Query Navigation Panel

The Navigation Panel provides a history and cache of all hashtag queries issued.



Tag Query History

1: science
2: art
3: google
4: apple
5: microsoft
6: sony



google

Request #: 2

Basic Information

Date: Sun Mar 06 2011

Req. Type: HASHTAGS_ONLY

Options

Sent. Analysis %: 100%

Used POS Tagger: true

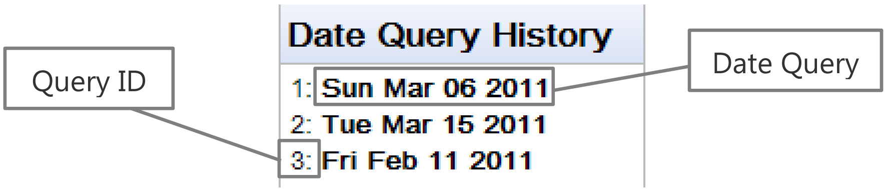
Map Plotting %: 100%

You can click on any of the queries to load that query's results back into the Component Area. This will often be considerably faster than re-querying, so utilise this to speed up usage of **Pulse.**

If you want to see the options used in calling that query, hover over the query text – a pop up will display detailing information about that query (left), such as the date searched, how many tweets were analysed, and so on. You can use this information when comparing queries to see how results differ for the same query depending on the options that were set.

### Date Query History Panel

The Navigation Panel also provides a history of all date-only queries (i.e. *time* queries; those that don't have any search keywords).

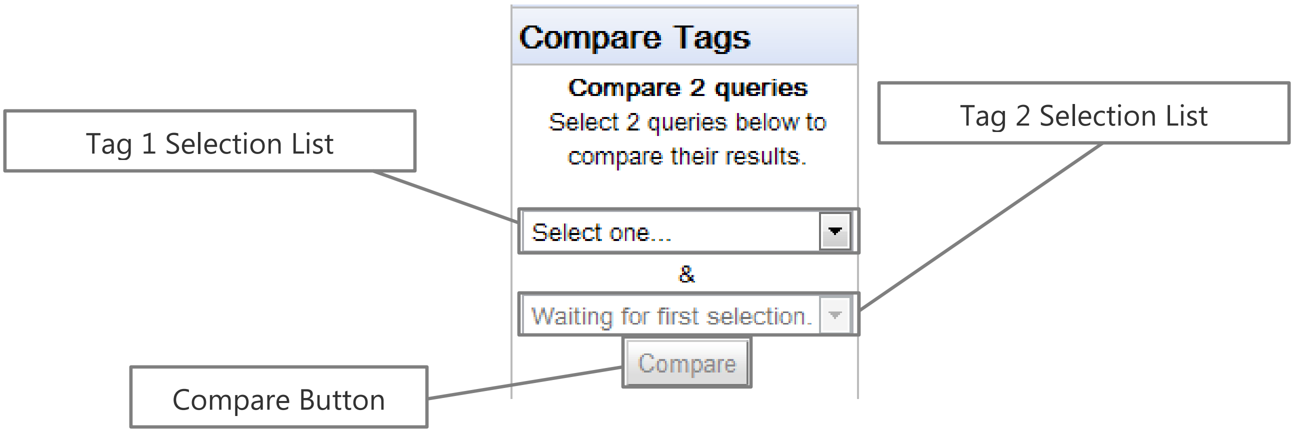


Date Query History

1: Sun Mar 06 2011
2: Tue Mar 15 2011
3: Fri Feb 11 2011

### Compare Tags Panel

Once you've queried two tags, regardless of the type of query, you can compare their results through special 'Duo' components. This can be achieved through the Compare Tags panel.



Compare Tags

Compare 2 queries

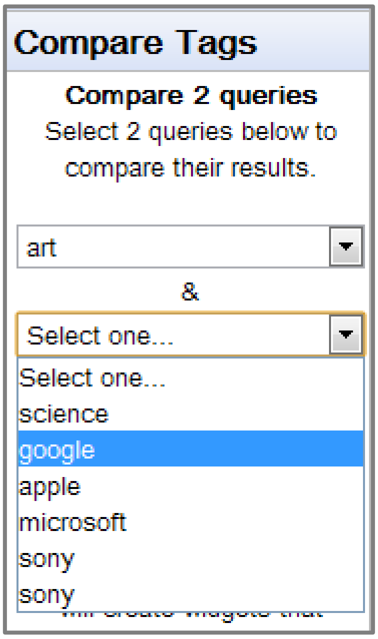
Select 2 queries below to compare their results.

Select one...

&

Waiting for first selection.

Compare



Compare Tags

Compare 2 queries

Select 2 queries below to compare their results.

art

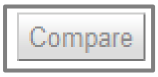
&

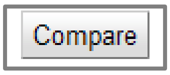
Select one...

Select one...
science
google
apple
microsoft
sony
sony

The Compare Tags panel has 2 selection lists: one for each tag to be compared. You cannot compare the same query twice (though you *can* compare the same tag in 2 different queries – see left).

Additionally, 2 queries need to have been made before you can compare. The Compare button will remain disabled (upper right) until the requirements for comparison have been met, at which point will it be enabled (lower right).



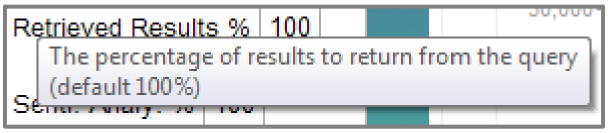


Using the Compare Tags option is analogous to a slashtag search, except all of the results are cached already (see the Pulse Tips or Querying Pulse sections for more information on slashtag searches).

## Options Panel

The Options Panel is used to set parameters for queries made within **Pulse.** Because these options apply to specific parts of the system, they are explained in detail in the relevant sections of this user guide. However, for all options:

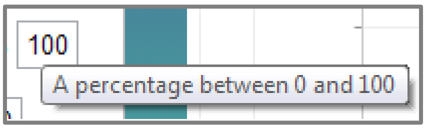
*Hover over* an option's description to see a tooltip and find out more about what it does:



Retrieved Results % 100

The percentage of results to return from the query (default 100%)

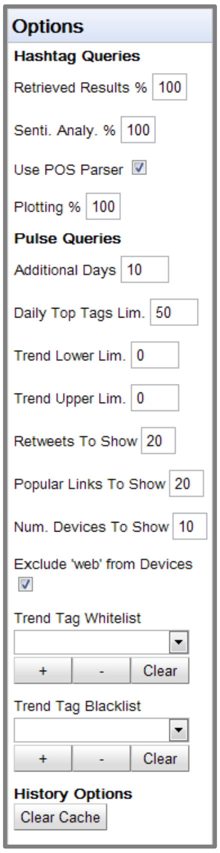
Or, hover over the option's input field to see a tooltip of typical or allowed values for that option:



100

A percentage between 0 and 100

Invalid input within options will be replaced by that option's default values.



Options

Hashtag Queries

Retrieved Results % 100

Senti. Analy. % 100

Use POS Parser ☑

Plotting % 100

Pulse Queries

Additional Days 10

Daily Top Tags Lim. 50

Trend Lower Lim. 0

Trend Upper Lim. 0

Retweets To Show 20

Popular Links To Show 20

Num. Devices To Show 10

Exclude 'web' from Devices ☑

Trend Tag Whitelist

+ - Clear

Trend Tag Blacklist

+ - Clear

History Options

Clear Cache

## Component Area

The Component Area is where all of the components generated for each query are displayed. This varies dramatically for different types of query, so the Component Area is discussed in more detail in the Querying Pulse section of this guide.

# Querying Pulse: Topics

**Pulse** provides a wealth of information, statistics and analysis of tweets over a given period, or for a particular topic (hashtag). This is presented using *components*, self-contained units which display information on a particular facet of search results. These components differ depending on the type of query issued.

## Hashtag Queries

Hashtag queries are the simplest type of queries within **Pulse**. To make a hashtag query:

1. Type a hashtag within the Query Box of the Query Bar. Make sure to prefix the query with '#'.

#winning

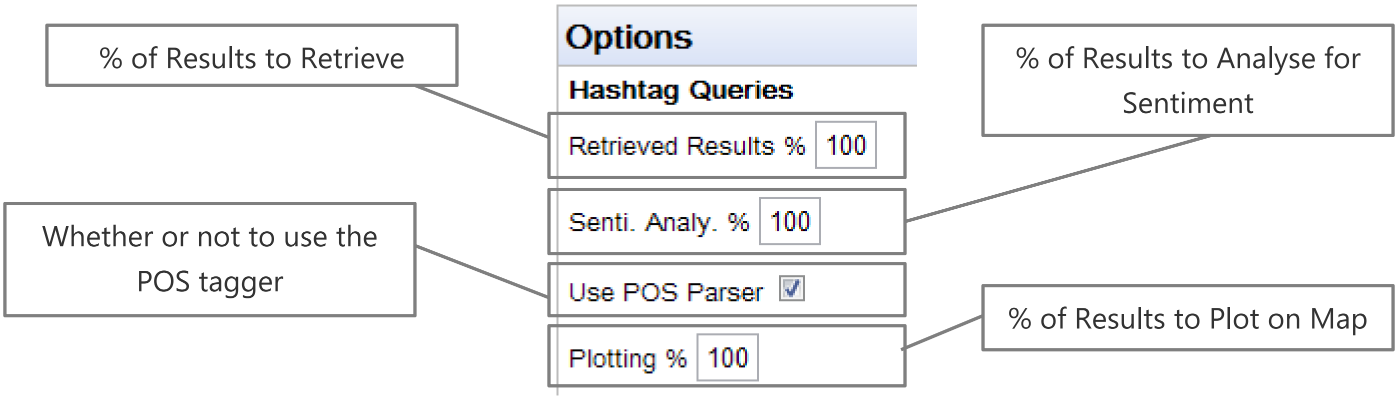
2. Select a date to query using the Date Picker. *Note: you cannot type directly into the Date Picker. Use the calendar that appears on clicking the Date Picker to select a date.*

Tue Feb 15 2011

3. (*Optional*) set options for your Query within the Options Panel of the Navigation Bar (see below).
4. Click the Search Button in the Query Bar

## Hashtag Query Options

Various options can be set for hashtag queries within the Options Panel under '*Hashtag Queries*'.



Options

Hashtag Queries

Retrieved Results % 100

Senti. Analy. % 100

Use POS Parser ☑

Plotting % 100

% of Results to Retrieve

% of Results to Analyse for Sentiment

Whether or not to use the POS tagger

% of Results to Plot on Map

1. *% of Results to Retrieve*: Set the percentage of all results you want to be returned. This can be useful for very popular tags where you don't require all of the results but rather a subset. It also results in faster processing and querying.

2. *% of Results to Analyse for Sentiment*: Set the percentage of all results to be processed by the sentiment analyser. Again, this is useful for tags with a large

number of results where only a subset is required. It results in faster processing, but obviously reduces the accuracy of sentiment analysis.

3. *Whether or not to use the POS tagger*: The POS (or part-of-speech) tagger splits each tweet's text into adjectives, verbs, nouns and adverbs. These are then analysed for sentiment according not only to their content, but the part of speech that they belong to. However, this takes a considerable amount of time.

   You can switch this off to greatly speed up sentiment analysis; however analysis will then use a combination of all entries in the sentiment analysis dictionary to determine a per-word score, rather than factoring in part of speech information.

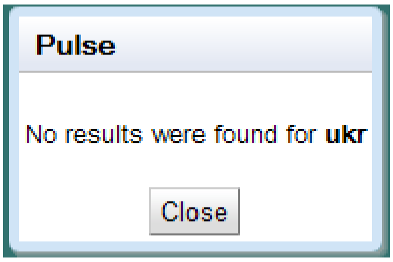
4. *% of Results to Plot on Map*: Set the percentage of all results to be plotted in the Map component. Plotting itself takes little time, but the process of geocoding each tweet (that is, establishing the location of the tweet from tweet metadata) takes time. With very large result sets, this means the geocode process will be on-going even once the initial query has been completed.

   Also, maps with a large amount of markers tend to use more system resources, so for slower machines it is advisable to reduce this.

*Note:* for all options that use percentages the number returned may vary slightly from the percentage specified. This is because a random sample of that percentage is taken. However, for anything other than a trivial number of results this will have little effect.

### Hashtag Query Results

Once the hashtag query has completed either the results or a 'no results found' error will be displayed (below).



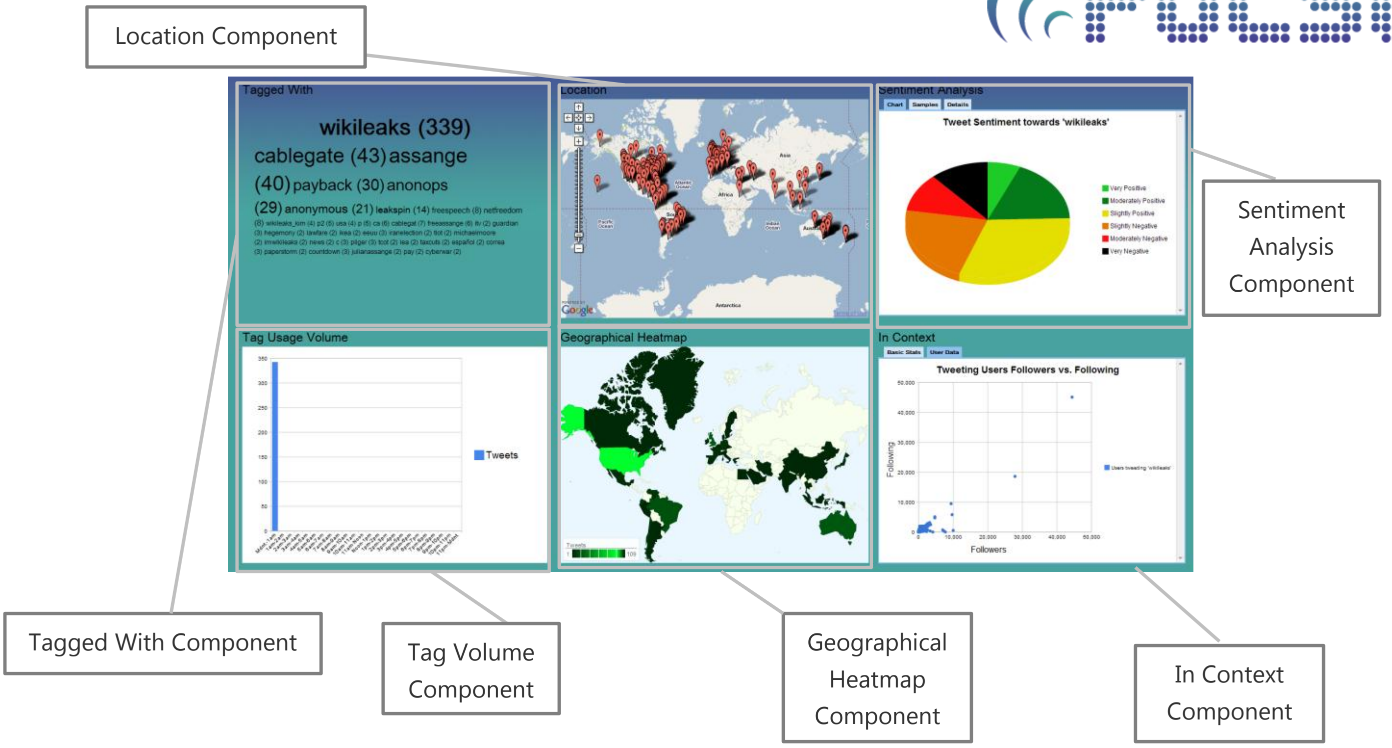
Pulse

No results were found for **ukr**

If there are any results, the component area will now contain the hashtag query result components. Additionally these results will be cached in the Tag Query Navigation Panel.

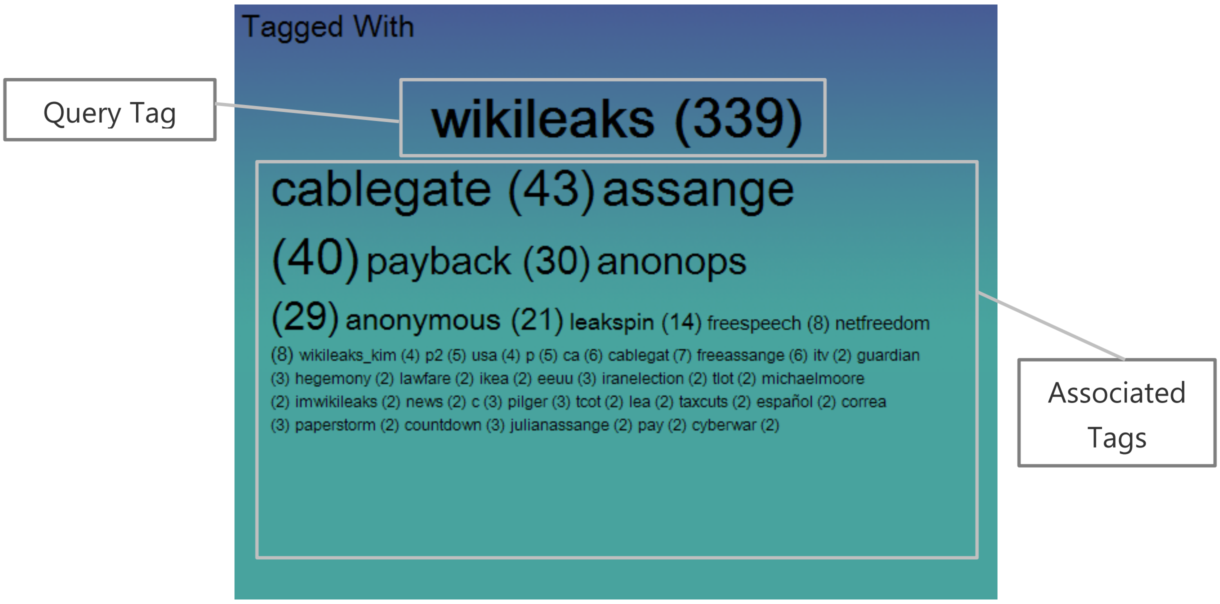
### Hashtag Query Components

Successful hashtag queries will result in six components being displayed inside the Component Area:



## Tagged With Component

The Tagged With Component displays a word cloud showing tags that were used in conjunction with the query tag.


Tagged With

wikileaks (339)

cablegate (43) assange (40) payback (30) anonops (29) anonymous (21) leakspin (14) freespeech (8) netfreedom (8) wikileaks_kim (4) p2 (5) usa (4) p (5) ca (6) cablegat (7) freeassange (6) itv (2) guardian (3) hegemony (2) lawfare (2) ikea (2) eeuu (3) iranelection (2) tlot (2) michaelmoore (2) imwikileaks (2) news (2) c (3) pilger (3) tcot (2) lea (2) taxcuts (2) español (2) correa (3) paperstorm (2) countdown (3) julianassange (2) pay (2) cyberwar (2)

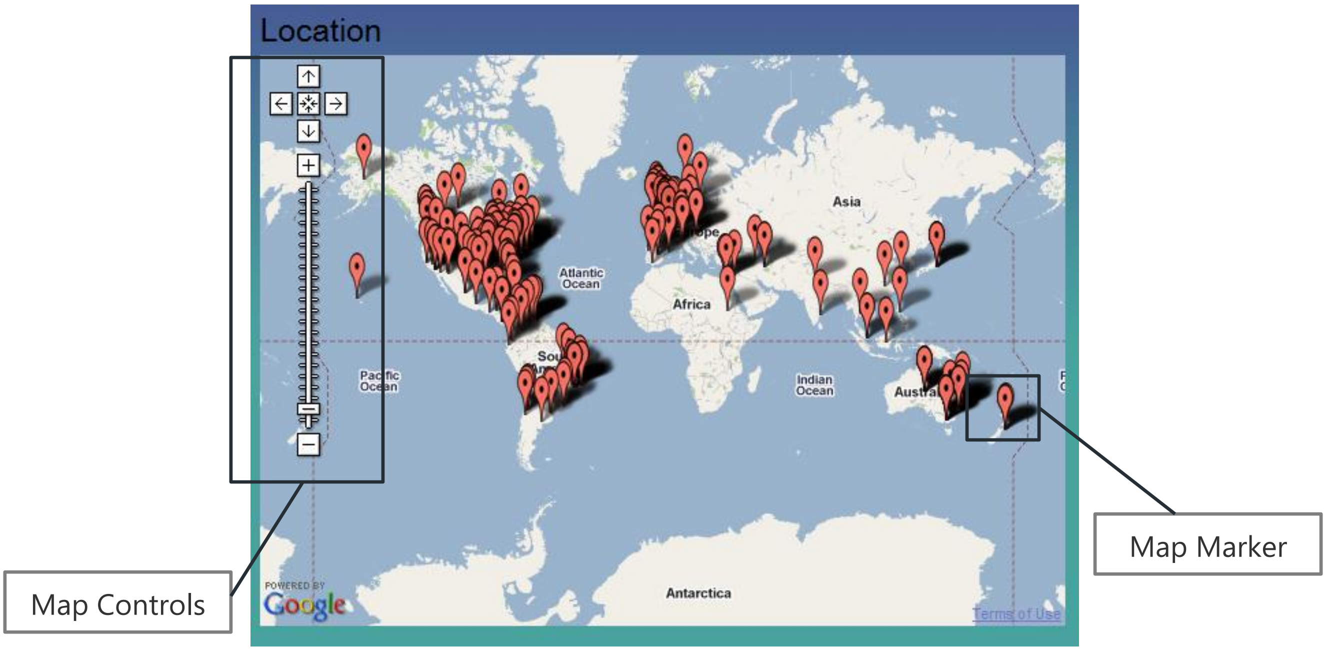
The Query Tag shows how many times that tag was mentioned in brackets. The size of all other tags is relative to the number of times they appear within the results (again, displayed in brackets).

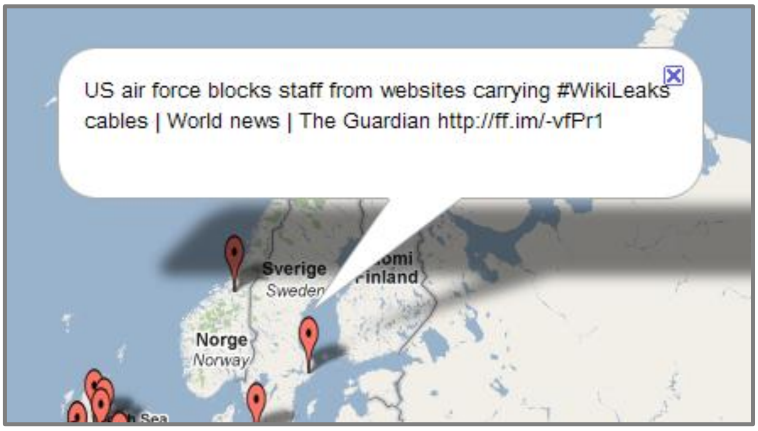
Hover over any tag to highlight it, and click the tag to automatically query for that tag.

*Note:* Tags that appear only once in conjunction with the query tag are not displayed.

## Location Component

The Location Component displays a map of markers, each corresponding to one tweet.





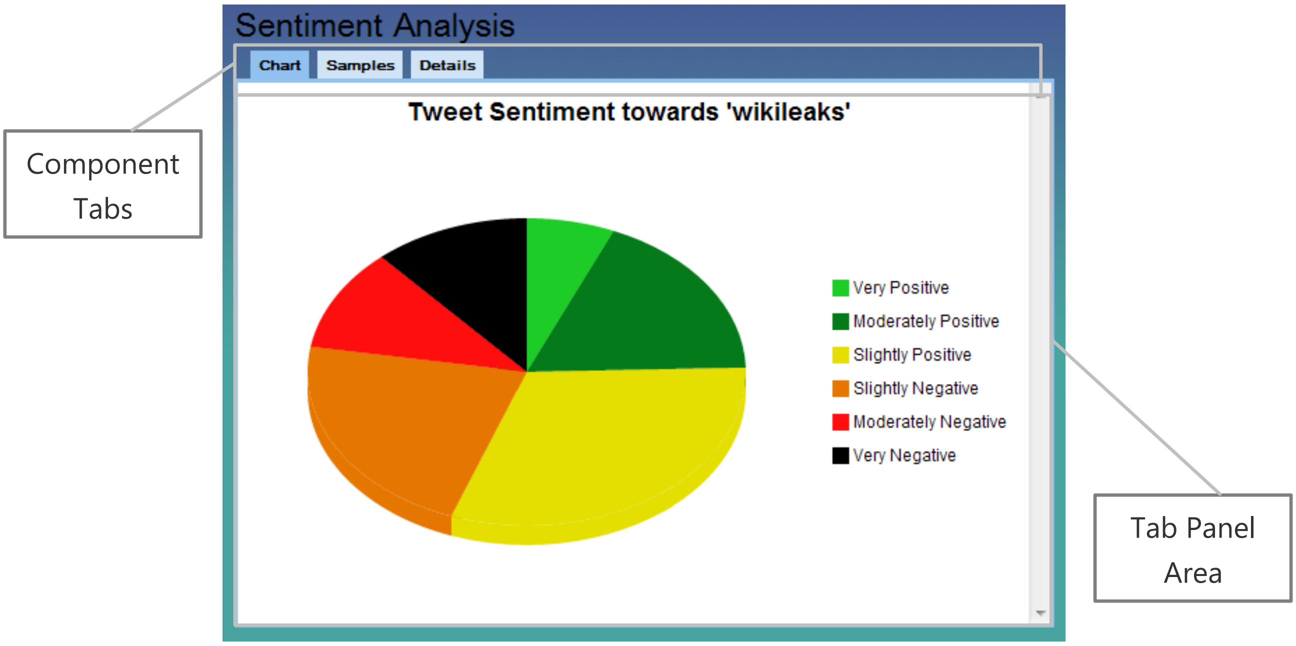
US air force blocks staff from websites carrying #WikiLeaks cables | World news | The Guardian http://ff.im/-vfPr1

The Map Controls can be used to pan and zoom into / out of the map (as well as the mouse scroll wheel or by dragging the map). Each marker can also be clicked, revealing the content of the tweet (left). Click the cross to close the tweet text information window.

## Sentiment Analysis Component

The Sentiment Analysis Component processes the text content of each tweet, and tries to establish the sentiment of that text.

This component displays inside a *tab panel*. Clicking any of the tabs along the top of the panel will change the content. In this component, you can view a pie chart summarising overall sentiment, review some samples and their individual sentiment, or view numerical details of how the chart was created.



Component Tabs

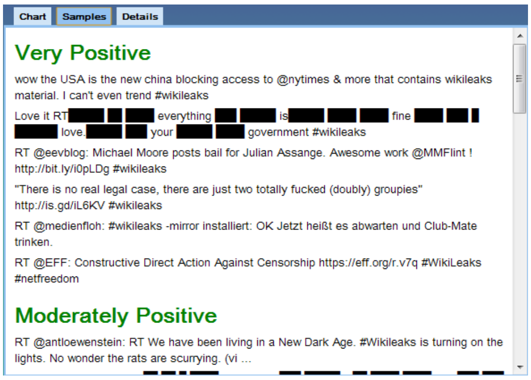
Tab Panel Area

### *Chart Tab*

The Chart Tab shows a pie chart of overall sentiment, divided into 6 categories: very positive, moderately positive, slightly positive, slightly negative, moderately negative and very negative. Each tweet is assigned a score, and it is this score that determines the level of sentiment.

Clicking on a segment of the chart will display an information window with further details about how many tweets were categorised in that band, and what percentage that was of all tweets.

### *Samples Tab*



Very Positive

wow the USA is the new china blocking access to @nytimes & more that contains wikileaks material. I can't even trend #wikileaks

Love it RT[redacted] [redacted] [redacted] everything [redacted] [redacted] is[redacted] [redacted] [redacted] fine [redacted] [redacted] [redacted] [redacted] love.[redacted] [redacted] your [redacted] [redacted] government #wikileaks

RT @eevblog: Michael Moore posts bail for Julian Assange. Awesome work @MMFlint ! http://bit.ly/i0pLDg #wikileaks

"There is no real legal case, there are just two totally fucked (doubly) groupies" http://is.gd/iL6KV #wikileaks

RT @medienfloh: #wikileaks -mirror installiert: OK Jetzt heißt es abwarten und Club-Mate trinken.

RT @EFF: Constructive Direct Action Against Censorship https://eff.org/r.v7q #WikiLeaks #netfreedom

Moderately Positive

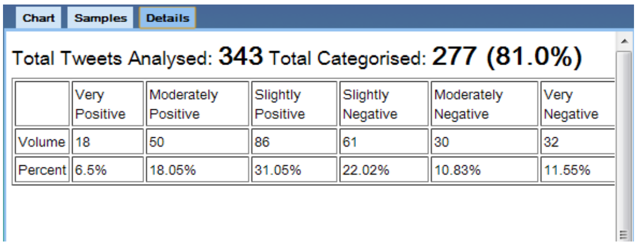
RT @antloewenstein: RT We have been living in a New Dark Age. #Wikileaks is turning on the lights. No wonder the rats are scurrying. (vi ...

The Samples Tab shows a small sample of tweets and the category they were placed in.

Use the scroll bar on the right hand side to view all samples, from 'very positive' to 'very negative'.

### *Details Tab*

The Details Tab shows exact numbers of tweets analysed, the relative rankings of those tweets, and how successful the analysis was.

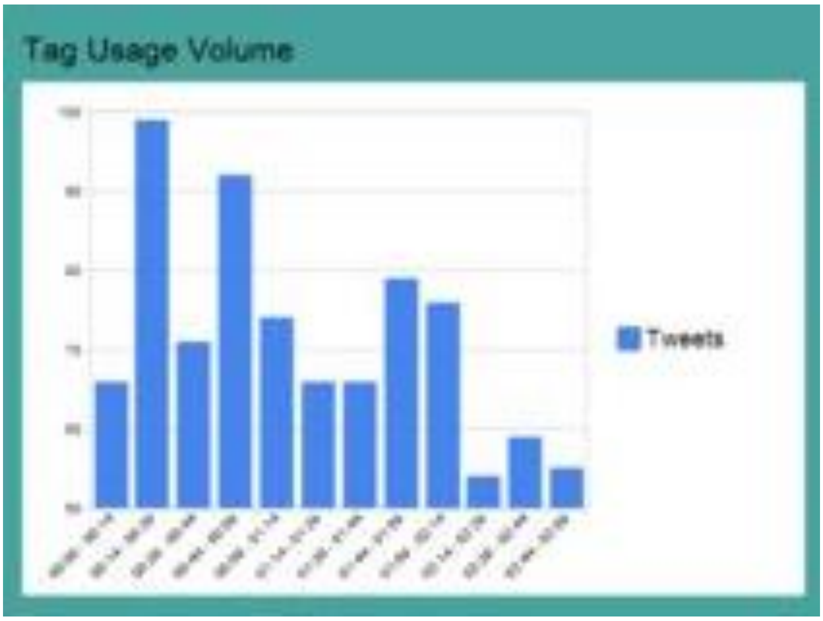


Total Tweets Analysed: 343 Total Categorised: 277 (81.0%)

| | Very Positive | Moderately Positive | Slightly Positive | Slightly Negative | Moderately Negative | Very Negative |
|---|---|---|---|---|---|---|
| Volume | 18 | 50 | 86 | 61 | 30 | 32 |
| Percent | 6.5% | 18.05% | 31.05% | 22.02% | 10.83% | 11.55% |

## Tag Usage Volume Component

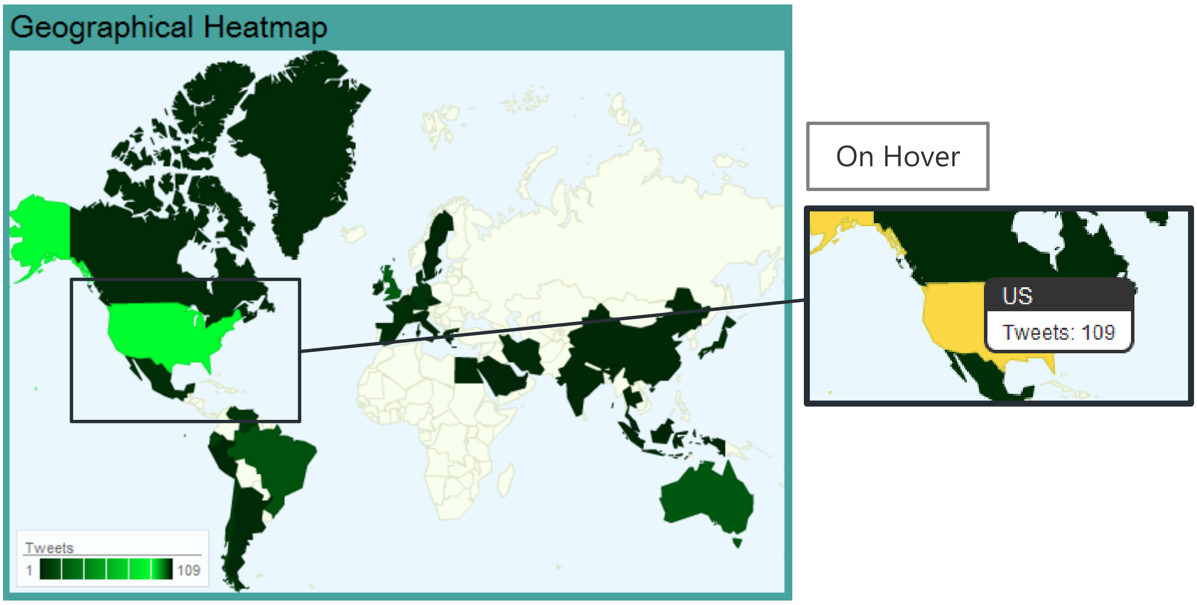
The Tag Usage Volume Component charts the usage of the queried tag over the course of the query day, divided into 24 hours.



Each of the bars can clicked for more information about that particular hour (exact number of tweets, etc.).

## Geographical Heatmap Component

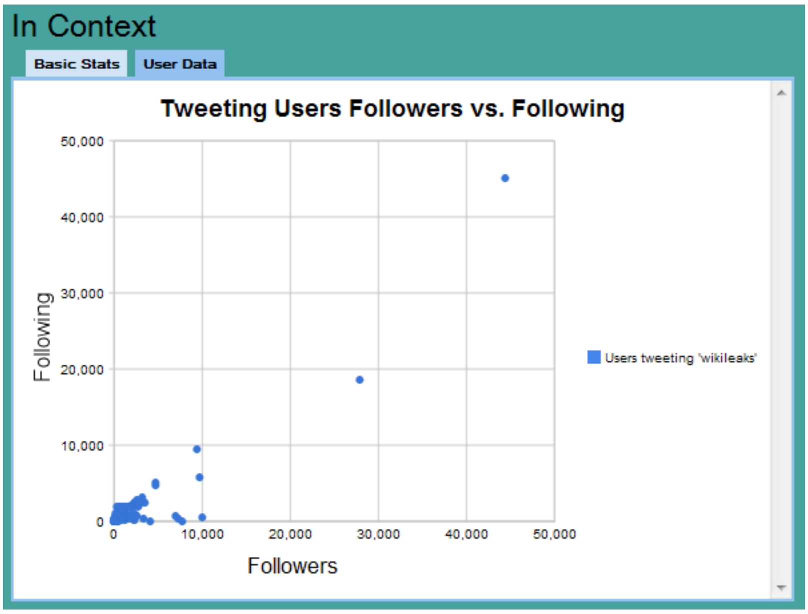
The Geographical Heatmap Component shows the usage of a particular term by country, with the relative intensity of the colour indicating the level of usage.



The scale in the bottom left indicates usage popularity. However you can also hover over any country to see an exact number (see top right).

### In Context Component

The In Context Component displays additional ancillary information about the query hashtag.



This component, as with the Sentiment Analysis Component, utilises a tab panel.

#### *User Data*

The User Data tab provides a scatter graph with details about those users who tweeted the query hashtag. This can be used to establish whether a tag is being tweeted casually by many 'average users' (low follower / following count), or often by follower-heavy users (often news organisations, celebrities, businesses etc.).

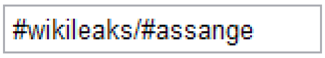
#### *Basic Stats*

This section of the component provides basic information about the frequency of the query hashtag, and the size of the collection on which the query was run. This component is a work in progress, in line with **Pulse** initial release status.

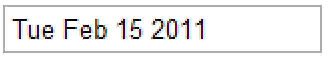
### Slashtag Queries

Slashtag queries in **Pulse** are used to compare two hashtags within specialised 'Duo' versions of the hashtag query components. These provide additional comparison functionality so that the results of two queries can be compared. To make a slashtag query:

1. Type a hashtag within the Query Box of the Query Bar, followed by a slash, followed by another hashtag.


#wikileaks/#assange

2. Select a date to query using the Date Picker. *Note: you cannot type directly into the Date Picker. Use the calendar that appears on clicking the Date Picker to select a date.*


Tue Feb 15 2011

3. (*Optional*) Set options for your Query within the Options Panel of the Navigation Bar (see below).
4. Click the Search Button in the Query Bar

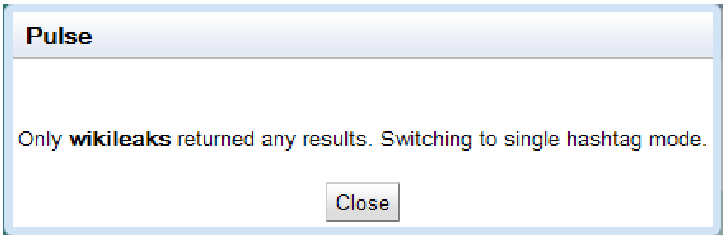
### Slashtag Query Options

Various options can be set for slashtag queries within the Options Panel under '*Hashtag Queries*'. These will apply to both hashtags specified in the slashtag query. See 'Hashtag Query Options' for specific details of how these work.

### Slashtag Query Results

Once the slashtag query has completed there are three possible outcomes:

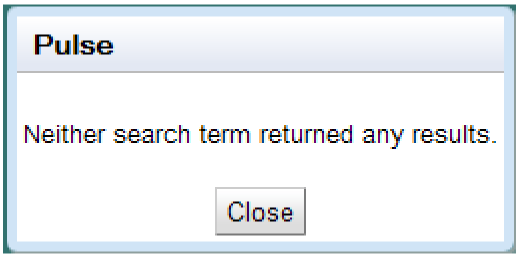
1. Both queries return results. These will be displayed within the component area as described below.
2. Only one query returns any results. In this case, an error message will be shown, and the tag that did have results will be displayed as if it were a hashtag query.



Pulse

Only **wikileaks** returned any results. Switching to single hashtag mode.

3. Neither hashtag returns any results. In this case, an error message will be shown.



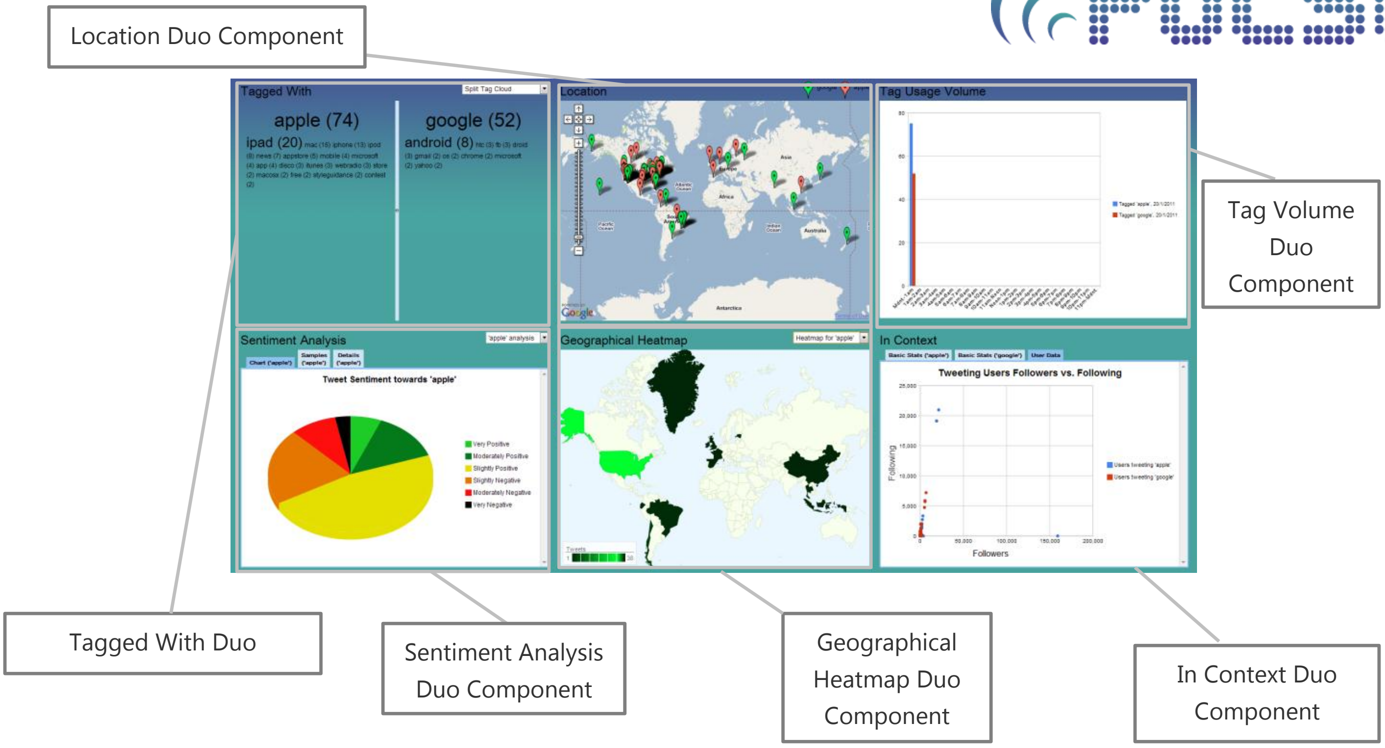
Pulse

Neither search term returned any results.

### Slashtag Query Components

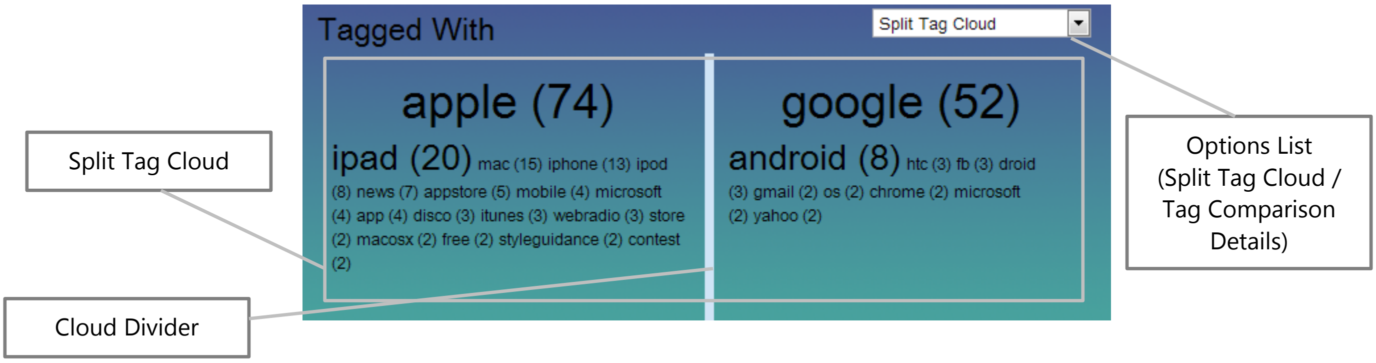
Successful slashtag queries will result in six 'Duo' components being displayed inside the Component Area, as well as both queries being cached.

*Note*: these components will also be used when comparing two existing queries from the query cache, should both queries have results.



## Tagged With Duo Component

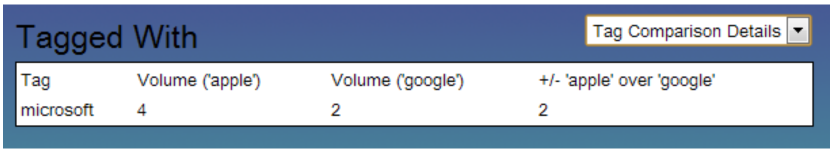
The Tagged With Duo Component is based on the Tagged With Component, with two notable differences – the split tag cloud, and the tag comparison details.



apple (74)
ipad (20) mac (15) iphone (13) ipod (8) news (7) appstore (5) mobile (4) microsoft (4) app (4) disco (3) itunes (3) webradio (3) store (2) macosx (2) free (2) styleguidance (2) contest (2)

google (52)
android (8) htc (3) fb (3) droid (3) gmail (2) os (2) chrome (2) microsoft (2) yahoo (2)

The default view of this component shows each tag cloud with a divider in between. This divider can be moved to focus on one or the other set of tags. Functionality is as the standard Tagged With Component, with the addition of the Tag Comparison option. To view this, select ‘Tag Comparison’ from the Options List.

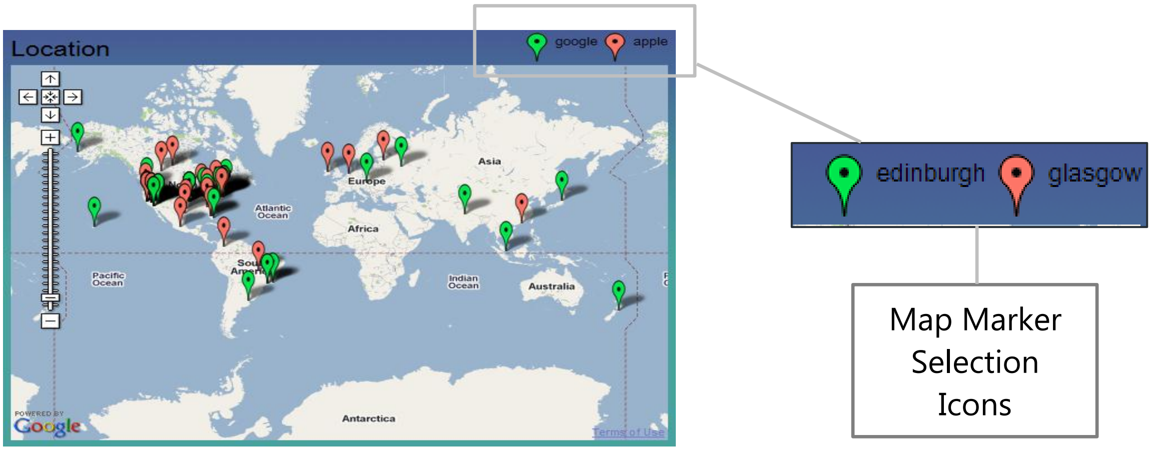


| Tag | Volume ('apple') | Volume ('google') | +/- 'apple' over 'google' |
|---|---|---|---|
| microsoft | 4 | 2 | 2 |

The Tag Comparison view (above) shows all the tags that each query has in common, and the difference in volume between each query.

### Location Duo Component

Based on the Location Component, this Duo version displays different coloured markers for each query.



Map Marker Selection Icons

You can also choose to switch off markers for one or both locations by clicking the marker icon corresponding to the query hashtag in the component title bar (above right).

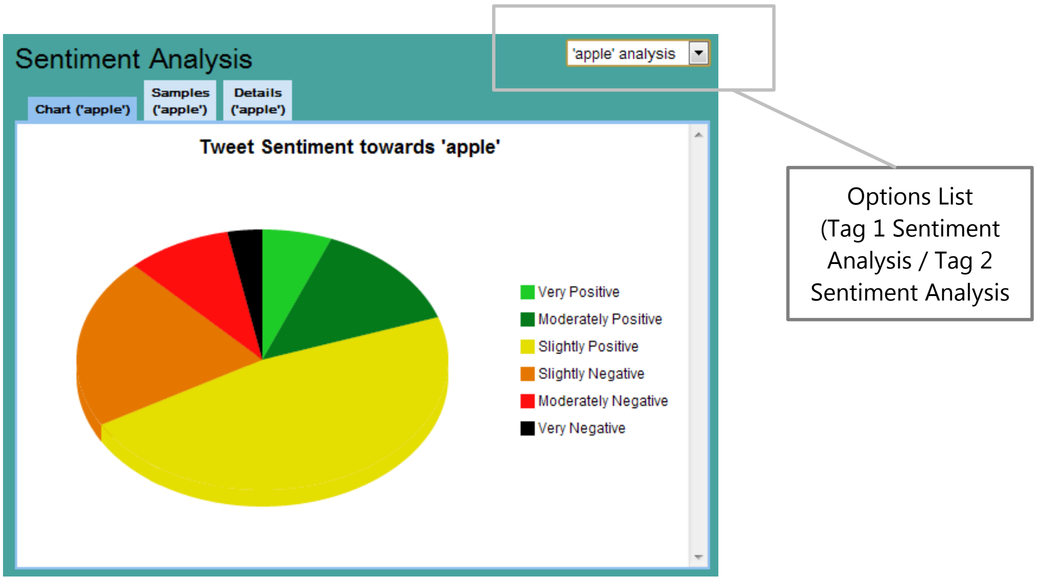
*Note*: for queries with a large number of results it is recommended that the Plotting % option is reduced.

### Tag Volume Duo Component

The Duo version of this component shows the volume of each query every hour side by side as bars in a bar chart, in the same style as the Tag Volume component.

### Sentiment Analysis Duo Component

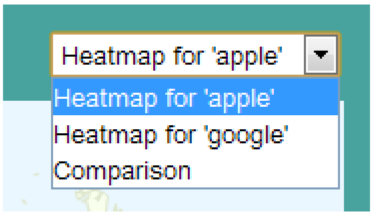
The Sentiment Analysis Duo Component presents the results of sentiment analysis across two tab panels. These can be changed using the options list in the upper right corner of the component.


Sentiment Analysis

'apple' analysis

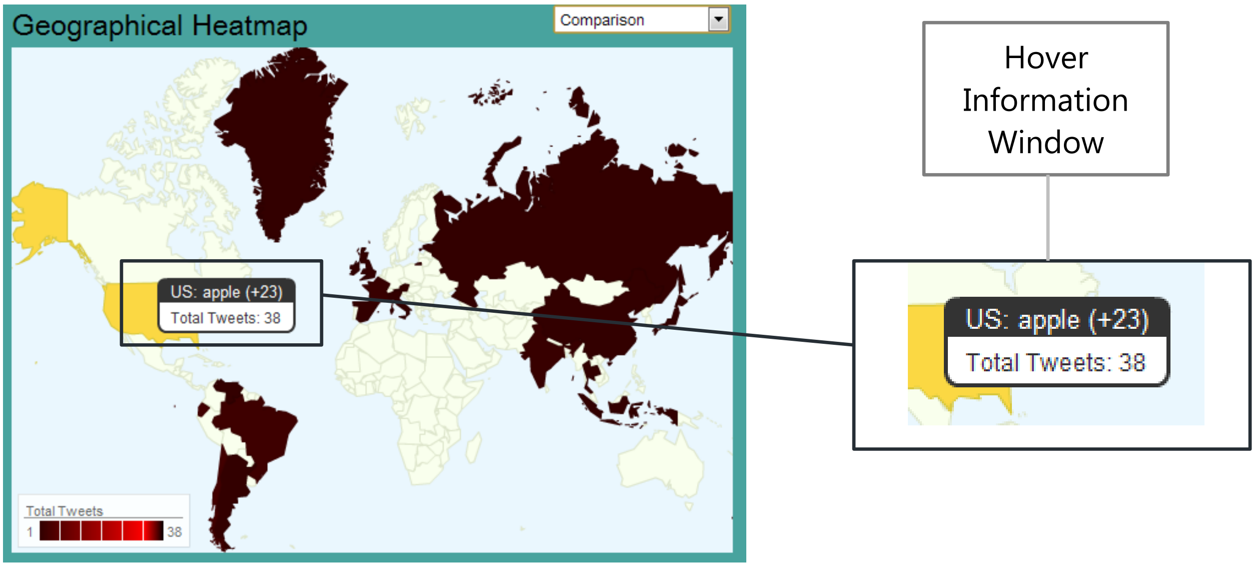
Options List (Tag 1 Sentiment Analysis / Tag 2 Sentiment Analysis

## Geographical Heatmap Duo Component



The Duo version of the Geographical Heatmap Duo Component provides maps for each query, as well as a comparison map that shows the most popular tag per country. You can select the map to view using the Options List (right).

The comparison map (below) highlights the most popular tag per country. By hovering on a particular country a window appears with further details: which tag was more popular, and by how much. It also shows the total number of tweets for both tags in that country.



## In Context Duo Component

Like the Tag Volume Duo Component, this component shows both queries scattergraph points as different colours for comparison. Additionally there are two tabs of basic statistics, one for each tag.

## Comparing Multiple Queries

Any hashtag or slashtag query can be compared against any other in the Compare Panel of the Navigation Bar. For example, searching '#apple/#google', then '#ps3/#xbox', will still allow '#google/#xbox' to be compared from cache – each slashtag query result is stored independently of the other tag.

# Querying Pulse: Time

As well as querying **Pulse** using topic based queries, you can also search solely based on time periods.

### Time Queries

Time queries present results such as trends, popular terms or device usage from across a day, week or even month. To make a time query:

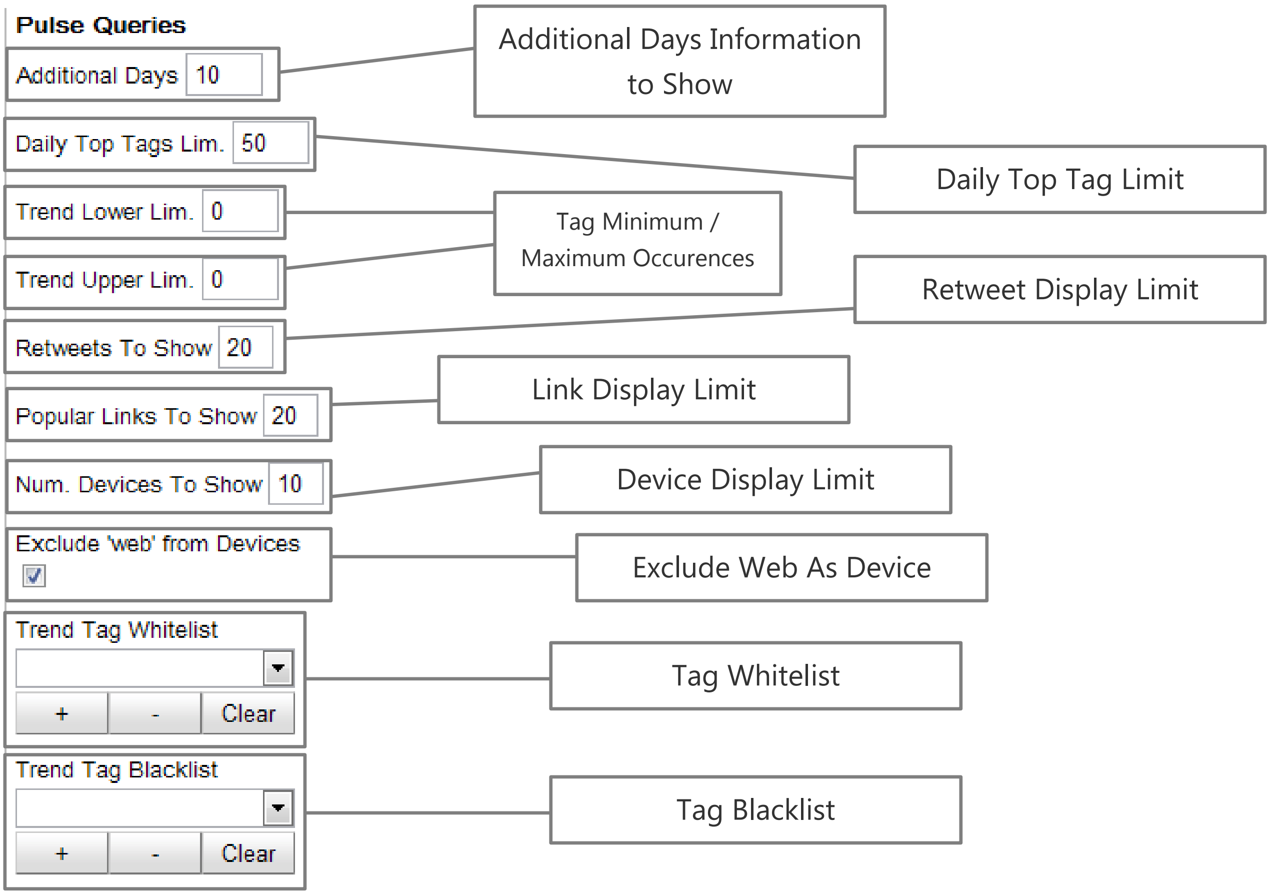
1. Clear the Query Box in the Query Bar. This signifies a time search, rather than a topic search of a given time.
2. Select a date to query using the Date Picker. *Note: you cannot type directly into the Date Picker. Use the calendar that appears on clicking the Date Picker to select a date.*

Tue Feb 15 2011

3. (*Optional*) set options for your Query within the Options Panel of the Navigation Bar (see below
4. Click the Search Button in the Query Bar

### Time Query Options

Various options can be set for time queries within the Options Panel under '*Pulse Queries*'.



Pulse Queries

Additional Days 10 — Additional Days Information to Show

Daily Top Tags Lim. 50 — Daily Top Tag Limit

Trend Lower Lim. 0 — Tag Minimum / Maximum Occurences

Trend Upper Lim. 0 — Tag Minimum / Maximum Occurences

Retweets To Show 20 — Retweet Display Limit

Popular Links To Show 20 — Link Display Limit

Num. Devices To Show 10 — Device Display Limit

Exclude 'web' from Devices ☑ — Exclude Web As Device

Trend Tag Whitelist — Tag Whitelist

Trend Tag Blacklist — Tag Blacklist

- *Additional Days Information to Show*: the number of days either side the queried date to fetch results from. For example, selecting '5', then searching for '15th March 2011', will return results from 10th March 2011 to 20th March 2011. The maximum is 30.

- *Daily Top Tag Limit:* **Pulse** Time Queries will display the top 50 tags for a day within the results by default. Modify this setting to change this.

- *Tag Minimum / Maximum Occurences:* Set boundaries for the number of occurences a tag must have to be displayed. Setting a high 'lower limit' will result in only the most popular tags appearing. Similarly, setting a low 'higher limit' will show only the less popular tags.

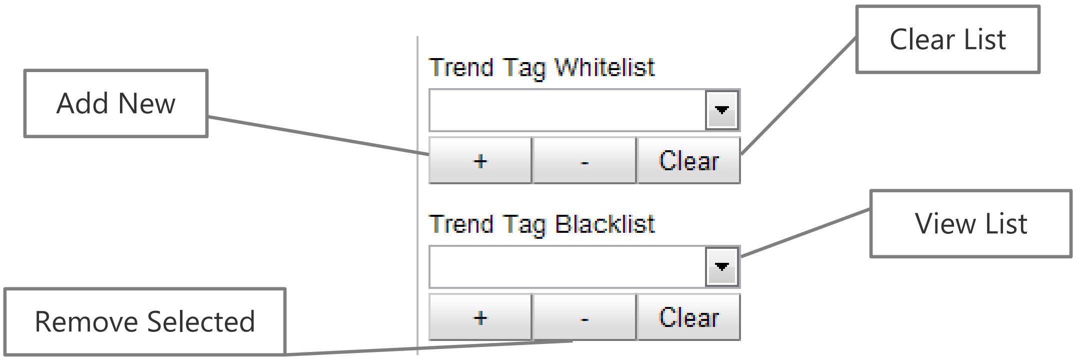
- *Retweet Display Limit:* Set the number of retweets to display. Generally only the top 20 retweets are stored for any given day.

- *Link Display Limit:* Set the number of links to display.

- *Device Display Limit:* Set the number of devices to show in the Places / Devices Component. A higher limit will show more devices, however on low resolution monitors the line graph will quickly become crowded. Default is 10

- *Exclude Web As Device:* 'Web' is by far and away the most popular way to tweet on Twitter. To see Place / Device results which exclude 'Web' as a device, select this checkbox.

- *Trend Tag Whitelist / Blacklist:* The tag whitelist and blacklists allows specification of which tags to *only* show, or specifically which tags *not* to show, from the day's top tags. Any tag not on the whitelist will be ignored, or similarly any tag on the blacklist will be ignored.



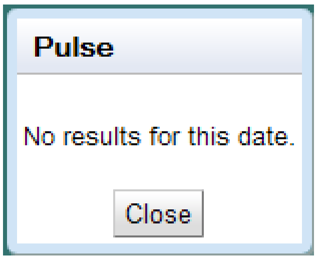
To add new items, click the '+' button. A text box will appear below: type in either 1 tag, or a comma-separated list of tags, then press 'Enter' on the keyboard. These tags will now be visible in the list.

*Note:* Only one of the two lists may be used at any time.

*Note:* for all options that use percentages the number returned may vary slightly from the percentage specified. This is because a random sample of that percentage is taken. However, for anything other than a trivial number of results this will have little effect.

### Time Query Results

Once the time query has completed either the results or a 'no results found' error will be displayed (below).



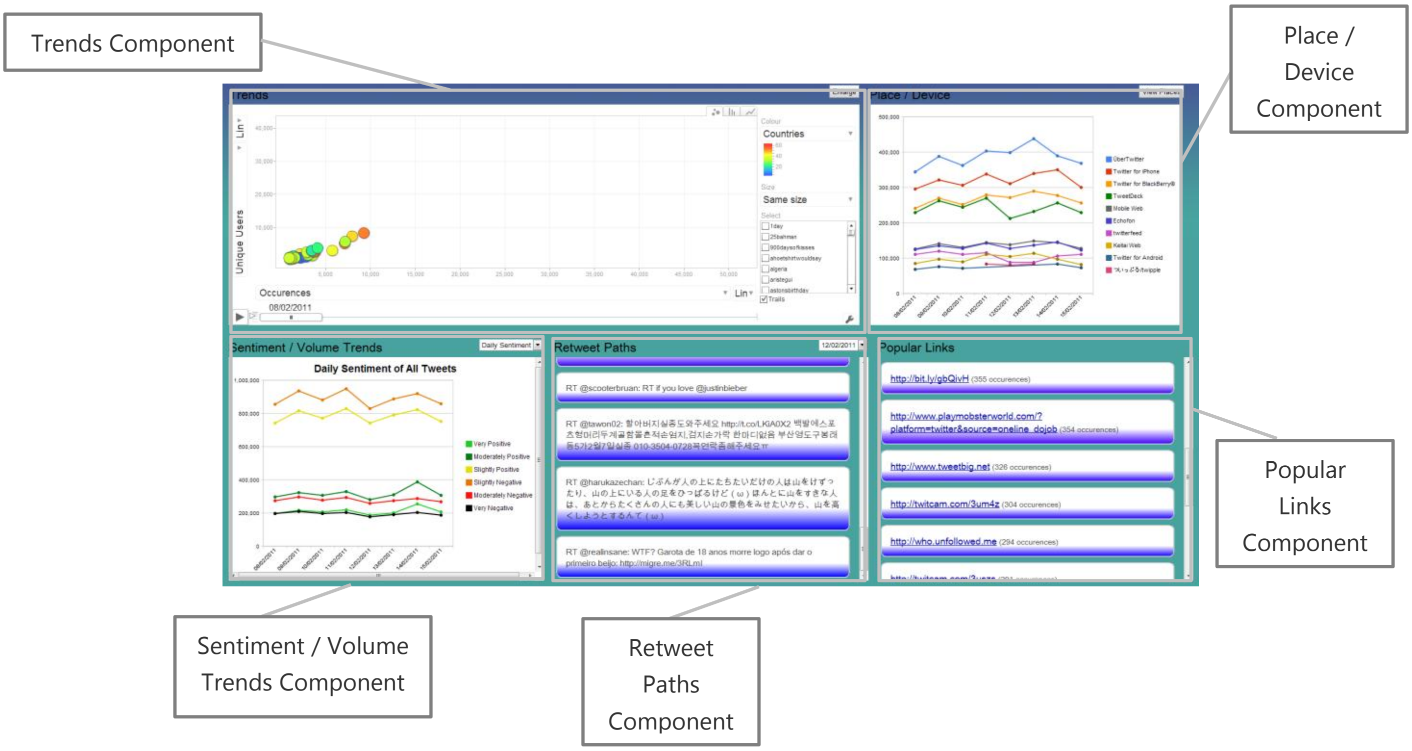
Pulse

No results for this date.

If there are any results, the component area will now contain time query result components.

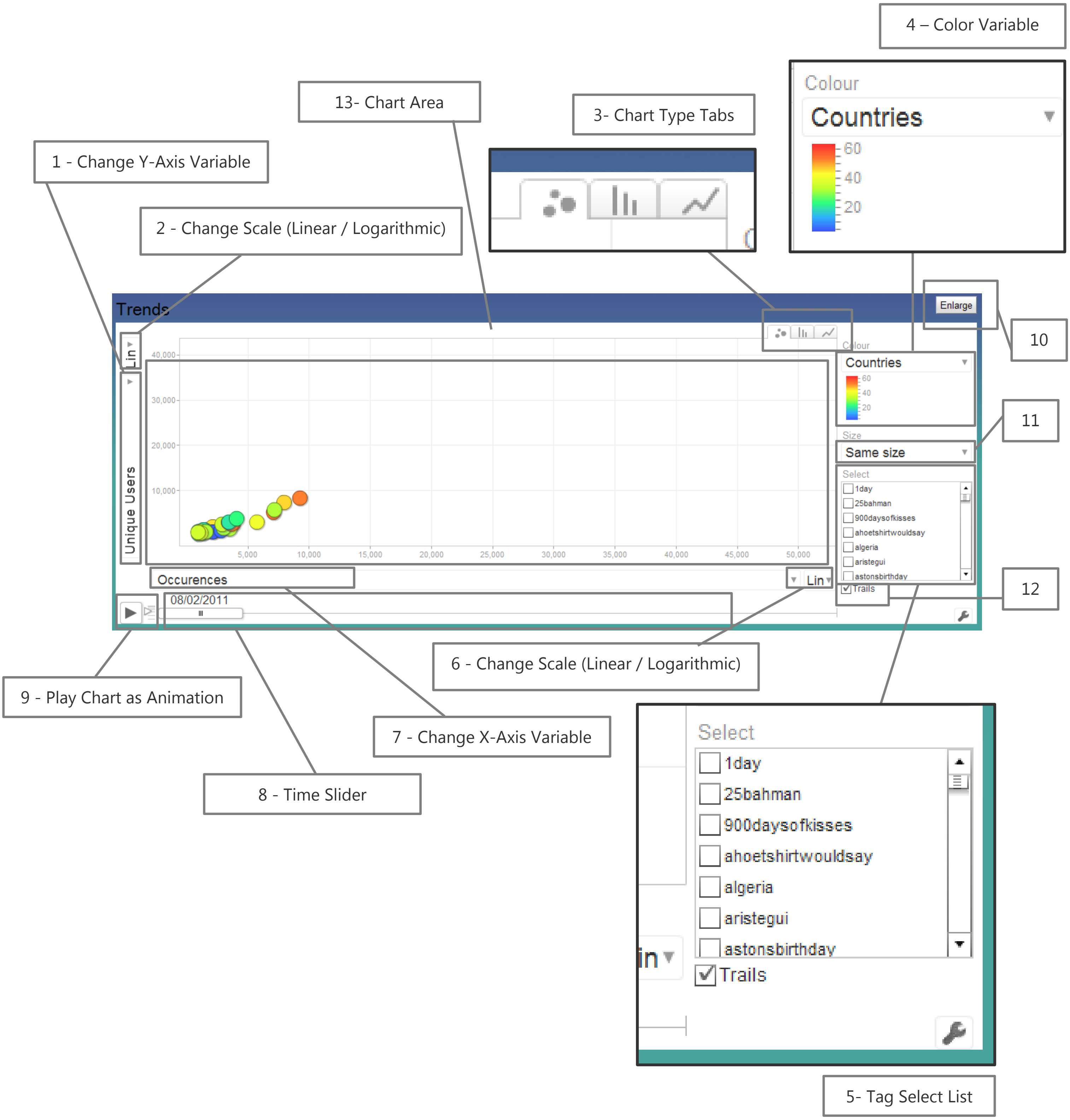
### Time Query Components

Successful time queries will result in five components being displayed inside the Component Area:



### Trends Component

The Trends Component is a multifunction pivot chart, which allows for analysis of daily trend data in a number of formats, comparing multiple variables.

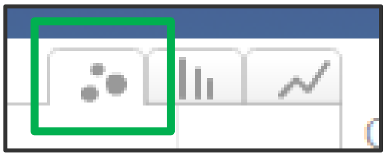


### *Diagram Key*

1. *Change Y-Axis Variable:* Allows the Y-axis variable to be changed. Available options are 'Countries', 'Occurences' and 'Unique Users'.
2. *Change Scale (Linear / Logarithmic):* Change the scale type for the Y-axis. Can be either linear or logarithmic.
3. *Chart Type Tabs:* Change the type of graph displayed. The 3 main chart types are discussed in detail later in this section – 'Scatter', 'Bar' or 'Line'.
4. *Colour Variable*: Colouring can be used represent a third variable, along with the X and Y axises. Available options are 'Countries', 'Unique Users', 'Occurences', 'Same Colour' (all points are same colour'), or 'Unique Colour' (all points have different colour'). The gradient below indicates what each colour represents.
5. *Tag Select List*: Choose specify tags to show on each chart. Click 'Deselect All' to reset this list.
6. *Change Scale (Linear / Logarithmic):* Change the scale type for the X-axis. Can be either linear or logarithmic.
7. *Change X-Axis Variable:* Allows the X-axis variable to be changed. Available options are 'Countries', 'Occurences' and 'Unique Users', 'Time' and 'Order: Alphabetical'
8. *Time Slider:* Scroll to watch graph points change over time. Will only be visible if X-axis is set to anything other than 'Time'.
9. *Play Chart As Animation:* Animate the graph points over time. Use the slider to the left of the play button to change playback speed.
10. *Enlarge Trends Component:* The Trends Component contains much detail. In order to better use it, click this button for the component to occupy all space in the component area.
11. *Size Variable:* The size of each node can be used to represent a fourth variable. Available options are 'Same Size', 'Countries', 'Unique Users' and 'Occurences'.
12. *Use Trails*: Show trails as each node is animated. Use in conjunction with the time slider to create ad-hoc line graphs
13. *Chart Area*: Area used to display results.

### *Using the Trends Component*

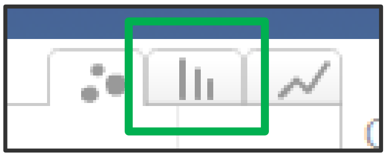
The Trends Component has a number of options and variables which you can manipulate to view data several ways. The three main views of this component are:

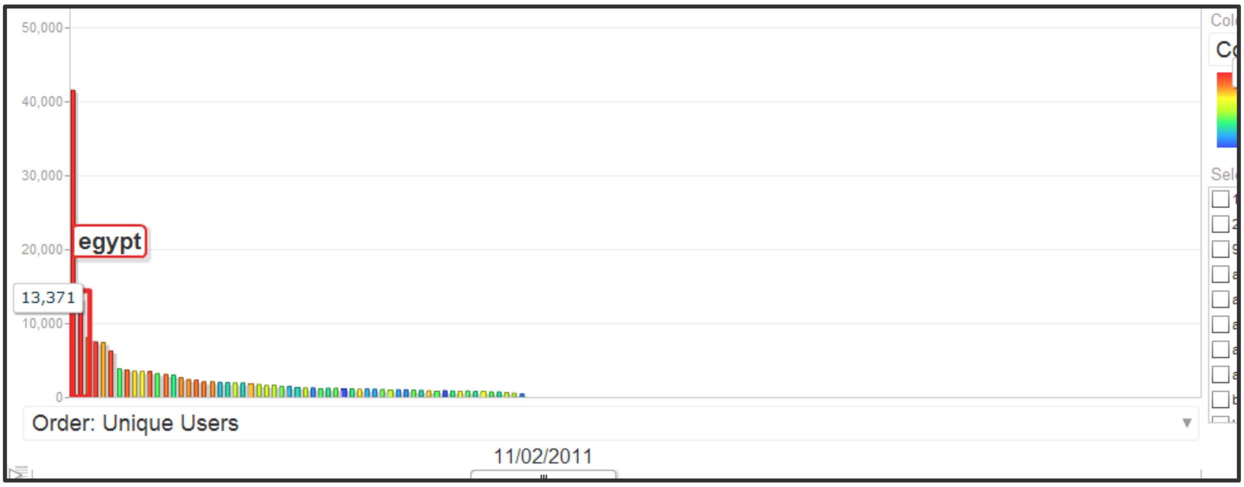


- **Scatter View**: each tag is displayed as a node in the graph area. You can change the X and Y axises, as well as the size and colour of each node, to represent up to four variables (number of unique users using the tag, number of occurences, number of countries using tag and time tag used).

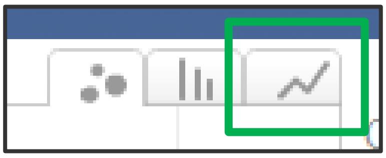
  Scatter view is shown in the diagram on the previous page.

- **Bar View**: each tag is displayed as a single bar in the graph area. Again, you can change the variables assigned to either axis, the colour or size of each bar. Bar View is shown below (cropped for space):

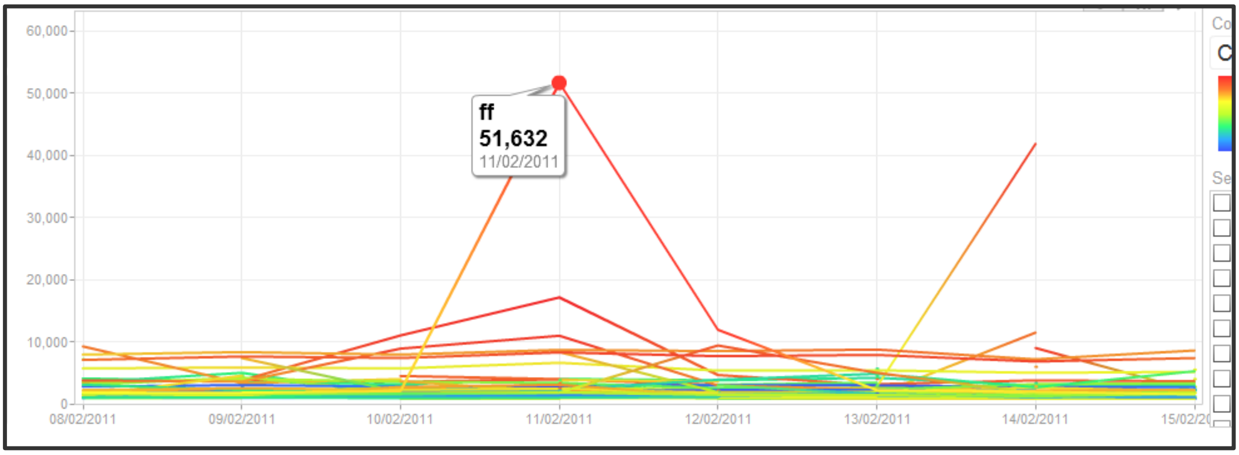




- **Line View**: each tag is displayed as a single line, with the X axis fixed to represent time. The Y axis and color attribute can be change to represent other variables.



Selecting a point within any line will show details for the tag represented by that line, at the given point in time. It also highlights that line within the graph area. Line view is shown below (cropped for space):



### *View Functionality*

The best way to discover the capability of the Trends Component is to use it – there are lots of ways to pivot daily data. However, the following table gives an overview of the main functionalities available:

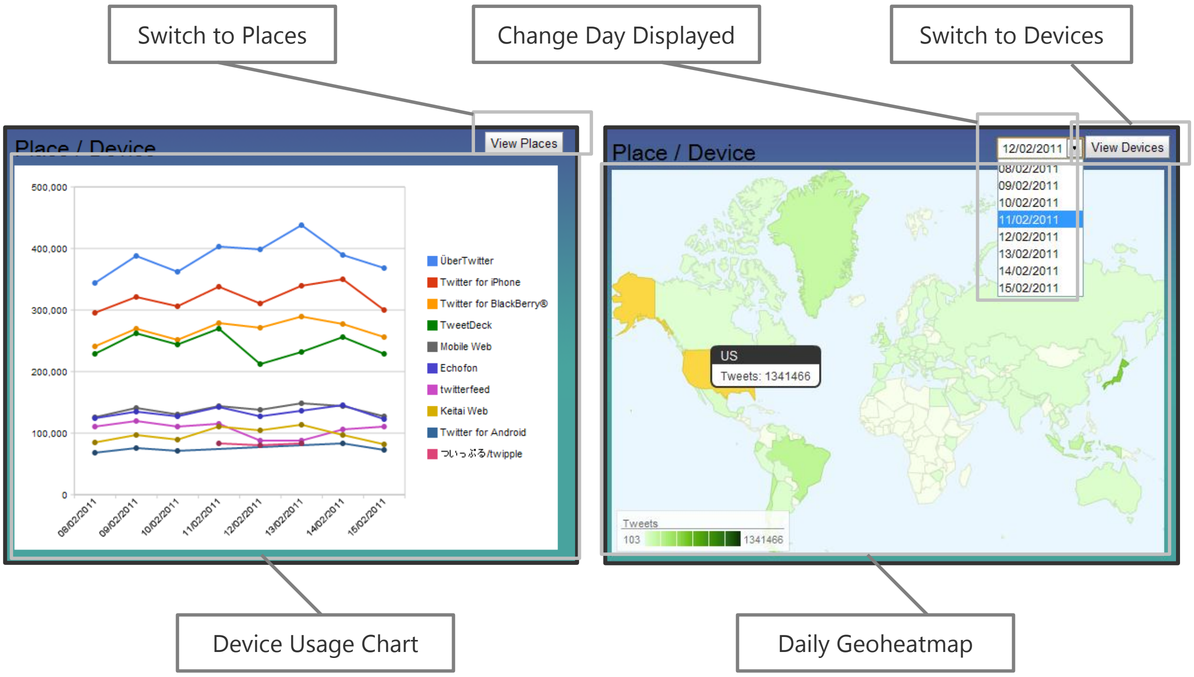
| *Action* | *Scatter View* | *Bar View* | *Line View* | *Notes* |
|---|---|---|---|---|
| **Hover over chart node / bar / line** | Highlight specific node / bar / line, show data for each visualisation method (axises, colour, shape) | | | - |
| **Click chart node / bar / line** | Selects specific node / bar / line in Selected Tags list, fades non-selected nodes / bars / lines out | | Additionally zooms in on tag and adjusts axises | These functions apply also to selecting a tag from the tag list |
| **Click 'Trails'** | Show trails as node moves over time | - | - | - |
| **Click and drag to draw box in chart area** | Option to 'zoom in' on smaller area of graph for closer detail of close together nodes / bars | | - | - |

***Trend Component Options***

The majority of the options described in the Time Query Options apply to this component (e.g. number of tags, upper / lower boundaries / whitelists etc.)

## Place / Device Component

The Place / Device Component details how popular tweeting is in various countries using particular devices or software.



Switch to Places

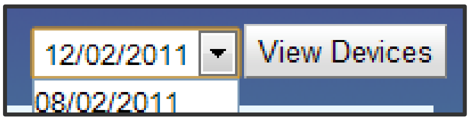
Change Day Displayed

Switch to Devices

Device Usage Chart

Daily Geoheatmap

The Place / Device Component has two main views: a line chart showing the number of tweets posted using a particular device or software, and a geographical heatmap showing which countries were most active during a selected day.



To switch between views, click the button in the top right corner (above).

***Device View***

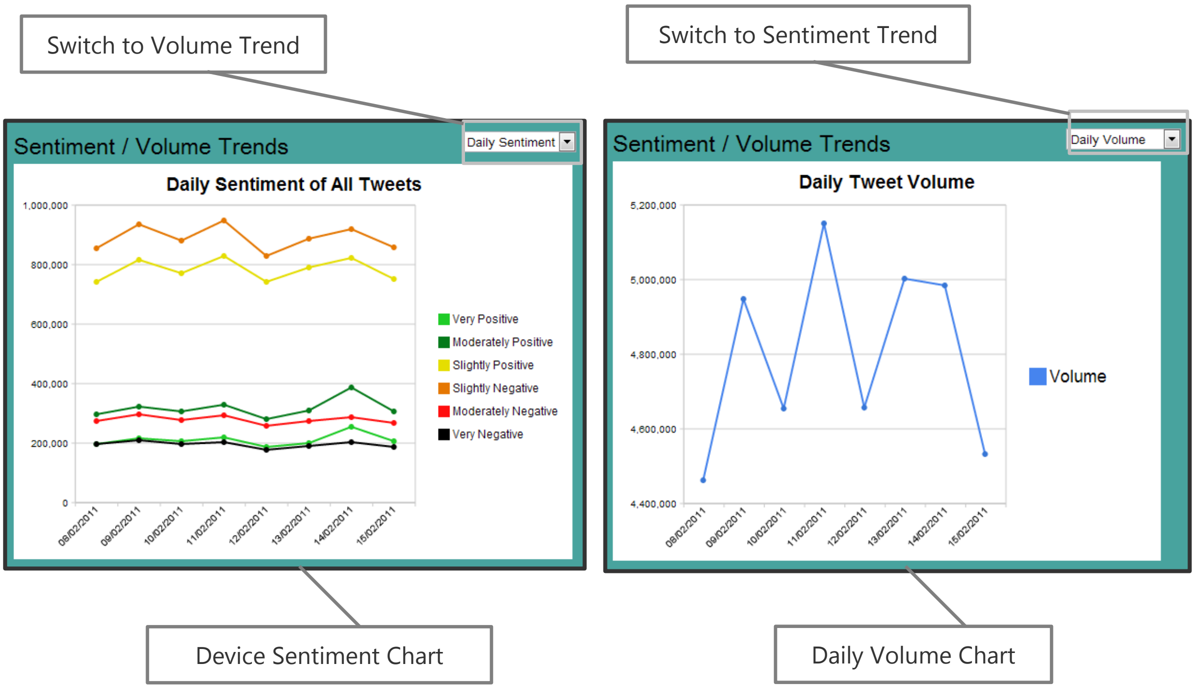
The Device View tracks device or software used to post to Twitter across multiple days. The days shown consist of the query day, plus the number of additional days either side (configured in the Time Query Options). Should 'exclude web' be selected in the options then 'web' as a tweet source will not be shown.

***Place View***

The Place View shows a geographical heatmap of tweet activity on a particular day. In this view an additional drop down list is visible that allows the selection of a particular day to view. Hovering over a country shows the number of tweets posted that originated from that particular country (similar to the Geographical Heatmap Component).

## Sentiment / Volume Trends Component

The Sentiment / Volume Trends Component shows daily sentiment across all tweets, as well as daily volume of tweets.



Switch to Volume Trend

Switch to Sentiment Trend

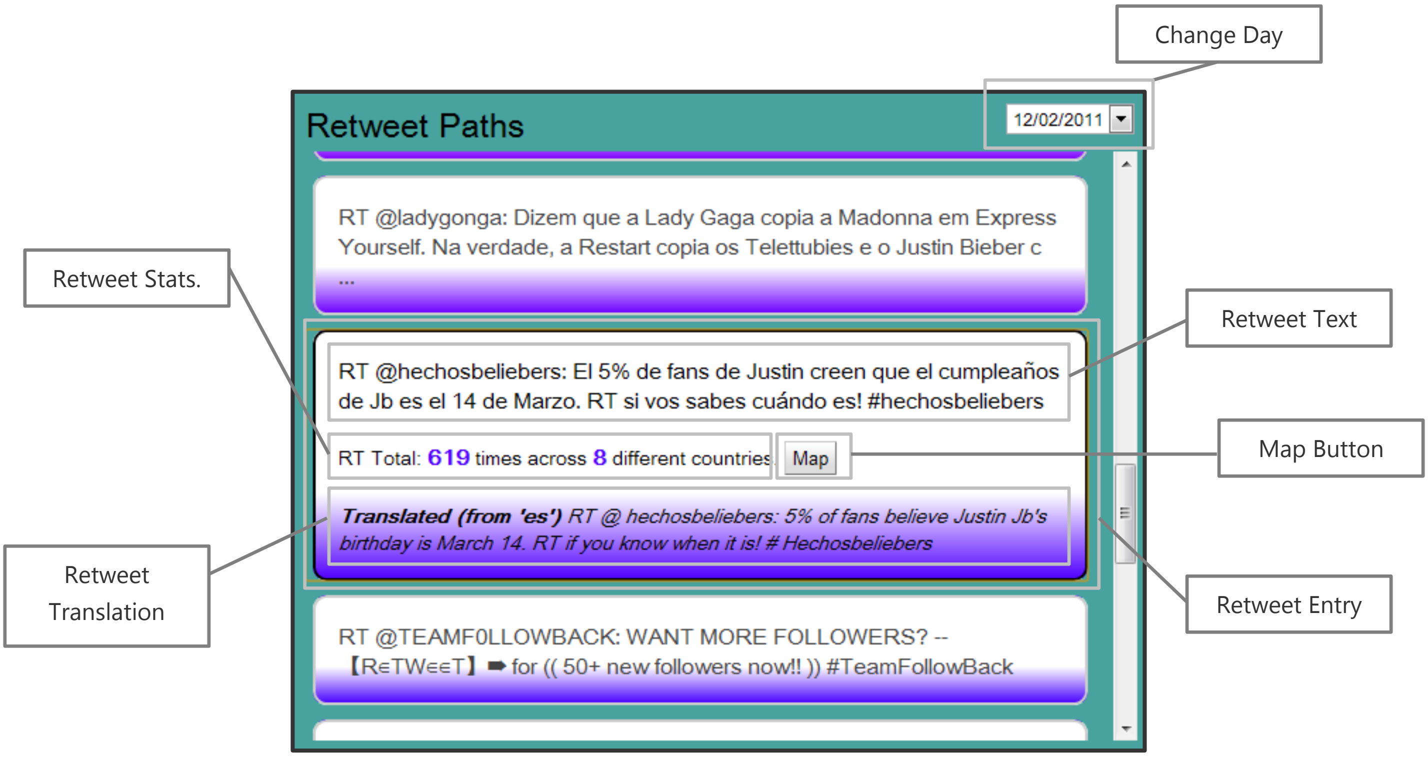
Device Sentiment Chart

Daily Volume Chart

To changes views, select the desired view from the View List in the upper right corner.

## Retweet Paths Component

The Retweet Paths Component provides information about the top retweets of a particular day.



Change Day

Retweet Stats.

Retweet Text

Map Button

Retweet Translation

Retweet Entry

Retweet Paths

12/02/2011

RT @ladygonga: Dizem que a Lady Gaga copia a Madonna em Express Yourself. Na verdade, a Restart copia os Telettubies e o Justin Bieber c ...

RT @hechosbeliebers: El 5% de fans de Justin creen que el cumpleaños de Jb es el 14 de Marzo. RT si vos sabes cuándo es! #hechosbeliebers

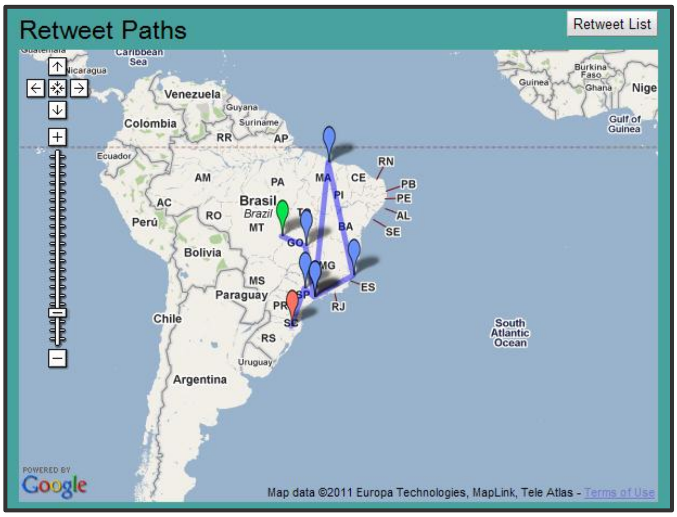
RT Total: 619 times across 8 different countries

*Translated (from 'es') RT @ hechosbeliebers: 5% of fans believe Justin Jb's birthday is March 14. RT if you know when it is! # Hechosbeliebers*

RT @TEAMF0LLOWBACK: WANT MORE FOLLOWERS? -- 【R≡TW≡≡T】➡ for (( 50+ new followers now!! )) #TeamFollowBack

Each retweet within the Retweet Paths Component is contained within a Retweet Entry. The gradient inside the retweet entry is a color between solid red and solid blue denoting the popularity of that retweet visually.

Similarly, the popularity of each tweet is also denoted by its position within the list – the nearer the top, the more popular the retweet. The most popular retweets feature a solid red gradient and are at the top of the list.



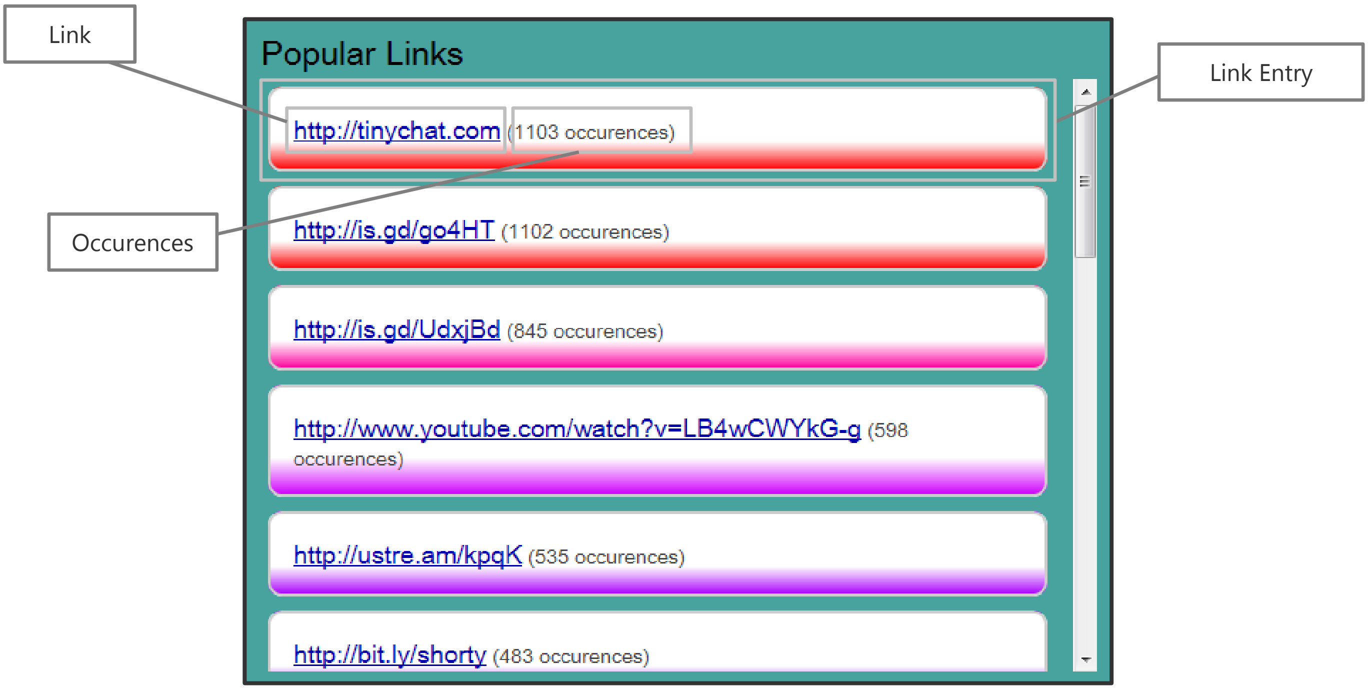
Within the retweet entry is the original text of the retweet. If this is in a language other than English, a translation will be provided underneath.

In addition to RT text, the volume of retweets, and number of countries is also displayed. The Map Button will attempt to recreate the path of the retweet, from first appearance to last (right). *Whilst this gives a rough approximation of the retweet path, it is generated using a small subset of location data for the retweet.*

To return to the retweet list after viewing a map, click the 'Retweet List' button.

### Popular Links Component

The Popular Links Component is a simple list of the top most tweeted links on Twitter for the query day.



Popular Links

http://tinychat.com (1103 occurences)

http://is.gd/go4HT (1102 occurences)

http://is.gd/UdxjBd (845 occurences)

http://www.youtube.com/watch?v=LB4wCWYkG-g (598 occurences)

http://ustre.am/kpqK (535 occurences)

http://bit.ly/shorty (483 occurences)

As with the Retweet Paths Component, the gradient for each Link Entry denotes the popularity of the link, from solid red to solid blue.

For each link, the link itself and the number of occurences is given. Clicking on a link will open in a new browser window / tab.

# About

**Pulse** was developed by Paul Holmes, a level 4 student of Computing Science at the University of Glasgow. The project was supervised by:

- Dr Iadh Ounis (iadh@dcs.gla.ac.uk)
- Dr Craig Macdonald (craigm@dcs.gla.ac.uk)

# Appendix B

# Pre-Evaluation Questionnaire

# Pre-Evaluation Questionnaire

| *Version* | *Change log* |
|---|---|
| **1.0** | Fixed minor errors |
| 0.1 | Initial draft |

# Information

## Overview

Before you attempt the task list we would like to establish your experience in the use of social media, Twitter and analytic tools. The following questions will ask about each of these fields, and there are no right or wrong answers. Please complete them as fully as possible.

## Instructions

This questionnaire has 4 types of questions:

- **Yes / No** questions.
- **Explanatory questions**, where we ask you to explain and expand upon previous answers.
- **Closed rating questions**, where we ask you to rate some element on a scale of 1 to 7. Usually the most positive response corresponds to '1' and the least positive to'7'.
- **Open opinion questions**, where we ask for your comments and opinions. Please answer these as comprehensively as possible.

Whilst we would appreciate you completing the entirety of the form, you may omit any question which you do not wish to answer.

Should you have any questions, please speak to the demonstrator.

# Questionnaire

## Social Network Experience

1. Which social networks sites or applications do you use, if any?

| | | | |
|---|---|---|---|
| *Facebook* | □ | *Bebo* | □ |
| *Twitter* | □ | *Last.fm* | □ |
| *LinkedIn* | □ | *Forums / Usergroup* | |
| *MySpace* | □ | *Other (please specify)* | |

2. What are the main reasons you use social networking sites?

| | | | |
|---|---|---|---|
| *Keeping in touch with friends / family* | □ | *Follow figures of interest* | □ |
| *Organising part of your life (work / home)* | □ | *Discussion / debate* | □ |
| *Sharing interests or hobbies* | □ | *Other (please specify)* | |
| *Stay informed about news* | □ | | |

3. How much (of all your time spent online) do you spend using social networking sites?

| 1 – All time online | 2 | 3 | 4 | 5 | 6 | 7 – No time online |
|---|---|---|---|---|---|---|
| □ | □ | □ | □ | □ | □ | □ |

## Trend Analysis Experience

1. Have you ever used any trend analysis sites or tools? (please specify)

| | | | | | |
|---|---|---|---|---|---|
| *Yes* | □ | *No* | □ | *Tools Used* | |

2. For what reason do you use these tools?

| | | | | | |
|---|---|---|---|---|---|
| *Market Research / Business Use* | □ | *Academic Purposes (Computing)* | □ | *Academic Purposes (Other)* | □ |
| *Own Interest* | □ | *Other* | | | |

3. Do the tools have any particular features you heavily utilise?

4. Which, if any, Twitter analysis websites or tools have you used (for topic analysis, follower comparison, hashtag monitoring etc.)

| | | | |
|---|---|---|---|
| *Twitter Search / Trends* | □ | *Analytic.ly* | □ |
| *Twitalyzer* | □ | *Radiant 6* | □ |
| *Twitterurly* | □ | *Other (please specify)* | |
| | □ | | |

5. Of these tools, what features were most useful?

6. Of these tools, what features were least useful?

7. Did any of the tools lack a feature you would have liked, and if so, what?

## Twitter Experience

1. Do you have a Twitter account?

*Yes* □ *No* □

2. What do you primarily use Twitter for? (select as many as required)

| | | | |
|---|---|---|---|
| *Posting own Tweets* | □ | *Research / academic purposes* | □ |
| *Following likeminded Twitter users* | □ | *Have account but do not use* | □ |
| *Following organisations / news* | □ | *Linked to blog / website / other social media* | □ |
| *Following celebrities / media personalities* | □ | *Other (please specify)* | |

3. What platform / application do you typically use to post on Twitter?

| | | | |
|---|---|---|---|
| ***Application*** | | ***Platform*** | |
| *From Twitter.com* | □ | *In browser* | □ |
| *Official Twitter Application* | □ | *Mobile Phone* | □ |
| *3rd Party Twitter Application* | □ | *Standalone Application* | □ |
| *Through another website (blog etc.)* | □ | *Other (please specify)* | |

4. How many users do you follow, and how many follow you (rough numbers are O.K.)

| | | | |
|---|---|---|---|
| *Followers* | | *Following* | |

5. How often do you post updates to Twitter?

| Multiple times a day | Daily | Multiple times a week | Weekly | Monthly | Rarely | Never |
|---|---|---|---|---|---|---|
| □ | □ | □ | □ | □ | □ | □ |

6. How do you primarily choose to follow new users?

| | | | |
|---|---|---|---|
| *Friends / colleagues* | □ | *Recommendations from others* | □ |
| *Through work* | □ | *By following public timeline* | □ |
| *Hobbies / interests* | □ | *Other (please specify)* | |
| *News organisations* | □ | | |

7. Do you actively promote your own Twitter account?

*Yes* □ *No* □

8. If so, how? (particularly if you use any tools to do so)

9. How often do you 'retweet' others' posts?

| Most posts are retweets | Many posts are retweets | Some posts are retweets | Few posts are retweets | Never |
|---|---|---|---|---|
| □ | □ | □ | □ | □ |

10. What are the main reasons you would retweet another user's post?

| | | | |
|---|---|---|---|
| *Useful to your own followers* | □ | *News you feel is important* | □ |
| *Humorous* | □ | *Other (please specify)* | |

11. What other web services do you connect your Twitter account to, if any?

| | | | |
|---|---|---|---|
| *Facebook / Social Networking site* | □ | *Blog* | □ |
| *LinkedIn* | □ | *Other (please specify)* | |

12. When posting updates, is the content primarily…

| | | | |
|---|---|---|---|
| *Work related* | □ | *In relation to news or other's posts* | □ |
| *Personal* | □ | *Other (please specify)* | |

13. Compared to other web services (social networks, blogs, commenting etc.) how much of your time is spent using Twitter?

| 1 – Most time on Twitter | 2 | 3 | 4 | 5 | 6 | 7 – No time on Twitter |
|---|---|---|---|---|---|---|
| □ | □ | □ | □ | □ | □ | □ |

## Appendix C

# Post-Evaluation Review

PULSE

# Post-Evaluation Review

| *Version* | *Change log* |
|---|---|
| **1.0** | Minor fixes |
| 0.2 | Addition of word list. |
| 0.1 | Initial draft |

# Information

## Overview

Now that you have completed the user evaluation tasks, we would be grateful if you could complete the following tasks related to your experience using **Pulse.** This consists of 2 sections:

- **Word Selection & Discussion:** We'd like you to pick as many words as you like from the list provided that you feel describe the **Pulse**. After this, we'd like to discuss why you selected these words.
- **Questionnaire**: A final questionnaire to further establish your thoughts on **Pulse**

# Word Selection

Please check the words that you feel best describe **Pulse** (as many or as few as you like). Once you've done this, please circle **5** that you feel are the most relevant.

□ Simple
□ Intuitive
□ Clear
□ Bright
□ Effective
□ Usable
□ Entertaining
□ Unpredictable
□ Old
□ Patronising
□ Familiar
□ Secure
□ Too technical
□ Responsive
□ Convenient
□ Credible
□ Fun
□ Professional
□ Empowering
□ Busy
□ Complex
□ Energetic
□ Meaningful
□ Time-saving
□ Stimulating
□ Dull
□ Friendly
□ Approachable
□ Advanced
□ Business-like
□ Ambiguous
□ Illogical
□ Unattractive
□ Innovative
□ Difficult
□ Confusing
□ Irrelevant
□ Controllable
□ Frustrating
□ Time-consuming
□ Poor quality
□ Stressful
□ Trustworthy
□ Reliable
□ Effortless
□ Relevant
□ Fresh
□ Ineffective
□ Cutting edge
□ System-oriented
□ Simplistic
□ Straightforward
□ Organised
□ Ordinary
□ Intimidating
□ Boring
□ Counter-intuitive
□ Powerful
□ Fast
□ Contradictory
□ Vague
□ Faulty
□ Overwhelming
□ Distracting
□ Clean
□ Annoying
□ Rigid
□ Efficient
□ Misleading
□ Attractive
□ Creative
□ Unconventional
□ Consistent
□ Predictable
□ Accessible
□ Dated
□ Exciting
□ Useful
□ Incomprehensible
□ Sophisticated
□ Hard to Use
□ Motivating
□ Insecure
□ Flexible
□ Easy to use
□ Engaging
□ Unrefined
□ Impressive
□ Desirable
□ Obscure
□ Expected
□ Cluttered
□ Inconsistent
□ Appealing
□ Comprehensive
□ New
□ Compelling
□ High quality
□ Stable
□ Awkward
□ Inadequate
□ Non-standard
□ Understandable
□ Satisfying
□ Slow

# Usability

| | *Statement* | 1 – Strongly Agree | 2 | 3 | 4 | 5 | 6 | 7 – Strongly Disagree |
|---|---|---|---|---|---|---|---|---|
| 1 | Overall, I am satisfied with how easy it was to use **Pulse.** | □ | □ | □ | □ | □ | □ | □ |
| 2 | It could effectively complete tasks set using **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 3 | I could complete tasks set quickly using **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 4 | I felt comfortable using **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 5 | It was easy to learn how to use **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 6 | I believe I became productive quickly using **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 7 | The information (such as help and other documentation) provided with **Pulse** was clear | □ | □ | □ | □ | □ | □ | □ |
| 8 | It was easy to find the information I needed | □ | □ | □ | □ | □ | □ | □ |
| 9 | The information provided by **Pulse** was easy to understand | □ | □ | □ | □ | □ | □ | □ |
| 10 | The information was effective in helping me complete the set tasks | □ | □ | □ | □ | □ | □ | □ |
| 11 | The organisation of information within **Pulse** is clear | □ | □ | □ | □ | □ | □ | □ |
| 12 | The user interface of **Pulse** is pleasant | □ | □ | □ | □ | □ | □ | □ |
| 13 | I liked using the interface of **Pulse** | □ | □ | □ | □ | □ | □ | □ |
| 14 | Overall I was satisfied with **Pulse** | □ | □ | □ | □ | □ | □ | □ |

# Further Development

1. Compared to other trend analysis / social media analysis tools you have used, how does **Pulse** compare?

| 1 – Much better than existing tools | 2 | 3 | 4 | 5 | 6 | 7 – Much worse than existing tools | *Not Applicable* |
|---|---|---|---|---|---|---|---|
| □ | □ | □ | □ | □ | □ | □ | □ |

2. If you were able to access **Pulse** on a regular basis, what purposes would you primarily use it for, if any?

| | | | |
|---|---|---|---|
| *Understanding particularly topic trends on Twitter* | □ | *Tracking events* | □ |
| *General interest / browsing* | □ | *Wouldn't use it* | □ |
| *Comparing Twitter activity across time* | □ | *Other (please specify)* | |

3. Which feature / functionality of **Pulse** did you find most useful?

4. Which feature / functionality of **Pulse** do you think needs further work?

5. If you could add any additional functionality to **Pulse**, what would it be?

# Appendix D

# Evaluation Task List

# Evaluation Tasks

| *Version* | *Change log* |
|---|---|
| **1.0** | Fixed minor typographical errors, amended certain searches |
| 0.1.1 | Revised after reviewing available data |
| 0.1 | Initial draft |

# User Evaluation

## Introduction

Thank you for volunteering to take part in the **Pulse** user evaluation.

The following pages detail a series of tasks we would like you to attempt using **Pulse**. All of these are operations we expect users to undertake on a day-to-day basis.

There are no incorrect ways of completing any of the tasks. Should you have any difficulty completing a task, or see any errors which you do not understand, feel free to skip the task and move onto the next one.

Should you have any questions you can ask the demonstrator at any time.

## Points to Consider

As you complete the tasks, please consider the following:

- Clarity in the wording of options, menus, dialogs and errors.
- Ease of navigation around the system
- Ease in finding particular pieces of information
- Usefulness of each component
- Speed and performance
- Overall look and feel

## During the Evaluation

You will have access to the **Pulse** user guide throughout the evaluation. Please make use of this as you conduct each task.

Certain tasks will ask you to note information. Once you have found the required information, please note it in the notes sheet.

## Leaving the evaluation

You are free to pause or leave the evaluation at any point. If you would like to pause or end the evaluation, please speak with the demonstrator.

## Questionnaire

Once you have completed the task list, please speak to the demonstrator. They will then provide you with a short questionnaire to garner information on your experience using **Pulse**. It would be greatly appreciated if you could answer as fully and accurately as possible, bearing in mind the above points.

# Tasks

## Walkthroughs

The first tasks of the evaluation are designed to accustomise you to the system, providing an overview of each key feature.

### Walkthrough 1

**Hashtag Search**

1. Search for **'#gaddafi'** selecting date **'March 1st'**
2. Once the results have appeared, note the number of times **'#gaddafi'** was used as a hashtag on this day

### Walkthrough 2

**Hashtag Comparison Search by Date**

1. Search for **'#gaddafi'**, selecting date **'February 11th'**
2. Once the results have appeared, select the '**Compare**' tab in the Navigation Bar
3. Select **'gaddafi'** from *both* drop down menus, and click 'Compare'
4. Since these results have been cached, they should appear instantaneously. Note the difference in number of occurences of the term **'#gaddafi'** between these two dates

### Walkthrough 3

**Hashtag Comparison Search by Tag**

1. Search for **'#valines'**, selecting date '**February 14th'**
2. Once the results have appeared, search for **'#up',** selecting date '**February 14th'** again
3. Once the results have appeared, select the '**Compare'** tab in the navigation bar
4. Select '**valentines'** and '**up'** from the drop down lists, and click '**Compare'**
5. Note the tags that both hashtags have in common from the Tagged With Component.

### Walkthrough 4

**Using Options**

1. For large queries it is sometimes convenient to slightly reduce the comprehensiveness of results to obtain a large speed-to-display gain. To see this, search for **'#fb'**, selecting date '**February 1st'**
2. You should notice it takes slightly longer than normal to return these results. This is because of the processing time in certain components with particularly large hashtags
3. Once these results have loaded, select '**Options'** from the navigation bar
4. Reduce the **Map Plotting %** to 20
5. Reduce the **Sentiment Analysis %** to 20

6. Search for **'#fb'**, selecting date '**February 1st'**. The results should load considerably faster. This is because certain components are using less intensive methods to get results faster, as well as caching of results by the server.
7. To compare the differences in results, click the '**Compare'** tab in the Navigation Bar, select '**fb'** and '**fb'** in each of the drop down menus, and click '**Compare'**. You should see slightly different results in the **Location** and **Sentiment Analysis** components. (For full details of these Options, see the '**Options'** chapter of the user guide).

## Walkthrough 5

### Slashtag Search

1. In the Query Box search for **'#apple/#google'**, selecting data '**February 21st**
2. Once the results have appeared, *note which tag had more uses in the U.K*. by selecting the '**Comparison'** option from the **Geographical Heatmap Component** drop down list and hovering over the U.K
3. Slashtag searches can be used to quickly obtain comparisons of 2 queries on a particular day. These results are also cached individually, so that each can be individually compared against any other query. However, slashtag searches do requires 2 separate index lookups, so once retrieved use the tag query history to retrieve this if required again.

## Walkthrough 6

### Time Search

1. Clear the Query Box if there is any text in it.
2. Without typing anything in the search box, select date '**February 26th**' and click 'Go'
3. With no tag specified, **Pulse** will retrieve *daily trends* for the day specified if it has the required data.
4. Once the results have appeared, information about that day's Twitter output should appear
5. The user guide details the information available using time search, however feel free to explore each component to understand its functionality.

# Scenarios

These tasks require you to obtain certain pieces of information using the various components and features of **Pulse**. There are multiple ways to find the answers, each using different features of the system.

If you cannot find a particular piece of information, please note this and continue to the next scenario.

Once you have the required information please note it in the Notes sheet.

## Scenario 1

Which country tweeted using the tag '**#cnn'** the most on **February 16th**?

## Scenario 2

What time of day was the hashtag **'#arsenal'** most popular at on **March 2nd**?

## Scenario 3

How many more tweets were tweeted featuring the hashtag **'#ipad2'** on **March 2nd** compared to **February 28th**?

## Scenario 4

What hashtags did the terms '**#google**' and '**#microsoft'** have in common on **March 1st**?

## Scenario 5

What hashtag **spiked** in number of occurrences on **February 14th**?

## Scenario 6

When in **February** did the hashtag **'#egypt**' reach its highest number of occurrences?

## Scenario 7

Where did the most popular retweet of **February 9th** originate?

## Scenario 8

When did the hashtag '**#winning'** start showing as a top tag? (Hint: use the Options to set number of additional days to include in time queries).

## Scenario 9

What is the most popular device used for tweeting in February, after the actual Twitter site? How did this change on 19th February?

## Exploration

You've now completed all of the formal tasks in this evaluation.

However, **Pulse** allows for exploration of tags, trends and topic. As such, should you wish to use **Pulse** further with your own queries, please feel free to do so. *Your thoughts and opinions through free use would be greatly appreciated.*

## Finished?

Once you're ready to continue with the post-evaluation questionnaire please let the demonstrator know.

## Appendix E

# Administrator Guide

# Administrator Guide

| *Version* | *Change log* |
|---|---|
| **0.2** | Added property file info |
| 0.1 | Initial draft – branched from User Guide |

# Pulse for Administrators

## Background

**Pulse** is built using the Google Web Toolkit framework. GWT supports the building of complex browser-based applications which are written in Java then compiled to JavaScript. These are then deployed as a Java Servlet / WAR for production use.

## Requirements

- **Pulse** WAR (Web Archive)
- Apache Tomcat v7 or greater (http://tomcat.apache.org/)
- **Pulse** Data Source (daily data directories)
- For best results, Google Chrome (see 'Compatibility')

## Installation (Local)

1. Ensure you have downloaded the Apache Tomcat server. You can download this for free at http://tomcat.apache.org/download-70.cgi
2. Once you have installed Tomcat, copy the **Pulse** WAR ('pulse') into Tomcat's `/webapps` directory
3. Modify the `pulse.properties` file in `/pulse/WEB-INF`
    - **pulse.data.dir** should have the path to the **Pulse** data folder
    - **sentiment.dictionary.file** should have the path to the sentiment dictionary file (provided: /pulse/WEB-INF/dictionary
    - **sentiment.pos.tagger.file** should have the path to the sentiment POS tagger file (provided: /pulse/WEB-INF/tag.tagger)
4. Start Tomcat using the startup script for your OS (usually `<TOMCAT_HOME>/bin/startup.bat` or `.sh`)
    - You may have to set an environment variable in your OS for Tomcat – JAVA_HOME or JRE_HOME, pointing to your Java installation
    - Additionally, setting CATALINA_OPTS to "-Xmx1500m" will set the Java heap to 1.5Gb, resulting in a faster application experience. Adjust the number based on your system spec.
5. Access http://localhost:8080/pulse
6. **Pulse** is now ready to use

## Additional Deployment Steps

Operation outside of 'localhost' requires a valid Google Maps API key.

1. Sign up for a key for your domain at http://code.google.com/apis/maps/signup.html
2. Amend line 10 of /pulse/Pandemos.html, from this:

```
<script type="text/javascript"
src="http://maps.google.com/maps?file=api&v=2&sensor=false&key=ABQIAAAAuF5VcB7UH-
od3DosLmwT-BQrOhho5J3TAk93_AaOLIoHABI7TBRxYEuUrp3FSZh4Rq93_bhTluiISQ"></script>
```

To this:

```
<script type="text/javascript"
src="http://maps.google.com/maps?file=api&v=2&sensor=false&key=YOUR-KEY-
HERE"></script>
```

## Compatibility

This is the first **Pulse** release, and so it has only been fully tested using *Google Chrome*. As such, we recommend this browser is used when working with **Pulse**. It can be downloaded at http://www.google.com/chrome (Windows, Mac, Linux compatible).

# Attribution