Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations | Graduate Studies

8-2015

Improving the Scalability and Usability of the Public Information Officer Monitoring Application

Rohan D. Shah
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Shah, Rohan D., "Improving the Scalability and Usability of the Public Information Officer Monitoring Application" (2015). *All Graduate Theses and Dissertations*. 4407.
https://digitalcommons.usu.edu/etd/4407

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

IMPROVING THE SCALABILITY AND USABILITY OF THE PUBLIC INFORMATION OFFICER MONITORING APPLICATION

by

Rohan D. Shah

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Amanda Hughes Major Professor	Dr. Vladimir Kulyukin Committee Member
Dr. Kyumin Lee Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

Copyright © Rohan D. Shah 2015

All Rights Reserved

ABSTRACT

Improving the Scalability and Usability of the

Public Information Officer Monitoring Application

by

Rohan D. Shah, Master of Science

Utah State University, 2015

Major Professor: Dr. Amanda Lee Hughes
Department: Computer Science

This thesis work addresses the scalability and usability limitations of a web application called the Public Information Officer Monitoring Application (PMA). This application helps Public Information Officers (PIOs) to gather, monitor, sort, store, and report social media data during a crisis event.

To address the scalability issues of PMA, this research examines the algorithms used by the application along with the database architecture and implementation. This thesis describes the improvements made to these algorithms and the database, that together, increase PMA's ability to handle large amounts of data. In addition, PMA has been deployed using Amazon Web Services (AWS), which also improves PMA's scalability.

To address the usability issues of PMA, the application has been tested with PIOs to get feedback about using the application in emergency situations and suggestions for further improvements. All test participants found the application useful and relevant to

their work. Testing also revealed many ways to improve the usefulness of the application. Finally, the thesis concludes with suggestions for future work and distribution of PMA.

(103 pages)

PUBLIC ABSTRACT

Improving the Scalability and Usability of the

Public Information Officer Monitoring Application

Rohan D. Shah

This thesis work addresses the limitations of a web application called the Public Information Officer Monitoring Application (PMA). This application helps Public Information Officers (PIOs) to gather, monitor, sort, store, and report social media data during a crisis event. Before this work, PMA was unable to handle large data sets and as a result, it had not been adequately tested with potential users of the application.

This thesis describes changes made to PMA to improve its ability to handle large data sets. After these changes were made, the application was then tested with target users. All test participants found the application useful and relevant to their work. Testing also revealed many ways to improve the usefulness of the application, which were subsequently implemented. The thesis concludes with suggestions for future work and distribution of PMA.

ACKNOWLEDGMENTS

Foremost, I take this opportunity to acknowledge God for all the strength and vigor He gave me to accomplish this research, and how He showed me the way in unsettling times.

It is a great pleasure to acknowledge and express my sincere thanks towards my major professor, Dr. Amanda Lee Hughes. I thank her for all the guidance and time she gave me. Her thorough analysis of situation and attention to minute details has been very helpful in conducting this research. Her minute observation and clear advice on various things have helped me to be a better researcher and accomplish this research. I thank her for believing in me and strengthening my confidence throughout the process. Her constant motivation gave me energy to work harder.

I would like to thank my committee members, Dr. Curtis Dyreson and Dr. Kyumin Lee, for all their support and guidance throughout the master's program. I also acknowledge the classes I have taken from both of them, for they are the perfect teachers who not only care about their subject matter but also their students.

I would like to extend my thanks and gratitude to all the people who agreed to participate in my user study for testing the application. These tests were immensely valuable to answering my research questions.

I also want to thank my parents for the emotional support they have provided me throughout this journey. They have been there with me, constantly bolstering in times of hardship and when the going got tough.

Last but not least, I would like to thank my eldest cousin, Apurva J. Shah, for giving valuable guidance through this journey, and my sister-in-law, Avani A. Shah, for

taking care of me like a little brother. I also wish to thank my nephew Akshay A. Shah and my niece Alisha A. Shah for constantly suprising me with their innovative, childlike mischief and cheering me up.

Rohan D. Shah

CONTENTS

Page

ABSTRACT iii
PUBLIC ABSTRACT v
ACKNOWLEDGMENTS vi
LIST OF TABLES xii
LIST OF FIGURES xiii

CHAPTER

1: INTRODUCTION 1
1.2 Social Media in Emergency Response 2
1.3 The Public Information Officer Monitoring Application (PMA) 3
1.4 Research Questions 4
1.5 Research Design 4
1.5.1 Optimize and Scale PMA 4
1.5.2 Improve Usability of PMA 5
1.6 Thesis Overview 5
2: PIO MONITORING APPLICATION (PMA) 6
2.1 PMA Implementation 6
2.2 PMA Functionality 7
2.2.1 Streaming View 8
2.2.2 Reporting 9
2.3 PMA Limitations 10
2.3.1 Performance and Scalability Issues 10
2.3.2 Usability Issues 10
2.4 Summary 11
3: OPTIMIZE AND SCALE PMA 12
3.1 Environment and Scope 12
3.2 Improving Referential Integrity 13
3.2.1 “coding” Table 14
3.2.2 “event” Table 15
3.2.3 “maxhash[eventid]” and “topuser[eventid]” Tables 17

3.2.4 “maxrt[eventid]” Table 17
3.2.5 “tweetevent” Table 18
3.2.6 “twitterdailyactivity” Table 18
3.2.7 “categorycount” Table 19
3.2.8 “filter” Table 19
3.3 Report Generation 19
3.3.1 Top Twenty Retweets 20
3.3.1.1 Problem Assessment 20
3.3.1.2 Solution 22
3.3.1.3 Possible Drawbacks 23
3.3.2 Top Twenty Hashtags 25
3.3.2.1 Problem Assessment 25
3.3.2.2 Solution 26
3.3.2.3 Possible Drawbacks 27
3.3.3 Twitter Category Breakdown 27
3.3.3.1 Problem Assessment 27
3.3.3.2 Solution 28
3.3.3.3 Possible Drawbacks 29
3.3.4 Twitter Daily Activity 29
3.3.4.1 Problem Assessment 30
3.3.4.2 Solution 33
3.3.4.3 Possible Drawbacks 34
3.3.5 Top Ten Users 35
3.3.5.1 Problem Assessment 35
3.3.5.2 Solution 37
3.3.5.3 Possible Drawbacks 40
3.4 Summary 40
4: PERFORMANCE EVALUATION 42
4.1 Performance Data Collection 43
4.2 Execution Time of Report Features in the Report Controller 44
4.2.1 Top Twenty Retweets 44
4.2.2 Top Twenty Hashtags 45
4.2.3 Twitter Category Report 47
4.2.4 Twitter Daily Activity 48

4.2.5 Top Ten Users 49
4.3 Execution Time of Report Preprocessing in the Consume Script 51
4.4 Summary 53
5: DEPLOYING PMA ON THE CLOUD 54
5.1 Cloud Computing 54
5.2 Cloud Computing Criteria 55
5.2.1 Constant Usage Over Time 55
5.2.2 Spiked Internal Load 56
5.2.3 Spiked External Load 56
5.2.4 Steady Growth 56
5.3 Amazon Web Services (AWS) 56
5.4 Deploying PMA on AWS 57
5.5 Summary 58
6: USER STUDY 59
6.1 Previous Usability Enhancement Work 59
6.2 User Study Testing Procedures 60
6.2.1 Pre-Test 60
6.2.2 PMA Testing 60
6.2.3 Post-Test 62
6.3 User Study Findings 62
6.3.1 Participants 62
6.3.2 Results 62
6.3.3 Recommendations 64
6.3.3.1 Addressed Here 64
6.3.3.2 Future Work 65
6.4 Summary 66
7: CONCLUSION AND FUTURE WORK 67
7.1 Collecting Tweets Based on Location 68
7.2 Adjusting the Time Zone 71
7.3 Active/Non-Active Mode of Search Terms 72
7.4 Archiving Data Periodically 72
7.5 Efficient architecture for deploying the application using AWS 74
REFERENCES 76
APPENDICES 78

Appendix A Previous PMA Database Schema 79
Appendix B PMA User Testing Interview Questions..... 80
Appendix C Task List for PMA User Testing 81
Appendix D User Study with Participant P1..... 82
Appendix E User Study with Participant P2..... 85
Appendix F User Study with Participant P3..... 88

LIST OF TABLES

Table Page

1: Previous Query for Retrieving the Top Twenty Retweets 20

2: Profiling Information for Old Top Twenty Retweets Query 21

3: Profiling Information for New Top Twenty Retweets Query 24

4: Query to Get Tweets Belonging to an Event 26

5: Old Query to Get Twitter Daily Activity 30

6: MySQL "EXPLAIN" Result of Queries in Table 5 31

7: MySQL "EXPLAIN" Result of Queries in Table 5 34

8: Old Query to Get Data for the Top Ten User Feature 36

9: MySQL "EXPLAIN" for Query in Table 8 36

10: MySQL EXPLAIN of Query in Table 7 after Adding an Index to “eventid” 38

11: Test Codes for the Various Testing Conditions 43

12: Average Execution Time for Consume Script Per Tweet 52

LIST OF FIGURES

Figure Page

1: Categories for Organizing Tweets 8

2: Streaming View in PMA 9

3: Table Structure of "coding" Table 14

4: GUI for Different Categories Showing the Root 16

5: Table Structure for "events" 16

6: Profiling Information for the Queries in Table 5 32

7: Sample Data and Structure of Table "twitterdailyactivity" 34

8: Profiling Information of Query in Table 7 37

9: Profiling Information for the Query in Table 7 after Adding an Index to "eventid" 39

10: Query to Retrieve Top Users in the New Approach 40

11: Execution Time for Reporting the Top Twenty Retweets 45

12: Execution Time for Reporting the Top Twenty Hashtags 46

13: Time to Get the Top Twenty Hashtags from "maxhash[eventid]" Table 47

14: Execution Time for Reporting the Daily Twitter Activity 49

15: Execution Time for Reporting the Top Ten Users 50

16: Time to Get the Top Ten Users from " topuser [eventid]" Table 51

17: Execution Time for Preprocessing a Tweet by Report Feature 52

18: Cloud Computing Architecture 55

19: Placeholders and Google Map to Fill in "Location" 64

20: GUI Showing Future Work to Incorporate Location-based Tweet Searching 69

21: GUI for Location-based Tweets Search 70

22: GUI of Streaming View to Accommodate Location-based Tweet Search 71

23: GUI of Setting Search Terms..... 72

24: Proposed Architecture for Deployment on AWS 74

CHAPTER 1

INTRODUCTION

In recent times, social media and their use have increased dramatically. Through social media people can interact with each other and exchange information, thoughts, and opinions. As social media use has become more pervasive, people have found the technology useful in many different circumstances and situations. The particular focus of this work is on emergency responders and how they use social media to communicate and interact with the public during a crisis event.

In the United States, the emergency response role that is most affected by social media is the Public Information Officer (PIO). The primary responsibility of an emergency PIO is to communicate correct, up-to-date information concerning the status of emergency incidents to the media, members of the public, and other directly or indirectly affected stakeholders [1]. PIOs can use social media during an emergency to 1) obtain real-time information 2) gather information from various sources, 3) collect information specific to the conditions of the emergency, and 4) support two-way communication between PIOs and members of the public [1].

Despite the advantages of social media for emergency response work, there are several challenges that can make their use difficult for PIOs. One challenge is that it can be difficult to determine the correctness and/or authenticity of information collected through social media [2]. Another challenge is that social media users produce a large amount of data in a short amount of time and hence collecting, processing, and analyzing this data can be difficult. This thesis research aims to address these challenges by

improving and testing a software tool designed to help PIOs collect, process, report, and analyze social media data in times of crisis.

1.2 Social Media in Emergency Response

Members of the public have used social media in times of crisis to find information, offer assistance, seek support, and provide information to others [3]. Emergency managers recognize the value of these activities and have sought ways in which they can leverage the information that the public produces over social media [1]. However, recent research [4] shows that social media use by emergency responders is still relatively low and there is still much we do not know about how social media fits with emergency responder practice.

Several studies report how social media has been used by emergency responders. During the 2011 Shadow Lake Fire in Oregon, emergency responders used a Virtual Operations Support Team (VOST) to help manage social media [5]. A VOST is a team of volunteers comprised of experienced, remotely-located emergency managers. During the fire, the VOST helped to monitor and distribute information over social media like Twitter, Facebook, and YouTube [5]. In addition, they posted maps, statistics, images, press releases, and other data on the official blog for the response effort. The data that the VOST collected helped the emergency response team better understand the status and opinions of the public and make better decisions.

Another study [4] examined the use of social media by police and fire departments during the 2012 Hurricane Sandy. Social media, like Facebook and Twitter, were used as two-way communication channels through which the departments responded to queries, addressed concerns, and distributed information. By monitoring

social media, departments could quickly find and correct false rumors and misinformation.

PIOs face many challenges in using social media during emergency events. For example, Hughes and Palen [1] discuss how social media are one more communication channel and many PIOs lack the time and resources to attend to it. PIOs also find that there is so much information flowing from social media that it becomes difficult to retrieve and makes sense of the information [6]. Further, this information is changing so rapidly that PIOs are unsure where to look. PIOs often lack adequate training and the required tools to sift through large amounts of social media data and derive inferences quickly.

1.3 The Public Information Officer Monitoring Application (PMA)

Though social media use by PIOs is increasing, little work has sought to design and build technology to help PIOs with social media. The PIO Monitoring Application (PMA) is a tool that seeks to fill this gap. PMA allows PIOs to gather, monitor, sort, and report social media activity and data. It provides features like searching for and viewing social media based on search terms and creating reports from the collected data. However, before this work began, PMA was very limited because it could only handle small amounts of data. Whenever the number of social media messages collected by PMA increased above roughly 50,000 messages, the creation of reports and viewing of live message streams became difficult—often causing the application to error or timeout. Due to these limitations, PMA has been difficult to completely test with users. Consequently, the primary goals of this research work are to increase the data scalability

of PMA and improve PMA usability through user testing with a more scalable version of the application.

1.4 Research Questions

This research addresses the scalability and usability limitations of PMA with the following research questions:

1) What design and implementation changes will enable PMA to handle a larger volume of data?
2) How can PMA better address PIO usability needs?

1.5 Research Design

My research design breaks the research into two activities that address the research questions above: 1) optimize and scale PMA so that it can handle larger amounts of data quickly and effectively and 2) test and improve PMA's usability to ensure that it meets the social media needs of PIOs. In the remainder of this section, I provide more detail around these two research activities and the approach I took to accomplish them.

1.5.1 Optimize and Scale PMA

To optimize and scale PMA, I focus my attention on two problem areas—the creation of reports and the database implementation. The creation of reports in the original version of PMA performed very slowly as the number of social media messages increased. To address this concern, I employed new strategies to store and collect the data required to generate these reports. Thus, the data would not have to be newly derived every time a user creates a report through PMA. Next, I focused on improving the database implementation. I first evaluated the old architecture and then made several key

changes that significantly improved database performance, such as adding referential integrity in the form of indexes and foreign key constraints. These new strategies and changes are described in great detail in Chapter 3.

1.5.2 Improve Usability of PMA

The second research activity is to improve the usability of PMA. To accomplish this activity, I conducted user tests where I had PIOs use PMA and offer feedback on their experience. After completing these tests, I analyzed the results and made several improvements to PMA's interface design. The user tests and their results are documented more fully in Chapter 6.

1.6 Thesis Overview

This thesis document contains six additional chapters following this introductory chapter. Chapter 2 contains a description of PMA's features, capabilities, and limitations. Chapter 3 documents the changes I implemented to optimize and scale PMA so that it can efficiently handle larger volumes of data. Next, Chapter 4 describes the results of tests I conducted with PMA to verify that the optimization and scaling changes implemented in Chapter 3 improved performance. To further improve PMA's scalability and execution time, I deployed the application on Amazon Web Services (AWS). Thus, Chapter 5 explains AWS, the deployment process, and the benefits and motivation behind deploying PMA on AWS. Chapter 6 outlines the processes I used to test the usability of PMA with users, the results of the testing, and the recommended changes to PMA based on these results. Finally, Chapter 7 concludes with a summary of research findings and suggestions for future work.

CHAPTER 2

PIO MONITORING APPLICATION (PMA)

PMA is a web application that supports the social media monitoring, documenting, reporting, and organizing needs of PIOs during emergency events. PMA monitors social media (only Twitter at this time) by collecting and storing tweets based on keywords.

In this chapter, I first describe the features of PMA and how the application works. Then, I describe the limitations of PMA. These limitations form the motivation for the research questions given in Chapter 1 (optimize and scale PMA, and improve PMA's usability).

2.1 PMA Implementation

PMA is based on a PHP framework called Yii [7] that uses a Model-View-Controller (MVC) software architecture. The MVC architecture separates an application into three modules (Model, View, and Controller):

- The **model** manages the application's data; it stores and retrieves the data entities used by an application, usually from a database, and contains the logic for the application.
- The **view (or presentation)** displays the application's data in a specified format to be presented to the user. For example, in a web application, the view might present information to a user as an html web page.
- The **controller** ties the model and view modules together. The controller receives a request from the client, invokes the model to perform the requested operations and sends the data to the view to display to the application user.

To store data, PMA uses a MySQL database. MYSQL is an open source Relational Database Management System (RDBMS). Initially, PMA was deployed using an Apache web server. Later in this research, I deploy PMA using Amazon Web Services (AWS) (for more information on AWS, refer to Chapter 5) using a LAMP stack. A **LAMP stack** is a group of open source software products used to get web servers up and running. The acronym stands for Linux, Apache, MySQL, and PHP.

2.2 PMA Functionality

The user starts using PMA by creating an event (corresponding to an emergency situation) that they want to monitor. The user then enters keywords to find relevant Twitter messages for that event. Once these keywords are entered and activated, PMA uses the Twitter streaming API to collect tweets that match the provided keywords.

After tweets are collected, PMA displays them in its streaming view so that users can see and sort the tweets into different categories. Users can also generate reports based on the tweets collected for an event. A report can contain information about the top ten Twitter users, top twenty hashtags, top twenty retweets, a Twitter category report, and the level of Twitter activity for the event.

In PMA, users can also create various categories within an event (see Figure 1). These categories allow PIOs to organize and make sense of the tweets they have collected.

Figure 1: Categories for Organizing Tweets

The primary focus of the work done here affects the streaming view and the reporting functionality of the application. I describe these two PMA features in more detail below.

2.2.1 Streaming View

In this view, the tweets which match the active search terms (keywords) for the selected event are displayed (see Figure 2). The streaming view shows the most recent 20 tweets in real-time. New messages appear at the top of the screen and scroll down with a 2 second delay.

On the left-hand side of the streaming view, the user can create categories. These categories are customizable and can be used to sort and store tweets by topic. A user can drag-and-drop tweets into these categories. Categories can have sub-categories as well. To show that a category is a sub-category, it is indented to the right and appears in another color than the parent category. When the user clicks any of the categories, the user is taken to the archive view that displays all the tweets in that category. All the tweets in a sub-category are also a part of the parent category and therefore they are also

displayed when viewing this parent category. A user can also create rules for categories. Rules allow users to automatically categorize tweets based on a user name or the content of the tweet.

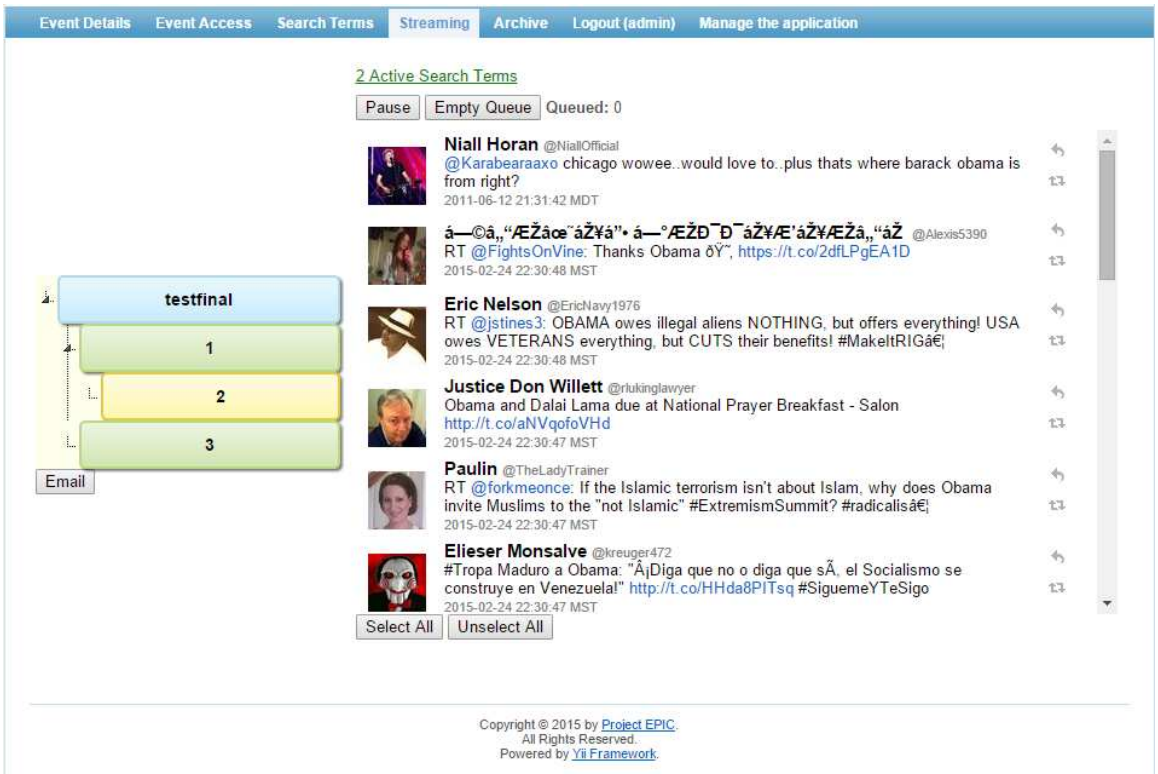

Figure 2: Streaming View in PMA

2.2.2 Reporting

PMA can create reports using collected Twitter data. PIOs can use the report function to automatically generate metrics, charts, and graphs, which they can then share with their management to demonstrate the usefulness of social media [6]. The report is created as a Microsoft Word file containing various charts and graphs of the 1) top twenty hashtags, 2) top ten twitter users, 3) top twenty retweets, 4) recent twitter activity, and 5) a category breakdown of tweets.

2.3 PMA Limitations

Before I improved PMA for this thesis work, the application had several limitations that made it unsuitable for widespread distribution and use. I describe these limitations below.

2.3.1 Performance and Scalability Issues

Several important features of PMA slowed down and became inefficient as the data volume increased. One such feature is the report generation feature of PMA. This feature made many queries to the PMA database, and the query execution time increased as the number of tweets in the database increased. Another feature with performance issues was the streaming view. As the number of tweets increased in the database, the streaming view began to show inconsistency in displaying new tweets. Sometimes the tweets were displayed and sometimes they were not. At other times, PMA simply froze. The work I describe in Chapter 3 addresses these issues.

2.3.2 Usability Issues

PMA had usability issues identified in earlier research [6] that I address in this work. The biggest issue was that the application had no support for multiple users or role-based access. The application also contained several software bugs that affected the user experience. I describe these issues and the PMA changes to correct them in Chapter 6.

Because the application had so many scalability issues, it could not be run outside of a highly-controlled setting. Once these issues were fixed, then testing to identify and correct other usability issues could take place. Consequently, I conducted a user study (described in Chapter 6) that tested the improved PMA with PIOs to discover further areas of improvement for the application.

2.4 Summary

In this chapter, I described PMA, its features, and how to use it. I also discussed the limitations of PMA when I began this work. The following chapters provide more detail around these limitations, the approach I took to address them, and the final solution.

CHAPTER 3

OPTIMIZE AND SCALE PMA

The first task in this thesis work is to address the scalability concerns of PMA that were briefly discussed in Chapter 2—namely, improving referential integrity in the database and optimizing the report generation feature. I begin this chapter by discussing the environment and scope for the scalability work I conducted. I then describe in detail the approach I took to improve referential integrity and optimize the report generation feature.

3.1 Environment and Scope

I conducted this PMA scalability work in a single software/hardware environment and under a set of conditions (or scope). I used a single environment and scope to ensure consistency and comparability across testing and results. The following list describes these criteria:

1) The application will have limited users (2-3).
2) The keywords used to gather tweets will be minimal (approximately 2-3).
3) All the tweets used for performance evaluation will belong to the same event.
4) The application will be executed on the same machine for pre and post analyses.
5) Only the number of tweets in the database will be variable.
6) The report creation task will always be executed with all reporting options.
7) Tests were executed on an Intel Core i5 machine, with 6GB RAM, and a 1TB hard disk.
8) The local server is XAMPP, which contains a MySQL database server and an Apache web server.

9) The application runs on Google Chrome Version “37.0.2062.103 m”. The browser will be updated to the current version throughout the research.

3.2 Improving Referential Integrity

To improve PMA’s database design and performance, I evaluated and made changes to the database’s referential integrity. Referential integrity, in this context, is concerned with setting appropriate primary and foreign key constraints among tables. In addition, referential integrity includes indexing appropriate columns in the tables for efficient query execution. Configuring foreign key constraints requires the column to be indexed. So, most of the indexing work will be covered while setting up foreign key constraints in the table. After adding foreign keys and indexes, many of the queries will no longer require a full table scan for retrieving data,[1] which results in a performance gain. The schema of the database before improving referential integrity can be seen in Appendix A.

There are tradeoffs to adding indexes and foreign key constraints, which I list below:

1) The speed of INSERT and UPDATE operations decrease.
2) Extra locking occurs because a parent row must be locked when a child row is modified to ensure referential integrity.
3) Lookup overhead is increased. For example, the database must find the correct parent record corresponding to the child reference in order to validate that the parent exists. If it does not exist, one will encounter referential integrity errors.
4) The definition of indexes and constraints must be stored in the database. Which means more checking and space utilization.

[1] http://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html

Even though there are disadvantages when using foreign keys and indexes, the advantages of using them outweigh the disadvantages. For instance, the use of foreign key constraints and indexes results in faster select queries with WHERE clauses, efficient queries with JOINS, and data consistency among the many tables. Below, I describe the tables where I added foreign keys and indexes.

3.2.1 "coding" Table

The application has a feature that allows a user to insert collected tweets into various categories, ultimately "coding" the tweets. To use this feature, a user can drag-and-drop a tweet into one of the categories that they created for the given event. The "coding" table (see Figure 3) maintains a record of the tweets and the categories in which they have been categorized. This table stores the ID of the user who categorized the tweet, in the "userid" column, along with the ID of the category that the tweet is assigned to in the "coding" column.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**tweetid**	varchar(20)	latin1_swedish_ci		Yes	*NULL*	
2	**id**	int(11)			No	*None*	AUTO_INCREMENT
3	**coding**	int(11)		UNSIGNED	No	*None*	
4	**userid**	int(11)			No	*None*	
5	**level**	int(11)			Yes	*NULL*	

Figure 3: Table Structure of "coding" Table

I added a foreign key constraint on the "userid" and "coding" column to refer to "user.id" and "category.id" columns, respectively, where "user" and "category" are the respective table names. The application uses the "userid" and "coding" columns in multiple MySQL statements with WHERE clauses. I could have just added indexes to

them but adding appropriate foreign keys provides for consistent data across various tables.[2] Some of the reasons for using foreign key constraints appear below:

1) A foreign key exists as a constraint on the table to stop entities from inserting data that do not point to anything.
2) Foreign keys are used as a clue for the query optimizer.
3) A foreign key helps ensure referential integrity, meaning that if there is a value in a foreign key there will be a corresponding record with that value as a primary key in the parent table.

Since this application is built using the Yii framework, there are a number of MySQL queries written in Yii that use the "userid" and "coding" columns in WHERE clauses. This change has affected the efficiency of those queries positively because now when these queries retrieve data, a complete table scan is no longer required.

3.2.2 "event" Table

This table stores all the information about a particular event. This table is one of the most frequently used tables because much of the content displayed by the application depends on the currently selected event. In this table, the columns "id", "startdate", and "rootcategoryid" are used frequently in WHERE clauses. The column "rootcategoryid" stores the ID of the category. This ID is for the root/parent of all the other sub-categories in the event (see Figure 4).

[2] http://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html

Figure 4: GUI for Different Categories Showing the Root

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(11)			No	None	AUTO_INCREMENT
2	name	varchar(128)	latin1_swedish_ci		No	None	
3	location	varchar(128)	latin1_swedish_ci		Yes	NULL	
4	startdate	date			Yes	NULL	
5	enddate	date			Yes	NULL	
6	summary	text	latin1_swedish_ci		Yes	NULL	
7	rootcategoryid	int(11)		UNSIGNED	No	None	

Figure 5: Table Structure for "events"

Since the columns “startdate” and “rootcategoryid” are used while generating reports and while loading the hierarchy of categories for an event, respectively, I chose to add an index on these two columns. Since every event will have a root and thus a root category ID, I added a foreign key constraint, referencing “rootcategoryid” to “category.id” to maintain consistency. Now, when an event is deleted, its corresponding root category in the “category” table will also be deleted (enabling data consistency across tables).

3.2.3 "maxhash[eventid]" and "topuser[eventid]" Tables

These tables are used to store the count of hashtags and the count of tweets by a Twitter user, respectively, so that it can be used for generating reports for an event.

Initially, the column "count" in both of these tables had an index. But since this column will be updated constantly in both the tables, it would be expensive to have an index on the column. The drawback of not having an index is that queries using the ORDER BY clause on "count" will be slower. This tradeoff is acceptable as updating the "count" column for any given record happens much more frequently than ORDER BY queries on the column "count."

An index has been assigned to columns "hashtag" and "userid" in tables "maxhash[eventid]" and "topuser[eventid]" respectively as they are often used in the WHERE clauses of search queries. The column "userid" is a foreign key that refers to "tweets.fromuserid" so that joins can be made.

3.2.4 "maxrt[eventid]" Table

This table stores the retweeted tweets and their corresponding retweet count. This table is used for generating reports based on tweets that were retweeted the most for this event. This table stores the tweet ID of the tweet that was retweeted. I added a foreign key constraint on column "tweetid" so that it refers to "tweets.tweetid" where "tweets" is another table. This change helps maintain consistency across the tables. If a tweet is deleted in table "tweets," the corresponding entry in table "maxrt[eventid]" will also be deleted and will no longer show up in the report.

3.2.5 "tweetevent" Table

This table stores the tweets belonging to their respective events. It has a "tweetid" and "eventid" column along with another primary key column. Column "eventid" is used by many queries for displaying tweets in an event's views (e.g.: streaming, archive). Constantly using this table to access many tweets is inevitable and needs to be efficient. Column "eventid" is used in WHERE clauses in a number of queries and hence adding an index offers performance improvements. Adding foreign key constraints on "tweetid" and "eventid" helps ensure consistency across the database. These constraints ensure that if an event is deleted from the table "events," all the corresponding rows in the table "tweetevent" having the ID of the deleted event in column "eventid" will be deleted. Likewise, when a tweet in the "tweets" table is deleted, it will no longer show up in the event's view(s) as well.

3.2.6 "twitterdailyactivity" Table

This table stores the count of tweets tweeted on a particular date within a particular time interval. This table is used to get the daily tweet count for the reporting feature. This table is not specific to a particular event like the "maxhash[eventid]" or "topuser[eventid]" tables but stores data for all the events.

An index has been assigned to "eventid" column because this column is frequently used in WHERE clauses. The column "eventid" also has a foreign key in the table that refers to the column "id" in the table "events." To get the tweet count for various criteria like time and date, the columns "date," "fromtime," and "totime" are also assigned indices as they are also frequently used in WHERE clauses.

3.2.7 "categorycount" Table

This table stores the number of tweets in each category for all the events. Whenever this table is queried, column "eventid" will be used in a WHERE clause as it is a common table for all the events. Hence column "eventid" is a good candidate for an index. In addition, the column has a foreign key constraint which refers to column "id" in table "events" because a join must be created every time this table is queried. Since this table also stores each category's ID, the "categoryid" column is a foreign key that refers to the ID of the category in the table "categories." An index has been assigned to the column "tweetcount" as PMA frequently uses the column in SELECT queries and WHERE clauses within those queries. This column will be updated less often so the problem of "slow updates" for an indexed column should not be too bothersome but it will be faster to retrieve data.

3.2.8 "filter" Table

This table stores the search terms for each event and also stores values corresponding to whether the search term is active or not. PMA contains queries that select the search terms based on their active status, hence indexing the column "active" can offer performance improvements. Also, PMA performs several simple SELECT queries that use the search term stored in the column "filter." Thus, indexing the "filter" column helps improve performance since there would be many search terms as the number of events in the application increase.

3.3 Report Generation

The report generation feature has five options for the user to select from. Based on the options selected, PMA generates analytics for those options and places those

analytics in a report. The five options are 1) top twenty retweets 2) top twenty hashtags 3) twitter category report 4) top ten twitter users and 5) daily twitter activity. I describe how I optimized the implementation for these options below.

3.3.1 Top Twenty Retweets

This feature calculates the twenty tweets that were retweeted most frequently for a particular event.

3.3.1.1 Problem Assessment

The previous query (see Table 1) has ORDER BY and GROUP BY clauses that hamper performance because they require sorting of large volumes of data.

Table 1: Previous Query for Retrieving the Top Twenty Retweets

```
select tweetid, text, fromuser, max(retweetcount) as
MaxTweetCount from tweets a
INNER JOIN tweetevent b USING(tweetid)
where b.eventid=".$user->selectedevent."
group by tweetid order by MaxTweetCount DESC  limit 20
```

This query could be improved if 1) indexing was done properly and 2) the number of rows to search decreased. In the query, the columns “tweetid” and “eventid” were properly indexed. However, when the data increased the performance began to degrade because of the GROUP BY and ORDER BY clauses. The query takes 14 seconds on average to iterate through 111,469 rows when there are 124,834 tweets belonging to the selected event and 145,979 total rows in the “tweets” table.

Table 2 displays the profiling information for the query shown in Table 1 above. I generated this information using the profiling tool phpMyAdmin.[3] The profiling information shows that sorting takes a significant amount of time. In addition, the query takes a lot of time to allocate and free resources as well (e.g., Creating Tmp Table).

Table 2: Profiling Information for Old Top Twenty Retweets Query

Status	Time
Starting	74 µs
Checking Permissions	5 µs
Checking Permissions	3 µs
Opening Tables	40 µs
Init	48 µs
System Lock	11 µs
Optimizing	15 µs
Statistics	44 µs
Preparing	19 µs
Creating Tmp Table	260 µs
Sorting Result	4 µs
Executing	2 µs
Sending Data	11.5 s
Converting HEAP To MyISAM	307.3 ms
Sending Data	2.3 s
Creating Sort Index	60.8 ms
End	10 µs
Removing Tmp Table	9 ms
End	14 µs
Query End	12 µs
Closing Tables	16 µs
Freeing Items	149 µs
Logging Slow Query	3 µs
Cleaning Up	21 µs

The "Converting HEAP to MyISAM" entry in Table 2 is a thread state and takes the maximum amount of time. Here, the thread is converting an internal temporary table from a *MEMORY* table to an *on-disk MyISAM* table.[4] This happens when the temporary

[3] http://www.phpmyadmin.net
[4] http://dev.mysql.com/doc/refman/4.1/en/general-thread-states.html

tables created by MySQL in the intermediate steps of query processing grow too big to be stored in memory. When doing this, MySQL writes temporary MyISAM tables irrespective of the database engine uses (it is just a temp table that will be deleted at the end of the query). The problem is that this process takes a lot of time, and unless the size of the data used in this query decreases it will continue to be time consuming.

3.3.1.2 Solution

After improving the indexes and joins for the "top twenty retweet" query, I reached a point where tweaks to the database could no longer lower the execution time. I began to look for other ways to improve the execution time and realized that the query was repeating tasks every time the user generated a report. If the user generates reports at regular intervals, very little would have changed as far as the result of this feature/option is concerned. So, instead of computing most of the results repeatedly, I decided to compute them once and thus develop a solution where the query has a constant look-up time.

To achieve this constant look-up time, I made a table which stores the top twenty tweets. The table is solely dedicated to an event and every event will have its own table. This table is updated every time a tweet is encountered which has a retweet count more than the tweet having the least retweet count in this dedicated table. The tweet with the least retweet count is deleted and the new tweet is inserted in this table. When the user chooses to create a report with the top twenty retweets, the application queries this dedicated table and returns the data in sorted order. Since there will only 20-30 records in this table, sorting will not be an expensive operation. The purpose of adding 30 tweets in the table when the requirement is only of 20 tweets is to keep a buffer. If one of the

tweets is deleted and the user runs a report, he would get only 19 tweets when there should be 20 in the report. Also if in future, a user wants to see the top thirty retweets instead of twenty, it can be achieved with minimal changes in the code.

Every tweet has a parameter named “retweet_count” in the tweet’s JSON record retrieved through the Twitter Streaming API. PMA uses this parameter to determine how many times a tweet has been retweeted. At the same time when a tweet is collected and inserted into the `tweets` table, PMA compares the “retweet_count” parameter of that tweet to values in the “top twenty tweets” table and creates an entry for that table if appropriate. The naming convention of the “top twenty tweets” table is “maxrt[eventid].” For example, for an event having ID 47, the name of the table would be “maxrt47.”

3.3.1.3 Possible Drawbacks

As expected the execution time of the new “top twenty retweets” query decreases considerably and takes just 0.0005 seconds to iterate through 20 rows. Table 3 contains the profiling information for this new query.

Table 3: Profiling Information for New Top Twenty Retweets Query

Status	Time
Starting	29 µs
Checking Permissions	5 µs
Opening Tables	23 µs
Init	16 µs
System Lock	7 µs
Optimizing	4 µs
Statistics	11 µs
Preparing	9 µs
Executing	2 µs
Sending Data	71 µs
End	4 µs
Query End	5 µs
Closing Tables	6 µs
Freeing Items	60 µs
Cleaning Up	9 µs

Since a "top twenty retweets" table is created for every event, one might think that the performance will in turn degrade as the number of tables in the same database increases. However, this assumption is only partially correct. Performance will degrade if the number of tables in the database is in the thousands. This probability of having this many events is rare. Furthermore, we plan to mitigate this problem by archiving unused data periodically to keep the number of tables from growing beyond capacity. MySQL databases have the capacity to store 4Billion tables.[5]

The current database architecture uses the InnoDB engine. InnoDB stores the indexes of the table in a file. If the file is too large, because it stores indexes of many tables, then it will not fit in RAM and it will reside on the server's hard drive. This will cause the opening of tables to be a little slower. However, it is unlikely that this issue will cause a significant decrease in performance.

[5] http://dev.mysql.com/doc/refman/5.5/en/database-count-limit.html

The solution implemented here shifts responsibility for computing the maximum retweet count from the MySQL server to the web-hosting server. This computation is done on an individual tweet level rather than on a large number of records in a MySQL table. The script that collects tweets will now be slightly slower as it has to do more than just collect tweets and store them in the "tweets" table. This script will now also check the retweet count of a tweet, check to see if it is greater than the least retweet count in the table, and if yes, store the new tweet and delete the tweet with the least retweet count. I found this trade-off worthwhile because the look-up happens in constant time. Also, I found through testing that this slight increase in work had a negligible effect on server performance.

3.3.2 Top Twenty Hashtags

This feature in the report creates a table that displays the twenty most popular hashtags used in the tweets for a particular event.

3.3.2.1 Problem Assessment

The "tweets" table stores the information for all the tweets in PMA. Therefore, any query to that table will take time as it can contain millions of tweets. Further, joining this table with another table could consume significant time and resources. The old query for determining the top twenty hashtags does exactly that (see Table 4); it first performs a join to determine which tweets belong to the current event and then it searches through all of those tweets for hashtags. This would not be problematic if an event only has a few thousands tweets, but it will significantly slow down the query if an event has hundreds of thousands of tweets or even millions of tweets.

Table 4: Query to Get Tweets Belonging to an Event

```
select text from tweets a
INNER JOIN tweetevent b USING(tweetid)
where b.eventid=".$user->selectedevent
```

The query in Table 4 retrieves all the tweets which belong to the currently selected event. Next, the code extracts all the hashtags from the returned tweets and stores them in a hash map where the key is the hashtag and the value is the number of times that hashtag has occurred. PMA then uses this hash map to determine and display the top twenty hashtags in a report.

The problem with this approach is that every time the user wants to create a report containing the top twenty hashtags, the present algorithm iterates through all the tweets, extracts the hashtags and computes the maximum count of hashtags. These operations are done on tweets which have often already been evaluated. Also, the extraction of hashtags slows down the process, as PMA uses the regular expression function preg_match_all()[6] [7] to find the hashtags.

3.3.2.2 Solution

This problem is similar to the problem discussed for calculating the "top twenty retweets," hence the solution is also similar. The only difference is that the JSON for each tweet that we retrieve does not contain any information about the hashtag counts. Hence, we need to calculate the hashtag count programmatically. This changes the

[6] http://maettig.com/code/php/php-performance-benchmarks.php
[7] http://we-love-php.blogspot.com/2014/03/strpos-vs-pregmatch-vs-stripos.html

operation on the table which stores the top twenty hashtags, because we can not delete the least occurring hashtag as it may appear in future tweets and its count may increase.

The naming convention of the table that stores the top hashtags is "maxhash[eventid]." For example, "maxhash52" stores the count of hashtags occurring in the tweets belonging to the event with ID 52.

3.3.2.3 Possible Drawbacks

Since this solution is similar to the "top twenty retweets" solution, the drawbacks are also mostly the same and will not be repeated here. However, there is one drawback that is unique to this solution:

1) Since PMA does not delete the hashtags with a minimum count, the size of `maxhash[eventid]` table will likely increase indefinitely as new tweets are collected. Thus, querying this table could become cumbersome. But, this query is a normal select query with an ORDER BY clause that is not as costly as many other queries. Also, once an event ends the number of tweets about that event decrease, and the number of new hashtags decreases as well. Lastly, as an event winds down, the user will likely generate fewer reports for that event. Based on these factors, it is unlikely that this drawback will greatly affect the system.

3.3.3 Twitter Category Breakdown

This report feature creates a graph of the categories in the current event and the number of tweets in those categories.

3.3.3.1 Problem Assessment

PMA implements this feature by querying the "category" table and retrieving all the categories whose "root" column values matches the "rootcategoryid" column of the

event table. This query returns all the categories belonging to the current event. The returned categories are stored in an array. Then another table “coding” is queried with the “id” column of the retrieved categories and a count function is applied for each category to get the number of tweets. The “coding” table stores a tweet’s ID coupled with the category ID where a user has categorized that tweet.

The Twitter category breakdown feature counts the number of tweets in the “coding” table that belong to each category for an event. Every time a user creates a report using this feature, PMA recalculates these counts. I sought to devise a solution that would not repeatedly retrieve and recount tweets, but rather count the tweets once and update these counts as more tweets are added to the event.

Despite inefficiencies, this feature does not have the worst performance because the table “coding” will only have a row for as many tweets as have been assigned to a category. Typically the number of these rows is not large and the MYSQL server can handle it well.

3.3.3.2 Solution

Even though this is not a huge problem at present, it could cause bigger problems if the size of the “coding” table grows. The “coding” table will grow as the number of events increase and more and more tweets are categorized. With this scenario in mind, I sought to improve the way PMA calculates category counts.

I created a table named “categorycount” which stores the number of tweets in each category for each events. The table has a column that stores the event ID of the event each category belongs to. I then changed PMA, so that when a record is created in

the “coding” table, the tweet count of this category in the “categorycount” table will be increased by 1.

3.3.3.3 Possible Drawbacks

1) The new table “categorycount” counts only those tweets which are manually assigned to the category (i.e., by dragging and dropping). Some work needs to be done to count the tweets that are categorized by setting up rules.
2) Like other tables (maxrt[eventid], maxhash[eventid]), there is a minor chance this table may grow too large. Since this table stores the categories for all the events and the count of tweets in those categories, the size of this table depends on the number of categories. According to the use case, having more than ten categories per event is highly unlikely. The performance of the table might slow if there are more than 1 million entries in the table. That means there has to be 100,000 events where each event has 10 categories. This would be a lot of events before the performance starts to degrade. Also because of good indexing, the performance loss should be negligible. The other factor preventing performance degradation is the number of queries to this table. Since it would be queried only when a report is generated and when a tweet is put into a category, the number of queries to this table should be low and hence performance degradation should be minimal.

3.3.4 Twitter Daily Activity

This feature of the report returns a chart that contains the dates for an event and the number of tweets collected for each date.

3.3.4.1 Problem Assessment

PMA previously used three MySQL queries to retrieve the data for the Twitter daily activity feature (see Table 5). First, the oldest and newest tweets for the selected event were found by querying the "tweetevent" table twice—once to find the oldest tweet and once to find the newest tweet. Retrieving the oldest and newest tweets allows PMA to calculate the range of the dates for that event. Finally, the third query finds the count of the tweets on each day by grouping the count by dates within the range of dates.

Table 5: Old Query to Get Twitter Daily Activity

```
$sql="select created from tweets a INNER JOIN tweetevent b
USING(tweetid) where b.eventid=".$user->selectedevent." order
by created ASC limit 1";
$command = $connection->createCommand($sql);
$oldestTweetDate = $command->queryAll();
$sql="select created from tweets a INNER JOIN tweetevent b
USING(tweetid) where b.eventid=".$user->selectedevent." order
by created DESC limit 1";
$command = $connection->createCommand($sql);
$newestTweetDate = $command->queryAll();
$sql="SELECT DATE(created) as date, COUNT(*) as
tweetcount FROM tweets a INNER JOIN tweetevent b
USING(tweetid) where b.eventid=".$user->selectedevent." AND
DATE(created) >= DATE('".$oldestTweetDate[0]['created']."')
AND DATE(created) <=
DATE('".$newestTweetDate[0]['created']."') GROUP BY
DATE(created)";
$command = $connection->createCommand($sql);
$tweetCountByDate = $command->queryAll();
```

The EXPLAIN command generates an analysis of the statement succeeding it. When you precede a SELECT statement with the keyword EXPLAIN, MySQL displays

information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order.[8] With the help of EXPLAIN, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order.[8]

The fields found in the tables displaying EXPLAIN result are defined below:

1) Select_type: the kind of select statement used in this query
2) Table: the table on which the query is performed
3) Possible_keys: the possible indexes to be chosen
4) Key: the index actually chosen
5) Ref: the columns compared to the index
6) Rows: estimate of the number of rows examined
7) Extra: additional information about the query

Table 6: MySQL "EXPLAIN" Result of Queries in Table 5

Select_type	table	Possible_keys	key	ref	rows	extra
PRIMARY	b	TweetEventAssociation, eventid	eventid	const	514,89	A*
PRIMARY	a	tweetid_UNIQUE, created	tweetid_UNIQUE	twitterbucketsort.b.tweetid	A*	B*
SUBQUERY	b	TweetEventAssociation, eventid	eventid	const	514,89	A*
SUBQUERY	a	tweetid_UNIQUE	tweetid_UNIQUE	twitterbucketsort.b.tweetid	A*	NULL
SUBQUERY	Tweetevent	TweetEventAssociation, eventid	eventid	const	51,489	A*
SUBQUERY	tweets	tweetid_UNIQUE	tweetid_UNIQUE	twitterbucketsort.b.tweetid	A*	NULL

* A – Using where ; Using temporary, Using filesort , B– Using where

When analysis for Daily Twitter Count feature (refer Table 6) was done, the "tweetevent" and "tweets" table had around 103,000 records each. So it spans through

[8] https://dev.mysql.com/doc/refman/5.0/en/using-explain.html

50% of the records thrice. With an increase in the size of the tables, this number and execution time will increase.

The problem with these queries is that they are performing costly operations (sorting and creating temporary tables) for a small piece of data. These queries are run every time a report is made. The queries also contain multiple ORDER BY and GROUP BY clauses along with joins. As previously mentioned, these operations are expensive and if done on a table with many rows it can slow down the process considerably. There is also an aggregate function which consumes resources and time.

I profiled these queries (see Figure 6) when the “tweets” and “tweetevent” tables had 120,007 records each.

Status	Time
Starting	243 µs
Checking Permissions	7 µs
Checking Permissions	2 µs
Checking Permissions	2 µs
Checking Permissions	2 µs
Checking Permissions	2 µs
Checking Permissions	3 µs
Opening Tables	66 µs
Init	112 µs
System Lock	17 µs
Optimizing	20 µs
Statistics	128 µs
Preparing	27 µs
Creating Tmp Table	489 µs
Sorting Result	4 µs
Executing	4 µs
Sending Data	50.2 ms
Optimizing	26 µs
Statistics	81 µs
Preparing	18 µs
Creating Tmp Table	444 µs
Sorting Result	4 µs
Executing	2 µs
Sending Data	2.9 s
Creating Sort Index	8.4 ms
Optimizing	22 µs
Preparing	29 µs
Creating Tmp Table	442 µs
Sorting Result	5 µs
Executing	2 µs
Sending Data	2.5 s
Creating Sort Index	3.1 s
Creating Sort Index	68 µs
End	3 µs
Removing Tmp Table	9 µs
End	3 µs
Removing Tmp Table	244 µs
End	4 µs
Removing Tmp Table	199 µs
End	3 µs
Query End	6 µs
Closing Tables	13 µs
Freeing Items	106 µs
Cleaning Up	26 µs

Figure 6: Profiling Information for the Queries in Table 5

In column 2 of Figure 6, we can see that creating a sort index is taking a long time and so is sending data for computation. The aim of the solution below is to reduce these execution times because they are the most expensive operations.

3.3.4.2 Solution

The problem here is that there are costly operations done multiple times on the same data. The solution implemented here is to get rid of the GROUP BY and ORDER BY clauses and store the result incrementally in order to not perform the task on the same data every time.

First, I created a table called "twitterdailyactivity" that stores the number of collected tweets that were sent on a particular date along with their event ID. Next, I changed PMA so that when a tweet is collected and inserted into the "tweets" table, the date and event ID for that tweet would also be extracted and the corresponding count field in the "twitterdailyactivity" table would be incremented by 1. So now, when creating a report using the Twitter daily activity feature, PMA will query this table and display the tweet count for each day in the date range specified by the user. The time to execute this new query is much faster than three queries used previously.

This "twitterdailyactivity" table also stores the time interval for a date. These intervals allow flexibility in specifying date ranges as well as the future possibility of displaying tweet counts in intervals less than a day in length. The interval is currently set for 30 minutes. Figure 7 shows what these intervals look like in the "twitterdailyactivity" table.

			id	date	fromtime	totime	count	eventid
Edit	Copy	Delete	337	2015-02-06	07:00:00	07:29:59	1790	34
Edit	Copy	Delete	338	2015-02-06	05:00:00	05:29:59	72	34
Edit	Copy	Delete	339	2015-01-29	12:00:00	12:29:59	3	34
Edit	Copy	Delete	340	2015-02-04	16:00:00	16:29:59	4	34
Edit	Copy	Delete	341	2015-02-06	06:00:00	06:29:59	63	34
Edit	Copy	Delete	342	2015-02-05	22:30:00	22:59:59	36	34
Edit	Copy	Delete	343	2015-02-05	16:30:00	16:59:59	14	34
Edit	Copy	Delete	344	2015-02-06	06:30:00	06:59:59	71	34
Edit	Copy	Delete	345	2015-02-05	00:00:00	00:29:59	7	34

Figure 7: Sample Data and Structure of Table "twitterdailyactivity"

To get the daily twitter activity data for the report a simple look-up query can be used (see Table 7). In this query “$fromd” and “$tod” are provided by the user for more specific data to retrieve. The “fromtime” and “totime” column has been added to support future functionality. If the user wants to have more time-granular data result, a filter can be applied on those two columns and a report can be made on date as well as time.

Table 7: MySQL "EXPLAIN" Result of Queries in Table 5

```
SELECT DATE(date) as date, sum(count) as tweetcount FROM
twitterdailyactivity WHERE eventid = 34
AND DATE(date) BETWEEN DATE('".$fromd."') AND
DATE('".$tod."') GROUP BY date;
```

3.3.4.3 Possible Drawbacks

The size of the table will increase as tweets are collected over time. Consequently, fetching records could take longer over time. To solve this issue, I have indexed the

"date" column because it is used in the GROUP BY clause. The other columns (except for the "count" column) have also been indexed because they are used in the WHERE. This improvements will optimize query execution time even as the data set increases in size. The column "count" has not been indexed because it will be updated frequently—every time the number of tweets in that time interval increases. Indexing a column that is frequently updated will slow down the "update" process considerably.[9]

3.3.5 Top Ten Users

This feature in the report gives the name of the ten users who sent the most collected tweets for a particular event.

3.3.5.1 Problem Assessment

In the past implementation of the top ten users feature, PMA would go through all the tweets belonging to a particular event. An aggregate function was then used to count the number of tweets by the user by grouping the user names in a MySQL query. This list of users was maintained in a hash table with the user name as the key and the tweet count as the value. This operation is expensive because a join had to be made between the "tweetevent" table and "tweets" table using the ORDER BY and GROUP BY clauses (order by and group by clauses add overhead cost as mentioned before).

[9] http://dev.mysql.com/doc/refman/5.5/en/optimization-indexes.html

Table 8: Old Query to Get Data for the Top Ten User Feature

```
select fromuser, count(*) as tweetcount from tweets a INNER JOIN
tweetevent b USING(tweetid) where b.eventid=".$user->selectedevent."
group by a.fromuser order by count(*) DESC limit 10;
```

Another reason why this query was not efficient is because the "eventid" column in the table did not have an index even when it was used in WHERE clause.[10]

Table 9: MySQL "EXPLAIN" for Query in Table 8

Select_type	table	Possible_keys	key	ref	rows	extra
SIMPLE	b	TweetEventAssociation	TweetEventAssociation	NULL	126,201	A*
SIMPLE	a	tweetid_UNIQUE	tweetid_UNIQUE	twitterbucketsort.b.tweetid	A*	NULL

*A – Using where ; Using temporary, Using filesort

When assessing the problem, I ran the query used in the top ten user feature when there were 120,007 records in the "tweetevent" and "tweets" table. Figure 8 contains the profiling result of this query and Table 9 contains the execution plan of this query. Profiling shows that the most time is taken sending data because many records are read and hence much data is sent and the execution plan shows that it iterates through 126,201

[10] http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html

rows, which is a large number and unnecessary because it iterates over the same rows time and again and repeats the same work.

Status	Time
Starting	66 µs
Checking Permissions	4 µs
Checking Permissions	2 µs
Opening Tables	42 µs
Init	46 µs
System Lock	8 µs
Optimizing	11 µs
Statistics	27 µs
Preparing	16 µs
Creating Tmp Table	277 µs
Sorting Result	6 µs
Executing	2 µs
Sending Data	18.9 s
Creating Sort Index	6.6 ms
End	11 µs
Removing Tmp Table	433 µs
End	4 µs
Query End	6 µs
Closing Tables	24 µs
Freeing Items	101 µs
Logging Slow Query	3 µs
Cleaning Up	115 µs

Figure 8: Profiling Information of Query in Table 7

3.3.5.2 Solution

To improve the top ten user feature, I first added an index on the “eventid” column to make the query faster. But indexing alone would not solve the problem as the number of records increases. A complete revamp of the existing solution was needed.

I used the same approach as I used for the top twenty hashtags feature. The idea is simple–compute the top ten users incrementally from the incoming tweets. Using this method, PMA would not need to query all of the tweets again for each report. The approach is to store a user's name in a dedicated table for this event and increment the tweet count for that user when a tweet from that user is collected. When a PIO wants to generate a report for a particular event, the algorithm will just query to this dedicated table and get the results immediately. The name of this dedicated table is "topuser[eventid]" where eventid is the ID of the event for which it stores the user name with its tweet count.

After adding an index on the "eventid" column, I ran the old query for determining the top ten users. I analyzed the query using MYSQL EXPLAIN (see Table 10) and found that the number of rows checked by the query reduced by more than 50%, which is a huge improvement. Now, the data to be read has reduced drastically and hence less data is now getting sent which has reduced the overall time (see the profiling information for this query in Figure 9).

Table 10: MySQL EXPLAIN of Query in Table 7 after Adding an Index to "eventid"

Select_type	table	Possible_keys	key	ref	rows	extra
SIMPLE	b	eventid, tweetid	eventid	CONST	59,984	A*
SIMPLE	a	tweetid_UNIQUE	tweetid_UNIQUE	twitterbucketsort.b.tweetid	A*	NULL

* A – Using where ; Using temporary, Using filesort

Status	Time
Starting	48 µs
Checking Permissions	4 µs
Checking Permissions	3 µs
Opening Tables	33 µs
Init	37 µs
System Lock	8 µs
Optimizing	10 µs
Statistics	176.7 ms
Preparing	62 µs
Creating Tmp Table	886 µs
Sorting Result	16 µs
Executing	4 µs
Sending Data	2.6 s
Creating Sort Index	7.4 ms
End	11 µs
Removing Tmp Table	511 µs
End	6 µs
Query End	8 µs
Closing Tables	15 µs
Freeing Items	113 µs
Cleaning Up	21 µs

Figure 9: Profiling Information for the Query in Table 7 after Adding an Index to "eventid"

Even though adding an index to the “eventid” column created marked improvements over not using an index (by comparing Table 9 and Table 10), I found that changing the way that PMA calculates the top ten users to use a dedicated table would improve performance even more. The addition of an index to the “eventid” column will, however, optimize other queries that use the “tweetevent” table.

```
select username, count from topuser34 order by count desc limit 20
```

Figure 10: Query to Retrieve Top Users in the New Approach

Therefore, I chose to use the same approach used in prior sections and create a dedicated table, named "topuser[eventid]," where the users and the number of tweets they have sent are set at the time when tweets are collected. The table "topuser[eventid]" also has an index on column "count," making this query efficient even if there are many rows in the "topuser[eventid]" table.

3.3.5.3 Possible Drawbacks

The possible drawbacks are the same as those encountered for the top twenty hashtags feature—namely, an increase in the number of tables and an increase in the computation cost to pre-compute required data.

3.4 Summary

In this chapter, first describe the environment in which I conducted this research as well as the research scope. Following which, I improved the referential integrity of PMA's databases by adding indexes and foreign key constraints. For each index and foreign key I considered adding, I considered the benefits and disadvantages of adding them. I only added indexes and foreign keys when the benefits outweighed the disadvantages. Careful assignment of indexes and foreign keys has greatly increased the efficiency of PMA's database. These efficiency gains will be analyzed and reported in the next chapter.

Next, I analyzed the report generation feature of PMA and implemented new approaches to make the feature more scalable and efficient. In most cases, these new approaches to report generation included shifting away from running complex database queries for each report toward creating smaller tables that track information needed for a single report feature. In all cases, the new approaches have increased performance. Finally, I outline the drawbacks of each new approach and suggest solutions or reasons why the benefits outweigh the drawbacks. In the next chapter, I evaluate the performance of the improvements I have made to report generation and compare it with the old approach.

CHAPTER 4

PERFORMANCE EVALUATION

In this chapter, I evaluate the performance of PMA after implementing the scalability changes and new strategies in chapter 3. This chapter describes the way testing was done. It also reports the results of the testing in various graphs that help to visualize the efficiency of the application after applying the changes to optimize and scale the application.

The main changes in the new approach were made to the reporting algorithm in the ReportController.php file and in the "consume" script, which collects tweets from the Twitter API and puts them into the PMA database. The new approach is basically a streaming solution that updates the various tables for report generation while inserting each tweet into the database, thereby storing and pre-computing the data for a faster response. The "consume" script has been updated to accommodate this streaming solution and the reporting implementation has also been updated to access the data through the new approach.

I evaluated the performance of the new approach by comparing it to the old approach. I logged the timing information into a database table of the execution of both, the old and the new approaches, and then analyzed and compared them.

I am logging the execution time of all the new section of the code and comparing it with the execution time of the old code used for achieving the same functionality. I have given specific codes for each report feature and the files in which the changes were made (see Table 11).

Apart from comparing the new approach against the old approach, I also suspected that since the reporting algorithm is putting the data into a word file, the generation of this word file may significantly contribute to execution time. Hence, I consider and analyze this possibility in my study.

Table 11: Test Codes for the Various Testing Conditions

	Changes in the report controller file	Putting the data into word	Changes in the consume script
Category count	11	12	13
Twitter daily activity	21	22	23
Top 10 user	31	32	33
Top 20 hashtags	41	42	43
Top 20 retweets	51	52	53
Total time	61	62	63

Table 11 contains the codes I used for each of the execution time tests. The rows represent the reporting features and the column represent the operations to be measured. For the older approach a “0” is appended to the end of the codes in Table 11, and for the new approach “1” is appended to the end of the codes. The timing information for the respective codes is then logged into a database table for analysis. There are two tables that store the logging information for the report:

1) performanceaudits – for the logging information of the report controller.
2) consumescriptperformance – for the logging information of the consume script.

4.1 Performance Data Collection

I used a stopwatch in the code to record the execution time for each feature. The time required to put the timing data into the database along with any other operations unrelated to the feature, have not been included in the execution times.

These tests were conducted under the same conditions described in section 3.1. Care was taken that other applications were not running in the background so that they would not use processor time or resources. However, I could not stop applications required by the operating system from running and utilizing processor time and resources. For this reason, the timing data might be a little skewed.

4.2 Execution Time of Report Features in the Report Controller

In this section, I give the execution time in the Report Controller for the different report features, based on an increasing number of tweets. I collected these execution times both before and after I made the improvements to the Report Controller described in Chapter 3. Each timing data point in the following graphs represents an average of ten executions.

4.2.1 Top Twenty Retweets

Figure 11 shows the execution time for the "top twenty retweets" feature for an increasing number of tweets. It also shows the difference in the execution time between the old and the new approaches. The execution time for the older approach increases linearly as the number of tweets increases. In contrast, the new approach updates the table of "top twenty retweets" when inserting tweets, so we see an almost constant execution time. Since PMA reads the already computed table for the "top twenty retweets" report feature, the execution time for this feature is independent of the number of tweets. In the older approach we were computing the feature on the fly and hence the execution time was dependent on the number of tweets. The table that stores the retweet count will not have records more than 30 rows at any given time. Hence, reading data from that table will never be an expensive operation.

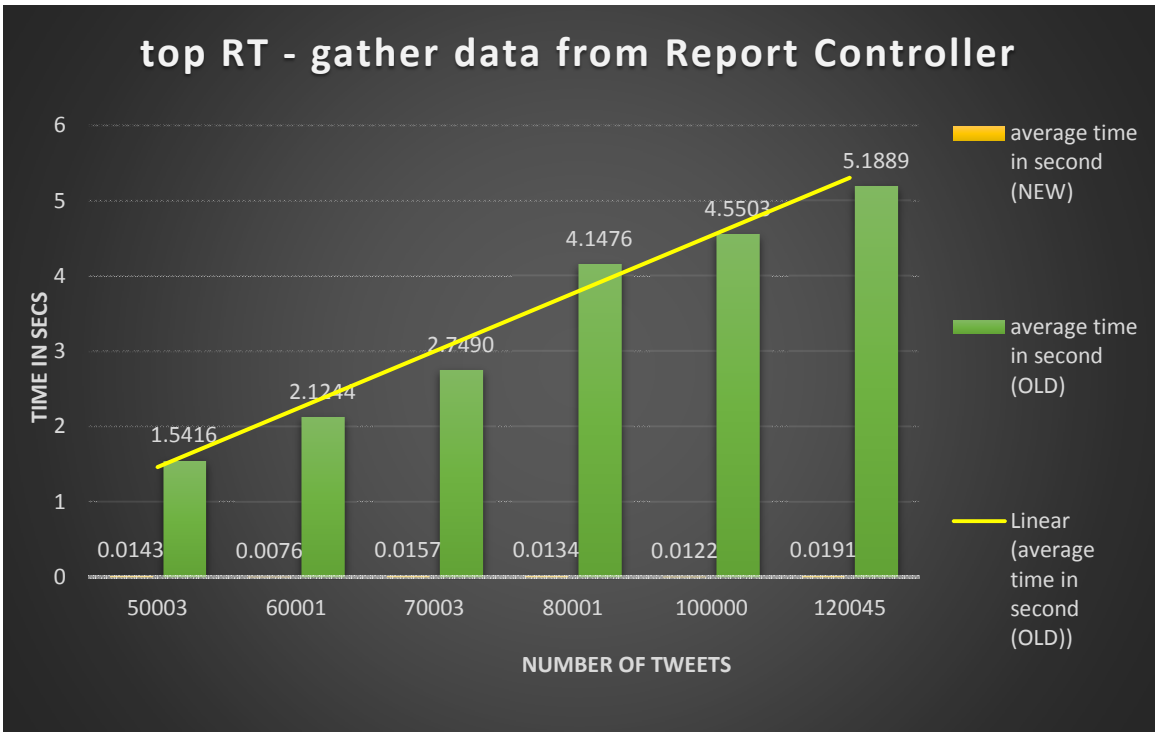

Figure 11: Execution Time for Reporting the Top Twenty Retweets

4.2.2 Top Twenty Hashtags

In Figure 12 we again see that the execution time for the old approach of calculating the "top twenty hashtags" increases linearly with an increase in the number of tweets. When an event has 120,000 tweets, PMA takes five seconds to generate the report for the "top twenty hashtags" feature, which is a significant amount of time. Compared to the old approach, the new approach is much more efficient. It only takes 0.19 seconds for 120,000 tweets. However, the execution time for this feature is not constant because the table that stores the hashtag counts (maxhash[eventid]) can have any number of records and this number depends on the number of hashtags contained in an event's tweets.

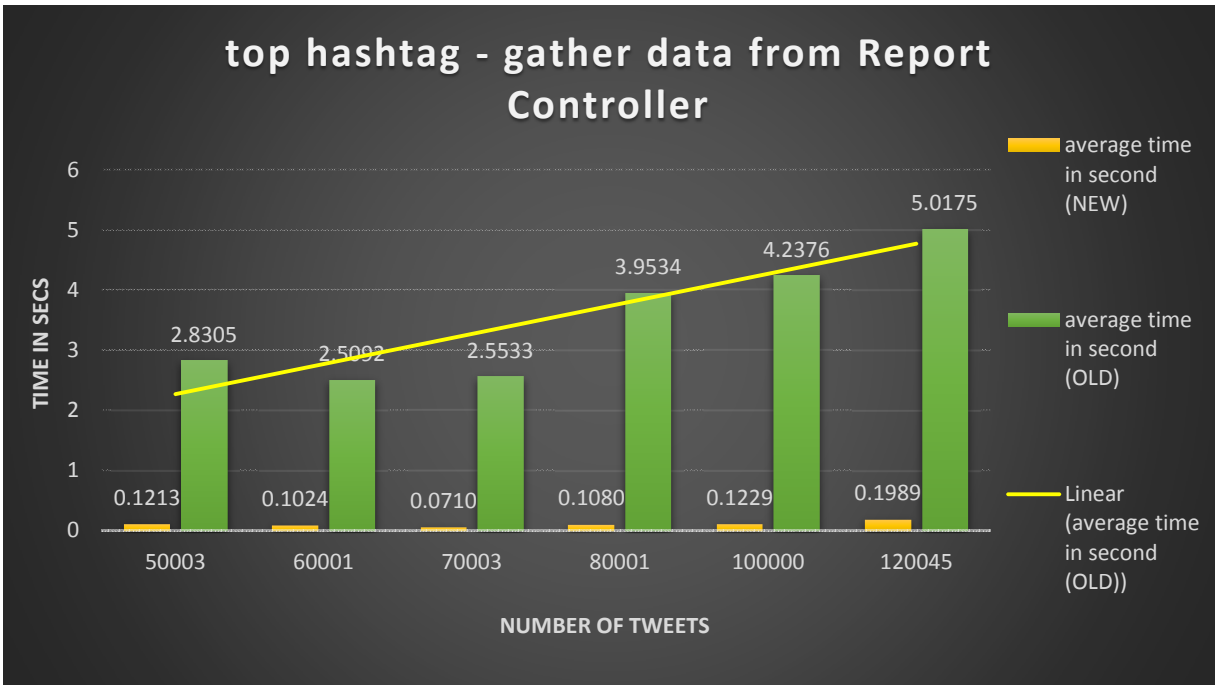

Figure 12: Execution Time for Reporting the Top Twenty Hashtags

The size of this table is likely to increase as new tweets are collected. But, it is unlikely that the size of the table will increase so much that this feature takes a significant amount of time to execute. Also, the indexes added to this table will help to keep the computation time efficient. Figure 13 charts the read time for retrieving the "top twenty hashtags" from the table "maxhash[eventid]" as the number of hashtags in the table increases.

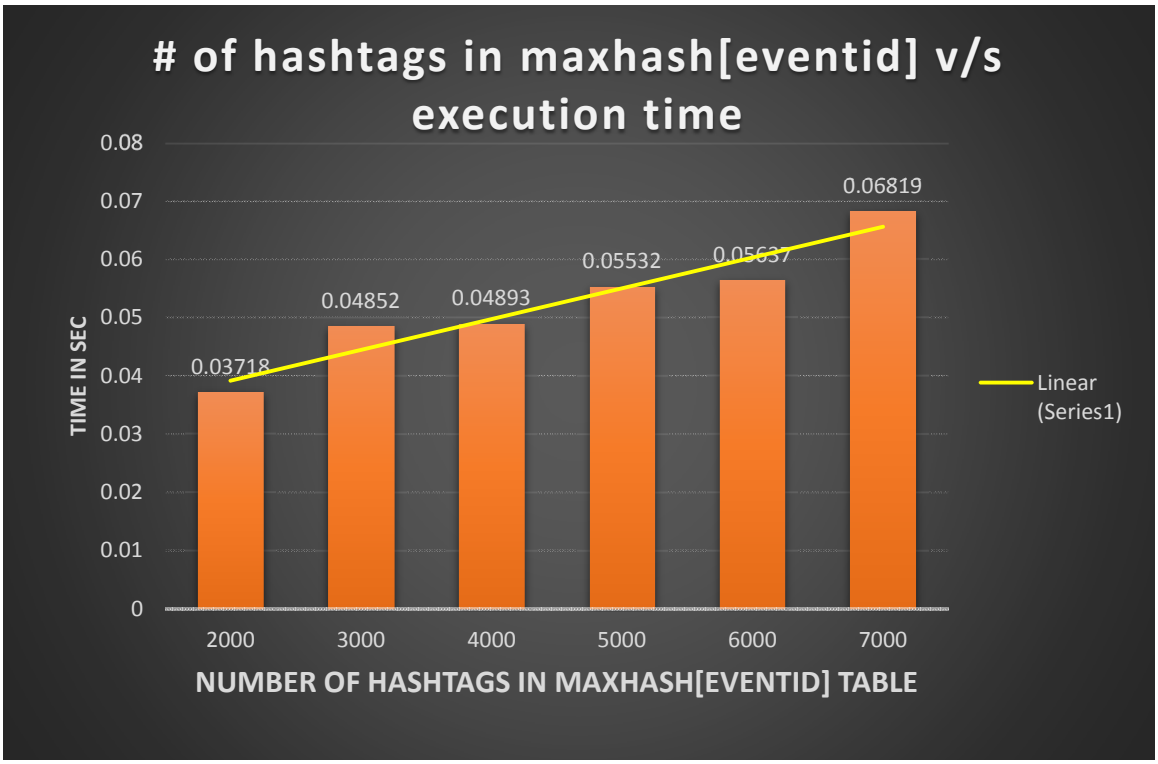

Figure 13: Time to Get the Top Twenty Hashtags from "maxhash[eventid]" Table

4.2.3 Twitter Category Report

It is difficult to test the execution time for generating the “twitter category report” as the number of tweets increases because it requires significant effort manually adding categories and tweets to categories. Since the testing would involve too much manual work (e.g., putting 100k tweets manually into a category), I have not run any timed tests to demonstrate to prove that the new approach is more efficient than the old. Rather, I support my claim that the new approach is more efficient than the old approach with a theoretical explanation.

To generate the “twitter category report” using the old approach, PMA would query the “coding” table for all the categories in this event and their respective tweet counts. This execution time is dependent on the number of tweets in each category and

the time to perform an ORDER BY operation. In the new approach, the query reads the already stored count of tweets for a category from the "categorycount" table whose event ID matches with the ID of the current event, a constant time lookup. The new approach is more efficient because it will always have a constant time lookup, whereas the execution time of the old approach increases as the number of categories and tweets in those categories increases.

4.2.4 Twitter Daily Activity

Figure 14 shows the execution time for the "daily twitter activity" report feature as the number of tweets increase, and it compares the old and new approaches for implementing this feature. This feature is the most expensive feature to compute in the old approach and was most responsible for slow report generation. However, after making improvements, the execution time significantly dropped. For instance, when PMA contained approximately 120,000 tweets, the old execution time was 10.47 seconds versus 0.0782 seconds for the new approach. This improvement in the new approach results because PMA is now only doing a read query to get the necessary count. The new approach's efficiency is further strengthened by proper indexing and foreign key constraints on the table "twitterdailyactivity."

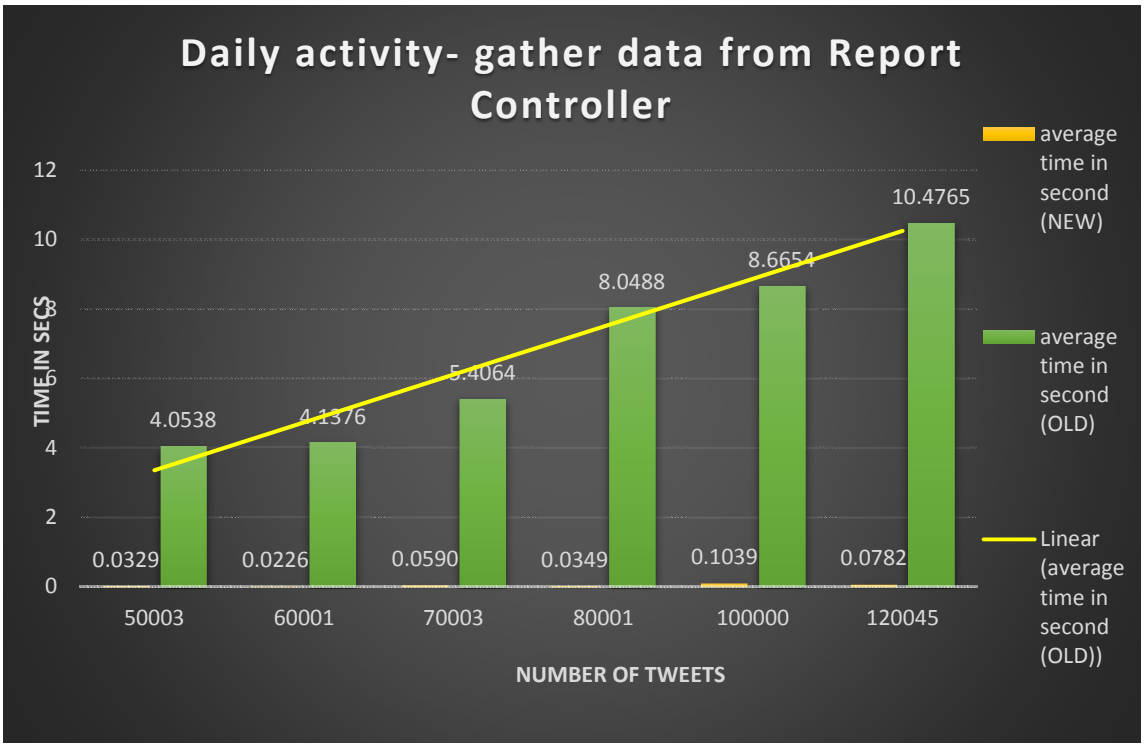

Figure 14: Execution Time for Reporting the Daily Twitter Activity

4.2.5 Top Ten Users

This feature reports the top ten Twitter users who tweeted the most tweets for a particular event. This report can help PIOs discover the most active Twitter users for an event.

The execution time for the old approach to this feature increases linearly as the number of tweets increases. The new approach stores the users and their tweet count for an event in a table. Then to retrieve the tweet counts, PMA only has to read the data from the table and does not have to compute the tweet counts, which results in a quicker response time.

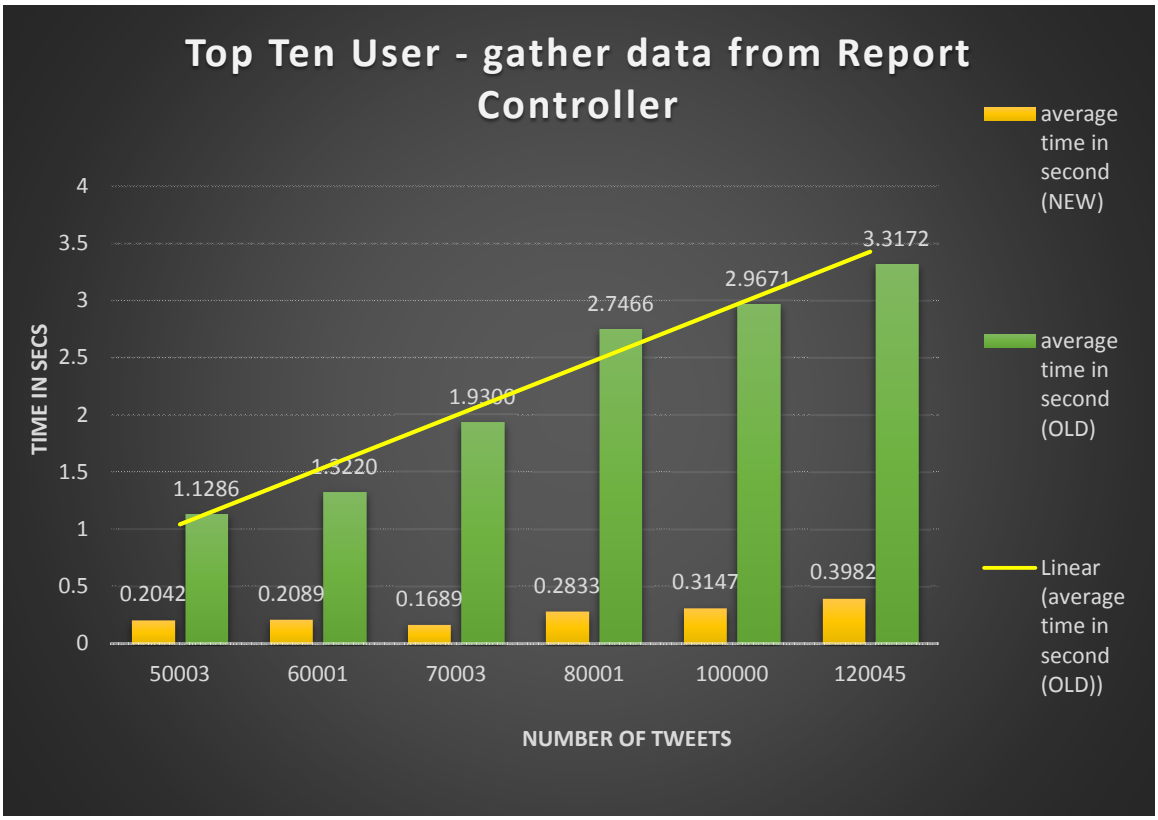

Figure 15: Execution Time for Reporting the Top Ten Users

The execution time for the new approach will increase gradually because the "topuser[eventid]" table that stores the tweet count data will grow as PMA collects more tweets for an event. However, because I have assigned indexes to the "topuser[eventid]" table, the response time will minimally increase over time (see Figure 16).

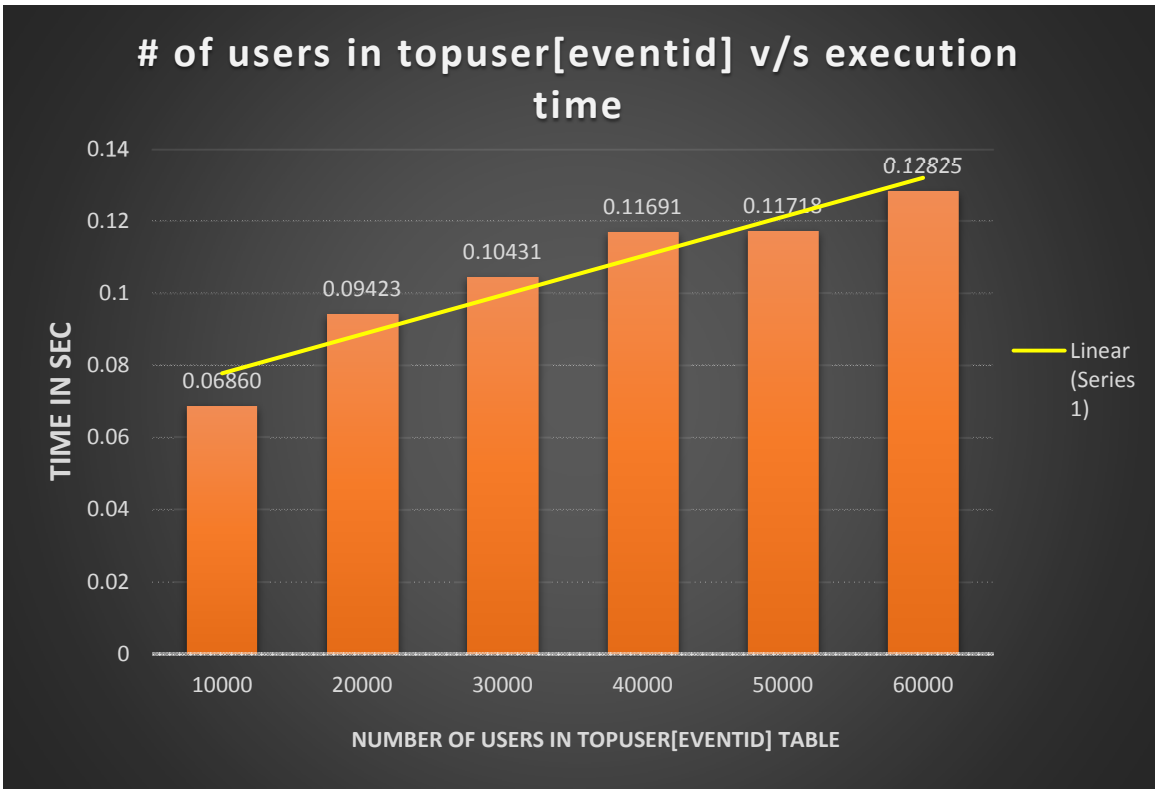

Figure 16: Time to Get the Top Ten Users from "topuser [eventid]" Table

4.3 Execution Time of Report Preprocessing in the Consume Script and Possible Drawbacks

As part of the new approach for the report generation feature, PMA is calculating and storing the required data beforehand, when new tweets are collected. This approached has added extra processing and computation time to the consume script–the script which puts the collected tweets into the database along with other required data for PMA to function. Below, I explore the extra time it takes to preprocess tweets in this new approach.

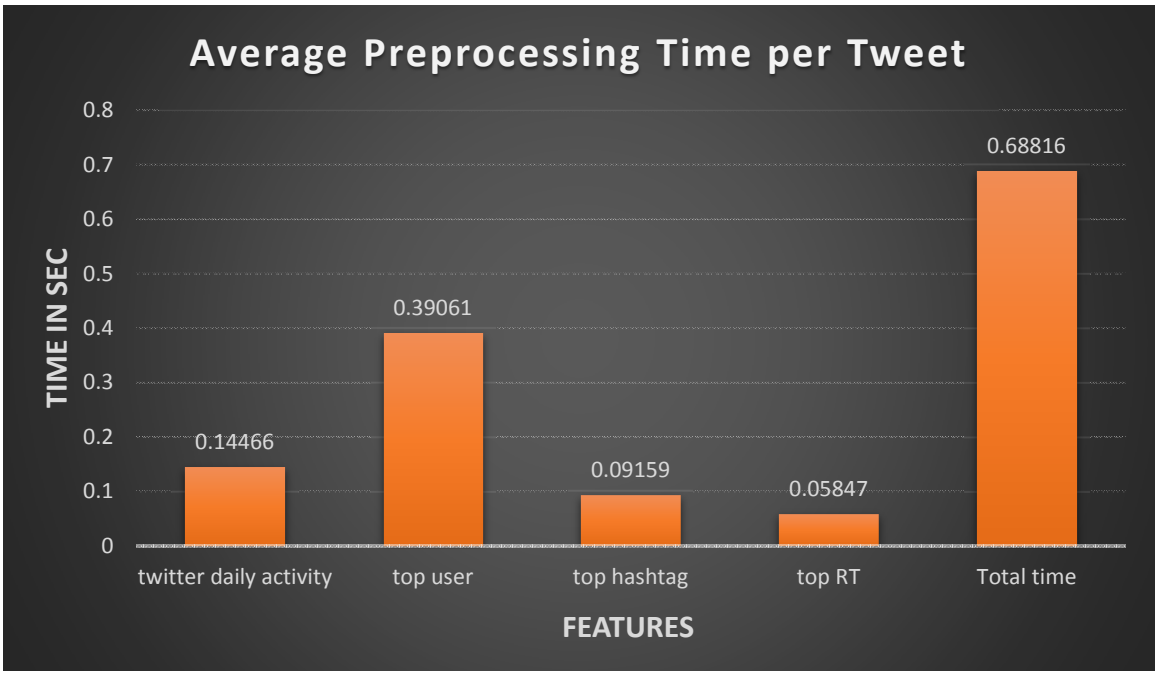

Figure 17: Execution Time for Preprocessing a Tweet by Report Feature

To better understand the execution time effects of the new approach, I calculated the average time for preprocessing a tweet for each report feature (see Figure 17). I also found the execution time per tweet for the old approach. In the old approach there was no preprocessing, so I simply calculated the total time for the script to run per tweet. These timings were stored in a table called "consumescriptperformance" with the task IDs mentioned in Table 11. I ran these tests on an instance of PMA that had 120,007 tweets. Table 12 reports the average execution time for the old and new approaches.

Table 12: Average Execution Time for Consume Script Per Tweet

	Average Time Per Tweet (in Seconds)
New Approach	0.6882
Old Approach	0.2850
Difference	0.4031

In Table 12, the execution time of the consume script per tweet for the old approach is much less than the new one. This time increase could result in slow insertion of tweets into the database and the problem could become severe if the number of events or search terms significantly increases. However, PMA can generate reports much more quickly with these changes. Ultimately, to save time in one area of the application, more time must be spent in another.

Even if we use the old consume script, it will run slower with an increasing number of events and search terms but much later than if we run the new consume script. Thus, in the long term, the problem of decreased performance will not be avoided even if PMA runs the older version of the consume script.

One way to mitigate this performance problem is to run PMA on a more powerful server. In the next chapter, I explore hosting PMA using Amazon Web Services (AWS). AWS offers the ability to scale resources according to the needs of the application, which promises to further increase the performance efficiency of PMA.

4.4 Summary

In this chapter, I evaluated a new approach, discussed in Chapter 3, for creating reports in PMA. I presented graphs that compared the execution time of the old approach to the new approach, and found that the new approach offers significant performance gains for all features when creating a report. While the new approach improves report generation performance, it does slow the collection and processing of newly collected tweets. The next chapter will investigate ways of improving PMA's performance through the use of AWS.

CHAPTER 5

DEPLOYING PMA ON THE CLOUD

This chapter briefly defines cloud computing and specifically outlines the features and benefits of using the cloud computing services of AWS (Amazon Web Services). Next, I offer reasons for deploying PMA on AWS and I describe the deployment process.

PMA is a web application that can be used by PIOs during an emergency to gather, retrieve, sort, and analyze social media data. During an emergency, there tends to be an abundance of relevant social media data. Conversely, when there is no emergency, social media data are sparse. Ideally, PMA should function efficiently under both conditions. However, emergencies are unpredictable, making it difficult to know when a surge in requests to the PMA database and application will peak and when it will recede. Cloud computing services can help optimize resources for such fluctuating needs.

5.1 Cloud Computing

A cloud is a large-scale, publicly accessible collection of computation, storage, and networking resources. These resources are typically allocated via web service calls and are available for short- or long-term use in exchange for payment based on the resources consumed.

A cloud consists of a multi-user environment that can handle a large number of users simultaneously. It is responsible for managing and verifying user identity, tracking allocation of resources to the users, providing exclusive access to the resources owned by each user, and preventing one user from interfering with other users.

Cloud computing services provide tools to prepare for and cope with web traffic surges. Figure 18 shows a typical cloud computing architecture. By moving PMA over to

a cloud computing environment, the application's reliability (i.e., ability to handle traffic surges) will be improved.

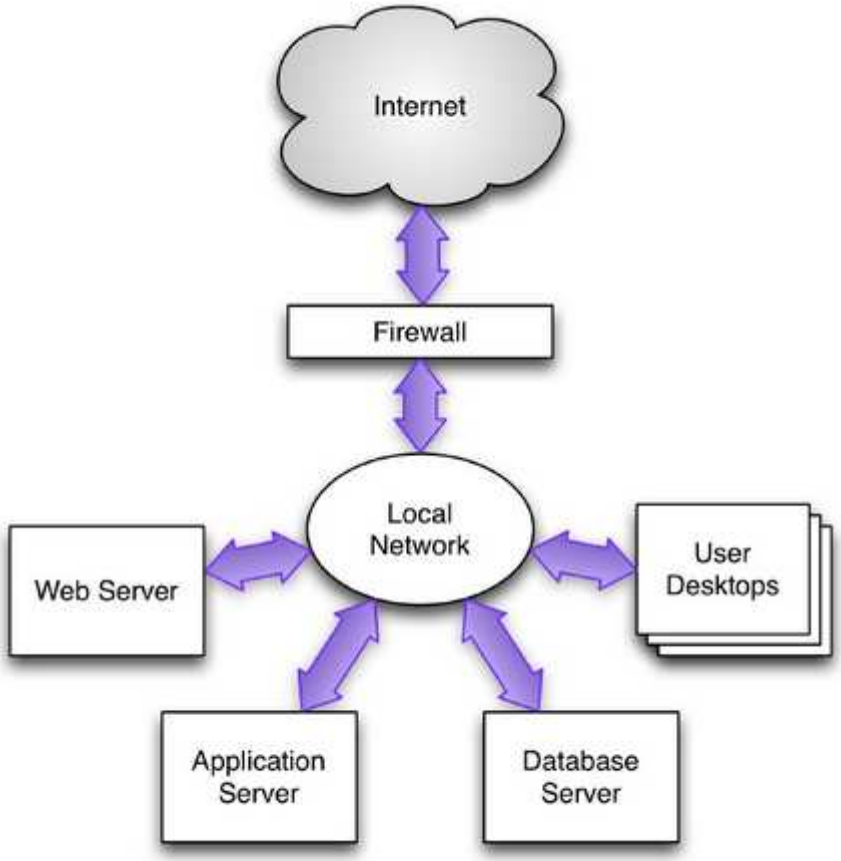

Figure 18: Cloud Computing Architecture

5.2 Cloud Computing Criteria

PMA satisfies many of the criteria for a good cloud computing application. These criteria are listed below:

5.2.1 Constant Usage Over Time

This criteria describes applications that have little variation in usage or load from day to day or hour to hour. PMA has little variation in use during non-emergency periods.

5.2.2 Spiked Internal Load

A spiked internal load is typical in environments where people can submit large-scale, one-time jobs for processing. In this situation, the demand is usually unpredictable. An expensive job that PMA performs for PIOs is the report generation feature.

5.2.3 Spiked External Load

This situation can be seen as a previously unknown site suddenly becomes popular. In PMA, this situation occurs when a PIO creates a new emergency event and tracks social media data associated with that event until the event is over.

5.2.4 Steady Growth

When an application or web site matures, it typically grows in size and uses more resources. As more PIOs start to use PMA to track events, more search terms will be stored in the database and more tweets will be collected. Hence, I expect PMA will have steady growth over time.

5.3 Amazon Web Services (AWS)

AWS is a highly reputed cloud services provider that offers a variety of choices for storage, processing, operating system, and cost models [8]. With AWS a consumer can quickly purchase new hardware to handle unexpected need. Once these resources are no longer needed, they can be returned to the cloud. AWS also offers consumers near-infinite scalability, where surges and growth can be handled as they occur. In addition, AWS allows consumers to experiment without making long-term commitments to hardware.

5.4 Deploying PMA on AWS

AWS has a large set of services that a user can use for deploying an application. I chose to use Elastic Compute Cloud (EC2) for deploying PMA. Amazon EC2 is a web service (virtual machine) that provides resizable computing capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

AWS is a pay-as-you-go (pay only for the resources you use) service that provides a free tier[11]. The free tier lets consumers use resources under a certain limit without any charges. Since the amount of resources provided in the free tier are sufficient for PMA, I decided to use the free tier to deploy this application.

The procedure for deploying PMA to AWS is explained in brief, as there were many small steps that needed to be taken. I first created a general purpose, micro instance (t2.micro[12]) of EC2. I then generated a private key for this instance so that I could access this instance remotely using a SSH client (e.g.: PuTTY). Amazon EC2 provides various operating systems for creating an EC2 instance and I selected Ubuntu 14.04.1 LTS for this instance's environment. The next steps were to install php, MySQL, and an Apache server on this instance. Since the application uses the Yii framework, I also had to configure Yii on this instance.[13] Next, I migrated all the PMA code into the directory where Apache serves web applications using g a simple open source tool called WinSCP.

After successfully transferring PMA's code files, the database had to be uploaded to the instance. I also installed phpMyAdmin (a GUI based tool to access MySQL databases on the local system) to help me administer the MYSQL database. Using

---

[11] http://aws.amazon.com/ec2/pricing/
[12] http://aws.amazon.com/ec2/instance-types/
[13] http://www.faceyspacey.com/resources/linux/server-setup-3-installing-yii-and-your-application

phpMyAdmin, I migrated my entire database to the instance and the the application was up and running.

To access deployed applications on the Internet, Amazon provides a feature called "elastic IPs" to assign an IP address to the application server. Therefore, my last step was to assign an elastic IP to the PMA instance so that it could be accessed on the Internet.

5.5 Summary

In this chapter, I described the motivation behind deploying PMA on a cloud-based architecture rather than a traditional web server. I also outlined the reasons for and benefits of choosing AWS. Lastly, I briefly described the steps I took to deploy the application on an Amazon EC2 instance. In the next chapter, I describe how I tested this application with users and the findings from this testing.

CHAPTER 6

USER STUDY

The last component of this research will test PMA with real users so that I can find and correct usability issues with the application. In this chapter, I first discuss previous usability enhancements that I made to the application prior to user testing. These enhancements were identified in prior research on PMA [3]. Next, I outline the testing procedure and discuss the findings of the user study.

6.1 Previous Usability Enhancement Work

During the spring semester of 2014, I made several usability improvements and bug fixes to PMA to ready the application for field deployment. Below is a list of the changes I made:

1) Implemented Role Based Access Control (RBAC)
2) Made the admin page more user-friendly by making the display more meaningful
3) Provided capability to add users to an event through the user interface
4) Enhanced the archive mode so that it displays the correct category name and deletes a category appropriately

While doing this work, it became apparent that the application was unable to handle large number of tweets. So even after addressing important usability concerns, the application was not ready to be tested by and deployed to potential users. The work I conducted in Chapters 3, 4, and 5 addressed these scalability issues so that PMA could be deployed and tested in more realistic settings. I now describe the user study I conducted.

6.2 User Study Testing Procedures

For this user testing, I recruited three PIO participants to step through a set of tasks designed to test and improve the usability of PMA. Each test session lasted approximately one hour. Since the application resided on a web server (AWS) at the time of testing, participants used a computer connected to the Internet to log into the application and test it.

6.2.1 Pre-Test

Much of this research revolves around making the application handle large amount of data, so to test its scalability and performance, I preloaded PMA with 200,000 tweets. This load helped me understand the behavior of the application when working with large data sets.

Before testing PMA, I asked each participant a few questions about their background (refer to the questionnaire in Appendix B). The information obtained from this pre-test interview helped me understand the experience and skill level of the participant in regards to emergency management, computer applications, and social media.

6.2.2 PMA Testing

The PMA testing component of this user study, began with a brief overview of PMA's features and the testing procedure. Following which, I gave the participant a list of tasks to complete using PMA (see Appendix C). These tasks focused on testing the major features of PMA. These features and the method of testing are discussed in more detail below:

- *Event Feature:* This feature deals with creating and maintaining events. An event corresponds to an emergency event that a PIO would want to collect and monitor social media data around. To track an event with PMA, a user must provide search terms that PMA then uses to fetch tweets relating to those search terms. Tasks for testing this feature consisted of creating an event, managing (i.e., creating, replacing, updating, deleting) search terms for an event and adding new users to an event.
- *Streaming View Feature:* In this view, PMA displays the tweets it collects in real-time. This view also allows a user to categorize the displayed tweets by dragging and dropping them into various categories. These categories can have rules so that the tweets automatically get categorized into a specified category. Tasks for this feature include creating categories, putting tweets in categories, creating rules for categories, and examining the user interface.
- *Archive View Feature:* This view enables the user to sift through all the tweets collected for that event. The user can categorize tweets, filter and search them, save them to their local drive, or just view them. The user can also generate a report based on the collected tweets. The tasks for this feature involve searching for a tweet, creating a report with different parameters, saving all the tweets to a user's local drive, and closely examining the user interface for intuitiveness and user friendliness.

PIO participants were encouraged to use the "think-aloud" method as they stepped through the tasks; that is, they were asked to explain their thought process so I could understand how they perceived and thought about PMA. While the participants

stepped through the tasks, I observed them and took notes. These sessions were also audio-recorded for future reference and analysis.

6.2.3 Post-Test

After the participants complete the list of tasks, I conducted a short post-interview with the participants to better understand their experience with the application (see Appendix B). I asked about which features they liked or disliked. I also investigated whether they would use this application for their work and emergency events.

6.3 User Study Findings

6.3.1 Participants

All three participants were Public Information Officers (PIOs) for local emergency response organizations.[14] Each participant has worked for more than 5 years as a PIO in their current position and gets called out to perform their duties as a PIO very often. One participant frequently uses social media to gather information and track public opinion and sentiment. The other two participants have not used social media as often to track the public, but they have used services like Google Alerts to track the online media around an emergency event. All participants described using Twitter and Facebook to distribute information to the public during an event.

6.3.2 Results

Overall, the participants found the application intuitive, but some of the features were more difficult to understand. For example, some points of confusion include the ability to create rules for categorizing tweets and the difference between the streaming

[14] The organizations have not been identified here to protect the anonymity of the participants.

and the archive view. Also, the names of some of the fields were confusing and did not help the user understand the fields' purpose. One participant found that the form fields to create an event were ambiguous in that the participant did not understand whether PMA was asking for personal information or the information for an event. This same participant did not understand the format of the information to be entered in the "location" field of the same form. The other field that was confusing on the create event page was the "summary" field. All participants did not know what to do with this field. On the search term page, one participant could not figure out how to edit a search term as there was no "edit button" like there was a "delete button" for deleting a search term. Editing the search term by double-clicking was not intuitive for that participant but was fairly easy to understand for the rest of the participants. Another participant was worried about the tweets getting backed up in the streaming view queue and wanted a solution for that.

All of the users liked the reporting feature and the ability to categorize tweets with rules. Participants all understood the streaming view and how to set up search terms for that view. The participants also liked the parameters displayed for generating reports as it is helpful for their work, however, one participant suggested that having an hourly report for the "Twitter Daily Activity" feature would be helpful. The "export" feature (the ability to download collected tweets to a excel spreadsheet) was also greatly appreciated by all participants.

I asked participants if they would like to give those that they might work with access to the events they work with in PMA. Generally, the participants felt that this would be a good idea as it would allow them to share the social media monitoring work

around an event. However, the application would need to be tested with more users in a multi-user setting to understand more completely how this feature would work.

For more detailed descriptions (notes and observations) of each participant's user test, refer to Appendix D, Appendix E, and Appendix F.

6.3.3 Recommendations

Based on the user study, I offer recommendations to improve PMA and address the issues discovered through user testing. Of these recommendations, some have been addressed here and others have been postponed as future work.

6.3.3.1 Addressed Here

1) The form fields for creating a new event were unclear and ambiguous. To make the fields more clear, as shown in Figure 19, I have placed "placeholders" in the text boxes.

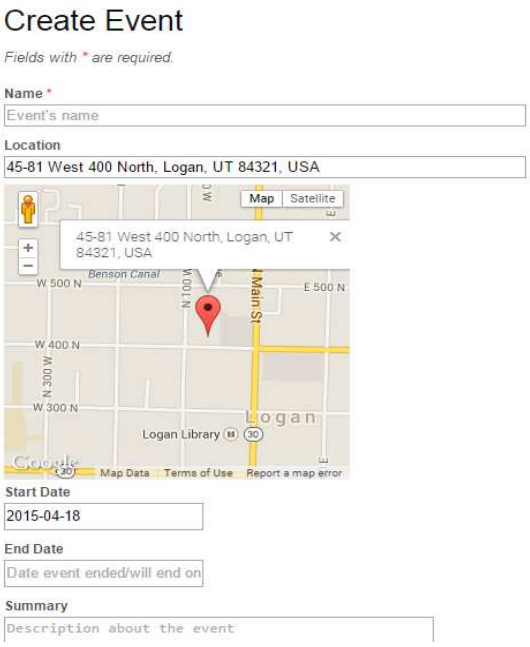

Figure 19: Placeholders and Google Map to Fill in "Location"

2) One user suggested that they would like to have the ability to generate an Hourly Twitter Activity report. To provide this capability, I have made changes to the database and the collection script so that PMA records the tweet count for an event on a half-hour basis. A simple aggregate function can be applied to get the hourly tweet counts.
3) One user wanted an easier way to populate the "location" field in the Event Creation form. I addressed this feedback by combining this field with the Google Maps API. The User can click anywhere on the map and the address or the coordinate of that point will populate the "location" textbox, as shown in the Figure 19.
4) Some features were difficult to understand quickly, such as creating rules for categorization, exporting tweet data for an event, and generating a report. To address this problem, I have included a "Help" link in the navigation bar, which will help users better understand these features and how to use them. I also created a demo video that is included in the "Help" section that demonstrates how to use PMA's features. This video serves as a visual aid that supplements the textual description found in the "Help" link.

6.3.3.2 Future Work

1) One user explained how he often got irrelevant results when searching Twitter data using key words. One solution that might help return more accurate search results it to provide the ability to search for tweets based on location (for more detail refer to section 7.1).

2) The ability to edit a search term in PMA was misunderstood by one user. To make this ability more intuitive, we could either add text explaining how to edit the search term or we could add an “edit” button.

6.4 Summary

This chapter describes how I tested PMA with PIO users. I first outline how the testing sessions were conducted. Next, I gave background information for each test participant and briefly described the results of the testing. Based on these results, I recommend changes that will improve PMA (some have been implemented here, and others have been left for future work).

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this research, I improved and evaluated PMA—an application designed to help PIOs retrieve, sort, store, monitor, and report social media activity, Before I started this work, PMA had many usability issues and it was unable to handle large amounts of data. I addressed these issues and limitations in this thesis research.

I investigated PMA's volume limitations by performing tests with increasing numbers of tweets (data), and observing PMA's performance. Through these tests, I found the features that were hampering performance. For instance, the report generation feature was computationally expensive, taking much time and resources. Also, the streaming view behaved abnormally when the number of tweets increased. After some exploration, I found that PMA's database was not efficient with respect to indexes and foreign key constraints. I also observed that the algorithm for generating a report could be improved. The algorithm would perform the same tasks again and again over the same data whenever a report was generated. I implemented a solution based on the idea of "computing once and using multiple times." I stored the results incrementally and changed the queries for generating a report so that PMA does not perform complex queries, rather, it just reads the data that has been computed before.

I compared the new PMA (with the changes described above) to the old application and discovered a considerable performance boost in generating reports due to the changes made in the report generating algorithm, adding appropriate indexes to the databases tables, and assigning appropriate foreign key constraints. To further improve performance, I deployed the application on AWS servers. The performance testing

indicated that PMA was ready to handle larger data sets and to be used in realistic settings.

Finally, I conducted a usability study with PIOs to test the newly modified application. The application was well received by test participants as they all agreed that the application would be useful in their work as PIOs. The usability testing also revealed areas of improvement for PMA. Many of these issues I addressed in this work, but there are also several issues that were beyond the scope of this work and consequently, have been left as future work. I outline these issues and possible solutions in the following section.

While this research addressed many limitations of PMA, the application still has much room for improvement (i.e., adding new features and further improving existing functionality based on users' needs and requirements). Below, I describe future work for PMA.

7.1 Collecting Tweets Based on Location

The current approach for collecting tweets uses search terms. A user provides a search term and a script collects tweets that contain that search term, provided the search terms is active for the event. A new future feature will let users collect tweets based on a location. Using a search term, PMA might retrieve tweets that are irrelevant to the emergency event, whereas, if PMA retrieves tweets based on location, the retrieved tweets are more likely to be relevant to the event. Figure 20 shows how the "collecting tweets by search terms" and "collecting tweets by location" features can co-exist.

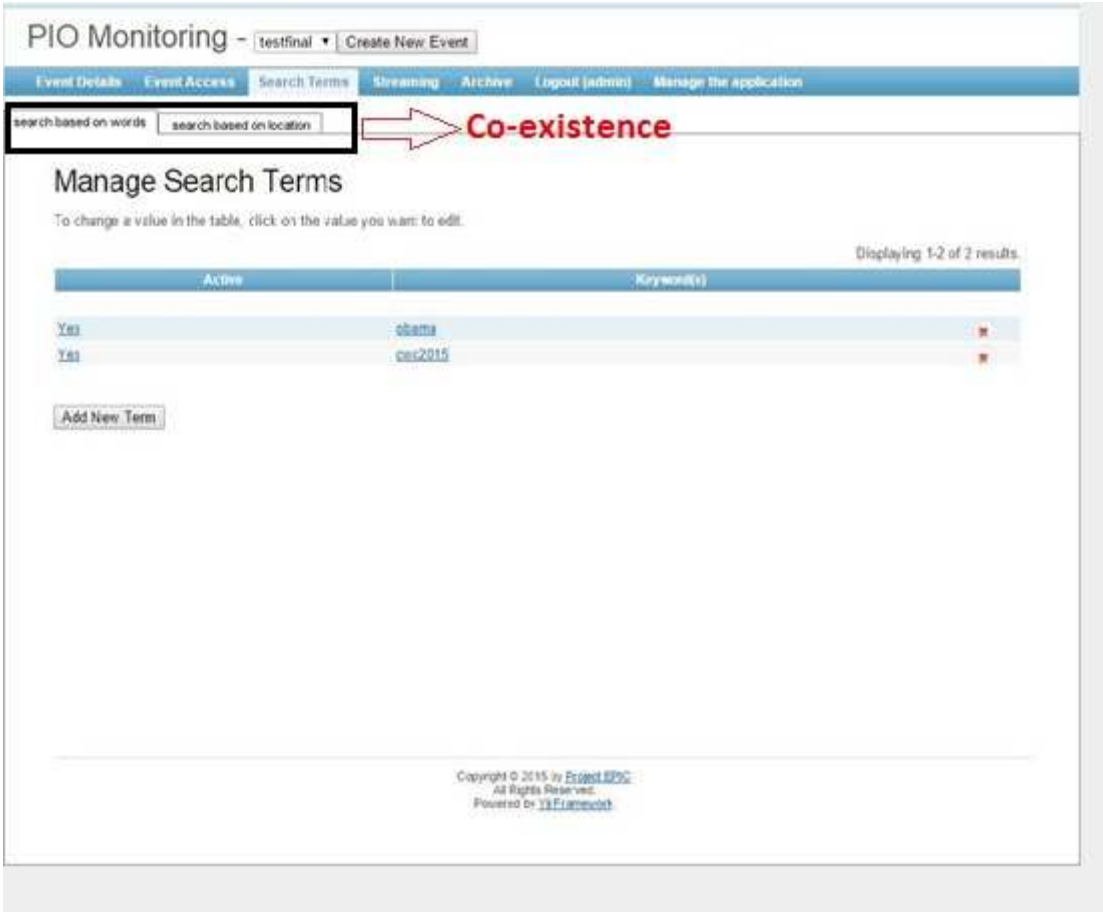

Figure 20: GUI Showing Future Work to Incorporate Location-based Tweet Searching

PIOs can be provided with a map interface in the application and can click anywhere on the map. The location of the clicked point can be saved by using map APIs. PIOs will also have to specify the radius so that the Twitter API can be used to collect all the tweets within the radius of the selected point on the map. This form is shown in Figure 21.

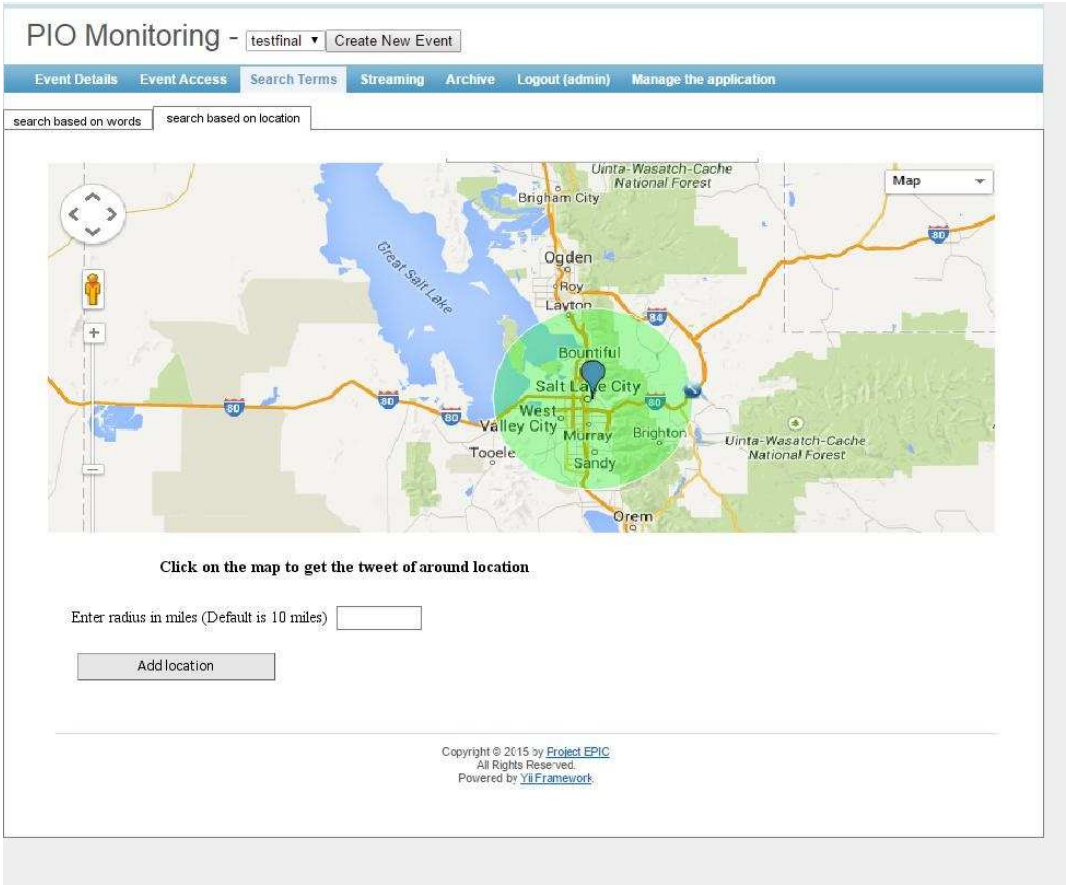

Figure 21: GUI for Location-based Tweets Search

When adding a location-based search feature, we need to decide how it will work with the existing search term feature. One way would be to have checkboxes or radio buttons in the streaming view for showing only those tweets retrieved based on search terms, for streaming only those tweets which are retrieved based on location, and for streaming those tweets retrieved based on both, the search terms and location, respectively (see Figure 22). Another challenge would be to optimize the performance of displaying the tweets as the user might change his options frequently and hence loading those tweets should not take a long time.

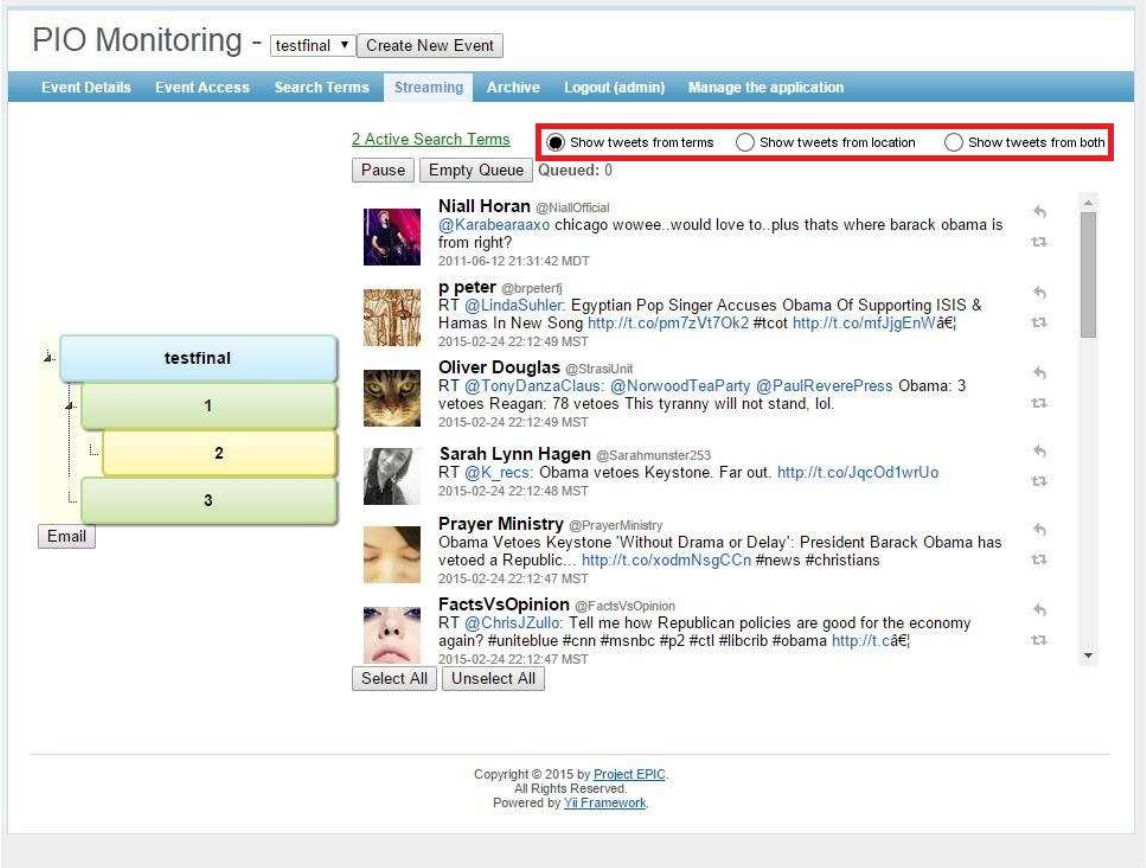

Figure 22: GUI of Streaming View to Accommodate Location-based Tweet Search

7.2 Adjusting the Time Zone

This application is expected to be used by PIOs located throughout the United States (US). Since the US has multiple time zones, the application should be able to display time and date related data with respect to the PIO's time zone. One solution is to store the user's time zone. When it comes to displaying date and time data, some formula can then be applied to the data according to the user's time zone and then display it accordingly.

7.3 Active/Non-Active Mode of Search Terms

One of the shortcomings of this application is that the consume script will keep on collecting tweets for an event based on the search terms as long as the search term is active, even when the event is no longer being tracked.

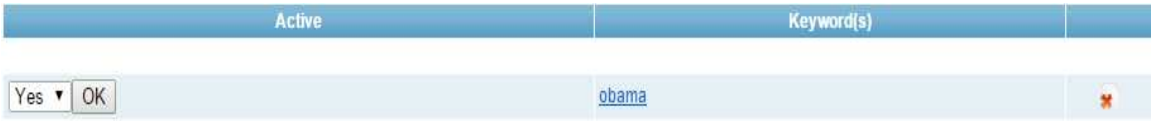

Figure 23 GUI of Setting Search Terms

The problem with this is that the database will get populated with data that is not needed. This, in turn, will cause the other algorithms that access the data to slow down. The consume script will also slow down as it is retrieves tweets for more search terms. One solution is that the user should deactivate search terms when they are done tracking an event. However, the application should not rely on the user to remember or to have time to disable inactive events and search terms. Instead, a solution is to have a field in the “events” table which stores a timestamp representing when the event was last accessed. There has to be another procedure which regularly checks for the “last accessed” fields of the event and if the event was last accessed more than 10 (this can be changed) days ago, all the search terms for that event should be made inactive.

7.4 Archiving Data Periodically

Over time, the data for various events will pile up if the respective search terms of that event are active but the event is no longer used. The other possibility is that an event has collected a lot of data and now the event is closed and all its search term are inactive.

In these cases, the database size could grow so much that any database queries will slow down.

A solution to this problem is to archive data for events that are no longer in use. Users with Admin privileges will have the right to declare an event “active” or “inactive.” A script can then be written which checks for inactive events and transfers all the data related to those events to a data warehouse. This warehouse will seldom be queried. It will be queried only if a user wants to retrieve historical data. This way the size of the database which is accessed regularly will hopefully remain manageable.

7.5 Efficient architecture for deploying the application using AWS

At present, PMA's EC2 instance has a web server as well as a database server. AWS provides a dedicated web server called Relational Database Service (RDS[15]). This service is eligible to use for free tier users. Using this service will facilitate the transfer of

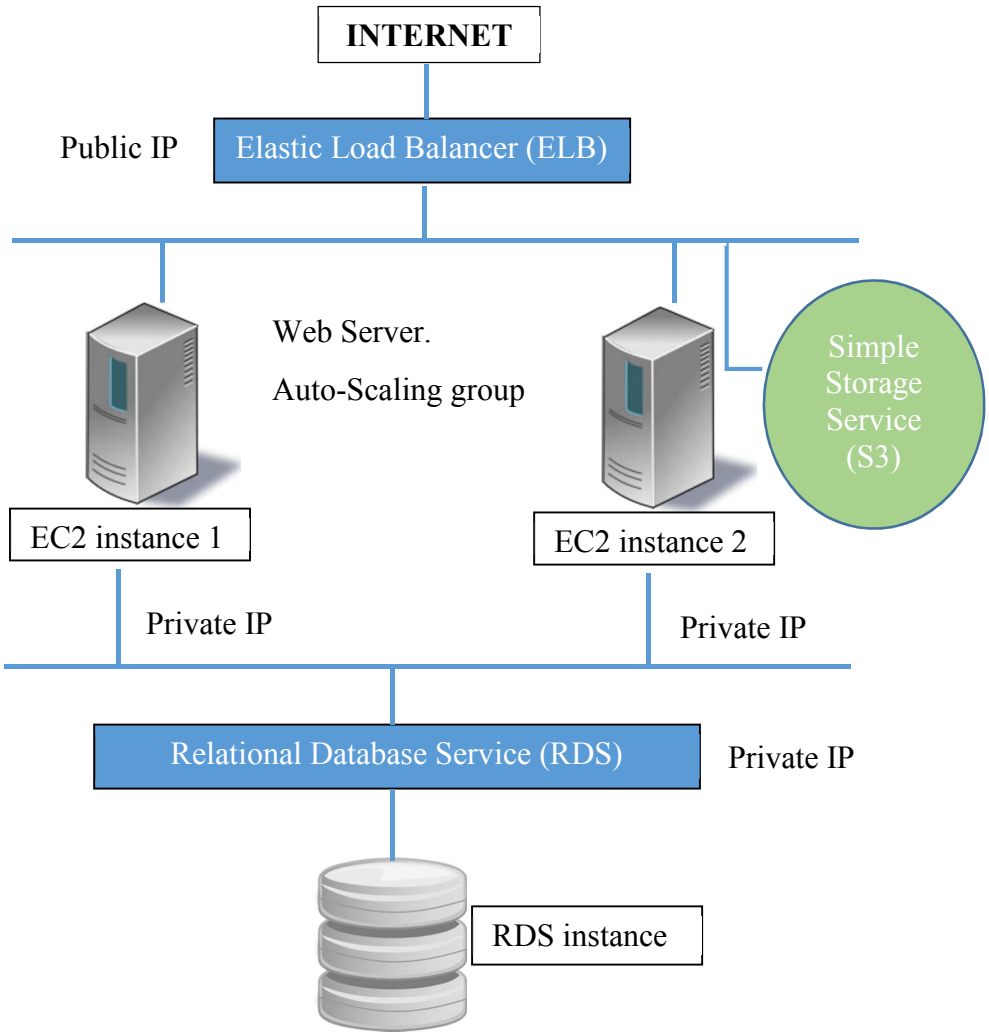

Figure 24: Proposed architecture for deployment on AWS

[15] http://aws.amazon.com/rds/

the computation load to a dedicated database server rather than the MySQL server residing in the EC2 instance. This service also provides benefits[16] like improved scalability and flexible storage. Having a separate database server will enable the developer to create multiple EC2 instances over different geographical locations connected to this RDS instance. This will provide better availability as well as faster reads and writes from and to the database. Amazon also provides a load balancer that can be applied to two or more EC2 instances, which balances the load on these instances for better throughput. I recommend a new architecture for AWS as below for deploying this application.

[16] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

REFERENCES

[1] Amanda L. Hughes and Leysia Palen, "The evolving role of the public information officer: An examination of social media in emergency management." *J. Homeland Security and Emergency Management*. vol. 9, no.1, pp. 1-12, 2012.

[2] N. A. Karlova and K. E. Fisher. A social diffusion model of misinformation and disinformation for understanding human information behavior. Information Res., vol. 18, no.1. [Online]. Available: http://informationr.net/ir/18-1/paper573.html

[3] Hughes, A.L., S. Peterson, and L. Palen, "Social media in emergency management." In *Issues in Disaster Science and Management: A Critical Dialogue Between Scientists and Emergency Managers,* J.E. Trainor and T. Subbio, eds. FEMA in Higher Education Program, 2014, pp. 2-15.

[4] Amanda L. Hughes et al. Online Public Communications by Police & Fire Services during the 2012 Hurricane Sandy. In *Proc. of the 32nd International Conf. on Human Factors in Computing Systems*. Toronto, Canada, 2014, pp. 1-4.

[5] Lise A. St. Denis, Amanda L. Hughes, and Leysia Palen. Trial by fire: The deployment of trusted tigital volunteers in the 2011 Shadow Lake Fire. In *Proc. of the Information Systems for Crisis Response and Management Conf.* (ISCRAM), Vancouver, Canada, 2012, pp. 1-7

[6] A. L. Hughes. "Supporting the social media needs of emergency public information officers with human-centered design and development." PhD dissertation. University of Colorado at Boulder, 2012.

[7] [Online] Available: http://www.yiiframework.com/about/

[8] "Welcome to cloud computing." In J. Barr, *Host Your Website on the Cloud:Amazon Web Services Made Easy*, SitePoint Pty Ltd, 2010, pp. 6-20.

APPENDICES

Appendix A Previous PMA Database Schema

Appendix B PMA User Testing Interview Questions

Pre-Testing Interview:

1) What is your current position?
2) How long have you worked at this position?
3) What are you responsibilities in this position?
4) How often are you called to ask as a public information officer?
5) Have you used social media in your role as a public information officer? How?

Post-Testing Interview:

1) What features did you like the most in the application?
2) What features did you dislike in the application?
3) Could you use this application in your work? Why or why not?
4) Could you use this application in an emergency situation? Why or why not?

Appendix C Task List for PMA User Testing

Testing "Event" feature:

1) Create an event
2) Adding search term to an event
3) Edit a search term (deleting the term and making it non-active)
4) Add another user to this event

Testing streaming view:

1) Create categories
2) Create rules for categories
3) Put tweets in category

Testing archive view:

1) Search a tweet/use the search feature
2) Retrieve all the tweets for this event and save it in the local drive
3) Create a report of the event

Appendix D User Study with Participant P1

Tasks	**Action by P1**	**Comments by P1**	**Observations**
Creating event	Clicked on "create event" button	"I don't know what goes into the summary field. Since it is not required I would leave it blank and proceed"	Had doubts about how long will the event last. So P1 did not know what date to put. Summary field is a little confusing to understand.
Adding search term	Figured out how to insert search terms easily by looking at the navigation bar	Had questions about how the matching is done. "Fairly easy to set up events and start streaming".	
Adding user to an event	Skimmed through the navigation bar, clicked on "event access" and then clicked on "add user"	"Collaboration on the same event happens regularly, and we used to share username and passwords for the same account with different people in order to collaborate. Since I trust the people I work with, giving other users minimal authority on the event is not required, a full access is to them on an event is good for collaboration. But in some community full access might not be helpful"	Understands this feature very well.
Create categories	Reading the text on the view, and figuring out what it means.	"that's easy and intuitive"	P1 did not have any trouble setting up the category. Understands the "pause" and "empty queue" buttons in the streaming view.

			Setting up event is easy and intuitive
Putting tweets in categories	P1 dragged the tweets into categories	"that's super easy, and super intuitive. I like this feature, I also like how you can click on tweets and then click on the category you want to put the selected tweets in, it is intuitive to right click and create categories"	P1 liked this feature and figured it out almost instantly
Creating rules for categories	Reading the text on the view, and figuring out what it means.	"I don't know what that (the text in the rules' view) means. If I was in a middle of an event I wouldn't do this and my queue is filling up!!!" Talked about the way P1 can use this feature in times of emergency	Creating rules not super intuitive and the P1 took some time to understand the way to set it up. P1 understood it pretty well and correctly.
Searching tweets using search feature	Goes to the correct textbox to input a search term and clicks "enter"	"Does it search tweets only in the current category or all the categories?" had different questions about how does the search filter works when you have multiple search terms.	Understands the search features and had good questions about it. P1 did understand how to search easily
Saving tweets locally	Clicks on "export"	"EXPORT !!"	Easily figured out the export feature and liked it very much.
Creating report of the event	Clicks on "report"	"Previously we did not used to create reports because we were not able to capture the Twitter	Understands the feature's parameters and makes sense to P1.

		information. We used employ primitive ways though, like taking a screenshot. But this feature is great. Knowing popular hashtags, top twenty retweets will be interesting to know. These parameters are very helpful”	P1 likes the report feature.

Appendix E User Study with Participant P2

Tasks	**Action by P1**	**Comments by P2**	**Observations**
Creating event	Clicked on "create event" button and entered the required information	"What should I put as end date? End date is necessary because there will be a bell curve kind of distribution for the activity for the event. I like the name field as we name a fire "	
Adding search term	Tried some of the links on the navigation bar until found the correct page. Clicked on the correct button	"the tweets matching these search terms will show up?"	
Adding user to an event	Skimmed through the navigation bar, clicked on "event access" and then clicked on "add user"	"I would probably go to event access page and there it is – add another user and click on. During big events, we work with other departments for their resources, but I may not want to give full access (admin rights) to the event. Right now collaborators know the username and password of other departments' twitter account"	Understands this feature very well.
Editing a search term	Clicks on the "cross" to delete a term. Double clicks on the term to rename	"You could just double click to edit, and over here it says active or inactive. When it is active, the application will look for those	Editing a term was very easy and intuitive for P2 and P2 understood the various functionality on the search terms' page

		words in the tweets and vice versa”	
Create categories	Created categories by right clicking. Created sub categories to understand the levels	“I will just create the category by right clicking. Pause means if I want to pause the stream so that the tweet does not move down and resume means resuming the stream again. These buttons makes sense. Empty means dump it all”	P2 did not have any trouble setting up the category. Also understands the levelling and sub categories. The indentations helps to figure out the structure
Putting tweets in categories	P1 dragged the tweets into categories	“I will just grab one of these tweets and put it into the category”	P2 figured it out almost instantly
Creating rules for categories	Reading the text and making sense out of it	“When I put Logan Fire, the google alerts give me information about Logan Australia, Logan Boston and all sorts of irrelevant places. So when I put England in the text box, it will search all the tweets having England in it and put it into this folder”	Creating rules not super intuitive and the P1 took some time to understand the way to set it up. P1 understood it pretty well and correctly.
Searching tweets using search feature	Goes to the correct textbox to input a search term and clicks “enter”	Had different questions about how does the search filter works when you have multiple search terms.	Understands the search features and had good questions about it. P1 did understand how to search easily
Saving tweets locally	Clicks on “export”	“I am guessing you could export the tweets by email”	Could not clearly understand what export button does, but after a little

			explanation got the concept quickly
Creating report of the event	Clicks on “report”	“Top twitter users will give you the data of users and how many times did they tweet about your event. Daily is kind of similar, how many times people are talking about your event (during the specified date range), category report will give the number of tweets in different folders (categories), top twenty retweets means how many times the tweet you have put out, got retweeted. That’s pretty cool (says while reading the report)”	Understands the feature’s parameters and makes sense to P2.

Appendix F User Study with Participant P3

Tasks	**Action by P3**	**Comments by P3**	**Observations**
Creating event	Clicked on "create event" button and entered the required information	"Is the name field asking for my name or the name of the event? There has to be more details on how to enter a location like city-state, coordinates, street address. I will just put city and state. I am really not sure what to put in summary, I am going to skip that. The lesser information to enter the better"	P3 was confused whether the form asked personal information or about the event.
Adding search term	Went to correct tab and clicked on the correct button.	"I will go to the search terms tab and add the search term. Application will pull tweets relating to the search terms"	P3 understood this feature very well and knew what to do with it
Editing a search term	Clicks on the "cross" to delete a term. Double clicks on the term to rename	"I am not able to see any edit button so I will just delete a term and add the correct term again. I can see a little 'X' button at the side, so I assuming its delete "	Editing a search term was not very intuitive for P3 and was not able to figure out how to edit a term
Create categories	Struggled to create new category	"I don't know how to create new category"	Creating category was very difficult for P3
Creating rules for categories	Reading the text and making sense out of it	"I don't know how to assign rules"	P3 struggled with creating rules
Saving tweets locally	Clicks on "export"	"Wouldn't I have to export the report? That will	Could not clearly understand what export button does, but after a little

		export the report we just created”	explanation got the concept quickly
Creating report of the event	Clicks on “report”	“I don’t care about the retweets feature. I would certainly want daily twitter activity. The daily twitter activity gives you day to day activity. I also would want an hour to hour report with daily. An hourly report will be a very good idea. I think you are onto something“	Understands the feature’s parameters.