An Approach to a Disaster Management Information System

by

Mohammed Faisal Khan

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

UNIVERSITY OF NEW HAVEN

December 2018

©University of New Haven, 2018

Author . . . .
Department of Computer Science
May 1, 2018

Certified by. . . . .
Amir Esmailpour
Associate Professor
Thesis Supervisor

Accepted by . . . .
Ronald S. Harichandran
Chairman, Department Committee on Graduate Theses

An Approach to a Disaster Management Information System

by

Mohammed Faisal Khan

Submitted to the Department of Computer Science
on May 1, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

In this thesis, I designed and implemented a disaster management information system which collects data from social media platform like Twitter in real time using stream listener and stores the information in MongoDB, a NoSQL database, which is used to parse the tweets. Hadoop map and reduce was used to extract related and meaningful data from the tweets which can be used directly with the provided API, as well as effectuate various applications using Machine Learning algorithms for modeling the various disaster scenarios. The data extracted was used to provide information about the resources available during a disaster like shelter capacity, food provisions and safe zones.

Thesis Supervisor: Amir Esmailpour
Title: Associate Professor

Acknowledgments

I would greatly like to thank my supervisor, Dr. Amir Ismailpur for his support and guidance. Dr. Ismailpur gave me the freedom and flexibility to pursue my research and provided to me with resources to succeed in every regard. Additionally, I would like to thank Professor Barun Chandra for encouraging me to aim higher and reach my goals during the pursuit of my degree.

I would like to thank my friends; Apoorva for his advisory role and Rebecca for being a great company at Starbucks, and not being annoyed by all the grammatical questions that popped up. Also, I would like to thank Linda, for being a amazing friend through the trying times and Danielle, for her joyful company, apart from her "out of this world" proofreading skills.

Furthermore, I am grateful to work with a number of professors, including Dr. David Eggert, Dr. Ted Markowitz and Professor Dennis Clemence for imparting knowledge that has helped me grow my technical skills. Additionally, I would like to thank Dr. Leon Weinmann, for being a tremendous supervisor during my time working under him in the Center for Learning Resources at the University of New Haven.

Finally, I would like to thank my parents, for their unconditional support during this journey and my brother, Faizan and sister, Sobia to always keep me motivated. I could not have done this without them.

Contents

List of Figures

Chapter 1

Introduction

A Disaster Management Information System (DMIS) can be a useful tool to extract data from social media platforms and parse information to save time during a crisis, which can mean life or death in such situations.

The subsequent chapters describe the literature, architecture and design of a disaster management system, and the motivations behind the design decisions made.

1.1 Inspiration

Disasters can strike without notice and no amount of preparation can compete with the knowledge to overcome the effects of a disaster during and after the incident. For example, a flash flood can happen with little to no warning and knowing safe zones during this disaster can help to save lives. The aim is to propose and implement a platform that can extract and store data from multiple social media platforms, news websites, and various other disaster-related data stores, and process this data to present valuable information in real-time.

The advent of digital age and resulting ever-connected society that shares localized information that was not available in the past has presented us with an opportunity to tap into these resources to gather very fine-tuned information of the incidents happening on the ground. Integrating this information in conjunction with the traditional ways of collecting data (e.g. Geo-spatial data, sensor data, satellite imagery) can provide better choices in decision making during a catastrophe.

1.2 Leveraging existing data

When disaster strikes, there's often no time to sift through data, much less try to analyze it. One way to avoid such a situation is to imagine all the possibilities of an emergency situation and line up the data beforehand. Unfortunately, that is not always possible, as you never know what data you will need until a situation occurs. The DMIS aims to do that by storing the needed information beforehand and processing the information so that it saves time during a disaster.

The storing of information and processing can be done since the abundance of data available on-line on Twitter and other social media platforms [12]. Websites like Twitter provide Application Programming Interfaces (APIs) to stream their data which can be filtered and saved in a database. The data can then be analyzed and filtered to gather useful information that can be acted upon.

1.3 Scope of data collection

The mushrooming of social networking has made the general population inadvertently participate in data collection. This is comparable to having resources on the ground, providing access to information in real-time [6].

The DMIS platform aims to collect data obtained from these sources on a streaming basis. Consequently, as the data is being obtained from a large pool of diverse information, the gathered data has to be specifically filtered for disaster-related information.

The information generated by the various participants can be classified into four disaster management phases, as illustrated in Figure 1.1: mitigation, preparedness, response, and recovery [4]. Mitigation involves the undertaking of steps to reduce the impact of a disaster on the loss of life and property before it occurs. This is done by meticulously planning infrastructures and strengthening them with additional efforts to make the community more resilient to catastrophic events. Preparedness is concerned with formulation of plans and protocols to follow during a disaster event. The goal is to be ready for a catastrophic event which includes developing response mechanisms, procedures, and strategies that can be adhered to during an event, while also educating the communities with these response mechanisms. In addition to these efforts, there is the setting up of warning systems, shelters and storing of reserves such as food, water, medicines and other equipment and essentials that will be needed during a crisis.

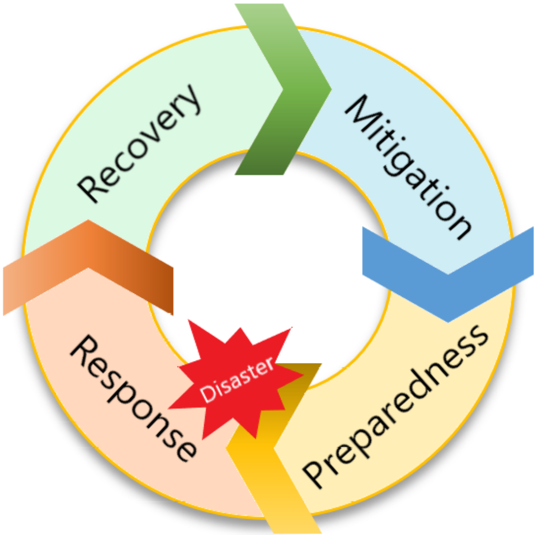


Figure 1-1: Phases of disaster management.

The data during the mitigation and preparedness phases ranges from response plans, emergency procedures, records for training exercises, and available resources. The response phase is triggered when a catastrophe occurs. This phase deals with the immediate assistance to save lives and improve health and morale of the affected population. Assistance includes providing safe zones and shelters, preservation of property, and provides information such as places to avoid, evacuation plans, extraction zones and protocols to follow.

The aftermath of a disaster is when the recovery phase starts. This is done by providing assistance to bring normalcy back, including repairing of damaged structures, clearing of debris, helping to resume business operations and getting the population back on their feet. Examples of data generated during the response and recovery phases consists of incident reports, lessons learned and use of the resulting information to make better disaster plans.

The data generated during the response phase is very crucial and if this data is carefully extrapolated and presented in real-time, it can make a difference during an incident. The approach proposed in this study aims to deliver this data efficiently with solutions to various challenges arising in the development of the system. Additionally, the scope of the system encompasses to be a knowledge base before, during and after a disaster. This can done by collecting data during all the phases of the disaster but effectively delivering during the response phase.

1.4 Glimpse into the system

Recent advancements in cloud computing, Big Data and NoSQL databases as well as the rising popularity of data science, machine learning and deep learning techniques have changed how data is being captured, stored and analyzed. NoSQL solutions are especially popular in Web applications [17], that include Facebook, Twitter, and Google as well as popular news websites like CNN and New York Times. However, the usage of cloud computing technologies and NoSQL solutions in addition to data science algorithms have been not fully explored in solutions to a disaster management system.

Benefits of storing and analyzing disaster related data in a cloud environment can have advantages [13] such as:

- *High Availaibility.*

  In a cloud environment the data is made redundant by using replication techniques where the data is stored in separate locations which are ofter geographically apart across large distances. This leads to data persistence even if a local data center fails as a switchover can occur almost instantaneously, resuming normal operation of the system without any delay.

- *Scalability and elasticity.*

  The amount of data being captured can be massive, also the computational power required to process this data can also vary greatly at a particular time a cloud based solution can adapt storage and processing resources on demand based on real-time needs and priorities. Additionally the data can be distributed heterogeneously on servers that can be run in parallel to increase performance of the system.

- *Cost Effective.*

  The initial investment to start and running the system on a cloud platform can be minimal. Additionally the system can be expanded by adding new nodes as and when needed.

On the other hand there are many advantages of using NoSQL data stores for disaster data management, including:

- *Flexible data structure.*

  Since the data collection includes data from various sources this makes the disaster data extremely diverse making it challenging to store information in a predetermined data structure, NoSQL data stores make it possible to store a variety of data in the same database.

- *Horizontal scalability.*

  NoSQL data stores were explicitly designed to handle large amounts of data by making it easy to scale on an as needed basis. This can be done easily by adding additional nodes to increase storage and handling.

- *Performance.*

  NoSQL data stores are faster on simple read/write operations. They are designed to do parallel processing on different nodes compared to relational databases, giving them an edge in performance oriented applications.

The increase in popularity of data science techniques has demonstrated the power of building complex quantitative algorithms to organize and synthesize large amounts of information used to answer questions and develop strategies to tackle complex problems in faster and efficient way. Advantages to using data science to filter and analyze the data being stored include:

- *Faster processing.*

  Using machine learning algorithms based on proven statistical models the data processing can be speedy.

- *Easy classification.*

  Sorting through the variety of the data being collected can be a daunting task as the data being stored is structured, semi-structured and unstructured.This data can be easily filtered by using classifiers, saving time and decreasing the overhead of searching through all the data in the data store.

1.5 Thesis Structure

The chapters in this thesis are organized as follows:

- **Chapter 2** describes the concepts and technologies relevant to this study: Big Data, cloud computing, NoSQL data stores and Machine Learning. Big Data is introduced as the data is gathered and stored from diverse sources. Also, since the platform designed is cloud-based, the nature of cloud computing model is delved into. Next, the storage model incorporated in the system; NoSQL is introduced. Also, different data models are discussed in reference to DMIS platform. Finally, machine learning is introduced and its applications in DMIS.

- **Chapter 3** defines the DMIS platform, listing the specific technologies used to implement the platform. Here, data from twitter is streamed and stored as 'proof of concept'. Next, the filtering techniques are described. Finally, the cloud based platform used to host the platform is introduced.

- **Chapter 4** concludes this study by discussing the contributions of this research. The main contribution is the implementation of the Disaster Information Management System as an information delivery service. The implementation is generic and can be used in other domains.

- **Chapter 5** focuses on the future aspects of the system, such as the potential of expanding of the system to increase the reach by creating portable and platform specific applications. Further work also includes working on data filtering techniques, verifying the data authenticity, etc.

Chapter 2

Background

This chapter introduces the main concepts and technologies relevant to this work: Section 2.1 introduces Big Data, Section 2.2 portrays cloud computing, and Section 2.3 presents the background on NoSQL data stores.

2.1 Big Data

The onset of information age brought on by the digital revolution has lead to a shift in how data is being produced and stored. Moreover, the recent evolution in web technologies leading to a transition from static websites to dynamic websites that are interacted with, and also with the recent proliferation of smart sensors and smart mobile devices that are always connected to the Internet, it has resulted in enormous amounts of data sets being generated that must be stored and processed. For example, YouTube has 1.9 billion monthly active users, with the website containing more than 5 billions videos with 300 hours of content being added per minute.

The traditional relational database management systems (RDBMS) were designed in an era when the available hardware, storage and processing techniques were comparatively different than they are at present. Hence, traditional approaches to storing and processing data are facing difficulties in meeting the requirements of Big Data processing. This also includes management, search, data transfer between storage nodes, data analysis and visualization.

"Big Data" is a terminology used to define large sets of data that can be both structured and unstructured, as well as complex for traditional data processing techniques. As stated by Beyer and Laney, Big Data has the following three main characteristics also known as the 3V's of Big Data:

- *Volume.*

  This refers to the quantity of raw data being generated or processed as well as being stored. Usually limiting data capture to a percentage of statistically relevant data.

- *Velocity.*

  This concerns with the speed at which the data is being sent, shared or processed. Additionally it involves dealing with data that is not static but moves, changes and grows as it is being generated.

- *Variety.*

  It is defined as the different types of data being produced or processed, this includes structured, unstructured and semi structured data and also data in different formats.

Occasionally, a fourth V is added: Veracity. Veracity deals with the quality of the data being captured that can lead to more accurate analysis which can help in making right decisions.

Big Data in reference to disaster data management, and more specifically in DMIS, refers to the large collections of disaster-related data sets being collected from various sources and disaster contributors. Integration of these data sets has to be done to provide efficient support for disaster management. Besides volume, the variety of the disaster-related data is a crucial problem that DMIS must tackle to be able to support the platform. Also to gather accurate information for decision making the veracity of the data has to be significantly taken into account.

Organizations are cognizant that Big Data has the potential to impact core business processes, provide competitive advantage, and increase revenues. Therefore, many enterprises are exploring ways to make better use of Big Data by analyzing them to find meaningful insights which can potentially usher to better business decisions and add value to their ventures. In the area of disaster management efficient use of available information has the potency to improve decision making which can mitigate the impact on human lives and property.

Participation adds value to data, the essence of Big Data is the collaboration of various sources to provide information that would be otherwise difficult to generate individually is of special interest to this research. As stated above, this can also add challenges of its own, however in the domain of disaster management collaboration among many participants is essential for successful response and recovery operations. In the proposed approach to DMIS platform data is sourced from various participants who share information on variety of data sources which are owned by different collaborators. This combined data is processed and analyzed to give useful insights.

2.2 Cloud Computing

The word cloud is used as a metaphor for the Internet and the term "Cloud Computing" is used to describe platforms for distributed computing over the Internet. According to the National Institute of Standards and Technology (NIST), cloud computing is

"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Since Big Data is taxing on resources due to its size, volume and velocity it induces growing computational demands on traditional computing techniques. Cloud computing platform assures that it can fulfill these demands by providing resources in a distributed environment of networked devices to offload the workload for efficient processing. Here, it is important to note the codependency of both these platforms and how it can be beneficial for faster processing leading to cloud computing being one of the key enabling techniques for handling data that is big in every sense of the word; thus it is the logical choice for implementing DMIS on one of these platforms.

Cloud computing is offered as a service, over the network, with varying degree of resources available for consumers to use as per requirements. Many IT companies offer public cloud service at a cost, including Amazon, Microsoft, Google and IBM to name a few. This makes it relatively easy to deploy and maintain the system without the overhead of managing the resources usually associated with setting up a platform at this scale. Alternatively, a private cloud can be setup which may have some advantages over the public model.

NIST has defined the essential characteristics of cloud computing as follows:

- *On-demand self-service.*

  Services can be unilaterally provisioned as required, without the need for human interaction.

- *Broad network access.*

  Services are available over the network and can be accessed through standard mechanisms over all platforms (e.g., mobile phones, tablets, laptops, and workstations).

- *Resource Pooling.*

  Resources are pooled, using multi-tenancy, to serve multiple consumers based on the demand, while different physical and virtual resources being dynamically alloted; all abstracted from the consumer.

- *Rapid elasticity.*

  Elastic provisioning of resources to rapidly scale capabilities outward or inward based on demand.

- *Measured service.*

  Consumer pays only for the resources used metered by the service provider using a pay-per-use pricing model.

As a result, cloud computing systems aim to provide the following benefits:

- *Availabilty.*

  The system is operational and accessible even in case of server, network, or data center failure making it highly available.

- *Scalibilty.*

  This means the ability to handle growing demands.

- *Elasticity.*

  This means to dynamically allocate resources based on demand or workload by scaling up or down.

- *Performance.*

  In a pay-per-use model, performance is directly correlated with cost.

- *Multi-tenancy.*

  Same hardware and software infrastructure is shared between multiple tenants (e.g., services, applications)

- *Fault tolerance.*

  This refers to the ability of a system to operate even in the presence of failures.

- *Load balancing.*

  Workload is automatically distributed among servers to achieve efficient resource utilization.

- *Ability to run on heterogeneous commodity servers.*

  Heterogeneity is almost unavoidable in infrastructures involving a large number of nodes.

In this research, all the benefits mentioned above make cloud computing a choice for managing disaster related data. However, it is important to note that scalability and availability make the decision even more ideal. Scalability facilitates the system to start with minimum resources and expand as the needs grow by adding heterogeneous nodes, while high availability ensures that the system remains accessible in case failures which is particularly of high importance in disaster management domain as it can be expected that disasters can lead to a variety of failures.

The three main cloud computing models that are offered are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The IaaS model provides resources such as servers (physical or virtual), networks, storage and operating systems. The PaaS model offers a higher-level environment and delivers a computing platform including data storage, programming languages, and Web application servers. Finally, the SaaS model provides on-demand software by offering access to software applications through the Internet.

Specialized variations of these three models have emerged, including Storage as a Service, Database as a Service, Security as a Service, Integration as a Service, and Testing as a Service. The DMIS platform approach proposed in this work applies IaaS model to host the environment.

Cloud computing is also not devoid of any challenges and since the approach proposed in this thesis draws on cloud computing platform, it is faced with the same challenges:

- **Security and privacy.**

  Public cloud services are offered by third party vendors where data is stored and processed on their premises, while resources being shared in a multi-tenant environment; thus, inducing risk to security and privacy of the data. This makes providing a satisfactory solution challenging as it needs to include both the services provider and the service consumer.

- **Customer lock-in.**

  Since different vendors offer proprietary products as solutions for various use cases, it makes it difficult to move from one cloud service provider to another. This lack of standardization make consumers vulnerable to price increases.

- **Data transfer challenges.**

  The locations of the data center and the consumer can result in significant network traffic which must be considered when evaluating performance and cost.

- **Legal issues.**

  Public cloud resources may reside in a geographical region with different security and privacy regulations than those in the cloud consumer region. For example, European companies storing data in the United States expose their data to easier access by government agencies due to the U.S. Patriot Act [8].

- **Application parallelization.**

  Cloud computing offers additional resources by adding nodes to the system on-demand. Horizontal scaling by adding new servers provides additional processing; however, only applications designed to utilize parallelization can take advantages of such resources.

Although, the challenges described above are generic and outside the scope of this work, they have a considerable amount of implications on possible implementation of this work in practice. Hence, they need to be taken into consideration when implementing the proposed approach in practice.

As this research includes data storage and processing in the cloud, modern data storage technologies, namely NoSQL data stores, are introduced in Section 2.3.

2.3 NoSQL Data Stores

Relational databases (RDBs) are traditional data storage systems designed for storing structured data. They have been used for decades dues to their reliability, consistency, ACID (Atomicity, Consistency, Isolation, Durability) transactions and query capabilities through SQL. However, RDBs exhibit horizontal scalability challenges, Big Data inefficiencies, and limited availability [10]. In an attempt to address the challenges encountered by RDBs in handling Big Data and in satisfying cloud requirements, new storage solutions, namely NOSQL data stores [13], have emerged. Since this work aims to provide a storage solution for disaster-related Big Data, the proposed solution takes advantages of NoSQL data stores.

The terminology "NoSQL" references to "Not only SQL" which highlights the fact that SQL-style querying is not a crucial objective of these data stores. Thus, the term constitutes a large variety of data stores that are not based on the relational model, including specialized solutions designed for highly specific applications such as graph storage. Although there is no set of definition on what constitutes a NoSQL solution, the following set of characteristics is often attributed to them [11] [8] [16]:

- Straightforward and flexible non-relational data models. NoSQL databases provide flexible schema, occasionally offering completely schema-free designs. This makes them ready to handle a wide array of data structures [11] [3].

- Ability to scale horizontally over many nodes, which can provide additional processing as well as storage. Some data stores can also improve performance by parallelization of read and/or write operations.

- High Availability. NoSQL data stores were designed to provide distributed processing, this facilitates redundancy. Many forms of distributed scenarios can be configured, while considering partition tolerance as unavoidable. NoSQL solutions choose to compromise consistency in lieu of availability, resulting in AP (Available / Partition-tolerant) data stores, whereas most RDBMSs are CA (Consistent / Available)

- NoSQL movement trades off ACID compliance for other properties, unlike a RDBMS. NoSQL data stores are occasionally referenced as BASE (Basically Available, Soft state, Eventually consistent) [2] systems. In BASE, Basically Available means that the data store is available whenever it is accessed, even if certain parts are not accessible; Soft state calls attention to the fact that the data store can tolerate inconsistency for a certain time period; Eventually consistent emphasizes that after a certain amount of time period, the data store will arrive at a consistent state; it is also known as delayed consistency.

- Schema normalization is not stressed. A De-normalized schema can provide straightforward data access, decrease the use resource hungry operations such as joins, and can more easily scale horizontally.Although, it is important to note that this approach will result in utilization of more storage for disaster-related data compared to a normalized schema [8].

Two fundamental elements in enabling NoSQL data stores are distributed computing and cloud computing. When traditional data stores were introduced, available storage space was very limited and hence normalization was highly desired, while redundancy was avoided. As data storage grew larger and cheaper as well as faster due to introduction of new technologies in the area of non-volatile storage, also due to distributed and cloud computing helping in scaling storage to provide massive amounts of space, the need to save on space has decreased. However, the immense quantity of operations imposes strict performance requirements, thus, the focus has shifted from minimizing redundancy and storage requirements to improving performance [8]. As a consequence, NoSQL schemas are often de-normalized, resulting in large storage size, but provide a number of advantages including:

- Improved horizontal scaling as de-normalized schema can be partitioned easily,

- Since the data is redundant by nature, it can be replicated in order to simplify data access.

- Operations that are resource intensive, such as joins, can be avoided.

- Schema can closely resemble application object model and hence reduce impedance mismatch.

The attributes affecting the choice to make NoSQL data stores a suitable storage option for disaster information data management solution proposed in this work include their flexible data model, horizontal scalability, and high availability. A flexible data model facilitates storage of diverse disaster-related data, horizontal scalability facilitates a NoSQL data store to accommodate growing storage needs by adding commodity servers, and high availability ensures uninterrupted operation in case or failures caused due to disasters.

NoSQL data stores are basically classified according to their data model. Since there is no definition on what exactly constitutes a NoSQL data store, various categorizations have been proposed [3] [16]. This research adopts the categorization into four categories: document stores, key-value data stores, column-family stores, and graph databases [11] [16] [5]. The following discussion introduces the four NoSQL data store categories and focuses on the main characteristics relevant for their use in the DMIS platform.

Document Data Stores are designed around the concept of a document and focus on optimizing storage and access for semi-structured documents as opposed to rows or records. These are derivatives of the key-value store data model with documents stored in the value part of the key-value pair. The documents, typically in JSON (JavaScript Object Notation) or BSON (Binary JSON) representation, are hierarchical trees which encapsulate and encode data. The documents within the data store can have different structures, which provide storage flexibility. At the same time, the document structure enables querying capabilities as fields within documents that can be used as query criteria. Example data stores from this category include CouchDB, MongoDB, and Couchbase Server [16].

In the context of DMIS, document data stores provide two advantages: querying capabilities and flexible storage. Querying capabilities are made possible by the structure of the documents within the data store, while storage flexibility is achieved by allowing documents within the store to have different structures. However, querying capabilities and storage flexibility are competing attributes: a certain structural consistency among documents is needed to support querying, while excessive structural consistency decreases storage flexibility.

Key-Value Data Stores have the simplest data model: they provide a simple mapping from each key to its corresponding value. They are primarily used for simple operations in which all access to the store is through a primary key. Client applications can set the value for a key, get the value corresponding to a specified key, or delete a key. The value can be just about anything, and the client application is responsible for interpreting what is stored. Therefore, when using a key-value data store, relations between data are handled at the application level. Although such a simple data model is somewhat restrictive, accessing data only through the primary key provides for good performance and easy scalability. Examples of key-value data stores include Redis, Riak, and Berkeley [16].

In spite of their flexibility, scalability, and performance characteristics, key-value stores have major drawbacks with respect to DMIS platform. Relations between data are handled by the application, and data are accessed only through the primary key. Since the relations among data are not expressed in the data storeâĂŸs data model, integration possibilities are limited. Moreover, accessing data only through the primary key greatly restricts querying capabilities. In the context of DMIS, limited querying capabilities and integration possibilities present a major drawback.

Column-Family Data Stores, similar to key-value stores, map keys to their corresponding values; however, each value consists of a name-value pair. Key-value pairs can be viewed as tuples or rows in a relational database, while name-value pairs can be discerned as column names or attributes and their corresponding values. Therefore, a high level view may make a document data store look similar to relational databases; however, in relational databases, columns are predefined, and each row contains the same fixed set of columns, whereas in the column-family data store, the columns that form a row are determined by the client application, and each row can have a different set of columns. Column-family data stores provide query capabilities. Cassandra, HBase, and Amazon SimpleDB belong to this category [16].

With regards to DMIS platform, column-family data stores provide the same advantages as document data stores: querying capabilities and flexible storage. Querying capabilities are supported by name-value pairs within rows, while storage flexibility is achieved by allowing each row to have a different set of columns. Similarly to document databases, a certain level of consistency among rows is needed to support querying capabilities.

Graph Databases originated from graph theory and use graph-like structures with nodes, edges, and properties to store data. This data model is distinct from the key-value, document, and column-family data models and is designed for efficient management of heavily linked data. Applications based on data with many relationships are well suited for graph databases because the cost of intensive operations like recursive joins can be replaced by efficient graph traversals [11]. Neo4J and Allergo Graph are example stores from this category [16].

In context of DMIS platform, graph databases can be used to store locations of the events, as well as the disaster-related data for that location.

n addition to differences in their data models, data store implementations differ greatly in other aspects, such as scalability, fault tolerance, consistency, and concurrency control. These characteristics, in addition to the data model, are influential factors in determining the most suitable data store for the task at hand. Disaster-CDM offers a choice of storage solutions according to the characteristics of the data to be stored. Specifically, the data store category is chosen according to the data to be stored, and a specific data store implementation is then selected by matching the desired storage attributes with the characteristics of various data store implementations. Because one of the main characteristics of NoSQL data stores is their ability to scale horizontally and effectively by adding more servers to the resource pool, scaling aspects are discussed further here. With regard to what is being scaled, three scaling dimensions can be distinguished: scaling read requests, scaling write requests, and scaling data storage. The partitioning, replication, consistency, and concurrency control strategies used by NoSQL data stores have significant impact on their scalability. For example, partitioning determines the distribution of data among multiple servers and is therefore a means of achieving all three scaling dimensions.

Another important factor in scaling read and write requests is replication: storing the same data on multiple servers so that read and write operations can be distributed over them. Replication also has an important role in providing fault tolerance because data availability can withstand the failure of one or more servers. Furthermore, the choice of replication model is also strongly related to the consistency level provided by the data store. For example, the master-slave asynchronous replication model itself cannot provide consistent read requests from slaves.

2.4 Machine Learning

Machine learning can be accomplished in a supervised or an unsupervised way. In supervised learning, the system receives a dataset with different example parameter values and decisions/classification, from which it infers a mathematical function,which automatically maps an input signal to an output signal. So, it figures out what it is supposed to do.

Unsupervised learning, on the other hand, means that the system acts and observes the consequences of its actions, without referring to any predefined type-cases other than those previously observed. This is pure 'learning by doing' or trial-and-error. Compared to supervised learning, unsupervised methods perform poorly in the beginning, when they are not tuned, but as they tune themselves,performance increases. It can be argued that using unsupervised learning, a classifying system should be able to set up hypotheses that no human can figure out,due to their complexity. If unsupervised methods were used for this project, the machine learning system would have to find out the learner stage hypothesis all on its own, which would probably require much more training data than is available.One would also run the risk of obtaining a hypothesis too complex or specific to aid researchers.

To evaluate classifier performance given by a machine learning scheme, either a special testing set or a cross validation technique may be employed. A test set contains pre-classified examples different to those in the training set, and is used only for evaluation, not for training. If data are scarce, it is sensible to use cross validation in order not to waste any data, which could be useful to enhance classifier performance; all data are used both for training the classifier and for testing its performance.

More examples does not necessarily mean better classifier performance. Even though the classifier becomes better on the training set it could actually perform worse on the test data. This is due to the over-fitting of the classifier transfer function, so that it fits too tightly to the training data and the border between classes is jagged rather than smooth, unlike how it usually should be.

In DMIS platform machine learning can help in filtering and categorizing the data using context-based filtering models.

2.5 Summary

This chapter introduces the high level concepts related to the Disaster Management Information System and how each aspect of the concepts explained can help to achieve such a system.

Big data is relevant here as the disaster related data being gathered is vast as well as varies in type. Here the 3 V's of Big Data is explained with regards to disaster related data.

Next, cloud computing is introduced to create a disaster resilient platform since the solution proposed in this work is cloud-based, the main characteristics, goals and challenges of cloud computing have been discussed. The choice of cloud environment for the storage of disaster-related data has been primarily motivated by its scalability and availability attributes.

Also, since the Disaster Management Information System platform uses a storage model that incorporates NoSQL solutions, NoSQL data stores were introduced and their characteristics described. The motivating factors for choosing NoSQL data stores in the proposed approach included data model flexibility, horizontal scalability and high availability. Furthermore, the four NoSQL data models were described with an emphasis on the characteristics relevant to the disaster-related information being stored.

Additionally, Machine learning context were discussed due to the need o f the system to process accurate information. This can only be achieve by models that can filter the disaster-related data based on the context of the information being processed. The system proposes to use a cognitive based filtering classifier to achieve this goal.

Chapter 3

Design and Implementation

This chapter of the study deals with the design and the implementation of the system being proposed to store disaster-related information. Here, we will look into the specific technologies used to construct the system. This system is intentionally designed to be modular so that specific technologies used can interchanged to provide improvements to the system while also making it possible to adapt the system to different types of use cases.

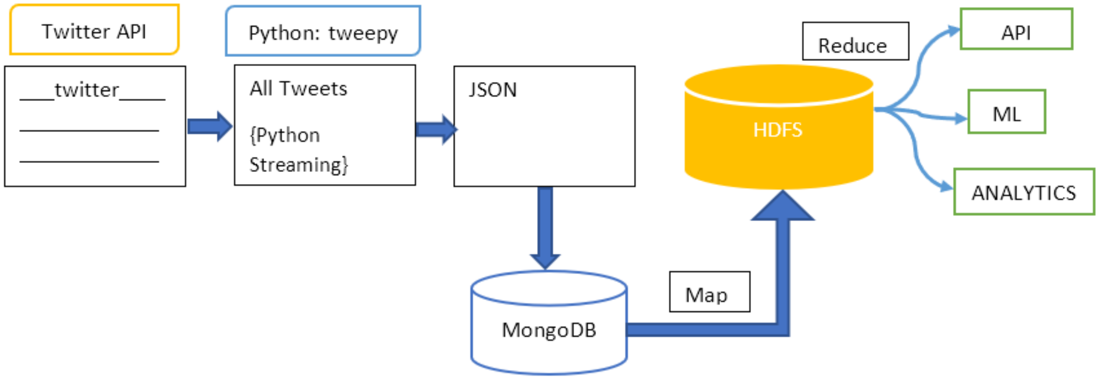


Figure 3-1: Disaster Management Information System design.

The flow of the Disaster Management Information System design is shown in Figure 3-1. Here the Twitter Streaming API provided by Twitter is incorporated in to the system by using tweepy, a python library, to extract the disaster related data. The Streaming API provides a JSON output which is stored into MongoDB. The output of the MongoDB is parsed using a cognitive filtering system to filter out the irrelevant data and then sent to Hadoop mapper to store all the relevant information in the HDFS data store. Hadoop reduce is used to structure the relevant information and provide it to the end user.

3.1 Twitter

3.1.1 Twitter API

Twitter as a platform has a lot of users and it generates a lot of data that can be used for analysis. Data can include user tweets, user profiles, user followers, user status, what's trending, etc. This data can be extracted using Twitter Application Programming Interface(API). There are three methods to get this data: the REST API, the search API, and the Streaming API. The Search API is retrospective and allows to search old tweets [with severe limitations], the REST API allows you to collect user profiles, and followers, and the Streaming API collects tweets in real time as they happen. As this study concerns with the real-time analysis of the data, it is important to highlight the fact that the Streaming API is most suited for those needs.

Using the Twitter API requires a few steps:

1. Authenticate with OAuth

2. Make API call

3. Receive JSON file back

4. Interpret JSON file

3.1.2 Authenticate with OAuth

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords [9]. Authentication with OAuth on Twitter requires you to get keys from the Twitter developers site using a Twitter developer account. There are four keys (Consumer Key, Consumer Secret Key, Access Token, and Secret Token) that are required to access the API and need to be used during a handshake, once authenticated the program can make API calls.

3.1.3 Make API call:

When making an API call to twitter it has parameters incorporated into the URL, the wrapper looks like this:

```
https://stream.twitter.com/1.1/statuses/filter.json?track=twitter
```

This call is being done through the streaming API, where it is asking to connect to Twitter and once connection is established it will track the keyword ’twitter’. To track specific information, keywords can be listed and if done carefully it can filter a lot data at an early stage that is irrelevant in this research. For example, keywords such as ’hurricane’, ’flood’, ’epidemic’, etc., can be listed to stream tweets containing these words.

Using python library called tweepy the working of an API call is abstracted from the user, nevertheless understanding the working of making a Twitter API call is necessary to extract the relevant data.

3.1.4 Receive JSON file back:

JavaScript Object Notation (JSON) is an open standard file format that has data formatted as attribute-value pairs. The format is language independent and is commonly used for asynchronous browser-server communication for a data request.

JSON files is also the data structure that Twitter returns when an API call is made. The amount of data returned depends on how the keywords are defined, but is usually rather comprehensive and needs to be parsed.

3.1.5 Interpret JSON file:

JSON file can be stored as raw file or can be stored using a SQL/NoSQL database. Since the data received is unstructured the most logical way to store the data is using a NoSQL database. In this study MongoDB is used to store the tweets we receive and parse through. Next, queries are run based on the keywords that are needed to select information.

3.2 Python:

While there are many different programming languages that can be used to interface with the API, the flexibility and huge community support behind python as well as its relevance in data science makes python the ideal choice in this research. Python has many libraries that has different use cases, here tweepy is used to stream the disaster-related data from twitter.

3.3 Tweepy:

Python is a versatile language with adaptability to various use cases. These are done by extending the language by using libraries which are community created. One of these libraries is tweepy. Tweepy is open-sourced, hosted on GitHub and enables Python to communicate with Twitter platform and use its API [14]. This makes it easier to access the platform to collect and monitor tweets for analysis.

3.3.1 Using tweepy:

Command to install the tweepy library:

```
\$ pip install tweepy
```

Tweepy supports OAuth authentication. Authentication is handled by the tweepy.AuthHandler class [15]. A consumer token and a secret key is needed to connect with the Twitter stream API, this uses the keys that were generated after a Twitter developer account was created.

These keys are a pair of private and public (secret and non-secret) keys and used to maintain security. The consumer key pair authorizes the program to use the Twitter API, and the access token essentially signs the application into specific Twitter user account. This framework makes more sense in the context of third party Twitter developers like TweetDeck where the application is making API calls but it needs access to each user's personal data to write tweets, access their time-lines, etc. [7].

The tweepy library can be imported as shown:

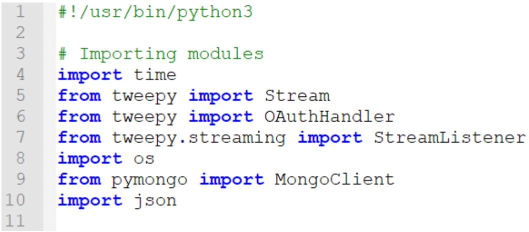
- from tweepy import Stream
- from tweepy import OAuthHandler
- from tweepy.streaming import StreamListener

The above tweepy class imports will be used to construct the stream listener.

3.4 Diving into the code:

This section helps to describe how the information is stored as a flat file and the advantages of storing the information into a NoSQL data store like MongoDB. Here, the tweepy library is shown using the Streaming API to store information.

3.4.1 Importing the modules:

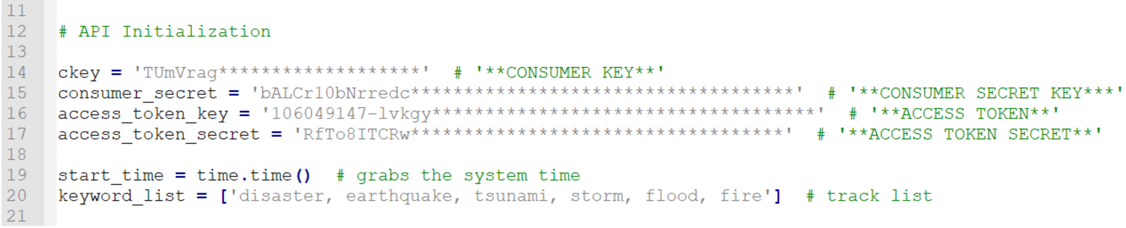


```
#!/usr/bin/python3

# Importing modules
import time
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import os
from pymongo import MongoClient
import json

```

Figure 3-2: Importing libraries.

As shown in Figure 3-2, apart from the three tweepy class imports used to construct the stream listener, the time library is used to create a time-out feature for the script, and the os library is used to set the working directory.

3.4.2 Setting the Variables:



```
# API Initialization

ckey = 'TUmVrag*******************'  # '**CONSUMER KEY**'
consumer_secret = 'bALCr10bNrredc***********************************'  # '**CONSUMER SECRET KEY***'
access_token_key = '106049147-lvkgy***********************************'  # '**ACCESS TOKEN**'
access_token_secret = 'RfTo8ITCRw**********************************'  # '**ACCESS TOKEN SECRET**'

start_time = time.time()  # grabs the system time
keyword_list = ['disaster, earthquake, tsunami, storm, flood, fire']  # track list
```

Figure 3-3: Setting tokens.

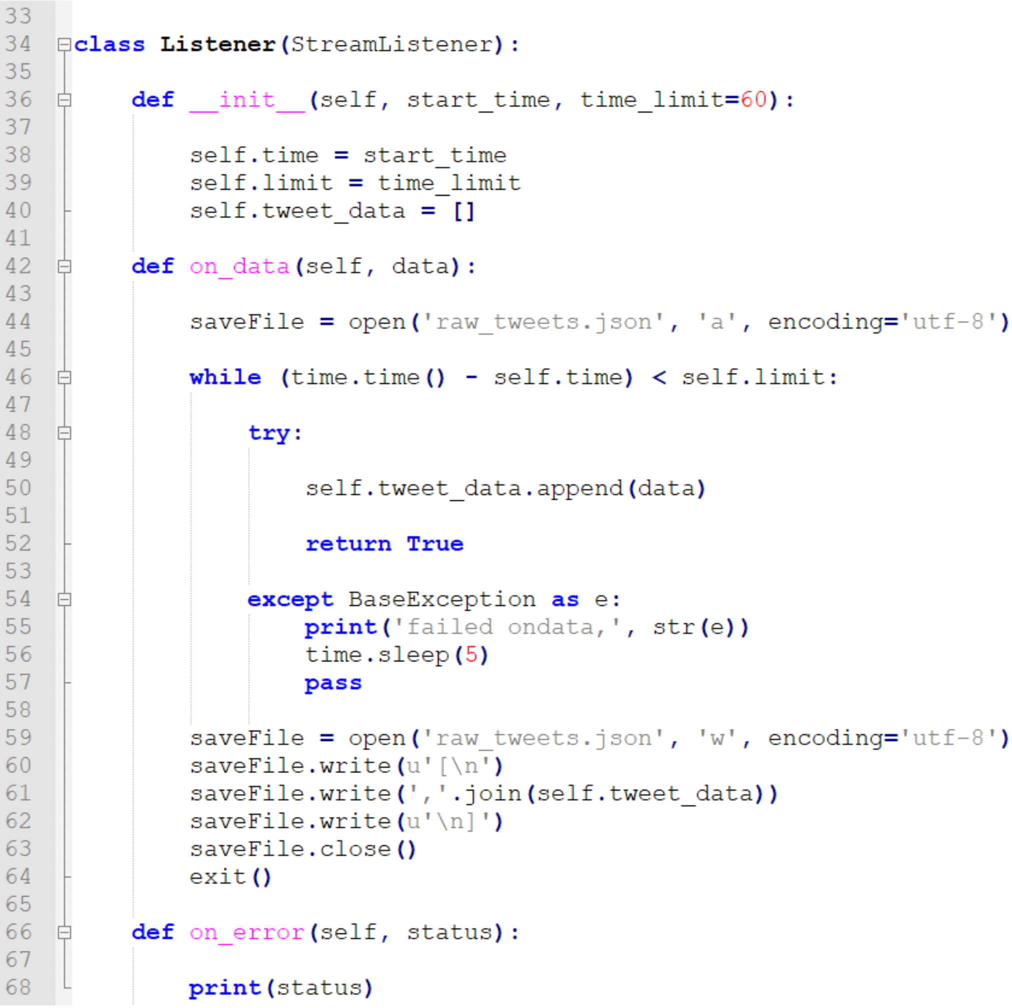
As shown in Figure 3-3, the variables have to be set, which is then used in the stream listener by being fed into the tweepy objects. Here, we set the different token keys generated by signing up for the Twitter API. These keys help to provide access without twitter divulging its authentication tokens.

3.4.3 Using and Modifying the Tweepy Classes:

The code shown in Figure 3-4 does the following:

- Creates an OAuthHandler instance to handle OAuth credentials.
- Creates a listener instance with a start time and time limit parameters passed to it.
- Creates an StreamListener instance with the OAuthHandler instance and the listener instance.

However, before these instances are created, the StreamListener class has to be modified by creating a child class to output the data into a .csv file. This data is outputted into MongoDB by reading this .csv file. The advantage we gain here is that the data can be queries using the NoSQL data store's querying methods to search



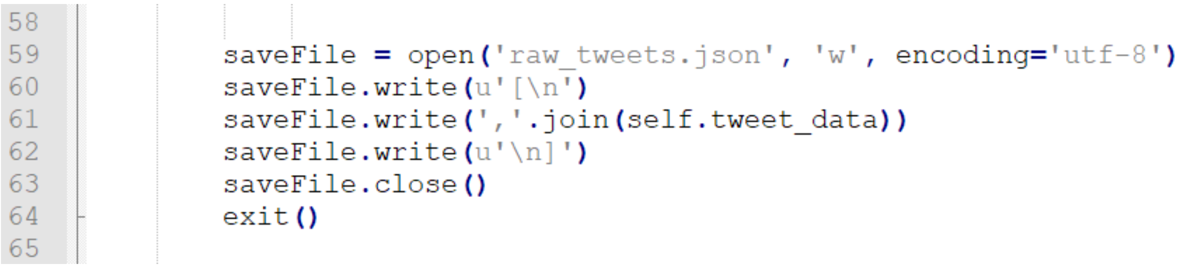
```
class Listener(StreamListener):

    def __init__(self, start_time, time_limit=60):

        self.time = start_time
        self.limit = time_limit
        self.tweet_data = []

    def on_data(self, data):

        saveFile = open('raw_tweets.json', 'a', encoding='utf-8')

        while (time.time() - self.time) < self.limit:

            try:

                self.tweet_data.append(data)

                return True

            except BaseException as e:
                print('failed ondata,', str(e))
                time.sleep(5)
                pass

        saveFile = open('raw_tweets.json', 'w', encoding='utf-8')
        saveFile.write(u'[\n')
        saveFile.write(','.join(self.tweet_data))
        saveFile.write(u'\n]')
        saveFile.close()
        exit()

    def on_error(self, status):

        print(status)
```

Figure 3-4: Stream listener.

for information. This makes it easier to continuously store data on a streaming basis without interrupting the data flow. Also, using a NoSQL data store we can easily without any downtime or interruption to the streaming process.

To elaborate more on the writing of the data to a file after the StreamListener instance receives data:

As shown in Figure 3-5, the block of code opens an output file, writes the opening square bracket, writes the JSON data as text separated by commas, then inserts a closing square bracket, and closes the document. This is the standard JSON format with each Twitter object acting as an element in a JavaScript array. If this is brought into the Python built-in parser, the JSON library can properly handle it.



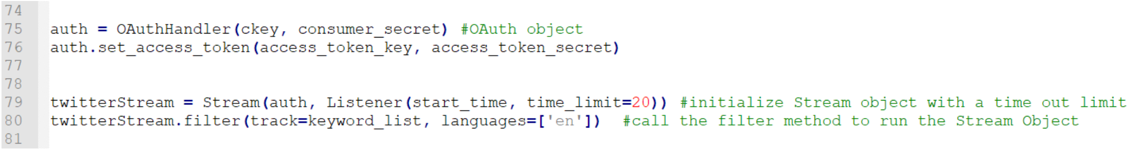
```
            saveFile = open('raw_tweets.json', 'w', encoding='utf-8')
            saveFile.write(u'[\n')
            saveFile.write(','.join(self.tweet_data))
            saveFile.write(u'\n]')
            saveFile.close()
            exit()
```

Figure 3-5: Storing data.

This section can be modified to or modify the JSON file. For example, other properties/fields like a UNIX time stamp or a random variable can be placed into the JSON format. Additionally, the output file can also be modified to eliminate the need for a .csv file and insert the tweet directly into a MongoDB database. As it is written, this will produce a file that can be parsed by Python's JSON class.

After the child class is created , the instances can be created and then the stream listener can be started to store the information.

3.4.4 Calling the Stream Listener:



```
auth = OAuthHandler(ckey, consumer_secret) #OAuth object
auth.set_access_token(access_token_key, access_token_secret)


twitterStream = Stream(auth, Listener(start_time, time_limit=20)) #initialize Stream object with a time out limit
twitterStream.filter(track=keyword_list, languages=['en'])  #call the filter method to run the Stream Object
```

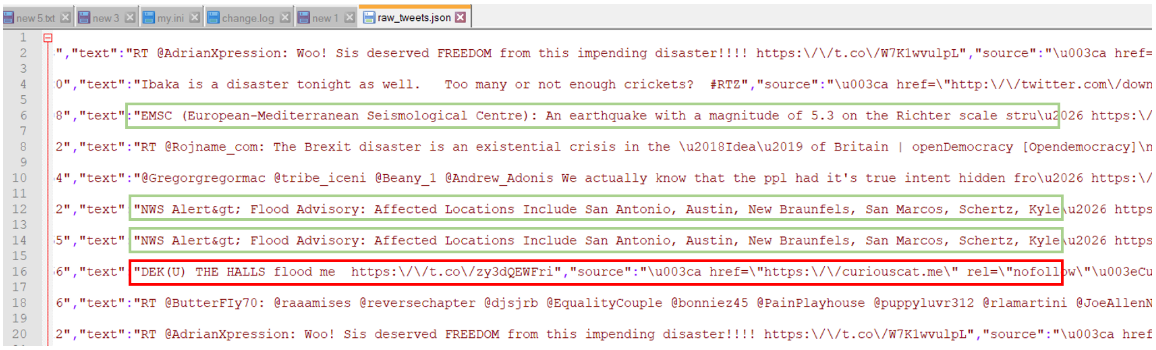
Figure 3-6: Calling and filtering tweets in real time.

As shown in Figure 3-6, the OAuthHandler uses the generated API keys [consumer key and consumer secret key] to create the auth object. The access token, which is unique to an individual user [not an application], is set in the following line. This will take all four of the credentials from the Twitter Dev site. The modified StreamListener class, simply called listener, is used to create a listener instance. This contains the information about what to do with the data once it comes back from the Twitter API call. Both the listener and auth instances are used to create the Stream instance which combines the authentication credentials with the instructions on what to do with the retrieved data. The Stream class also contains a method for filtering the

Twitter Stream. The parameters are passed to the Stream API call.

3.4.5 Twitter Streaming Output:

Figure 3-7 depicts the output of the Twitter Streaming API, here the keywords used are earthquake, disaster, flood. The output is store to a flat .csv file. In the figure the green highlighted tweets specify the relevant output while the red highlighted tweets show the irrelevant information to this system. This can be parsed by passing the information through a cognitive filtering system.



```
","text":"RT @AdrianXpression: Woo! Sis deserved FREEDOM from this impending disaster!!!! https:\/\/t.co\/W7K1wvulpL","source":"\u003ca href=

0","text":"Ibaka is a disaster tonight as well.   Too many or not enough crickets?  #RTZ","source":"\u003ca href=\"http:\/\/twitter.com\/down

8","text":"EMSC (European-Mediterranean Seismological Centre): An earthquake with a magnitude of 5.3 on the Richter scale stru\u2026 https:\/

2","text":"RT @Rojname_com: The Brexit disaster is an existential crisis in the \u2018Idea\u2019 of Britain | openDemocracy [Opendemocracy]\n

4","text":"@Gregorgregormac @tribe_iceni @Beany_1 @Andrew_Adonis We actually know that the ppl had it's true intent hidden fro\u2026 https:\/

2","text":"NWS Alert&gt; Flood Advisory: Affected Locations Include San Antonio, Austin, New Braunfels, San Marcos, Schertz, Kyle\u2026 https

5","text":"NWS Alert&gt; Flood Advisory: Affected Locations Include San Antonio, Austin, New Braunfels, San Marcos, Schertz, Kyle\u2026 https

6","text":"DEK(U) THE HALLS flood me  https:\/\/t.co\/zy3dQEWFri","source":"\u003ca href=\"https:\/\/curiouscat.me\" rel=\"nofollow\"\u003eCu

6","text":"RT @ButterFIy70: @raaamises @reversechapter @djsjrb @EqualityCouple @bonniez45 @PainPlayhouse @puppyluvr312 @rlamartini @JoeAllenN

2","text":"RT @AdrianXpression: Woo! Sis deserved FREEDOM from this impending disaster!!!! https:\/\/t.co\/W7K1wvulpL","source":"\u003ca href
```

Figure 3-7: Tweet stream output.

3.5 MongoDB

Storing JSON tweets as a .csv file works well, but they don't always make good flat .csv files as not every tweet has the same structure nor do every tweet contain the same fields. Some data is well nested into the JSON objects. It is possible to write a parser that has a field for each possible subfield, but this can take a lot of time as involves a lot of considerations and will also create a large .csv file or SQL database.

NoSQL databases like MongoDB works very well as a simple tweet storage, search and recall, which eliminates the need to use an extensive tweet parser.

3.5.1 What is MongoDB?

It is a document-based database that stores data using documents rather than using tuples in tables like traditional relational databases. These documents are similar in structure to JSON objects using key-value pairs and are called BSON (Binary JSON). JSON and BSON have similar properties as JS objects and Python dictionaries.

3.5.2 Why store in MongoDB?

Storing tweets in MongoDB makes sense as BSON and JSON are so similar and that makes putting the entire content of a tweet's JSON string into an insert statement and executing that statement to store the data. This also makes recalling and searching for tweets simple although it does require a change in thought process of rather executing traditional SQL commands to treating data as OOP structures.

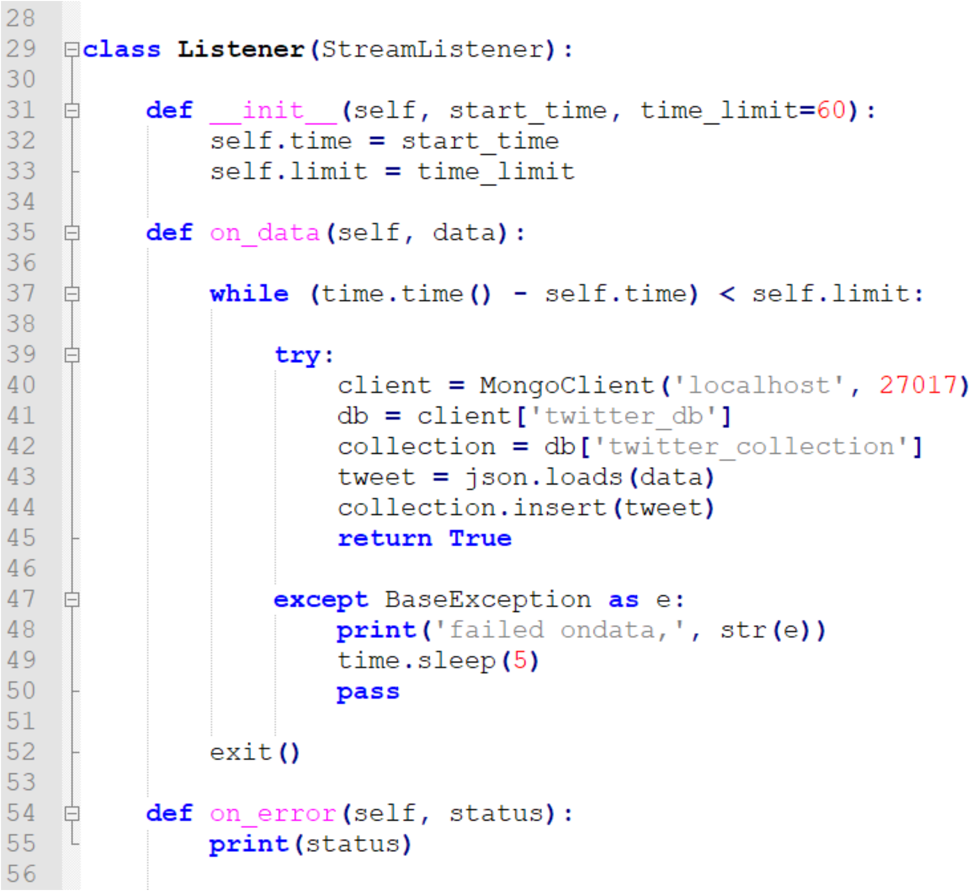
3.6 Storing Tweets in MongoDB

Once MongoDB is installed and configured storing tweets is simple using the Python stream listener. Modifying the code shown above the pymongo and JSON libraries can be imported. The JSON library is the default python library and will be available to import, pymongo needs to be set up using the following command:

```
\$ pip install pymongo
```

The main changes in the code that was done is in the listener child class as shown in Figure 3-7.

as shown in Figure 3-8, MongoClient creates the MongoClient instance which interfaces with the database. The client['twitter_db'] call designates the database that is going to be used, and the db['twitter_collection'] call selects the collection where the documents will be stored. The json.loads() call converts the string returned from the Twitter API into a JSON object in Python. Finally, the collection.insert() call inserts the JSON object into the MongoDB database. From this rather simple



```
class Listener(StreamListener):

    def __init__(self, start_time, time_limit=60):
        self.time = start_time
        self.limit = time_limit

    def on_data(self, data):

        while (time.time() - self.time) < self.limit:

            try:
                client = MongoClient('localhost', 27017)
                db = client['twitter_db']
                collection = db['twitter_collection']
                tweet = json.loads(data)
                collection.insert(tweet)
                return True

            except BaseException as e:
                print('failed ondata,', str(e))
                time.sleep(5)
                pass

        exit()

    def on_error(self, status):
        print(status)

```

Figure 3-8: Storing tweets in MongoDB.

change to the Python stream listener all the tweets can be saved into a MongoDB database.

3.7 Recalling Tweets from MongoDB

The function to retrieve any document from a MongoDB database is collection.find(). Here, I can specify what I want or leave it black to get all the documents returned, in my case it will be all the tweets.

Calling using the .find() method, Python returns a MongoDB cursor, which can be iterated through by putting it in a for loop. The for loop will run the loop for each object in the iterator.

3.8 Context based filtering

Context-based filtering, also referred to as cognitive filtering, recommends elements based on a comparison between the content of the elements and the requirement of the output. Content of each element is represented as a set of descriptors or terms, typically a set of words that occur in a document or in this case in the data stored in MongoDB.

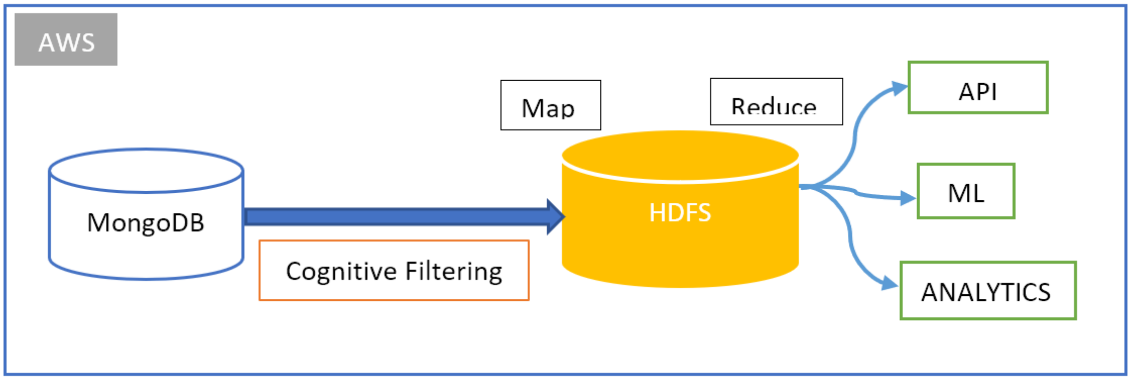


Figure 3-9: Cognitive filtering model.

As shown in Figure 3-9, the cognitive filtering system will filter relevant information from the Streaming API output as shown in Figure 3-7. The cognitive filtering model can represent each disaster element as a set of terms that can be based on the context of the output to filter out the information.

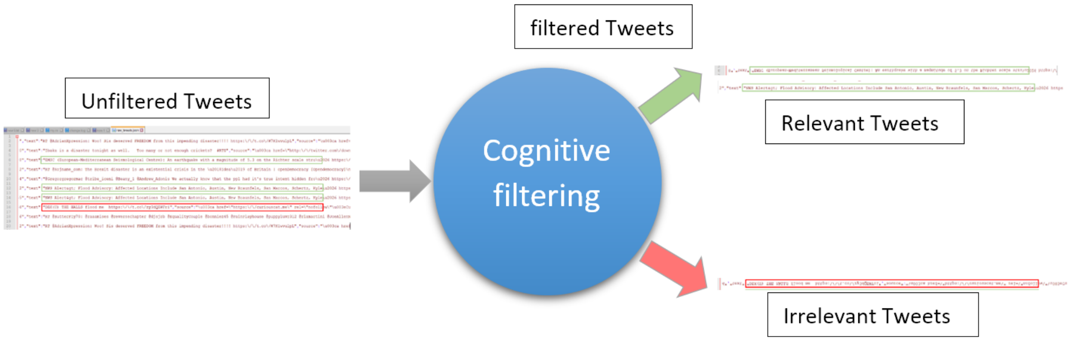


Figure 3-10: Cognitive filtering model.

As shown in Figure 3-10, the context filtering model works to filter the tweets base on relevance of the information being fed to it. The model is trained to understand the context of the data and provide output based on those requirements. With respect to filtering the disaster related data, the model takes the unfiltered data and filters the data into relevant and irrelevant information. The relevant information is sent to the Hadoop data store for further processing, the irrelevant information is discarded, saving space.

Although, it is out of scope of this research to implement a context-based filtering system, it is absolutely necessary to mention a need of a cognitive based system to filter the data before processing it, as the goal of the DMIS design is to provide a solution for fast delivery during response phase of a disaster.

3.9 Hadoop

Hadoop is a programming framework used to support the processing of large data sets in a distributed computing environment [18]. Hadoop ecosystem consists of Hadoop Kernel, Mapreduce, HDFS and number of various components like Apache Hive, Base and Zookeeper.

This study uses MapReduce framework to process the data to output the information. The framework allows processing of large dataset by using divide and conquer strategy, and run it in parallel.

The input of the Hadoop data store is the output of the context filtering model. The relevant tweets are stored in the data store. The system processes the tweets using Hadoop Map and Reduce to provide location-based information.

3.10 Amazon Web Services

Amazon Web Services (AWS), a cloud computing service, provides a platform that is ideally suited for building fault-tolerant software systems [1]. As the DMIS platform aims to be operational even during a disaster, hosting it on cloud platform like AWS is a logical choice.

Amazon Elastic Compute Cloud (Amazon EC2) is a web service within AWS that provides computing resources that can be used to build and host systems. Using multiple EC2 instances, a highly reliable and fault-tolerant DMIS can be built. Also AWS provides the flexibility to scale using tools and ancillary services such as Auto Scaling and Elastic Load Balancing.

On the surface, Amazon EC2 instances behave like traditional hardware servers as they provide a virtual shell to host familiar operating systems like Linux, Windows, or Debian. Thus, providing flexibility to accommodate any type of platform to run on these systems. Additionally, a system can start small and grow when needed by adding heterogeneous nodes.

Using AWS, DMIS is poised to realize the full potential of the platform and use its services in the fullest possible way. Furthermore, gradual expansion of the system can be modular and new features can be incorporated in to the system over new nodes.

3.11 Summary

In this chapter the design of the system was described while also detailing the choices made to use certain technologies to implement the system in practice. The twitter data streaming API called the Twitter Streaming API is used and integrated in to the tweepy python library to abstract the API call for easier implementation.

The Streaming data is stored in a .csv file which is then read into MongoDB to store the information. During the streaming API call the Twitter Streaming API provides the option to extract data based on keywords. This is specified during the DMIS API call and is the first step to filter the data collected from the Twitter data store.

As the data collected is still unfiltered as it contains relevant information mixed with irrelevant information, the system has to filter this before processing it further. This is done before the data is passed to Hadoop for further processing. The unfiltered data is passed through a cognitive filtering system to sort the data based on the context of the information. The cognitive filtering model works as shown in Figure 3-10.

Once the data is filtered the relevant information is passed to the Hadoop system for processing, the Hadoop map and reduce functions are used to parse the information and structure the data that can be queried using an API call to serve different use cases. Additionally, the raw information can be provided that can be used for further analysis or parsing as needed.

All the components of the system are hosted on Amazon Web Services cloud computing platform for making it resilient to disaster related failures. In the context of this study, the system described here provides with an approach to a system that can parse disaster related data and store it continuously.

Chapter 4

Conclusion

Recently, we have been witnessing an increase in the the number and severity of extreme weather events and natural disasters around the globe. As a result, disaster impacts on human lives and property have risen, escalating the importance of minimizing disaster impacts and making an effective response imperative in today's society.

The aim of disaster management is to mitigate disaster impact, and a vital element of achieving this goal is to effectively make decisions through all four disaster phases: mitigation, preparedness, response, and recovery. Decisions made in response phase is critical as they can be a make or break situation in many cases. Successful and effective disaster decision-making requires information gathering, sharing, and integration by means of collaboration on a global scale and across governments, industries and communities. However, it is problematic to search for information while experiencing a critical moment, as time is a constraint. A multitude of disaster-related data is available, including response plans, records of previous incidents, simulation data, social media data and websites.

Currently, very few disaster management solutions exist, and many don't integrate these available resources in real-time to provide effective solutions during a disaster response phase. Also, the introduction of new technologies in hardware and software have potentially created opportunities to develop new solutions in disaster management domain. Especially, recent advances in cloud computing, Big Data, NoSQL, and Machine learning techniques have enabled implementation of these solutions.

Consequently, this research proposed a Disaster Management Information System (DMIS), implemented on Infrastructure as a Service (IaaS) framework for disaster cloud data management in real-time. The ultimate goal of DMIS is to facilitate improved and informed disaster decision-making in a timely manner and to reduce the impact of a impending catastrophe on human lives and property. DMIS aides in information gathering and sharing through knowledge acquisition and delivery; stores and processes huge amounts of disaster-related data from various sources, in real-time, by taking advantage of cloud computing, NoSQL data stores and machine learning techniques.

The design presented in this research describes a system that can store and process information, from diverse sources, on a streaming basis in real-time.

Section 4.1 discusses the contributions of this research.

4.1 Contributions

The contributions of this thesis can be summarized as follows:

Disaster Management Information System

This research has proposed a Disaster Management Information System, a data collection and delivery platform. DMIS provides a flexible and a customizable disaster data management and processing solution, hosted on Infrastructure as a Service framework, that can be scaled and altered with ease according to the changes in requirements. DMIS achieves the following objectives:

- Information is gathered from Twitter, demonstrating social media sources as a trove of information that is filtered through for disaster management information delivery. Knowledge is acquired through diverse collaboration partners and heterogeneous data sources and then stored. Data is processed to output relevant data which can be integrated to various platform sources, delivering disaster-related data as a service.

- Cloud computing and NoSQL data stores are used to effectively store and process large amounts of disaster-related data which is sourced from diverse sources. This provides significant advantage over traditional processing and storing techniques. Utilizing cloud computing and NoSQL data stores enables the platform to start small and scale horizontally as on a need basis by adding heterogeneous nodes. Also, within the cloud environment, databases, relational or NoSQL, are replicated, often across large geographical distances. This leads to the system having high availability and fault tolerant even in the presence of failures, which is fundamental in an operational disaster management system during a crisis. Moreover, NoSQL data stores offer a flexible data model, and therefore enables storage of diverse disaster-related data from various sources. DMIS provides choices of different solutions that suits a variety of data structures and access patterns.

- Search, data delivery and integration is provided on the output by batch processing the data in real-time. The data stored in the data stores is processed or provided raw depending on the requirement of the application. These platform choices lead the system in being flexible, and support a wide arrays of applications that can potentially utilize the information for further improvements to the system. This approach focuses on real-time information collection and delivery, as well as explores the various techniques to build a resilient platform that provides accurate information.

Already stated above, DMIS is a flexible and scalable disaster management platform that can accommodate a variety fo data sources. Thus, this thesis structured a process to introduce new data sources into the system. The process can be described as:

- adding new processing services for dealing with the new data source;

- defining data processing rules for the new data source;

- determining appropriate data storage, including choosing the type of data store and designing a data model.

Introduction of a new source will consider all the three properties stated above, but adding a new source does not necessarily mean that it introduce new cloud components. To rephrase, depending on existing processing capabilities, a new data source may not use resources to warrant scaling to include additional processing node.

DMIS approach used twitter's streaming API to stream data continuously as this helps to collect data in real-time, apart from the streaming information, static file-styles data can be read in to the data store, showing the strength of a NoSQL data stores. The streaming information is collected and filtered by using keywords in the GET request provided to the streaming API and also simultaneously stored in a NoSQL database. Sourcing data from various sources can create a structural disparity in data consistency; however, the data can be filtered and sorted using machine learning techniques.

Disaster-related data storage is performed in two stages:

- The data is streamed in real time or read from various sources and stored in a document data store.
- The data from the document data store is read and sent for processing and storage to column based data store.

The system is designed to be modular and scalable so the platform can be fitted with competing, but consisting of similar model, NoSQL solutions without affecting the data storage and output. Of course, all the previous data has to be migrated in case of using a new data store.

Cleaning the data is equally important as the output is pure based on the quality of the data. It is observed that ML techniques, such as context-based filtering, can benefit the system by passing relevant information while processing. Also, by integrating more artificial intelligence (AI) into the system will make the system more capable of making intelligent choices in various phases of data storage, processing and presentation. This can be a part of future work on improving the platform to produce better results.

Chapter 5

Future Work

This study has primarily addressed a solution to data acquisition from various sources, and defined a system that can make parsing and storing that information in real-time relatively simpler. This approach has established the foundation of a system that can be only be improved by introducing innovative elements for it to process and output data faster and accurately. Future directions related to disaster management system include:

- **Web Application to access the disaster information:**

  Providing a web application to access the information the Internet provides a way to have universal access to disaster-related data. Thick or thin clients (e.g. mobile, laptops, workstations and servers) can have access to information for decision making or further analysis.

- **Native applications for various platforms:**

  Native applications on various platforms can be built, which can use push notification system to deliver updated information to smart connected devices. For example, an Android application can periodically request information to keep the user updated with relevant information.

- **Context filtering:**

  The data collected through social media sites can have different context. A context filtering system using machine learning techniques can improve the result and provide relevant disaster related data. For example, taking flooding as a keyword to filter and store data, the system can have information such as "flooding in my bathroom" or "flash flooding in the street" can mean very differently and the system is not capable of recognizing these differences, thus a context filtering system can filter these results to provide accurate information. This also saves a lot of overhead during processing and storing information.

- **NoSQL data store comparison:**

  In this study, the document store model specifically MongoDB, was chosen for storage of streaming-style data using the streaming API. Also, Hadoop was used for processing as it provides distributed processing which is suitable for parallelization. However, a detailed comparison of different data store implementations would assist in choosing the most suitable NoSQL implementation for the task at hand.

- **Required storage space:**

  This work did not analyze the storage space requirements for the proposed approach. DMIS stores all the streaming data as well the data produced after processing the information. Additionally storing data also creates indexes, and must be considered which calculating space estimates.

- **Data analytics services:**

  The processed information can be pre-analyzed using statistical methods to form prediction models that can be used to study the information, to gather insights into disaster-related events.

- **Information conflict:**

  Since data is collected from various sources, the data gathered can have collisions, i.e., the data collected from independent sources can contradict each other, giving a conflicted information. A way to resolve the conflicts can be devised.

- **Data verification:**

  Since data is collected from public sources, a system can be created to verify the quality as well as the accuracy of the information. This can help to avoid the case of "false positives" or giving out wrong information as it is a life or death situation during a crisis.

- **Avoiding Redundancy:**

  There can be similar data from different sources, that can be purged using a filtering model, from the system so as to reduce redundancy in the data store as this adds complexity and unnecessary overhead while processing and storing information.

This study has presented the main idea of Disaster management as an information service, while thoroughly working out the details and challenges faced by the platforms utilized, future work that can focus on this includes:

- **Privacy and security:**

  Providing adequate security and privacy for such a framework is challenging for a number of reasons, including cloud storage on third-party premises and in a shared multi-tenant environment, diversity of the storage models involved, and the large number of collaboration participants

Appendix A

Abbreviations

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

AI - Artificial Intelligence

AWS - Amazon Web Services

BASE - Basically Available, Soft state, Eventually consistent

BSON - Binary JSON

DMIS - Disaster Management Information System

EC2 - Elastic Cloud Compute

IaaS - Infrastructure as a Service

JSON - JavaScript Object Notation

NIST - National Institute of Standards and Technology

NoSQL - Not only SQL

PaaS - Platform as a Service

RDB - Relational database

RDBMS - Relational Database Management System

SaaS - Software as a Service

Bibliography

[1] Jeff Barr, Attila Narin, and Jinesh Varia. Building fault-tolerant applications on aws. *Amazon Web Services*, pages 1–15, 2011.

[2] Eric Brewer. Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29, 2012.

[3] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[4] Damon P Coppola. *Introduction to international disaster management*. Elsevier, 2006.

[5] Olivier Curé, Robin Hecht, Chan Le Duc, and Myriam Lamolle. Data integration over nosql stores using access path based mappings. In *International Conference on Database and Expert Systems Applications*, pages 481–495. Springer, 2011.

[6] Kelli de Faria Cordeiro, Tiago Marino, Maria Luiza M Campos, and Marcos RS Borges. Use of linked data in the design of information infrastructure for collaborative emergency management system. In *Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on*, pages 764–771. IEEE, 2011.

[7] Sean Dolinar. Collecting twitter data: Getting started. `https://stats.seandolinar.com/collecting-twitter-data-getting-started/`.

[8] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud computing: concepts, technology & architecture*. Pearson Education, 2013.

[9] Whitson Gordon. Understanding oauth: What happens when you log into a site with google, twitter, or facebook. `https://lifehacker.com/5918086/understanding-oauth-what-happens-when-you-log-into-a-site-with-google-twitter-or-facebook`.

[10] Jing Han, Meina Song, and Junde Song. A novel solution of distributed memory nosql database for cloud computing. In *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on*, pages 351–355. IEEE, 2011.

[11] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341. IEEE, 2011.

[12] Vagelis Hristidis, Shu-Ching Chen, Tao Li, Steven Luis, and Yi Deng. Survey of data management and analysis in disaster situations. *Journal of Systems and Software*, 83(10):1701–1714, 2010.

[13] Donald Kossmann and Tim Kraska. Data management in the cloud: promises, state-of-the-art, and open questions. *Datenbank-Spektrum*, 10(3):121–129, 2010.

[14] Ahmet NovaliÄĞ. Introduction to tweepy, twitter for python. `https://www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/`.

[15] Joshua Roesslein. Authentication tutorial. `https://github.com/tweepy/tweepy/blob/v3.6.0/docs/auth_tutorial.rst`.

[16] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[17] Sherif Sakr, Anna Liu, Daniel M Batista, and Mohammad Alomari. A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys & Tutorials*, 13(3):311–336, 2011.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. Ieee, 2010.