IT UNIVERSITY OF COPENHAGEN

Clustering Player Behaviors in Data Streams using K-means in MapReduce

by

Sigurdur Karl Magnusson

A thesis submitted in partial fulfillment for the
degree of Master of Science in IT

in the
Computer Science
Software Development and Technology

Supervisors
Julian Togelius, Rasmus Pagh

May 2013

"I know not with what weapons World War III will be fought, but World War IV will be fought with sticks and stones."

Albert Einstein

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

LAH **L**ist **A**bbreviations **H**ere

Chapter 1

Introduction

1.1 Motivation

We live in a world where data is being generated at an amazing rate everywhere around us. Data can describe characteristics of e.g. Internet activities, social interactions, user behaviors in games, mobile phone activities, scientific experiments and measurements from different devices and sensor equipments. Amount of data that is being registered and stored around us is growing massively in volume and complexity. Organizations are storing more and more historic data than before, moving from large databases towards more commonly online distributed file systems or storage web services providing scalability and high availability at commodity costs where *petabytes* (10^{15} = 1.000 *terabytes*) of information can be stored. We live in a world of *Big Data*, exploring unknown patterns and structures without knowing where it will lead us in the future.

"Information is the oil of the 21st century, and analytics is the combustion engine."

Peter Sondergaard, senior VP at Gartner

In digital games, data about in-game user interactions have been logged and behaviors analyzed since the first game came out. Analyzing the user experience and behaviors of players have mostly been done in laboratories in the past, both during game development and after game launch to see if the game was played as designed. Game designs

have becoming increasingly complex in the recent years offering much more freedom to the players by increasing the number of actions available, items to interact with and Massively Multi-player Online (MMO) persistent worlds that continue to exist after a player exits a game [3, 4]. This complexity generates much more user-centric data than before and is increasingly challenging when evaluating game designs [5, 6]. The user interactions being registered is called *user telemetry* and is translated to *game metrics* as referred in game development, providing detailed and objective numbers, e.g. total playtime, monsters killed, puzzles solved.

Collecting user's telemetry can give very detailed quantitative information on player behavior and using data mining techniques can supplement traditional qualitative approaches with large-scale behavioral analysis [7], for example show where users are getting stuck and finding actionable behavioral profiles [3, 4, 8]. In the recent years user behavior analysis have in part been driven by the emergence of MMO games and Free-to-Play (F2P) games which can have millions of users and objects that can form highly complex interactions. These game models, especially of persistent nature, are monitoring users actions and their behaviors to drive their revenue with subscriptions or offer players to buy virtual items via micro transactions [3, 4, 6, 9].

One way of doing a behavioral analysis is use an unsupervised machine learning technique called clustering. Cluster analysis is a popular exploratory data mining technique that groups set of data objects together in a cluster that are more similar to each other than data objects in other groups [10]. Human beings categorizes or classifies a new object or a phenomenon based on similarity or dissimilarity of the object's descriptive features and is one of most primitive activities of humans [11]. Clustering explores the unknown patterns of the data and provide compressed data representation for large-scale data. In computer games cluster analysis or behavioral categorization can find behavioral profiles that are actionable and give high valuable insights into the game development as well as increasing the monetization [12, 13].

Many clustering algorithms are designed for modern sizes of datasets where the whole data can fit into memory or allowing few passes into a database (where each data object is read more than once). It can be very expensive analyzing large-scale datasets and to get answers efficiently then one needs to reduce the set of data to be analyzed, e.g. sample fewer players and have fewer features (dimensions) to be compared. Computations

for large-scale data takes time and needs to be distributed to be able to complete in reasonable amount of time. Google's MapReduce programming model was introduced in 2004 [14] and allows automatic parallelization and distribution of computations on large clusters of commodity computers. Allowing programmers and researchers to easily implement highly scalable algorithms to process large amount of data using the MapReduce model without worrying about handling failures and distributing the data with a large amount of complex code.

1.2 Problem Statement

> How can clustering using incremental k-means find general player behaviors in large-scale behavioral game data in reasonable time?

Considering the massive size of user telemetry data being logged and processed, and the complexity of game designs. There is a knowledge gap when it comes to analyzing such large-scale data efficiently. Number of players are increasing and the complexity of player-game and player-player interactions grows exponentially. The largest massively multiplayer online role-playing game (MMORPG) *World of Warcraft* [1] had a population around of 10 million users (in 2012 from MMOData [2]), where players live in a persistent world that can create many millions of different and complex interactions in the game.

User telemetry from games can arrive in daily chunks and need to be processed incrementally (in mini batches). There is a need for algorithms that can process massive amount of data that doesn't fit in a computer memory to extracts knowledge in a reasonable time. The k-means algorithm can find clusters that represent general game behaviors. When implemented in the MapReduce framework, k-means can cluster the data in parallel and is highly scalable with running time increasing linearly with the size of the input.

Our goal with this project is to implement a scalable clustering algorithm to find the general behaviors in a specific real life game dataset in collaboration with GameAnalytics (GA) [16]. The goal is not to implement a complete product but a scalable algorithm that provides information about the general player behavioral profiles for a specific game.

[1] http://en.wikipedia.org/wiki/World_of_Warcraft
[2] http://mmodata.blogspot.com

Also the algorithm should allow GA to easily develop a product that can cluster general player behaviors in large-scale games.

The success criteria of this project is:

- A scalable k-means clustering algorithm finding clusters describing the general behaviors of a real life game dataset provided by GA.
- The general behaviors found must be intuitively interpretable and actionable to game developers.
- The algorithm must be able to process incrementally cluster daily arriving chunks of game metric data.

GameAnalytics is Software as a Service (SaaS) start-up, a data and analytics engine for game studios with its headquarters located in Copenhagen. Analyzing large quantities of game metric data that needs to be processed efficiently returning actionable results to aid game design and development.

1.3 Method

TODO A short description of the methods

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

1.4 Contributions

TODO Description of the contributions made in the project

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

1.5 Project Outline

References are cited by index in the bibliography and are in order of appearance, e.g. [2] is a citation number two that is referenced in the thesis. Referring to other sections is by the number of that section, e.g. 4.1.2.

The organization of the thesis is as follows:

- **Chapter 2 - Background** Describes a short background theory about clustering player behaviors and the MapReduce framework for large-scale data parallel processing.
- **Chapter 3 - Related Work** Overview of related and recent work regarding clustering player behaviors and large-scale data with k-means as focus.
- **Chapter 4 - Methodology** Our design and implementation work is described; Description of the real game dataset and selection of features, the k-means algorithm in MapReduce and the experimental set-up.
- **Chapter 5 - Results** Experiment results and observations are explained.
- **Chapter 6 - Conclusions** Conclusions are drawn from the study including future research.

Chapter 2

Background Theory

2.1 Player Behavior

When a user plays a game there can be a high number of action combinations that can be performed that affects the user and his surroundings. Interacting with objects or other players differently over a different time period. These user actions (user telemetry) are registered in the game and play a crucial role for many game genres to analyze and find player behaviors and how they are evolving over time. Online game genres like social online games or Free-to-Play (F2P), in *Facebook* and *Google Play*, drive their revenue by e.g. analyzing player behavior to create actionable knowledge to be able to offer players to buy certain in-game items and upgrades for real money [3, 4, 6, 9].

Games running online persistent worlds where a game continues even a user quits the game, use player behavior analysis e.g. for balancing gameplay, in-game economy and game design. Major commercial games have been using behavior analysis for many years to help game design, Massively Multi-Player Online Role Play Games (MMORPG) play a special interests where the goal is to increase engagement of players playing the game in persistent worlds to have as many active monthly subscribers [7, 17].

F2P and Multi-Player Online (MMO) games allow large groups of players to interact in real-time, many of them in virtual worlds that are always running, allowing emergence of social communities in-game. These types of games can be highly driven by game metrics in game development, analyzing behaviors for e.g. customer acquisition and retention, evolving player communities and monetization [18].

2.1.1 Game Player Metric

When a user is playing a game he can e.g. interact with many items and other users, move through the game environment and purchase items. These kind of user actions is called user telemetry in context of collecting raw measurements data of user interactions in the game that is transmitted remotely to a game server where data about all users are stored and analyzed.

Game player metrics are interpretable quantitative measures of one or more attributes from user telemetry, related either to revenue or players perspective [1]. The revenue perspective player metric can be, e.g. Daily Active Users, Average Revenue Per User or when analyzing user attrition (churn) rate (a measure of the number of leaving users over a specific period of time). In players perspective, metrics are calculated related to user actions in-game, e.g. playtime of a user, average score, average hit/miss ratio, average puzzles solved.

Game metrics can be variables (features) or more complex aggregates or calculated values. For example hit/miss ratio can be calculated by aggregating actions of a user when firing a weapon by sum multiple attributes like "number of hits" and "number of misses" to calculate the hit/miss ratio feature. Also calculating a metric as function of time, e.g. using the features "player id", "session length" and "monsters killed" to calculate the metric "monsters killed per minute" for each player [1].

An example of a feature selection in a work by Drachen et al. [8], analyzing user behavior telemetry in a major commercial title called *Battlefield 2: Bad Company*, a first-person shooter game strategic and tactical elements. Of over 100 features to choose from the authors carefully selected 11 features to allow for evaluating the most important gameplay mechanics [12], relating to character performance, game features and playtime. Some of the features selected were:

- **Score:** Number of points scored in total.
- **Skill level:** Aggregated value of player skill.
- **Total playtime:** Sum of player's time in total.
- **Kill/Death ratio:** Number of kills the player has scored divided with number of his deaths.

- **Score per minute:** Average score per minute of play.

A sub-category of player metrics called *gameplay metrics*, measures of all in-game player behavior and interactions, e.g. navigation, using items and abilities, trading, using the game interface buttons and even the system/objects responses to player actions. Customer metric is another sub-category of the player metrics that usually receives higher priority considering the revenue chain in game development where it covers aspects of the user as a customer, e.g. virtual item purchases and other interactions with the game company, metrics that are interesting to marketing and the management [1]. See Figure 2.1 showing a game metrics diagram with focus on the player metrics hierarchically structure.

Gameplay metrics are however the most important player metric when evaluating player experience, game designs and quality assurance [3, 4, 8], but are generally in low priority compared to the customer metrics [1]. There are huge amount of player behavior

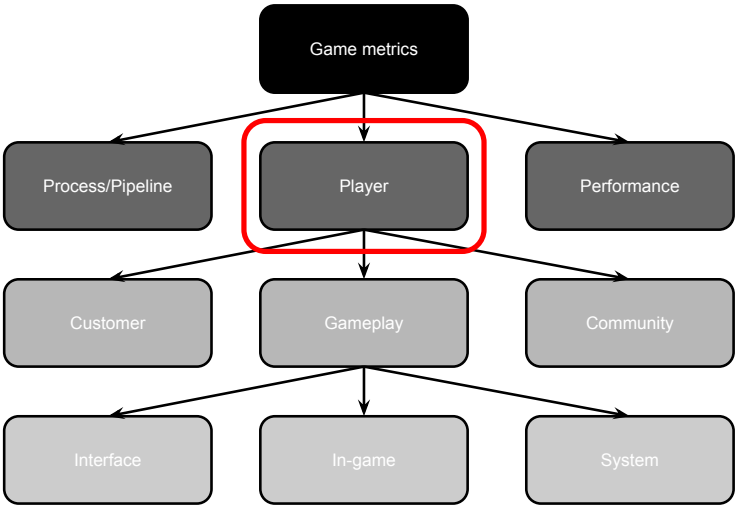

FIGURE 2.1: Player metrics hierarchical diagram. Figure taken from Drachen et al. *Game Analytics - The Basics* [1] and modified accordingly.

measures that are being logged each second in a complex game like MMORPG [3, 4], e.g. location of player, current health, stamina, status on weapons and a many more attributes that could describe a user state in a game also state/reactions of objects and monsters affected by the user. Its clear that this can generate a lot of information and game analysts face the problem of selecting the most essential pieces of information to analyze, imposing a bias but is necessary to avoid information overload [1].

2.2 Clustering

The goal of clustering is to categories or groups similar objects (players, items, behaviors or any observations) together into so called clusters (hidden data structures) while different objects belong to other clusters. A cluster is a set of objects that are similar to each other while objects in different clusters are dissimilar to each other, having high intra-cluster similarity and low inter-cluster similarity [19]. Identifying descriptive features of an object one can compare these features to a known object based on their similarity or dissimilarity based on some criteria. Cluster analysis can be achieved by various algorithms and is a common technique in statistical data analysis that is used in many fields, e.g. machine learning, pattern recognition, image analysis and bioinformatics. The main purposes of clustering has been used for following purposes

- Gaining insights into data, anomaly (outlier) detection and finding the most important describing features in the underlying structure.
- Identify the degree of similarity between objects, like natural classification.
- Compressing or organizing the data and summarize through cluster prototypes.

In this thesis we focus on a popular clustering algorithm called k-means that is a centroid based clustering algorithm where a cluster prototype is represented by a central vector containing average value of all the objects in a cluster, e.g. describing the average behavior.

2.2.1 K-means clustering method

Many clustering methods exists but one of the most popular ones is called *k-means* [20, 21], also known as the Lloyd algorithm [22] which was further generalized for vector quantization [19, 23, 24]. K-means seeks to group similar data points into k partitions or hyperspherical clusters giving insights into the general distributions in the dataset. A cluster is represented by a centroid that characterize the geometric center of the cluster that is calculated as the mean of all data instances belonging to that cluster. The objective function in k-means is to minimizes the squared error between a cluster's centroid (mean) and its assigned points and over all set of clusters minimizing the *Sum*

of Squared Error (SSE) also called the *within-cluster variation*. Let a set of data points $x_i \in \Re^d, i = 1, ..., N$, where each point x is a real number d-dimensional vector and we want to partition them into K clusters $C = \{c_1, ..., c_K\} \in \Re^d$, then the objective function is defined as

$$SSE = \sum_{k=1}^{K} \sum_{x_i \in c_k} dist(x_i, \mu_k)^2$$

where μ_k is the mean vector for the cluster centroid k and $dist(x_i, \mu_k)$ is a Euclidean distance measure between two points the data vector x_i and the mean vector μ_k. The mean vector μ_k is defined as

$$\mu_k = \frac{1}{N_k} \sum_{x_i \in c_k} x_i,$$

where c_k is a cluster number k and its N_k data points $i = 1, ..., N_k$. The Euclidean distance is the most popular distance measure for numeric attributes, also known as the *Minkowski distance* [1] a more generalized version when we have L_2 norm, the Euclidean distance function is defined as

$$dist(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

When running the k-means algorithm it starts by initializing K centroids by choosing random data points from the dataset or according some heuristic procedure. In each iteration of the algorithm it assigns data points to it's nearest centroid by calculating the minimum Euclidean distance to the K centroids for each instance. After assigning all the data points to clusters the centroids are updated so they represent the mean value of all the points in the corresponding cluster. The algorithm stops the iteration when the centroids do not change from the previous iteration or the error (SSE) is below some specific threshold. Also possible to manually define a maximum number of iterations to be run. Algorithm 2.1 shows a pseudo-code of k-means. The algorithm can be seen as a gradient descent procedure which iteratively updates its initial cluster centers so as to decrease its error function.

[1] *http://en.wikipedia.org/wiki/Minkowski_distance/*

Algorithm 2.1 K-MEANS(S, K)

Require: data set of instances $S = x_i, ..., x_n$. K number of clusters.
Ensure: K clusters

1: Initialize K cluster centers
2: **while** termination condition is not satisfied **do**
3: Assign instances to the closest cluster center
4: Update cluster centers based on the assignments
5: **end while**

One of the weaknesses of k-means is that it is sensitive of the initial selection of the centroids which can lead to local optimum, that is the algorithm converges and fails to find the global optimum. One solution is to run the algorithm n times and pick the initialization that gave the lowest SSE result. Another weakness of k-means is that a user has to predefine the number of centers k-means need to cluster, this is most often not known in advance. Many methods exist and most popular one is to run the algorithm with by increasing the k number of clusters to some K and pick a good k candidate where the *"elbow"* starts in the curve in a Scree plot [25]. Noise in data and outliers can dramatically increase the squared error and centroids shifting from data distribution in question towards outliers far away, thus representing skewed distributions. Solutions involve removing these noise in preprocessing or normalize the data with the zero mean normalization [10, 25].

TODO INSERT PICTURE, showing a example of local optimum.

The solution to the optimal partition can be found by checking all possibilities using a brute force method but that is a NP-hard problem [26] and cannot be solved in a reasonable time. The k-means algorithm is a heuristic approach for the clustering problem with running time of the algorithm $O(NKdT)$ where N is number data examples in d-dimensional space and T is the number of iterations. Usually K, d and T is much less than N meaning that k-means is good for clustering large-scale data because of approximately linear time complexity. The most computational work in k-means is the calculation of distances between objects and the cluster centers, in each iteration would require $O(Nk)$ distance computations. Calculating distances for each object is independent of each other, therefore these computations can be parallel executed.

The above implementation of k-means is called the *batch* k-means, where the centroids are updated after all the data points have been assigned. The *online* (incremental) mode of the algorithm processes each data point sequentially. For each data point x

the nearest cluster centroid c_{min} is calculated and that centroid is updated right away, defined as

$$c_{min}(old) = \underset{k}{\operatorname{argmin}} \|x - c_k\|$$

$$c_{min}(new) = c_{min}(old) + \eta(x - c_{min}(old))$$

where the cluster centroid c_{min} is updated towards the data point x using the learning rate η which determines the adaptation speed to each data point. The online approach is however highly dependent on the order of which the data points are processed. A variant of this method is used when clustering an endless stream of data where data points arrive one at a time or in chunks.

2.2.2 Clustering Player Behaviors

Cluster analysis in the context of game development is to find patterns of user behaviors, by reducing the dimensionality of the dataset in order to find the most important behavioral features and able to assign an expressive label to each found player profile. Behavioral clustering can answer questions like e.g. how people play the game, the average playing behavior, evolving behaviors, finding extreme archetypes and find clusters that describe unwanted behavior. Clustering player behaviors can give insights if a game is played as it is designed and obtain actionable knowledge that can be used to refine a game design [8].

There are many clustering methods that can be applied to find player behaviors, the most popular ones in recent years finding interpretable behavior types (basis vectors or centroids) are *k-means* and Simplex Volume Maximization (SIVM) [2, 8]. Both clustering algorithms are applicable to large-scale data but are very different how they represent the behavior types. Figure 2.2 shows an example of how different basis vectors can be found using SIVM and k-means.

SIVM is a Archetype Analysis (AA) approximation algorithm for large-scale data and its goal is to find the extreme (special) player behaviors whereas k-means focuses on the average player behaviors in compact clusters. Its easier to interpret the basis vectors from SIVM since they represent actual sparse data points where k-means cluster centroids

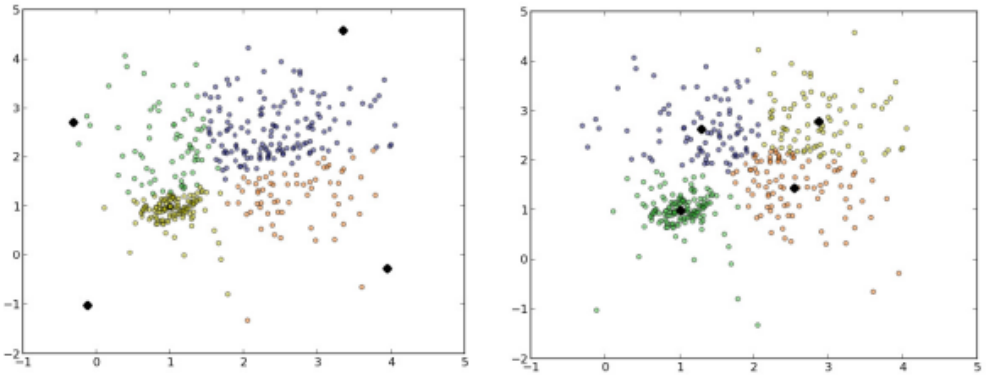

FIGURE 2.2: Simplex Volume Maximization (left picture) vs k-means. SIVM has four sparse basis vectors (black dots) and k-means has very similar centroids. The example is from a work of Drachen et al. [2]

(average of all data points in each cluster) can be very similar (in similar space) and more difficult to assign an expressive label. There are can be many challenges that need to be addressed when applying clustering algorithms to game behavior telemetry data e.g.

- The data can have a very high dimensionality, e.g. complex game interactions/mechanics from MMOGs.
- The need to mix many different behavioral data types and use of normalization methods, e.g. numeric, categorical and bi-nominal.
- Game telemetry data are often noisy and cleaning may be necessary, e.g. player pausing in a game, providing long play sessions.
- The need for defining parameters for clustering, e.g. number of clusters for k-means.
- Able to translate the findings to game developers, a knowledge that is actionable.

K-means algorithm has been shown to be very useful in behavioral analysis to obtain useful insights into in the general behaviors found in games, where AA like the SIVM algorithm can extract behavior player profiles based on the different distances to each of the extreme behaviors [2]. When choosing features for behavioral cluster analysis, Drachen et al. [8, 27] suggests that one should focus on features related to the core game mechanics and playtime. Interpreting basis vectors found by SIVM and k-means are illustrated in Figure 2.3 and Figure 2.4, taken from a very recent study by Drachen et

al. [2], authors selected playtime and leveling speed as behavioral variables showing five leveling behaviors they found in the popular MMORPG game World of Warcraft. Each basis vector is a 2.555 dimensional vector where each entry represents the level of the player each day in the last 6 years.

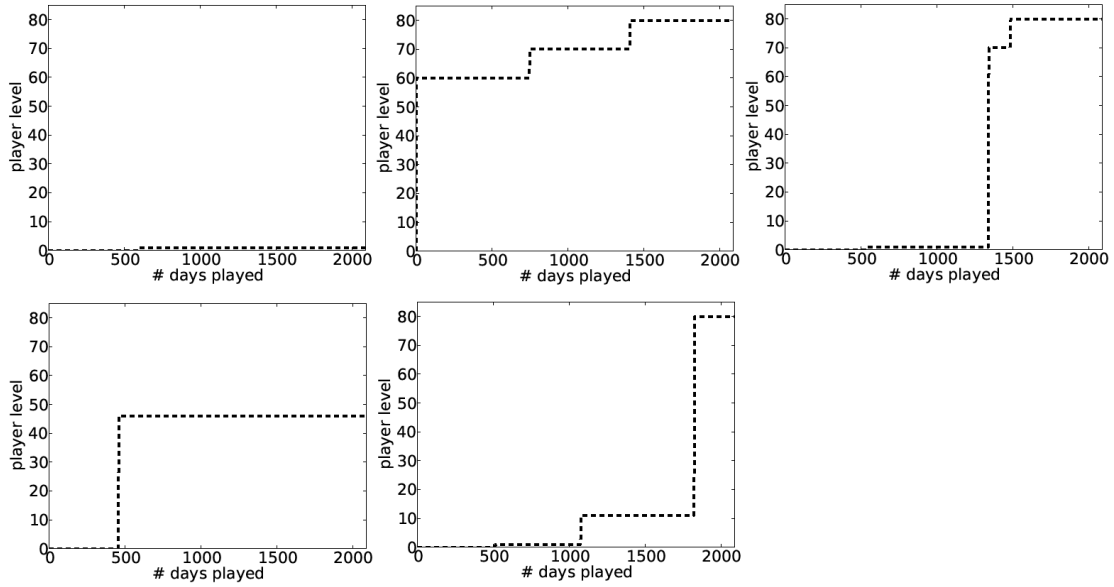

FIGURE 2.3: Five SIVM basis vectors showing extreme leveling behaviors for Archetype Analysis. Each basis vector is an actual player and is a legal playing behavior thus very easy to interpret. This figure is from [2].

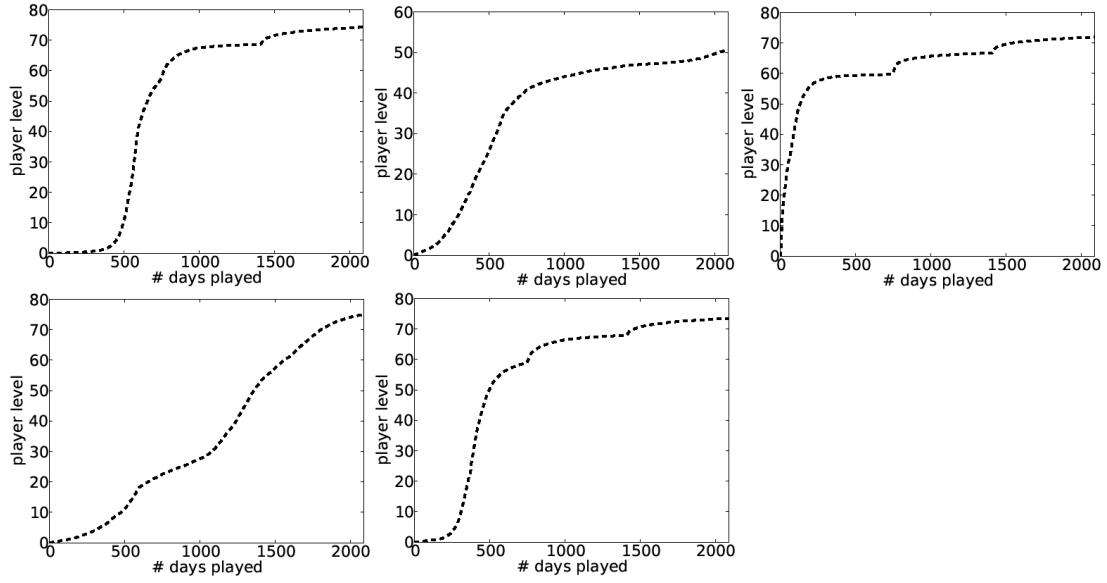

FIGURE 2.4: K-means five cluster centroids showing average leveling behaviors representing each cluster of players. They give good idea about leveling behavior of many players but some of them are also very similar, interpretion is not straightforward. This figure is from [2].

2.3 MapReduce and Large-Scale Data

MapReduce is a programming model introduced by Google in 2004 [14], built on the divide-and-conquer paradigm, dividing a large-scale data into smaller chunks and process them in parallel. MapReduce enables fault-tolerant distributed computing on large-scale datasets and is a new way to interact with *Big Data* where as old techniques are more complicated, costly and time consuming [14, 28]. Google also introduced along with MapReduce a powerful distributed file system called *Google File System* (GFS) that could hold massive amount of data. This led to a new open source software framework called Hadoop [29], written in Java and is now maintained by Apache Foundation [2], a end-to-end solution for organizations that want to apply MapReduce. There are many Hadoop-related projects at Apache including the popular scalable machine learning and data mining library called Mahout, written also in Java.

Hadoop builds on the MapReduce and GFS foundation, designed to abstract away much of the complexity of distributed processing running on large clusters of commodity computers. The MapReduce programming model allows developers to write parallel distributed programs very easily by only implementing two functions called *Map* and *Reduce*. Developers don't need to worry about doing a complicated code to e.g. distribute work to computers, internal communication, data transfers and dealing with fault tolerance. Instead they can focus on the logic to solve the problem at a hand.

2.3.1 Programming Model

As mentioned before a developer only needs to implement two functions Map and Reduce. The *Map* function takes as input pair and produces an intermediate $(key, value)$ pair. The Map functions are run in parallel and produce many intermediate output pairs which is then grouped together with the same *key* by the MapReduce framework and is passed along to the *Reduce* function. The Reduce function receives a key and all of its set of values from all the Map functions, then it merges these values and typically produces zero or one output key and value pair per Reduce invocation.

Example: The problem of counting the number of occurrences of each word in a document, solved in MapReduce [14]. The Map function receives each word as input and

[2] http://www.apache.org/

emits the word as a key with count of 1, e.g. we have set of words $w_1, ..., w_n$ in a document then the output from all Map functions would be the sequence of key value pairs: $(w_1, 1), ..., (w_n, 1)$. See the pseudo-code for the Map function in Algorithm 2.2.

Algorithm 2.2 MAP$(key, value)$

Require: key is document name, value is document contents.
Ensure: $(key, value)$ pair: (word, occurrence of one).

1: **for all** $word \in value$ **do**
2: $\quad EmitIntermediate(word, 1)$
3: **end for**

The MapReduce framework then groups and merges all the $(key, value)$ pairs from all the Map functions and invokes the Reduce function with input key as a word and list of all the Map counts as the values: $(w_i, [v_1, ..., v_n])$ where w_i is a specific word with $1, ..., n$ counts of one, e.g. $(w_1, [1, 1, 1])$ if w_1 had three occurrences in the document. The reduce function will simply sum up all the counts and output a total count for each word, Algorithm 2.3 shows the pseudo-code for Reduce.

Algorithm 2.3 REDUCE$(key, values)$

Require: key is a word, values are list of occurrences for the word.
Ensure: number of occurrences for a word.

1: $count \leftarrow 0$
2: **for all** $value \in values$ **do**
3: $\quad count = count + value$
4: **end for**
5: $Emit(AsString(key, count))$

The MapReduce also allows the developer to implement a *Combiner* function to do a partial reducing task that is executed on the same computer node as the Map function, combining the intermediate values after execution of the Map function. In the example mentioned above if a Combiner is implemented it will receive the Map output $(w_1, 1), ..., (w_n, 1)$ as input. The Combiner would then partly sum up the occurrences for each word (key) and output a intermediate sum for each word from the Map function. E.g. if w_1 had three occurrences then the output pair would be $(w_1, 3)$ instead of $(w_1, 1), (w_1, 1), (w_1, 1)$. The algorithm for a combine function can be found in Algorithm 2.4, it can look exactly like the Reduce function we saw above since it is combining/reducing the information.

Algorithm 2.4 COMBINE($key, values$)

Require: Key pair from Map, where key is a word and value is a occurrences of one.
Ensure: ($key, value$) pair: (word, partly sum of occurrences).

```
1: count ← 0
2: for all value ∈ values do
3:     count = count + value
4: end for
5: EmitIntermediate(key, count)
```

Using a Combiner reduces the information sent over the network to the Reduce functions by eliminating repetition of intermediate keys produced by the Map tasks. Input for the Reduce function is the same as before, a word and a list of all occurrences where they are now a partly sum values from all the Combiners, e.g. if w_1 has three occurrences from one Combiner and two occurrences from another, the input will be: $(w_1, [3, 2])$, instead of $(w_1, [1, 1, 1, 1, 1])$

2.3.2 MapReduce Hadoop Execution Flow

When a user program calls the MapReduce program the following sequence of occurs. MapReduce splits the input data into M pieces (typically 64MB) then starts up many copies of the user program on a cluster of machines. One of the copies is a *Master controller*, the rest are workers which each get assigned a Map or a Reduce task from the master. The master keeps track of the M Map and R (user specified) Reduce tasks statuses, scheduling of tasks, re-executing work from failed workers and responsible for messaging and information flow between workers.

Each Map task is assigned one or more chunks of the input data and executes the Map function code written by the user. The Map task output is stored on a local disk in R partitions, the master then notifies the Reduce task about the intermediate data. The Reduce task reads the data and sorts it by the intermediate keys, grouping all occurrences of the same key together. For each unique intermediate key encountered it passes the key and the corresponding occurrences to the Reduce function written by the user. Figure 2.5 illustrates the execution flow that we just described, for more details one can look at work of Dean and Ghemawat [14] and Rajaraman and Ullman [30].

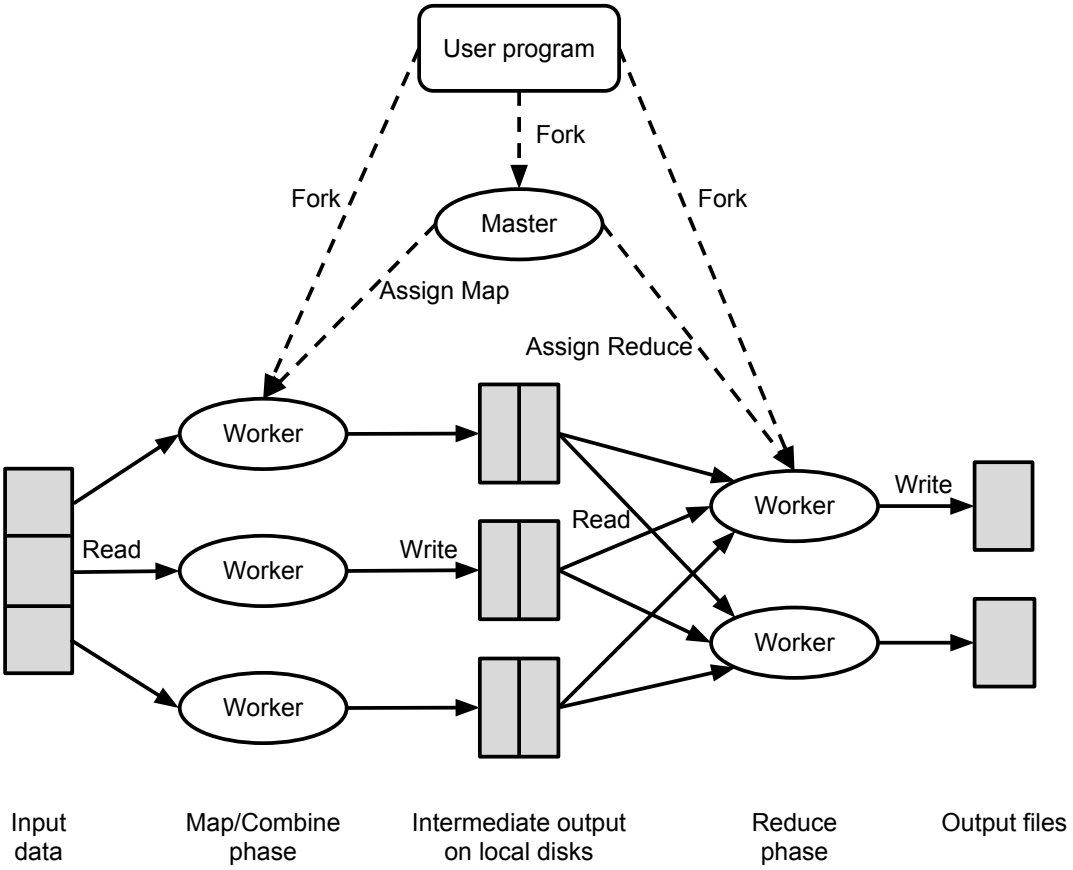

FIGURE 2.5: Overview of the MapReduce execution flow

Chapter 3

Related Work

Clustering player behaviors and processing large data sets have been researched actively in the recent years. In our work we find general player behaviors in a real life game data, which GameAnalytics contributes. Using k-means clustering algorithm to find k clusters representing average player behavior, implemented in the MapReduce framework for high scalability and parallel processing for large-scale data. The behavior of the data can change from day to day like when dealing with an endless evolving stream of data is discussed. In subsequent sections some of the related work and recent studies are given a short introduction.

3.1 Clustering Player Behaviors

Many researches have been done on clustering and predicting player behaviors over the last years to get a better understanding which kind of user behaviors are to be found when playing a game that can be actionable for game developers [4, 8, 31–33]. User behavior analysis has becoming increasingly popular in the recent years because of rise of the Free-to-Play (F2P) genre games in *Facebook* and *Google Play* where populations can be in millions creating complex game interactions [3, 4]. Playing these games are free and many are of persistent nature where the world in the game continues when a player exits. To be profitable these games drive their revenue via micro transactions, e.g. players buying upgrades or virtual items in game for real money [3, 4, 6, 9]. Major game publishers have also been collecting and analyzing large scale of behavior telemetry

data, e.g. maximizing player engagement to sell them monthly subscriptions, but details of their methods are kept confidential [7, 17]. Most available research work is case-based where a specific algorithm is applied to a specific game and commercial game data sets has only become accessible recently for academic researchers [7].

Predicting player behavior in a major commercial game Tomb Raider: Underworld (TRU) was presented in a study of Drachen et al. [12]. Authors classified 1365 players in a moderate data set into four user behavioral groups using six statistical gameplay features based on core game design as inputs of an emergent self-organizing map to identify dissimilar behavior clusters. Behavior profiles covering 90 percent of the users in the dataset were labeled in game terminology usable for game designers. Mahlmann et al. [13] did a follow up on the research using eight gameplay features and classified behavior of 10,000 players. The authors presented also how to predict behavior based on early play analysis, a popular topic which can be used to prevent attrition [9].

Analyzing social groups in the highly popular Massively Multiplayer Online Role-Playing Game (MMORPG) World of Warcraft was done by Thurau and Bauckhage [15]. They analyzed how groups (guilds) evolve over time from both American and European based guilds. Their paper is the first study analyzing such amount of data in a MMORPG, analyzing large-scale data gathered on-line from 18 million players belonging in 1.4 million groups over a period of 4 years. Convex-Hull Non Negative Matrix Factorization (CH-NMF) [33] technique was applied to the data to find the extremes rather than averages and the results show no significant cultural difference in formation processes of guilds from either the US or the EU. Interpretability of CH-NMF was more distinguishable and representing archetypal guilds than the more conventional clustering method k-means that represent the cluster centroids with similar characteristic.

Drachen et al. [8] did a clustering analysis for two major commercial games applied to large-scale of high-dimensionality player behavior telemetry. K-means and Simplex Volume Maximization (SIVM) clustering were applied to the MMORPG *Tera* and the multi-player first-person shooter strategy game *Battlefield 2: Bad Company 2*. SIVM clustering is an adaption of Archetype Analysis (AA) for large-scale data sets to find extreme player behaviors profiles [33, 34]. The authors show the contribution differences from the two algorithms where k-means gives insights into the general distribution of behaviors vs. SIVM showing players with extreme behaviors. The selection of the

most important features from the data set were followed by a method suggested by Drachen et al. [12], behavioral profiles were extracted and interpreted in terms of design language [12, 17].

In a recent study by Drachen et al. [2] the authors compare four different popular methods with purpose of clustering player behaviors and develop profiles from large-scale game metric data set from the highly popular commercial MMORPG World of Warcraft. The data set was collected from mining the Warcraft Realms site, recordings of on-line time and what level each player reached for each day in the years 2005-2010 for approx. 70 thousands of players, creating a 2.555 dimensional feature vector where each entry corresponds to the level of the player for each day. The authors selected playtime and leveling speed as their behavioral variables to show a measure of the overall player engagement in the game, where playtime is one of the most important measure for calculating the churn rate [6, 9]. Interpretable behaviors profiles where only generated by the k-means and the SIVM algorithm. The SIVM Archetype Analysis algorithm produces however significantly different behaviors that result in easier interpretation of behavior profiles compared to the k-means algorithm where the centroids are overall similar.

3.2 Clustering Large Data

K-means [20–22] is one of the most studied clustering algorithm out there and is still actively researched. It's a simple algorithm that partition the data into k partitions by minimizing its objective function sum of squared error (SSQ). From its appearance almost 50 years ago it has been one of the most popular clustering algorithm to research because its ease of interpretation and simplicity [10, 35, 36]. One of the problems with k-means is that it is a heuristic algorithm and has no guarantee to converge to a global optimum (optimal solution). Many work have been done in researching approximation guarantees (guarantee a approximated solution which is a within a constant-factor of the optimal solution) for k-means both in non-streaming and streaming versions [37–41]. In recent years k-means has also been very popular algorithm to study in the MapReduce framework where the algorithm can easily be applied to cluster large amount of data sets in parallel [14, 42–45].

Guha et al. [46] designed an algorithm in 2003 called STREAM that is based on the divide and conquer strategy to solve the k-median problem (a k-means variant), authors guarantee a approximated guaranteed solution. The algorithm divides the dataset into m pieces of similar sizes, where each of the pieces are independently clustered sequentially and all the centers from all the pieces are then clustered further. They show a new k-median algorithm called LSEARCH that is used by the STREAM algorithm and is based on a local search algorithm solving the facility location problem [47] to solve the k-median problem. Results show that STREAM LSEARCH produced near optimal quality clusters and better than STREAM k-means also with smaller variance. Compared to the the hierarchical algorithm BIRCH [48] it took 2-3 times longer to run but produced 2-3 times better quality clusters (SSQ), showing superior strength when the goal is to minimize the SSQ like detecting intrusions in networks [49].

In 2009 Ailon et al. [39] extended the the work of Guha et al. mentioned above by introducing a new single pass streaming algorithm for k-means, first of its kind with approximation guarantees. Achieving this they also designed a new algorithm called k-means# that is based on a randomized seeding procedure from the non-streaming algorithm k-means++ by Arthur and Vassilvitskii [38]. The k-means++ chooses k centers non-uniformly whereas k-means# selects $O(k \log k)$ centers and achieves a constant approximation guarantee. In the streaming algorithm they run the k-means# independently on each divided piece of the data to achieve $O(k \log k)$ random centers non-uniformly and use the k-means++ algorithm to find k centers from the intermediate centers from all the pieces of the data set.

Another approach using a *coreset* by selecting a weighted subset from the original dataset such that by running any k-means algorithm on the subset will give near similar results to running k-means on the whole dataset. Ackermann et al. [50] introduced a streaming algorithm called StreamKM++ that is a streaming version of k-means++ algorithm from Arthur et al. [38] to solve k-means on the weighted subset and a new data structure called coreset tree, speeding up the time for the sampling in the center initialization. Their approach was shown to produce similar quality of clusters (in terms of SSQ) as the STREAM LSEARCH [46] algorithm but scaling much better with number of clusters centers.

Shindler et al. [41] proposed an algorithm called *Fast streaming k-means* based on the

online facility location algorithm [51] and extends the work of Braverman et al. [40]. Authors prove that their algorithm has a much faster running time and often better cluster quality than the divide and conquer algorithm introduced by Ailon et al. [39]. The algorithm however show similar average results as StreamKM++ [50] in both running time and quality tho with better accuracy.

Clustering data streams of an unknown length, evolving over time [52, 53] are challenging where it is not possible to access historic data points because of the amount of data arriving continuously. Aggarwal et al. [54] proposed a well-known stream clustering framework called CluStream for clustering large evolving data streams and is guided by application-centered requirements. CluStream has an online component that maintains snapshots of statistical information about micro-clusters (a.k.a. *cluster feature vector* [48]) in a pyramidal time window and an offline component that uses the compact intermediate summary statistics from the micro-clusters to find higher level k clusters using k-means, in a time horizon defined by an analyst. A new high-dimensional highly scalable data stream clustering algorithm called HPStream was also proposed [55]. HPStream uses projected clustering [56], which can determine clusters for a subset of dimensions, to data streams and a new data structure called *fading cluster structure* that allows historical and current data to integrate nicely with a user-specified fading factor.

Another approach clustering evolving streams Zhou et al. [57] presented a algorithm called SWClustering to cluster evolving data streams over so called sliding windows, where the most recent records are considered to be more critical than historic data [58]. Allowing to analyze the evolution of the individual clusters by eliminating influence by outdated historic data points when new data points arrive. Authors show that the CluStream algorithm [54] is much more sensitive to influences of outdated data and is less efficient.

Processing large of amount of data efficiently using parallel processing is an active research and gain much of popularity when the Google's MapReduce programming model was introduced by Dean and Ghemawat [14]. Zhao et al. [42] implemented a parallel version of k-means (PKMeans) in the MapReduce framework and showing their algorithm can be effectively run on large data sets. The Map function calculates the distance to the

closest cluster centroid for each data point at a time, after assigning to the clusters a combiner sums up all the data points dimensions for each cluster from that Map function and outputs the key and value pair $(clustercentroid, [sumforalldimensions, numberofpoints])$. The Reduce function then sums up all the intermediate sum values for each cluster and calculates the mean for the new centroids.

Li et al. [59] implemented the algorithm MBK-means in MapReduce using the Bagging ensemble learning method [60], using replacement sampling to generate k new data sets from the original data. K-means algorithm clusters each new data set using the MapReduce framework until convergence and in the end all the centroids from the k sets are merged to form the final k centroids.

Many extensions on the traditional MapReduce framework have been proposed to support efficient algorithms running iteratively [61–66] and incrementally [66–68]. The incremental MapReduce frameworks are interesting and relates to our work since we are incrementally clustering chunks of data but are not under study in this thesis.

3.3 Our study

The algorithm in this thesis is most similar to PKMeans [42] described above, a parallel k-means implementation in MapReduce using a Combine function to reduce the intermediate data sent between the mappers and the reducers. We however implement a parallel k-means algorithm in MapReduce so that each Map function efficiently calculates the distance to nearest cluster centers by calculating the distance matrix between all data points and the cluster centers instead of processing each data point separately. We apply the algorithm to incrementally but non-iteratively cluster player behaviors on theoretically large data sets that arrive daily.

In our work we show k-means is a good approach to provide valuable insights into the general of behaviors for a specific real game data provided by GameAnalytics [16]. Work of Drachen et al. [2, 8] use k-means to cluster player behaviors and was a inspiration to our study such as interpretation of cluster centroids and approaches to select and build important game behavioral variables from user's telemetry and extracting behavioral profiles. Additional experiments were performed with a controlled data set having

different normal distributions of data in separate datasets to experiment with the idea of different distributed data arriving each day.

Chapter 4

Methodology

In this chapter we introduce and describe the real life game dataset that we got from GameAnalytics (GA), selection of user events and building player behavior features to construct a game metric data as input to our clustering algorithm. In GA the user telemetry data arrives in mini batches from the games, a batch each day. In this thesis the idea is then to process each batch and incrementally cluster the data, where the output from one batch is the input to the next.

We implemented a incremental MapReduce (MR) k-means clustering algorithm and apply it on multiple batches of the game metric data to find average player behaviors described by a the constructed player behavioral feature vector. The algorithm is highly scalable and can easily be executed on numerous virtual computers using the Amazon Elastic MapReduce (Amazon EMR) web service.

We test the MR k-means algorithm on multiple batches of the real game data. Where each batch of game data is processed in one iteration and the output is used as the input to the next batch. Using one iteration we allow for efficient computation and given the changing nature of the data over time in each batch we don't risk falling into local optimum. This method is compared against multiple iterations on the real game data and on a larger generated example, where 3 normal distributions shift between data batches.

Different MR k-means implementation were tried in the process of the thesis and different efficient computational methods were compared on a larger generated dataset,

including scalability (scale-up and scale-out) tests running on different cluster setups in the Amazon EMR.

4.1 Dataset

The dataset is from a Free-to-Play (F2P) online game that is available on Google Plus and Facebook. This real life game dataset is from one of the many games that GameAnalytics is analyzing for their customers. Because of confidentiality issues we are not allowed to mention its name but lets call it *Free Battle Online* (FBO). Same is about the real names of events or features in the dataset is fictional, this unfortunately means that we also have very limited knowledge about the data, e.g. what individual events mean and the timespan. The goal of FBO is to fight battles against both non-playing-characters (NPCs) and other players. You can click on various places to travel to finish battle missions or just visit other players and initiate battles to steal their resources.

The dataset is a user telemetry data, e.g. user behavioral events, where each line represents an action performed by the user or is a direct influence from a user behavior. There are over 5.000.000 rows in this data set, with approx. 550 unique events generated by approx. 94.000 players. Average events per player is $\mu = 54$ with standard deviation $\sigma = 145$, indicating that the data is spread out over a wide range of values. The measure of the spread is in Table 4.1 showing the five number summary; The first quartile Q_1 cuts off the lowest 25% of the data, the second quartile gives the center of the distributions known as the $Median$, the third quartile Q_3 cuts of the highest 25%. The median is lower than the mean of the data describing a positively skewed distribution, see Figure 4.1 for a visualization of the distribution.

	Min	Q_1	$Median$	Q_3	Max
Events	1	4	23	61	10865

TABLE 4.1: This table shows the five number summary about the number of events generated by each unique player in the dataset.

The two lines outside the box in Figure 4.1 are called whiskers and represent the extreme low and high values that are less than $1.5 \times IQR$ beyond the quartiles. The first and third quartiles represent the ends of the box and the median is the red line. The red *squares* are individual points called *outliers*. Only about 6% or $\approx$ 5500 players of the entire population generated more than 150 events, 0.3% or $\approx$ 300 players have more than

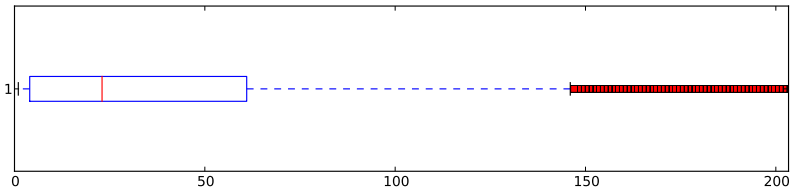

FIGURE 4.1: A boxplot visualizing the Events distribution incorporating the five number summary.

1000 events and less than ten people have a range of $5.000 - 10.000$ generated events! A histogram showing the frequency of events generated by players is shown in Figure 4.2, zooming in on the spread that gives the range covered by the middle half of the data.

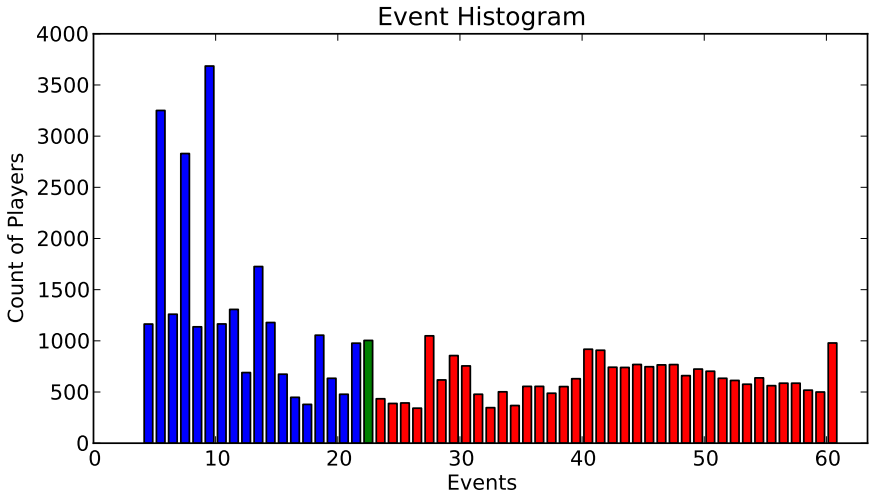

FIGURE 4.2: A histogram for number of events using singleton buckets in the interquartile range (IQR) defined as $Q3 - Q1$, range covered by the middle half of the data.

4.1.1 Game Metric Construction

For this thesis we tried to keep the number of game features in minimal and decided to build our feature vector with three features. Given the limited knowledge and information that we have about individual events we decided to select events that have high frequency compared to other events and hopefully could describe different behaviors between groups of players in those features. A $3 - dimensional$ feature vector was constructed for each unique player, containing a frequency of three events generated by the player. The aggregated events are:

- Login: Number of logins/access into different areas in the game($\mu = 2.4, \sigma = 4.8$)

- Battle: Number of battles this player have started ($\mu = 3.7, \sigma = 9.6$)
- Premium Spending: Number of times this player spends a in-game money on virtual items or resources ($\mu = 1.1, \sigma = 2.7$)

We tried to chose features that could describe player engagement to the game. How active the player is exploring various areas, fighting battles and spending in-game money to advance in the game quicker. The five number summary of the skewed distribution is in Table 4.2. Boxplots visualizing the distributions for these events are shown in relevant figures below, Logins in Figure 4.3, Battles in Figure 4.4 and Premium spent events in Figure 4.5.

	Min	Q_1	*Median*	Q_3	*Max*
Logins	0	1	1	2	220
Battles	0	0	1	4	439
Premium spent	0	0	0	1	100

TABLE 4.2: This table shows the five number summary for the Logins, Battles and Premium spent events generated by each unique player in the dataset.

The features values are spread out over large range of values, in the preprocessing Section 4.1.2 we explain how we detect and remove the outliers or anomalies from our data before we apply our MR k-means algorithm using a *z-score* method.

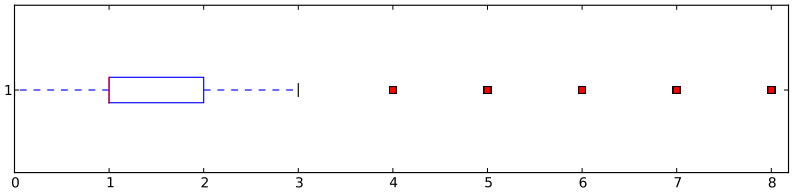

FIGURE 4.3: A boxplot visualizing the Login events distribution incorporating the five number summary.

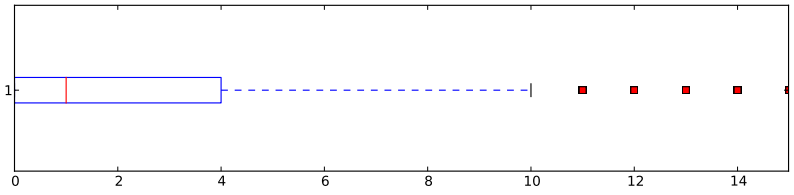

FIGURE 4.4: A boxplot visualizing the Battle events distribution incorporating the five number summary.

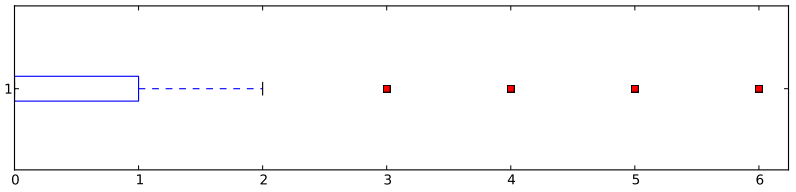

FIGURE 4.5: A boxplot visualizing the Premium spent events distribution incorporating the five number summary.

4.1.2 Preprocessing

User telemetry from games can be very noisy in that sense it can contain lots of irrelevant events and logging information and even developers from the game studio can generate events. For this dataset this we performed several preprocessing steps:

- Sort the events after event timestamp.
- Split the dataset into 10 approx. equally sized data set pieces.
- Filter out irrelevant events and logging information. E.g. Events not origin from Facebook or Google Plus, non related events and in game studio debugging generated events.
- For each data set piece:
  - Counting the events mentioned above per user to build our feature vector.
  - Remove outliers or anomalies from the data.
  - Standardize the data to standard scores also called *z-score*. Transforming the data to standard normal distribution, with $\mu = 0$ and $\sigma = 1$.

To apply our incremental MR k-means algorithm we need to have multiple batches of data to process incrementally. We use the real game dataset that we got from GA and split it up into 10 equally sized data batches sorted after the event arrival timestamp. By doing so we try to simulate that we are processing multiple batches of data in the same order as the events were generated over mulitple time-periods or days, like at GA. Next we filter out irrelevant information, e.g. information that don't contribute as a event logging or are obviously from the game designers themselves.

When processing each data batch we aggregate or count the events that we selected before to build our feature vector for each unique player in the data set. Next we find the

outliers by transforming the data to standard scores using the *zero mean normalization* (ZMN), also called *z-score*, with standard normal distribution having zero mean and unit variance $\mu = 0$ and $\sigma = 1$. A feature vector that has a feature value z-score that is more than 3 standard deviations away from the mean is considered an outlier in the data and is removed from the original data batch. Having a data batch without outliers we now can transform the data again using ZMN, without having the outliers to effect our transformation. The ZMN is defined as follows

$$z = \frac{x - \mu}{\sigma}$$

Using ZMN is widely used in machine learning algorithms and is needed when calculating the distance between two points (feature vectors) such that each feature contributes approximately equally to the final distance.

Removing the outliers from the whole dataset, using the ZMN method described above, the total number of unique players become approx 90.800 instead of approx. 94.000, removing $\approx 4\%$ of the population. The five number summary for the whole dataset can be found in Table 4.3, notice the range of values compared to the Table 4.2 in Section 4.1.1. The multiple data batches that we feed into our clustering algorithm had $\approx 5\%$ player feature vectors removed on average, see Table 4.4 below.

	Min	Q_1	*Median*	Q_3	*Max*
Logins	2	1	1	2	16
Battles	0	0	1	4	32
Premium spent	0	0	0	0	9

TABLE 4.3: This table shows the five number summary for the Logins, Battles and Premium spent events generated by each player from the whole dataset with outliers removed.

	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
Players	14054	13365	13403	13503	13441	13276	12839	13413	13473	13412
Outliers	658	741	661	673	748	698	685	724	669	680
% removed	4,68%	5,44%	4,93%	4,98%	5,56%	5,25%	5,33%	5,39%	4,96%	5,07%

TABLE 4.4: This table shows the player population, number of outliers found and the percentage of the population removed, for each data batch.

4.2 K-means algorithm in MapReduce

A k-means algorithm was implemented in MapReduce (MR) that is able to incrementally process large-scale datasets in parallel when running on e.g. Amazon EMR. We implemented two versions of MR k-means that use different MR execution flows. We first started to implement a naive and a common implementation version of MR k-means that assigns each point (feature vector) to a nearest cluster in the *Map* phase and updates the centroids for the clusters in the *Reduce* phase. The programmer only needs to fill in two functions, the Map and Reduce. The other version has a *Combiner* phase implemented that performs a reduce work on the same computer node as the Mapper, by calculating intermediate sums of all the points for each cluster from a Map function. This minimizes the data needed to be transferred and shuffled by the MR framework to the Reducer, sending only an intermediate sum with each centroid from each mapper instead of list of all data points belonging to each cluster [14]. The second implementation using a Combiner is used in the experiments (with some efficiency adjustments) where we cluster the real game data, since it is a more efficient and scalable when processing larger amount of data as explained in Section 2.3 and Section 4.2.2 and as result show from experiments in Chapter 5.

In the next two subsections we will describe the two implementations and their pseudo codes. Then we will explain the *Incremental MRK-means Program* that controls the incremental data batch processing by executing n iterations of the MRK-means job. The most computational expensive part in k-means is to find the nearest centroid for a given feature vector by calculating the distance between the two. We implemented three different distance calculation versions to find the most efficient one when running on a cluster setup using Amazon EMR. In the last subsection we give a quick overview of the development environment and tools that were used in this thesis.

4.2.1 K-means version 1 - Map and Reduce

Implementing the Map and a Reduce function version is the most common implementation of k-means in MR. Where The Map function calculates the nearest centroids to each data point and emits that pair to the Reduce function that sums up all data point values for each centroid and outputs updated centroids. Our Map function is called for each

player feature vector found in the data batch and we iterate through all centroids that we have in memory to calculate the distance to the nearest one. The nearest centroid and its data feature vector is emitted from all the Map tasks running in parallel to the MR framework that shuffles and sorts all feature vectors together belonging to the same centroid. The pseudo-code for the Map function can be found in Algorithm 4.1 and for Reduce in Algorithm 4.2.

Algorithm 4.1 KMEANS VERSION 1: MAP($key, value$)

Require: $key = document\_name$, $value = feature\_vector$
Ensure: $key = centroid\_id$, $value = feature\_vector$.

$current\_centroids \leftarrow C = \{c_1 ... c_k\}$ {loaded into memory}
$feature\_vector \leftarrow= value$
$mindist \leftarrow +\infty$
$minCentroid \leftarrow None$
for all $centroid \in current\_centroids$ **do**
    $distance \leftarrow getDistance(feature\_vector, centroid)$
    if $distance < mindist$ **or** $minCentroid = None$ **then**
        $mindist \leftarrow distance$
        $minCentroid \leftarrow centroid$ {nearest centroid}
    end if
end for
$EmitIntermediate(minCentroid, feature\_vector)$

Algorithm 4.2 KMEANS VERSION 1: REDUCE($key, value$)

Require: $key = centroid\_id$, $values = feature\_vector\_list$
Ensure: $key = centroid\_id$, $value = average\_feature\_vector$.

$feature\_vector\_list \leftarrow values$
$sum\_feature\_vector \leftarrow [0, 0, 0]$
for all $feature\_vector \in feature\_vector\_list$ **do**
    $sum\_feature\_vector \leftarrow sum\_feature\_vector + feature\_vector$
end for
$count\_feature\_vectors \leftarrow getCount(feature\_vector\_list)$
$average\_feature\_vector \leftarrow sum\_feature\_vector \div count\_feature\_vectors$
$EmitIntermediate(key, average\_feature\_vector)$

The Reduce function then gets one input per centroid and all its feature vectors. Iterating through all the feature vectors, summing up their values and divide with their count to get a new updated centroid and outputs that centroid or the mean that represents the average feature vector for the cluster population.

4.2.2 K-means version 2 - Map, Combiner and Reduce

By adding the Combiner phase in between the Map and Reduce phase can reduce a large amount of overhead in data transfer and computation in the MapReduce framework. The Combiner phase is executed on the same computer as the Map phase and is thought as a reduce phase inside the Map phase. The Combiner function takes input directly from the Map function and calculates intermediate results that is emitted over the network instead of Map emitting a key and value pair for each feature vector, which can be problematic for large datasets.

Algorithm 4.3 KMEANS VERSION 2: COMBINE($key, value$)

Require: $key = centroid\_id$, $values = feature\_vector\_list$
Ensure: $key = centroid\_id$, $value = [sum\_feature\_vector, count\_feature\_vectors])$.

```
1: feature_vector_list ← values
2: sum_feature_vector ← [0, 0, 0]
3: for all feature_vector ∈ feature_vector_list do
4:     sum_feature_vector ← sum_feature_vector + feature_vector
5: end for
6: count_feature_vectors ← getCount(feature_vector_list)
7: EmitIntermediate(key, [sum_feature_vector, count_feature_vectors])
```

In this version we use the same Map function as described above in Algorithm 4.1 but we need to change the Reducer. The Combiner function however becomes very similar to the old Reducer in Algorithm 4.2. The only difference being that the Combiner doesn't compute the average feature vector or the mean and emits little more complex key and value pair, where $key = centroid\_id$ and $value = [sum\_feature\_vector, count\_feature\_vectors]$, see pseudo-code in Algorithm 4.3.

The new Reducer algorithm then sums up the intermediate sums and counts from the Combiners and calculates the average feature vector. The reducer works with much smaller data as it receives a sum of values per centroid in stead of a long list of feature vectors, this is of great help when working with large data, see the Reducer pseudo-code in Algorithm 4.4. We will also discuss the Combiner vs. non-Combiner better in the results in Chapter 5.

Algorithm 4.4 KMEANS VERSION 2: REDUCE($key, value$)

Require: $key = centroid\_id$, $values = [sum\_feature\_vector, count\_feature\_vectors]$)
Ensure: $key = centroid\_id$, $value = average\_feature\_vector$.

```
sum_and_count_list ← values
sum_total_feature_vector ← [0, 0, 0]
count_total ← 0
for all sum_and_count ∈ sum_and_count_list do
    sum_vector ← getSumVector(sum_and_count)
    count ← getCount(sum_and_count)
    sum_total_feature_vector ← sum_total_feature_vector + sum_vector
    count_total ← count_total + count
end for
average_feature_vector ← sum_total_feature_vector ÷ count_total
EmitIntermediate(key, average_feature_vector)
```

4.2.3 Computing Efficiently

The final k-means version uses a more efficient way of calculating the distance to the nearest centroid for each point. Using a matrix calculation that returns a distance matrix that represents distances from all points to all centroids. These different versions are compared below in respect to time and space complexity.

When calculating the distance between two points or vectors we calculate the Euclidean distance between them. The most computational expensive work in k-means is when we need to find the nearest centroid for a point, that is the centroid that has the minimum Euclidean distance. The naive version when calculating the distance in k-mean is to calculate the distance to each of the centroids one at a time by iterating over all the features or attributes in the vector and summing up their squared error. As in shown earlier in Subsection 2.2.1, the Euclidean distance is defined as follows

$$dist(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}.$$

Trying to make these calculations more efficient we implemented two other versions how we calculate the Euclidean distance. First we used *vectorized* approaches found in the Python library called NumPy. By grouping simple element-wise numerical operations together to perform much more efficiently on arrays. The NumPy operations are implemented in the *C* programming language, giving them very high speed improvement. When applying vectorisation we don't have to have a *for loop* that iterates through all

centroids, we simply calculate the distances to all centroids in one command by passing to the method the feature vector and array or a matrix of centroid vectors. The larger the data is and more centroids we have the slower the naive solution will be perform but vectorisation is a safe bet.

The second version we implemented was also using array operations but doing things a bit differently. The idea is to have all the data and the centroids in memory then call one operation to get the similarity or dissimilarity matrix for all the data points to all the centroids, easily to identify what is the nearest centroid (with the minimum distance) for all data points in one operation. This method performed amazingly fast when we implemented it and executed it locally without the MapReduce version.

Implementing it in MapReduce we needed to use the Map function to gather one point at a time and store in an array in memory. Then we implement a function that is called by the framework directly after the Map function finishes and before the Combine function. In that function called *Map Final* we calculate the distance matrix between all the data points and the centroids in memory, emitting each centroid and its nearest points. Then the Combine function will be called and sums up all the values for each centroid as described before.

However there is a problem with this solution that it needs to hold everything in memory and whereas the Map phase in MapReduce framework is designed to process one point at time and solve the computational problem for large data in parallel, e.g. splitting the data and run on hundreds or thousands of computers.

We had problems running this implemention on larger data sets when performing experiments on a cluster in the Amazon EMR framwork, we had a memory fail in the Mapper. Thus the first vectorisation approach mentioned in above was used in our experiments. For comparison results between these versions see our scalability results in Chapter 5.

Analyzing the memory complexity for the both of the efficient versions we implemented. For our second version when calculating the Euclidean distance by storing everything in memory would be first need to store the data point vectors in memory from the Map phase with memory complexity $O(\frac{N \times d}{s})$ where s is the number of splits/Mappers, N is number of data points and d is the dimensionality. Next we generate the distance matrix that costs $O(\frac{N \times k}{s})$ and storing the nearest centroids for each point $O(\frac{N}{s})$,

next we create boolean arrays to extract the set of points for each k that we want to emit to the reducers, approx. $O(\frac{N \times k}{s})$. Putting this together we would need at least $O(\frac{N \times d}{s} + \frac{N \times k}{s})$ memory, compared to the first vectorisation method we would need about $O(d)$ for the incoming data point and $O(k)$ for the distances to the centroids with total memory complexity of $O(d + k)$.

4.3 Incremental MRK-means Program

This program controls the incremental MR k-means data batch processing execution flow. It executes on a data batch basis, when a data batch is ready to be processed then this program is called and it returns the current intermediate means found. When the next data batch is ready the program is executed and expects to receive the previous found intermediate means as initial means for the current incoming data batch. The program expects following inputs:

- *input_file*: Path to data batch file (can also be an Amazon S3 URI). The file is read by the MapReduce framework, split and distribute the data batch to the Mappers.
- *initial_means*: Path to a means file. If running the program for the first time then these means must be selected from the data batch. Else this is the intermediate means output from the previous data batch processing.
- *k_means*: Number of centroids that we want to find in the data batch.
- *max_iterations*: The number of iterations to run k-means on the data batch.
- *threshold_delta*: A value that defines the minimum centroids difference/movement found between iterations. (e.g. 0.01)

If setting the maximum iterations for a data batch too high then there is a risk of k-means adjusting the intermediate means to close to the distribution of the current data batch and can be costly when they are used as input to the next data batch that has a slightly different distribution. See Algorithm 4.5 for the pseudo-code of the program.

Algorithm 4.5 IncrementalMapReduceKmeans()

Require: $input\_file$, $initial\_means$, k_means, $max\_iterations$, $threshold\_delta$
Ensure: $means$

```
1: intermediate_means ← initial_means
2: for i to max_iterations do
3:     MRjob_parameters ← [input_file, intermediate_means, k_means]
4:     output ← runMRJob(MRjob_parameters)
5:     means ← getMeans(output)
6:     means_difference ← dist(intermediate_means − means)
7:     if means_difference < threshold_delta then
8:         return means
9:     else
10:        intermediate_means ← means
11:    end if
12: end for
13: return intermediate_means
```

The main steps in the execution flow for the incremental MR k-means process, see Figure 4.6, are described as following:

- 1. Input: Means or centroids found from previous data batch processing iteration and the current batch of data to be processed.
- 2. MR k-means: Run n-iterations of MR k-means job. In each iteration; The Map/Combine phase assigns data points to nearest means, the Reduce phase updates the means of the assigned points. If more than 1-iteration is performed then the intermediate means from the previous iteration is used as input to the MR k-means job.
- 3. Output: After n-iterations the program outputs the means found for the current data batch.
- 4. Process: If there is multiple data batches to be processed then the program is run again with means from step 3. as input and the next data batch file.

4.4 Development Environment and Tools

The MapReduce k-means algorithm was implemented using a MR development framework called *mrjob* [1]. Mrjob is a open-source Python framework actively maintained

[1] http://pythonhosted.org/mrjob/

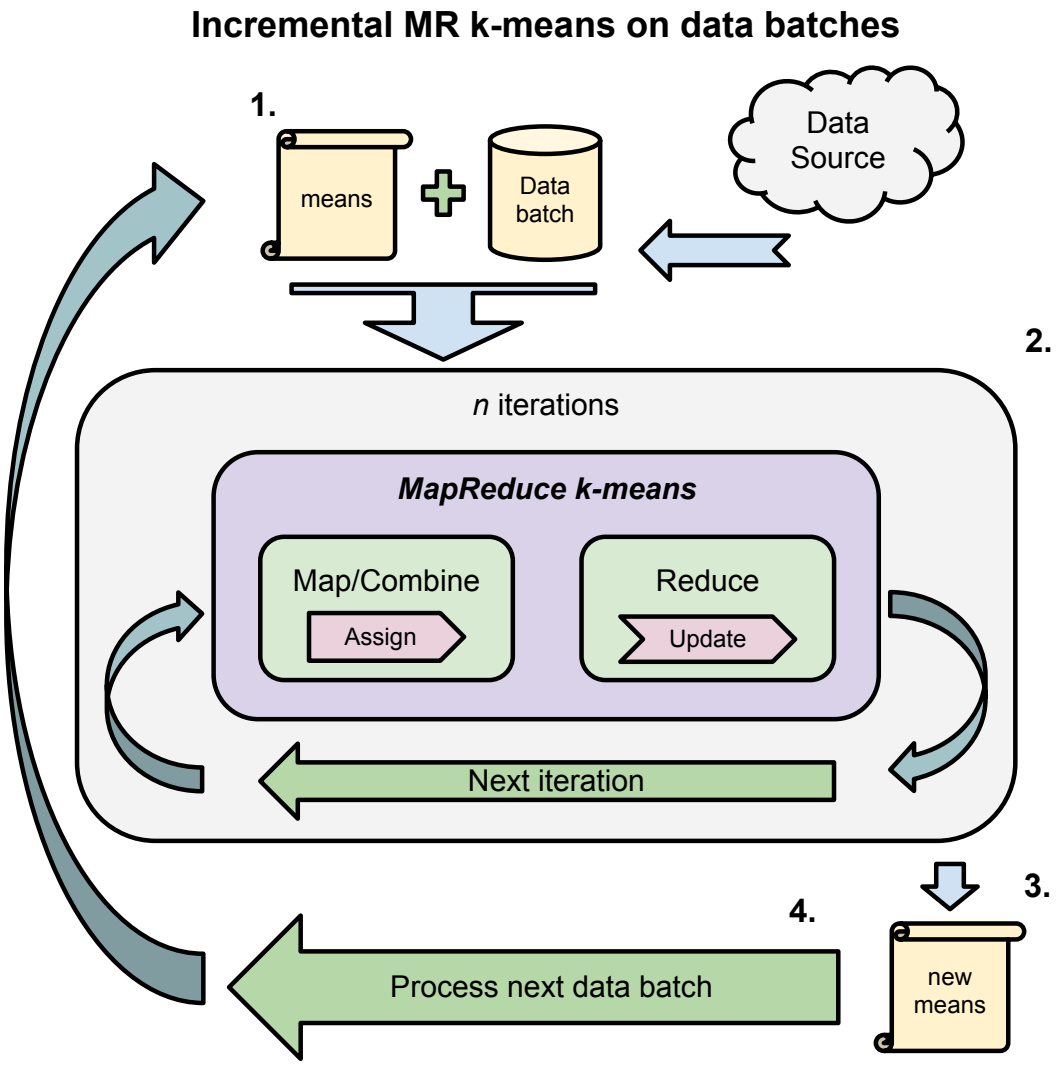

FIGURE 4.6: The incremental MR k-means execution flow.

by Yelp [2], that allows MapReduce jobs to be written in Python 2.5+ and executed on several platforms. Using mrjob allows for rapid implementation of MapReduce jobs by running them locally for development purposes and easily run them on your own Hadoop cluster or even easier using the Amazon Elastic MapReduce (Amazon EMR)[3]. Amazon EMR is a web service that allows developers to buy time on a Hadoop cluster to process large-scale data easily and cost-effectively. For the experiments in this thesis we ran our MR k-means algorithm on Amazon EMR web service for scalability tests.

The Python programming language was chosen for this study because GameAnalytics (GA) also uses Python and mrjob to implement their MapReduce jobs. Allowing GA

[2] http://opensource.yelp.com/
[3] http://aws.amazon.com/elasticmapreduce/

easily to use and build further on the implementation from this study. In this thesis we used the NumPy/SciPy Python library that enables vectorisation approaches and array/matrix data structures for easier and more efficient data vector manipulations. The implementation of algorithms and running experiments were executed on a Linux operating system, recommended by GA instead of using a Windows OS, because of Python libraries compatibility and mrjob.

Chapter 5

Results and Discussion

5.1 Results

TODO Show results with relevant pictures and what they mean

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

5.2 Discussion

TODO Describe the results with more detailed explanations

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis

egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Chapter 6

Conclusions

6.1 Conclusions

TODO Write Conclusions. Convince the reader that the research question was answered/solved. Write what is relevant to the research question. Use short statements directly related to the research question.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes,

nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6.2 Summary of Contributions

TODO Describe the contributions may overlap Conclusions (Note: maybe move into Conclusions section). Short numbered statements.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6.3 Future Research

TODO What is the future work. What can be done differently, what needs to be addressed?

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] Anders Drachen, Magy Seif El-Nasr, and Alessandro Canossa. Game analytics - the basics. In Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa, editors, *Game Analytics*, pages 13–40. Springer London, 2013. ISBN 978-1-4471-4768-8. doi: 10.1007/978-1-4471-4769-5˙2.

[2] A. Drachen, C. Thurau, R. Sifa, and C. Bauckhage. A comparison of methods for player clustering via behavioral telemetry. In *Foundations of Digital Games 2013*, 2013.

[3] Jun H. Kim, Daniel V. Gunn, Eric Schuh, Bruce Phillips, Randy J. Pagulayan, and Dennis Wixon. Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 443–452, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: 10.1145/1357054.1357126.

[4] Anders Drachen and Alessandro Canossa. Evaluating motion: Spatial user behaviour in virtual environments. *International Journal of Arts and Technology*, 4 (3):294–314, 2011. doi: 10.1504/IJART.2011.041483.

[5] Randy J. Pagulayan, Kevin Keeker, Dennis Wixon, Ramon L. Romero, and Thomas Fuller. User-centered design in games. In Julie A. Jacko and Andrew Sears, editors, *The human-computer interaction handbook*, chapter User-centered design in games, pages 883–906. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003. ISBN 0-8058-3838-4. URL `http://dl.acm.org/citation.cfm?id=772072.772128`.

[6] M Seif El-Nasr and Canossa A Drachen A. *Game Analytics: Maximizing the Value of Player Data*. Springer, 2013. ISBN 978-1-4471-4768-8.

[7] Geogios N. Yannakakis. Game ai revisited. In *Proceedings of the 9th conference on Computing Frontiers*, CF '12, pages 285–292, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1215-8. doi: 10.1145/2212908.2212954.

[8] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 163–170, 2012. doi: 10.1109/CIG.2012.6374152.

[9] Tim Fields and Brandon Cotton. *Social Game Design: Monetization Methods and Mechanics*. CRC Press, 12 2011. ISBN 978-0240817668.

[10] Rui Xu and II Wunsch, D. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005. ISSN 1045-9227. doi: 10.1109/TNN.2005.845141.

[11] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[12] A. Drachen, A. Canossa, and G.N. Yannakakis. Player modeling using self-organization in tomb raider: Underworld. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 1–8, 2009. doi: 10.1109/CIG.2009.5286500.

[13] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G.N. Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 178–185, 2010. doi: 10.1109/ITW.2010.5593355.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251254.1251264`.

[15] C. Thurau and C. Bauckhage. Analyzing the evolution of social groups in world of warcraft ő. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 170–177, 2010. doi: 10.1109/ITW.2010.5593358.

[16] GameAnalytics Aps. Data and analytics engine for game studios, 2013. URL http://www.gameanalytics.com.

[17] G Zoeller. Game development telemetry. In *Proceedings of the Game Developers Conference*, 2010.

[18] Anders Drachen, Christian Thurau, Julian Togelius, GeorgiosN. Yannakakis, and Christian Bauckhage. Game data mining. In Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa, editors, *Game Analytics*, pages 205–253. Springer London, 2013. ISBN 978-1-4471-4768-8. doi: 10.1007/978-1-4471-4769-5˙12.

[19] Rui Xu and Don Wunsch. *Clustering*. Wiley-IEEE Press, 2009. ISBN 9780470276808.

[20] E. W. Forgy. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965. URL http://ci.nii.ac.jp/naid/10009668881/en/.

[21] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, pages 281–297, 1967. URL http://www-m9.ma.tum.de/foswiki/pub/WS2010/CombOptSem/kMeans.pdf.

[22] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489.

[23] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1):84–95, 1980. ISSN 0090-6778. doi: 10.1109/TCOM.1980.1094577.

[24] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991. ISBN 0-7923-9181-0.

[25] J. Han, M. Kamber, and J. Pei. *Data Mining, Second Edition: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2006. ISBN 9780080475585. URL http://books.google.dk/books?id=AfL0t-YzOrEC.

[26] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5103-0.

[27] Anders Drachen and Alessandro Canossa. Towards gameplay analysis via gameplay metrics. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, MindTrek '09, pages 202–209, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-633-5. doi: 10.1145/1621841.1621878.

[28] R.D. Schneider. *Hadoop for Dummies*. John Wiley & Sons Canada, Limited, 2012. ISBN 9781118250518. URL http://books.google.ca/books?id=0xhYLwEACAAJ.

[29] Andrzej Bialecki, Michael Cafarella, Doug Cutting, and Owen OŠMALLEY. Hadoop: a framework for running applications on large clusters built of commodity hardware. 11, 2005. URL http://lucene.apache.org/hadoop.

[30] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011. ISBN 1107015359, 9781107015357.

[31] Tim Marsh, Shamus Smith, Kiyoung Yang, and Cyrus Shahabi. Continuous and unobtrusive capture of User-Player behaviour and experience to assess and inform game design and development. In *1st World Conference for Fun 'n Games*, Preston, England, 2006.

[32] Olana Missura and Thomas Gärtner. Player modeling for intelligent difficulty adjustment. In João Gama, VítorSantos Costa, AlípioMário Jorge, and PavelB. Brazdil, editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 197–211. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04746-6. doi: 10.1007/978-3-642-04747-3˙17. URL http://dx.doi.org/10.1007/978-3-642-04747-3_17.

[33] C. Thurau, K. Kersting, and C. Bauckhage. Convex non-negative matrix factorization in the wild. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 523–532, 2009. doi: 10.1109/ICDM.2009.55.

[34] K. Kersting, M. Wahabzada, C. Thurau, and C. Bauckhage. Hierarchical convex nmf for clustering massive data. In Qiang Yang Masashi Sugiyama, editor, *Proceedings of the 2nd Asian Conference on Machine Learning (ACML–10)*, Tokyo, Japan,

Nov 8–10 2010. URL http://www-kd.iai.uni-bonn.de/pubattachments/477/kersting10acml.pdf. draft.

[35] Anil K. Jain. Data clustering: 50 years beyond k-means. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD '08, pages 3–4, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87478-2. doi: 10.1007/978-3-540-87479-9˙3.

[36] Lior Rokach. A survey of clustering algorithms. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 269–298. Springer US, 2010. ISBN 978-0-387-09822-7. doi: 10.1007/978-0-387-09823-4˙14.

[37] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, SCG '02, pages 10–18, New York, NY, USA, 2002. ACM. ISBN 1-58113-504-1. doi: 10.1145/513400.513402.

[38] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL http://dl.acm.org/citation.cfm?id=1283383.1283494.

[39] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. *Advances in Neural Information Processing Systems*, 22:10–18, 2009. URL http://www1.cs.columbia.edu/~rjaiswal/ajmNIPS09.pdf.

[40] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 26–40. SIAM, 2011. URL http://dl.acm.org/citation.cfm?id=2133036.2133039.

[41] Michael Shindler, Alex Wong, and Adam W. Meyerson. Fast and accurate k-means for large datasets. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*

24, pages 2375–2383. NIPS, 2011. URL http://books.nips.cc/papers/files/nips24/NIPS2011_1271.pdf.

[42] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 674–679, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-10664-4. URL http://dx.doi.org/10.1007/978-3-642-10665-1_71.

[43] Makho Ngazimbi. Data clustering using mapreduce. Master of science in computer science, Boise State University, March 2009. URL http://cs.boisestate.edu/~amit/research/makho_ngazimbi_project.pdf.

[44] Georgios Christopoulos. Fast, parallel stream clustering using hadoop online. Diploma thesis, Technical University of Crete, July 2011. URL http://titan.softnet.tuc.gr:8080/softnet/GetFile?FILE_TYPE=PUB.FILE&FILE_ID=201.

[45] Grace Nila Ramamoorthy. K-means clustering using hadoop mapreduce. Msc advanced software engineering in computer science, University College Dublin, September 2011. URL http://www.resumegrace.appspot.com/pdfs/kmeansCluster.pdf.

[46] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O"Callaghan. Clustering data streams: Theory and practice. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):515–528, 2003. ISSN 1041-4347. doi: 10.1109/TKDE.2003.1198387.

[47] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 378–388, 1999. doi: 10.1109/SFFCS.1999.814609.

[48] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25(2):103–114, June 1996. ISSN 0163-5808. doi: 10.1145/235968.233324.

[49] David Marchette. A statistical method for profiling network traffic. In *Proceedings of the 1st conference on Workshop on Intrusion Detection and Network Monitoring - Volume 1*, ID'99, pages 13–13, Berkeley, CA, USA, 1999. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267880.1267893.

[50] Marcel R. Ackermann, Christiane Lammersen, Marcus Märtens, Christoph Raupach, Christian Sohler, and Kamil Swierkot. Streamkm++: A clustering algorithms for data streams. In *ALENEX*, pages 173–187, 2010. URL http://www.siam.org/proceedings/alenex/2010/alx10_016_ackermannm.pdf.

[51] A. Meyerson. Online facility location. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 426–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1390-5. URL http://dl.acm.org/citation.cfm?id=874063.875567.

[52] Charu C. Aggarwal. An intuitive framework for understanding changes in evolving data streams. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 261–, 2002. doi: 10.1109/ICDE.2002.994715.

[53] Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 575–586, New York, NY, USA, 2003. ACM. ISBN 1-58113-634-X. doi: 10.1145/872757.872826.

[54] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 81–92. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://dl.acm.org/citation.cfm?id=1315451.1315460.

[55] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 852–863. VLDB Endowment, 2004. ISBN 0-12-088469-0. URL http://dl.acm.org/citation.cfm?id=1316689.1316763.

[56] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, pages 61–72, New York, NY, USA, 1999. ACM. ISBN 1-58113-084-8. doi: 10.1145/304182.304188.

[57] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. Tracking clusters in evolving data streams over sliding windows. *Knowl. Inf. Syst.*, 15(2):181–214, May 2008. ISSN 0219-1377. doi: 10.1007/s10115-007-0070-x.

[58] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002. doi: 10.1137/S0097539701398363.

[59] Hai-Guang Li, Gong-Qing Wu, Xue-Gang Hu, Jing Zhang, Lian Li, and Xindong Wu. K-means clustering with bagging and mapreduce. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–8, 2011. doi: 10.1109/HICSS.2011.265.

[60] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. ISSN 0885-6125. doi: 10.1023/A:1018054314350.

[61] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855711.1855732.

[62] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8. doi: 10.1145/1851476.1851593.

[63] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1863103.1863113.

[64] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2): 285–296, September 2010. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1920841.1920881.

[65] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. The haloop approach to large-scale iterative data analysis. *The VLDB Journal*, 21 (2):169–190, April 2012. ISSN 1066-8888. URL http://dx.doi.org/10.1007/s00778-012-0269-7.

[66] Cairong Yan, Xin Yang, Ze Yu, Min Li, and Xiaolin Li. Incmr: Incremental data processing based on mapreduce. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 534–541, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4755-8. URL http://dx.doi.org/10.1109/CLOUD.2012.67.

[67] Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar, and Rafael Pasquin. Incoop: Mapreduce for incremental computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 7:1–7:14, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0976-9. URL http://doi.acm.org/10.1145/2038916.2038923.

[68] Pramod Bhatotia, Marcel Dischinger, Rodrigo Rodrigues, and Umut A Acar. Slider: Incremental sliding-window computations for large-scale data analysis. *MPI-SWS-2012-004*, September 2012. URL http://www.mpi-sws.org/tr/2012-004.pdf.