SCALABLE STREAMING PROFILE CLUSTERING FOR TELCO ANALYTICS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mehmet Ali Abbasoğlu

August, 2013

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Buğra Gedik (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof Dr. Hakan Ferhatosmanoğlu (Co-Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Fatih Emekçi

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

SCALABLE STREAMING PROFILE CLUSTERING FOR TELCO ANALYTICS

Mehmet Ali Abbasoğlu
M.S. in Computer Engineering
Supervisors:
Asst. Prof. Dr. Buğra Gedik
Assoc. Prof Dr. Hakan Ferhatosmanoğlu
August, 2013

Many telco analytics require maintaining call profiles based on recent customer call patterns. Such profiles are typically organized as aggregations computed at different time scales over the recent customer interactions. Clustering these profiles is needed to group customers with similar calling patterns and to build aggregate models for them. Example applications include optimizing tariffs, segmentation, and usage forecasting. In this thesis, we present an approach for clustering profiles that are incrementally maintained over a stream of updates. Due to the large number of customers, maintaining profile clusters have high processing and memory resource requirements. In order to tackle this problem, we apply distributed stream processing. However, in the presence of distributed state, it is a major challenge to partition the profiles over machines (nodes) such that memory and computation balance is maintained, while keeping the clustering accuracy high. Furthermore, to adapt to potentially changing customer calling patterns, the partitioning of profiles to machines should be continuously revised, yet one should minimize the migration of profiles so as not to disturb the online processing of updates. We provide a re-partitioning technique that achieves all these goals. We keep micro-cluster summaries at each node, collect these summaries at a centralized node, and use a greedy algorithm with novel affinity heuristics to revise the partitioning. We present a demo application that showcases our Storm and Hbase based implementation in the context of a customer segmentation application.

Keywords: Distributed clustering, aggregate profile clustering, telco.

ÖZET

TELKO ANALİZLERİ İÇİN ÖLÇEKLENEBİLİR AKAN PROFİL KÜMELEMESİ

Mehmet Ali Abbasoğlu
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri:
Asst. Prof. Dr. Buğra Gedik
Assoc. Prof Dr. Hakan Ferhatosmanoğlu
Ağustos, 2013

Birçok telekom analizi geçmiş arama desenlerine dayalı arama profillerine gereksinim duyar. Bu arama profilleri değişik zamanlardaki müşteri etkileşimlerinin yığılması ile oluşmaktadır. Telekom şirketlerinin pazarlama ve satış gibi operasyonlarını iyileştirecek analizler müşteri arama profilleri üzerinden yapılmaktadır. Örnek uygulamalar olarak tarife iyileştirme, müşteri bölümleme ve kullanım öngörüsü gösterilebilir. Bu tezde güncelleme katarları ile oluşan müşteri profillerinin kümelenmesi için bir yöntem sunulmaktadır. Profil kümeleri yüksek sayıda müşteri olması nedeniyle yüksek bellek ve işlemci gücü gerektir. Bu gereksinimleri karşılayabilmek için çözümümüzde dağıtık veri katarı işleme yöntemleri kullandık. Ancak profillerin makinalara dağılımını kümeleme kalitesini yüksek tutarken, her makinanın eşit miktarda profil saklamasını ve işlemesini sağlamak, dağıtık sistemlerde önemli bir zorluk. Buna ek olarak, müşterilerin arama deseni değiştirmesi ihtimali nedeniyle, profillerin makinalara dağılımı düzenli olarak güncellenmeli. Bu güncelleme işlemi çevirimiçi işleme sürecini aksatmamak için asgari miktarda yer değişimi gerçekleştirmeli. Bu tezde tüm bu ihtiyaçları karşılayan bir tekrar dağıtım tekniği sunulmuştur. Her makina kendi içerisinden mikro-kümeler oluşturmakta ve onların özetlerini merkezi makinaya göndermektedir. Merkezi makina mikro-küme özetlerini üzerinde yeni aitlik buluşsal yöntemleri içeren açgözlü bir işlemsel süreçten geçirerek profil dağıtımını güncellemektedir. Tezde ayrıca sunulan çözümün Storm ve Hbase tabanlı gerçekleştirmesini gösteren, telekom şirketleri için müşteri bölümleme amacıyla kullanılabilecek bir demo uygulaması sunulmuştur.

Anahtar sözcükler: Dağıtık kümeleme, yığın profil kümeleme, telko.

Acknowledgement

First of all, I would like to thank to my supervisor, Asst. Prof. Dr. Buğra Gedik for his guidance, support and always being available to me when I needed help during my graduate study. We have worked together just for 1 year, and I have already learned a lot from him.

I also thank to my co-advisor Assoc. Prof. Dr. Hakan Ferhatosmanoğlu for his excellent guidances, valuable suggestions and patience throughout this study.

I am grateful to my jury members, Prof. Dr. Özgür Ulusoy and Asst. Prof. Dr. Fatih Emekçi for reading and reviewing this thesis.

I would like to acknowledge KORVUS Ltd. for their financial and hardware support. I would also like to acknowledge TÜBİTAK for their financial support.

I thank to my family for supporting me with all my decisions and for their endless love, especially my mother for her support and faith during my thesis work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Telecommunications (telco) is a data-intensive domain where live feeds that carry customer interaction data stream into the data centers of service providers. Analytics performed on such data can help improve operations (such as forecasting for resource provisioning), marketing (such as customer segmentation for campaign management), and sales (such as regression for churn prediction).

Keeping a recent history of customer calling behavior, creating customer calling profiles from it, and maintaining such profiles as clusters are key enabling techniques for many of the telco analytics. For example customer segmentation by clustering is a fundamental operation for churn analysis [1] and customer profiling [2]. Also, modeling and forecasting the call patterns of users is more effective when applied on customers with similar calling profiles rather than on individual customers [3].

As these examples motivate, many telco analytics operate on clusters of customer calling profiles. Given the continuous and live nature of these analytics and the potentially dynamic behavior of customers, there is a clear need to maintain the customer call profiles in a clustered manner. However, processing customer interactions for performing analytics on a large set of customers requires high processing resources.

We present our system for scalable profile management and clustering in a streaming setting. The demo highlights the effectiveness of our solution in the context of a telco customer segmentation application. The main idea is to cluster large number of profiles, where each profile is an incrementally updated aggregate over streaming updates — aka an *aggregate profile*. While the problem has general applicability, our work is strongly motivated by the telco domain. For instance, our updates are *call detail records* (CDRs), which contain meta-data about calls made between customers. Each CDR has a caller associated with it and contributes to that caller's aggregate profile (e.g., the daily number of international calls can be a feature in the aggregate profile). The goal is to maintain profile clusters, so that callers with similar behaviors are grouped together, and use these profile clusters for analytics such as forecasting and customer segmentation.

Given the large number of profiles, maintaining these clusters on a single machine may not be feasible, especially if the profiles are large in terms of size or the cost to process each profile update is high (e.g., updating a forecasting model for the profile or for the cluster). Furthermore, in most real-world scenarios the profile updates are not used for the sole purpose of cluster maintenance and clustered analytics, but for miscellaneous processing, such as enrichment, model scoring, visualization, etc. Thus, the need for parallel and distributed processing is paramount.

To address this challenge, *partitioned stateful parallelism* is employed. Incoming stream is partitioned over a set of processing nodes based on a partition by attribute (such as the caller id in a CDR) and each node process its portion of the sub-stream, maintaining a subset of the clusters and the associated state needed to maintain the aggregate profiles. Here, we want to make sure that each node gets assigned similar amount of processing load, since the slowest node will form a bottleneck for the system.

There are a number of challenges in achieving this. First, in order to distribute the incoming updates over the set of nodes, we need a way of partitioning them such that each update is routed to the node that contains profiles similar to its own. Note that the similarity here applies to the aggregate profiles, and *not* to

the update itself. Initially, there is no information on the profile clusters, and as a result the partitioning will be hash based. Thus, after some time all nodes will form similar clusters. This is a problem, since similar profiles cannot be co-located on the same machine and as the number of nodes increase the fidelity of the clusters will decrease. As we know more about the nature of the profiles and frequencies of the partitioning attribute values, we need to incrementally update our partitioning scheme and migrate profiles as needed, to increase the clustering quality.

Second, this re-distribution has to make sure that each node gets a similar sized flow of updates (good *processing balance*). Furthermore, the changes in the partitioning function should be incremental, so as the migration of profiles do not cause a pause in processing (low *migration overhead*).

Our aim is to distribute profiles to nodes according to their similarities to each other, while at the same time considering the balance between the nodes with respect to processing and memory cost. In our proposed solution, while stream of CDRs update the profiles in distributed nodes, each processing node in the system calculates its micro-cluster summaries and informs the master node for a new distribution periodically. The master node builds the new distribution with the latest version of the micro-cluster summaries. New distribution is built by calculating an overall affinity value for every micro-cluster summary towards each processing node. Each micro-cluster is directed to the node that has the highest ranked overall affinity. Overall affinity consists of clustering disaffinity which is calculated with average distance of k nearest neighbors, balance disaffinity which compares the fullness of nodes and migration affinity which aims to minimize the amount of state migration between nodes.

We present a demo application that uses the proposed solutions for telco customer segmentation. The system relies on distributed stream processing middleware Storm [4] for processing updates and maintaining the profile clusters in memory, and HBase [5] for partial fault-tolerance and for facilitating the migration. The application uses a CDR stream in which every CDR is labeled with call tariff data of the customer. Tariffs are rated considering their distribution over

customer profiles in order to detect poorly defined tariffs, and clusters are rated by considering the heterogeneity of the tariffs they contain in order to detect new possible call tariffs. Tariffs which are scattered over multiple clusters in which they are not well represented are identified as not well defined tariffs, and clusters with high entropy are identified as potential targets for defining new tariffs.

The remainder of this thesis is organized as follows. Chapter 2 formalizes scalable and distributed profile clustering problem. Chapter 3 discusses related work on distributed clustering methodologies. Chapter 4 explains the proposed greedy algorithm that includes three heuristics: cluster affinity, processing affinity and migration affinity. Chapter 5 describes our experiment environment and evaluation results. Chapter 6 presents demo application which uses aggregate profile clustering to perform cluster segmentation for call tariff optimization. Finally, Chapter 7 concludes this thesis.

Chapter 2

Problem Definition

In this section we formalize our problem. We define our requirements as clustering quality, balance quality, and migration quality. Using clustering quality we formalize how successful our clustering is compared to a centralized clustering approach. Using balance quality we compare the processing loads of nodes and measure how balanced they are. Migration quality formulation aims to capture how much migration is done in the process with respect to the total state size. Finally, we present an overall quality formulation which is a combined measure of quality that relies on the clustering, balance, and migration qualities.

Let S denote a stream of updates, where $u \in S$ denotes an update. We use $\iota(u) \in D$ to denote the value of the profile id for the update, where D is the domain of the profile ids. Let $P(d)$ denote the aggregate profile for profile id $d \in D$. We assume that $P(d)$ is a multi-dimensional vector. We use $f(d)$ to denote the frequency of updates with profile id $d \in D$.

We define a partitioning function $p : D \to [0..N)$ that maps each profile id to a node, where we have N nodes. When an update u is received, it is forwarded to the node at index $p(d)$, where $\iota(u) = d$. There, it contributes to the aggregate profile information $P(d)$, via the transformation:

$$\langle P(d), S(d) \rangle \leftarrow \gamma(P(d), S(d), u).$$

Here, γ is an aggregation function and $S(d)$ is the state maintained to compute it continuously for profile id d.

At time step s, the partitioning function will be updated from p_{s-1} to p_s, with the goal of keeping the clustering quality high, the processing and/or memory loads balanced, and the migration cost low. We now define each of these metrics.

2.1 Clustering Quality

Let $\mathcal{C}_i$ be the set of clusters on node i after applying a local clustering algorithm $\mathcal{A}$, that is $\mathcal{C}_i = \mathcal{A}(\{P(d) : p(d) = i\})$. Let $\mathcal{C}$ be the set of all clusters from all nodes, that is $\mathcal{C} = \bigcup_{i \in [0..N)} \mathcal{C}_i$. Further assume that $\mathcal{C}^*$ denotes the clustering that would be formed if the same clustering algorithm is applied on all profiles, that is $\mathcal{C}^* = \mathcal{A}(\{P(d) : d \in \mathcal{D}\})$.

We define the *clustering quality* as the *Normalized Mutual Information* (NMI) between the ideal clustering denoted by $\mathcal{C}^*$ and the distributed clustering we computed denoted by $\mathcal{C}$:

$$Q^c = NMI(\mathcal{C}^*, \mathcal{C}) \tag{2.1}$$

NMI is defined as:

$$NMI(X, Y) = \frac{H(X) + H(Y) - H(X, Y)}{(H(X) + H(Y))/2}, \tag{2.2}$$

where $H(X)$ is the entropy of the clustering X and $H(X, Y)$ is the joint entropy of X and Y.

With this definition of clustering quality, we aim to compare our distributed clustering results with the clustering that is formed when all profiles are collected at a single node and the same clustering method is applied. NMI incurs a low penalty for clusters that are split into multiple sub-clusters. This is crucial for

our evaluation, because as a result of distributed processing our clusters might split into several pieces.

Here, it is important to use a clustering algorithm $\mathcal{A}$ whose parameter settings are not impacted by the number of nodes, N. For instance, for small number of dimensions a density-based clustering algorithm (such as DB-scan [6]) will work well. For k-means based algorithms, the distribution of k over the N nodes will be a problem. To aleviate this, k-means algorithms that use automatic determination of the k value can be used, such as those that rely on the BIC metric to determine k [7].

For this work, we have used EM clustering [8] – a distribution-based clustering algorithm. The WEKA [9] implementation of the EM clustering can set the number of clusters automatically, which makes it easy to use in our setup, as its avoids having to adjust the number of clusters based on the node count.

2.2 Balance Quality

Let $R_i = \sum_{p(d)=i} f(d) \cdot \beta(|S(d)|)$ denote the processing cost required to handle the profiles assighed to the ith node. Here, β is a function that defines the relationship between the amount of state maintained and the required processing to update the aggregate profile. We define the *processing balance quality* as $Q^{pb} = 1 - \text{CoV}(\{R_i\})$, where CoV is the coefficient of variation (ratio of std. deviation to mean). When the std. deviation in the balance is 0, then the balance quality is 1. When the deviation reaches a single node's share of the load (i.e., the mean), then the quality reaches 0. Let $M_i = \sum_{p(d)=i} |S(d)|$ denote the size of state stored on the ith note to maintain the profiles assigned to it. We define the *memory balance quality* as $Q^{mb} = 1 - \text{CoV}(\{M_i\})$.

Depending on the nature of the state maintained ($S(d)$ for profile d), the memory may or may not be a concern. For instance, if the state is constant size and small, then it may fit on a single machine. In this case we can take the *balance quality* as $Q^b = Q^{pb}$, ignoring the memory balance. On the other

hand, when the state is linear in the frequency ($|S(d)| \propto f(d)$), such as for an aggregation γ defined over time-based sliding windows, then the memory balance may factor into the balance quality and thus we take $Q^b = (Q^{pb} + Q^{mb})/2$. Other combinations are possible.

2.3 Migration Quality

As the system knows more about the nature of the profiles and frequencies of the partitioning attribute values, partitioning scheme needs to be incrementally updated and profiles should be migrated as needed. In this operation, migration overhead should be low, therefore we compare migrated amount of state with the total state to define the migration quality.

We formalize the migration quality as follows:

$$Q^m = 1 - \frac{\sum_{d \in D} |S(d)| \cdot \mathbf{1}(p'(d) \neq p(d))}{\sum_{d \in D} |S(d)|} \tag{2.3}$$

Here, p' is the previous partitioning function. For no migration, the migration quality is 1. When the entire state needs to move, then the migration quality is 0.

2.4 Overall Quality

We define overall quality with the help of previously defined concerns: clustering quality, balance quality, and migration quality. There is a trade-off between clustering quality and balance quality. When the importance of balance quality is set high, the clustering quality may suffer, as keeping the balance quality high may necessitate splitting clusters into several sub-clusters. Therefore, there is a need to strike a good balance between clustering quality and balance quality.

We denote the overall quality as Q, and define it as:

$$Q = (\alpha \cdot Q^c + (1 - \alpha) \cdot Q^b) \cdot Q^m \tag{2.4}$$

Here, $\alpha \in [0, 1]$ adjusts the relative importance of clustering quality versus load balance.

2.5 An Example

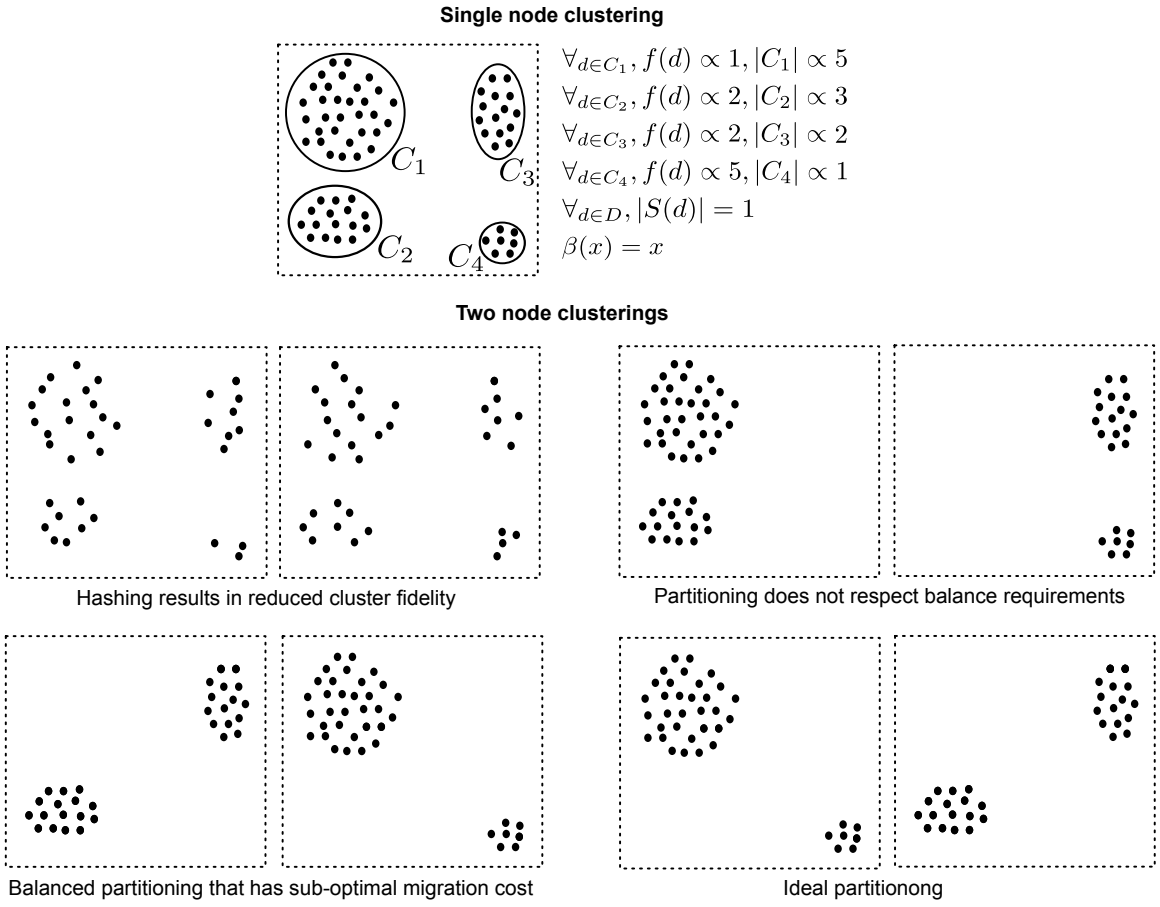

Figure 2.1: An illustration of alternative partitionings.

Figure 2.1 illustrates a toy scenario with 4 clusters of profiles, namely C_1, C_2, C_3, and C_4. Among these, C_1 is the largerst in terms of the number of profiles ($|C_1| \propto 5$), but the frequency of updates for profiles in this cluster is the lowest

($f(d) \propto 1, \forall_{d \in C_1}$[1]). C_4, on the other hand, has the lowest number of profiles ($|C_4| \propto 1$), but the highest frequency of updates ($f(d) \propto 5, \forall_{d \in C_4}$). C_2 and C_3 have values in between. In this example, we have constant state ($|S(d)|, \forall d \in D$) for aggregation and processing time linear in state size ($\beta(x) = x$). Given processing load for a cluster is $p(C_i) = \sum_{d \in C_i} f(d) \cdot \beta(|S(d)|)$ and memory size is $m(C_i) = \sum_{d \in C_i} |S(d)|$, we have the following characteristics for the clusters: $\langle p(C_1), p(C_2), p(C_3), p(C_4) \rangle = \langle 5, 6, 4, 5 \rangle$ and $\langle m(C_1), m(C_2), m(C_3), m(C_4) \rangle = \langle 5, 3, 2, 1 \rangle$.

Figure 2.1 shows four different partitionings for $N = 2$. The first alternative represents hashing. The problem with this alternative is that, the fidelity of the clusters are reduced. As N increases, this alternative will further degrade with respect to the cluster quality. The second alternative puts together clusters that are similar on the same node (akin to clustering the clusters). However, this is not an appropriate goal, as it does not balance the load. In this alternative, the processing load for the first node is ≈ 11, whereas that of second node is ≈ 9. The third alternative balances the load perfectly (10 and 10), has the same clustering result as the single node case, but compared to the last alternative, it has higher migration cost (6 versus 5). As a result, the best alternative is the rightmost one.

[1] In this example, the frequency is taken as equal for all updates in the same cluster.

Chapter 3

Related Work

Clustering can be defined as the task of grouping a set of objects in such a way that objects in the same group are more similar in some sense to each other than to those in other groups. Clustering is the main task of exploratory data mining and is commonly used in statistical data analysis.

We classify existing work on clustering using the following dimensions:

A) Nature of processing:

- *Centralized*: In centralized processing, there is a single processing node which performs the clustering.
- *Distributed*: In distributed processing, multiple nodes are used to perform the clustering. In general, one or more of these nodes can be used as a master to establish cross-node arbitration.
- *Decentralized*: In decentralized processing, all nodes have the same task. There is no master, all nodes are equivalent.

B) Dynamicity of the dataset:

- *Static*: In static data sets, the objects to be clustered do not change. As a result, the processing is performed offline.

- *Dynamic*: In dynamic data sets, the objects to be clustered are updated frequently. As a result the clusters need to be maintained in an online manner, usually via incremental schemes.

C) Home location of clustered objects:

- *Centralized*: In centralized setting, all clustered objects are stored at a single node.
- *Distributed*: In distributed setting, clustered objects are partitioned over nodes.

D) Mobility of clustered objects:

- *Fixed*: In fixed setting, each clustered object stays at its home location.
- *Migratable*: In migratable setting, clustered objects can be migrated across nodes.

E) Source of updates to the clustered objects:

- *Local*: In local setting, the updates to the clustered objects are generated at their home location.
- *Remote*: In remote setting, the updates to the clustered objects come from a remote place and have to be routed to the home location.

F) Nature of the updates:

- *Complete*: In complete updates, each update replaces the clustered object.
- *Partial*: In partial updates, each update contributes to the value of the clustered object.

We classify existing clustering methodologies into five main categories. In the following sections we explain these methodologies and their categories according to aforementioned dimensions. A summary of the comparison is also given in Table 3.1.

Clustering	A - Processing	B - Dynamicity	C - Home Location	D - Mobility	E - Update Source	F - Update Type
Our aim	distributed	dynamic	distributed	flexible	remote	partial
Traditional clustering	centralized	static	centralized	fixed	local	complete
Distributed/parallel traditional clustering	distributed	static	centralized	fixed	local	complete
Distributed clustering for remote monitoring	distributed	dynamic	distributed	fixed	local	complete
Data clustering in sensor/p2p networks	decentralized	dynamic	distributed	fixed	local	complete
Incremental data stream clustering	centralized	dynamic	centralized	fixed	remote local	complete

Table 3.1: Comparison of different approaches to clustering.

3.1 Traditional Clustering

In traditional clustering, a single node is responsible for performing clustering. All data is gathered to a single node and the clustering process is performed over the data. Objects to be clustered are not updated, they are static. Objects reside on the same node that is responsible for performing the clustering.

K-means [10] is one of the traditional clustering methodologies. K-means clustering is used to cluster objects into groups of related objects without any prior knowledge of those relationships. The algorithm clusters objects into k groups, where k is provided as an input parameter. Initially it picks k random objects as the cluster centers. It then assigns each object to a cluster based upon the object's proximity to the cluster center. After the assignments are complete, it recomputes the cluster centers as the averages of all points assigned to the same clusters. The process repeats until convergence. In our proposed solution, we use k-means to create micro-clusters by setting the k parameter relatively high.

Density-based spatial clustering of applications with noise (DBSCAN) [6] is another example of traditional clustering. DBSCAN is a *density-based clustering* algorithm, where the number of clusters is not pre-determined and can change depending on the nature of the data. DBSCAN is designed to discover clusters of arbitrary shape. It requires two parameters: ϵ and the minimum number of points required to form a cluster. The algorithm starts with an arbitrary starting point that has not been visited and computes the density-reachable ϵ neighborhood. If the neighborhood contains sufficiently many points, a cluster is started.

Clustering using Expectation-Maximization (EM) [8] is another traditional clustering method. EM clustering is an example of *probability distribution based clustering*. EM algorithm is an iterative method for finding maximum likelihood values where the model depends on unobserved latent variables. The EM clustering relies on Gaussian mixture models, where the object labels are the unobserved variables. It follows an iterative approach, which tries to find the parameters of the Gaussian mixture that provides the maximum likelihood for the data at hand. In our proposed solution, we use EM clustering to create final clusters from micro-clusters.

There are various other clustering methods, such as link-based clustering used to create hierarchical clusters. We do not cover them here, as our focus is on the distribution and partitioning rather than the core algorithms.

3.2 Distributed and Parallel Implementations of Traditional Clustering

Distributed and parallel versions of clustering algorithms have been developed to provide speedup, scale-up, and size-up. In distributed and parallel implementations, multiple nodes are used to perform clustering. Clustered objects are stored on the master node, which is also responsible for establishing cross-node arbitration and distribution of the data.

Parallel clustering algorithm PDBSCAN [11] is based on DBSCAN for knowledge discovery in large datasets. PDBSCAN uses 'shared-nothing' architecture, therefore it can be scaled up to hundreds of computers. In a similar fashion, Parallel k-means [12] offers a parallel clustering algorithm on shared-nothing parallel machines. Mahout [13] provides distributed implementation of several traditional clustering algorithms based on the Map/Reduce [14] framework.

3.3 Distributed Clustering for Remote Monitoring

In remote monitored distributed clustering, clustering task is distributed over several nodes and the objects in the dataset are updated frequently. Therefore clusters need to be updated in an online matter, when a change appears in dataset. In contrast with distributed and parallel versions of traditional clustering methods, objects to be clustered are also distributed to several nodes. Furthermore, the updates are local to the home location of the items.

Research by Januzaj et al. [15] propose clustering data locally and extract suitable representatives out of these clusters. These representatives are sent to global master node, where complete clustering based on local representatives are built.

As another example to remote monitored distributed clustering, Cormode et al., 2007 [16] declare that collecting voluminous data over distributed network to a central location is undesirable. They suggest to perform clustering in-place where data is collected, send result information to a central location, and form high accuracy clusters while minimizing the communication and computational cost.

3.4 Data Clustering in Sensor/P2P Networks

Clustering in sensor networks and peer-to-peer environments is done without a central node. In such systems, each node holds some data value, e.g., a local sensor reading, and is responsible from participation in a distributed clustering algorithm. We classify these clustering approaches as having decentralized nature of processing.

Eyal et al. [17] propose a solution where numerous interconnected sensor nodes partition their data into multiple clusters, and describe each cluster concisely. They observe that distance criterion is not suffient to provide good clustering results, and for this reason they develop a generic algorithm that models the values as a Gaussian mixture model. Gedik et al. [18] propose ASAP, an adaptive sampling approach to data collection in sensor networks. ASAP uses sensing-driven clustering to group nodes into clusters. This clustering technique tries to form clusters that contain sensor nodes that are not only spatially close but also their sensor readings are close.

3.5 Incremental Data Streaming Clustering

Incremental data streaming is used in cases where the data is evolving dynamically. A single node is responsible for maintaining clusters in an online manner with incremental schemes. The data is scanned only once, thus it is streaming.

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [19] is an unsupervised data mining algorithm used to perform hierarchical clustering over incrementally and dynamically incoming data. Guha et al. [20] also study k-Median problem in the streaming context and provides a new streaming clustering algorithm which is based on a facility location algorithm.

Chapter 4

Proposed Solution

In this chapter, we give an overview of our solution, describe its core algorithm in detail, and provide the system architecture of its implementation.

4.1 Solution Overview

Our aim is to cluster large number of profiles which are formed with incremental updates. Maintaining profile clusters on a single machine is not feasible, especially when profiles have high numbers of updates. Thus, we propose parallel and distributed clustering for aggregate profiles.

In our solution, profiles are distributed to nodes according to their similarities to each other, while at the same time considering the balance between the nodes with respect to processing and memory costs. First, the incoming stream is partitioned over processing nodes by a simple hash function using a key attribute from the update (such as the caller id in a CDR) and each node process its portion of the substream. Aggregate profiles are incrementally built on the processing nodes. After some Δt time, each node applies k-means clustering with relatively high k parameter to create micro-clusters. Micro-clusters are then sent to the master node to form new clustering results and update the partitioning function.

After the master node gets all micro-clusters from processing nodes, the system pauses until the new partition mapping is created. For each micro-cluster, the master node ranks the processing nodes and assigns the micro-cluster to the most appropriate node. Ranking is performed using three considerations: (1) keep the clustering quality high by placing micro-clusters that are close to each other on the same nodes, (2) keep the total processing/memory balanced, and (3) minimize the amount of state migration that will result from updating the partitioning function. After every micro-cluster is assigned to a processing node, a new partition mapping is created from profile-ids to processing node-ids. All processing nodes are informed about the new partition mapping, and state migrations are performed for the profiles whose node mappings have changed. After the migration operation is complete, the system resumes processing again.

4.2 Updating the Partitioning Function

Recall that the main idea is to update the partitioning function periodically, by collecting summary information at a master node. The goal is to balance the load and keep the clustering quality high, while incurring low migration cost.

At step $s = 0$, we set the partitioning function to the consistent hash function $\mathcal{H}_N$, that is $p_0(d) = \mathcal{H}_N(d)$. For the purpose of updating the partitioning function, each node creates micro-clusters [21] over the profiles they maintain. The summaries of these micro-clusters are then sent to a master node, which computes a new partitioning function p_s.

A micro-cluster, denoted as $M \subset D$, keeps a set of profile ids. It is summarized as a 5-tuple: $\hat{M}=\langle o, r, p, m, l\rangle$. Here, o denotes the centroid of the micro-cluster, that is $\hat{M}.o = \sum_{d\in M} P(d)/|M|$. The radius of the micro-cluster is denoted by r. We have $\hat{M}.r = \max_{d\in M} \|P(d) - \hat{M}.o\|$. The total processing cost for the profiles in the micro-cluster is denoted as p. We have $\hat{M}.p = \sum_{d\in M} f(d) \cdot \beta(|S(d)|)$. The total memory cost for the state associated with the profilies of the micro-cluster is denoted by m. We have $\hat{M}.m = \sum_{d\in D} |S(d)|$. Finally, l denotes the current

location of the micro-cluster. We have $\hat{M}.l = p_{s-1}(d), d \in M$.

The master node, upon receiving all the micro-cluster summaries, creates a new partitioning function. For this purpose we use the greedy procedure described in Algorithm 1. The algorithm iterates over the micro-clusters and for each micro-cluster it makes a node assignment. We consider micro-clusters in the order of their state sizes during the assignment.

For making assignments, the algorithm makes use of a heuristic metric. It picks the assignment that maximizes this metric. Let $\mathcal{M} = \{M_i\}$ be the list of all micro-clusters, and assume that $i-1$ assignments are made and we are to make an assignment for the ith micro-cluster, M_i. In order to do this, we first compute the *affinity* of this micro-cluster to each node. Let $A(M_i, j)$ denote the affinity of M_i to the jth node. We set $\forall d \in M_i$, $p_s(d) = \text{argmax}_{j \in [0..N)} A(M_i, j)$. That is, the node for which the micro-cluster has the highest affinity becomes the new mapping for all the profiles of the micro-cluster.

Algorithm 1: updatePartitioning$(\mathcal{M}, N, O)$

```
Param : M, micro-clusters
Param : N, number of nodes
Param : O, ordering policy
p ← {}                                   ▷ The partitioning function to be constructed
M' ← Sort(M, O)                          ▷ Order micro-clusters based on policy
for M ∈ M' do                            ▷ For each micro-cluster, in order
    i ← −1; a ← 0                        ▷ Best assignment and affinity
    for j ∈ [0..N) do                    ▷ For each node
        ▷ Compute the affinity of M to the current node j
        v ← A^m(M, j) · (1 − ((1 − α) · A^c(M, j) + α · A^c(M, j)))
        if v > a then ⟨i, a⟩ ← ⟨j, v⟩;   ▷ Update best, if necessary
    p[d] = i, ∀d ∈ M                     ▷ Create mappings for the profiles in M
return p                                 ▷ Return the fully constructed mapping
```

Affinity has three aspects to it: the *clustering disaffinity* denoted by A^c, the *balance disaffinity* denoted by A^b, and the *migration affinity* denoted by A^m. Let $\mathcal{M}_l$ denote the set of micro-clusters assigned to the lth node so far, i.e., $\mathcal{M}_l = \{M \mid p_s(d) = l \wedge d \in M \in \mathcal{M}\}$.

4.2.1 Clustering Disaffinity

As clustering disaffinity, we aim to calculate how far a micro-cluster is to a processing node in the multi-dimensional space. k micro-cluster members of the processing node that are closest to the micro-cluster at hand are found, and the sum of their distances is calculated. We normalize this value with the sum of all clustering disaffinities towards all nodes.

The clustering disaffinity is formalized as follows:

$$A^c(M_i, j) = \frac{\sum_{x \in \text{min-}k(L_{i,j})} x}{\sum_{l \in [0..N)} \sum_{x \in \text{min-}k(L_{i,l})} x}, \tag{4.1}$$

where $L_{i,j} = \{\|\hat{M}_i.o - \hat{M}.o\| \mid M \in \mathcal{M}_j\}$ and argmin-k is a function that takes the smallest k elements from a list. $L_{i,j}$ represents the list of distances from the micro-cluster centroid $\hat{M}_i.o$ to the centroids of micro-clusters that are assigned to the jth node so far. $k \geq 1$ is a parameter of our algorithm. The clustering disaffinities sum up to 1, that is $\sum_{j \in [0..N)} A^c(M_i, j) = 1$.

4.2.2 Balance Disaffinity

One of our aims is to assign a similar sized flow of updates to each processing node. Therefore we use balance disaffinity, which calculates how much processing capacity is used in a processing node after the micro-cluster at hand is assigned to it. Balance disaffinities are normalized with the total used processing capacity across all processing nodes.

The balance disaffinity, for processing, is formalized as:

$$A^b(M_i, j) \approx \frac{C_{i,j}}{\sum_{l \in [0..N)} C_{i,l}}, \tag{4.2}$$

where $C_{i,j} = \hat{M}_i.p + \sum_{M \in \mathcal{M}_j} \hat{M}.p$. Here, $C_{i,j}$ represents the amount of processing capacity used up on the jth node once the micro-cluster M_i is placed on it, also considering all the previous micro-clusters that were placed on that node. We normalize balance disaffinities so that $\sum_{j \in [0..N)} A^b(M_i, j) = 1$. The balance disaffinity for memory is defined similarly, by replacing p with m.

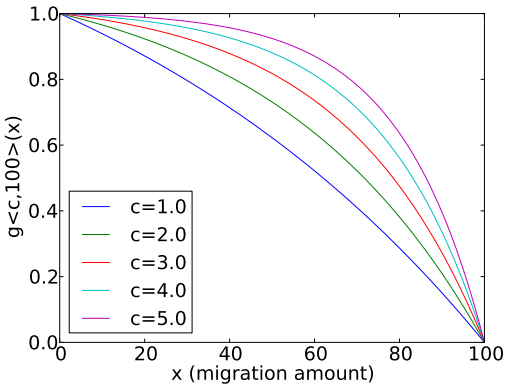

Figure 4.1: g functions for different c values.

4.2.3 Migration Affinity

To calculate the migration affinity, we compute how much migration is performed so far and compare it to the maximum migration cost that is allowed. Migration affinity decreases as the total amount of migration so far increases. The rate at which migration affinity decreases increases with increasing migration cost. This means that migration affinity has lower impact initially, when the total migration is quite small compared to the maximum allowed.

The migration affinity is defined as follows:

$$A^m(M_i, j) = g\langle c, G\rangle(R + \hat{M}_i.m \cdot \mathbf{1}(\hat{M}_i.l \neq j)), \tag{4.3}$$

where $R = \sum_{i\in[0..N)} \sum_{M\in\mathcal{M}_i} (\hat{M}.m \cdot \mathbf{1}(\hat{M}.l \neq i))$ represents the total migration cost so far. $G = \frac{2}{N} \cdot \sum_{M\in\mathcal{M}} \hat{M}.m$ represents the maximum allowable migration cost (taken as the twice the amount of state per node). $g\langle c, G\rangle(x)$ is a function of the form $y = a - b \cdot e^{c\cdot x/G}$ that satisfies $g\langle c, G\rangle(0) = 1$ and $g\langle c, G\rangle(G) = 0$. c is a parameter that adjusts the skew of the function. For instance, for $G = 100$, we get the graph shown in Figure 4.1. The motivation for having the g function is to penalize migrations less when initially there is a high budget for migration.

4.2.4 Overall Affinity

Given these definitions, we define the overall affinity as follows:

$$A(M_i, j) = A^m(M_i, j) \cdot (1 - ((1 - \alpha) \cdot A^c(M_i, j) + \alpha \cdot A^b(M_i, j))) \tag{4.4}$$

Here, $((1 - \alpha) \cdot A^c(M_i, j) + \alpha \cdot A^b(M_i, j))$ defines the combined clustering and balance disaffinity, where $\alpha \in [0..1]$ adjusts there importance of one compared to the other. As the value of α increases, the clustering becomes less decisive compared to the balance. Subtracting the combined clustering and balance disaffinity from 1 give the combined affinity, which we multiply with the migration affinity to get the overall affinity value.

4.2.5 Handling Edge Cases

There are a few edge cases to handle with the partitioning algorithm we have described so far. The first one is about the proper computation of the clustering disaffinity when a node has no micro-clusters assigned so far. The second is about the over-sensitivity of the balance disaffinity when there are very few assignments performed. We now look at how these issues are resolved.

Initial Condition Handling for Clustering Disaffinity

In order to be able to compute a clustering disaffinity for a node that has no assignments so far, we come up with initial cluster centers for each node to be able to compute a clustering dissaffinity. In particular, we take all micro-clusters and use k-means clustering to create N clusters out of them. We take the centroids of the resulting clusters and assign each one to one of the nodes as that node's initial cluster center. When the clustering disaffinity is to be computed for a micro-cluster that has no assignmets, this initial cluster center is used to compute the distance.

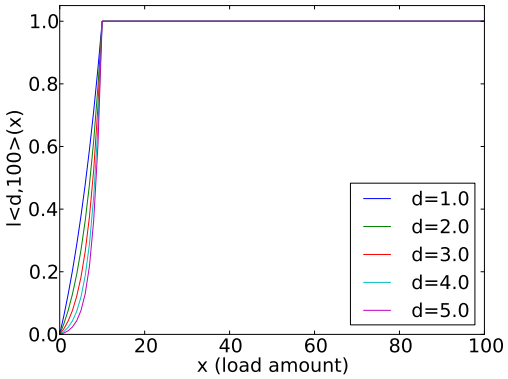

Figure 4.2: l functions for different d values.

Start-up Phase Handling for Balance Disaffinity

During the start-up phase of the algorithm, the balance disaffinity may prevent micro-clusters that are close to each other to be assigned to the same node, as that might hurt load balance. However, initial imbalances are not that important, as there would be plenty of opportunities for correcting them later in the assignment process. To capture this, we scale the importance of the load disaffinity (originally α) by a *scaler function*. We denote the scaler function as l and define it as follows:

$$l\langle d, L\rangle(x) = \begin{cases} a - b \cdot e^{d \cdot x/L} & x < L/10 \\ 1 & \text{otherwise} \end{cases} \tag{4.5}$$

The function takes as a parameter, the total amount of load assigned to the nodes so far. L is the maximum amount of load to be assigned and d is a parameter that adjusts the skew of the function. After 10% of the load is assigned, the scaler function defaults to 1. Furthermore, the scaler function l satisfies $l\langle d, L\rangle(0) = 0$ and $l\langle d, L\rangle(L/10) = 1$. For instance, for $L = 100$, we get the graph shown in Figure 4.2. The motivation for having the l function is to gradually increase the penalty due to the load imbalance.

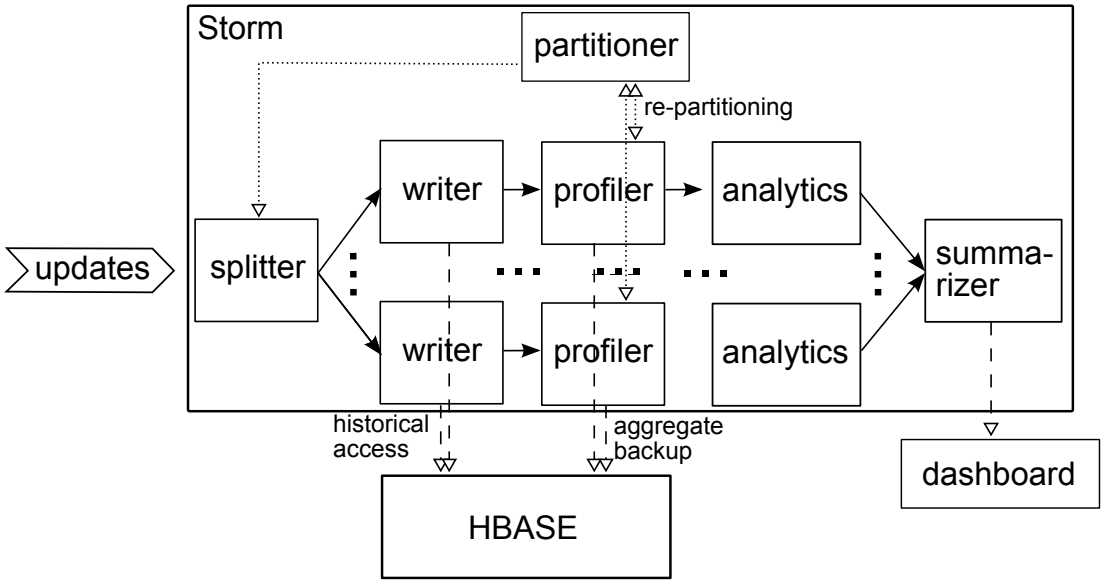

Figure 4.3: The architecture of the aggregate profile clustering system running on the telco analytics platform.

4.3 Implementing the Partitioned Clustering

We implemented our profile clustering technique in the context of a telco analytics platform. Figure 4.3 depicts the system architecture.

The updates (CDRs in the telco domain) stream into the system and are processed by a topology that runs on the Storm distributed stream processing system. The updates are tuplized and partitioned using the splitter operator. The splitted flows first go through the writer operator, which persists the updates to the HBase distributed key value store for historical access. This parallel write feature is not strictly needed for our aggregate profiling technique, but is part of the analytics platform.

The updates are then sent to the profiler operators, which are responsible for updating the in-memory profiles and performing clustering. The profiler interacts with the partitioner operator, which in turn interacts with the splitter, for implementing the re-partitioning. In particular, when re-partitioning is initiated the partitioner asks the splitter to pause the flow. After all in-flight tuples are processed, the micro-clusters are shipped from the profilers to the partitioner. The

partitioner executes the re-partitioning algorithm and computes the new partitioning. Using this partitioning, it computes migration schedules and sends these to the profilers. To minimize the coupling between profilers, the actual migration of state is performed through HBase. Each profiler writes to HBase the state that it no longer has to keep. After a synchronization step, it also borrows the state that it needs to maintain from now on. Once the state migration is completed, the partitioner sends the new partitioning to the splitter operator, which installs it and resumes the flow.

The profilers also use the HBase store to backup their state periodically, to support fault-tolerance. While the profile maintenance is not sensitive to short term tuple loss, this backup is needed to avoid losing long-term aggregations that are computed over large time scales.

Analytics operators use the clusters formed by the profiler operators to perform mining tasks such as usage forecasting and customer segmentation. The summarizer operator is a bridge between the analytics operators and the dashboard that visualizes the results.

Chapter 5

Experimental Setup and Results

We have performed a number of experimental studies with the aim of evaluating the success of the proposed architecture. In this chapter, we first describe the experimental setup and then discuss our experimental studies and their results in detail.

5.1 Experimental Setup

In this section machines that are used as processing nodes are explained in detail, the source of incoming CDR stream is explained, and the default parameters of the system are presented.

5.1.1 Machines

Evaluating the scalability of our solution requires multiple machines. We use kernel-based virtual machines in amd64 architecture with 4 cores and 4GB memory. The machines have CentOS 6.4 operating system, 1.7.0 version of Java, 1.0.3 version of Hadoop [22], 0.94.9 version of Hbase [5], and 0.8.2 version of Storm [4] installed.

5.1.2 Dataset

To evaluate the system with varying workload characteristics, there is a need for a data provider. We built a CDR generator for this purpose. CDRs are generated according to predefined customer profiles. Predefined customer profiles have target, time, and duration features, and each of those features have different values with their associated probability. Before the CDR generator starts working, system builds its customer base by selecting one of the predefined profiles for each customer using a Zipf distribution [23]. When CDRs are being generated, target, time, and duration values are determined by using probabilities which are defined in customer profiles.

5.1.3 Experimental Parameters

In the proposed solution, we use many parameters. As a result of our focus on scalability, *Number of Nodes* is the main parameter we study. In our heuristic algorithm, when we define clustering disaffinity in Equation 4.1, k is used for deciding how many nearest points will be taken into account. And also when we define overall quality in Equation 4.4, α is used for adjusting the relative importance of balance disaffinity and cluster disaffinity.

To experiment and evaluate the proposed solution, we feed the system with different types of customer bases. *Number of Profile Types* defines how many predefined profiles exists in the CDR generator, and it gives insight about how customers are really clustered. As we mentioned earlier, customer base is built by assigning each customer to one of the predefined profiles. Profile selection is performed using a Zipf distribution, and Z is used to adjust the skew of the distribution.

The aforementioned parameters will be analyzed separately, assuming other variables are assigned to their default values in the process. Default values are given in Table 5.1.

Parameter	Default Value
Number of Nodes	16
Number of Nearest Neighbors	10
Alpha	0.5
Number of Profile Types	6
Zipf Z	1

Table 5.1: Default values of the experimental variables.

The system is fed with $NumberOfNodes*100000$ CDRs from $NumberOfNodes*$ 1000 customers. In k-means clustering part of the algorithm where we compute micro-clusters, k is taken as $NumberOfCustomers/\ (NumberOfNodes * 30)$. In other words, each micro-cluster contain, on average, 30 profile summaries.

5.1.4 Evaluation Metrics

We evaluate the proposed solution with two fundamental metrics; quality and execution time. There are four quality metrics: *Cluster Quality*, *Balance Quality*, *Migration Quality*, and *Overall Quality*. Quality definitions and formulations are given in *Problem Definition* part of the thesis in Chapter 2.

There are three execution time metrics to analyze: clustering time, partitioning time and total time. For jth round of the system let $T(C_{ij})$ be the time node i spent on calculating the micro-clusters, $T(S_j)$ be the time that summarizer module spent on building resulting clusters and $T(P_j)$ be the time that system spent on creating a new partition for the next incoming batch of updates. We define clustering time $T(C)$ and partition time $T(P)$ as follows:

$$T(C) = \sum_j \left(\sum_i T(C_{ij}) + T(S_j) \right) \tag{5.1}$$

$$T(P) = \sum_j T(P_j) \tag{5.2}$$

Total time is the total time spent on consuming all incoming batch of updates.

5.2 Experiment Results

We now present our experimental results, studying the impact of each one of the system parameters on the evaluation metrics.

5.2.1 Scalability Experiment

To test scalability, we investigate the change in the quality metrics by running the proposed solution with varying number of nodes.

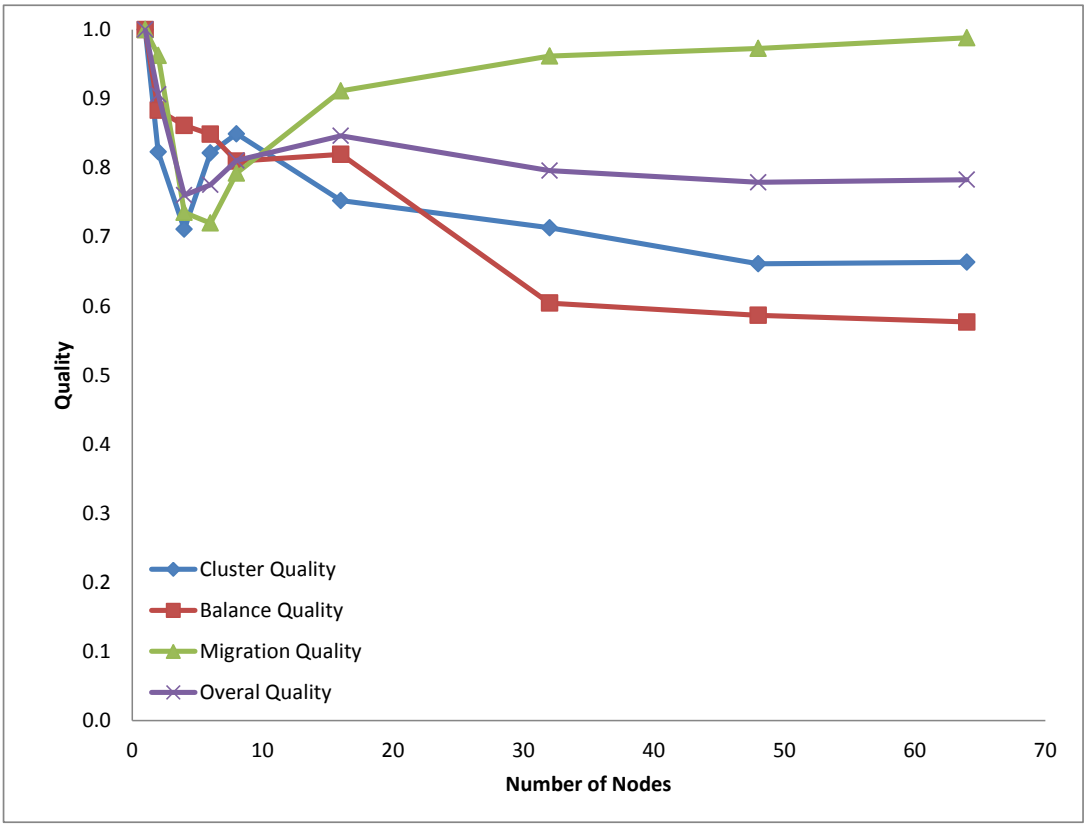

Figure 5.1: Impact of the number of nodes on the quality metrics.

Figure 5.1 plots different quality measures as a function of the number of nodes used. As we observe from the figure, for small number of nodes (< 10) overall quality drastically decreases. Cluster quality drops because when the system runs on small number of nodes, cluster affinities have similar values for every node.

Some small clusters do not exactly belong to one of the nodes, and they migrate from one node to another. As a result of these unnecessary migrations, migration quality also decreases. The decrease in cluster and migration qualities result in decrease in the overall quality as well.

Cluster quality reaches its maximum point when number of profile types nearly equal to the number of nodes. We investigate this situation further in the predefined clusters experiment.

After reaching maximum overall quality, further increase in number of nodes decreases cluster and balance quality. Cluster quality decreases because the system needs to split some of the clusters into multiple nodes, but in order to prevent a drastic decrease in cluster quality, the system decreases balance quality after some point. As a result of the decrease in both cluster quality and balance quality, overall quality also decreases but this is tolerable because system offers scalability when clustering large number of profiles.

5.2.2 Nearest Neighbor Experiment

Cluster disaffinity is calculated as average of distances to k nearest micro-clusters in target node. To investigate the effect of selection of k such micro-clusters on the quality measures, we experiment with varying the number of nearest neighbors.

Figure 5.2 plots the quality measures as a function of the number of neighbors (k) used for computing the clustering disaffinity. As we can see from the figure, increasing the number of nearest neighbors results in decreasing the cluster quality, but balance and migration qualities increase very slightly.

For high values of k, the average distance of micro-clusters to nodes becomes very similar to each other and similar clusters are formed in all nodes, albeit with decreased fidelity. This provides additional flexibility for migration and balance, as clustering is often a conflicting goal.

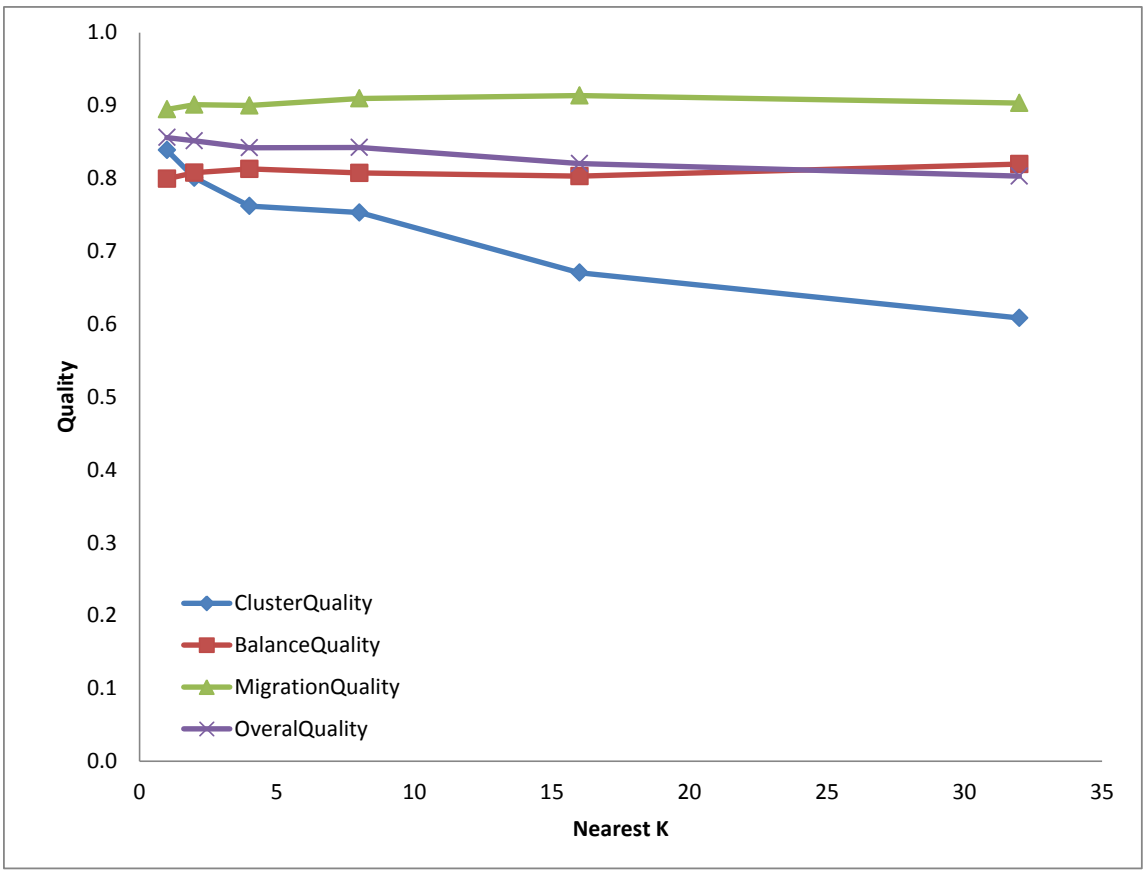

Figure 5.2: Impact of the number of neighbors on the quality metrics.

For small values of k, the clustering quality is high, because clustering disaffinity computation produces distinctive results. Although the overall quality increases as k gets smaller, it should not be selected as 1 if a more resilient result is desired. In cases when there are high number of outliers, same clusters can be formed in every node.

5.2.3 Clustering vs. Balance Experiment

As mentioned earlier, when we define the overall quality in Equation 4.4, α is used for adjusting the relative importance of balance disaffinity and cluster disaffinity. For lower values of α cluster affinity has more importance than balance affinity in our heuristic algorithm. Conversely, for higher values of α, balance affinity has

more importance.

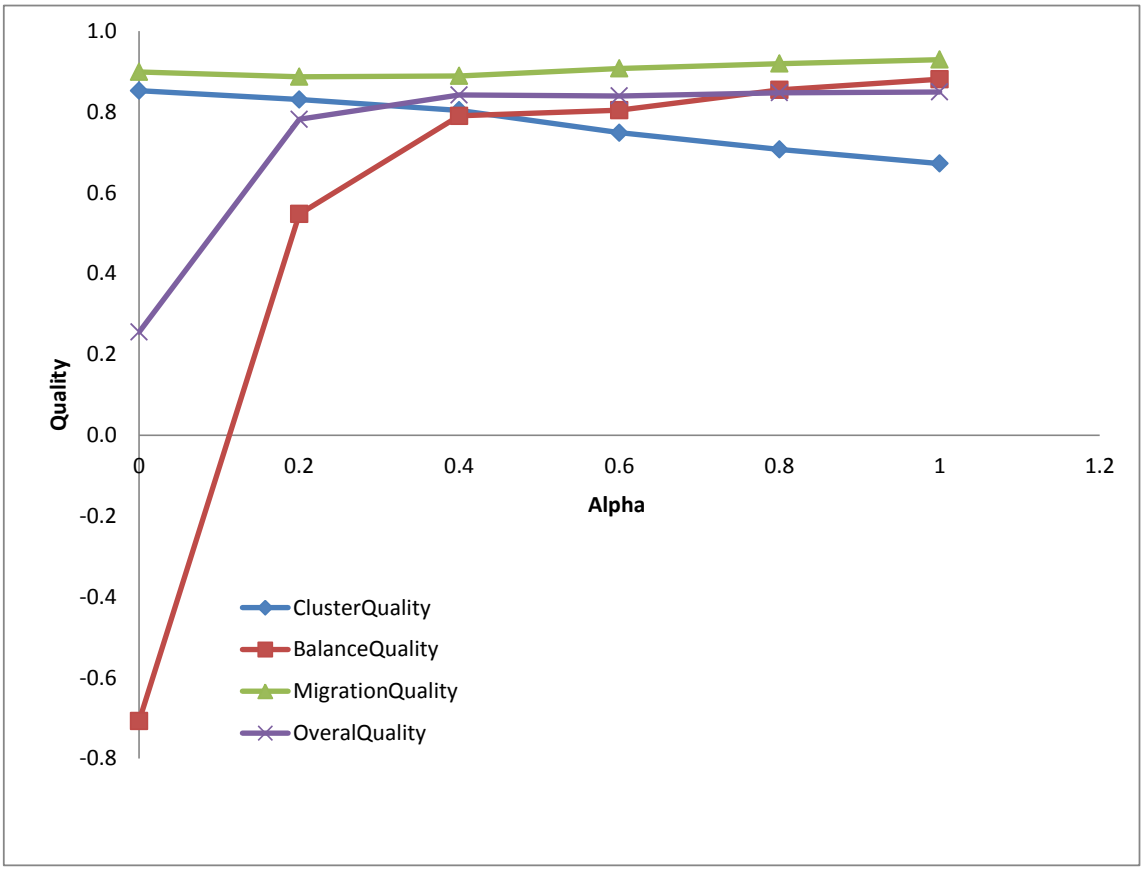

Figure 5.3: Impact of α on the quality measures.

Figure 5.3 plots the quality measures as a function of α. It shows that for lower values of α, the proposed solution cannot achieve balanced distribution and tries to collect all similar clusters to one node, and cluster quality becomes high, but balance quality becomes too low. Conversely, for higher values of α, the proposed solution just tries to achieve good balance, resulting decrease in the cluster quality. When we analyze overall quality, it reaches maximum point around $\alpha = 0.4$ and continues stable. In order to achieve the best results, α should be assigned to values in between 0.4 and 1 according to main our concern.

The figure also shows that even for $\alpha = 1$, the balance quality is not 1, because micro-clusters have variety of profiles and system migrates cluster vectors, not profiles. Therefore achieving total balance is not easy.

5.2.4 Predefined Clusters Experiment

To analyze the behavior of the proposed solution with different types of customer bases, we run experiments on different datasets. The parameter *Number of Profiles* defines the number of clusters that exist in the customer base.

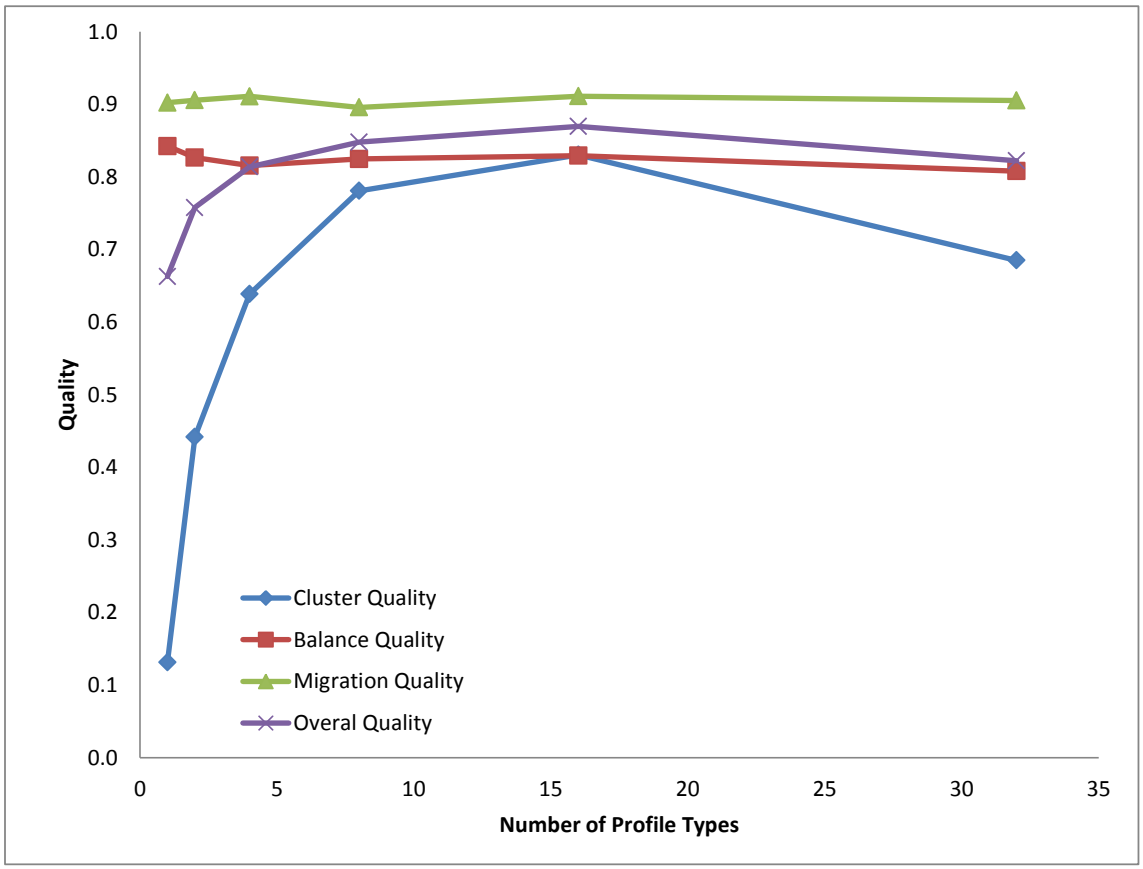

Figure 5.4: Impact of number of profile types on the quality metrics.

Figure 5.4 plots the quality metrics as a function of the number of profile types. It shows that overall quality increases as the number of profile types increases up to some point. As we mentioned before, 16 nodes are used in this experiment. When the amount of nodes and number of profiles types are nearly equal, system splits every profile type into one node and reach maximum overall quality.

When there are more nodes than customer profile types, cluster quality decreases. The reason behind that is one profile type needs to be seperated into

multiple nodes in order to preserve balance quality. Overall quality does not decrease too much, because balance quality is preserved.

5.2.5 Cluster Size Experiment

We experiment our proposed solution with varying sizes of customer clusters. As mentioned before, customer distribution to profile types is done using a Zipf distribution. For low values of Z, customers have a more balanced distribution over the clusters. Increasing values of Z creates skew.

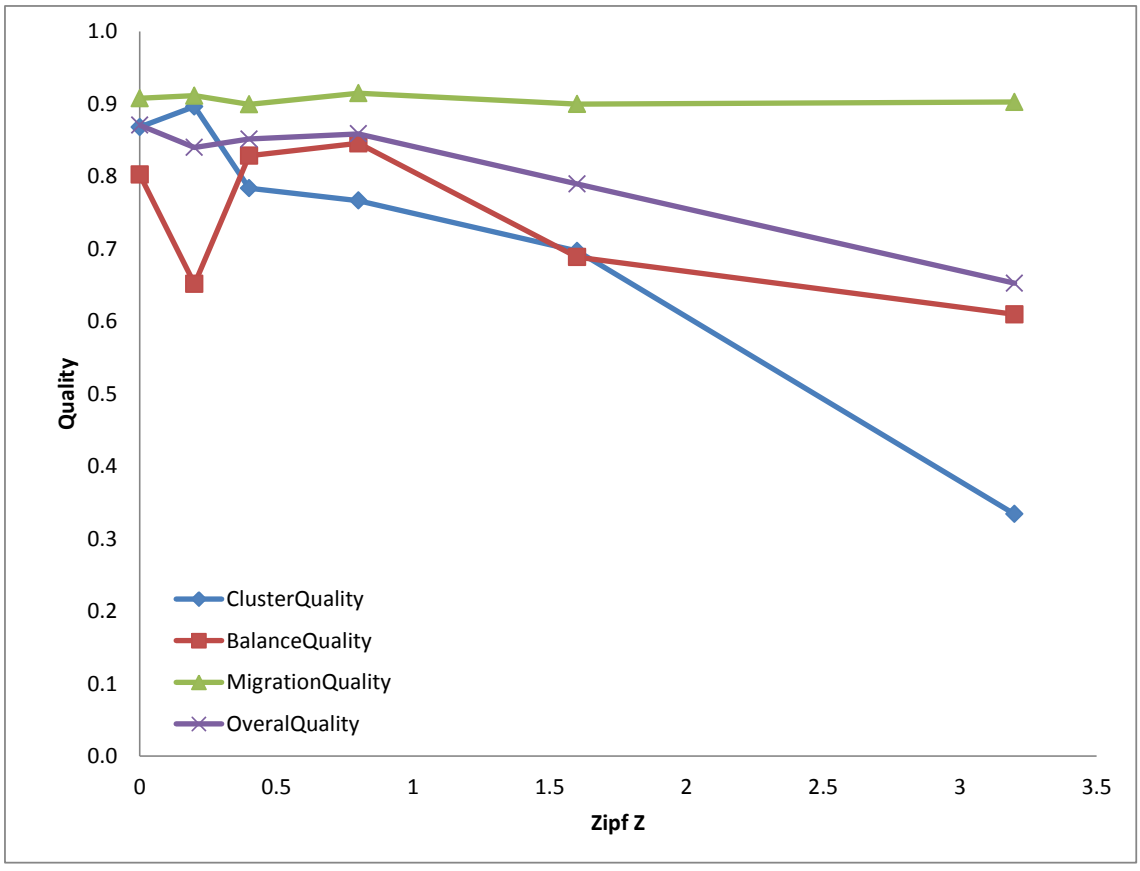

Figure 5.5: Impact of varying values of Zipf Z on the quality metrics.

Figure 5.5 plots the quality measures as a function of the Zipf skew parameter Z. We observe that the balance and cluster qualities decrease as the skew is increased. The clustering quality is effected the most. As the skew increases we

get a few large clusters. Such clusters won't fit into a single node, and as a result they must be divided. This results in decreased clustering quality. While the situation is hopeless for the very large clusters, the clustering concern tries to improve the situation as much as possible for the other clusters that can still be located on the same node. However, this happens at the cost of reduced load balance.

Overall, high skew negatively impacts the scalability of our solution. yet, for a Zipf skew value of $K = 1$, best results are obtained.

5.2.6 Execution Time Experiment

To analyze the execution time of our distributed clustering algorithm, we run our system with varying number of nodes.

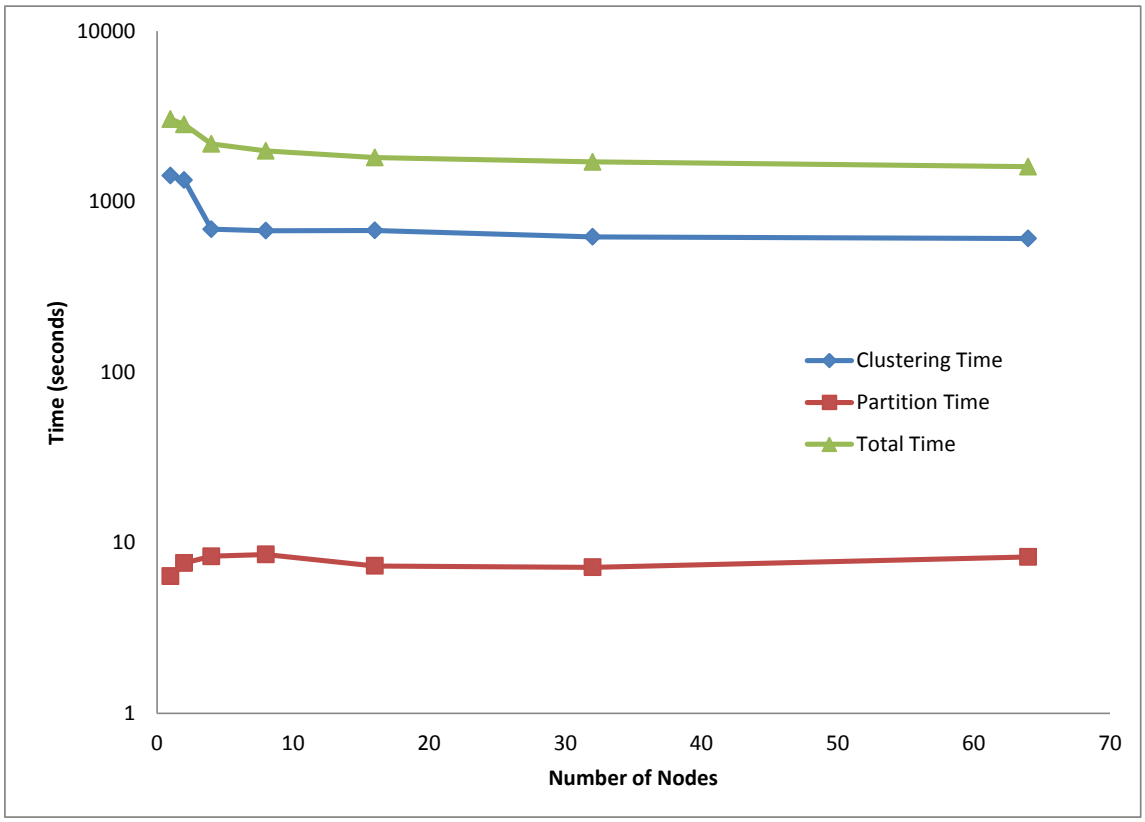

Figure 5.6: Impact of number of nodes on the clustering time.

Figure 5.6 plots the execution time of the distributed clustering step (in seconds) as a function of the number of nodes. We observe that ad the number of nodes increase the total time and the local clustering time decrease (less profiles per node), but the partition time (the centralized part) increases. This trend continues up to 8 nodes. Since the processing load for clustering is shared over nodes, it is normal that the local clustering time decreases with increasing number of nodes. On the other hand increase in possible targets in the centralized partitioning algorithm results in increased execution time for large number of nodes.

After 8 nodes, the system reaches maximum share of processing load and becomes I/O bound. Increasing number of nodes does not affect any of the time metrics after system splits processing load to the available hardware. If we had more nodes, we could have demonstrated additional speedup.

Chapter 6

Application

Telco companies provide their customers with tariffs that regulate base fees and call charges according to call types. Correctly defining tariffs can be useful to telecom companies in several ways. By means of tariff, not only the customer will gain benefits, but also the telecom companies can analyze customer orientation better and develop the necessary infrastructure and better optimize resources.

To define well targeted tariffs, telco companies need to understand call patterns of their customer base. Whenever a customer makes a call, a call detail record (CDR) comes to the data center of the telco company. The CDR has a caller associated with it and contains information about the call, such as; call target, call time, call duration, etc. When CDRs of a customer are aggregated, customer call patterns can be understood.

We built an application that uses aggregate profile clustering to perform customer segmentation for tariff optimization. The system uses CDRs as profile updates and builds aggregate customer calling profiles. Each profile has tariff information associated with it, and resulting cluster has labeled points with tariff information. The goal is to perform tariff optimization by detecting *poorly defined tariffs* and *potential new tariffs*.

The main idea is that customers who have similar call patterns should have

the same tariff, if that tariff is well defined. The system analyzes the clustering results and detects if a tariff is scattered over many clusters where the tariff in question is a minority, or concentrated on a few clusters where the tariff in question is a majority. If a tariff is scattered over many high-entropy clusters, then it could not reach its target audience. Therefore it is a *poorly defined tariff*.

Using a similar line of thought, the majority of customers in a cluster should have the same tariff, if there is a tariff that meets the expectations of the clustered customer group. Therefore clusters with high entropy are identified as *potential new tariffs*.

We modified the CDR generator we have used in our evaluation for demo purposes. Tariff labels are added as a feature to pre-defined customer profiles with their associated possibilities. When the customer base is being built, each customer gets a tariff, from its predefined possible tariff targets. Other generator features are preserved as is.

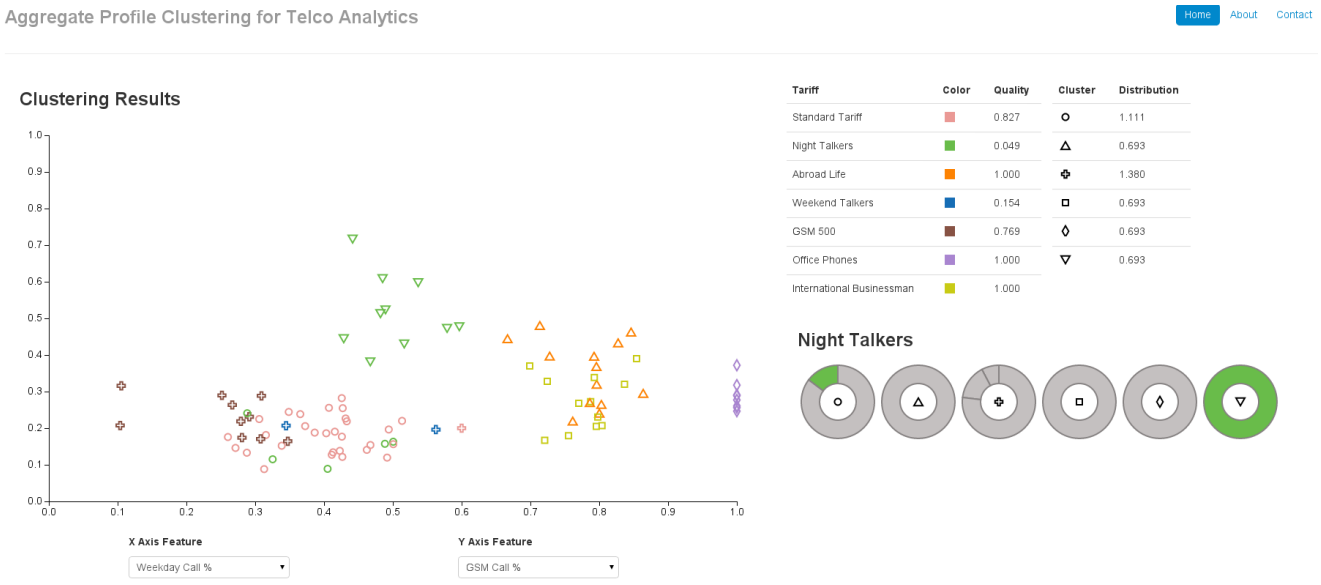

Figure 6.1: A sample screenshot from the demo dashboard.

The CDRs are processed to compute customer calling profiles. We define a number of features, based on the kind of the destination number of the call (local, trunk, GSM, international), based on the time of the call (nighttime, daytime, weekday, weekend), as well as the length of the call (short, long). For each call

category, we maintain separate aggregates of the percentage of calls falling into that category. These form a call profile vector, which is updated each time a new call is received. Whenever the infrastructure creates a new clustering, micro-clusters are also sent to the demo application.

The demo application uses a visualization dashboard to identify tariff quality, a sample screenshot of which is shown in Figure 6.1. Each tariff is assigned to a color and clusters are shown as shapes in the clustering results panel. Tariff legend with quality and cluster legend with entropy can be found on the right side of the visualization dashboard.

The analytics operator in our application computes the quality value for each tariff and displays the distribution of the tariff over the clusters. Let C_{ji} be the percentage of jth cluster members that use the ith tariff, and T_{ij} be the percentage ith tariff users that are member of the jth cluster. The quality of the ith tariff, denoted by $Q(T_i)$, is calculated as follows:

$$Q(T_i) = \sum_{i,j} T_{ij} \cdot C_{ji} \tag{6.1}$$

The quality of a cluster is taken as the entropy of the cluster with respect to the tariffs of the users contained within.

Chapter 7

Conclusion

In this thesis, we introduced the problem of scalable streaming profile clustering for the telco domain. We propose a solution that employs partitioned stateful parallelism and heuristic re-partitioning techniques. The solution consist of three main parts: micro-clustering, re-partitioning, and migration. K-means with relatively high k-parameter is used for creating micro-clusters on processing nodes. After creating micro-clusters, each node sends its micro-cluster summaries to a centralized master node and the tuple flow is temporarily paused via buffering. Master node ranks micro-clusters towards possible target nodes and calculates three metrics: clustering disaffinity, balance disaffinity, and migration affinity. Using these metrics, it calculates the overall affinity of a micro-cluster for every target node, and assigns each micro-cluster to a node that provides the highest overall affinity. The micro-clusters are considered in decreasing order of state size. When every micro-cluster is assigned to a node, the master node creates a new partition mapping, and informs every node and the splitter about the new partitioning. Each node reads its new profile information from persistent storage and performs necessary state migrations. Finally, the tuple flow is restarted.

We have evaluated the performance of our proposed solution using a CDR generator, and studied the impact of various system parameters on the performance metrics. When we experiment for scalability, we observe that the system

reaches its maximum quality when number of profile types is close to the number of nodes. After reaching its maximum, the quality decreases slowly when the number of nodes is increased. We experiment with different values for the algorithmic parameters, such as the number of nearest neighbors used (k) and scaler used to adjust the trade-off between balance and clustering quality (α) in order to decide best values for achieving maximum quality. We evaluated the system with varying number of clusters and skewed cluster sizes to understand how the proposed solution behaves for different datasets. Finally, we evaluated the running time cost of the re-partitioning, which has shown that most of the cost is incurred during local clustering at each node, which is effectively parallelized when the number of nodes used is increased.

Last, but not the least, we presented a demo application that illustrates the use of our solution for telco customer segmentation. Our application aims to perform tariff optimization by detecting poorly defined tariffs and potential new tariffs.

Bibliography

[1] S.-Y. Hung, D. C. Yen, and H.-Y. Wang, "Applying data mining to telecom churn management," *Expert Systems with Applications*, vol. 31, no. 3, pp. 515–524, 2006.

[2] M. J. Brusco, J. D. Cradit, and A. Tashchian, "Multi-criterion clusterwise regression for joint segmentation settings: An application to customer value," *Journal of Marketing Research*, vol. 40, no. 2, pp. 225–234, 2003.

[3] İ. Gür, M. Güvercin, and H. Ferhatosmanoğlu, "Scaling forecasting algorithms using clustered modeling," in *Technical Report BU-CE-1302, Bilkent University*, Agust, 2013.

[4] "Storm." `http://storm-project.net/`. retrieved March, 2013.

[5] "HBase." `http://hbase.apache.org/`. retrieved Agust, 2013.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (E. Simoudis, J. Han, and U. M. Fayyad, eds.), pp. 226–231, AAAI Press, 1996.

[7] D. Pelleg and A. W. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *International Conference on Machine Learning (ICML)*, pp. 727–734, 2000.

[8] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*. Wiley series in probability and statistics, Hoboken, NJ: Wiley, 2. ed ed., 2008.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H., "The weka data mining software," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 225–234, 2009.

[10] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[11] X. Xu, J. Jger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 263–290, 1999.

[12] I. S. Dhillon and D. S. Modha, "A data-clustering algorithm on distributed memory multiprocessors," in *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, (London, UK, UK), pp. 245–260, Springer-Verlag, 2000.

[13] "Mahout." `http://mahout.apache.org/`. retrieved Agust, 2013.

[14] J. Dean and S. Ghemawat, "MapReduce: A flexible data processing tool," *Communications ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[15] Januzaj, Eshref, H.-P. Kriegel, and M. Pfeifl, "Towards effective and efficient distributed clustering," in *The Third IEEE International Conference on Data Mining*, 2013.

[16] G. Cormode, S. Muthukrishnan, and W. Zhuang, "Conquering the divide: Continuous clustering of distributed data streams," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 1036–1045, 2007.

[17] I. Eyal, I. Keidar, and R. Rom, "Distributed data clustering in sensor networks," *Distributed Computing*, vol. 24, no. 5, pp. 207–222, 2011.

[18] B. Gedik, L. Liu, and P. S. Yu, "Asap: An adaptive sampling approach to data collection in sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, pp. 1766–1783, Dec. 2007.

[19] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, (New York, NY, USA), pp. 103–114, ACM, 1996.

[20] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, pp. 515–528, Mar. 2003.

[21] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Very Large Databases Conference (VLDB)*, pp. 81–92, 2003.

[22] "Hadoop." `http://hadoop.apache.org/`. retrieved Agust, 2013.

[23] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT Press, 1999.