Aggregate Profile Clustering for Telco Analytics

Mehmet Ali Abbasoğlu[†,‡], Buğra Gedik[†], Hakan Ferhatosmanoğlu[†]
[†] *Computer Engineering Department, Bilkent University, Ankara, Turkey*
[‡] *Korvus Bilişim R&D, Bilkent Cyberpark, Ankara, Turkey*
[abbasoglu,bgedik,hakan]@cs.bilkent.edu.tr

ABSTRACT

Many telco analytics require maintaining call profiles based on recent customer call patterns. Such call profiles are typically organized as aggregations computed at different time scales over the recent customer interactions. Customer call profiles are key inputs for analytics targeted at improving operations, marketing, and sales of telco providers. Many of these analytics require clustering customer call profiles, so that customers with similar calling patterns can be modeled as a group. Example applications include optimizing tariffs, customer segmentation, and usage forecasting. In this demo, we present our system for scalable aggregate profile clustering in a streaming setting. We focus on managing anonymized segments of customers for tariff optimization. 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 centralize node, and use a greedy algorithm with novel affinity heuristics to revise the partitioning. We present a demo that showcases our Storm and Hbase based implementation of the proposed solution in the context of a customer segmentation application.

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 [6] and customer equity management [3]. 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 [5].

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 and keeping a recent history of interactions (such as a sliding window) for the purpose of maintaining calling profiles requires high memory resources.

In this demo, we present our system for scalable profile clustering in a streaming setting. The demo highlights the effectiveness of our solution in the context of a telco customer segmentation application for tariff optimization. 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 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, we employ *partitioned stateful parallelism*. We partition the incoming stream over a set of processing nodes based on a partition by attribute (such as the caller id in a CDR) and have 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. Similarly, and for memory constrained scenarios, each node needs to store similar amount of state for maintaining the profiles.

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, in order 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*). Similarly, each node should get around the same amount state used to compute the aggregate profiles (good *memory 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 system involves a smart re-partitioning to solve this problem. It periodically adjusts the assignment of profiles to nodes such that the memory and processing balance is improved, while still maintaining a high clustering accuracy and keeping the migration cost low. Each node creates micro-clusters and computes vectors that summarize the memory and processing requirements of the profiles stored in the micro-clusters, as well as the centroid and radius information for the micro-clusters. These summaries take small amount of space and are collected at a master node. This node uses the micro-cluster summaries to come up with a new partitioning. For this purpose, we use a greedy algorithm that iterates over the micro-clusters and computes the *affinity* of each micro-cluster to each node. The best assignment that maximizes the affinity is taken. The definition of the affinity considers clustering quality, balance, as well as the migration cost.

We present a demo application that uses the proposed solutions for telco customer segmentation. The system relies on distributed stream processing middleware Storm [8] for processing updates and maintaining the profile clusters in memory, and Hbase [2] for partial fault-tolerance and for facilitating the migration.

2 Problem Definition

In this section we formalize our problem. We define our requirements as clustering quality, balance quality and migration quality.

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 $p : D \rightarrow [0..N)$ 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.

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 clusters that would be formed if the same clustering algorithm is applied on all profiles, that is $\mathcal{C} = \mathcal{A}(\{P(d) : d \in \mathcal{D}\})$.

Let us denote our clustering metric as E, which assigns an error value to a cluster. For instance, for k-means based clustering, we have: $E(C_{i,j}) = \sum_{P_k \in C_{i,j}} \|P_k - \mu_{i,j}\|$, where $C_{i,j}$ is the jth cluster in $\mathcal{C}_i$, and $\mu_{i,j}$ is the cetroid of cluster $C_{i,j}$.

Given these definitions, we define the *clustering quality* as:

$$Q^c = \frac{\sum_{C \in \mathcal{C}} E(C)}{\sum_{i \in [0..N)} \sum_{C_{i,j} \in \mathcal{C}_i} E(C_{i,j})} \tag{1}$$

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, a density-based clustering algorithm (such as DB-scan [4]) will work well. For k-means based algorithms, the distribution of k over the N nodes will be a probelem. 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].

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.

Finally, we define the migration quality as follows:

$$Q^m = 1 - \frac{\sum_{d \in D} |S(d)| \cdot \mathbf{1}(p'(d) \neq p(d))}{\sum_{d \in D} |S(d)|} \tag{2}$$

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

Finally, we denote the overall quality as Q, and define it as:

$$Q = (\alpha \cdot Q^c + (1 - \alpha) \cdot Q^b) \cdot Q^m \tag{3}$$

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

3 Solution Overview

We now describe our solution at a high level. Recall that the main idea is to update the partitioning function periodically, by collecting summary information at a master node.

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 [1] 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. Alternative orders can be used for the iteration, which we discuss later. 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 : $\mathcal{M}$, micro-clusters
Param : $N$, number of nodes
Param : $O$, ordering policy
$p \leftarrow \{\}$                          ▷ The partitioning function to be constructed
$\mathcal{M}' \leftarrow$ SORT($\mathcal{M}, O$)          ▷ Order micro-clusters based on policy
for $M \in \mathcal{M}'$ do                    ▷ For each micro-cluster, in order
    $i \leftarrow -1; a \leftarrow 0$                   ▷ Best assignment and affinity
    for $j \in [0..N)$ do                        ▷ For each node
        ▷ Compute the affinity of $M$ to the current node $j$
        $v \leftarrow A^m(M,j) \cdot (\alpha \cdot A^c(M,j) + (1-\alpha) \cdot A^c(M,j))$
        if $v > a$ then $\langle i, a \rangle \leftarrow \langle j, v \rangle$;   ▷ Update best, if necessary
    $p[d] = i, \forall d \in M$          ▷ Create mappings for the profiles in $M$
return $p$                          ▷ Return the fully constructed mapping
```

Affinity has three aspects to it: the *clustering affinity* denoted by A^c, the *balance affinity* 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., $\mathcal{M}_l = \{M \mid p_s(d) = l \wedge d \in M \in \mathcal{M}\}$.

We denote the clustering affinity of micro-cluster M_i to node j as $A^c(M_i, j)$. To compute $A^c(M_i, j)$, we look at the current assignments $\{\mathcal{M}_l\}$ and calculate how close the current micro-cluster M_i is to the already assigned micro-clusters, by computing an average distance to the k closest micro-clusters for each node. We then create a normalized metric, such that $\sum_{j \in [0..N)} A^c(M_i, j) = 1$.

We denote the balance affinity, for processing, of micro-cluster M_i to node j as $A^b(M_i, j)$. To compute $A^b(M_i, j)$, we again look at the current assignments $\{\mathcal{M}_l\}$ and calculate how much load is currently being handled by each node, considering the potential assignment of M_i to node j as well. We then create a normalized metric, such that $\sum_{j \in [0..N)} A^b(M_i, j) = 1$.

We denote the migration affinity of micro-cluster M_i to node j as $A^m(M_i, j)$. To compute $A^m(M_i, j)$, we look at the current assignments $\{\mathcal{M}_l\}$ and calculate how much total migration has been performed so far, considering the potential assignment of M_i to node j as well. We then use an exponential function to scale this value, so that when the total migrated state size is 0, then the migration affinity is 1; when the migrated state size is the total size of the aggregate profile state, it is 0; and the rate at which the migration affinity drops increases as the total migrations so far increases.

Given these definitions, we define:

$$A(M_i, j) = A^m(M_i, j) \cdot (\alpha \cdot A^c(M_i, j) + (1-\alpha) \cdot A^b(M_i, j)) \quad (4)$$

The iteration order of the micro-clusters have an impact on the algorithm performance. We experiment with several orders (both ascending/descending), based on micro-cluster memory size, processing cost, and processing cost per byte of memory.

4 System Architecture

We implemented our profile clustering technique in the context of a telco analytics platform. Figure 1 depicts the system architecture.

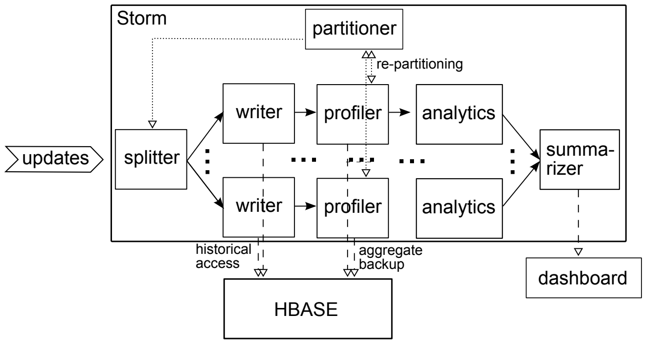


Figure 1: The architecture of the aggregate profile clustering system running on the telco analytics platform.

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, in order 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 tasks such as usage forecasting and customer segmentation. The summarizer operator is a bridge between the analytics operators and the result visualization dashboard.

5 Demonstration

We built an application that uses the system explained in Section 4 to perform customer segmentation for tariff optimization. The system uses CDRs as profile updates and builds aggregate customer calling profiles.

Telco companies provide their customers with tariffs that regulate base fees and call charges according to call types. Correctly defining tariffs not only benefits the customers by lowering their bills, but also it benefits the telco companies, as they 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 CDR is sent 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.

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, PRS), based on the time of the call (night time, 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. The clusters are maintained over these aggregate call profiles.

Each profile has tariff information associated with it and thus the resulting clusters from our distributed aggregate profile clustering solution have labeled points with tariff information. Our 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.

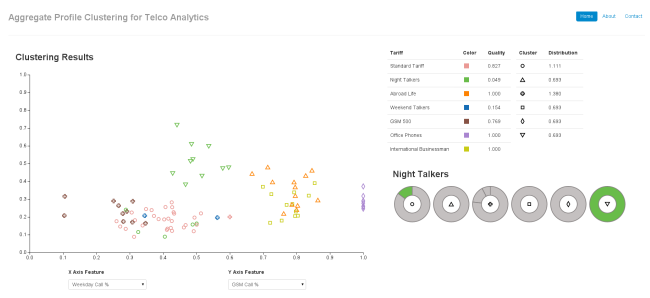
Using a similar line of thought, the majority of customers in a cluster should have the same tariff, if there is a tariff that meets the expectations of the clustered customer group. Therefore clusters with high entropy are identified as *potential new tariffs*.

5.1 Dataset

For the demo to be presented, we cannot use the real CDRs due to privacy concerns, and thus we use a CDR generator to feed the application. The CDRs are generated according to predefined customer profiles, which have target, time, duration and tariff features. Each of these features take different values based on their associated value ranges and value distribution probabilities. Before the CDR generator starts working, it builds the customer base by selecting one of the predefined profiles for each customer using a Zipf distribution. When CDRs are being generated, target, time and duration values are determined and each customer gets a tariff by using probabilities from its predefined customer profile.

5.2 Visual Dashboard

The demo application uses a visualization dashboard to identify tariff quality, a sample screenshot of which is shown in Figure 2. Each tariff is assigned to a color, and clusters are shown as shapes in the clustering results panel. In the clustering results panel, user of the system can analyze customer distribution with respect to the call profile features. Tariff legend with quality measures and cluster legend with entropy measures can be found on the right side of the visualization dashboard.



Tariff	Color	Quality
Standard Tariff	🟥	0.827
Night Talkers	🟩	0.049
Abroad Life	🟧	1.000
Weekend Talkers	🟦	0.154
GSM 500	🟫	0.769
Office Phones	🟪	1.000
International Businessman	🟨	1.000

Cluster	Distribution
○	1.111
△	0.693
✚	1.380
□	0.693
◊	0.693
▽	0.693

X Axis Feature: Weekday Call %
Y Axis Feature: GSM Call %

Figure 2: A sample screenshot from the demo 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:

$$Q(T_i) = \sum_{i,j} T_{ij} \cdot C_{ji} \tag{5}$$

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

6 Conclusion

In this work, we introduced the problem of aggregate profile clustering in a streaming setting and developed its application to telco customer segmentation. Many telco analytics operate on groups of similar customer profiles, such as usage forecasting and customer segmentation. We outlined a scalable solution to the aggregate profile clustering problem that enables these analytics performed on customer interaction streams carrying high volume data from large number of customers. We presented a demo application that illustrates the use of our techniques for customer segmentation. Our results provide useful insights on the performance of distributed systems for streaming profile clustering.

Acknowledgement. This study was funded in part by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under grants EEEAG #111E217 and #112E271.

7 References

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

[2] HBase. `http://hbase.apache.org/`. retrieved March, 2013.

[3] 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*, 40(2):225–234, 2003.

[4] 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)*, pages 226–31, 1996.

[5] İ. Gür, M. Güvercin, and H. Ferhatosmanoğlu. Scaling forecasting algorithms using clustered modeling. In *Technical Report BU-CE-1302, Bilkent University*, August, 2013.

[6] S.-Y. Hung, D. C. Yen, and H.-Y. Wang. Applying data mining to telecom churn management. *Expert Systems with Applications*, 31(3):515–524, 2006.

[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)*, pages 727–734, 2000.

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