department of computer science

facult of sciences

vrije Universiteit amsterdam

Thesis: **Master of Science**
Specialization: Computer Science

Data Clustering for Autonomic Application Replication

Jie Yang
Student number: 1430602

Supervisors:
Dr. Guillaume Pierre
Swaminathan Sivasubramanian

Department of Computer Science
Vrije Universiteit Amsterdam

Aug 2005

Abstract

This thesis has been realized in the context of GlobeDB, a system for hosting Web applications that can automatically replicate application data and maintain distributed consistency. GlobeDB adopts partial replication to reduce the network latency and traffic, and adopts data clusters to reduce the overhead of fine-grained replication. However, GlobeDB only proposed a naive clustering algorithm, which was a bottleneck to the system's performance. This thesis discusses the issue of data clustering in GlobeDB. The main challenges include evaluating the quality of clusters, selecting a clustering algorithm, and deciding on a suitable number of clusters. We systematically study various clustering algorithms and proposed some new algorithms. Experiments prove that the new algorithms can efficiently improve the performance of GlobeDB. We also propose criteria to select the best clustering algorithm and parameters according to the situation. In addition, we found that reclustering periodically can improve performance compared with non-reclustering strategy, and the best reclustering period is based on the stability of application data's popularity.

Contents

List of Figures

Chapter 1

Introduction

With the explosive growth of the World Wide Web, popular Web sites receive enormous Internet traffic. These sites have a competitive motivation to offer the best possible service to their clients at lower cost. To address the need, there has been an increasing trend toward outsourcing content delivery to networks such as Akamai [1], Digital Island [2], etc.

A Content Delivery Network (CDN) consists of a collection of edge servers that offload work from origin servers by delivering content on their behalf [7]. The edge servers belonging to a CDN may be located at different locations around the network, with some or all of the origin server's content cached or replicated amongst the CDN edge servers. For each request, the CDN attempts to locate a server close to the client to serve the request, where the notion of "close" can include geographical, topological, or latency considerations. With content distribution, the origin servers have control over the content and can make separate arrangements with servers that distribute content on their behalf. Figure 1.1 shows an example of a CDN.

CDNs are mainly used for hosting static contents, such as Web pages, documents, media files, software, etc. However, in practice, most e-commerce Web sites run Web applications, such as CGI, PHP and JSP, on top of databases. The content is generated using Web applications that take individual user profiles, request parameters, etc. into account. The response for each Web request is generated by the application code, which in turn issues transactions (read or write queries) to the underlying database. Figure 1.2 gives an example of a Web application. The clients send requests to the Web server which issues queries to the underlying database. After the database

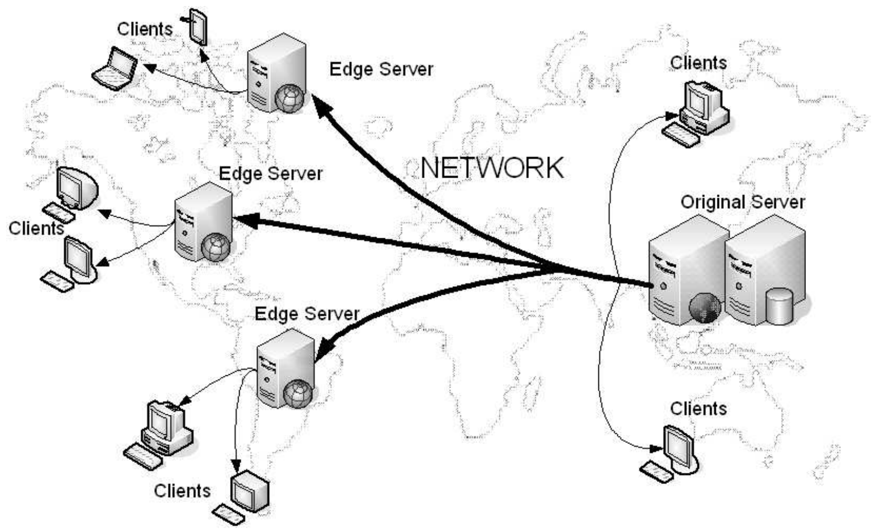


Figure 1.1: Example of Content Delivery Network.

returns the data according to the queries, the Web server then generates Web pages including the required data, and responds to the clients.

Caching static content provides little help for an interactive application. The embedded images and other objects may be delivered from an edge cache, but if the base HTML pages are generated by the application, user requests still have to travel a long distance on the Internet to contact the origin application that generates the content.

To handle Web applications, traditional CDNs use techniques such as fragment caching whereby the static fragments (and sometimes also certain dynamic parts) of a page are cached at the edge servers [13, 14, 15]. However, this strategy cannot handle the cases that database updates are frequent or have poor locality. Consequently, caching becomes inefficient. Applications that meet such problem require different solutions.

Another solution that aims at addressing this issue is edge computing, which shifts the computing of application logic in edge servers and accesses the underlying data from the original database [16, 17]. Although this solution reduces the computation load of the original server, a query still has to travel a long way to fetch data from the original server. In addition, the solution also brings a lot of load at the database. Therefore, if a Web application needs intensive accesses to its database, the effectiveness from edge computing is significantly reduced.

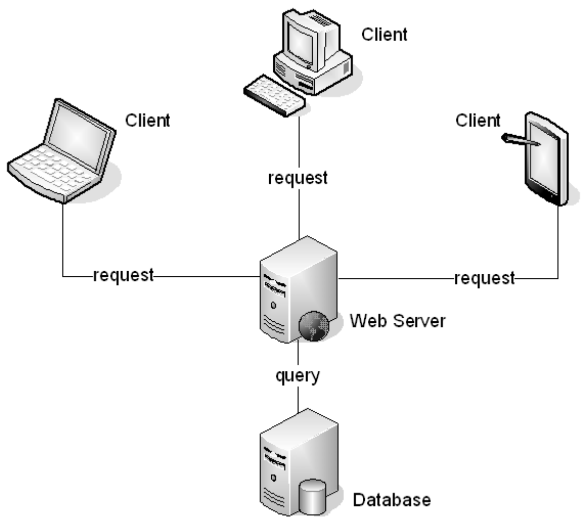


Figure 1.2: Example of Web Application.

Replicating the database is the third solution. Wide-area database replication technologies and the availability of data centers allow database copies to be distributed across the network. This requires a complete e-commerce web site suite (i.e., Web servers, application servers, and databases) to be distributed along with the database replicas. Each request only accesses the nearest edge server and therefore achieves low access latency. However, this solution increases update traffic and server loads to a large extent because each update must be propagated to all the servers. The problem is especially severe for update-intensive applications.

Instead of replicating none or all of the data to each edge server, a hybrid solution is to replicate part of the application data to each edge server. In most cases many parts of the data are not necessary at all the places. For instance, a worldwide online bookshop, such as Amazon.com, sells books in many different languages. It is reasonable that Chinese books are required mostly by customers from China and Singapore. If most requests for those books come from East Asia, then only the edge servers close to East Asia need to replicate the data about Chinese books. The above observation suggests that we can divide the application data into data units and replicate data units individually to a partial of all servers according to their individual spatial popularity. Partial replication reduces network latency compared to a non-replicated system since most queries need not access the original server. It also reduces update traffic compared to fully replicated system since an

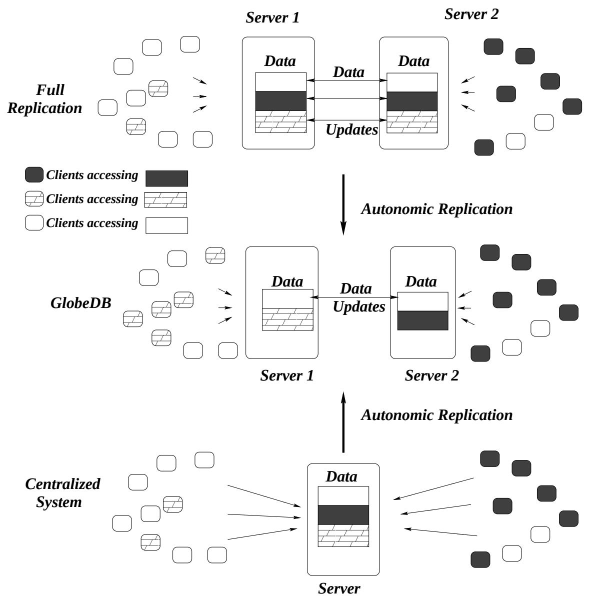


Figure 1.3: Example of benefits of autonomic replication.

update need not forward to all the edge servers.

The major challenges of partial replication are how to split the application data and where to replicate each partition. If we partition and replicate the data manually, the process requires significant effort and expertise from application developers. Furthermore, the request popularity patterns may change over time, thereby making the design of an optimal strategy even more complex.

GlobeDB is an autonomic data replicating middleware that employs partial replication to provide Web-based applications the advantages of low network latency and low update traffic at the same time [5]. One distinct feature of GlobeDB is that it automatically splits the data into individual "data units," which usually are records of a database. It distributes each data unit only to a fraction all of servers based on the number of accesses that each server issued to the data unit.

Research on replication for static Web pages suggests that the optimal replication performance of both client-perceived latency and update bandwidth can be achieved if each data unit is replicated according to its individual access pattern [4]. However, such fine-grained replication can result in significant overhead because the system has to manage partial replication information for each record. To solve this problem, GlobeDB groups a number of data units which have similar access patterns into a cluster and subsequently handles replication at the cluster level. Since the number of clusters is much lower than the number of data units, managing clusters is more feasible than managing data units. Since all the data units in a cluster have similar access patterns, they can be distributed to the same servers and consequently they can be treated like one data unit. However, if the access patterns change, then the system must perform re-clustering to sustain good performance.

Determining the right granularity of data clusters is an important design issue in GlobeDB. If the granularity is too fine, each cluster only contains a few data units. The system performance is good but the management cost is high because of the large number of clusters. If the granularity is too coarse, the management cost is reduced but it is difficult to ensure that all the access patterns in the cluster are similar. Consequently, the performance of the system is affected because it becomes hard to find a good placement strategy for all the data units in the cluster. Therefore, we need to find a good granularity to balance the two extremes.

GlobeDB proposes a simple clustering algorithm but it has not been systematically evaluated so far. Reclustering has not been studied either. The goal of this thesis is to find a good mechanism for clustering and reclustering in the context of GlobeDB. To fulfill this goal, we must overcome the following questions.

First of all, we need a criterion to guide the clustering algorithm and evaluate a given clustering method. Many clustering algorithms can be used in GlobeDB and different algorithms have different outcomes. Only once we have a good evaluation criterion, we can choose the best one for GlobeDB.

Second, we need to find a good clustering algorithm. One requirement is that this algorithm can generate clusters of high quality in terms of the evaluation criterion. The other requirement is that this algorithm is scalable, i.e., it can be applicable for a large database. If the clustering procedure spends too much time and affects the normal usage of system, it is unacceptable in most cases.

Third, we need to find a suitable granularity of data clusters in the system. As we discussed before, determining granularity is a major problem in clustering. The granularity is reflected by the number of clusters. We aim to find an appropriate number of clusters which is high enough to allow good system performance, but also low enough to minimize the management and placement costs in the system.

Finally we need to determine how often the data units should be reclustered. It is usually reasonable to predict the future access patterns according to the recent past, so we can cluster data units based on the past access patterns [4]. However, as time goes on, the access patterns may change and the old clusters may not be accurate anymore. Consequently, the quality of clusters is affected. So we need to recluster the data units. However, we cannot perform reclustering too often because the cost of clustering large number of data units is high. But if reclustering is too infrequent, the change of access patterns may degrade the performance significant. The goal is to find a balance between them.

In this thesis, I studied these issues and proposed solutions to address them. First, I adopted squared error as a criterion of data cluster quality and a cost function as the evaluation method of the system. Experiments prove that squared error is an accurate way to predict the performance of whole system. Second, I applied a set of clustering algorithms and compared their performance. In addition, to improve the existing GlobeDB and K-Means clustering algorithms I developed a novel clustering algorithm which is especially well suited for periodic reclustering. All the experiments are based on simulations with both synthetic traces and real-world traces taken from two Web sites of different types. Comparisons of different clustering algorithms and metrics show that the new algorithm can achieve similar system performance as the others but reduces the computation cost dramatically. To determine the granularity of clusters, I examined the variance of system performance when the number of clusters changes, and proposed a suggestion of what an appropriated number should be. Finally, I studied how often reclustering should occur and presented a suitable frequency and trigger for reclustering.

The remainder of this thesis is structured as follows. Chapter 2 introduces the design and implementation of GlobeDB. Chapter 3 gives an overview of data clustering techniques. Chapter 4 studies multiple clustering algorithms used in GlobeDB. Chapter 5 evaluates the performance of clustering using a simulator. Chapter 6 focuses on reclustering. Finally, Chapter 7 describes the future work and concludes.

Chapter 2

Related Work on GlobeDB

Since all the work in this thesis is based on GlobeDB, we first need to introduce it. Generally speaking, GlobeDB is a system for hosting Web applications, which handles distribution and partial replication of application data automatically and efficiently [5]. In this chapter, we introduce GlobeDB from the viewpoint of the system architecture, components and workflow. Finally, we introduce the cost function used in GlobeDB to evaluate the performance of system.

2.1 System Architecture

Figure 2.1 shows the system architecture of GlobeDB. A Web application, including code and data, is hosted by multiple edge servers spread across the Internet. The code is written using a dynamic Web page technology such as PHP and is replicated to each of the edge servers. It is executed each time the Web server receives an HTTP request from its client and issues read/write accesses to the relevant data in the database to generate a response. Each request is assumed to be redirected to its closest edge server. Communication among edge servers is realized through wide-area networks.

As already mentioned, GlobeDB relies on partial replication. This means that an edge server may only hold part of the clusters and that we need a special mechanism to read or write the data. Access to the data is realized by a data driver, which is the central component of the system. The driver can locate the data units required by the application code and maintain consistency of replicated data. When one of the edge servers receives a request, it always issues database queries to its local data driver. The data driver

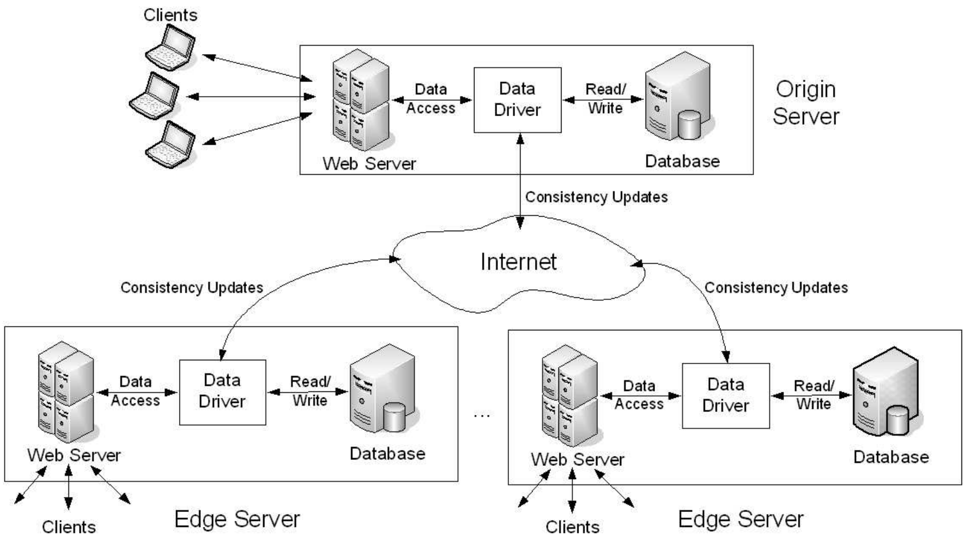


Figure 2.1: System Architecture

is responsible for finding the relevant data either from the local database, or from a remote edge server if the requested data are absent locally. In additional, when handling write data accesses, the driver is also responsible for ensuring consistency with other replicas of the updated data unit.

Generally, a traditional CDN contains a number of identical edge servers. However, from the point of view of GlobeDB, servers may take three different roles: edge server, master server and origin server, which are introduced below.

Edge Server

Edge servers are the basic components in the system. Each of them can (partially) replicate application data and serve Web clients. An edge server is made of a Web server, a data driver and a database server. Because an edge server only has a part of the application data, sometimes it has to fetch data from the origin server to fulfill Web requests from clients.

Master Server

To keep the system consistent, GlobeDB serializes concurrent updates of a replicated data unit by means of a *master server*. Each data cluster is assigned a master server which is in charge of updating the cluster. For

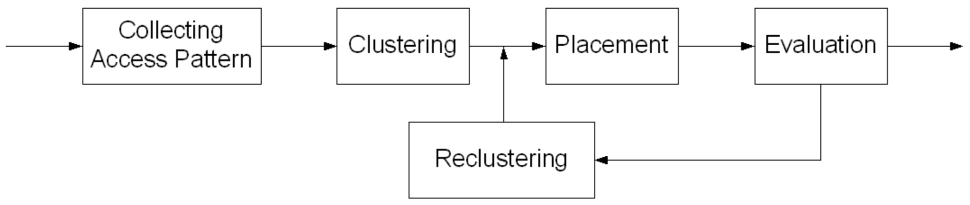


Figure 2.2: System Workflow

example, the server which has the least write-latency of a cluster can be selected as the master of this cluster. All updates to a cluster are forwarded to its master. The master consequently processes the update requests and then propagates the results to the replicas of this cluster. This provides sequential consistency, which means that developers can write applications as if the data resided in a centralized location, and ignore the distribution issues.

Origin server

Each application must also have an *origin server*, which has a full replica of the application data. It is in charge of making application-wide decisions such as clustering data units, selecting master servers and placing data clusters. In addition, if an edge server needs any data or wants to check the cluster ID of a data unit, the origin server is responsible to provide them. The origin server can be the master server of some clusters as well.

2.2 System Workflow

As we have seen, the primary good of the origin server is to determine the system configuration.

A number of operations need to be realized before the system reaches its optimal configuration. This includes collecting the data units' access patterns, clustering the data units, placing the clusters and reclustering the data units. Before a client fetches a data unit from the system, the data unit is already clustered, placed and may be reclustered. Finally the driver locates and transfers the data unit to the client. Therefore the workflow of the system is logically composed of the following modules, as illustrated in Figure 2.2.

2.2.1 Collecting Access Patterns

Since the system clusters data units based on their access patterns, the initial task is to collect the access patterns. The access pattern of a given data unit i is a vector $A_i =< r_{i1}, r_{i2}, \ldots, r_{im}, w_{i1}, w_{i2}, \ldots, w_{im} >$, where r_{ij} and w_{ij} are the number of read and write queries respectively that server j has issued to the data unit. In a specified period, the system records the number of accesses to each data units from each server and forms the access pattern for the next procedure.

2.2.2 Data Clustering

As mentioned before, data units with similar access patterns are clustered together to reduce the overheads of data management and placement. The origin server is responsible for clustering the data units during the initial stages of system. Clustering takes n data units and their access patterns as input, where n is usually a large number. Clustering outputs the membership of k clusters, that is the definition of which data units belongs to which cluster. The number of clusters k is usually much smaller than n. Data clustering algorithms are detailed in Chapter 3 and 4.

2.2.3 Data Placement

To reduce network latency and update traffic, GlobeDB supports partial replication on the granularity of data cluster. So each data cluster is replicated to a subset of all servers. Proper replication of data cluster is important for obtaining good performance of the system. This module receives the access patterns and clusters' membership as input, and then places the clusters according to the placement algorithm. GlobeDB defines a family of placement heuristics Px where an edge server hosts a replica of a data cluster if this server generates at least $x\%$ of requests to this cluster. Obviously, the value of x affects the placement of clusters and thereby impacts the performance of the system. A high value of x may lead to few replicas, so many requests will be forwarded to the origin server. On the other side, a low value of x will lead to many replicas, so the update traffic may also be high. GlobeDB evaluates the cost value for placement configurations for different values of x, and selects the one that yields the least cost as the best placement configuration. The output of this module is the configuration of placement, as well as the membership of data clusters and access patterns of data units.

2.2.4 Performance Evaluation

In face of ever-changing access patterns and network condition, the system must adapt its configuration. This evaluation module receives the output from the placement module and then measures the performance of the current configuration. The system can adjust the configuration according to the feedback from the evaluation module. The most important metrics are latency, bandwidth and server load. The evaluation aggregates these three metrics using a cost function, which is introduced in Chapter 2.4.

2.2.5 Data Reclustering

Data clustering only works if data units once clustered do not change their access pattern too much. However, if they do, then the clusters must be re-evaluated and re-placed. The reclustering module receives the already existing clusters, the access patterns and the evaluation results as its input and decides when to start up reclustering according to the system load and access variance. We discuss the issue of reclustering in Chapter 6.

2.3 Replicated Data Management

Once data units have been clustered and placed, the data drivers are in charge of finding the data units required by clients. Since GlobeDB provides partial replication, it can happen that some data required to answer a query are absent from the edge server where they are requested. In this case, the data driver forwards read queries to the origin server and gets the result.

To answer a write query, the data driver first gets the ID of the data cluster containing the required data unit. Then the driver finds the location of the cluster and its master. If the edge has not the necessary information, it can ask for the origin server. Consequently, the driver requests the master server to execute the write query. Finally, the master server will return the result to the edge server and propagate the update to all the replicas of this cluster. The pseudocode for executing queries by the data driver is shown in figure 2.3.

```
if (read) { /* read query */
  Execute query locally;
  if (execution returns result)
      return result;  /* Data unit present locally */
  else /* Query returned empty result as data not present locally */
      execute on origin server and return result;
} else { /* write query */
  Get cluster id of data unit from local or origin server.
  Find the master of the cluster from cluster-membership table.
  Execute query on master server and return result;
  The master server updates all replicas of the cluster;
}
```

Figure 2.3: Pseudocode used by a data driver for executing queries

2.4 Cost Function

When the system clusters and places data units, the origin server must select the best replication strategy to provide the system with optimal performance. First of all, the system administrator must specify what "optimal" actually means. In GlobeDB, we represent the overall system performance by a *cost function*. A cost function aggregates several evaluation metrics into a single figure. By definition, the best configuration is the one with the least cost. In GlobeDB, we use the following cost function to measure the overall performance of a replication strategy s during a time period t:

$$cost(s,t) = \alpha * lat(s,t) + \beta * bw(s,t) + \gamma * load(s,t)$$

where *lat* is the average read and write latency, *bw* is the average bandwidth used by each request, and *load* is the average load in all servers for handling a request. The value α, β and γ are weights associated to metrics *lat*, *bw* and *load* respectively.

To provide the system with enough flexibility, these weights can be set by the system administrator based on the system constraints and application requirement. A large weight implies that its associated metric has more influence in selecting the "best" strategy. For instance, if the administrator primarily wants to reduce the bandwidth consumption, then β can be increased.

Based on the assumption that the past access patterns are a good indicator for the near future, the configuration with the least cost can be selected as the best strategy for the near future. This assumption has been shown to be true for static Web pages, and we expect the dynamic Web pages to have similar behavior [4]. Therefore, finding the “best” system configuration now equals to evaluating the value of the cost function for each configuration and selecting the one which has the least cost.

2.5 Summary

This chapter introduced the GlobeDB application replication system. Different from traditional CDNs and Web applications, GlobeDB inserts a data driver in each edge server, and adopts some techniques such as data clustering and partial replication to improve the performance of system. Not all servers are the same in the system. According to their duties, they can be defined as edge server, master server or origin server. The cost function is an important tool in GlobeDB to evaluate the performance and thereby the application administrator can adjust the system flexibly. In the next chapter, we focus on the clustering techniques that may be used for GlobeDB.

Chapter 3

Overview of Data Clustering

Cluster analysis has a long history and wide application fields, such as social science, biology, chemistry, statistics, pattern recognition, machine learning, and so on. It is an important research issue in multivariate analysis. In this thesis, we only study the effect of clustering which are relevant to GlobeDB.

Data clustering is defined as the unsupervised classification of patterns into groups (*clusters*) [3]. A pattern is usually presented by a vector of numbers, which represent different *features* of an object. The features are extracted and measured by some metrics. For example, an object in a 3-dimension space has 3 features: the values in dimension of X, Y and Z. In GlobeDB, such features are the number of read and write queries that each CDN server has issued to a given data unit. The goal of clustering is to gather similar patterns together and separate dissimilar patterns from each other. Generally speaking, a cluster contains a number of similar patterns. Figure 3.1 illustrates an example of data clusters in a 2-dimension space which contains lots of objects. After data clustering, all the objects are grouped into 4 clusters.

The first step of clustering is to extract the patterns of objects which need to be clustered. In GlobeDB we obtain the access patterns by observing the number of requests issued by each server. Then we must define a similarity metric so that we can judge which patterns are similar. Section 3.1 introduces some popular similarity measurements. Finally we classify the patterns into clusters based on their similarities. A number of clustering algorithms have been proposed in literature [9, 8, 3]. In section 3.2, we introduce several important clustering methods. Since an object, or a data unit can be represented by its pattern, we treat them as equal when dis-

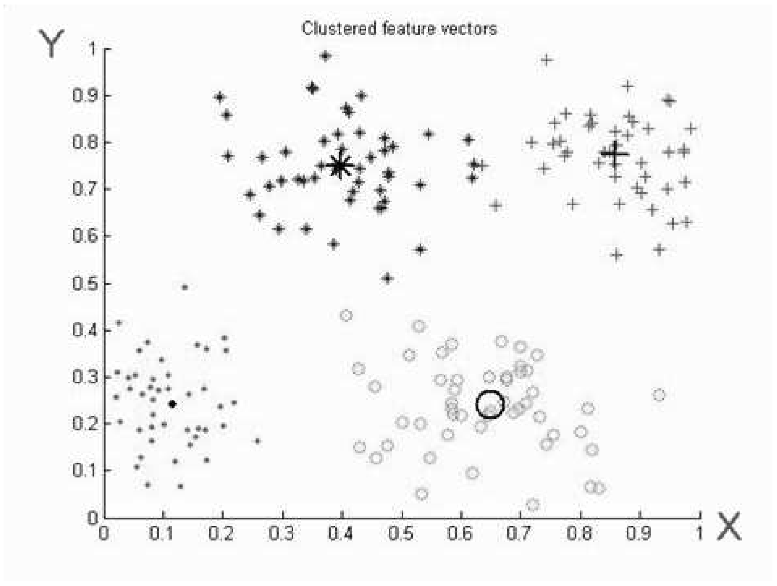


Figure 3.1: Example of data clusters in a 2-dimension space

cussing clustering.

3.1 Similarity Measurement

As mentioned before, a measure of the similarity between two patterns is essential to all clustering procedures. Different similarity metrics can produce different results, so the similarity measure must be chosen carefully. Some classical measurements are introduced in the following.

It is common to calculate the similarity between two patterns using a distance measure defined on the feature space. We only focus on the well-known distance measures used for patterns whose features are all continuous.

Perhaps the most common metric for measuring the similarity of two patterns is a distance metric Δ that maps $\mathbb{R}^m \times \mathbb{R}^m$ onto $\mathbb{R}^1$. Here $\mathbb{R}^m$ is a m-dimension space. The vector of m elements $X_i =< x_{i,1}, x_{i,2}, \ldots, x_{i,m} >$. In GlobeDB, we define X_i as the access pattern of a data unit d_i in database, and $x_{i,k}$ as the number of read or write queries that the server S_k has issued to the data unit d_i. The bigger the distance, the less similar the points. Therefore distance represents dissimilarity between two objects. The following lists some common metrics of distance (as dissimilarity).

Euclidean Distance The most popular metric for continuous features is the *Euclidean distance* defined as:

$$D(X_i, X_j) = \sqrt{\sum_{k=1}^{m} (x_{i,k} - x_{j,k})^2} = ||X_i - X_j||$$

City Block Distance A similar metric like Euclidean distance is the *City Block Distance*, defined as:

$$D(X_i, X_j) = \sum_{k=1}^{m} ||x_{i,k} - x_{j,k}|| = ||X_i - X_j||$$

Minkowski Distance This metric is a generalization of the previous two. It takes a parameter p. When $p = 1$ we obtain the City Block Distance; when $p = 2$ we obtain the Euclidean Distance. The *Minkowski Distance* is defined as:

$$D_p(X_i, X_j) = [\sum_{k=1}^{m} (x_{i,k} - x_{j,k})^p]^{1/p} = ||X_i - X_j||_p$$

Chebyshev Distance The *Chebyshev Distance* is similar to the City Block distance, but it only take one feature of the two patterns to compute. It is defined as:

$$D(X_i, X_j) = max||x_{i,k} - x_{j,k}||, \quad here\ 1 \leq k \leq m$$

Correlation Coefficient Another classical metric is the *Correlation Coefficient*, which denotes the quality of a least squares fitting to the original data. The coefficient has the range of $[-1, 1]$. It is close to 0 if two patterns are similar and is close to 1 or -1 if they are dissimilar. The *Correlation Coefficient* is defined as:

$$D(i,j) = \frac{\sum_{k=1}^{m}[(x_{i,k} - \bar{x}_i)(x_{j,k} - \bar{x}_j)]}{\sqrt{\sum_{k=1}^{m}(x_{i,k} - \bar{x}_i)^2 \times \sum_{k=1}^{m}(x_{j,k} - \bar{x}_j)^2}}$$

where $\bar{x_i}$ is the mean of data unit i's access pattern, $x_{i,k}$, $(k = 1, 2, \ldots, m)$.

Although this metric is not a distance metric, its absolute value can be used to measure the similarity of patterns.

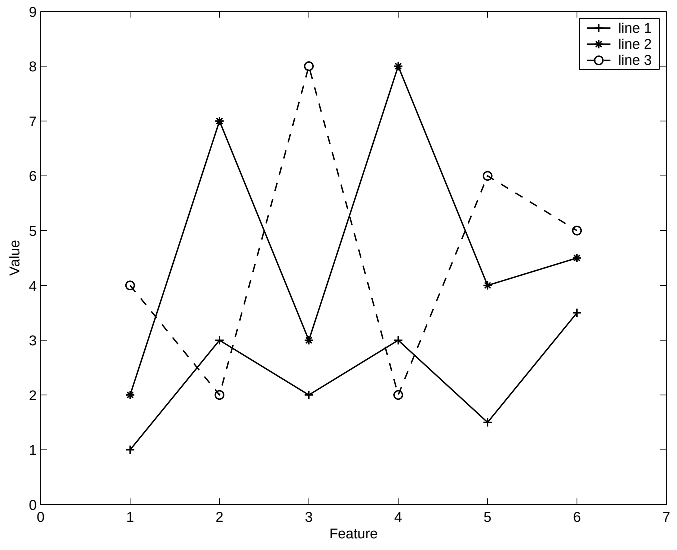


Figure 3.2: Example of similar and dissimilar shapes.

Other Distance Metrics There exists many other metrics, such as *Cosine distance*, *Canberra Distance*, *Hamming distance*, *Jaccard distance*, *Mahalanobis Distance* and so on. Since they have different applicable fields and are similar to the above metrics in some extent, we do not study them all.

The Minkowski distance (and its specialization, the Euclidean and City Block distance) has an intuitive appeal as it is commonly used to evaluate the proximity of objects in two or three-dimensional space. It works well when a data set has "compact" or "isolated" clusters. The drawback of Minkowski metrics is that the largest-scaled features tend to dominate the others. So they are not very reliable for features of different scale. Potential solutions to this problem include normalization of the continuous features (to a common range or variance) and other weighting schemes.

Correlation-based metrics, such as correlation coefficient and cosine degree, are suitable for applications more concerned with the "shape" than with the "scale" of patterns. Similar shape means that the distributions of major features and minor features in two patterns are roughly the same, while their scales (values) may be variant. In figure 3.2, line 1 and line 2 are similar in that both of them have major values in feature 2, 4, 6 and minor values in feature 1, 3, 5. But line 3 is dissimilar to the others since it has different major and minor values compared to the others.

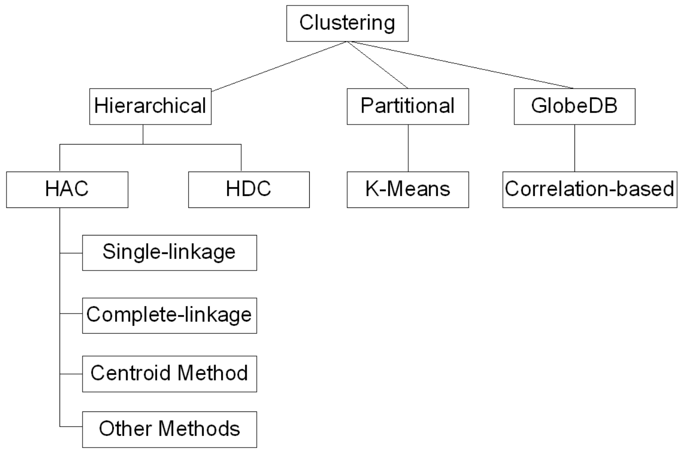


Figure 3.3: A taxonomy of clustering approaches

Different distance measures incur different cluster outputs. It is an important factor in the process of clustering, and we should choose it carefully based on the characteristics of application. The selection of distance measures for GlobeDB is discussed in Chapter 5.

3.2 Clustering Techniques

Once a similarity metric is selected, one needs to choose an algorithm to actually cluster the objects. We present a selection of clustering algorithms below. Figure 3.3 shows a taxonomy of these algorithms.

3.2.1 Hierarchical Clustering

Hierarchical clustering is the most traditional family of clustering algorithms. It includes agglomerative approach and divisive approach. In the following, we refer to the problem of clustering n objects into k clusters.

Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) requires a similarity metric between two clusters, where each cluster contains one or more objects. If each of the clusters only contains one object, the cluster-to-cluster similarity metric equals to the object-to-object similarity metric defined in Section 3.1.

The algorithm starts by creating n clusters, each one containing a single object. It then identifies the closest pair of clusters and merges them into a single cluster. This operation is repeated until the desired number of clusters is achieved.

HAC can use different clusters similarity metrics, which are presented below.

Single Linkage (nearest neighbor) Method If C_1 and C_2 are two clusters, then the distance between them is defined as the smallest distance between a member of C_1 and a member of C_2. Namely,

$$d_{(C_1)(C_2)} = min\{d_{rs} : r \in C_1,\ s \in C_2\}$$

where r, s denotes "object r and s".

Complete Linkage (farthest neighbor) Method This method is the opposite of the single linkage method in that the distance between two clusters is defined as the largest distance between a member of C_1 and a member of C_2. Namely,

$$d_{(C_1)(C_2)} = max\{d_{rs} : r \in C_1,\ s \in C_2\}$$

Centroid Method The distance between two clusters is defined as the "distance" between the cluster centroids. If

$$\bar{x}_i = \sum_{j \in C_i} \frac{x_j}{|C_i|}$$

is the centroid of the n_1 members of C_1 and $\bar{x_2}$ is similarly defined for C_2, then

$$d_{(C_1)(C_2)} = P(\bar{x}_1, \bar{x}_2)$$

where P is a proximity measure such as Euclidean distance or correlation coefficient.

Other Methods In the literature there are some other methods to decide which two clusters should be merged, such as *Ward's Method*, *Median method*, *Group Average Method (average linking)*, *Lance and Williams Flexible Method*, *Information Measures Method*, etc. All those methods have only very specific application domains, so we do not study them in this thesis.

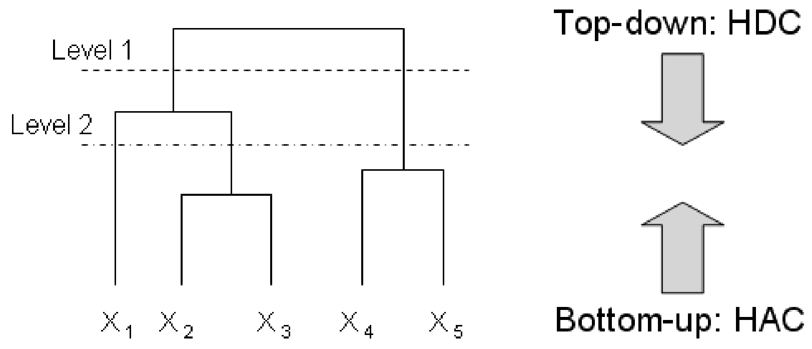


Figure 3.4: HDC and HAC

Hierarchical Divisive Clustering

The Hierarchical Divisive Clustering (HDC) clustering is the opposite of HAC. It starts with a single cluster containing all the objects and repeatedly splits clusters, until the desired number of clusters is achieved. Figure 3.4 illustrates the difference between HDC and HAC. HDC is a top-down approach while HAC is a bottom-up approach. If we stop at the level 1, we can obtain 2 clusters as $\{x_1, x_2, x_3\}$ and $\{x_4, x_5\}$. If we stop at the level 2, we can obtain 3 clusters as $\{x_1\}$, $\{x_2, x_3\}$ and $\{x_4, x_5\}$.

The decisions about which cluster to split and how to split it can be based either on one feature considered at a time, or on all features considered simultaneously. The former are called *monothetic* and the latter *polythetic* techniques. When n is very big, the polythetic technique has high computation cost computation. On the other hand, the Monothetic approach is simple and fast but sensitive to errors of the deciding variable.

HDC includes two main procedures. First we need to find which cluster should be divided. Then we need to specify how to split the cluster and assign the membership of objects afterwards.

In the first step, we can choose the cluster which is either the biggest or the largest. Biggest means that it has the maximum number of objects. Largest means that it has the maximum diameter, which is defined as the longest distance between two objects in the cluster.

The second step depends on the technique of split decision. First we find an object which is most dissimilar from all the other objects in the same cluster. Then we split it out from this cluster and form a new cluster. The next step is to test all objects in the original cluster. If an object is more similar with the new cluster than the original one, it is assigned to the new cluster.

This technique requires a distance metric between an object and a cluster. One solution is to define it as the distance between the object to the cluster's centroid.

$$D(x, C) = d(x, \bar{x}_c) \qquad \bar{x}_c \text{ is the centroid of cluster } C$$

where x is an object, C is the cluster and d is a distance measurement.

Another solution is to define it as the average distance between the object to all the (other) objects in the cluster.

$$D(x, C) = \begin{cases} \dfrac{1}{n} \displaystyle\sum_{y \in C} d(x, y) & \text{if } x \notin C \\ \\ \dfrac{1}{n-1} \displaystyle\sum_{y \in C, x \neq y} d(x, y) & \text{if } x \in C \end{cases}$$

3.2.2 Partitioning Clustering

Traditional hierarchical clusterings are often slow and suitable only for small values of n. Partitioning Clustering, on the other hand, is an efficient algorithm and applicable for large values of n. The most famous partitioning method is K-Means. It works as follows.

First we specify in advance how many clusters are required: this is the parameter k. The algorithm starts by selecting k random points and considering them as k cluster centers. Objects are then assigned to their closest cluster according to the distance measure, such as Euclidean or correlation.

Then the centroid of each cluster is calculated. These centroids are taken to be the new centers for their respective clusters. Next, since the centers have changed, all objects have to be reassigned to their closest cluster.

The whole process is repeated until some stop condition is reached. When the stop condition is reached, we consider that the system is almost stable and will not change too much anymore. The stop condition can be that, for example, the number of data units which changed their membership is lower than a threshold, or the algorithm has already iterated enough times, or the average distance between each object and its centroid has not changed too much in recent iterations.

One potential issue of K-Means is that it may create empty clusters. If this is problematic for the application, we can split a “big” cluster into two small clusters. Similarly to techniques used in HDC, the “big” cluster can be defined as the one which contains most data units, or the one which has the biggest diameter.

Another issue is that it is difficult to obtain global optimization in K-Means because the randomly selected centers in the first step are important factors to affect the output of clusters. One solution is to repeat the K-Means algorithms several times with different initial centers, and then output the one which leads to the minimum average squared error. Another solution is to use a different clustering algorithm to generate initial centers for K-Means. In this case, K-Means can rather be looked as an optimal method for the previous clustering. If we simply need to update existing cluster rather than start from nothing we can use the previous clusters as the starting point for the re-clustering operation. We return to this topic when discussing re-clustering within GlobeDB in Chapter 6.

3.2.3 GlobeDB Clustering

The original clustering algorithm proposed in GlobeDB adopts Correlation Coefficient introduced in section 3.2 as its similarity metric between data units. Assume $Sim(i, j)$ is the similarity between two data units d_i and d_j. The two data units can be clustered together if $Sim(i, j) \geq Cx$, for some threshold value Cx, where $0 \leq Cx \leq 1$. Figure 3.5 shows the increment of clusters when the Cx increases. Obviously, the higher the Cx, the more clusters will be generated, and vice versa. It is because if Cx is high, only very similar data units can be put together, otherwise they have individual clusters. Thereby a high Cx can create more clusters.

The GlobeDB algorithm is simple and efficient. It first selects a data unit d_i which has not been clustered to form a new cluster k. Then it scans all the data units that are yet to be clustered. If a data unit d_j is sufficiently close to d_i, i.e., $Sim(i, j) \geq Cx$, d_j is merged into the cluster k. This operation is repeated until any of the data units has been assigned to a cluster. Obviously, the value of Cx has an impact on the quality of the clustering.

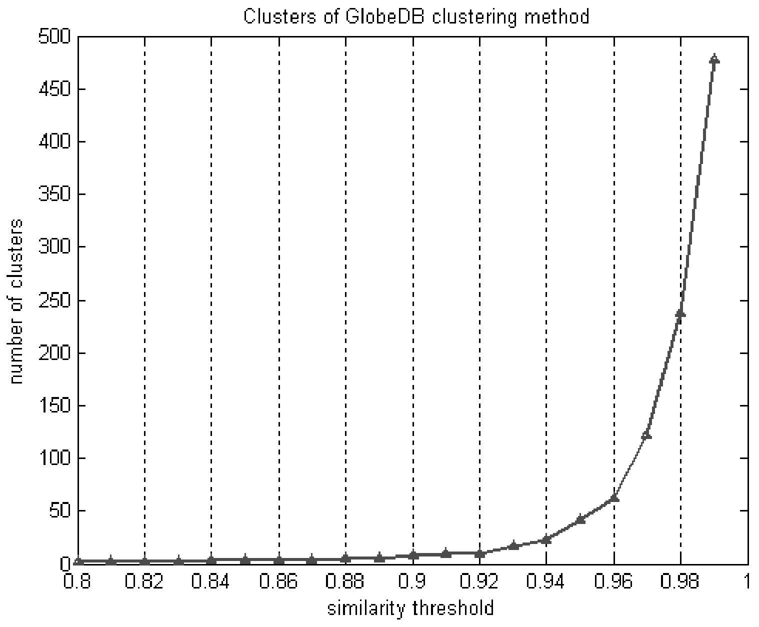


Figure 3.5: The increment of clusters when the similarity threshold of GlobeDB clustering increases.

3.3 Summary

This chapter gave an overview of data clustering and introduced some important similarity measurements and clustering methods which can be used by GlobeDB. The original GlobeDB clustering algorithm has been introduced as well. In the next chapter these measures and methods are applied in our system and the performance of these clustering are compared so that we can find a best one for GlobeDB.

Chapter 4

Proposed Clustering Algorithms

As we have seen in Chapter 3, many clustering algorithms have been developed to work in a large variety of environments. Clearly, different goals lead to different techniques. However, it is difficult to tell which clustering works best for a given application. Before systematically comparing algorithms, we present here some specific clustering algorithms that I developed specifically for GlobeDB. In the following we use n, m and k to present the number of data units, edge servers and data clusters respectively.

4.1 Revised GlobeDB Clustering

The original clustering algorithm proposed in GlobeDB is introduced in section 3.2.3. We call it the GlobeDB Clustering. However, one pitfall of this algorithm is that it cannot specify how many clusters will be created given a threshold Cx. By other words, it cannot take the desired number of clusters as a parameter of clustering. This shortcoming brings a difficulty when we compare it with other algorithms. After all, the number of clusters is an important factor that affects the performance of our system.

In order to specify the number of clusters, I developed a revised algorithm which adopts binary search to find the proper Cx in the range of $[0, 1]$. In the first iteration, we select any reasonable Cx as the first threshold. We run GlobeDB Clustering and count k', the number of generated clusters. If $k' = k$, the clustering stops and outputs the results. If k' is less than k,

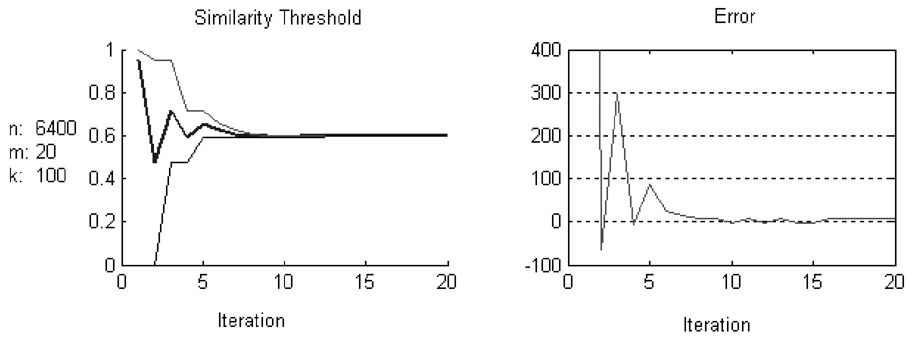


Figure 4.1: Revised GlobeDB Clustering

we increase the Cx to the half of its upper range to create more clusters. Otherwise we decrease the Cx to the half of its lower range.

However, sometimes the proper Cx is difficult to reach because the GlobeDB clustering cannot create the exact k clusters no matter how many iterations it takes. Therefore we have to change the stop condition. We decided that we can stop clustering if the number of clusters created is close to the goal, or if the clustering has already executed enough iterations. In addition, we can record the result of each iteration, and only output the best one.

Figure 4.1 presents the effects of revised GlobeDB clustering. The middle line in the left column shows the variance of threshold in each iteration. The other two lines are the bounds of threshold. The right column shows the difference between the target number of clusters and the actual number of created clusters in each iteration. In the revised clustering, the target number is almost achieved and the clustering can stop after about 8 iterations.

4.2 Revised K-Means Clustering

We introduced the K-Means clustering algorithm in Chapter 3.2.2. However, the performance of K-Means depends to a large extent on the initial cluster centroids, which are selected randomly. Therefore the performance is nondeterministic and can vary significantly because of the variance of the initial centroids.

To solve the problem, we defined a revised K-Means clustering which takes the cluster centroids generated by another clustering into account. We refer the clustering served for K-Means as the “basic clustering”. The basic clustering can be a hierarchical or GlobeDB clustering algorithm. Although the clusters created by a basic clustering are applicable by the system, their membership and centroids of clusters may not be stable from the viewpoint of the K-Means algorithm, so K-Means can take the existing cluster centroids as its initial stage and adjust the membership and centroids of the clusters until they are stable. Therefore revised K-Means clustering can improve the quality of clusters created by another clustering. In our following experiments, we adopted the revised GlobeDB algorithm as our basic clustering.

4.3 Binary Coding Clustering

Binary Coding Clustering (BCC) is a new algorithm which is totally different from the existing ones because it is not based on any distance measure, but on the placement of data units. Intuitively, since each data unit will be placed in some server eventually, we can cluster the data units which will have the same placement into a cluster.

The first step of this algorithm is to define a placement pattern for each data unit. Then each placement pattern is assigned with a unique code so that it is easy to gather the same patterns together. Finally, all the data units with similar patterns are clustered together. Therefore, the algorithm includes three phases: transferring, coding and clustering. They are detailed in the following.

Phase 1: Transferring

Assume there are n data sets, m CDN servers and we need k clusters. After clustering and placing, a data unit d_i has a placement pattern $P_i =< P_{i,1}, P_{i,2}, \ldots, P_{i,m} > (P_i = 0, 1)$. Here $P_i = 0$ means not to place the data set in the server i, and $P_i = 1$ means to place it. It must be pointed out that the placement pattern is a binary pattern, so it is very easy to compute. This is essential for the efficiency of Binary Coding Clustering.

Once each data unit is assigned an access pattern, we can transfer the access pattern into a placement pattern according to the GlobeDB placement algorithm. For example, if a data unit d_i has access pattern $< 4, 10, 7, 1 >$, its fraction of access on the servers is $< 0.18, 0.45, 0.32, 0.05 >$. If we define $Px = 0.25$, then only the second and third server can replicate the data unit, therefore the placement pattern is $< 0, 1, 1, 0 >$.

Binary Coding Clustering first computes placement patterns from the access patterns. All data units which have exactly the same placement pattern will be replicated to the same server(s), so we can put them into a cluster. We can therefore create at most 2^m clusters. However, if $k < 2^m$, then we need to merge some similar clusters or patterns into one cluster. The next two phases address this problem.

Phase 2: Coding

The goal of this phase is to assign a code to each placement pattern. If two patterns have similar codes, they can be clustered together. To compute the code, each server is assigned a weight. Because the servers which have most access are more important in the system, we assign them higher weights. To avoid different patterns having the same code, we adopt powers of 0.5 as the value of weights. For example, if the total access pattern of servers is $< 43, 102, 74, 12 >$, then the weight vector is $< 0.125, 0.5, 0.25, 0.0625 >$.

The code associated to a data cluster is defined as the aggregation of each server's weight multiplied by each corresponding value in the data unit's placement pattern. In the above example, the code of a pattern $< 0, 1, 1, 0 >$ is $0 * 0.125 + 1 * 0.5 + 1 * 0.25 + 0 * 0.0625 = 0.75$. Obviously, codes range within $[0, 1]$.

After coding, two similar patterns can have close codes because they have similar binary distribution. On the other hand, if two patterns have different values in high-weight servers, their codes will be very different as well. Therefore, we can simply cluster all the patterns based on their codes in the next phase.

Phase 3: Clustering

After we can get all the codes of data units, we can cluster these patterns based on their codes. Assume the minimum code is c_{min} and the maximum one is c_{max}. Then we separate all the codes into k segments of width $(c_{max} - c_{min})/k$. The patterns whose codes are followed into the same segment can be put into one cluster. For example, we need 3 clusters and the real range of

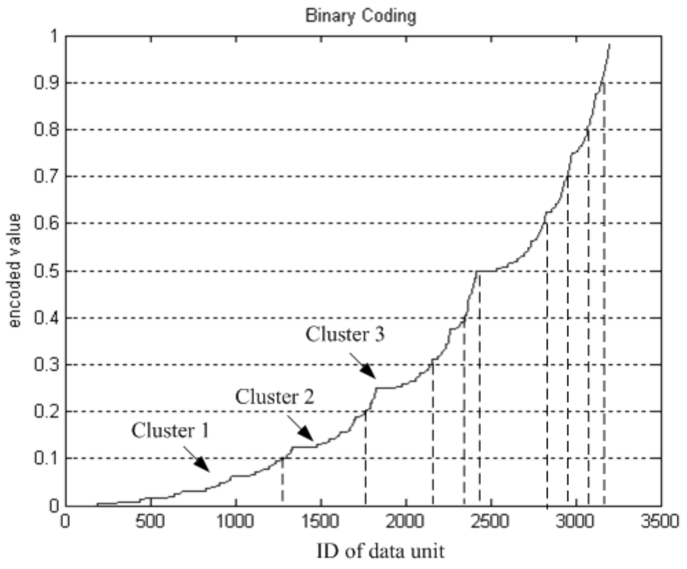


Figure 4.2: Binary Coding Clustering

codes is [0.3, 0.9]. So any pattern whose code falls into [0.3, 0.5] is assigned to the first cluster, any pattern whose code falls into (0.5, 0.7] is assigned to the second cluster, and all the remaining patterns are assigned to the third cluster.

If k is big, the range of codes of each cluster is small. It may happen that some clusters does not contain any data unit if k is too big. We simply ignore such case since the number of empty clusters is minor compared with k, and the empty clusters do not impact the system performance very much.

Figure 4.2 illustrates the Binary Coding algorithm. Here $n = 3200$ and $k = 10$. The data units are sorted by their codes. The dash line separate the data units into 10 segments, each of which is a cluster. The first about 1300 data units are put into the first cluster, the second about 500 data units are put into the second cluster, and so on.

It must be noted that the normal sequence of the system is to first cluster data units and then place data clusters. Contrarily, the BCC algorithm first performs the placement algorithm then clusters data units. This will only work well if the placement algorithm is efficient, such as the original one in GlobeDB. However, some fine placement algorithms have good performance but their complexity is high [6]. If we adopt these placement algorithms, then the efficiency of BCC will be impacted. This makes the BCC algorithm more suitable for reclustering than for clustering since before reclustering a placement of data clusters already exists. Therefore we can obtain the placement pattern directly and only perform the coding and clustering phases, which are very efficient.

4.4 Summary

In this chapter, we introduced three data clustering algorithms developed for GlobeDB. The Revised GlobeDB Clustering and Revised K-Means Clustering are improved algorithms on the basis of existing clusterings. The Binary Coding Clustering is a new algorithm which clusters data units based on their placements. In the next chapter these clustering algorithm are evaluated and compared using a GlobeDB simulator.

Chapter 5

Clustering Evaluation

So far we have discussed many clustering algorithms. The question is which one is the best one for GlobeDB. This chapter aims to answer this question.

First of all, a good clustering algorithm should reduce the latency, bandwidth and server load of the system in the greatest extent. After all, the main goal of GlobeDB is to improve the performance of Web applications.

Second, a good clustering algorithm should achieve high performance with the lowest possible number of clusters. As we discussed early, fine-grained replication can result in best performance, but also significant overhead of partial replication management. In addition, fine-grained replication will create too many data clusters which cause high computation time for the placement algorithm, especially for placement algorithms sensitive to the number of data clusters [6].

Third, a good clustering algorithm in GlobeDB must have high scalability with respect to the number of data units. Indeed, a distinct character of Web applications is that their underlying data may be enormous. Different from the traditional shops, an e-business company can provide much more commodities online since it does not need them in stock. For example, Amazon.com could sell over 110,000 different books as early as 1996 [23]. Nowadays, even for a middle-size e-commerce Web site, it is normal to have millions of data units in the database. In this scenario, the efficiency of clustering becomes a critical issue for the system. A time-consuming clustering algorithm is unacceptable in most systems even if it has excellent quality.

To summarize, a good clustering algorithm for GlobeDB should achieve high performance of the system with the least number of clusters, be scalable and efficient for large databases. In this chapter, we first detail the GlobeDB simulator that we built to evaluate clustering algorithms. Then we propose evaluation criteria of clustering. Finally, we compare the clustering algorithms, analyze the results and conclude. In the following experiments, we use n, m and k to present the number of data units, edge servers and data clusters respectively.

5.1 GlobeDB Simulator

Running experiments using the actual GlobeDB prototype is expensive because for each experiment, the system has to evaluate the network latency, calculate the consumed bandwidth and record the server load for each incoming request. Some experiments may need several days to complete. When we study the clustering algorithms, lots of parameters needs to adjusted and evaluated. It is therefore unacceptable to execute each experiment with the real system. Instead, we decided to execute our experiments using a simulator, which can reproduce the behaviors of the system and evaluate the performance at low cost. In this section, we introduce the workflow of the simulator, how it simulates the system and how it computes the cost of operations.

5.1.1 Simulator Workflow

According to the system workflow introduced in Chapter 2.2, we designed the simulator workflow as shown in Figure 5.1. It includes the following five modules.

Input Module

The data input module is used to generate the access patterns of all the data units. In GlobeDB, each data unit d_i's access pattern is modelled as a $2 * m$-dimensional vector, $A_i =< r_{i,1},r_{i,2},\cdots,r_{i,m},w_{i,1},\cdots,w_{i,m} >$, where $r_{i,j}$ and $w_{i,j}$ are respectively the number of read and write accesses issued by the edge server S_j to the data unit d_i. Unfortunately, we do not have proper experimental data from a Web application, which are usually commercial secrets. To address this problem, we adopted two different methods to generate the input data.

One method is to synthesize the input data given numbers of data units, servers, clusters, the total amount and ratio of reads and writes. According

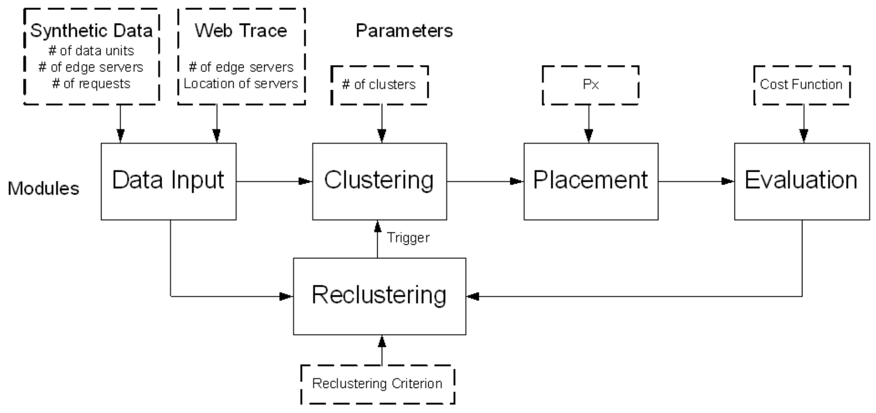


Figure 5.1: System Simulation

to previous research, the popularity of data units and the distribution of user requests can be modelled using power-law distribution (Zipf's law) [21]. This law can help us specify the access pattern of data units. If we sort the data units or edge servers according to their requests, each of them follows a Zipf distribution, which is defined as $P_n = k/n^{\alpha}$, where $0 < \alpha \leq 1$, n is the rank of an item and $1 \leq n \leq N$, P_n is the frequency of occurrence of the n^{th} ranked item, and k is a coefficient which depends on α and N to ensure that $\sum_{n=1}^{N} P_n = 1$.

The other method is to extract the access patterns from a real Web trace of static Web pages, such as an Apache Web log. Here a data unit is represented by a static Web page. From the trace, we can obtain the number of Web pages, the number of requests to each Web page, and the geographical location of each client by computing the request's IP address into longitude and latitude using the GeoIP library [24]. We decided to arbitrarily specify the number and location of edge servers. Each request is assumed to be served by the geographically closest server. Then we can obtain the number of requests issued from each edge server and thereby the access pattern of each data unit. In our experiments, we used two very different types of Web traces to examine the applicability of GlobeDB. One trace comes from the Web site of *www.cs.vu.nl, the Department of Computer Science, Vrije Universiteit*, which has high number of Web pages but low number of requests. The other one comes from *www.electoral-vote.com, the US Electoral Vote site*, which has low number of Web pages but high amount of requests in some days.

Using Web traces can represent real Web applications, but it only provides the access pattern of reads. Moreover, the number of data units and requests on a Web trace are fixed. On the other hand, synthetic data are flexible so we can easily change and evaluate different parameters. If a clustering algorithm can obtain satisfying results in both cases, we can say it is effective.

Clustering Module

The clustering module receives an access pattern from the input module, and clusters the data units using some clustering algorithm. This module can take the expected number of clusters as a parameter. The output contains the actual number of clusters created and the membership of data clusters. I developed several clustering modules that implement the following algorithms: Hierarchical Agglomerative Clustering, Hierarchical Divisive Clustering, Revised GlobeDB Clustering, Revised K-Means Clustering and Binary Coding Clustering. The interfaces of all these modules are the same.

Placement Module

The placement module receives as input the access pattern of data units and the membership of data clusters from the clustering module. This module can accept Px, the threshold percentage of replication, as its parameter. If a server generates at least $x\%$ of data access requests to a cluster, then the cluster is replicated in the server. The output contains the access patterns, the membership and placement of each cluster.

Evaluation Module

The evaluation module receives the output of placement module, which already includes all the necessary information for evaluation. Then the module calculates the cost of each request if it runs in the system. The simulation and calculation of cost is detailed in Section 5.1.2. This module can output the value of each performance metric, e.g., latency, bandwidth and server load, and the result of the cost function. In addition, the module can record the elapsed time of the experiment.

Reclustering Module

The reclustering module is a special module for studying the performance gain from reclustering. It takes a reclustering condition as its parameter, such as the reclustering period, and triggers the system to recluster the data units. This makes the system re-place and re-evaluate new clusters. Finally, the reclustering module records and outputs the performance variance after reclustering by means of the evaluation module.

5.1.2 Cost Function Computation

In the evaluation module, we need to simulate the running of GlobeDB so that we can compute the cost of each request in the system. This section details how this computation is done in the simulator.

Each query can cause several actions in some servers. To calculate the cost for each query, we first need to specify the cost of each action. To simplify the system, we make the following assumptions:

1. There are 1 origin server and p edge servers (excluding the origin server).

2. Each client addresses requests to its closest server, and the latency between the client and the server is a constant.

3. The latency between any pair of servers is a constant.

4. The consumed bandwidth to transfer any query or receive any result is a constant.

5. We only consider the following 4 atomic operations: send a request, result or query, receive a request, result or query, read the local database and write (update) into the local database. The load for each atomic operation is the same. Load of any other operation is negligible.

Note that most of these assumptions can easily be removed with minor modifications to the way the cost function is computed.

In the simulation, we assign each read or write operation a fixed latency cost, each upload or download a fixed bandwidth cost, and each atomic operation a fixed load cost. The three different costs are used in the cost function introduced in Chapter 5.2.2.

Next, we need to study the actions of the system when it receives a query and the corresponding costs of the actions. We already introduced how the system works in Chapter 2. To compute the metrics for the cost function, we must specify the actions of each case and compute the cost of

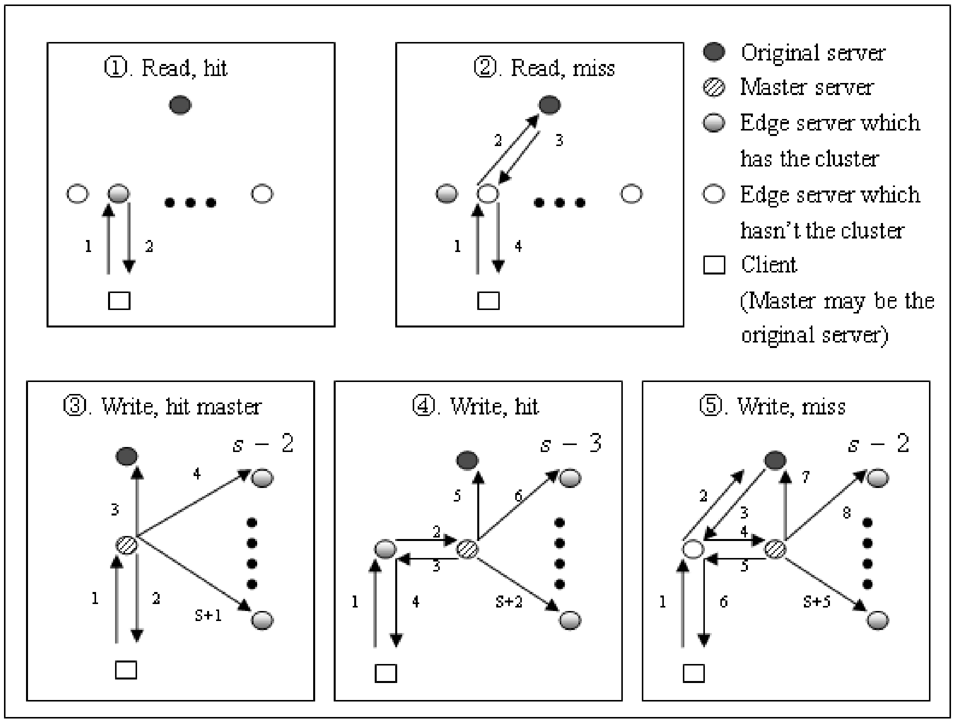


Figure 5.2: Actions of GlobeDB to handle a query

each metric precisely. Figure 5.2 presents the 5 types of queries that have different actions to deal with a query.

In the figure, "miss" means the server does not replicate the required data unit and "hit" means it does. "hit master" means the server is the master of the cluster containing the required data unit. If the query is an update, it must be forwarded to its cluster's master which consequentially updates all the servers which have the cluster. So it is different for "hit master" and "hit". In the following, we present the costs of each case. Assume S_0 is the server which received a Web request from a client, C is the cluster containing the required data unit, and s is the number of replicas of C.

1. Read, Hit The server S_0 has the cluster C and can reply the client directly. The latency for the client is 2, the load in S_0 is 3 (receive a request from a client, read the local database, and return the result to the client), and the total bandwidth is 2 (receive and answer the client's request).

2. Read, Miss The edge server S_0 has to first fetch the required data unit from the origin server, and then return it to the client. The latency to the client is 4, the total load is 7 (4 in S_0 and 3 in the origin server), and the total bandwidth is 6 (4 in S_0 and 2 in the origin server).

3. Write, Hit Master The master server S_0 has the cluster C and it can reply the client directly, but the master has to update all the other $s - 1$ servers which have replicated C. Obviously the latency is 2 with respect to the client, but it is more difficult to analyze the cost of load and bandwidth. The load is $s+2$ in the edge server (receive a request, read the local database, return the result, propagate the update query to $s - 1$ other servers), and 2 in each of the $s - 1$ servers which received the update query (receive a update query and update the local database). So the total load is $3 * s$. The total bandwidth is $2 * s$ (2 in S_0 to receive and answer the request of client, $s - 1$ in the master to propagate the update requests, and $s - 1$ totally in all the other servers to receive the update request).

4. Write, Hit The edge server S_0 (it may be the origin server as well) has to first send the update query to the master. Consequently, the master updates all the other $s-1$ servers which have replicated the cluster C. Using the same method as above to analyze the cost, we get that the latency to the client is 2, the total load is $3 * s + 2$, and the total bandwidth is $2 * s + 2$.

5. Write, Miss The edge server S_0 first has to contact the origin server to determine which server is the master of the cluster C. Then the edge server sends the update query to the master and consequently the master updates all the other $s - 1$ servers which have replicated C. The latency to the client is 4, the total load is $3 * s + 10$, and the total bandwidth is $2 * s + 7$.

5.2 Performance Evaluation Criteria

To evaluate whether a cluster has high quality, we at least need a criterion. In this section, we detail two evaluation criteria. One captures the quality of the clustering itself, while the other reflects the effect of the clustering on system's performance.

5.2.1 Squared Error

A widely used evaluation criterion for a clustering algorithm is the *squared error*. It is defined as the total distance between each of data units in a cluster to the cluster's centroid. If a cluster has a large squared error, then it contains many dissimilar data units. Usually we wish to decrease the squared error to make the clusters converged and thereby obtain high quality of clusters.

This metric is very easy to compute, however it does not directly reflect the effect of the clustering on system's performance. For this reason, we also need another criterion.

5.2.2 Cost function

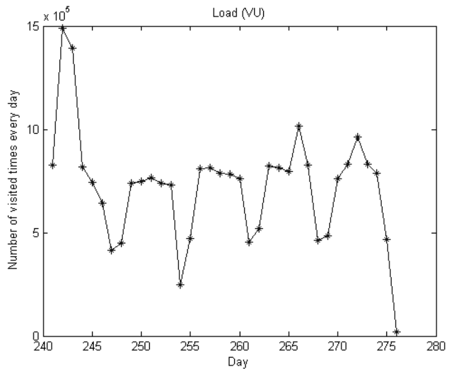
GlobeDB aims to reduce the latency, bandwidth and server load of Web applications. We use the cost function to evaluate the performance of the whole system. The cost function is more costly to compute than the squared error, but it can directly represent the system's performance. In Section 5.4.2 we verify if the squared error can represent the quality of clustering to a certain extent, so that we can decide if squared error is good enough to replace the cost function.

5.3 Experiment Setup

This section discusses the setup of experiments. In the following experiments, n, m and k represent the number of data units, edge servers and clusters respectively.

We use two static Web traces in the experiments. "US-VOTE" and "VU" represents the trace from the Web server of "www.electoral-vote.com" and "www.cs.vu.nl" respectively. Both traces recorded the Web accesses from September to October, 2004. The trace of US-VOTE only contains thousands of data units, but it has lots of access each day, especially in the vote day. On the other hand, the trace of VU contains about half a million data units, but most pages have only few accesses each day and the load is almost stable in a month. Figure 5.3 illustrates the variation of total access number per day in the two traces.

The read/write ratio is another important factor to impact the performance of the system. In most Web applications, there are more reads than



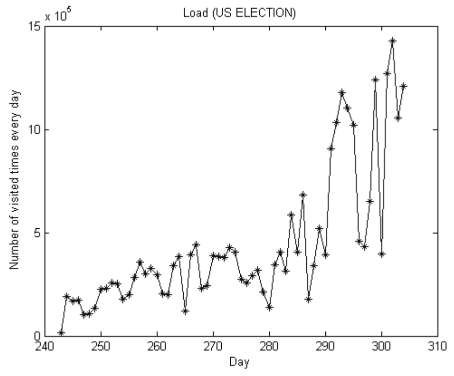


Figure 5.3: Total access number in traces per day

writes. We use 4/1 as the default read/write ratio in the following experiments.

When using synthetic data, we need two α values as the parameters of Zipf distribution: α_1 is used to model the popularity of data units, and α_2 is used to model the popularity of edge servers. We did experiments on real Web traces and found that the popularity of both data units and servers can fit the Zipf distribution. We set $\alpha_1 = 1$ and $\alpha_2 = 0.6$ as the default values.

In addition, in the input module the average number of requests to each data unit on each server is set to 5. The number of servers is at most 100, but the number of data units can be as big as millions.

The number of edge servers m is fixed as 10 by default. Because this thesis does not study the issue of placement, we simply fix the Px as $1/m$. Therefore if a server will replicate a cluster if it received more than the average number of requests to the cluster.

In our system, we think the latency is the most important factor for a Web application. Therefore, we set the default values of α, β and γ as 10, 1 and 1 respectively.

5.4 Experiment Results

So far, we have four similarity measures:

- *Euclidean Distance*
- *City Block Distance*

- *Chebyshev Distance*
- *Correlation Coefficient*

We have two evaluation criteria:

- *Squared Error*
- *Cost Function*

We have five clustering algorithms:

- *Hierarchical Agglomerative Clustering (HAC)*
- *Hierarchical Divisive Clustering (HDC)*
- *Revised GlobeDB Clustering (GlobeDB)*
- *Revised K-Means Clustering (K-Means)*
- *Binary Coding Clustering (BCC)*

In this section, the goal of this section is to find the best ones of them. All the following experiments were performed under the GlobeDB simulator.

5.4.1 Selection of Similarity Measure

First of all, we need to select a similarity measure between data units, since it is the basic of clustering. To find the best one, we adopted the US-VOTE trace and the HDC algorithm, adjusted the number of clusters, and computed their overall cost. Table 5.1 presents the results.

Obviously, the correlation based similarity measure is the best one since it issues the lowest costs. The reason is that similar shape is more important than spatial distance when we calculate the similarity of two access patterns

k	10	55	100	145
City Block	7.69795	7.45141	7.45141	7.44371
Euclidean	7.69795	7.45141	7.44068	7.44427
Chebyshev	7.69795	7.45141	7.43958	7.44317
Correlation	**7.69534**	**7.41978**	**7.41963**	**7.41963**

Table 5.1: Comparison of different similarity measures.

in GlobeDB. Two access patterns with similar shapes can be replicated to the same servers, even though their spatial distance can be large. Among the distance measures, the correlation coefficient is the only one to evaluate the similarity of shapes. In the following, we use the correlation coefficient as our similarity measure.

5.4.2 Selection of Performance Evaluation Criterion

Squared error is widely used to evaluate the quality of clusters. In GlobeDB, the cost function is the final criterion to evaluate the performance of system. We checked if the squared error can represent the quality of clustering. The results are presented in Table 5.2. The first two lines adopt synthetic data as input and the only difference is the number of data units. The next two lines use the traces from US-VOTE and VU as input respectively. The three clustering algorithms were evaluated using the both criteria. For each input and each algorithm, we executed the experiments for a series of $k = 20, 40, ..., 200$. Then for each series we computed the average squared error, the average cost and the correlation coefficient of these squared errors and costs in this series.

From the results we find that the values of correlation are not high in all cases. It means that squared error cannot represent the quality of clustering in a large extent. Usually if the average error is higher, the corresponded cost is higher, but there exists exceptions, especially for BCC. The Binary Coding Clustering does not aim to reduce the squared error, but it can also achieve low cost and thereby high performance. It is because BCC algorithm does not use distance measure to cluster data units and it does not aim to

	GlobeDB			K-Means			BCC		
	Avg error	*Avg cost*	*Corr coef*	*Avg error*	*Avg cost*	*Corr coef*	*Avg error*	*Avg cost*	*Corr coef*
$n = 10^3$	0.103	24.583	0.563	0.073	23.881	0.971	0.282	22.828	0.590
$n = 10^4$	0.106	26.025	-0.030	0.068	24.884	0.926	0.294	23.535	-0.217
US	0.008	7.492	0.975	0.005	7.436	0.591	0.059	7.453	0.471
VU	0.013	7.933	0.966	0.004	6.922	0.991	0.023	6.761	0.927

Table 5.2: Comparison of squared error and cost.

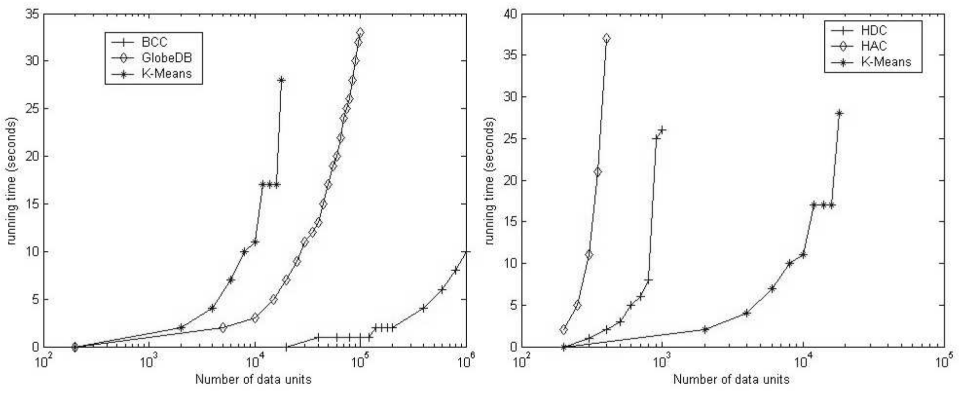


Figure 5.4: Comparison of running time of clustering algorithms

optimize its squared error. Therefore, we only can use cost function as the criterion to evaluate clustering algorithms.

5.4.3 Comparison of Scalability

A scalable clustering can complete clustering in acceptable time for any realistic number of data units. To evaluate the scalability, we fixed $m = 10$, $k = 100$ and adjusted n from 200 to 10^6. We recorded the elapsed time. Figure 5.4 presents the results. Obviously, the BCC algorithm is the fastest one. GlobeDB is also fast, but it is significant slower than BCC when there are over 10,000 of data units. The speed of K-Means is acceptable if there are less than 10,000 data units. The two hierarchical clustering algorithms are very inefficient, so they are not good candidates for GlobeDB. Therefore in the following experiments, we do not evaluate the hierarchical algorithms any more. If the number of data units is over 10,000, we do not evaluate K-Means algorithms either.

5.4.4 Comparison of Performance

This section compares the performance metrics of each clustering algorithm. First we applied the static Web trace to evaluate the performance metrics of each algorithm using a cost function. Then we applied synthetic data as input to verify the applicability of these clustering algorithms. If a clustering algorithm has better performance in both cases, we can conclude it outperforms the others.

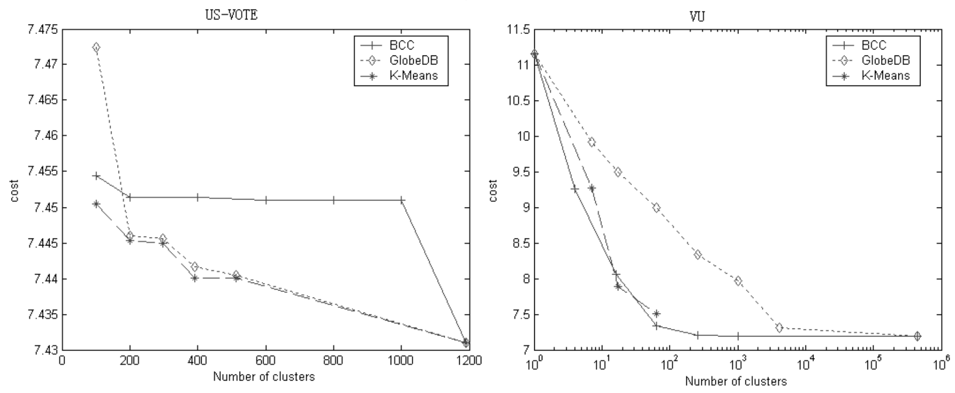


Figure 5.5: Comparison of clustering algorithms using Web traces with increase of k

We first used the both traces as input and compared the overall cost of different clustering algorithms with the increased number of clusters. Figure 5.5 illustrates the results. We can see that using US-VOTE, K-Means is the best one and BCC is the worst, while using VU, BCC is the best one and GlobeDB is the worst. The main difference of the two traces is n, the number of data units. US-VOTE only has around a thousand data units, but VU has about half a million data units.

To verify how far the n impacts the performance, we executed the same experiments using two synthetic data as input. One of the input data has 2,000 data units, and the other has 500,000 data units. Figure 5.6 presents the overall costs of the two groups. The results are similar like the trace-based simulation. BCC has the best performance in the case of large data units, but has the worst performance if n is low. GlobeDB clustering is almost the opposite of BCC, and K-Means is better than GlobeDB since it uses clusters generated by GlobeDB as its initial clusters.

In the above we only compared the algorithms by changing n and k. To compare the performance of clustering algorithms under different number of servers, m, we used synthetic data as input and fixed $k = 100$. We adjusted m from 10 to 100 and evaluated the overall cost. Figure 5.7 presents the results. BCC has better performance when m is low, but its performance turns worse with the increase of m. We think the underlying reason is that the precision of coding in BCC cannot increase anymore if m is high. Therefore, if the application has high m, BCC is not the best choice.

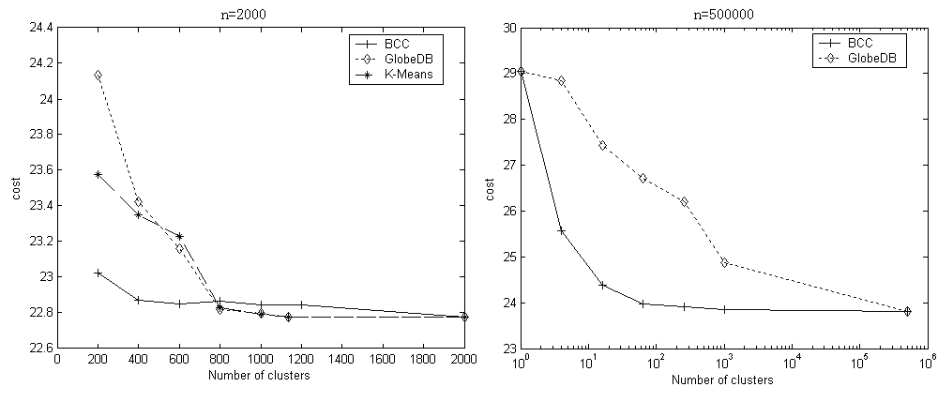


Figure 5.6: Comparison of clustering algorithms using synthetic data with increase of k

5.4.5 Selection of Number of Clusters

Until now, we have not chosen the proper number of clusters. As discussed before, the best clustering algorithm should achieve high performance with the lowest possible number of clusters. From the above experiment results, we can find that the overall cost value is close to the lowest value when the number of clusters is only 10% or even less than 1% of data units. For example, in Figure 5.5, the VU trace only needs around 200 clusters to reach almost the same cost as with 500,000 clusters. This observation indicates that the proper number of clusters can achieve similar cost as the fine-grained replication with the lowest value. Therefore the application administrator can plot the cost with the increase of k and select the point which cannot decrease the cost too much even when k increases. In addition, the proper k depends on the performance of placement algorithm since k is also an important factor to affect the placement performance.

5.4.6 Conclusion

From the above experiments, we can conclude as following.

- We suggest to choose correlation based distance as the similarity measure between data units.
- We suggest to adopt the cost function as the performance evaluation criterion.

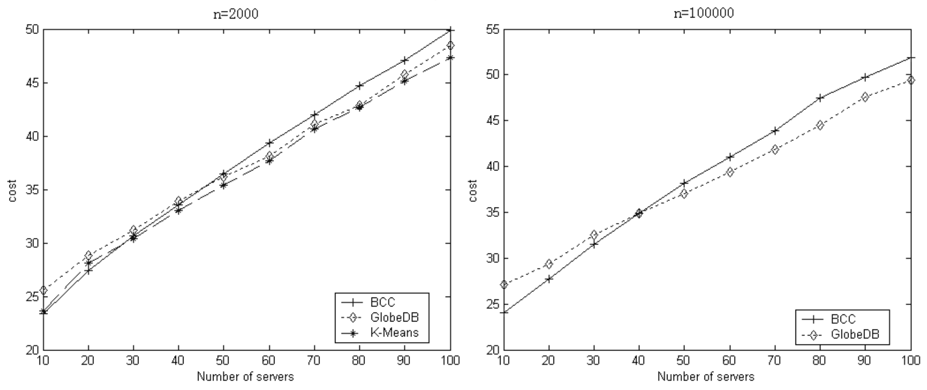


Figure 5.7: Comparison of clustering algorithms using VU trace with increase of m

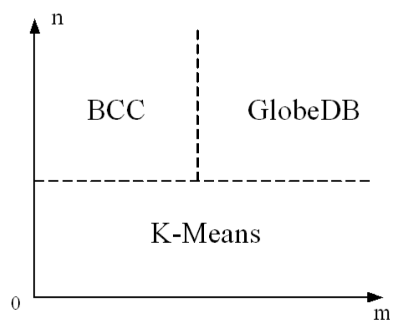


Figure 5.8: Suggested selection of clustering algorithm

- We suggest to use BCC algorithm to cluster data units if n is high and m is low, and use GlobeDB clustering if n is high and m is also high. Otherwise, K-Means is the best choice. Figure 5.8 illustrates this conclusion.
- We suggest to plot the curve of cost by increasing k, and select the k which cannot decrease the cost too much even if k is higher.

In addition to the above conclusion, the selection of clustering algorithm is also based on the system's computation capability. With increase of CPU frequency and widely using parallel or distributed computing techniques, we will be able to cluster more data units in less seconds. Therefore we suggest the system administrator tests the suitable threshold of k or m several times before he makes the decision.

5.5 Summary

In this chapter, we first detailed the GlobeDB simulator used in our experiments. Then we introduced the evaluation criteria. Finally we did some experiments on the simulator to evaluate the performance of clustering algorithms according to the different n, m and k. Finally we gave several suggestions about the selection of clustering algorithms under difference system situations.

Chapter 6

Reclustering

GlobeDB clusters data units based on their past access patterns. It is usually reasonable to predict the future according to the recent past since normally popularity does not change very quickly. But the access patterns will change as time goes on, so the old clusters may not be accurate anymore. Consequently, the quality of clusters will be impacted. That is why we need to recluster the data units.

We cannot perform reclustering too often because of the high cost of clustering and placing. But, on the other hand, if reclustering is too infrequent, the change of access patterns may degrade the system performance significantly. Therefore our goal is to find a balance between them. We want to maintain the performance of the system without spending too high cost in reclustering.

One reclustering strategy is to periodically cluster data units using the access pattern of the latest period. To apply this strategy we need to find an optional reclustering period.

In this chapter, we first compare the performance of non-reclustering and reclustering strategies, then we look for the proper period of reclustering.

6.1 Reclustering versus Non-reclustering

We used the GlobeDB simulator to execute the experiment of reclustering. To compare the performance changes between reclustering and non-reclustering, we divided a trace into many periods according to the time stamp of each record in the trace. A period can be an hour, a day or several days. According to the suggestion of clustering selection, we set $m = 10$

	Non-reclustering	Reclustering
US-VOTE	11.129	9.438
VU	10.470	9.784

Table 6.1: Average costs of non-reclustering and reclustering strategies.

and $k = 100$ for the US-VOTE trace and used K-Means as its clustering algorithm; we set $m = 10$ and $k = 1000$ for the VU trace and used BCC as its clustering algorithm.

The reclustering strategy works as following. In the first period, we only collect the access patterns. In the second period, we cluster all the data units using the access patterns from the previous period, and place the created clusters on edge servers according to the placement algorithm. Then we evaluate the system performance of clustering using the access patterns of this period. In the third period, we recluster data units and replace clusters using the access patterns of the previous period, then evaluate the system performance of reclustering using the access patterns of current period. All the next period will repeat the action as the third period. Finally we record the evaluated costs at the end of each period and compare the results.

For non-reclustering strategy, we only cluster and replicate data units at the second period according to the access patterns in the first period, then apply the clustering and placement configuration to any new coming data units in all the following periods. At the end of each period, the evaluation module calculates and outputs the overall cost in this period.

Figures 6.1 presents the results of above experiments on US-VOTE trace and VU trace respectively. The reclustering period is set to one day. It is clear that the reclustering strategy can achieve better performance than non-reclustering for both traces. To make it clear, we plotted the difference between the cost of non-reclustering and reclustering for US-VOTE in Figure 6.2. A sliding average line is used in this figure to present the tendency of the difference. We can see that most values in this figure are positive[1], which means that reclustering performs better than non-reclustering. Table 6.1 gives the average costs of both strategies.

[1]There are also a few negative points. Actually, the exception is caused by gaps in US-VOTE trace. These gaps will change servers' load in the pervious period, and consequently effects the placement configuration and the system performance.

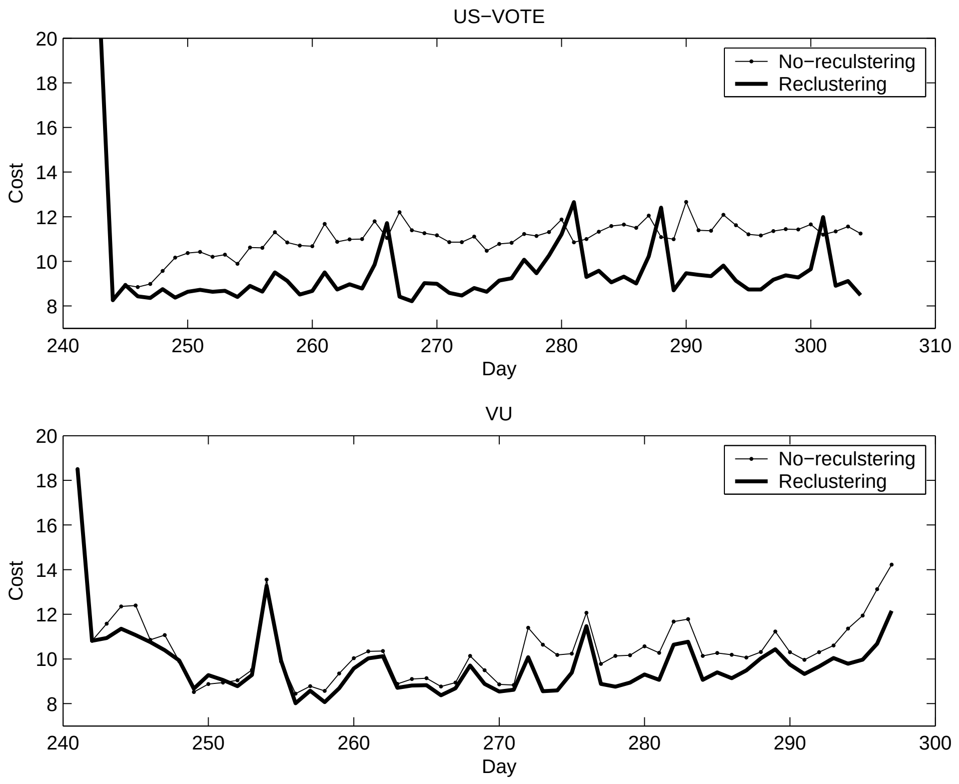


Figure 6.1: Non-reclustering vs. Reclustering

The performance gain from reclustering on VU trace is lower than from the US-VOTE trace. We think it is because US-VOTE's access patterns and loads change frequently every day, while VU's access patterns are relatively stable (see Figure 5.3). Reclustering is more efficient for Web applications whose popularity changes easily, since frequently reclustering and replacement can react to the changes quickly.

6.2 How often to recluster

The next issue is to select the length of reclustering period. To study this problem, we redid the reclustering experiments, but changed the reclustering period as one day, two days and four days respectively. Figure 6.3 illustrates the results, and Table 6.2 gives the average costs of each reclustering on both traces.

The results indicate that the best reclustering period is one day for US-

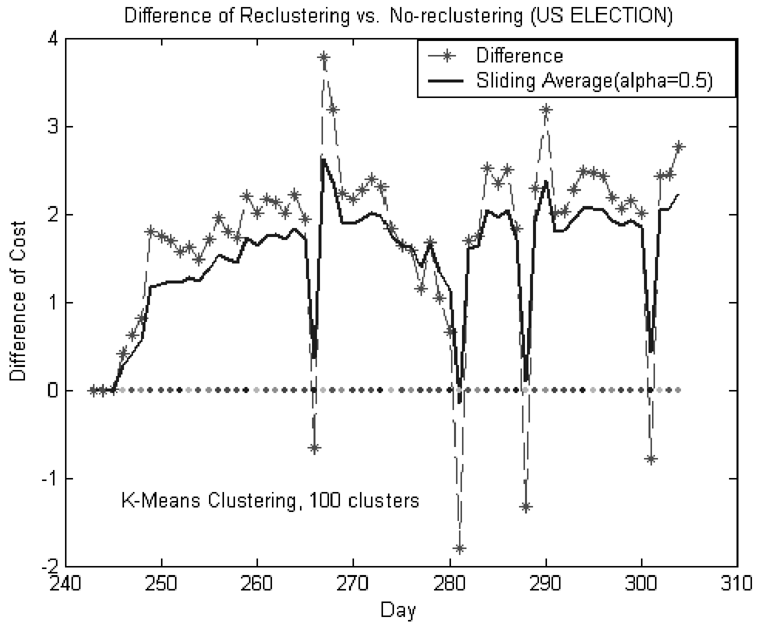


Figure 6.2: Difference of Clustering and Reclustering on US-VOTE trace

VOTE, and two days for VU. It is interesting that if the reclustering period increases, the cost of US-VOTE increases, but the performance of VU first decreases and then increases. The underlying reason, we believe, is also that the US-VOTE's popularity changes faster than VU's. A frequently reclustering strategy is more suitable for an "unstable" Web application, whose access patterns change easily, because this strategy can react to the variance quickly. However, a short reclustering period is not proper for a "stable" application, since a long-time popularity may be more accurate for predicting the near future. Therefore, it does not make sense to perform reclustering as often as possible, even without considering the cost of placement.

Although a short reclustering period is not suitable for any application, a long period can decrease the performance gain from reclustering either.

Period	1 day	2-day	4-day
US-VOTE	9.438	9.610	9.922
VU	9.784	9.549	9.6490

Table 6.2: Average costs of reclustering with different period.

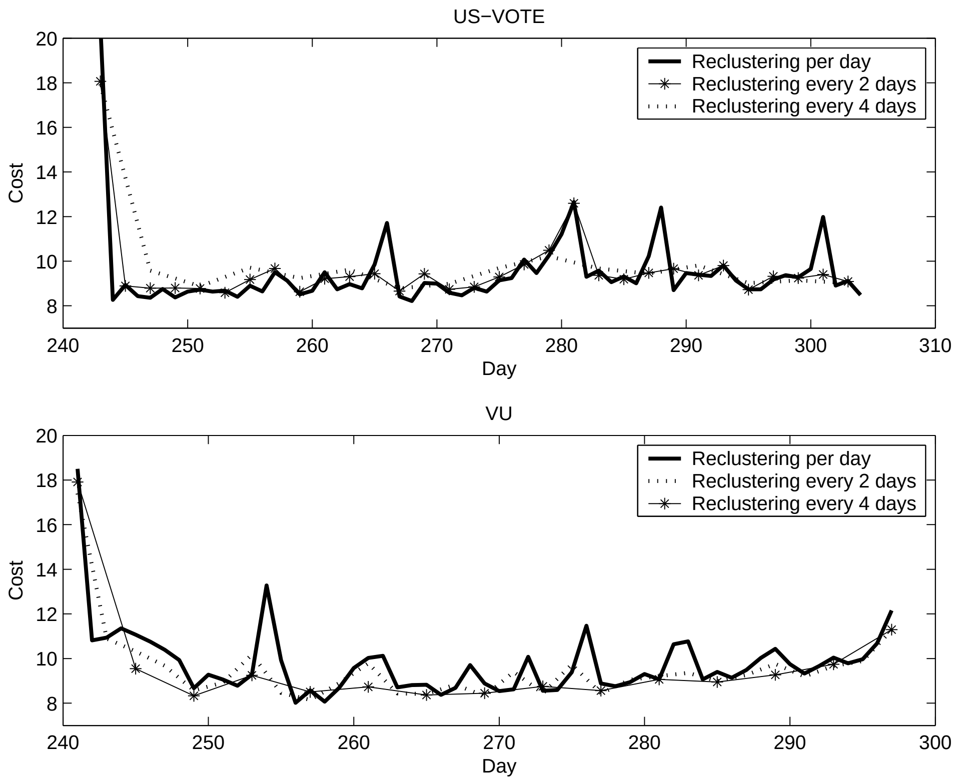


Figure 6.3: Reclustering on different periods

To find the best period, we can execute reclustering by different periods and plot the average costs. Then choose the period which achieves the least cost as the best one for reclustering.

6.3 Summary and Discussion

In this chapter we compared the performance gain from reclustering and non-reclustering strategies. The experiment results show that reclustering can effectively improve the system's performance, and the proper reclustering period is based on the stability of application data.

There are some issues on reclustering left for future study. Although periodically reclustering is simple, it does not care about the changes in access patterns. If the popularity does not change very much from the last period and the old clustering and placement configuration is still efficient, then we do not need reclustering yet. On the other hand, if for example a flash crowd happens, the periodical strategy cannot respond in time if the period will not finish soon. So we need some strategy which is flexible for both cases.

In the above experiments, we recluster all data units and replace all clusters in each period. Maybe in some cases only replacement is efficient enough. In addition, K-Means can use the clusters created in the previous period as its initial state. BCC can use the previous placement configuration to generate its placement pattern. We think these strategies can improve the efficiency of reclustering to some extent.

Chapter 7

Conclusion

This thesis first presented the data replication problem in CDNs and introduced a solution using partial replication. Then we introduced GlobeDB, an autonomic data replicating middleware that employs partial replication for Web-based applications. GlobeDB adopts data clustering techniques to overcome the overhead of fine-grained replication. But the original clustering algorithm in GlobeDB was never really evaluated. This thesis systematically studied the issues of data clustering in the context of GlobeDB.

The most important issue is to select a proper clustering algorithm for the system. This issue includes the following subproblems: 1) what are the current existing data clustering algorithms; 2) what is the best performance evaluation criterion, 3) how can we evaluate the quality of clustering; and 4) which one is the best algorithm for GlobeDB and how many data clusters we need.

To answer the first problem, we introduced several classical clustering algorithm in the literature, and concepts such as similarity measures. In addition, we improved the current GlobeDB and K-Means algorithms, and developed a new algorithm, Binary Coding Clustering, for the system. The revised algorithms and new algorithm can achieve better performance than the original ones.

To answer the second problem, we introduced squared error, a traditional criterion for clustering, and a cost function approach used in GlobeDB. We found the cost function can represents the system's performance realistically, but the squared error cannot in most cases. Therefore we decided to use the cost function to evaluate the quality of clustering.

To solve the third problem, we created a simulator for GlobeDB. It can accept Web traces and synthetic data as input, mimic the behavior of the system, and evaluate the cost of each experiment. We used two different types of Web traces to test the performance of different clustering algorithms, then adopted synthetic data to verify their applicability. Therefore we can widely study the quality of algorithms.

Finally, we concluded based on extensive experiments. Three clustering algorithms are suggested to use in the system, and each of them has a proper applicable range based on the situation of the system. We also introduced a way to find out the suitable number of clusters for the system. Properly choosing a clustering algorithm and number of clusters can improve the system's performance to a large extent.

Another issue studied in this thesis is reclustering. We need to recluster the data units to maintain high performance of the system. From the experiments, we found the period of reclustering should be set as a suitable value to achieve the best performance gain. The length of period is based on the stability of application data's popularity. The more stable the data, the longer the period should be. Currently we only adopt periodically strategy, which is simple but insensitive to events such as flash crowd. Future work should explore improved reclustering strategies.

GlobeDB will be implemented in Globule project [25]. We hope it can contribute to worldwide Web applications, and therefore provide Web clients better and faster Internet experience.

Acknowledgements

Personally, I always like reading the acknowledgements in scientific paper or book. They help to add a human side to the technical content. Now it is the time for me to enjoy writing the acknowledgements and express my appreciation to lots of people.

Pursuing the Master degree in Vrije Universiteit is an adventurous journey. There are many obstacles in the way, many ups, many downs, and a great deal of distractions too. Although it leads to an individual accomplishment, there are many people that help out along the way. Whether they help out with the research and writing, or whether they provide moral support, or the much needed distractions, all of them deserve thanks and credit.

First and foremost, I do appreciate both my thesis supervisors Guillaume Pierre and Swaminathan Sivasubramanian. Thank you for your constructive guidance and enduring patience. What I learned from you in the half year will be my precious treasure in my whole life.

Next, I would like to thank to my teachers Maarten van Steen, Zhisheng Huang, Bruno Crispo and Michal Szymaniak. I feel lucky to meet Maarten, who opened a gate leading to the palace of computer science for me. I also feel luck to meet Zhisheng, who provided me lots of help not only in study but also in life. Bruno offered me lots of guidelines in scientific writing and Michal performed some experiments for my thesis work. Thank you all!

I am grateful to my dear friends, Mengxiao Wu, Chenyan Xu, Taoying Yuan, Wenfeng Wang and Dakai Liu. My life abroad would be lonely and helpless without these Chinese friends, especially Mengxiao and Chenyan, my classmates, roommates and closest friends in the two years. The days and nights spent with you will keep on my mind forever.

Last, but certainly not least, I wish to thank my parents and my girlfriend. The main role of the family is to keep on asking when I can complete the thesis. Now I can happily say that I have done it. I would specifically like to thank my Mom, for everything past, present, and no doubt future, my Dad, for always keeping in touch, and always being close despite the great distances. To my lovely lady Bao Fang, I do not know how to express my appreciation for the great happiness, pleasure and courage you ever filled to me. Thank you for your great influence to my life. You are the people I will always appreciate.

Bibliography

[1] Akamai Inc, *Akamai Technology Overview*, `http://www.akamai.com/en/html/technology/overview.html`.

[2] Digital Island, `http://www.digitalisland.com`.

[3] A. K. Jain, M. N. Murty and P. J. Flynn, *Data Clustering: A Review*, ACM Computing Survey, Vol. 31, No. 3, September 1999.

[4] G. Pierre, M. van Steen and A. S. Tanenbaum, *Dynamically Selecting Optimal Distribution Strategies for Web Documents*, IEEE Transactions on Computers vol. 51(6), June 2002.

[5] S. Sivasubramanian, G. Alonso, G. Pierre and M. van Steen, *GlobeDB: Autonomic Data Replication for Web Applications*, Proc. 14th International World-Wide Web conference (WWW 2005), Chiba, Japan, May 2005.

[6] S. Sivasubramanian, G. Pierre and M. van Steen, *Autonomic Data Placement Strategies for Update-intensive Web Applications*, Proc. of the International AAA-IDEA Workshop, June 2005.

[7] B. Krishnamurthy, C. Wills and Y. Zhang, *On the Use and Performance of Content Distribution Networks*, ACM SIGCOM WIMW'01.

[8] G. A. F. Seber, *Multivariate Observations*, Jonh Wiley & Sons, 1983.

[9] A. D. Gordon, *Classification*, Chapman and Hall, 1981.

[10] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz, *Clustering web content for efficient replication*, In Proceedings of 10th IEEE International Conference on Network Protocols (ICNP02), 2002.

[11] C. Bornhvd, M. Altinel, C. Mohan, H. Pirahesh, and B. Reinwald. *Adaptive database caching with DBCache. Data Engineering*, 27(2):11-18, June 2004.

[12] P. Larson, J. Goldstein, H. Guo, and J. Zhou. *MTCache: Mid-tier database caching for SQL server*, Data Engineering, 27(2):27-33, June 2004.

[13] J. Challenger, P. Dantzig, and K. Witting. *A fragment-based approach for efficiently creating dynamic web content*, in the ACM Transactions on Internet Technology, 2004.

[14] F. Douglis, A. Haro, M. Rabinovich. *HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching*, USENIX Symposium on Internetworking Technologies and Systems, Dec. 1997.

[15] W. Li, W. Hsiung, O. Po, K. Hino, K. S. Candan, D. Agrawal. *Challenges and Practices in Deploying Web Acceleration Solutions for Distributed Enterprise Systems*, WWW2004, May 2004, New York, USA.

[16] A. Davis, J. Parikh, W. E. Weihl, *EdgeComputing: Extending Enterprise Applications to the Edge of the Internet*, WWW2004, May 2004, New York, USA.

[17] D. Margulius. *Apps on the edge*. InfoWorld, 24(21), May 2002. `http://www.infoworld.com/articles/fe/xml/02/05/27/020527feedgetci.html`

[18] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz. *Clustering web content for efficient replication*. In Proceedings of 10th IEEE International Conference on Network Protocols (ICNP02), 2002.

[19] S. Sivasubramanian, G. Pierre, and M. van Steen. *A case for dynamic selection of replication and caching strategies*. In Proceedings of the Eighth International Workshop Web Content Caching and Distribution, Hawthorne, NY, USA, Sept. 2003.

[20] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. *Application specific data replication for edge services*. In Proceedings of the Twelfth International World Wide Web conference, May 2003.

[21] C. Cunha, A. Bestavros, and M. Crovella. *Characteristics of WWW client-based traces*. Technical Report TR-95-010, Boston University, Computer Science Dept., USA, April 1995.

[22] A. S. Tanenbaum and M. van Steen, *Distributed Systems, Principles and Paradigms*, Prentice-Hall, 2002.

[23] `http://www.amazon.com`

[24] `http://www.maxmind.com`

[25] `http://www.globule.org`