CRANFIELD UNIVERSITY

ChenHan Liao

Transaction-Filtering Data Mining and A Predictive Model for Intelligent Data Management

Department of Applied Mathematic and Computing
School of Engineering
Cranfield University

PhD THESIS

CRANFIELD UNIVERSITY

School of Engineering

Department of Applied Mathematic and Computing

PhD THESIS

Academic Year 2005-2008

By
ChenHan Liao

Transaction-Filtering Data Mining and A Predictive Model for Intelligent Data Management

Supervisor: Prof. Frank ZhiGang Wang

November 2008

This thesis is submitted in partial fulfilment of the requirements for the Degree of Doctor of Philosophy

Abstract

This thesis, first of all, proposes a new data mining paradigm (transaction-filtering association rule mining) addressing a time consumption issue caused by the repeated scans of original transaction databases in conventional associate rule mining algorithms. An in-memory transaction filter is designed to discard those infrequent items in the pruning steps. This filter is a data structure to be updated at the end of each iteration. The results based on an IBM benchmark show that an execution time reduction of 10% - 19% is achieved compared with the base case.

Next, a data mining-based predictive model is then established contributing to intelligent data management within the context of Centre for Grid Computing. The capability of discovering unseen rules, patterns and correlations enables data mining techniques favourable in areas where massive amounts of data are generated. The past behaviours of two typical scenarios (network file systems and Data Grids) have been analyzed to build the model. The future popularity of files can be forecasted with an accuracy of 90% by deploying the above predictor based on the given real system traces. A further step towards intelligent policy design is achieved by analyzing the prediction results of files' future popularity. The real system trace-based simulations have shown improvements of 2-4 times in terms of data response time in network file system scenario and 24% mean job time reduction in Data Grids compared with conventional cases.

Acknowledgment

I wish to give my sincerely thankfulness to my supervisor Professor Frank Zhigang Wang. I thank him for his constant enlightenment, support and selfless delivering of his knowledge and experience. As a supervisor, he not only helps me to build my researching skills but also leads me to my research career. My deepest thanks are to Professor Chris Thompson and Dr. Stephen Hobbs for being my reviewing panels of last three years and their valuable advices.

I would also like to thank Dr. Sining Wu, Dr. YuHui Deng and Dr. Vineet Khare for their continuous help on every aspect. I thank them for their technical support, skills and valuable advices. It was an enjoyable time that we have been working together since my research started.

My special thanks and appreciation are to Prof. Yike Guo (Imperial College), Dr. Na Helian (University of Hertfordshire) and Dr. Peter Sherar (Cranfield University) for being examiners of my thesis.

Last but not least, my grateful appreciation goes to my parents and my wife for their countless help, encouragement and confidence. Without them, I could not reach my goals and achievements. I hope my success will make them proud of me as I am proud of them for holding their love.

Contents

List of Figures

List of Tables

List of Equations

Chapter 1 Introduction

Recently, countless successful data mining applications emerged in finance, marketing, insurance and banking businesses. The term-“data mining” is easily connected with these traditional application areas. However, the needs of knowledge from large amounts of data universally exist in this “data rich but knowledge poor” world. Unfortunately, mining useful knowledge for computer system engineering has not been given enough attention compared with other data mining applications. This thesis proposes a universal method to speedup database scanning for general data mining applications. Next, it addresses file and storage system issues from the view of data mining technology and reviews the related works given to this area. An incremental decision tree based predictive model is established to improve the efficiency of file system caching and achieve optimal disk layout. The deployed model in a flash memory oriented storage system featuring server caching environments will be discussed. The model is also extended to a Data Grid environment where a predictive file replication strategy is proposed and examined. In this chapter, the data mining technology from its general concepts, tools, process and application areas will be discussed.

1.1 Data mining Concepts and tools

“That's the thing with magic. You've got to know it's still here, all around us, or it just stays invisible for you.”---------Charles de Lint [1]. This is a world flooded with data but staved for knowledge. Although sheer volume of data churned out every second in our daily life, we know little about them. Why did they happen? When will they happen again? Data mining technology answers these questions by inspecting the internal relations of data. Data mining is also called knowledge discovery in database. It is an automatic mechanism deriving novel and previously unknown knowledge, patterns and rules from large amounts of data. The obtained knowledge, patterns and rules are considered as the abstractions of the input data. Data mining was originally applied for cross basket analysis to identify the purchase patterns from customers. Nowadays, it is broadly applied to many industrial areas such as finance, marketing, life science and bioinformatics.

With the fast development of media, database and communication technologies, the cost-effective collection and storage of massive amounts of data become universally. Digging out internal relations and forecasting future trends of these data can help policy makers to better understand their data. Data mining technology is such a combination of traditional science methodologies, which involves statistics, database, data structure, software engineering and information security.

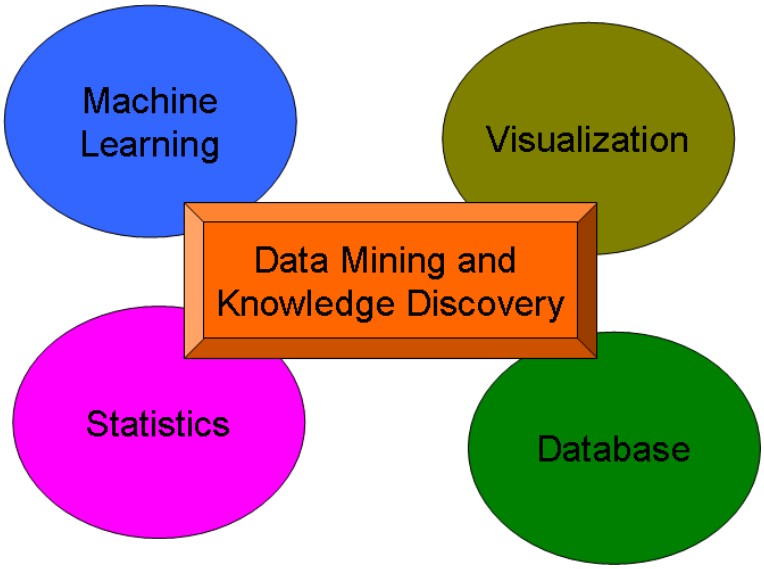


Figure 1.1 Science methodologies involved in DM [2]

Data mining is a fuzzy word that is always agued along with statistics. When one mentions data mining, it normally refers a full procedure of data analysis, which includes data cleaning, preparation and visualization. Different from statistics, data mining deal with the problem how to generate the results faster or even in real-time. Furthermore, rather than database queries and statistical analysis, data mining inspects data in a different way. For example, statistical analysis tools employ verification-based approaches to test whether a hypothesis exists or not. One might hypothesize that customers who buy milk, also hold a good chance of buying bread. However, such statistical analysis only takes into effects with a given assumption, which sometimes is narrowed down from user's creativity. In contrast, data mining deals with multidimensional data and detects unique and frequent data relationships without any previous knowledge of the data.

To solve a specific data mining problem, computers are responsible to iteratively process historic data via running dedicated data mining tools. The tools implement one or multiple

data mining functionalities and have universal interfaces supporting different kinds of data warehouses. Due to the variety of data, no single technique delivers the best results to every kinds of dataset. Therefore, data mining tools are also required to employ multiple techniques and if necessary to form a combination to discover from various complex data.

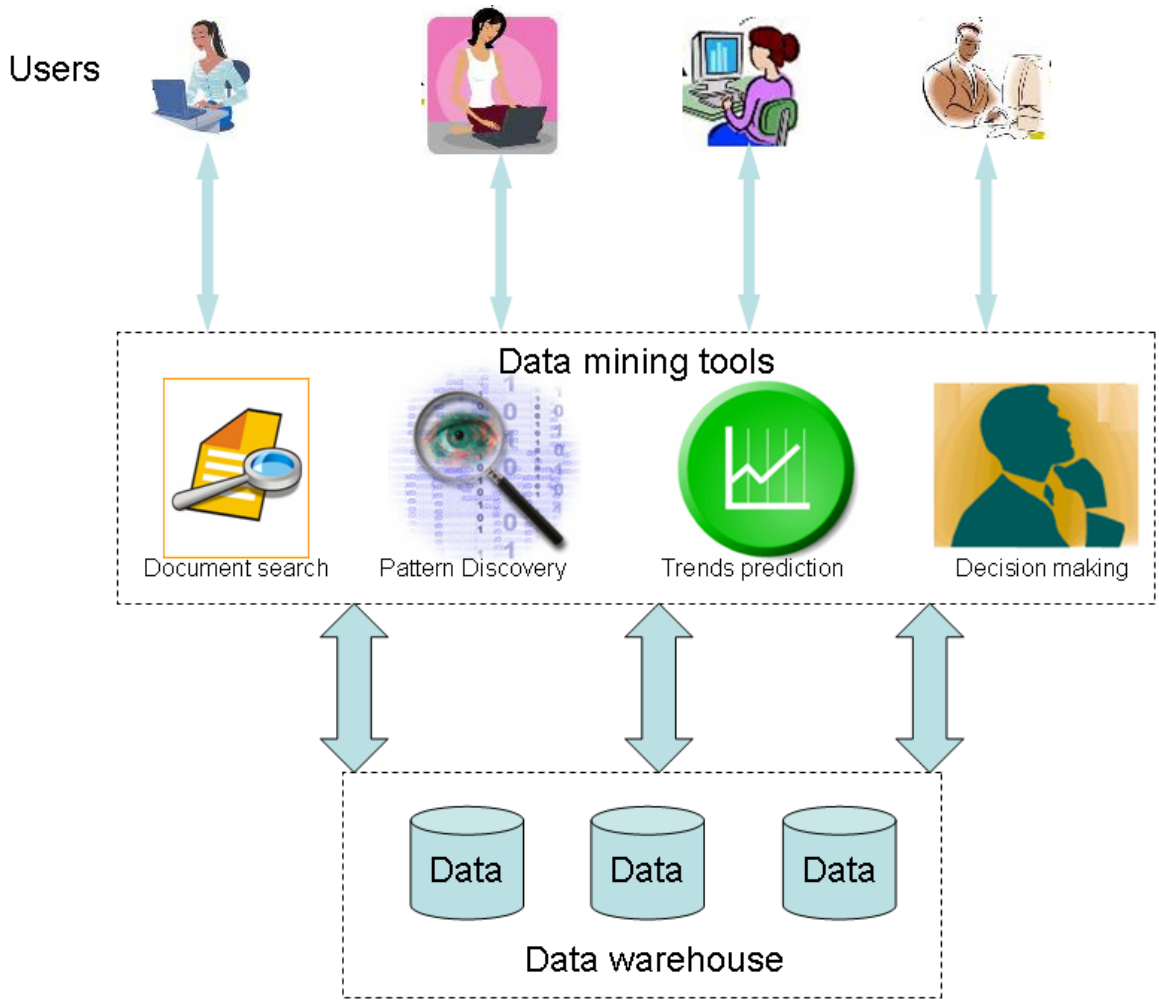


Figure 1.2 Multifunctional data mining tools, users and data warehouses

1.2 Data Mining Process

To enable data mining task repeatable and reliable to people with little data mining background, a standard data mining process is essential. In 1996, a project named Cross Industry Standard Process for Data Mining (CRISP-DM) was founded by European commission and contributed by over 200 data mining leading research communities, vendors, system suppliers and end users. The purpose of this project is to develop an

industry- and tool-neutral data mining process model. So far, CRIPS 1.0 has been released and being continuously refined. This standard process aims at making large data mining projects faster, cheaper, repeatable, more reliable and more manageable. The current CRISP model provides data mining practitioners an overall image of a data mining project. The model contains the essential steps of a project, relative tasks, and relationships between the tasks.

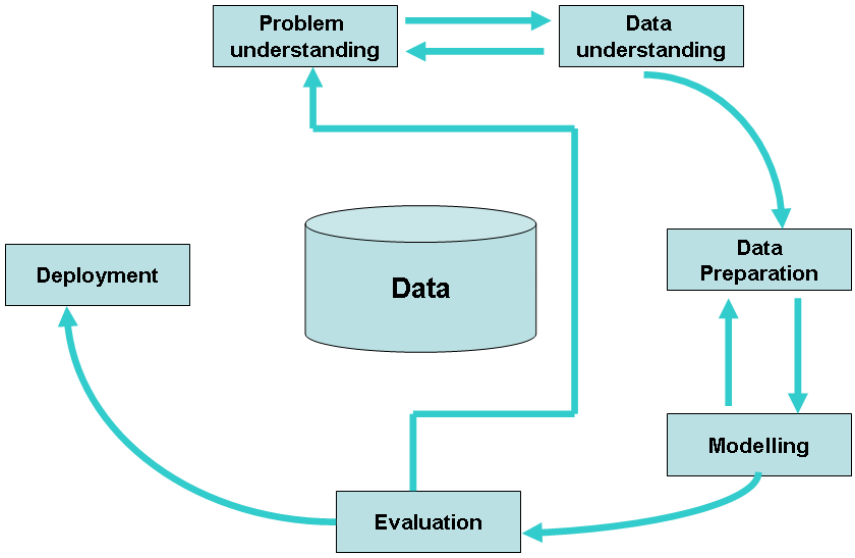


Figure 1.3 The CRISP-DM Model [3]

■Problem definition

A typical data mining application begins at the comprehension of the problem. Projects participants including data mining modeler, business experts, and software developers work closely to define the project objectives and identify the requirements of the users. The project objective is also converted into problem definition from data mining perspectives.

■ Data exploration

The raw data are collected, normalized and explored by domain specialists, who understand the meaning of the raw data. In this phase, traditional data analysis and statistics tools are used to explore the data.

■ Data preparation

Before the data modelling phase, data are collected, cleaned and reformatted since data mining models normally accept certain formats. Some of attributes of data are added such as average values and maximal or minimal values. In this phase, data are processed

multiple times to select tables, records, and attributes for modeling tools. However, this does not change the meanings and contents of the data.

■ **Modelling**

In this phase, various data mining models are assessed to decide which model is the most appropriate one for a defined problem. Domain experts and data mining modellers work together in this phase as some of mining functions require specific data types and formats that need to adjusted or reformatted. In addition, modelling and evaluation are coupled together in this phase, which can be repeated several times to modify the parameters until high quality models have been built.

■ **Evaluation**

During this phase, data mining models are evaluated through specific validation methods such as K-fold cross-validation and Leave one out validation methods. If the accuracy of the model does not satisfy users' expectation, the model will need to be modified or adjusted until the optimal values are produced. At the end of the evaluation phase, the data mining modellers need to work with domain experts to translate the data mining results into plain rules, patterns and policies.

■ **Deployment**

Data mining modellers deploy the mining models by exporting the data mining results into database tables or into other applications, for example, spreadsheets. Such integrations enable people with little data mining technology to apply data mining model to solve real problems such as decision making and policy design.

1.3 Data Mining in Various Application Areas

More and more organizations and companies tend to employ data mining techniques to handle their massive amounts of data. Besides those traditional applications such as customer relationship management (CRM) and credit scoring, some new areas such as bioinformatics, gambling and manufacturing are setting their feet in data mining. Successful Data Mining Applications [4] was polled by 421 data mining practitioners and researchers. It represents industries/fields where practitioners and researchers had successfully applied data mining in past three years. The figure below shows the distribution of successful data mining applications from these data mining practitioners and

researchers.

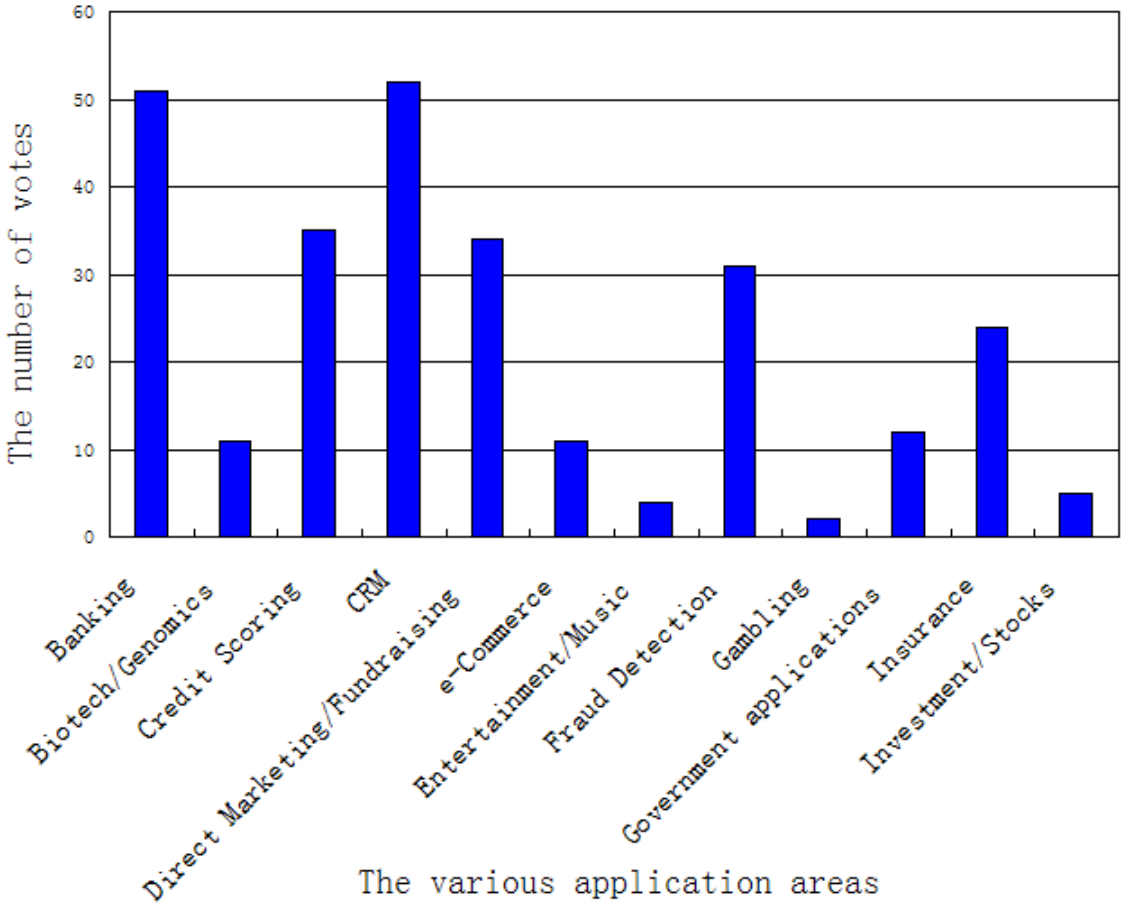


Figure 1.4 (a) the diagram shows the successful data mining application in various areas polled by data mining practitioners [4]

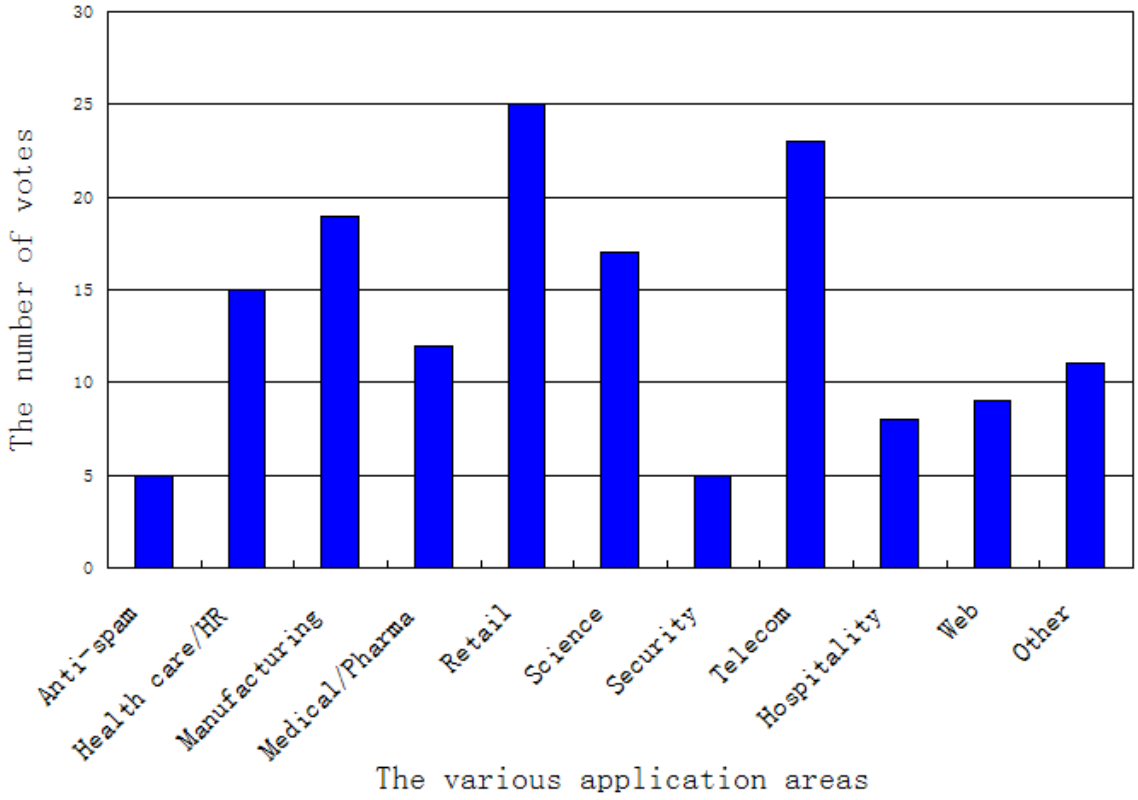


Figure 1.4 (b) the diagram shows the successful data mining application in various areas polled by data mining practitioners [4]

As we can see from the above figures, areas such as banking, credit scoring, credit fraud detection and retail represent the most successful application areas applying data mining techniques. In the following paragraphs, we discuss three major areas where data mining techniques are frequently adopted.

■Banking and Finance

Data mining techniques contribute to banking and finance by identifying various types of knowledge from transaction data that are impossible to be quickly identified by manpower since the volume of data may be too large or its' generation speed is too quick to be screened. The data mining tools are employed to find the periodicity and sequences of customers' transaction behaviours. The extracted results can help bankers to gain and keep higher profits from different customer bases. Data mining techniques are also applied to business intelligence such as identifying various classes of customers to design class based products and prices that may gain higher revenue.

■Financial Market Risk Measurement

Financial market risk management is related to the potential risk factors such as government policies, latest technologies and interest rates, which can influence the trends of financial markets. Finance experts are interested in the data mining models that are able to measure the correlations between financial instruments (e.g. price) and underlying risk factors. Most of the data mining models for evaluating financial market risks focus on single risk factor to analyze the impact to the overall market. Due to the fact that the data are normally not public available, the models can only be built by using various data mining techniques on the proprietary portfolio data.

■Credit Scoring

Credit scoring is the most important procedure in the process of commercial lending. To be able to decide whether or not to lend to the borrowers, accurate credit scoring must be made. Credit scoring is the process of making assignment of customers to risk levels. In essence, a typical credit scoring process is the problem of modelling credit values based on a set of given characteristics of the borrowers. There are three different methods to model credit risk: accounting analytic approaches, statistical prediction and theoretic approaches.

1.4 The Needs of Data Mining for File and Storage Systems

With the increasing scale of the components in computer systems including hardware and software, data generated from file and storage systems are no longer possible to be examined by system administrators or maintainers. Unfortunately, the generated data still continuously heap up and stay undiscovered, where some useful information may exist. For example, one of the most widely simulated disk level traces “cello” [5], which were generated from “cello” servers in Hewlett-Packard laboratory, has shown 10 times increase of daily I/Os. The table below shows the dramatically increased volume of data generated from cello traces.

	Total Size	Period	Size/day
Cello-92	603M	62days	10.05M/day
Cello-96	5028M	81days	62.1M/day
Cello-99	39014M	351days	111M/day

Table 1.1 The volume of Cello traces

However, the situation in file systems is even worse than in disk level. With the widespread

server applications such as web server, data hosting server, mail server and so on, intensive data requests inevitably occur in file systems. A mail system called CAMPUS [6] that is responsible for E-mail services in Harvard University campus can easily produce 1,604,848 requests/hour in average, which is equivalent of 3.7GB trace data generated per day, not to mention those large systems of commercial E-mail service providers.

Unfortunately, massive amounts of raw data representing user and system behaviours are churned out but often ignored until system failures are detected, which is the very original purpose of establishing system traces. Traditionally, traces are examined by system administrators to identify the causes of system failures. In one hand, it has been investigated that the cost of system administration and maintenance is 4-8 times than storage hardware and software cost [7], [8], [9]. On the other hand, even if the cost is affordable to allow system administrators to analyze traces, some useful hidden information can hardly be extracted by human eyes. For example, correlations among user behaviours may exist in hundreds of thousands data requests that do not pose clear patterns to us. In order to peel unnecessary noisy information, some specific algorithms and models need to be established to save expensive manual administration cost.

Despite saving system administration and maintenance cost, using hints derived from massive amounts of trace data to improve system performance is feasible. As system traces is designed to capture data requests generated from human activities, hence, the derived knowledge reflects the realistic rules, patterns and trends of user behaviours. Based on these hints, system designers can optimize various aspects of file and storage systems by introducing adaptive system policies featuring intelligent data management and humanized system design.

Summary

In this chapter, we have discussed data mining techniques and tools in concept level as well as relative application areas. Generally, data mining acts as an intelligent component interacts between users and data to deliver unseen information such as rules, patterns and correlations. Users can then utilize the discovered information to make correct policies and

decisions in order to achieve higher revenue or better management. A typical data mining problem normally follows the standard mining process: CRISP-DM defined and accepted by major data mining research community and practitioners. This model defines six steps involved in a typical data mining problem as well as the relative tasks and outputs of each steps.

The traditional areas such as customer's banking transaction behaviour analysis, financial market risk measurement and credit scoring are commonly accepted as the most successful data mining applications. Besides these, section 4 represents the demands of automatic data mining techniques for file and storage systems. Due to the exploration of data and the increasing scale of file and storage systems, mining file and storage systems not only helps to improve system performance but also saves expensive system administration and maintenance cost.

Although data mining has made countless succeeds in many application areas, the data mining techniques adopted may vary according to the specific applications. In the next chapter, we first examine the four most widely used data mining techniques: association rule mining, classification, clustering and frequent sequence mining, then a new transaction filtering technique for speeding up database scanning will be discussed.

Chapter 2 A New Data Mining Paradigm: Transaction Filtering Association Rule Mining

In general, data mining involves a number of sub-areas shown in Figure 2.1. Applying appropriate data mining techniques for different applications is essential as each of data mining methodologies has their own functionalities. In this chapter, we discuss four data mining techniques that had been successfully applied to file and storage systems.

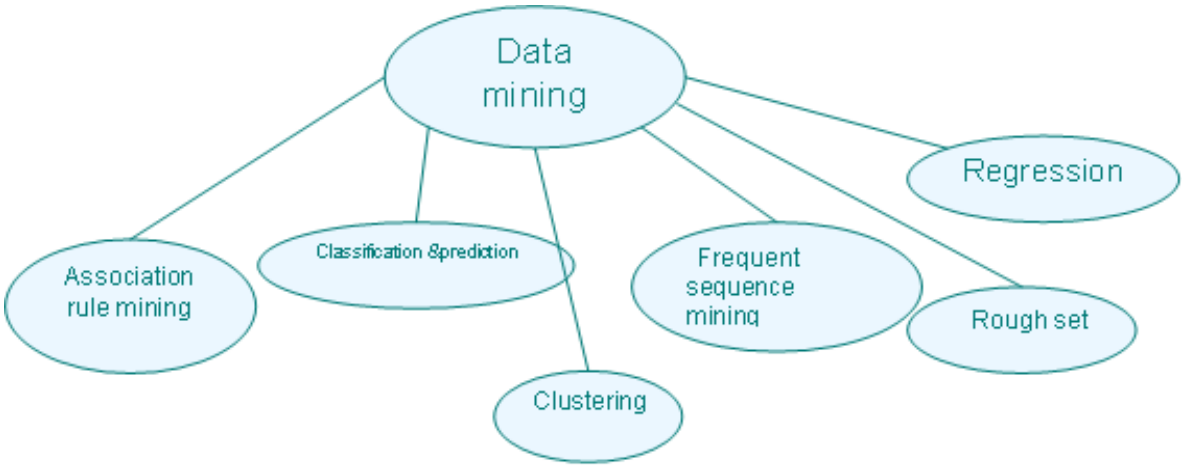


Figure 2.1 The most widely adopted data mining methodologies

2.1 Association Rule Mining (ARM)

Aassociation rule mining in transaction databases have been demonstrated to be useful and technically feasible in several application areas, particularly in retail sales. With the fast development of data storage and distribution, more and more commercial and non-commercial organizations tend to mine association rules from their data warehouses. The very original application was to discover the purchase patterns from customers for retails such as supermarkets. Some products always imply stable associations in most of purchase transactions of customers. For example, a purchase transaction containing

tomatoes and lettuce in a food store most likely contains something else related such as mayonnaise or salad dressing.

The following is a formal statement of ARM problem for transaction databases. Assuming I= $\{i_1, i_2, \ldots . i_m\}$ is a set of items, while D is a transaction database containing a number database transactions where each transaction T has a number of items so that T$\subseteq$I. Every transaction in database D is associated with an ID number called TID. Assuming J is a set of items. A transaction T_D is considered to contain J if and only if J$\subseteq T_D$. If another set of items V can be implied by J (J$\Rightarrow$V), where J$\subset$I, V$\subset$I, and J$\cap$V= {}, we can call J$\Rightarrow$V is an association rule. In the transaction database D, the association rule J$\Rightarrow$V has a support value s, which is the occurrence percentage of the transactions containing both J and V. We can also convert s into the probability P (J$\cup$V). That is, support (J$\Rightarrow$V) = P (J$\cup$V) [2].

The problem of mining association rules is to find out all the implications that exceed a predetermined threshold referred as Support. For a given dataset D as below, we can conclude all the implications (frequent itemsets) in accordance of their support.

Transaction ID	Items
T1000	**A,C,D**
T2000	**B,C,E**
T3000	**A,B,C,E**
T4000	**B,E**

Table 2.1 Example transaction database

Suppose that we have a Support set to 50%, the minimum support count is the number of transactions multiplied with the Support, which is 4*50%=2. It means that the occurrences of the frequent itemsets must exceed or equal to 2.

Support count	Itemsets
2	**AC,BC,BE,BCE**
3	**B,C,BE,E**
4	**None**

Table 2.2 Frequent itemsets with the given threshold

So far, many ARM algorithms have been proposed to make the mining procedure faster, cost effective and more reliable. The one of the most original ARM algorithms is the Apriori [10], which utilizes an Apriori property: if an itemset is frequent, then all its subsets must be frequent. Consequently, if a subset is not frequent, then all of its supersets must also not be frequent. However, for Apriori-like algorithms, multiple scans of database are necessary, which dramatically increases the computing overhead as it requires n scans, which n is the length of the longest pattern. So far, most of Apriori's variations and other association rule mining algorithms improve their performance on the below three aspects: reducing the number of candidates generated, reducing the number of database scans, and using compact tree-based data structure to store itemsets.

DHP [11] is one of the variations of Apriori. It uses a hashing based technique to reduce the number of candidates generated in each scanning. A hashing table is constructed to prune unnecessary itemsets for the generation of the next set of candidate itemsets. However, DHP still needs multiple database scans.

Another variation of Apriori, Partition algorithm [12], optimizes Apriori in terms of reducing the number of database scans. It consists of two phases. First of all, the database D is divided into a number of non-overlapping partitions. Secondly, in order to determine the global frequent itemsets, another scan for database D is implemented to extract the actual support count of each candidate itemset. However, it is noticed that the weakness of Partition algorithm is that it consumes large buffer space if the target database is huge.

FP-tree [2] and Sampling [13] typify the other two types of association rule mining algorithms. In FP-tree, a tree–based data structure is employed to compress large database into a compact form, which only stores frequent patterns. However, the FP-tree algorithm needs to maintain such a tree in memory space until the end of the mining process. Thus, if the database is large enough, it would be very resource consuming. Another algorithm, Sampling, can get approximate results in a normal case, but in a worse case, association rules can be lost from the loss of the rest of database scans [13].

Regarding to the bottle neck of Apriori families, a transaction pruning approach was

proposed in ODAM [14] to reduce the size of database transactions for entire scanning process. However, this method needs to store the new condensed transactions in a temporary file before scanning, which results extra storage and transaction replication overhead. Different from this, the filtering method presented in this thesis dynamically adapts the change of user specified association threshold, thus, no extra storage and database replication overhead produced during the procedure. In section 5, we will compare the performance of our filtering method along with Apriori and replicating condensed transaction approach.

Interestingly, besides traditional application areas such as cross basket analysis and customer relationship management (CRM), ARM can also help file system designers to develop more intelligent file system policies. In the next chapter, we will discuss the related work in detail.

2.2 Classification and Prediction

Classification and prediction are the other two data mining techniques that are used to build models identifying target data classes or forecasting upcoming data trends. The difference between classification and prediction may be defined variedly due to diverse backgrounds of data mining practitioners and researchers. J. Han [2] defined classification and prediction based on their outputs, where classification predicts categorical labels, while prediction models continuous-valued functions. For example, a classification model may be built to predict whether or not monthly-paid mobile phone contract should be issued to customers, while a prediction model can be developed to forecast the income of employees based on their working experience and qualifications.

Classification is a two step process. First of all, it takes previous known data as its training data, which in form contains a certain number of records. Every record has the same number of attributes belonging to finite class labels. The classification models are built based on analyzing the attributes with provided class labels. Since the class labels of previously observed records are provided, the first step is also called supervised learning. In contrast, the class labels and the number of classes of training data may not known

before training such as clustering technique, which is a typical unsupervised learning process.

The learned model can be expressed a set of classification rules, decision trees or formulas. This makes classification model easy to interpret and understand. For example, a classification model may be built on a given car insurance company database, where a set of classification rules can be extracted by the classification model as either safe or risky to issue a insurance contract. The classification rules can be used to identify the future customers' insurance requests and also provide better understanding of the data in their database.

The second step of classification is to use the model to classify future data records. First of all, the accuracy of the model must be evaluated through various methods. If the accuracy is considered in the range of acceptance, the model is then ready to classify future records or objects, which the class labels are unknown.

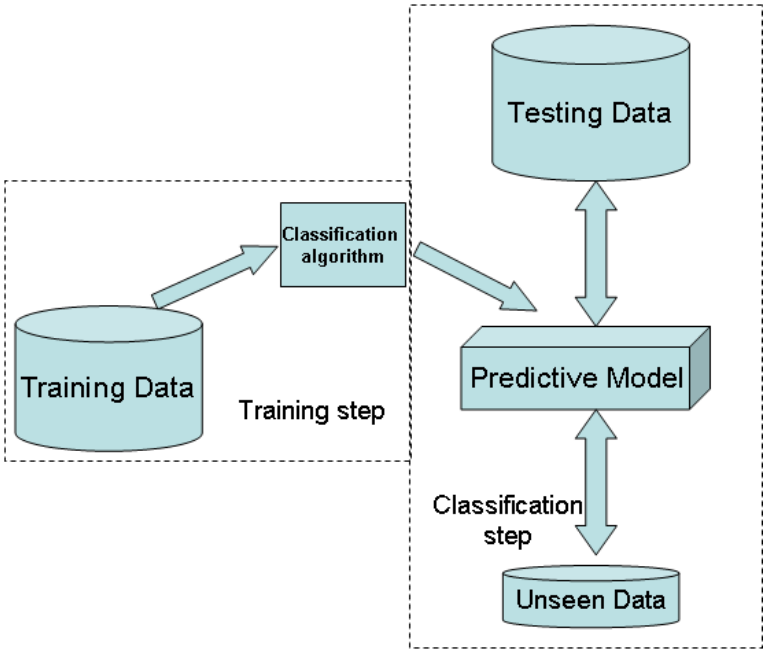


Figure 2.2 The 2 steps of classification.

In general, the first step trains the historic data using specific algorithms and the second step validates the model and predicts unseen data. Interestingly, people always argue the difference between classification and prediction. Some people define classification and

prediction based on the algorithms and models used. However, the most commonly accepted view is that people use predictive model to predict discrete classes or nominal values as classification. In contrast, we use predictive model to predict continuous or ordered values as prediction.

2.3 Clustering

Unlike classification, clustering is the process to group data with high similarity into the same class or cluster without knowing the class labels. A cluster is a collection of similar data or data objects that are dissimilar with other data or data objects in other clusters. Cluster analysis can be used for pattern recognition, image processing, customer grouping, bioinformatics and social network analysis. With clustering technique, one can discover the distribution of data and important associations among data attributes.

Cluster analysis is a traditional statistic branch that has been well studied for many years. The current statistic analysis software such as SAS and SPSS contain distance-based cluster analysis such as k-means, k-medoids and several other methods. However, in data mining, researchers are putting efforts on designing efficient cluster analysis algorithms and models for very large databases. The scalability of cluster analysis algorithms and models is considered the most important factor from data mining view. For example, some traditional cluster analysis methods work very well on small data set containing less than 500 data objects, however, a typical data mining task may involve large databases containing several terabytes data. Sampling a small faction of the entire database may cause inaccurate results. Besides scalability issue, a few other requirements are also addressed by research communities to evaluate a clustering algorithm. The table below lists the typical requirements for clustering analysis from data mining view.

Ability to deal with different type of attributes	Discovery of clusters with arbitrary shape	Minimal requirements for domain knowledge to determine input parameters	Ability to deal with noisy data	Insensitivity to the order of input records	High dimensionality	Constraint-based clustering
Applications may require various types of data such as numerical, binary, categorical, ordinal and mixtures of above.	Algorithms based on distance measures tend to find similar size clusters. However, sometimes, it is demanded to find clusters with different shape	The clustering results can be sensitive to the input parameters. A minimal requirement of parameters makes the quality of clustering controllable.	Noisy data such as missing values, unknown types, and erroneous data may lead to biased results and poor clustering quality.	Some clustering algorithm may generate different clusters by given different input orders.	It is challenging to cluster data objects in high dimensional databases, where data can be very sparse and skewed.	Real life applications require clustering works under different constraints. It is important to find groups of data satisfying specified constraints.

Table 2.3 various requirements of clustering analysis from data mining view

In literatures, lots of clustering algorithms have been developed. To choose a specific clustering algorithm, both the available data types and certain applications are considered. In general, clustering algorithms can be categorized as 5 different types: Partitioning methods, Hierarchical methods, Density-based methods, Grid-based methods and Model-based methods. However, some clustering algorithms may involve several different clustering methods. Therefore, it is very difficult to classify every particular clustering algorithm. In this part, let us review one of the most widely used clustering algorithm K-means, which is a typical Partitioning method.

First of all, the k-means algorithm randomly chooses k clusters containing a collection of data objects. The algorithm then calculates the mean value of data objects in the same cluster as the central point. Next, the data objects are re-assigned to a particular cluster according to the calculated least distance between the data objects and the central points. After the re-distribution, the above steps are iterated, until the criterion function converges. Generally, the squared-error criterion is used to make the modelled k clusters as compact

and as separated as possible. It is defined as:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \left| p - m_i \right|^2$$

Equation 2.1 Squared Euclidean distance for all data objects

Where E is the squared Euclidean distance for all data objects, p is the point in space representing a given object and m_i is the central point (mean value of data objects in a chosen cluster) of cluster C_i. Suppose that we have a set of data objects needed to be clustered as shown in figure below. With a given k=2, the task is to cluster the data objects into two compact clusters.

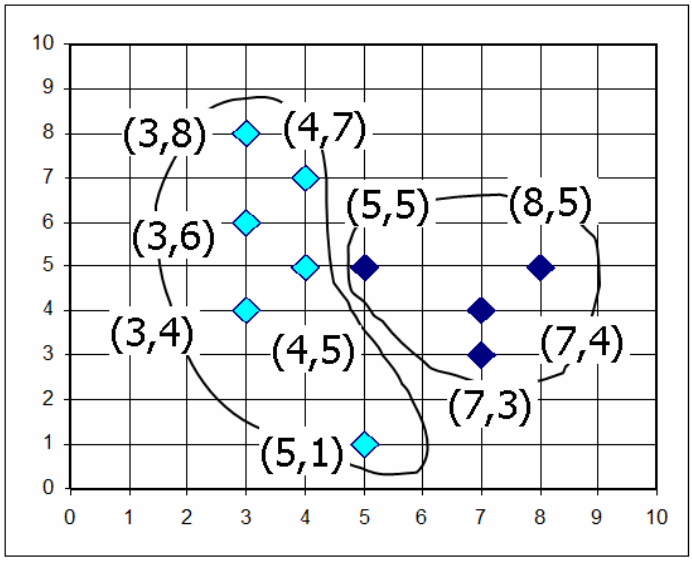


Figure 2.3 Initializing two clusters randomly [2]

First of all, we randomly initialize data objects into two clusters shown in figure above. Light blue points represent data objects in cluster one, while dark blue points represent data objects in cluster two. We then calculate the mean value of each cluster as the central points as follows:

For cluster one: $X_{mean} = \frac{3+3+3+4+4+5}{6} = 3.7$

$$Y_{mean} = \frac{8+6+4+7+5+1}{6} = 5.2$$

For cluster two: $X_{mean} = \frac{6+7+7+8}{4} = 6.9$

$$Y_{mean} = \frac{5+4+3+5}{4} = 4.3$$

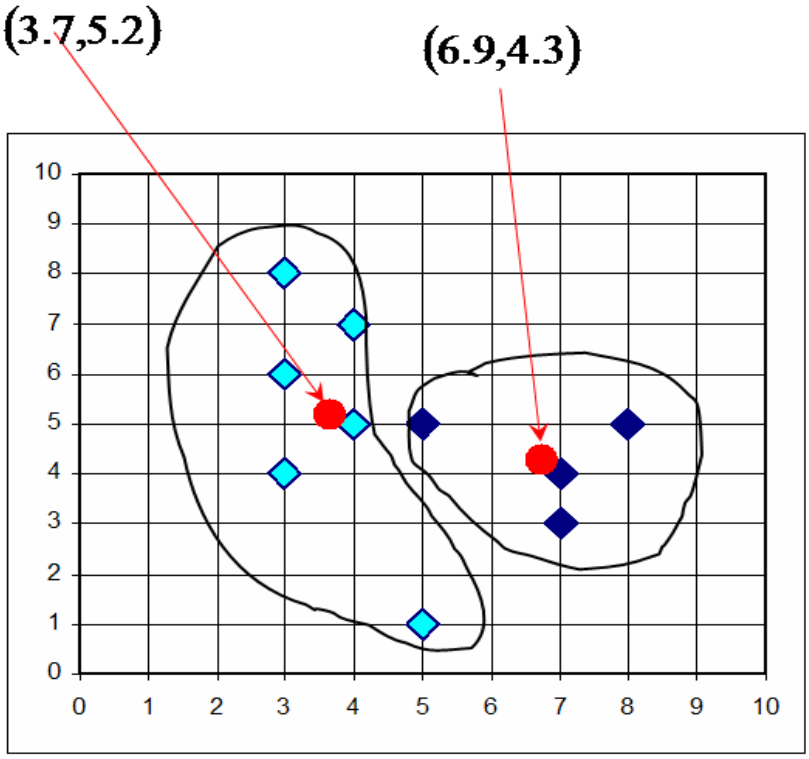


Figure 2.4 Mean values calculated in each clusters [2]

After calculating the mean value (central point) of each cluster, we can obtain the distance between the central point and data objects. The distance is used to decide which cluster the data objects more likely belong to. To calculate the distance, the most popular used distance measure, Euclidean Distance, is applied in K-means algorithm.

$$D\{(5,1),(3.7,5.2)\} = \sqrt{|5-3.7|^2 + |1-5.2|^2} = 4.4$$

$$D\{(5,1),(6.9,4.3)\} = \sqrt{|5-6.9|^2 + |1-4.3|^2} = 3.81$$

The above calculation shows the distances between data object (5, 1) and central points of two clusters relatively. We can see that the data object (5, 1) more likely falls into cluster two rather than its original cluster. The re-distribution should be done suggested by the calculated distances.

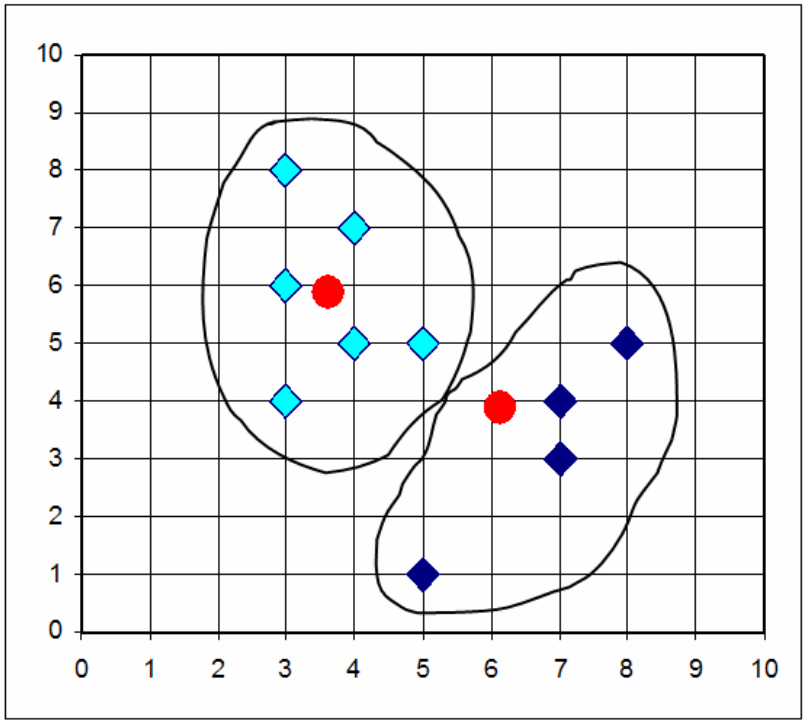


Figure 2.5 Re-distribution by calculated Euclidean Distance [2]

The circled areas above show the clusters after the re-distribution of the data objects. However, as the central points (mean values) will also be changed after re-distribution, the algorithm needs to re-evaluate the distances between data objects and the new central points. This step is iterated until there is no re-distribution caused.

Besides distance-based partitioning methods, various data types may apply different clustering methods. The most commonly seen data types are data matrix, dissimilarity matrix, interval-scaled variables, binary variables, nominal variables, ordinal variables and ratio-scaled variables. However, mixed data types may appear in particular applications.

2.4 Frequent Sequence Mining

Frequent sequence mining sometimes is also called sequential pattern mining, which is the process of searching for frequently occurring patterns in time series sequences. It is appropriate to apply frequent sequence mining in time series applications such as weather monitoring data, stock data analysis, sales trends forecasting and so on. Most researching efforts focus on symbolic patterns, while statistical time-series analysis concentrates on

predicting the numerical curves. One can treat the frequent sequence mining as the ordered and time-related association rule mining.

Some parameters may strongly affect the results of frequent sequence mining. The table below shows the parameters that should be set carefully during the mining process.

The duration of the time sequence	Event folding window	Time interval
The duration can be a whole sequence or a subsequence. The mining process then confines to the data within a certain period.	Events occurring with a specific period of time can be regarded as associated events (pattern).	The interval separates the events with a given period of time. The interval can be zero, a time range, or a none zero value.

Table 2.4 Important parameters affecting frequent sequence mining accuracy

Most of the frequent sequence mining algorithms are based on Apriori association rule mining since the Apriori property suggests that if a sequence S is infrequent, the all its super-sequence(the longer sequence containing S) can not be frequent. Another method for finding frequent sequence is based on FP-tree association rule mining algorithm we discussed in the previous section. This method avoids candidate generation during the process, which is good for time series data in terms of response time. Recently FP-tree based frequent sequence mining algorithm has been revised by research communities and many variations have been developed. In this section, we discuss one of the most broadly adopted frequent sequence mining algorithms called WAP-tree (Web Access Pattern Tree) [15]. The algorithm was originally designed for analyzing web access pattern. It was noticed that file requests and disk requests are very similar with web requests in terms of data format and structure.

Let us consider a set of events denoted as $\boldsymbol{E}$ and a disk requests access sequence $S = e_1 e_2 ... e_n$ ($e_i \in E$) for ($1 \le i \le n$) is a sequence of events, while n is called the length of the access sequence. An access sequence with length n is also called an n-sequence. Access sequence $S' = e_1' e_2' ... e_l'$ is called a subsequence of access sequence $S = e_1 e_2 ... e_n$, and S a super-sequence of S', denoted as $S' \subset S$, if and only if $S' \neq S$. For access sequence $S' = e_1 e_2 ... e_k e_{k+1} ... e_n$, if subsequence $S_{suffix} = e_{k+1} ... e_n$ is a super sequence of

pattern $P = e_1^{'}e_2^{'}...e_l^{'}$, and $e_{k+1} = e_1^{'}$, the subsequence of S, $S_{prefix} = e_1e_2...e_k$, is then called the prefix of S with respect to pattern P.

Given a group of disk access sequences $\{S_1, S_2, ..., S_m\}$ and a minimum support ***Min_Sup*** distinguishing frequent sequences and infrequent sequences, the problem is to mine all the complete set of the sequences with their frequency greater than ***Min_Sup***. The WAP-tree construction is illustrated as follows with a given disk access sequences database. Suppose the ***Min_Sup*** is 3, the table below shows the given disk access sequences and frequent subsequences.

Request ID	Access Sequence	Frequent Subsequence
100	*abdac*	*abac*
200	*eaebcac*	*abcac*
300	*babfaec*	*babac*
400	*afbacfc*	*abacc*

Table 2.5 A database of disk request access sequence

The WAP-tree is built via two database scans. The first scan is to find out all unique frequent events (has their frequency greater than ***Min_Sup***). Next, the second scan inserts frequent events node starting from a virtual root. The third column of the above table represents the frequent subsequences after first scan, which has already eliminated infrequent events from access sequences. The WAP-tree construction after the first scan is based on the fact that frequent sequences with same prefix can share same branch in a tree structure. Given a disk access sequence database shown in table 2.5, the following figures illustrate the WAP-tree construction process.

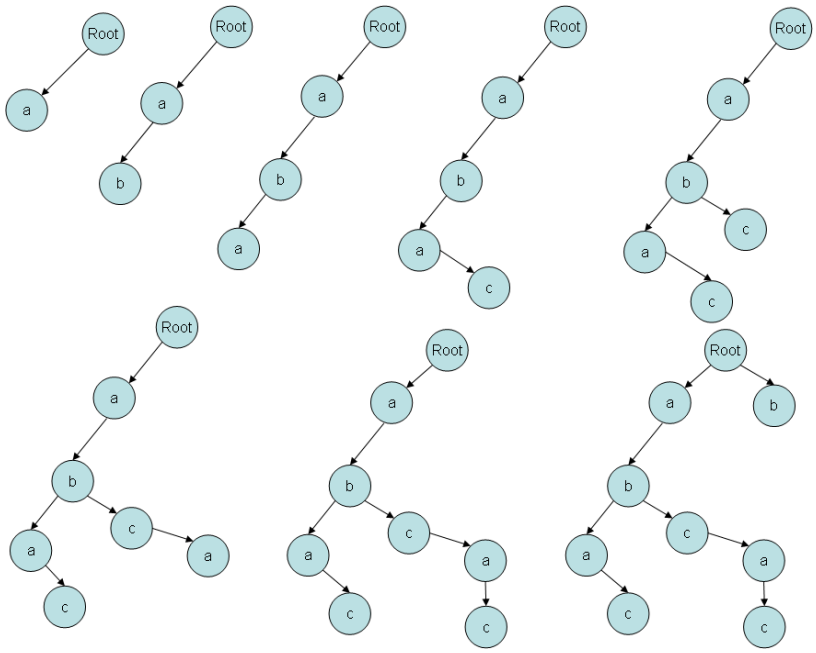


Figure 2.6 The WAP-tree construction processes

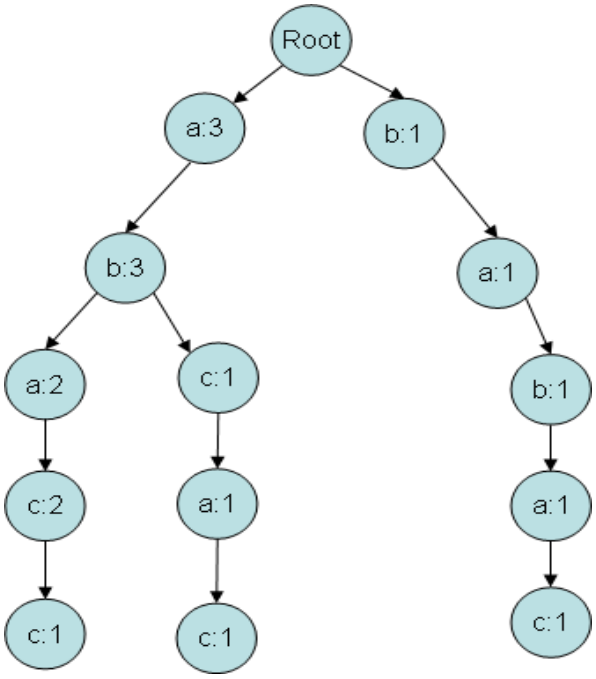


Figure 2.7 The fully developed WAP-tree based on the given disk request access sequence database

During the tree node insertions, each node is registered as a (label, frequency) pair shown in the node circle. From the root on the top of the tree, when a frequent sequence is going to be inserted, WAP-tree will check the first event in this new sequence to see whether it is

registered. If the first event is already registered, it will increase the relative frequency by 1 otherwise generate a new node extended from the root.

We can see that there are two significant advantages. First of all, the WAP-tree is very efficient compared with the original sequence database. It registers all frequency information and reduces the complexity of mining process from redundant database scans. Secondly, with a given condition (an event or a subsequence), their prefix based searching is much quicker without explosive candidate generation.

We noticed that the WAP-tree and its variation had been successfully adopted to analyze storage and file system access sequence patterns. In the next chapter, we will review the related works in detail.

2.5 Transaction-Filtering Data Mining For Association Rule Mining

In this section, a database transaction filtering method is proposed to speedup iterative database scanning, especially for association rule mining. Although many efforts have been made to improve association rule mining over the years, transaction scanning still suffers the huge scanning overhead of high dimensional databases. Regarding to the bottle neck of Apriori families, a transaction pruning approach was proposed in ODAM [14] to reduce the size of database transactions for entire scanning process. However, this method needs to store the newly condensed transactions in a temporary file before scanning, which causes extra storage and transaction replication overhead.

Different from this, the method proposed in this section utilizes a memory-resident data structure as a filter to shorten the length of database transactions for scanning. This filter is expected to reduce the scanning time over the filtered transactions. This filtering method dynamically adapts the change of user specified association threshold, thus, no extra storage and database replication overhead produced during the procedure. In this, we will compare the performance testing results of our filtering method along with Apriori and replicating condensed transaction approach (ODAM).

2.5.1 The Design of the Database Filter

Different from those algorithms we have discussed, this thesis proposes a transaction filtering technique that blocks out all infrequent items to generate filtered transactions for each database scan. The idea behind this filtering technique is that association rules mining only concerns potentially frequent itemsets in transactions. If we can remove those infrequent items in transactions, thus each database scan will be more efficient than using original transactions. However, deleting items in database transactions is not realistic. Once the user-specified threshold or minimum support changes, the boundary between frequent items and infrequent items will also change. Other purposes of database application will also not allow this deletion

To state the problem, let us denote C_k as a set of candidate k-itemsets, which represents the potentially frequent itemsets; and F_k as a set of frequent k-itemsets. In Apriori-like algorithms, after the first database scan, we can obtain all frequent 1-itemsets F_1, then generate candidate 2-itemsets C_2 and count their occurrences in the second database scan to determine the frequent 2-itemsets F_2. Similarly, to find out frequent k-itemsets F_k, we need to self-joining frequent (k-1)-itemsets F_{k-1} to generate candidate k-itemsets C_k and count their occurrences in the k-th database scan. However, in the k-th database scan (where k>1) the original transactions contain infrequent items, which do not exist in C_k. In other words, as those infrequent items do not involve in self-joining, we can safely block them from being read into the memory space for candidate itemsets verification.

Our method is to maintain and update a data structure as a filter, which only stores frequent items at the end of each iteration, to verify whether or not an item in a transaction deserves to be read into memory for scanning. Nevertheless, this filter needs to be updated after each iteration, because the items in the current filter may not be frequent in the later passes. After first iteration, we can find all the unique items and their occurrences. Those unique items with occurrences greater than Support threshold is the frequent 1-itemsets F_1. We use this frequent items list as the initial filter, $FT_1 = F_1$. During the second pass, we check whether or not the items of an original transaction, $i \in T$, can be found in FT_1. The items

that can be found in FT_1 are the elements of the new transaction for the next scanning. In the third pass, a few items may not be frequent since they do not appear in C_3, then we update FT_1 to FT_2 by eliminating the items that do not appear in C_3. In general, the filter in the (k+1)-th database scan can be described as: $FT_k = FT_1 \cap C_{k+1}$.

Considering a given minimum support count as 3, the procedure of our filtering process is presented in the following example. Table 2.6 is the sample transactions in database D used for the algorithm demonstration.

Transaction ID	Items
T100	A,C,D,F
T200	B,C,E,F
T300	A,B,C,E
T400	B,E,F
T500	E,F

Table 2.6 Transactions in a sample database

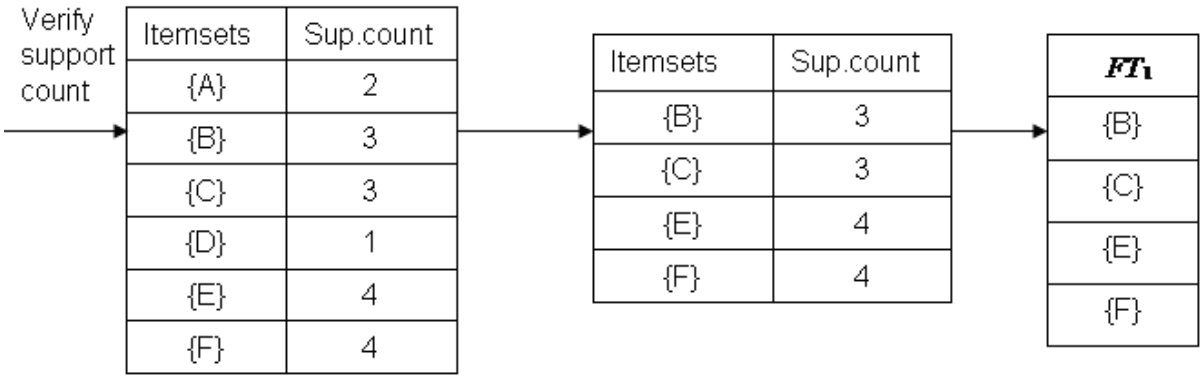


Figure 2.8 Initializ the database filter

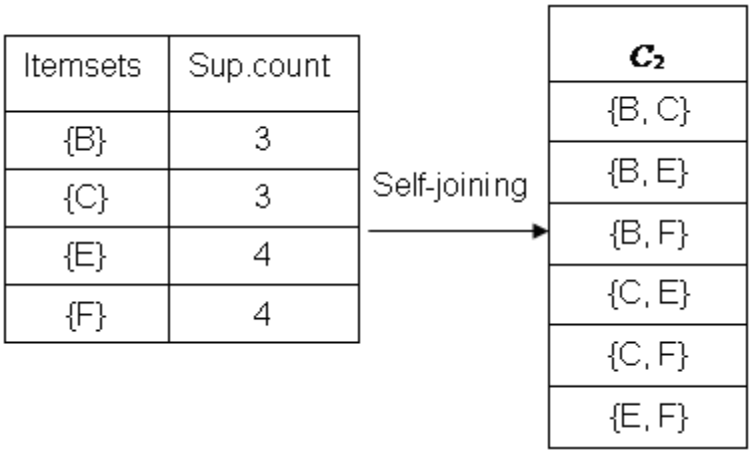


Figure 2.9 Generating candidate 2-itemset

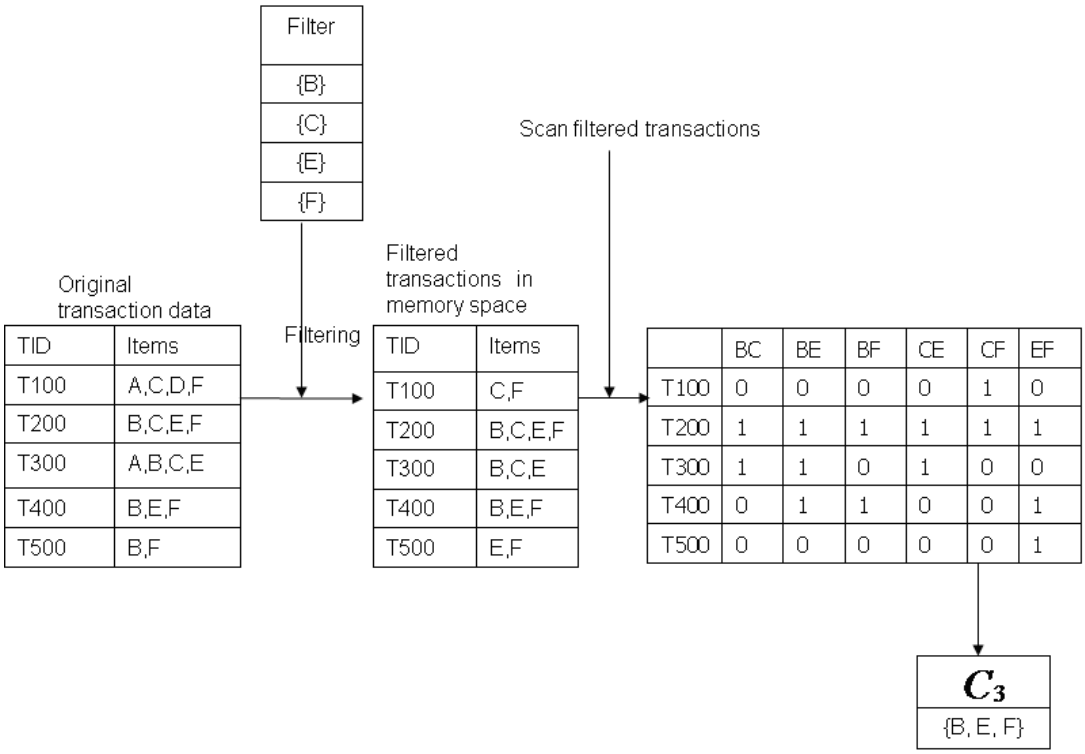


Figure 2.10 Forming new filtered transactions

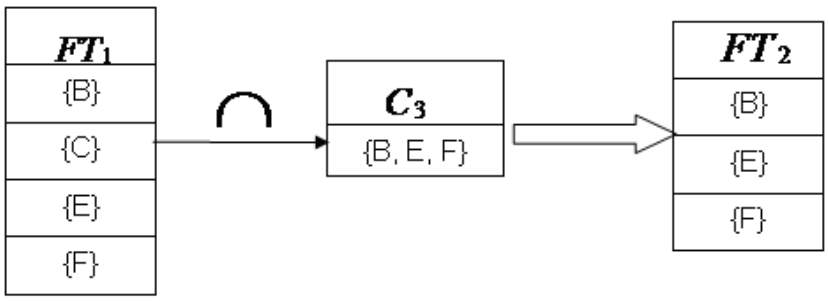


Figure 2.11 Updating the filter

The transactions of the database D are moved from the disk to the memory in groups of disk I/Os for scanning. When one transaction is being scanned, the filter qualifies all the items to determine whether or not to let them pass to form a new shortened transaction. In other words, if an item is matched in the filter, then it is kept in the memory, otherwise, it is discarded. The above procedure is repeated until all items in the corresponding transaction are tested.

In principle, the proposed filtering technique can be applied to other Apriori-like algorithms such as DHP and Partition. The formal statement of the filtering method is represented below in pseudo code shown in Figure 2.12.

```
Input: Database
Output: Frequent k-itemset
/* Database = set of transactions;
Items = set of items;
transaction = <TID, {x ∈ Items}>;
F(k) is a set of frequent k-itemsets
F(k) =φ(empty);
FT(k) is the filter for (k+1)-th database scan;
FT(k)=φ(empty);
C(k) is a set of candidate k-itemsets
C(k)= φ(empty);
Tmp_t is a temporary vector for storing filtered transaction;
Uni_itemsets(k) is a set of unique k-itemsets in database*/
```

```
/*Find F(k) , the set of frequent k-itemsets, where k ≥ 2*/
for each (k= 2; F(k-1)!= φ ; k + +)do begin
/* form candidate k-itemsets C(k)*/
C(k)=F(k-1)*F(k-1)
End
/*update filter for k-th database scan*/
FT(k-1)=FT(1) ⌒ C(k)
for each transaction t ∈ Database do begin
   for each item x in t do begin
If FT(k-1).find(x)
then
Tmp_t.push_back(x)
   End;
```

```
/*verify candidate k-itemsets using filtered t ransaction*/
   for each candidate k-itemset y in C(k) do
     if Tmp_t.find(y)
then Uni_itemsets(k).y.count++
else
Uni_itemsets(k).add(y)
Uni_itemsets(k).y.count=1
   End
 for each unique candidate k-itemsets y in Uni_itemsets(k) do
If
Uni_itemsets(k).y.count ≥ Sup_min
Then
F(k).add(y)
   End
End
Output F(k) (k=1;k<k+1;k++)
```

Figure 2.12 The pseudo code of database filtering method for Apriori-like ARM

Based on the above discussion, we can also extend the filtering method to the computation distributed ARM scenarios. In some cases, data mining tasks are geographically distributed and databases are still centralized. Such distributed ARM scenarios can be described as a multi-clients/server model.

The communication cost between data server and data mining client scales in various network conditions such as delays, traffic jam. Therefore, the conventional data collection method is not efficient, which transmits the data consisting both frequent and infrequent items directly to the local clients. To illustrate communication cost over networks, we

denote communication cost as CC, B as the number of bits in message, S as the transmission rate. The communication cost CC=c+(B/S), where c is a fixed cost of initiating a message from one site to another, known as access delay. In our instance, using an access delay of 1 second and a transmission rate of 10Mbps,we can calculate the time to collect 1,000,000 transactions, each consisting of 720 bits (in average) stored in a relational table: CC=1+ (10 × 720/10) =73 seconds.

From the above example, it is noticed that if we can shrink the average length of the transactions transmitted over the network, the over all communication cost will be reduced. The method is extended to a mobile agent-based filtering design to reduce the size of transactions transmitted over networks. In Figure 2.13, each local client running ARM engine sends out a filter agent to the data server along with the predetermined thresholds, then the filter agent runs on the data server and sends back the filtered transactions in group fitting in with the network bandwidth. Thus, only small amounts of data collected from data server compared with non-filtered transaction transmission. The amount of reduced communication cost highly relies on the predetermined threshold, the higher support threshold specified, more infrequent items filtered out.

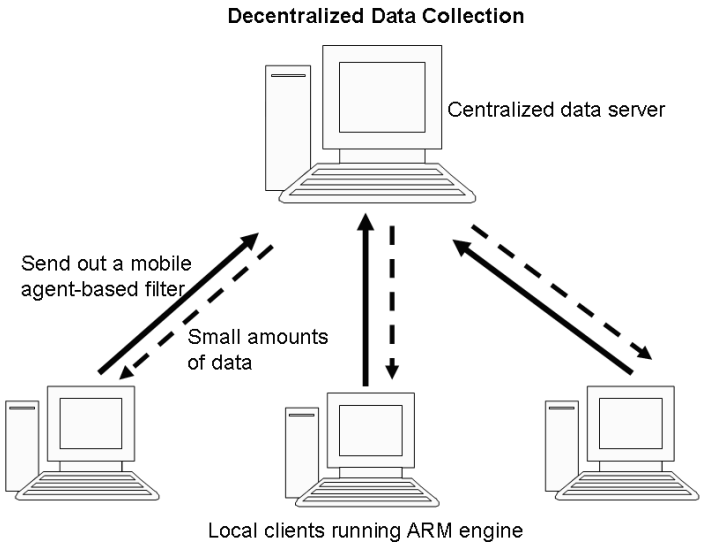


Figure 2.13 Mobile agent based filtering model

To illustrate the efficiency of this mobile agent-based filtering design, we conducted a set of simulations with various network delays. The experimental results are given in the next section.

2.5.2 The Implementation of the Proposed Data Pre-Processing Method

In this section, we discuss the validation results of the proposed filtering method as well as the simulation results of the mobile agent-based filtering design. To evaluate the performance of the proposed method, the filtering method is implemented in C++ language along with the Apriori algorithm on a Pentium IV 1.8 GHz PC with 512MB memory under Linux platform. The proposed method is also compared with the transaction pruning approach [14], which replicates the condensed transactions in a temporary file for scanning. To illustrate the method is not confined to single dataset; both synthetic datasets and real life datasets are adopted for the testing.

The first benchmark used is a set of synthetic datasets generated by the program Quest Synthetic Data Generation Code (QSDGC) [16] designed by IBM Almaden Research Center. This program has been widely used by many research communities [10], [11], [12], [17], [18]. By setting up parameters of the program, we can generate desired datasets as target association rules mining transaction data. Table 2.7 describes the parameters of the dataset configured in the program.

Dataset parameters	Definition
T	The average number of items in a transaction
I	The average size of maximal frequent itemsets
D	The number of transactions

Table 2.7 The definition of dataset parameters

The different experiments were conducted to test the efficiency and scalability of our filtering method. In the first test, 1 million transactions were used as the target mining transaction data and set the average size of the maximal frequent itemsets as 10. By increasing the parameter T, which is the average number of items in a transaction, from 15 to 40, the overall execution time decreases with the increase of support threshold. The reason is that higher support threshold produces more infrequent items; therefore, those increased infrequent items are blocked out from being scanned by the transaction filter. The following figures show the testing results.

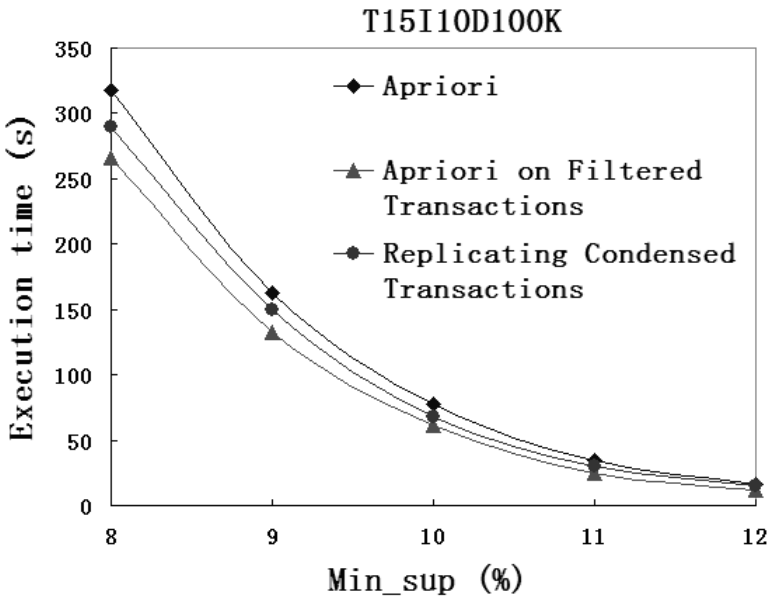


Figure 2.14 Execution time comparison (T=15)

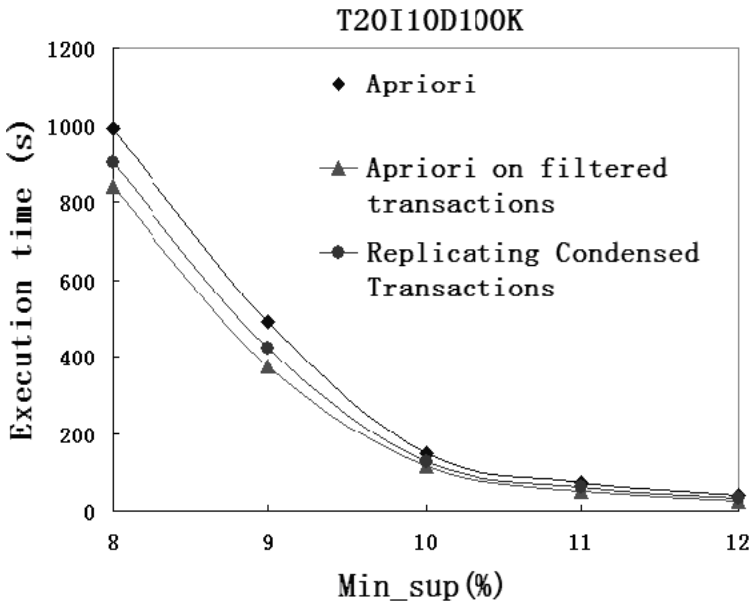


Figure 2.15 Execution time comparison (T=20)

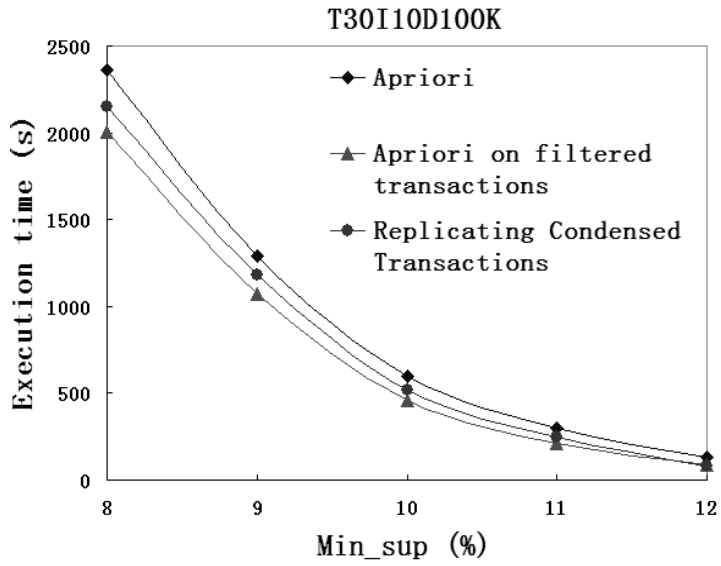


Figure 2.16 Execution time comparison (T=30)

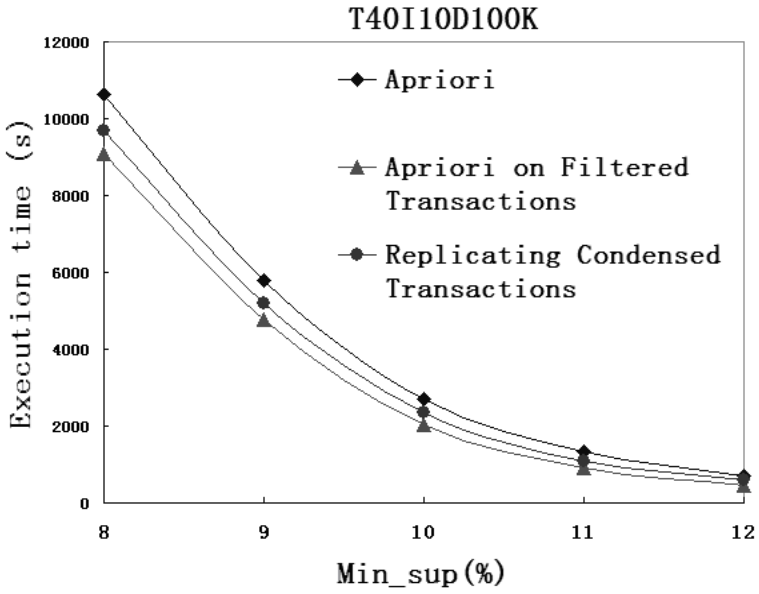


Figure 2.17 Execution time comparison (T=40)

It can be seen from Figure 2.14, 2.15, 2.16 and 2.17 that the parameter T doesn't contribute much to execution time enhancement ratio. This is because when we increase the length of a database transaction, both the number of frequent and infrequent items will increase simultaneously. As the designed filtering technique reduces the dimensionality of each transaction from n to n-x (n is the number of average transaction size and x is the number of infrequent items), the only factor directly influencing the efficiency is x (the number of infrequent items). Consequently, increasing the average number of items in transactions

can not affect x. To increase x, the larger association threshold (Min_sup) is needed. Therefore, we can conclude that the larger specified association threshold (Min_sup), the better filtering method performs. Compared with the replicating condensed transaction approach, our method does not require extra data replication cost.

The scalability of the proposed filtering technique was also tested along with ODAM. In Figure 2.18, 2.19, 2.20 and 2.21, the size of transaction data only linearly increases the execution time of our filtering method. However, the extra I/O overhead produced by replicating condensed transaction approach scales with database size. We can imagine that if the database is large enough, replicating all the condensed transactions for scanning is not efficient.

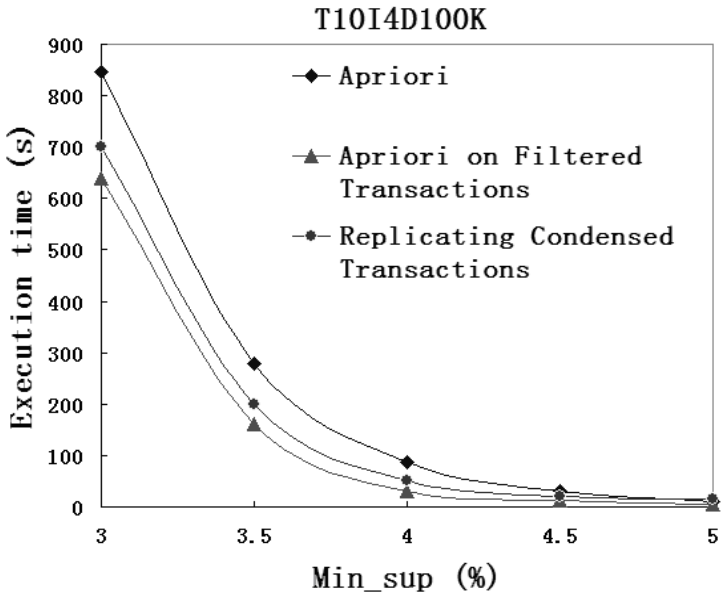


Figure 2.18 Execution time comparison (D=100k)

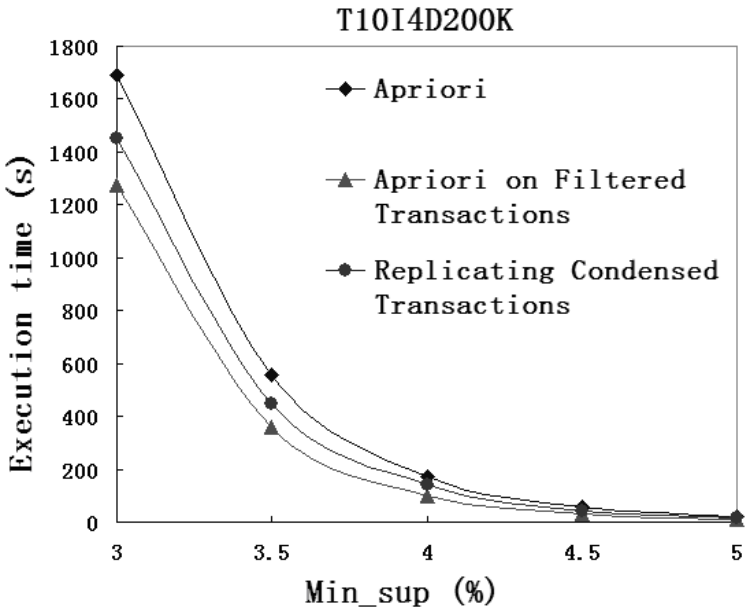


Figure 2.19 Execution time comparison (D=200k)

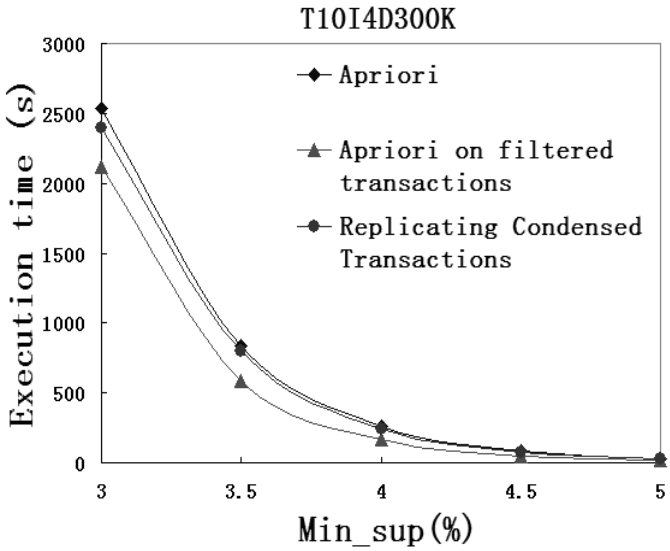


Figure 2.20 Execution time comparison (D=300k)

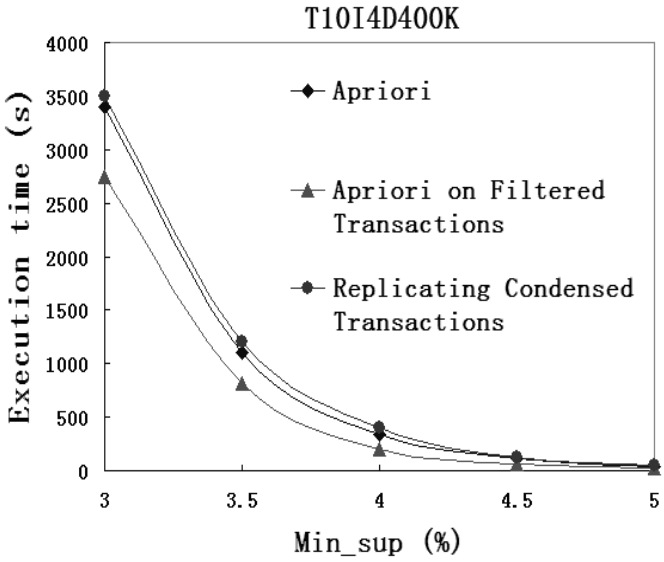


Figure 2.21 Execution time comparison (D=400k)

Additionally, another test shows that increasing parameter I (The average size of the maximal frequent itemsets) can cause the raise of execution time by a given pre-determined support threshold. In figure 2.22, Apriori and filtering method scale linearly in I as well as replicating condensed transactions. However, the execution time enhancement ratio fluctuates at a certain level in terms of scalability.

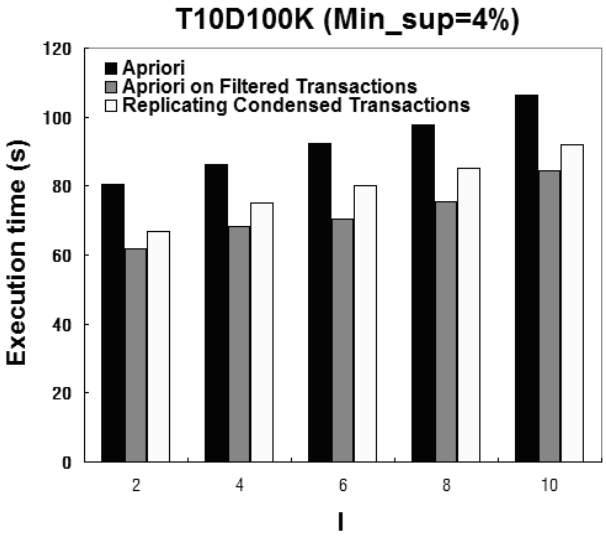


Figure 2.22 Execution time comparison based on parameter I

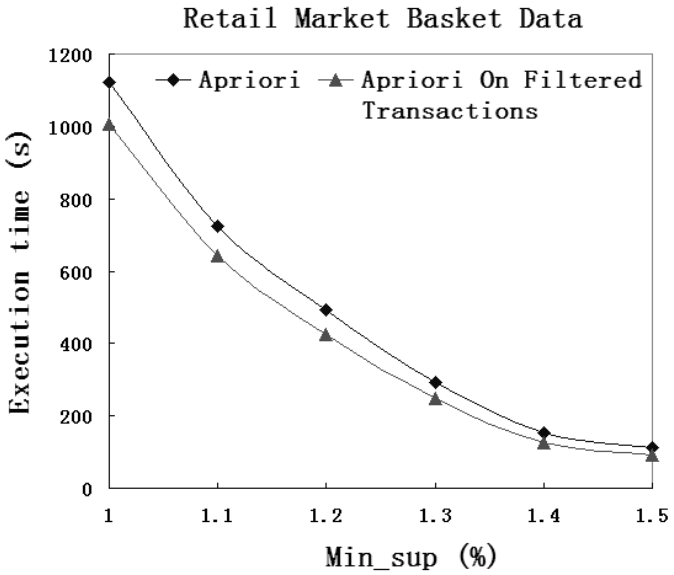


Figure 2.23 Execution time comparison on retail market basket data

The second real life dataset, Retail Market Basket Dataset [19] was donated by Tom Brijs [20] and contains the (anonymized) retail market basket data from an anonymous Belgian retail store. The data were collected over three non-consecutive periods. The first period runs from first half December 1999 to first half January 2000. The second period runs from the beginning of January 2000 to the beginning of June 2000. The third and final period runs from the end of August 2000 to the end of November 2000. Each record in the dataset contains information about the date of purchase, the receipt number, the article number, the number of items purchased, the article price in Belgian Francs and the customer number. The detailed description of the dataset can be found in [19] [20].

The results from this real life dataset are similar to IBM synthetic datasets. Figure 2.23 illustrates the execution time comparison between Apriori and Apriori over filtered transactions with support threshold from 1% to 1.5%. By applying the filtering method, the ratio of enhanced execution time rises from 10.34% to 18.76% with the increase in support threshold from 1% to 1.5% respectively.

Another real life dataset used is the Traffic Accidents Dataset [21]. This dataset of traffic accidents is obtained from the National Institute of Statistics (NIS) for the region of Flanders (Belgium) for the period 1991-2000. More specifically, the data are obtained

from the Belgian "Analysis Form for Traffic Accidents" that should be filled out by a police officer for each traffic accident that occurs with injured or deadly wounded casualties on a public road in Belgium. In total, 340.184 traffic accident records are included in the dataset. The detailed description of the dataset can be found in [21] [22].

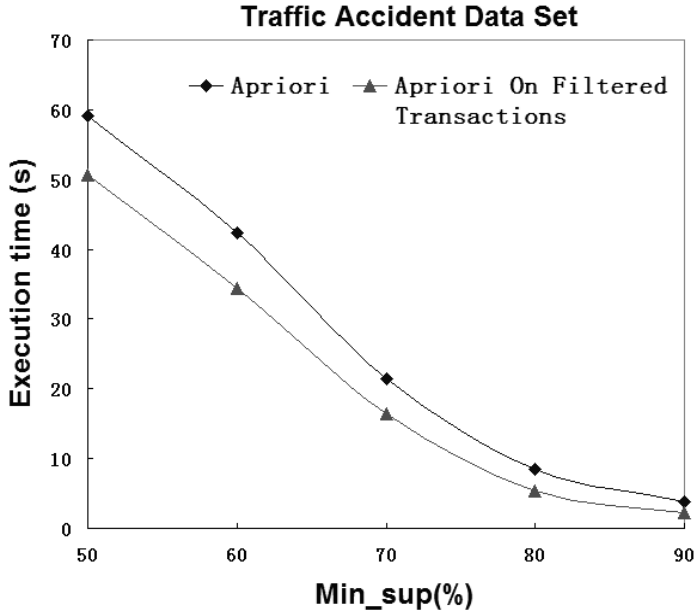


Figure 2.24 Execution time comparison on traffic accident data

This dataset produced the best performance results of the optimization. The ratio of enhanced execution time rises from 14.48% to 41.65% with the increase of support threshold from 50% to 90% relatively. It is noticed that in the nature of traffic accident data, some attributes have very high correlations. For example, the bad weather conditions and the critical road conditions are recorded together in most of the accidents. In other words, those attributes are not sensitive against the change of support threshold unless support threshold shifts to an extremely high level. The experimental results confirmed this when the support threshold was increased to 90%, a large number of the infrequent attributes (items) were filtered out from transaction scanning. That is the reason why the ratio dramatically jumps to 41.65%.

2.5.3 The Simulation of the Mobile-Agent-based Transaction Filter for Distributed Data Collection

In the following part of this section, we study the performance of the mobile agent-based filtering design. First of all, to build the various simulation environments, we set the networks delays in millisecond shown in the table 2.8.

Datasets	Delays	Support threshold
T40I10D100I	100ms,200ms, 300ms	11%,12%,13%,14%,15%

Table 2.8 Network delays setup

Figure 2.25, 2.26 and 2.27 show the communication cost reduction with various network delays compared with non-filtered data transmission. The filter is implemented as a mobile agent traveling between local clients and data server. When a local client requires data, it sends a filter along with a predetermined support threshold or any other constraints to the data server. The filter then runs on the data server side to produce cleaned transactions for the local client. Considering the usage efficiency of the network, transactions are processed in group to fit in with the bandwidth.

In figures 2.25, setting network delay as 100ms, the communication cost reduction scales 5.9%-55.7% with minimum support threshold 11%-15%. Correspondingly, in figure 2.26 and 2.27, setting network delays as 200ms and 300ms, the communication cost reduction scales 9.1%-66.7% and 12.3%-77.1% respectively.

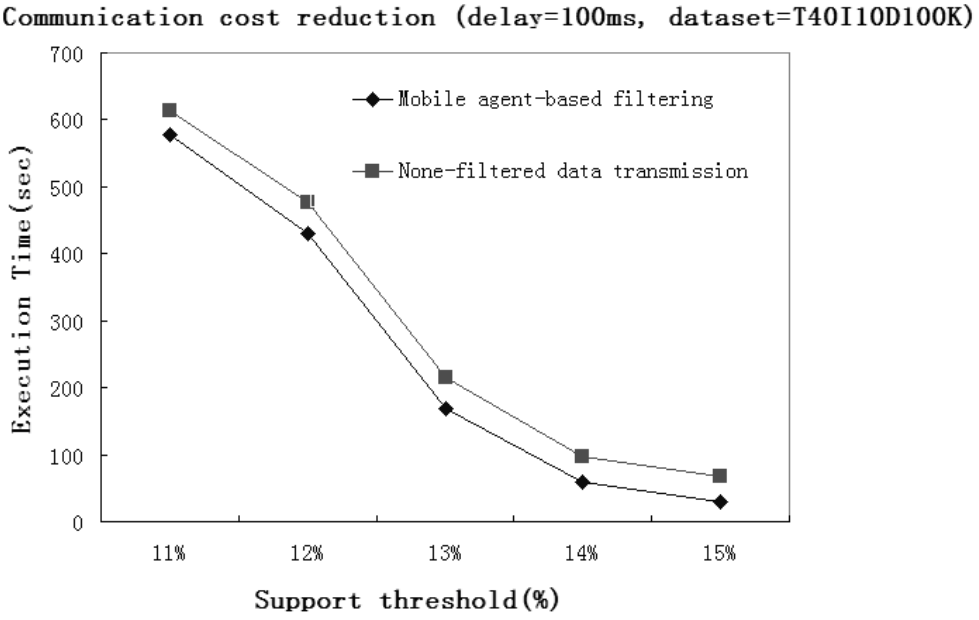


Figure 2.25 Communication cost reduction with 100ms network delay

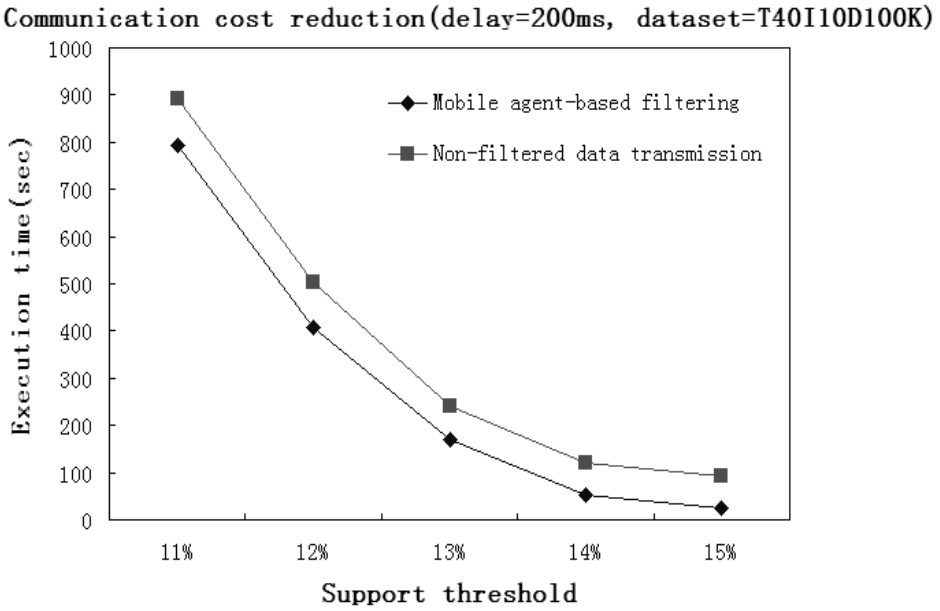


Figure 2.26 Communication cost reduction with 200ms network delay

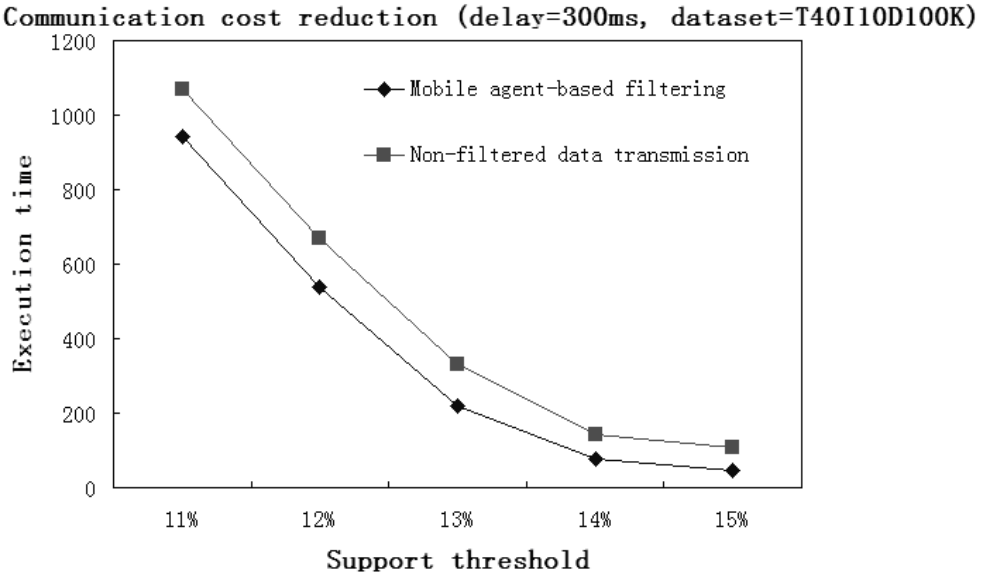


Figure 2.27 Communication cost reduction with 200ms network delay

Unlike other association rule mining algorithms, the proposed method focuses on speeding up the database scanning based on a database preprocessing technique. The experimental results have proved its efficiency and scalability over diverse transaction datasets especially when relatively high support thresholds are required. In the experiments, the proposed filtering technique reduces the average execution time of the Apriori algorithm by 12%-31%, given the thresholds are 8%-12%. Moreover, the simulation results of the mobile agent-based filtering design show that the proposed filtering method reduces communication cost 11%-67%, given the thresholds are 11%-15%. As the transaction filtering technique uniquely optimizes Apriori-like algorithms in terms of reducing database transaction length, it can be adopted for other Apriori-like variations such as DHP and Partition to further improve their efficiency.

Summary

In this chapter, four major data mining methodologies: Association Rule Mining, Classification, Clustering and Frequent Sequence Mining have been discussed in both conceptual and algorithmic level. Although the discussions only contain the very original algorithms, other variations of these data mining methodologies were developed mainly based on the original algorithms. The purpose of the data mining methodology review is to identify and clarify the data mining branches and their relative tasks as well as the likely

applications. With a clear understanding of various data mining methodologies, one can choose appropriate candidate algorithms and models to solve a specific data mining problem.

A universal data pre-processing method is proposed in this chapter, which enables fast database transaction scanning and data records collection in distributed data mining environments. A set of benchmark testing results have shown that the proposed data pre-processing method can effectively reduce database scanning time from 10% to 20% in average. In addition, the distributed data mining experiments showed that the method can also reduce the data collection time from 11% to 67% with the given thresholds from 11% to 15%. However, network delays play a very important role in the distributed association rule mining experiments. Basically, more data records filtered by the pre-processing method, higher data transmission improvements can be achieved. To conclude, both the given mining threshold and network delays contribute to the performance improvements offered by the proposed transaction filtering method.

Having reviewed various data mining methodologies, one may naturally connect these techniques with real applications. Although some of heuristics were developed to achieve better performance for file and storage systems, the potential of utilizing data mining techniques is still much less explored compared with the development of hardware and the so called data exploration. In the next chapter, we review the related works of applying data mining technique to file and storage systems.

Chapter 3 The Background of Intelligent Data Management in File and Storage Systems

Although data mining has been successfully established in many application areas, computer systems engineering benefits little from data mining as massive amounts of raw data are automatically generated from systems but still staying undiscovered. Such kinds of data, which are referred as traces or logs, are produced to monitor system and user activities during their operations. However, these data are seldom examined until systems encounter fatal errors or crash down. In fact, the monitored activities in traces imply rich underlying patterns, trends and rules, which can be used to further analyze system performance. For example, the temporal links between user accesses imply the relationships of user required data in forms of file access patterns and block access correlations. The previous researches [23] [24] have showed that the stable file access patterns and block correlations can be obtained through utilizing data mining techniques to examine the system traces. The derived knowledge (patterns and correlations) have been successfully implemented into file and storage systems that can reduce latency of data accesses. In this chapter, we discuss the background of file and storage systems and examine two previously developed optimizations based on data mining techniques. A hints-based storage system, Self-Tuning storage system, is also introduced in this chapter, which utilizes classification technique to optimize file management.

3.1 File System and File System Caching Strategies

File system is a method to organize and store data in a computer system. In file systems, the smallest unit is a file, where data are stored. Various file systems may have different organizations and policies, however, acting as a logical method and manipulating files are common in every file systems. In this section, we discuss the background of one of the most important and widely used file system, Linux file system.

In Linux file systems, files are categorized into 4 types: ordinary files (data files), directory files, special files (device files) and pipe files. Ordinary files store user-created data such as texts, images, audio or video clips and so on. Directory files contain the information stating what files in this directory, how large they are, when they were last modified, etc. Special files are used for representing physical devices such as scanners, printers, or compact disk drive and etc. Finally, the pipe files are the temporal holders of the data during the invocations from one command to another. Moreover, files are described by their attributes such as creation time, user ID, group ID, permission mode, last modified time, last access user and etc..

In Linux file system, user ID and group ID identify whom the file belongs to and its group. Correspondingly, file mode claims the permission title in three separate parts. For example, if a file contains a user ID “john”, group ID “G1” and the file mode “-rw-r-r”, then it claims that “john” is the owner of this file and has the group “G1”; in the file mode, first set value “rw” denotes that “john” can read and write this file, second set value “r” claims that group users belongs to “G1” can only read this file; the third set value denotes other users can only read this file.

Currently, most of Linux file systems manage files on magnetic hard disk drives. In order to create different divisions of a hard disk, Linux file systems partition a hard disk into one or more independent logical disks. The disk partitions are operated as devices by Linux using the file I/O mechanism through special files (device files) in the /dev directory. In Linux file system, there is a partition table stored at the beginning address of a disk. It abstracts general information of the partitions on the disk. To view the partition information, we can bring out the partition table using command “fdisk”. The following example shows the partition table of a hard disk.

```
#fdisk –l
Disk /dev/had: 160.0 GB, 160000000000 bytes
255 heads, 63 sectors/track, 19452 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device    Boot    Start    End    Blocks    Id
```

System					
/dev/hda1	*	1	13	104391	83
Linux					
/dev/hda2		14	6540	52428127+	83
Linux					
/dev/hda3		6541	6670	1044225	82
Linux swap					

The term /dev/hda in the above partition table denotes an IDE drive. With the number following, /dev/hda1 refers the first partition, /dev/hda2 refers the second partition and so on. For SCSI drives, prefix denoting drives becomes /dev/sda, /dev/sdb, /dev/sdc and so on. In Linux, partitions can be following types: primary, extended, or logical partitions. Primary partition is a holdover due to the limitation of other partitions on old x86 systems. Unlike DOS and Windows, Linux can boot from either a logical partition or a primary partition.

To manage and organize data in a partitioned disk, file systems must ride on it. In Linux kernel, Virtual File System (VFS) implements a set of data structures, super block, inode, dentry (directory file), and the data block. Each partition on a disk has a super block maintaining the information of the file system of this partition. The information includes a set of uniquely numbered inodes, the number of free and total inodes, the number of free and total data blocks and the state of the file system, which can be either clean or dirty. The inodes is designed to maintain the information of files. Each file has its own inode storing the following file information: address, type, size, owner, references to the files' data block, time stamps of the last file modification and access. Inodes of files can be viewed through issuing command "$ls –I" in Linux shell.

The most concerned problem of current file system is that the gap between CPU and I/O system is being constantly enlarged every few months. To smooth the gap, modern file systems normally maintain a certain sized area in main memory (RAM) to remain those recently used files for future access. This can avoid slow hard disk accesses and increase data response time. The area maintained in main memory is named cache or buffer cache.

Caching strategy or caching policy is responsible for selecting potentially active files to stay in cache and deleting those inactive ones to save expensive memory room. A good caching strategy has relative high hit ratio, which means the files requested have very good chance to be found or hit in the cache.

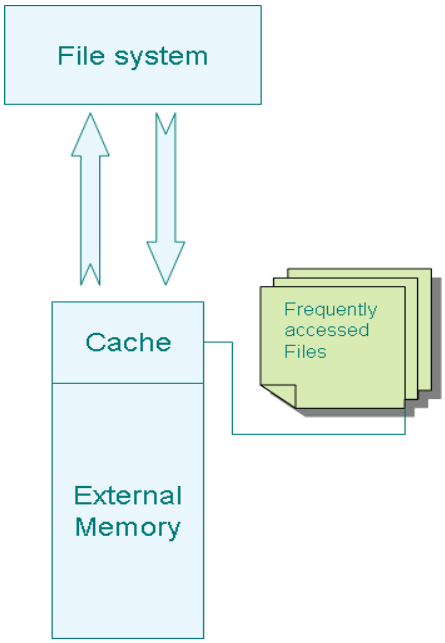


Figure 3.1 A simple file system caching structure

Traditionally, two typical caching strategies are widely adopted in file systems, least recently used (LRU) and least frequently used (LFU). In LRU, a recency recorder is established in caches to fetch the last time of accessing the files. Once the cache is full, the least recently accessed files are evicted from cache area. In LFU, it introduces a frequency counter in caches to record the frequency of files accessed. Once the cache is full, the least frequently accessed files are evicted from cache area. LRU caching strategy works very well in stand alone PC since the access pattern of files and user behaviours are relatively stable compared with server caches and multi-user caches. LFU is better than LRU in first level cache (server cache) environments and multi-user cache (shared cache) environments due to its fairness for the overall performance.

Some dedicated file systems may use different file system policies to maximize their performance. One of the heuristics is the Fast File System (FFS) [25]. It was designed to handle files in different manners regarding to their size. It places small files on disk so that they are near their metadata. Furthermore, it assumes that files in the same directory are

most likely accessed together in a short period.

In addition to file size, other properties such as access type: write-mostly or read-mostly, have been confirmed useful to design various file system policies. For example, Log-structured File System (LFS) [26] proposed by John K. Ousterhout and Fred Douglis, treats the disk as a circular log and writes sequentially to the head of the log. The design of log-structured file systems is based on the hypothesis that write latency is the bottle neck of modern file systems. As a result, a Hybrid file system structure designed against the nature of read and write has been found useful to form high performance file system [27].

3.2 Hard Disk Layout of Storage Systems

Storage systems are very slow component in computer systems compared with other components such as CPU and RAM. This is due to the mechanic feature of hard disks. For each disk access, disk head has to reposition and rerotate the disk plates. With moving the disk arms and rotating the plates, disk head can position at the right track point to access data. The figure below is the inside structure of a modern hard disk.

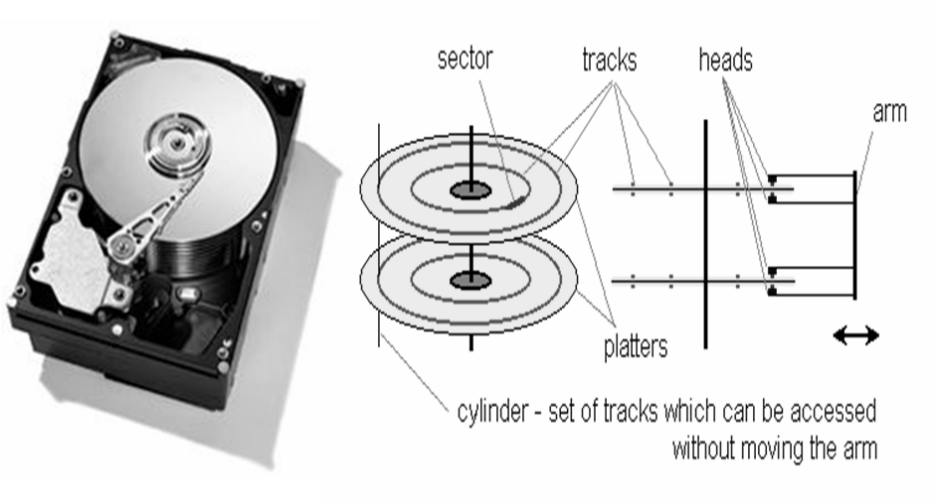


Figure 3.2 An inner structure of hard disk

It is easy to conclude that the data on the outer tracks can be transferred faster than inner tracks due to the relative higher linear velocity. Another fact is that data in the same cylinder can be transferred all together, because disk arms do not need to move during access. Therefore, for certain type workloads, storage systems may manage their data to

gain better performance by exploiting these features.

However, disk requests are always unpleasant compared with data access from caches. To avoid direct disk accesses as much as possible, disks also contain a disk cache, although it is very small compared with file system caches. For a storage system, the miss hit file requests from the file system cache are delivered to storage system cache to check whether the requested data exist or not. In case that both caches do not contain the requested data, the actual disk requests will be issued, which is not the situation what we want to meet every time. Similar to file system caching strategy, LRU and LFU are also the main solutions for disk caches.

Besides disk caching strategy design, many efforts were given to reorganize data layout in hard disks in order to reduce the response time of disk requests. As we can see from Figure 3.2 that during the spinning of the platters, the outer tracks hold higher linear velocity than the inner tracks and the data laid in the same cylinders can be extracted without moving the arm. According to this fact, it is ideal to store frequently accessed data in the outer tracks and store associated data in the same cylinders as much as possible. Most disk re-organization methods are implemented after a long period when the detected system performance (disk requests response time) fells down. Although efficient disk scheduling schemas can leads less impact on system performance, disk re-organization is always a slow process against large volume long-term accumulated shattered data pieces.

3.3 File Access Pattern and Block Correlation

3.3.1 File Access Pattern

File access patterns commonly exist in file systems, especially for PCs. It refers the logical relationships among files requested by users. In application level, programs or software invoke multiple files during their sessions to complete the tasks. For example, once file A is invoked file B may be invoked in a short time since they are always correlated.

To identify file access patterns, data mining techniques were adopted. Griffioen and

Appleton [28] originally developed a probability graph model recording access frequency in a window with a specified length. Each node in the window represents a unique file. If a file is requested, the count of the relative node will be increased by 1. For the subsequent file accesses, the edge connected to that file is also incremented. The model maintains one node for each file, and one edge for two related files. Due to the unordered nature, the complexity of this model is O (n^2). The figure below shows a sample probability graph.

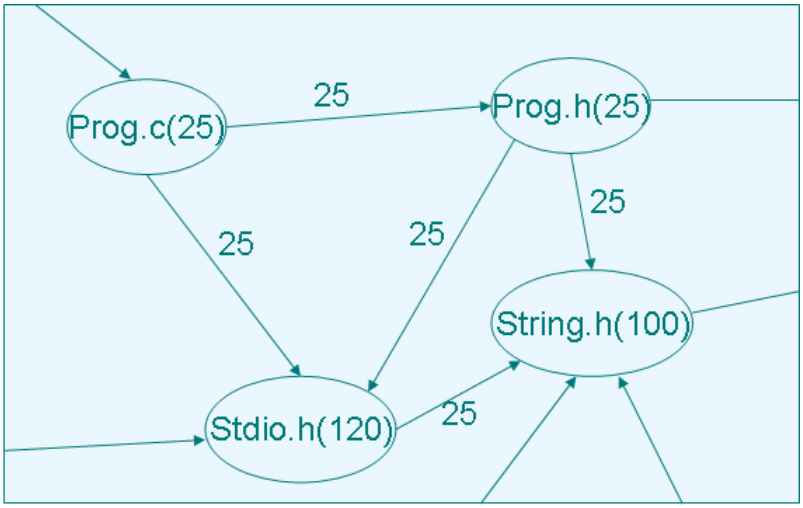


Figure 3.3 A simple probability graph

Another data mining model was adopted to improve the bottle neck of probability graph. Finite multi-order context model [29] was introduced to solve text compression problem. The model uses a trie tree data structure to efficiently store file access sequences. Each node in the tree contains a file name. From the root of the tree to each node, every branch represents a monitored pattern. Figure 3.4 below shows the development of the model when given file access sequence CACBCAABCA.

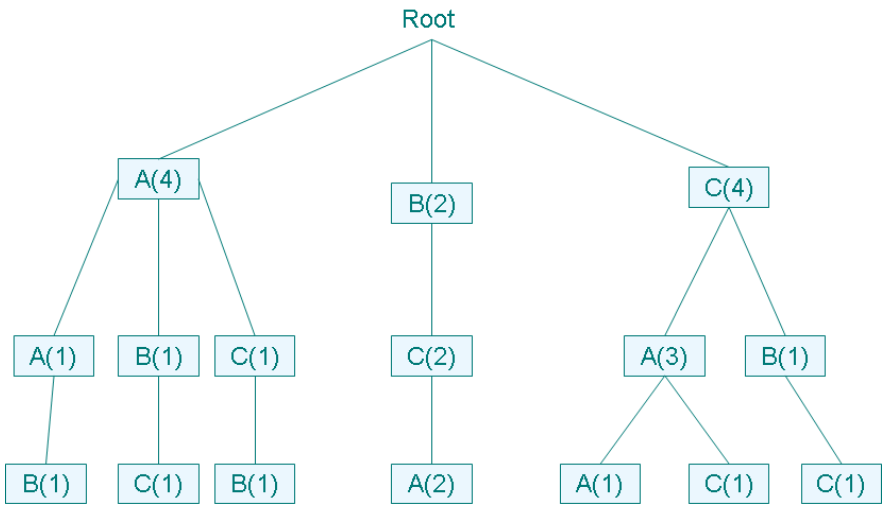


Figure 3.4 A simple tire tree for storing file access sequences

Circled A represents the pattern CA, which has accessed three times. The complexity of this model is $O(n^m)$, where m is the highest order tracked and n is number of unique files. With file access patterns observed, file system can prefetch the related or subsequent files once the first file access is captured. Also, caching related files to caches can increase caches' hit ratio as those files consequently have very good chance to be accessed in future. Furthermore, file access patterns can be used for computer system intrusion detection. When systems detect unusual file access patterns, where sensitive data may be involved, they can stop the access privilege of the relative user.

3.3.2 Block Correlations

Block correlations are common semantic patterns in storage system. Similar to file access pattern, block correlation refers the relationship of data blocks in storage level. Block correlations can be used to improve disk caching, prefetching and disk layout. Z. Li, et al [24] introduced a technique called C-Miner based on a data mining technique, frequent sequence mining, to uncover block correlations in block level.

In storage systems, two or more blocks are correlated if they are semantically linked together. For example, a file's directory block "/dir" is directly correlated to its inode block, which is also correlated to its data block. Compared with file access pattern, block correlations are more stable, because block correlations do not depend on user behaviours

and workloads.

In C-Miner, a full block access trace is split by a pre-determined time gap. This serves the purpose that only closed block access pattern are meaningful to us. Then, it uses a data mining algorithm called CloSpan [30] to find out all the frequent block access patterns within the subsequence of the trace. The observed frequent patterns are converted to readable rules, for example, if ABC is frequent block correlation with 90% confidence, then next time once block A is accessed→block B and C will be accessed with 90% chance. Disk cache can choose to prefetch block B and C into disk cache to reduce two extra disk accesses.

Block correlations can also be used for disk scheduling and disk layout. As we discussed in the section 2, blocks on outer tracks and same cylinders can be transferred much faster than in inner tracks and different cylinders. Therefore, with observed block correlations, we can allocate frequent block correlations on out tracks and same cylinders as much as possible to increase data transfer efficiency.

3.3.2.1 Transaction Filtering ARM Based Block Correlation Mining

Having reviewed the block correlation in storage systems, we propose to utilize the transaction filtering ARM technique to improve the efficiency of mining block correlation. In [24], a long sequence of block accesses is sliced into many relative short sequences according to the time restriction. In fact, however, in some circumstances the separated sequences still contains large number of block access entries, which in consequence form high dimensional block access sequence database.

Bear in mind that the transaction filtering ARM is suppose to speedup the database scanning procedure where large number of infrequent items resides. We broaden the concept of database transaction that each separated block access sequence can be regarded as a database transaction while each block access entry is regarded as a data item in a transaction. Hence, the problem becomes how to find out all the frequent itemsets based on a pre-determined threshold.

Assuming we have a block access sequence: {acdfbcefabcebefef} that can be splitted as a following access sequence database.

Sequence ID	Items
001	acdf
002	bcef
003	abce
004	bef
005	ef

During the scans of the database, each sequence is expected to be filtered by the proposed transaction filter. Rather than scanning the original access sequences, candidate verification process is executed over the filtered sequences in the main memory.

The filter is continually updated after each database scan. For example, the filter-1 in the above sample sequence database is {b,c,e,f} while the filter-2 after the second database scan is {b,e,f}. As the number of filtered items is determined by the size of the given thresholds, therefore higher threshold is higher performance gain can be obtained.

Suppose that the average length of the sequences is L, the given threshold is T and N is the number of access sequences, we can calculate the minimum number of occurrences of a correlation.

$$Min\_\sup = N \times T$$

Based on the Apriori principle: if an item is not potentially frequent, then all its' super sets containing this item are not frequent, any item having occurrences less than $Min\_\sup$ should be eliminated from the original sequence for verification purpose. On the other hand, if we fix the threshold, the number of eliminated candidate correlations rather depends on the size of a sequence since longer sequences normally contain more potentially infrequent items.

Some disk intensive environments such as database server, data hosting server and web server may yield large number of block access sequences within same application sessions, which forms high dimensional block access sequence database. Utilizing transaction filtering ARM technique can reduce the time consumed during the database scanning

procedure.

3.4 File Classification in Self-* storage system

In this section, we introduce another data mining technique: Classification, which was successfully applied to storage systems to classify file's future properties. The idea came from that due to the dramatically growing complexity of computer systems, the system administration cost and data maintenance cost are dominant factors of user dissatisfaction. It was reported that the system administration and data maintenance cost are estimated to be 4-8 times than storage hardware and software costs [7], [8], [9].

Regarding to the above fact, Self-* storage system (pronounced "self-star", the name came from UNIX wildcard character '*') [6] was designed to be self-configuring, self-organizing, self-tuning, self-healing and self-managing. The purpose of self-* storage system is to simplify human administrative works, reduce system cost and enhance system reliability. Ideally, it borrowed data mining technologies to make storage systems automatically realize and recognize the changes of data properties caused by user behaviours and consequently tune themselves.

In traditional storage systems, however, the future data properties such as life time, possible size and read/write dominated operations are monitored and judged by human administrators. To make storage systems tune and manage themselves, self-*storage system employs a static decision tree based classification model to learn and classify files future properties. In self-* storage system, file's future properties are defined as a set of possible categorical values such as read or write dominated operations, life time and size. One may argue that some values may not be continuous and not fit in the requirement of the decision tree. In fact, during the data preparing process, continuous values are pre-discretized according to the range of values. For example, attribute age may be continuous in form of numbers, as required by decision tree model, attribute age must be discretized such as teenager, youth, middle-aged, and the elderly. The classification model is built on a large collection of historical file traces, where records file requests information such as file name, creation time, operation mode and etc.

During the training process, each file attribute are evaluated to confirm its relevance with files' future properties via Chi-Square tests [31]. With such relevance tests, a set of selective attributes are extracted to support file property classification. After then, the selective attributes are used to build decision tree based on ID3 decision tree algorithm. Once the tree is induced, we can determine the class of a file by querying the tree. The values of the file's attributes determine the paths of the tree, which will finally direct to the classes predicted. The figure below shows the ID3 algorithm used to induce decision tree in self-* storage system.

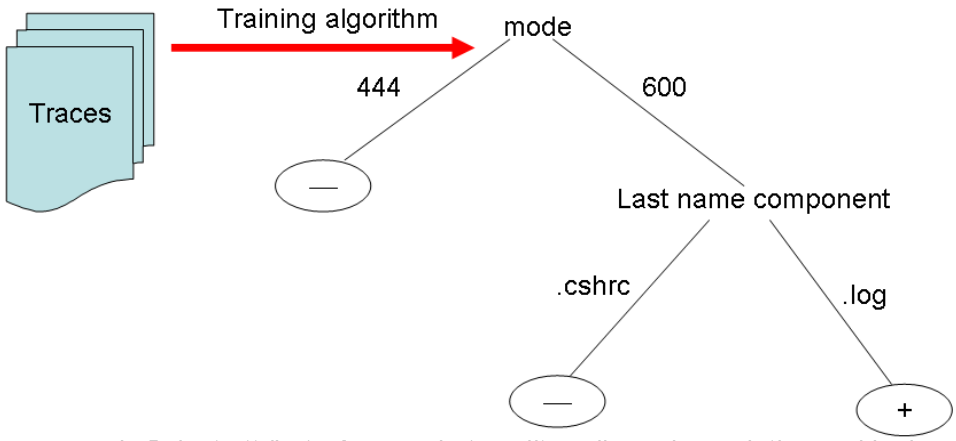


Figure 3.5 ID3 based predictive model developed in self-* storage system

The predicted files' future properties are then used to select proper file system policies or parameter settings when the file is newly created. For example, popular read-mostly files are replicated for load balancing and reduce disk service time. As another, files with short life span can be allocated in none-volatile RAM. The table below shows the example policies against the predicted properties.

File class	Example policy
File size is zero	Allocate with directory
0<size=<16KB	Use RAID1 for availability
File lifespan=<1 sec	Store in NVRAM
File is write-only	Store in an LFS partition
File is read-only	Aggressively replicate

Table 3.1 predicted classes and their relative storage policies configured in self-* storage system [6]

To conclude, classification technique had made positive impacts in self-* storage systems. The ID3 decision tree model can successfully predict file size, life span and dominated operations with sufficient accuracy. However, as the ID3 decision tree is a static model, which can only be built offline. It is noticed that the storage system workloads are rather time series data streams changing dynamically. Compared with the file classification in self-* storage system, our file access frequency predictor dynamically adapts the change of workloads and incrementally updated the prediction model with newly upcoming data. The detail of the model construction will be discussed in Chapter 4.

Summary

In this chapter, two heuristics of applying data mining in file and storage systems are illustrated. File access pattern analysis discovers the correlations between file access to identify the frequent user behaviours, which can be used for intrusion detection, optimal prefetching and so on. Another application block correlation analysis is similar to file access pattern analysis, which utilizes frequent sequence mining technique to discover the correlations among block requests in disk level. The identified block correlation can be used for block prefetching, optimal disk layout and disk security management. The main difference between these applications is that the former is implemented in file system level while the latter is implemented in disk level. In addition, file access pattern analysis identifies the correlations in terms of files, which is more comprehensible to applications and users. In contrast, the utilization of block correlation analysis is more biased to

improving low level system performance.

A decision tree based predictive model is also introduced in this chapter, which benefits intelligent data management in self-* storage system. The model features the ID3 decision tree model built based on file system traces. The outputs of the model can help file system to define various policies in order to achieve intelligent management. Unfortunately, the model is static trained based on only offline traces that do not adapt dynamism of file system workloads. In the next chapter, the heuristic of exploring file access frequency to improve storage system performance will be discussed.

Chapter 4 A Predictive Model Based on Trace Learning

Having discussed the heuristics of adopting block correlations, file access patterns and file's future properties in the previous chapter, it is noticed that file access frequency is also useful to optimize storage system performance. In this thesis, File Access Frequency refers the times of referencing a file during a certain period. File access frequency can be monitored, accumulated and recorded in file system traces. Some applications have already successfully explored the use of data access frequency in literatures.

In this chapter, we first bring out the needs of adopting file access frequency and then raise the question whether or not we can predict files' future access frequency to achieve advanced caching and optimal block allocation. Although many indicators can help file systems to decide its caching strategies, recency and frequency are always the two most commonly adopted parameters of files to design better caching strategies. As we have discussed in the section 3.1, LRU and LFU are the two representatives utilizing these two indicators to achieve caching functionality respectively. In some circumstances, however, recency based caching strategies are no longer suitable as they do not reflect correct file access patterns. Such circumstances are normally referred as server cache environments and multi-user cache environments, where the caches are shared by multiple clients or users.

In [32], experimental results revealed that LFU always performs better than LRU for the file server workloads that do not exhibit strong temporal locality (stable access patterns). Willick, Eager and Bunt [33] have demonstrated that the algorithm named FBR (frequency based replacement) performs better for file server caches than locality based replacement algorithms such as LRU (least recently) which works well for stand-alone system caches. In addition, in multi-level caching environments, the first level cache (client cache) normally maintains recency based caching algorithms which reflect client's stable access

pattern. Interestingly, due to the existence of the first level caches (caches on client side), there is no need for server caches to keep the data existing in client side duplicated. Instead, to serve fast data access for all clients, it is ideal to utilize data access frequency to evaluate the overall usage of requested data from clients.

Besides data caching, in a Hierarchical Storage Management system (HSM), file I/O may be constantly monitored in order to migrate the high frequency files to the fastest storage devices or to internal memory for better performance. HSM was first implemented by IBM on their mainframe computers [34] to reduce the cost of data management and maintenance. However, HSM only acts as an automatic data movement tool that monitors files' access frequency and re-allocate them until their frequency counts exceed a predefined threshold. In fact, these frequent files could be already potentially frequent although they have not met the frequency threshold. For example, suppose a popular singer's latest single is released on a data hosting server. According to our experience, this single is going to attract lots of hits as the singer is suppose to be one of the most popular signers. It makes sense that we do not need to monitor the hits in order to make it fast to be accessed until the hits exceed a threshold. The problem becomes how to decide a piece of data popular or not in advance.

In contrast of frequency monitoring and frequency accumulating mechanisms, this thesis argues that the file access frequency is predictable to achieve advanced file caching and block allocation. As we have discussed in Chapter 4, although traditional file access frequency collection methods can be done precisely via monitoring file requests and accumulating the number of referenced files from system traces, the lack of initial lead makes systems suffer from extra computing overhead and slow data response time.

File systems designers have suggested that the hints of data can help us better understand future user and system behaviours. In fact, the workloads, user behaviours and applications already provide very rich hints to conduct performance optimization. The interesting thing is that some of associations are clear and easily to be identified. For example, assume that a lecturer regularly updates his/her lecture notes in a file hosting server. Clearly, in the next

short period, the files created with UID=”Lecturer A” will be accessed frequently. In contrast, if a student submits his/her coursework to the file server, in the next short period, the files created with UID=”student B” will not hold a good probability of being accessed as frequent as previous one regarding to the popularity reason.

Similarly, file mode can also provide useful information to file access frequency. According to the observation in [6], any file with a mode of 777 in DEAS03 trace has a very short life span (normally less than one second) and contain no data. It suggests that if a file such as a lock file (files that are used as a semaphore for inter-process communication) is created with a mode that enables it readable and writeable for everyone is not going to be read or written by anyone. Therefore, the information hidden behind file access mode can also be used to evaluate for file access frequency.

However, to the best of our knowledge, there is no investigation against file access frequency prediction so far. To identify the associations between the create-time attributes of a file and its future access frequency, we first scan the traces to obtain the historic information and calculate the file access frequency. In the file system traces we used, the attribute “fid” uniquely identifies a file; therefore, we can record the access frequency of a file by accumulating the number of occurrences of “fid”. Once we obtain the access frequency of a file in a short period, say peak hours (10am-1pm), we are able to measure the statistical association between each attribute and the relative file access frequency.

4.1 Trace Analysis

The Traces we used were provided by Jonathan Ledlie [6]. These traces below were taken from three servers that have different user behaviors.

DEAS03 is the workload serving the home directories for faculties and students of the department of Engineering and Applied Sciences of Harvard University. It involves various kinds of network traffics such as Email, research and administrative communication. The DEAS03 trace begins at midnight on 2/17/2003 and ends on 3/2/2003.

EECS03 is the workload serving the home directories for faculties and students of the department of electrical Engineering and Computer Science of the Harvard University. It captures the canonical engineering workstation workload. The EECS03 trace begins at midnight on 2/17/2003 and ends on 3/2/2003.

CAMPUS is the workload serving the home directories for the faculties and students of Harvard College and Harvard Graduate School of Arts and Sciences (GSAS). This workload contains only email traffics. The CAMPUS trace begins at midnight 10/15/2001 and ends on 10/28/2003.

For the privacy issue, some information of these three traces is encoded by using the method described in [35]. The encoded traces contain anonymized UIDs, GIDs, and IP address remapped to the new values. There are no information lost among those identifiers and other variables. Unfortunately, some information about file names, directory names is intended to be hidden for the privacy issue.

Host	read	write	lookup	getattr	access
DEAS03	48.7%	15.7%	3.4%	29.2%	1.4%
EECS03	24.3%	12.3%	27.0%	3.2%	20.0%
CAMPUS	64.5%	21.3%	5.8%	2.3%	2.9%

Table 4.1 file operation statistics of three given NFS traces

Table 4.1 represents the summarized proportions of file operations of the workloads collected from the traces. It shows that the operation mix is different in each workload. In CAMPUS, read and write operations contribute to 85% of the total file accesses while the read operations are dominating the entire workload. DEAS03 contains fewer read/write operations but more metadata requests (getattr, lookup, and access) compared with CAMPUS. However, DEAS03 is still dominated by read operations. In addition, in EECS03, meta-data operations are dominating the workload.

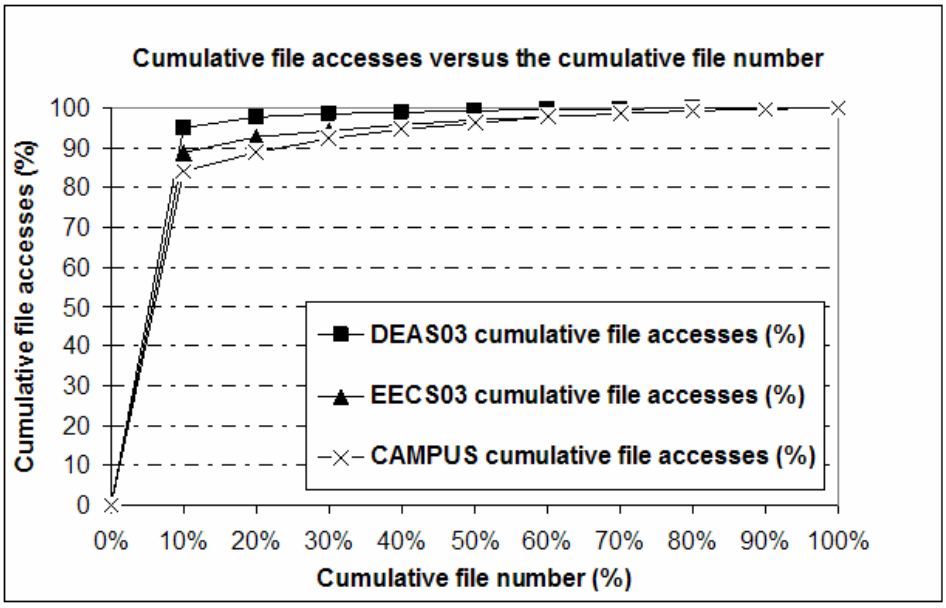


Figure 4.1 The cumulative file accesses versus the cumulative file number.

Both expressed as percentages. It was found that more than eighty percent of the file accesses are absorbed by less than ten percent of the files.

In Figure 4.1, we plot the cumulative file accesses versus the cumulative number of files, both expressed as percentages. The file accesses in our work include read, write, execute, lookup and rename operations. The curve of DEAS03 in Figure 4.1 shows 94.9% of the file accesses going to 10% of the files. In EECS03, 88.9% of the file accesses are going to less than 10% of the total files whereas in CAMPUS 84.1% of the file accesses are going to less than 10% of the total files. CAMPUS is similar to EECS03, but contains a lot of sequential access patterns. We can see that some file accesses are highly skewed and experience shows that only a small fraction of files in a file system is actively and frequently used. As also shown in Figure 4.1, the exact shape of the file request distribution varies from workload to workload.

In the former version of the predictive model, we used a fuzzy logic model to assign a fuzzy coefficient to each file. This is because the frequency boundary and the priority of files of the same frequency class are not clear to the caching policies. However, once a flash memory capacity oriented frequency boundary is clearly defined, we no longer need to consider the files' caching priority based on their previously defined fuzzy coefficients.

4.2 Attributes Selection

To statistically confirm the degree that how strong an attribute of a file relates to the file's access frequency, we use Chi-square test to roughly analyze the association between high access frequency files and each attribute. Chi-square test lets us know the degree of confidence we can have in accepting or rejecting a hypothesis. Concretely, the hypothesis of our Chi-square test is whether or not a high access frequency file's attributes and its access frequency class (high) are associated enough. Given access frequency boundaries calculated from three traces respectively regarding to the 10% top frequent accessed files out of the total files, the Figure 4.2, 4.3, 4.4 shows the confidence of the high access frequency class supported by each attribute over three NFS traces. (Note that the frequency boundaries are vary due to the diversity of traces)

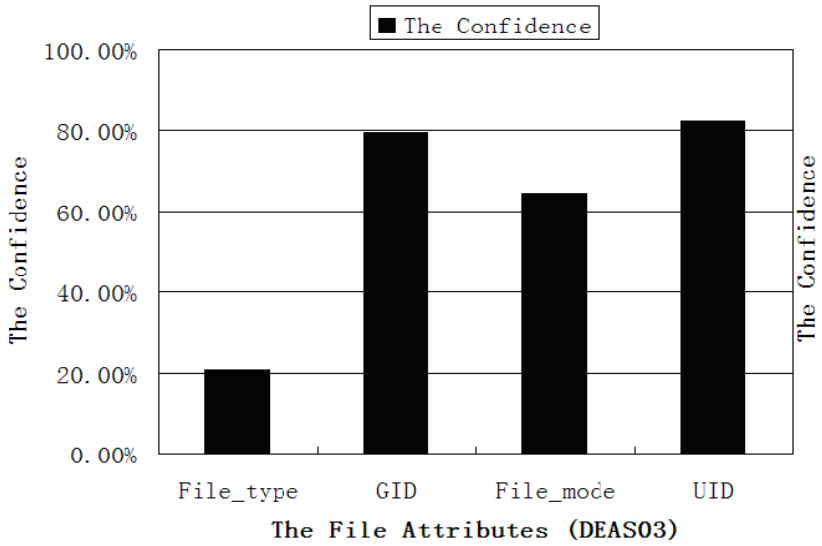


Figure 4.2 The confidence given by various attributes for supporting file access frequency prediction (DEAS03)

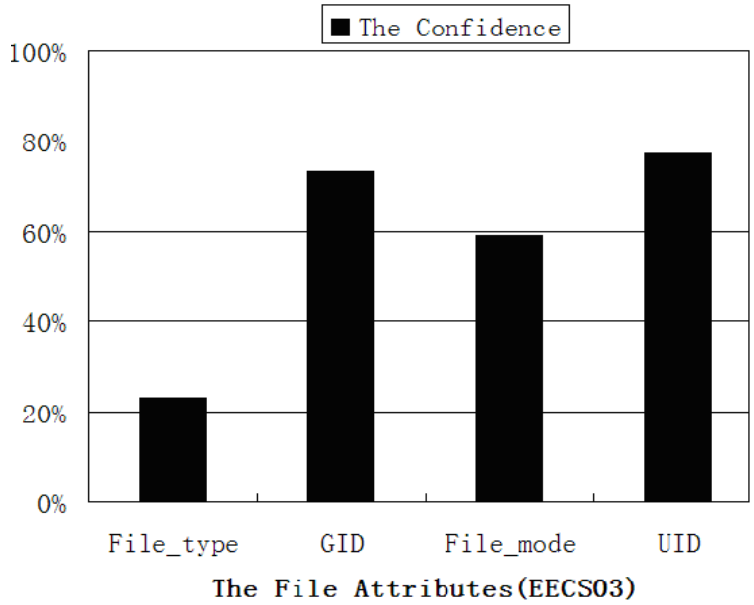


Figure 4.3 The confidence given by various attributes for supporting file access frequency prediction (EECS03)

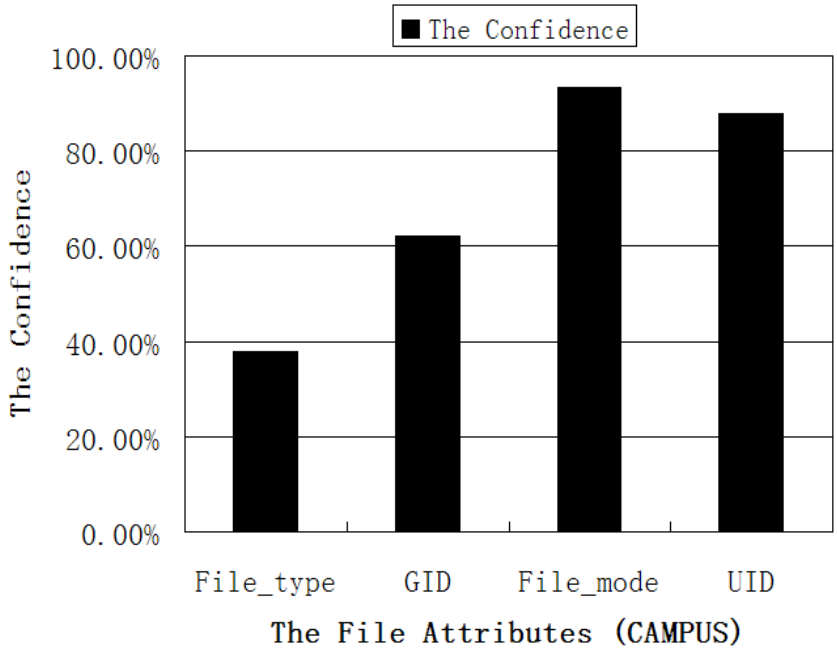


Figure 4.4 The confidence given by various attributes for supporting file access frequency prediction (CAMPUS)

The figures above reveal the confidence of file access frequency supported by the diverse attributes. It can be seen from the figures that the confidence values supported by each

attribute are different in each trace. However, some attributes such as UID always show great confidence to the file access frequency, which is one of the representative cases we have discussed at the beginning of this section.

As the Chi-square test is a rough statistical analysis of the confidence of file access frequency, there is a need to precisely evaluate the degree of confidence supported by each attributes. Here we borrow the concept from information theory, Information Gain, to discover how much contribution an attribute can provide to the confidence of file access frequency prediction. This approach is also used to build a more accurate statistical analysis model, decision tree. In the next section, we discuss our decision-tree based predictive model.

4.3 Incremental Decision Tree Based Predictive Model

The Chi-square test in the previous section has provided us the evidence that some of files' certain attributes are strongly related to files' access frequency. It suggests that once we know a file's attributes, we then can predict a file's frequency class (high or low) with a given frequency boundary. Apart from those representative attributes mentioned above, some other attributes such as the file's creation time, file name and suffix may also support file access frequency. Unfortunately, not every attributes imply positive results supporting file access frequency prediction. From our experiments, we found user ID, Group ID, file type and operation Mode are four common representatives among all three NFS traces.

There are many learning algorithm for data classification, one of the most popular algorithms is the Iterative Dichotomiser 3 (ID3) decision tree [36], which utilizes the information gain to evaluate the confidence supported by an attribute. However , ID3 algorithm trains the model based on offline training samples that do not satisfy the dramatically changes of system workloads. Ideally, to guarantee the model not only confines to stable workloads but also adopts to dynamic changes of the system, we employ the extension of ID3 algorithm: ID5R [37] as our training algorithm to built an incremental decision tree based predictive model dealing with time series file requests streams. We first obtained training sample to build the predictive model, and then validated the model with

new files to check whether or not the predictions are accurate. An N-fold cross-validation technique was used to enhance the reliability of the trained model.

Compared with other supervised learning models, incremental decision tree has a few advantages that are of benefits to our problem. First of all, decision tree is simple to understand and interpret. Due to the change of storage system behaviours, system administrator can easily capture the trends of future file accesses based on the current decision tree's layout. Secondly, decision tree based predictive models are capable to produce very useful insights describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes. Such information makes the prediction results more reliable. In addition, decision tree is transparent (also called white box) model. If a given prediction result is obtained from a decision tree model, then the path of the prediction can be easily tracked back from the bottom of the tree to the beginning of the tree. To the decision tree model, categorical attributes and fixed classes are required. For the first requirement, since file attributes such as file type, file mode, UID, GID are all symbolic represented with no inherent ordering, therefore our classification problem satisfies this requirement. For the second requirement, we can define a caching performance-based frequency boundary to distinguish low frequency and high frequency.

To determine the frequency boundary between High & Low, the capacity of the caching devices is considered. As the implementation of the predictive model is based on a flash memory-oriented storage system that stores frequently used files in a flash memory for caching and prefetching, the boundary of file access frequency is determined in such a way that the amount of I/O caused by accessing High frequency files should fit in the maximal capacity of the flash memory. In other words, the boundary is determined by the flash memory capacity. However, how much percentage of the storage space should flash memory occupies? As shown in Figure 4.1, the average workloads data skewness is 90%. In fact, various workloads may have different data skewness. For example, in our implementation, according to the workloads data skewness, we choose the flash memory with capacity equal to 10% of the hard disk storage capacity. That is, if hard disk is 10 GB, then flash memory is 1GB. Consequently, the file access frequency boundary is determined accordingly.

The files we collected are from three NFS traces: DEAS03, EECS03 and CAMPUS. Through scanning the traces, we can capture the file attributes (file type, file mode, UID, GID) and its access frequency by accumulating the occurrences of the file requests. The following table shows a sample taken from DEAS03, the frequency is regarding to one hour (03/02/2003, 00:00-01:00). According to the average data skewness in these three traces, 599 is set to the frequency boundary between low frequency and high frequency of the sample from DEAS03. Then, we can obtain the frequency class in accordance of their real access frequency

File Type	File Mode	UID	GID	Access Frequency
1	180	18b34	6	7964(high)
1	180	18ad4	6	242(low)
1	180	18aa2	6	299(low)
1	180	18c09	18b3b	567(low)
2	180	18c09	18b3b	21546(high)
2	7ff	0	0	6689(high)
2	5c0	18a89	18a89	7145(high)
1	5c9	18aa2	18ac2	54(low)
2	1c0	18aee	18ad5	3657(high)

Table 4.2 Training sample from DEAS03

The table above presents a training set data taken from DEAS03 NFS trace. The class label ‘Access Frequency’ has two distinct values (high, low). Tree is constructed in a top-down recursive manner starting from the root. At the beginning, records are partitioned recursively based on selected attributes. These attributes are categorized to a fixed number of classes, where continuous values are discretized in advance.

By a given training sample, ID5R algorithm takes all attributes and count their information gain, then chooses the attribute for which information gain is maximum, at last make nodes containing that attribute. From the observation of ID5R algorithm, we noticed that from the top of the tree to the bottom of the tree, the upper level nodes hold larger information gain, which means the upper level nodes that belong to a certain attribute make more contribution for classifying the access frequency.

Assume there are two classes, Y and N:

■Let the set of examples S contain p elements of class p and n elements of class N

■The amount of information, needed to decide whether an arbitrary example in S belongs to P or N is defined as:

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Equation 4.1 The amount of information calculated from a given sample

Assume that using attribute A, a set S will be partitioned into sets $\{S_1, S_2, ..., S_V\}$:

■If S_i contains P_i examples of P and n_i examples of N, the Entropy, or the expected information needed to classify objects in all sub-trees S_i is:

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

Equation 4.2 The Entropy needed to classify objects

■The encoding information that would be gained by branching on A

$$Gain(A) = I(p,n) - E(A)$$

Equation 4.3 The Information Gain can be calculated by minus Entropy from the amount of information

In principle, greater information gain the attribute offers, more contribution it gives for classifying instances. In other words, the decision tree will always choose the attribute with lowest E score as the level zero test attribute.

As an extension of ID3 algorithm, ID5R updates the test attributes in terms of the ordering based on updating the E scores of each attributes. Once a new training instance

is observed, for the attribute A_{new}, the number of negative and positive classes (suppose we have two classes) at the node A changes since the new training instance has brought in new information. Accordingly, the E scores are recalculated and the attribute with lowest E score is pulled up to the level zero.

Figure 4.5 is the decision tree built regarding to the above training sample. It can be seen from Figure 4.5 that begin from the Root, each branch represents a classification rule and its class label. In Figure 4.5, not all rules are populated in the example tree so that the example decision tree is more readable.

File type	File mode	UID	GID	Access frequency
1	180	18b34	6	7964 (high)
1	180	18ad4	6	242 (low)
1	180	18aa2	6	299 (low)
1	180	18c09	18b3b	567 (low)
2	180	18c09	18b3b	21546 (high)
2	7ff	0	0	6689 (high)
2	5c0	18a89	18a89	7145 (high)
1	5c9	18aa2	18ac2	54 (low)
2	1c0	18aee	18ad5	3657 (high)

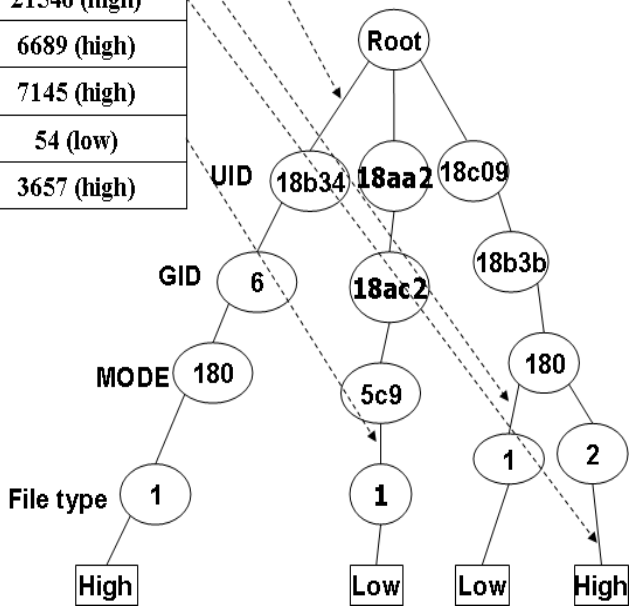


Figure 4.5 The decision tree of the training sample.

IF UID=18b34 THEN Check GID;

IF GID=6 THEN Check File_mode;

IF File_mode=180 THEN Check File_type;

IF File type=1 Then frequency class=high

In addition, as we have discussed that each branch represents a classification rule,

therefore, we then can obtain a Boolean representation of the rules. Thus, for a newly created file, it is very simple to judge its frequency class. One of the branches above can be represented as a Boolean rule. The classifier we built in this section is based on the decision tree classification algorithm ID5R. However, the training sample we used in the example for building the classifier is very small. In fact, to guarantee the accuracy, an appropriate validation method has to be considered. In the next section, we will discuss the validation of the predictive model.

4.4 Cross-Validation

Cross validation technique is used to estimate the error rate of the predictive model. It is sometimes also called rotation estimation, which partitions a sample data set into multiple subsets so that the model testing is implemented on one subset while other subsets are responsible for model training. The partitioned subsets are called training sets and testing set respectively.

When one mentions Cross validation, it actually involves three types of method: Holdout validation, K-fold cross-validation and Leave-one-out cross-validation (LOOCV).

■Holdout cross-validation

Holdout validation is the simplest cross validation method. It partitions the sample data into two fixed sized proportions, for example, training set (2/3), test set (1/3). The advantage of holdout validation is its cheap cost as it wastes the proportion of testing data. In the above example, 1/3 data are wasted from testing as they are not involved in the training process. The disadvantage of Holdout validation is that it sometimes will be misleading if the “unlucky” split of training data and testing data is chosen.

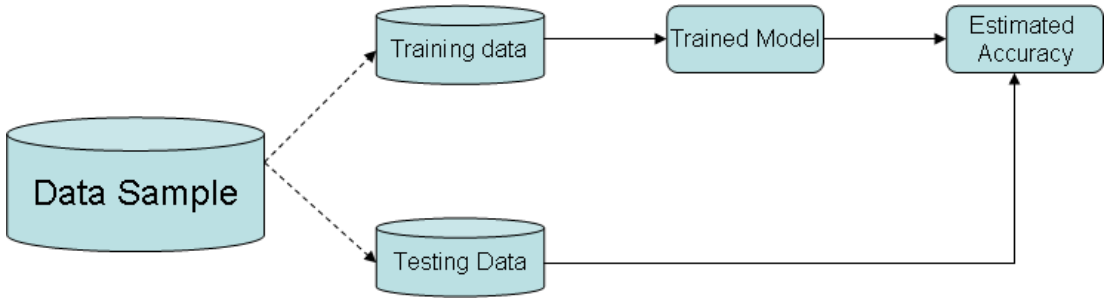


Figure 4.6 Holdout cross-validation procedures

■K-fold cross-validation

K-fold cross-validation divides sample dataset into k subsets that k-1 subsets are used as training set while the other one subset is used as testing set. The advantage of k-fold cross-validation is that all the data samples are eventually used for both training and testing. Compared with Holdout cross-validation, K-fold only has 1/k subset data wasted in a single fold process. In practice, the choice of the number of folds depends on the size of dataset. For very large data set, although three folds normally provide accurate error rate, 10 is the most commonly adopted K value for decision tree cross-validation. The true error is estimate as the average error rate:

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

Equation 4.4 The average error rate of K-fold cross-validation

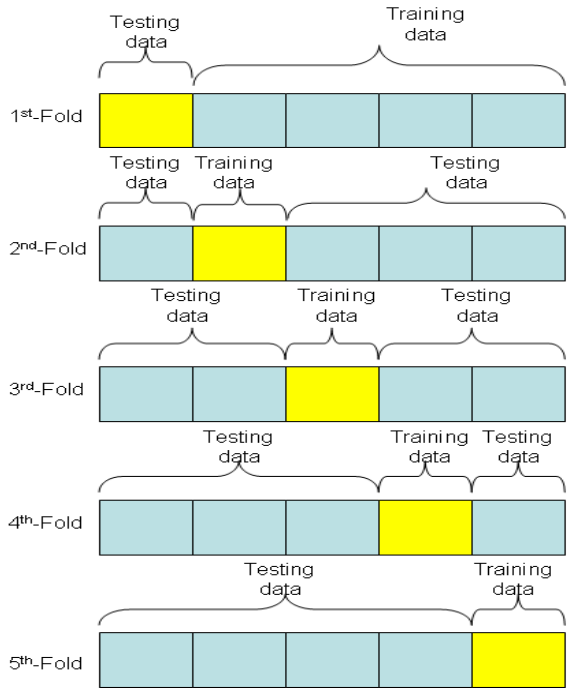


Figure 4.7 K-fold cross-validation procedures

■Leave-one-out cross-validation (LOOCV)

LOOCV is the degenerate case of K-fold cross validation, where K is the number of data instances in sample data. LOOCV takes K data instances as training set and leaves

only one data instance as testing set. By recursively testing all data instances in a data sample, every data instances are eventually involved in both training and testing. Only one data instance is missed during a single training step, which is identical to its name. The true error is estimated as the average error rate:

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i$$

Equation 4.5 The average error rate of Leave-one-out cross-validation

Compared with Holdout and K-fold cross-validation methods, LOOCV is very expensive as it fairly trains and tests all data instances, which could be of a huge amount in a large sample data.

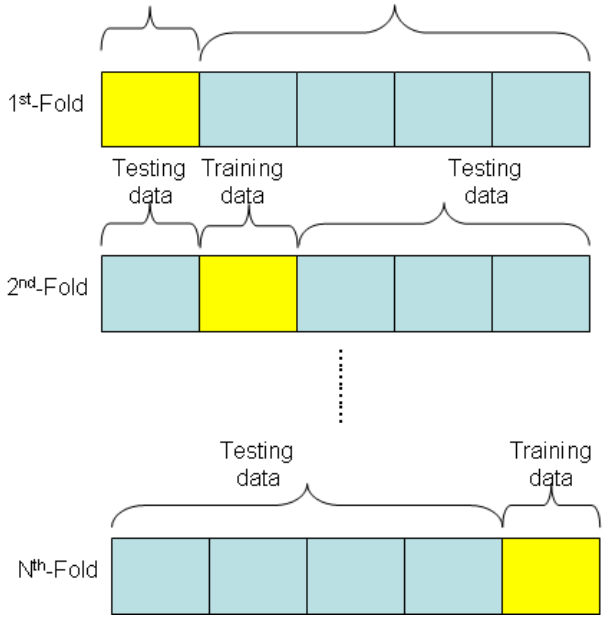


Figure 4.8 Leave-one-out cross-validation procedures

For other time-irrelevant classification problems, training samples and test samples are normally taken from same workloads to guarantee that both samples have same data characteristics. However, in our problem, those traces were fetched in an uninterrupted time series over many days. Thus, each hour's trace may differ from each other in terms of file access frequency. It is impossible to split the whole time series workload into first half and second half to build classifier and validate it. As we can imagine, in peak hours, file access frequency could be much higher than rest of hours.

To evaluate the accuracy of FFP over the time series traces, we built the classifiers based on the peak hours' traces (10am-1pm) on Monday (02/17/2003) as the training sample. Then the model's accuracy is tested by predicting the files created during the peak hours in the following days to check whether or not the precision is acceptable. Therefore, the validation method is an N-fold cross-validation where N is the number of days contained in the traces. The following tables show the accuracy of the predictions during the peak hours on Tuesday, Wednesday and Thursday.

Frequency class	DEAS03	EECS03	CAMPUS
High	90.1%	92.5%	93.3%
low	98.2%	91.2%	90.6%

Table 4.3 Prediction accuracy of FFP on Tuesday

Frequency class	DEAS03	EECS03	CAMPUS
high	88.4%	88.0%	90.1%
low	93.6%	89.6%	87.7%

Table 4.4 Prediction accuracy of FFP on Wednesday

Frequency class	DEAS03	EECS03	CAMPUS
high	82.6%	87.7%	84.1%
low	88.6%	85.3%	82.2%

Table 4.5 Prediction accuracy of FFP on Thursday

Figure 4.9 shows the accuracy curve of FFP from 02/17/2003-03/12/2003. Although the accuracy of FFP slightly decreases, the overall predicting accuracy remains steady during a reasonable long time.

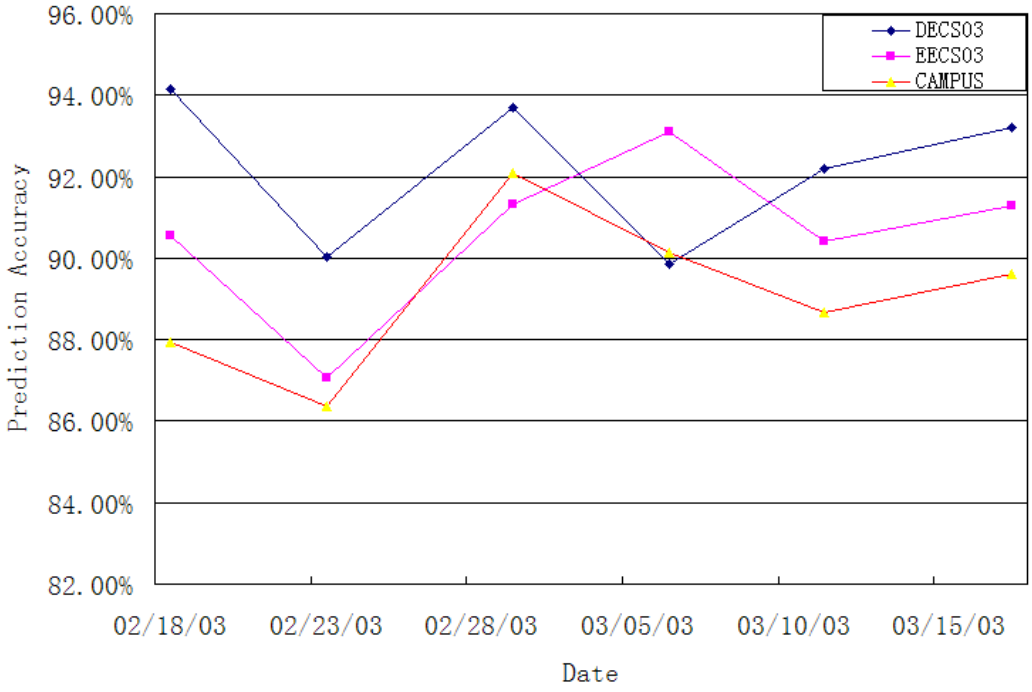


Figure 4.9 Predicting accuracy of FFP over 3 NFS traces from 2003-2-17 to 2003-3-17

The validation results show that the accuracy of the classifier decreases after few days and then climbs up again. This is caused by the slightly changed user behaviours and files' life span. For instance, some lecture notes might be accessed frequently in the next few days after the corresponding lectures were given. However, those lecture notes' access frequency may be no longer frequent once some new lectures are given and their relative lecture notes are created. Nevertheless, due to the dynamic adaptation, the incremental decision tree based model automatically realizes the update and regain sufficient accuracy.

4.5 Comparing the Different Supervised Learning Models

As predicting file access frequency is supervised learning in contrast with unsupervised learning such as clustering technique, we compare the three most widely adopted supervised predictive models in terms of both prediction accuracy and overhead.

In supervised learning (often also called directed data mining), the variables under investigation can be split into two groups: explanatory variables and one (or more)

dependent variables. The target of the analysis is to specify a relationship between the explanatory variables and the dependent variable. To apply directed data mining techniques the values of the dependent variables must be known for a sufficiently large part of the data set. In unsupervised learning situations, all variables are treated in the same way, there is no distinction between explanatory and dependent variables. However, in contrast to the name undirected data mining there is still some target to achieve. This target might be as general as data reduction or more specific like clustering. The dividing line between supervised learning and unsupervised learning is the same that distinguishes discriminant analysis from cluster analysis. Supervised learning requires that the target variable is well defined and that a sufficient number of its values are given. For unsupervised learning, typically either the target variable is unknown or has only been recorded for too small a number of cases.

4.5.1 Decision Tree Vs Naïve Bayes Networks

Naïve Bayes Classifier is based on the Bayes theorem, which is also known as Bayes' rule or Bayes' law. It is a result in probability theory, which relates the conditional and marginal probability distributions of random variables.

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers often work much better in many complex real-world situations than one might expect.

Our experiments were carried out on all the three traces. We used similar validation method for both Decision Tree and Naïve Bayes Classifier. The difference of the validation method compared with the previous Decision Tree-based predictive model validation is that we train the model based on longer period and test the accuracy based on the whole traces. However, due to the change of users' behaviors, both Decision Tree and Naïve Bayes Classifier perform worse eventually after a few weeks. The results in

the following tables are the summarized validation results regarding to the first two weeks of the test traces.

	DEAS03	EECS03	CAMPUS
Decision Tree	94.2%	90.6%	88%
Naïve Bayes	91.7%	85%	83.1%

Table 4.6 Accuracy of the predictive models between Decision Tree and Naïve Bayes Classifier

	DEAS03(1MB)	EECS03(1MB)	CAMPUS(1MB)
Decision Tree	35ms	38ms	32ms
Naïve Bayes	11ms	12ms	9ms

Table 4.7 The time of building the predictive models on 1 MB training data set between Decision Tree and Naïve Bayes Classifier

We noted that the Decision Tree model slightly outperforms Naïve Bayes Classifier by average 5% in terms of the prediction accuracy. We believe that the reason is due to the strong assumption of the attributes' independence to each other. However, as we expected, the simplicity of the Naïve Bayes Classifier makes the model training becomes very efficient compared with the Decision tree model training. Through training 1 MB data from all the three traces, Naïve Bayes Classifier outperforms the Decision Tree model by averagely 4 times faster in terms of model building.

4.5.2 Decision Tree Vs Neural Networks

Neural Networks are another representative of predictive models. They consist of an interconnected group of artificial neurons and process information using a connectionist approach to computation. "In most cases a neural network is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase" [38]. Neural networks are very sophisticated modeling techniques capable of modeling extremely complex functions. The neural network users gather representative data, and then invoke training algorithms to automatically learn the structure of the data. Although the users need to have some heuristic knowledge of how to select and prepare data, how to select an appropriate neural network, and how to interpret the results, the level of user knowledge needed to successfully apply neural networks is much lower than would be the case using some

more traditional statistical methods.

Neural Networks perform very well on difficult, non-linear domains, where it becomes more and more difficult to use Decision trees. However, one of the disadvantages in using Neural Networks for data mining is the slow learning process. Another disadvantage is that neural networks do not give explicit knowledge representation in the form of rules, or some other easily interpretable form. The model is implicit, hidden in the network structure and optimized weights, between the nodes. The following table shows the features of Neural Networks and Decision Tree models.

	Accuracy	Learning Speed	Input	Output	Transparency	Domain Knowledge
Neural Networks	High	Slow	Continuous (0 to 1)	Discrete	Black Box	Essential
Decision Tree	High	Fast	Continuous & Discrete	Continuous & Discrete	White Box	Insignificant

Table 4.8 Predictive model features between Decision Trees and Neural Networks

The Neural Networks training requires continuous input data. This requirement limits the use of Neural Networks since data preparing is a costly process in most of the data mining applications. Besides this shortcoming, the Black Box nature of its structure hardly lets the user track the process of the prediction.

As we have examined the three most widely adopted supervised predictive models, we can conclude that the Decision Trees beat the Naïve Bayes Classifier and Neural Networks in different aspects for our application. First of all, with the consideration of the flash memory hit ratio, we abandoned Naïve Bayes Classifier. Although Naïve Bayes Classifier training is much quicker than Decision Trees, the accuracy is still considered as the most important factor since it is the direct cause of the flash memory's caching performance. On the other hand, the slow learning process and the unexpected extra data preparing of Neural Networks does not fit in the periodical and frequent changes of the system traces.

Summary

Supervised learning methods can be used to extract models classifying or predicting future events. The incremental decision tree based predictive model described in this chapter is responsible for forecasting the future popularity of files, which are associated with files' attributes. The model is built in two stages. First of all, the hypothesis is established based on the principle of interactions between users and file systems, which commonly exists in storage systems but seldom examined by research communities. The hypothesis is then tested through the statistical analysis to prove its existence. Secondly, based on the tested hypothesis, the predictive model is built through training file system traces recording historic user behaviors. The trained model is validated by the N-fold cross validation to test its accuracy.

It is noted that trace skewness plays a very important role for model construction. As the observed distribution of file requests in the workloads is not even, one can inspect trace skewness by statistically populating the overall distribution of different user behaviors such as read, write and lookup. In our study, the results of trace skewness analysis show that 90% of total file accesses contribute to only 10% of total files. With such a given skewness statistic, the relative file system policies can be concluded and implemented.

In addition, the decision tree model is also compared with the other two supervised learning models: Naive Bays Networks and Neural Networks in aspects of accuracy and efficiency. The comparison results suggest that both Naive Bayes Networks and Neural Networks are not suitable for file access frequency prediction due to their disadvantages in various aspects. Specifically, Naive Bays Networks is removed from our candidate list due to the strong assumption of its' attributes independence and slightly lower accuracy. On the other hand, the slow learning process, unexpected extra data pre-processing cost and the "black box" nature of neural networks make it inappropriate for our application.

In the next chapter, the built model will be deployed in a hybrid file caching environment featuring server caching circumstances. Combined with the analyzed trace

skewness, a flash memory conducted hybrid caching hierarchy is established. The flash memory introduced in the design is supposed to store the predicted high frequency read-only files to reduce the data access latency.

Chapter 5 Case Study I: File Access Frequency Conducted Hybrid File Caching for NFS

The decision tree based predictive model built in the last chapter has shown sufficient accuracy that can be used to forecast a newly created file's future access frequency. Having revealed that traditional stand-alone system caching strategies such as LRU and LFU are not suitable for server environments, where the stable access patterns hardly exist, this chapter proposes to adopt the proposed decision tree based predictive model to benefit server caching performance. The idea came from the conclusion that file access frequency is the better indicator for server caching compared with recency.

However, the current access frequency collection methods are either supervisory or accumulating, which do not provide advanced estimations suggesting proper changes. In this chapter, the proposed predictive model is employed to predict file access frequency in network file system environments, where the server workloads represent multiple users and multi-level caching environments. A hybrid caching hierarchy is established in the design that utilizes flash memories as the secondary external storage components to achieve fast file caching.

The basic principle is to store the potentially frequent files in a flash memory since the data response time of flash memory is much faster compared with hard disks. However, flash memory has its own drawbacks, that is, the life cycle of flash memory rather depends on the number of write operations executed. In order to identify the specific situations that flash memories can contribute to overall performance improvements, it is necessary to study the performance indices of both flash memories and hard disks. In the first section of this chapter, we compare the performance for both flash memories and hard disks in terms of data response time. Next, the proposed hybrid caching

hierarchy will be presented in section two based on the previous analysis. The section three is the analytical study of the improvements of caching performance according to different situation of data layout. The last section is the discussion of the simulated results of the hybrid caching design.

5.1 Flash memories vs. Disks

Flash memory is non-volatile computer memory that can be electrically erased and reprogrammed. The term non-volatile stands that it does not need power to maintain the information stored in the chip (compared with static random access memory SRAM). Flash memory has two major advantages compared with hard disk: fast read access time and better kinetic shock resistance. These two advantages make flash memory popular in portable devices such as digital cameras.

When we mention flash memory, it normally refers two types of flash memory based on different architectures: NOR and NAND named after the NOR logic gate and NAND logic gate. NOR flash memory has faster read speed (compared with NAND) and random access capabilities enabling it suitable for code storage devices such as Personal Digital Assistant (PDAs) and mobile phones. However, write and erase operations of NOR flash memory are slower compared to NAND. Furthermore, the size of NOR flash memory does not scale as good as NAND flash memory since NOR has very large memory cell size restricting scaling capabilities.

In contrast, NAND flash memory offers faster write/erase capability and slower read capability than NOR. In general, the fast write/erase speed, scalability of capacity and its' low cost make it the favoured technology for file storage. Offering the functionality of rewriting data quickly and repeatedly, NAND flash memory is typically adopted for placing large volume of information in devices such as Flash drives, MP3 players, multi-function mobile phones, digital cameras and USB drives.

In the following analysis, we use Samsung NAND flash memory (K9F6308U0A) and Seagate Cheetah ST173404LC Ultra160 SCSI hard drive. The tables below are the basic performance parameters from manufacturers.

Performance Table	
Transfer Rates	
Internal Transfer Rate (min)	28 Mbits/sec
Internal Transfer Rate (max)	427 Mbits/sec
Formatted Int Transfer Rate (min)	26.700000 Mbits/sec
Formatted Int Transfer Rate (max)	40.200000 Mbits/sec
External (I/O) Transfer Rate (max)	160 MBytes/sec
Avg Formatted Transfer Rate	35.5 MBytes/sec
Seek Times	
Average Seek Time, Read	5 msec typical
Average Seek Time, Write	5.2 msec typical
Track-to-Track Seek, Read	0.6 msec typical
Track-to-Track Seek, Write	0.9 msec typical
Average Latency	2.99 msec
Other	
Default Buffer (cache) Size	4,096 Kbytes
Spindle Speed	10,000 RPM
Configuration	
Number of Discs (physical)	12
Number of Heads (physical)	24
Total Cylinders	14,100
Bytes Per Sector	512

Table 5.1 Performance characteristics of Seagate Cheetah ST173404LC Ultra160 SCSI hard drive

Type	K9F6408U0A
Program Page Size	(512 + 16)Byte
Block Erase Size	(8K + 256)Byte
Read Page Size	(512 + 16)Byte
Random Page Read	10μs(Max)
Sequential Page Read	50ns(Min)
Page Program time	200μs(Typical)
Block Erase Time	2ms(Typical)
Endurance Program/Erase Cycles	1Million

Table 5.2 Performance characteristics of Samsung NAND Flash Memory (K9F6408U0A)

First of all, let us examine the data access time of hard disks and flash memories. The average disk access time is:

$$T_{disk\_access} = T_{seek} + T_{rotational\_delay} + T_{transfer} + T_{controller\_overhead}$$

Equation 5.1 The average disk access time [73]

Where T_{disk_access} is the average disk access time; T_{seek} is the average disk seek time; $T_{rotational\_delay}$ is the average rotational delay; $T_{transafer}$ is the data transfer time and $T_{Controller\_overhead}$ is the time consumed by disk controller during the data access. These performance parameters can be obtained from manufacturers as shown in table 19. The average time to read or write 4KB block in a Seagate Cheetah ST173404LC Ultra160 SCSI Drive is

$$5ms + \frac{0.5}{10{,}000RPM} + \frac{4KB}{40.0MB/\sec} + 0.1ms = 5.0 + 3.0 + 0.096 + 0.1 = 8.196ms$$

Equation 5.2 The average time of reading or writing 4KB block in a Seagate Cheetah ST173404LC Ultral60 SCSI Drive

Assuming the measured seek time is one third of the calculated average, the measured average disk access is

$$5/3ms + \frac{0.5}{10{,}000RPM} + \frac{4KB}{40.0MB/\sec} + 0.1ms = 1.67 + 3.0 + 0.096 + 0.1 = 4.866ms$$

Equation 5.3 The measured average disk access of reading or writing 4KB block in a Seagate Cheetah ST173404LC Ultral60 SCSI Drive

We note that the disk transferring time takes only 0.3% of the total disk access time. Therefore, disks spend most of their time on waiting for the disk head to get over the data rather than transferring or writing the data.

NAND flash memory reads data in the unit of page, while erases and writes data in the unit of block containing a certain amount of pages. The time of reading 512bytes, which is the default page size of Samsung NAND Flash Memory (K9F6408U0A), is 10μs. To read 64KB in flash memory, it simply divides the 64KB into 128 pages.

$$\frac{64KB}{512bytes/page} \times 0.01ms = 1.28\text{ms}$$

Equation 5.4 The time of reading 64KB data for a Samsung NAND flash memory (K9F6408U0A)

However, for disk read and write, there are two situations that should be considered. The first situation, all blocks on the disk are evenly random; the second situation is that all blocks on the disk are sequentially mapped. In addition, different file system uses various minimal block sizes as their input unit from the disk. For example, EXT2 (second extended file system) file system uses 1KB, 2KB, 4KB, and 8KB as its input unit. Now, let's consider reading 64KB blocks from disk under the first situation and the default block size is 4 KB. Reading 64KB is to read 16 blocks with the default size 4KB. Under the first situation, all the blocks are evenly mapped on the disk. Therefore, each disk access needs to reposition the head and re-rotate the plates. The total access time for reading 16 blocks is:

$$16 \times 4.866\text{ms} = 77.856\text{ms}$$

Equation 5.5 The time of reading 64KB random blocks in a Seagate Cheetah ST173404LC Ultral60 SCSI Drive

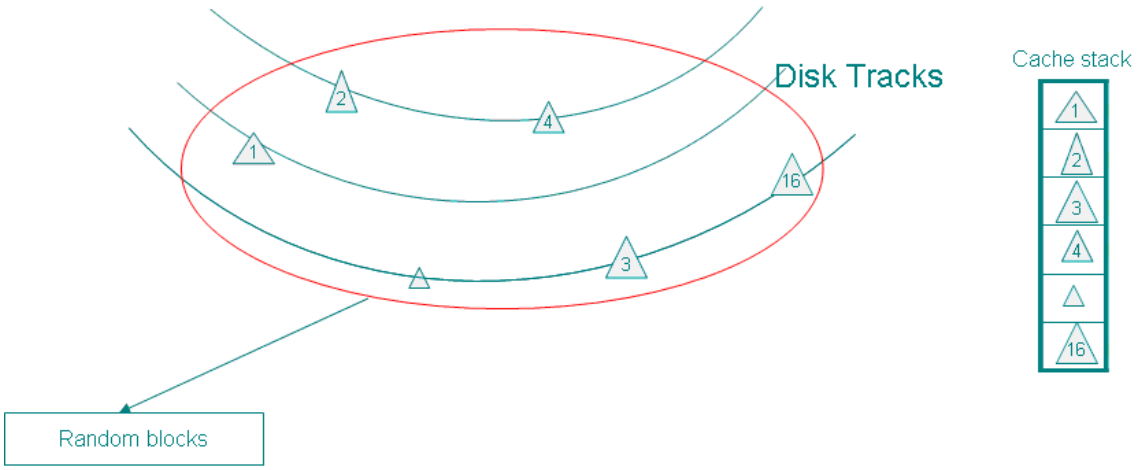


Figure 5.1 A demonstration of 16 random blocks

Under the second situation, these 16 blocks are sequentially mapped. So, only considering the change of data transfer time, the total access time for reading 16 blocks is:

$$5/3\,ms + \frac{0.5}{10{,}000\ RPM} + \frac{4\,KB}{40.0\,MB/\sec} \times 16 + 0.1\,ms = 6.306\ ms$$

Equation 5.6 The time of reading 64KB sequential blocks in a Seagate Cheetah ST173404LC Ultral60 SCSI Drive

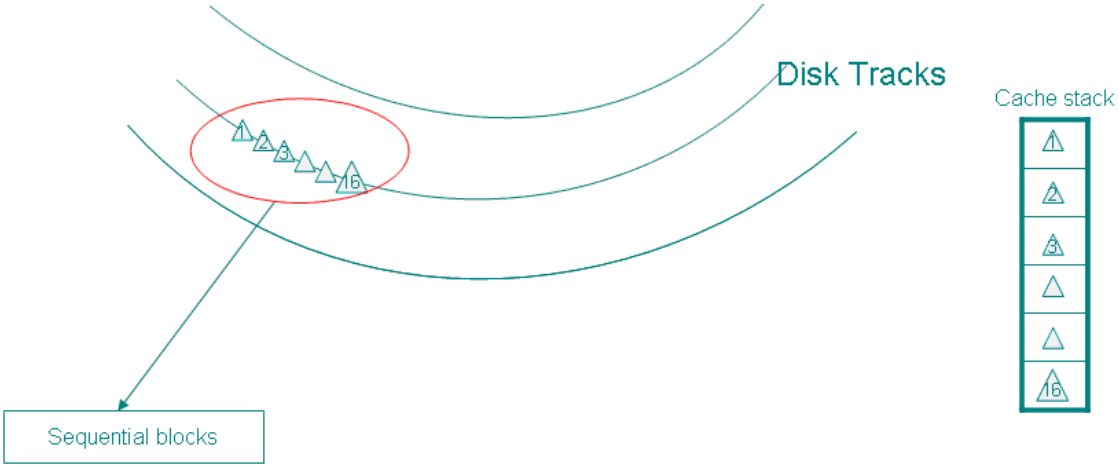


Figure 5.2 A demonstration of 16 sequential blocks

We can conclude that the time of reading 64KB from disk with 4 KB default block size is in a range of 6.306ms-77.856ms, which represents the perfect disk layout and the worst case respectively.

However, writing 64KB in flash is quite different. First of all, it erases 64KB by divide 64KB into 8 blocks (note that the block here is different with file system block) with default size 8 KB of Samsung NAND Flash Memory (K9F6408U0A). Then, writing 128 pages of the flash memory as we discussed above.

$$\frac{64KB}{8KB} \times 2ms + \frac{64KB}{512bytes/page} \times 0.2ms = 41.4ms$$

Equation 5.7 The time of writing 64KB data in a Samsung NAND flash memory (K9F6408U0A)

5.2 Hybrid Caching Hierarchy

Server such as file server, database server and web server are designed to provide services for multiple clients and maintain the data consistency and integrity. In order to improve the services, server usually preserves a large buffer to cache data. Muntz and Honeyman [32] investigated server caching in a distributed file system, showing that server caches have poor hit ratios. The poor hit ratios are resulted by the data sharing among multiple clients. This study raised the question whether the stand-alone buffer management policies works well for server caches. Willick, Eager and Bunt [33] have demonstrated that the Frequency Based Replacement (FBR) algorithm performs better for file server caches than locality based replacement algorithms such as LRU (least recently used) and LFU (least frequently used) which works well for stand-alone system caches.

However, embedding the frequency statistics into the server caching algorithm takes extra overhead. As we discussed in the previous section, if the heuristic is dynamic or wrong, this method can result degrade of system performance.

In the proposed design, such a hybrid caching architecture is built: between file system buffer cache and disk, a flash memory is added as an external secondary storage device storing frequently accessed files of a certain period. The file system buffer cache still retains the traditional caching replacement algorithms such as LRU or LFU. The architecture aims to provide faster data transfer of those highly frequent used data from

flash memory in contrast with caching data from disk.

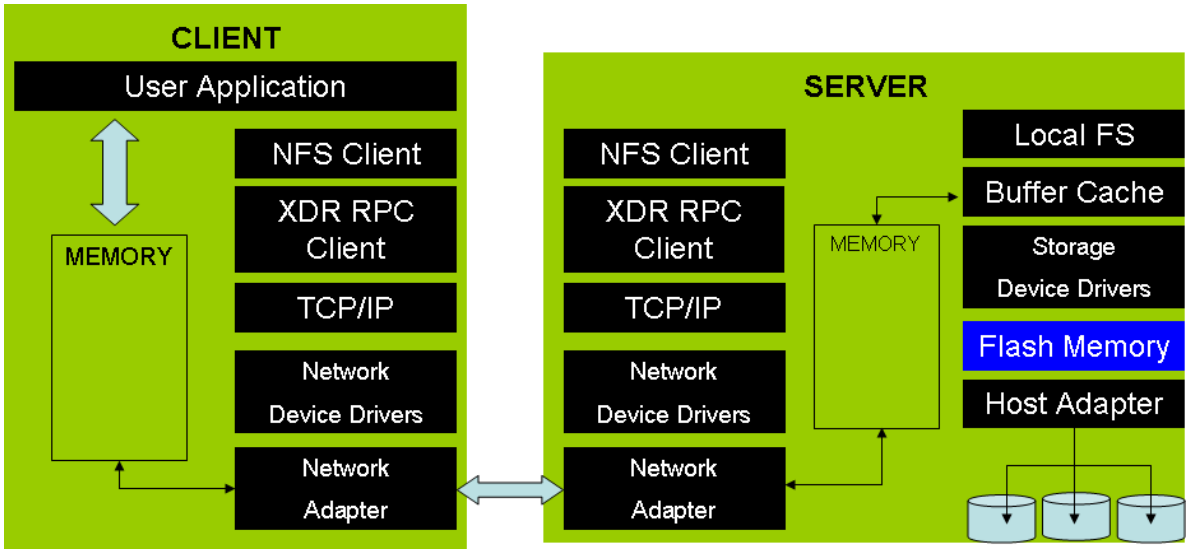


Figure 5.3 Flash memory-oriented hybrid caching architecture based on NFS

In this architecture, server buffer cache is shared by multiple users. As the traditional caching replacement algorithms such as LRU and LFU do not perform well in server buffer cache. For example, LRU algorithm replaces the block that has not been used for the longest period of time. It is based on the observation that blocks which haven been referenced in the recent past will likely be referenced again in the near future. Nevertheless, the assumption no longer exists in server buffer cache.

As we have analyzed that flash memory is much slower in writing (approximately 6 times) than disk accesses. The flash memories are not supposed to store frequently write type files. However, another crucial reason also suggests that flash memories should serve read-only file caching since the erase cycles of flash memories are normally 1 million times shown in table 20. Not just flash memories' life cycle, for some write dominant workloads such as EECS03, the hybrid caching scheme may suffer a performance degradation analyzed in section 3. Furthermore, the write back nature of file systems may destroy the flash memory in some write intensive workloads.

Based on the above discussion, we conclude that our hybrid caching scheme is very efficient in read dominant workloads. The flash memory in this architecture should serve read-only file caching. The calculated performance speedup based on the

frequency prediction model is 2.8~5 compared with the base case.

5.3 Analytical Performance Study

In this part, we investigate the Hybrid File System Caching performance by applying our Decision Tree-based predictive model.

The hybrid file system uses a flash memory-oriented caching scheme by extending EXT2. A new device driver for flash memories has been written, equipped with garbage collection and other flash-related management strategies. A Samsung 4G x 8 Bit NAND Flash Memory is used as both the write buffer and boot buffer. Every write operation is associated with an erasing time penalty. As well known, space on flash memory already written (programmed) with data cannot be overwritten unless it is erased. Erase must be performed over a relatively large space. A NAND flash memory is organized in terms of blocks, where each block is of a fixed number of pages. A block is the smallest unit for erase operations, while reads and writes are processed in terms of pages. The block size and the page size are (8K + 256) Byte and (512 + 16) Byte, respectively.

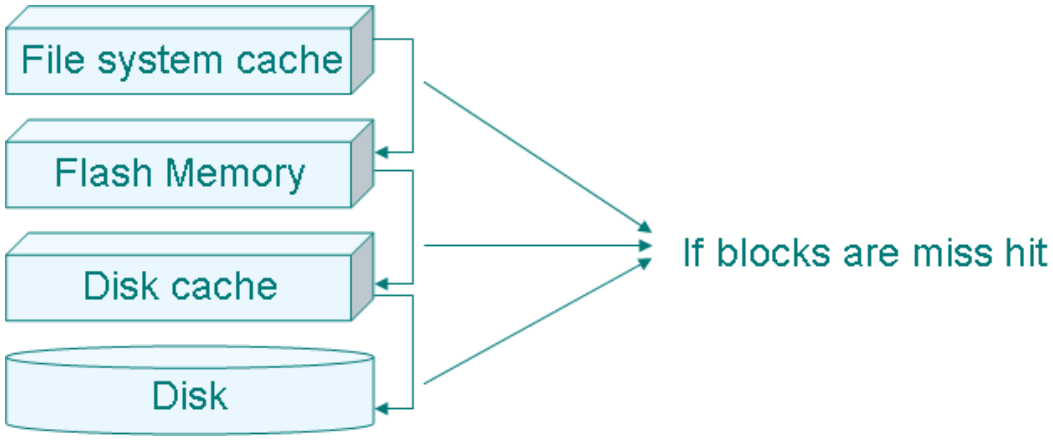


Figure 5.4 The flash memory oriented caching hierarchy

Based on our predictive model, highly frequent files can be predicted with 90% accuracy. In other words, if we move the highly frequent accessed files into flash memory, the miss hit blocks of buffer cache can be hit in flash memory with 90% hit ratio. Considering the trace skew, the actually migrated blocks to flash memory is 90%*the number of access over 10% top frequent files.

We first only consider the read operation under the first situation that all blocks are randomly mapped on the disk. Suppose that H is the disk cache hit ratio; R is the average size of read operation; N_R is the number of read operations; P is the prediction accuracy of the predictive model; $N_{skewness}$ is the number of file access of 10% top frequent files; T_{Flash_read} is the time of reading 4KB data in a flash memory; T_{Disk_read} is the time of reading a single 4KB block in a hard disk. The time T_1 of reading miss hit blocks from flash memory and the time T_2 of reading miss hit blocks from disk:

$$T_1 = (1-H)\times R\times N_R \times \{P\times N_{skewness}\times T_{Flash\_read} + (1-P\times N_{skewness})\times T_{Disk\_read}\}$$

Equation 5.8 The time of reading miss hit blocks from the flash memory [78]

$$T_2 = (1-H)\times R\times N_R \times T_{Disk\_read}$$

Equation 5.9 The time of reading miss hit blocks from the disk [78]

We can compare the total execution time as following:

$$\frac{T_1}{T_2} = \frac{P\times N_{skewness}\times T_{Flash\_read} + (1-P\times N_{skewness})\times T_{Disk\_read}}{T_{Disk\_read}}$$

Equation 5.10 The ratio of total execution time of reading miss hit blocks from the flash memory and reading miss hit blocks without flash memory (directly from the disk)

We can see that the performance speedup mainly relies on the flash hit ratio, which is affected by predictive model accuracy. In the following part, we calculate the overall performance of the hybrid caching scheme based on 3 NFS traces.

Cache Size	LRU
8MB	6.9%
16MB	13.8%
32MB	20.7%
64MB	25.7%
128MB	35.4%
256MB	60.6%

Table 5.3 LRU algorithm hit ratio of a NFS server buffer cache

	EECS03		DEAS03		CAMPUS	
Total operations	176,933,945		577,751,752		655,637,109	
Read operations	4,2994,948	24.3%	281,365,103	48.7%	422,885,935	64.5%

Table 5.4 Total operations of three NFS server traces

Table 5.3 shows the LRU (least frequently used) algorithm based caching hit ratio of a typical NFS server buffer cache. Table 5.4 shows the ratio of the read operation out of entire traces. Set the buffer cache size 256MB and the average block size of each read operation is 64KB. Under the first situation, all the blocks are randomly located on the disk. The ratio of the total execution time of all the read operations can be calculated.

EECS03:

$$\frac{T1}{T2} = \frac{81\% * \text{Flash\_read} + 19\% * \text{Disk\_read}}{\text{Disk\_read}} = \frac{0.0648 + 0.92454}{4.866} = \frac{1}{5}$$

T1/T2=1/5

Under the second situation, all these blocks are sequentially located on the disk. The total execution time of all the read operations can be calculated:

$$\frac{T1}{T2} = \frac{81\% * \text{Flash\_read} + 19\% * \text{Disk\_read}}{\text{Disk\_read}} = \frac{16 * 81\% * FLASH\_READ + 19\% * DISK\_READ}{DISK\_READ}$$

$$= \frac{1.0368 + 1.198}{6.306} = \frac{1}{2.8}$$

T1/T2=1/2.8

To only serve read operations, data response time takes big advantages from fast flash read. The results show that even all blocks are sequentially located on the disk, flash caching still outperforms disk caching by 2.8 times. The range of caching speedup is from 2.8~5 representing the worst and best disk situation.

Using the same setup, we can calculate total execution time of all the write operations under the first situation.

EECS03:

T1/T2=1/1.6

Under the second situation, all these blocks are sequentially located on the disk**.** The total execution time of all the write operations can be calculated

T1/T2=5.7

We can see from the calculated results that writing data in flash memory is much slower (around 6 times) than writing in disk. Even blocks on the disk demonstrate the worst locality, only 1.6 times speedup it can achieve compared with writing in hard disk.

To calculate the total execution time of both the read and write operation, we still use the same setup and discuss two different situations. We also assume that the average read and write sizes are same.

Under the first situation (blocks are randomly located on the disk):

$$T1/T2=\frac{read\_percent*(81\%*Flash\_read+19\%*Disk\_read)+write\_percent*(81\%*Flash\_write+19\%*Disk\_write)}{read\_percent*disk\_write+write\_percent*disk\_write}$$

$$=\frac{0.24+2.99}{4.866}=1/1.5$$

Under the second situation (blocks are sequentially located on the disk):

$$T1/T2=\frac{0.552+27.199}{6.306}=4.4$$

The above calculation demonstrates that the performance speedup of the hybrid caching scheme is 1.5~-4.4. It suggests that for all kinds of file operations, hybrid caching scheme does not perform very efficient compared with normal architecture. This is because the EECS03 trace contains large potion of write operations which result in performance degradation.

5.4 The Discussion of the Measurable Benefits

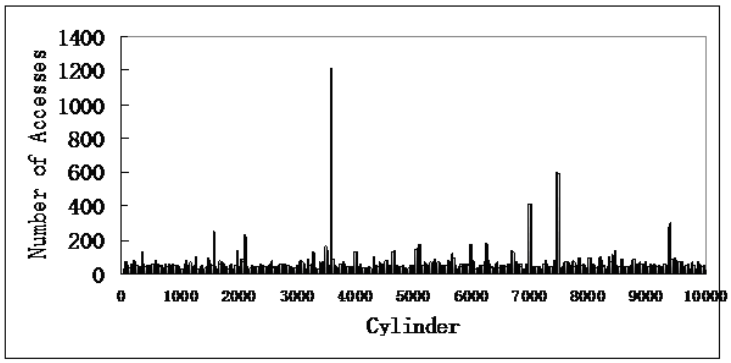
This section describes a deployed Evolutionary Storage System (EvoStor) with measurable benefits. It is a practical product with an embedded File Frequency Predictor (FFP). Degradation in performance over time is a serious and common concern in magnetic disks. Sometimes, people benchmark their disks when new, and then many months later, and are surprised to find that the disk is getting slower. In fact, the disk most likely has not changed at all, but the second benchmark may have been run on tracks closer to the center of the disk [39]. To counteract the above disk

devolution process, this section proposes to develop a FFP-plugged evolutionary storage system that is evolving over time.

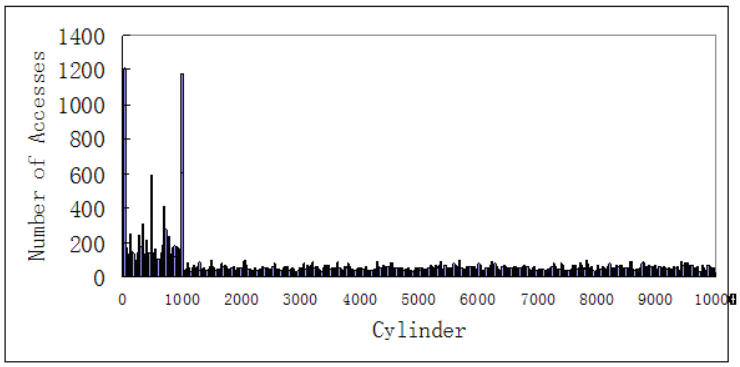
In Chapter 4, we found that more than eighty percent of the file accesses go to less than ten percent of the files. If the blocks storing hot (frequently accessed) files are spread over the surface of the disk, distant from each other, long seek delays may be caused. According to the trace skewness, we need only allocate a small percent of the data (files) to a fast device such as a flash memory based on the predicted frequencies by FFP to realize most of those performance gains. The EvoStor system combines NAND-based Flash with rotating storage media. The ultra-high-density benefits of magnetic storage technology are preserved, while the ultra-low-power, ultra-high-reliability and fast read/write access of advanced NAND technology enhances the overall value of the EvoStor system. Most traces exhibit a great deal of skewness in file request distribution and therefore the fast flash memory can be made very small (in this experimental setup it is 10% of the disk capacity) without severely impacting the performance of an adaptive disk.

The allocation may be imperfect -- files need only be near their optimal location. The hope of placing frequent files close to each other is that this will significantly reduce the disk seek time between accesses: If it is likely that consecutive accesses are to hot files, then it is likely that they will be to files in the flash memory, reducing the expected disk seek time. The disk rotational delays may also be reduced, if it is likelier that accesses will be to the same flash memory. As we will see, the improvements due to both effects can indeed be very significant.

Figure 5.5 shows a typical cylinder access distribution over 25,000 requests on a Quantum Atlas disk. In most environments, disk accesses display significant spatial and temporal locality. Indeed, as the narrowness of the peak in Figure 5.5(a) shows, only a small number of cylinders are accessed with any significant frequency. Figure 5.5(b) shows the effect of allocating the most frequently-accessed files predicted by FFP together into the flash memory with a capacity of 10% of the disk space (note that the entire storage space is mapped as a continuous linear range in most file systems).



(a) As measured on the original layout;



(b) After the predicted frequency is used to decide what files to keep in a flash memory (10% of the total space).

Figure 5.5 The predicted frequency information could be used to decide what files to keep in a flash memory, measured on a Quantum Atlas disk.

The Evolutionary Storage system allocates frequent files predicted by FFP to a flash memory. This is a “White Box” or “Gray Box” design in which a predictor passes semantic information of file frequency with a prior probability to a storage system when the file is created. Unlike the above, a “black box” design infers statistically object frequency with a posterior probability in a disk drive, which may fall short in terms of re-locating the recorded files. Using the three NFS system traces, the experimental results showed that the performance speedup compared with the base case without NAND flash memory is 3.9. It is a conceptually simple technique for dramatically improving disk performance. As a light-weight solution, the predictor can run on the same CPU using their spare CPU power without causing too much CPU and memory overhead. Because frequency distribution is relatively stable, it is unnecessary to keep swapping the files between the flash memory and the disk.

Summary

Instead of monitoring and accumulating file access frequency to capture the change of a file's popularity, the developed predictive model is deployed in server environments to achieve the advanced file popularity prediction. In this chapter, a flash memory conducted hybrid caching hierarchy is introduced and simulated in a network file system. Due to the superiority of data response time and low cost-per-byte, the flash memory is supposed to be a secondary storage device remaining the predicted high frequency files to serve fast caching. According to the trace skewness in the given traces, approximately 90% of total file access fall into only 10% of total files, which indicates that these 10% files are the most active ones. Combined with the predictive model, the number of the predicted high frequency files stored in flash memory is determined by both the prediction accuracy and trace skewness.

Unfortunately, according to the performance analysis, writing files in flash memory is rather a slow and expensive choice compared with hard disk writing. The analytical study in this chapter considers two extreme situations of hard disks and then evaluates the performance improvements boosted by the designed caching hierarchy under these two situations. A range of performance speedup 2.8-5 is concluded from considering read-only files kept in the flash memory. The simulation was conducted based on replaying the three NFS traces with the hierarchical caching design and the base case. The result showed a 3.9 times performance improvement, which is identical to the theoretical range of performance improvements. As we can see from the original disk layout (Figure 5.5.a) in the simulation, only small amount of cylinders are frequently accessed during the replay. Rather than directly accessing these files from the disk, the plugged flash memory is used to reallocating and grouping these frequent files for fast caching (Figure 5.5.b).

In the next chapter, the practical utility of the developed predictive model is extended to the Data Grid environments. Utilizing predictive model to predict the popularity of Grid files can help Data Grids in deciding the creation and allocation of file replicas. Although file replication in Data Grid is similar to file caching in file system, the

ultra-large scale of the Data Grid requires a balance among various performance indices to satisfy the entire Grid.

Chapter 6 Case Study II: Predictive File Replication on the Grids

6.1 File Replication on the Data Grids

Data Grids are responsible for dealing with large scale data communication and management. Many scientific and engineering applications require access to large amounts of distributed data (terabytes or petabytes). The size and number of these data collections has been growing rapidly in recent years and will continue to grow as new applications and scientific experiments come on-line.

Current large-scale Data Grid projects include the Biomedical Informatics Research Network (BIRN), the Southern California Earthquake Center (SCEC), and the Real-time Observatories, Applications, and Data management Network (ROADNet), all of which make use of the SDSC Storage Resource Broker as the underlying data grid technology. These applications require geographically distributed accesses to the data by many people around the world. Data Grids create virtual collaborative environments that support distributed but coordinated scientific and engineering research.

In Data Grids, jobs are implemented by invoking files from local or remote Grid nodes. Files needed for finishing jobs are stored in storage elements (SEs) where computing elements (CEs) may also be assigned. Moreover, computing elements are responsible to execute jobs by requesting relative files from the Grids. In general, to handle a submitted job, three kinds of resources are considered, computing resources, storage resources and networks. Besides Grid resources, Grid scheduler is responsible to arrange jobs based on the current state of the Grid resources.

Replicating files among Grid nodes is due to the distributed nature of the Data Grids where geographically distributed data lead to access latency. However, an appropriate file replication strategy can reduce access latency by creating file replicas spreading

over demanding nodes. Meanwhile, file replication strategies are also required to bring the needs to optimize the Grid resource usage for the whole Grid user community. Consequently, to evaluate a file replication strategy, both total job time and resources usage are considered in order to achieve the balance between single user and the whole Grid.

Most replication methods either monitor the popularity of files or use complicated functions to calculate the overall cost whether or not a replication decision or a deletion decision should be issued. However, the replication decision issued in normal instances is in some sense "late" considering the detected changes of files' popularity. When the replication decision is issued, the truth is that the popularity of the files had changed and may have already given impact on access latency and resource usage. This chapter proposes a predictive file replication strategy that forecasts files' future popularity based on their characteristics on the Grids. Besides file replication strategies, scheduling methods also influence the overall performance of the Grid jobs.

6.1.1 File Replication Strategies and Simulation Tools

Various replication strategies have been proposed by research communities so far. Least Frequently Used (LFU) and Least Recently Used (LRU) are most widely adopted methods, which maintain an access history on each Grid site to monitor the popularity of files. The LRU and LFU strategies will always replicate files to local sites where computing elements request the files for executing the jobs. Through browsing the replica catalogue, where the detail information of all the replicas is stored, it chooses the replica that can be accessed within shortest time. The difference between LFU and LRU is that when the local storage is full, deleting methods are different. LFU deletes the file least frequently accessed while LRU deletes the file least recently accessed in order to create space for new replicas. The simplistic of LRU and LFU make them successful and popular for file replication strategy design, however, the "always-replicate" policy leads to unbalance between job time and resource usage.

The Economic model [40], [41] was developed to estimate the file values, which are

used to decide files' future popularity, based on binomial distribution. The model assumes that the popularity of files obeys certain distributions, and then finds the file value where it lies on that distribution. The Economic model also employs a reverse auction strategy to message Grid sites in order to obtain the "cheapest" replicas. This strategy models the local sites and remote sites as replica "buyers" and "sellers", where the value of file replicas is evaluated by their historic "purchase" prices. According to the Economic model, a typical replication process is modelled as an investment on the market. If the candidate replica has greater value than the least-valued file in the local storage, then the least-valued file is then replaced by the new replica. Similar like LRU and LFU, if local storage is not full, replication decision will always be issued.

Besides replication strategies, many Grid simulation tools have also been developed recently. ChicagoSim [42], [43] was designed to simulate various replication and caching strategies. EDGSim [44] was developed to evaluate various scheduling algorithms vastly influencing the over all data Grid performance. Without considering replication issue, their study showed that the spatial locality of Grid data makes huge impacts on scheduling decisions. Another simulator called GridSim [45] was also developed to examine different scheduling algorithms by employing an economy model to emulate the purchase and selling of Grid resources in order to make appropriate use of Grid resources.

OptorSim [46], [47] is a simulator focusing on emulating a two-staged optimization that replications during the run-time of jobs benefit from the scheduling decisions pre-determined, which are based on both data spatial locality and network status. Compared with other simulation tools, the advantage of OptorSim is that it takes into account both scheduling and replication optimizations in aspects of the dynamism of the workloads, the spatial locality of Grid data and network status. Relying on such staged design, one can either purely evaluate the replication strategies or emulate the impacts given by other factors. Considering the advantages of OptorSim in simulating replication strategies, we employ OptorSim as the simulation tool to implement the proposed replication method.

Different simulation environments consist of various Grid topologies, the components of Grid sites and the parameters describing the dynamism of the Grids. In OptorSim, a Grid has several sites, where may contain computing elements (CEs) and storage elements (SEs). Computing elements act as computational resources by running the jobs submitted, which involve the files stored in storage elements. If a site contains zero computing elements and storage elements, it is regarded as a network node or a router. Besides these, network bandwidths among nodes are also simulated as well as background traffics. The most important component to our method is the replica manager implemented in each site. It manipulates replica selection, creation and deletion through communicating with replica catalogue where the detailed information of replicas is recorded.

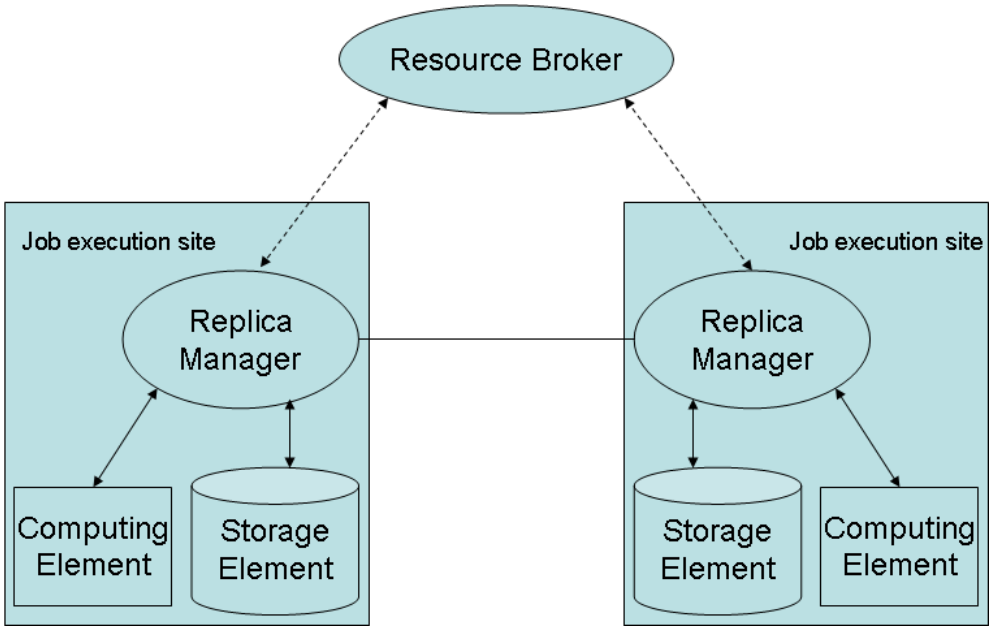


Figure 6.1 A diagram of EU Data Grid Components

In each site, a particular file is established to maintain the access history of the files in order to monitor the file access frequency and their temporal locality. For LRU and LFU replication strategies, access history is examined to decide which file to delete from the local storage in order to create space for new replicas.

6.2 The Predictive File Replication on the Data Grids

As we have discussed that the heuristics from file systems suggest that the file attributes

always imply hints that can be used to improve system caching performance. In fact, however, file replication on the Grids is similar with file caching in file systems. The common thing is that both techniques create copies of data to fast areas (main memory for file systems and local storage for the Grids) to reduce access latency (I/O latency to file systems and network latency to the Grids).

With the similarity between file caching and file replication in mind, the hypothesis that file characteristics are related to its future popularity on the Data Grids is raised. For example, a file created or last modified by certain users such as project managers or system administrators may have very good chance to be accessed in near future. If such correlations exist, then we can choose appropriate model to predict files' future popularity. Unfortunately, most Grid simulation tools run the simulation on generated synthetic jobs and files, which are virtually defined without realistic file characteristics such as user ID, creation time and etc. Hence, in order to find out the correlation between file characteristics and files' future popularity, we must analyze traces containing real-system file requests.

In my experiment, simulations employed real network file system traces to extract both file characteristics and observed file popularity. Then, chi-square tests were conducted between extracted file characteristics and observed file popularity to examine whether or not file characteristics have enough confidence to support our hypothesis. In addtion, to describe files' popularity, accumulated file access frequency are observed and recorded to maintain file access frequency history.

The traces used were obtained from three network file systems DEAS03, EECS03, and CAMPUS. These three workloads trace various Network Appliance Filters that serve the home directories for students and faculties of different departments of Harvard University. Table 6.1 shows the typical format of the file requests contained in one of the three traces, DEAS03.

Created time	Source address	Destination address	File type	Mode	User ID	Group ID	File system ID	File ID
1044248400.4	31.03ff	30.0801	2	180	18c09	18b3b	8664	4b8966
1044248400.8	83.03fd	30.0801	2	7ff	0	0	8664	1423391
1044248400.8	83.03f9	30.0801	1	5c9	18aa2	18ac2	8664	1423393
1044248400.9	66.03ff	30.0801	1	180	18b34	6	8664	1423395

Table 6.1 Trace sample of DECS03

A set of Chi-square tests were established in order to decide whether a certain attribute provide enough confidence supporting our hypothesis. The testing results in Chapter 4 have shown that for these three network file system traces, four attributes have shown very good confidence, which reflect the association between attributes and observed file access frequency. It can be seen from Figure 4.2, 4.3 and 4.4 that the confidence values supported by each attribute differ in each trace. However, attribute UID always show greater confidence to the file access frequency compared with others, which is one of the representative cases we have discussed.

We then import the file system traces into OptorSim through grouping file requests based on their high level sessions. However, Chi-square tests can only examine the existence of our hypothesis, but it can not tell us how to infer files' popularity relying on their attributes. Figure 6.2 represents the cumulative file accesses along with the number of files, which are ranked according to the access frequency from the highest one to the lowest one. This diagram shows the skewness of all the three trace-sample we extracted as the workloads for the simulations. Note that each workload is equivalent to a three-day trace from DEAS03, EECS03 and CAMPUS respectively. In Figure 6.2, DEAS03 exhibits approximately 80% file accesses contributed by 20% top frequent files, while EECS03 and CAMPUS show 87% and 82% file accesses absorbed by 20% files respectively.

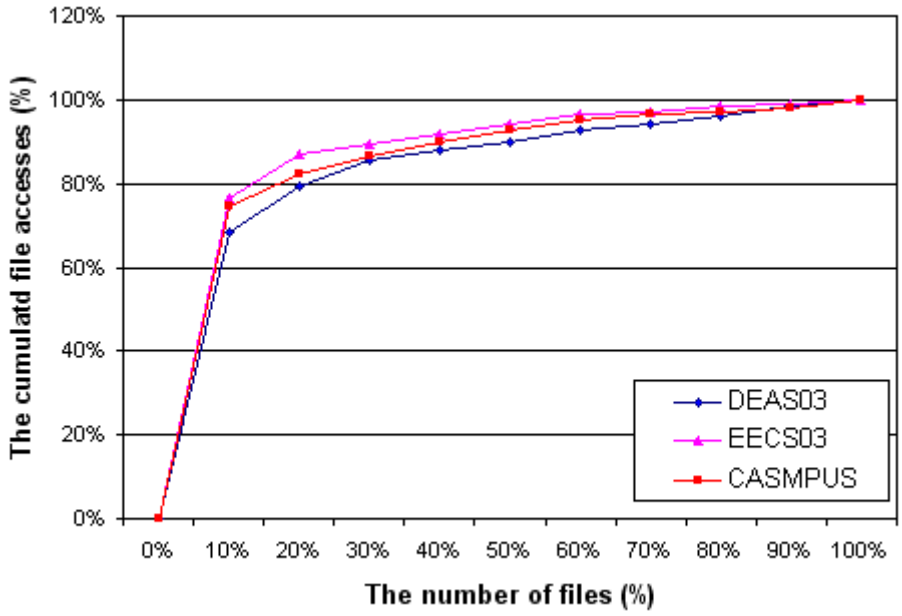


Figure 6.2 The cumulative file accesses versus the number of files

In order to examine the contribution of each file attribute for predicting file popularity, Incremental decision tree are adopted as a more accurate predictive model due to its transparency and dynamism. The ID5R incremental decision tree was used as a predictive model to obtain predicted future access frequency of files. When a file is requested by a remote Grid node, the model will calculate the estimated frequency rank based on the predetermined classification policies. Similar with LRU, replication will always be executed as long as there is still enough space available for new replicas. If the local storage is full, it will compare the least frequently accessed file with the new replica to see whether or not the future access frequency of the new replica would likely exceed the least frequent one. If the access frequency rank of the new replica is more than the least frequent one, file replication decision will be issued or vice versa. Compared with “always-replicate” method, the model only replicates those who will be most likely accessed frequently in future. The ID5R decision tree is dynamically updated due to the continuously monitored file requests. The replication decision is made by interpreting with the decision tree model to see whether the requested replication should be issued or not.

In the simulation, we define the jobs as a set of continuous file requests sent from same

high level sessions in these three traces. A job list can then be generated by regrouping the file requests according to their high level session ID observed in the traces such as source address ID in DEAS03. To ensure that these file requests belong to same applications, a time slice window is established to restrict the distance of file requests. According to the testing results, 30-minute is the average gap separating discontinuous request sessions. Figure 6.3 shows the interaction of various Grid components for the predictive file replication. Besides this, the definition of files is also modified by adding additional file characteristics that we have extracted.

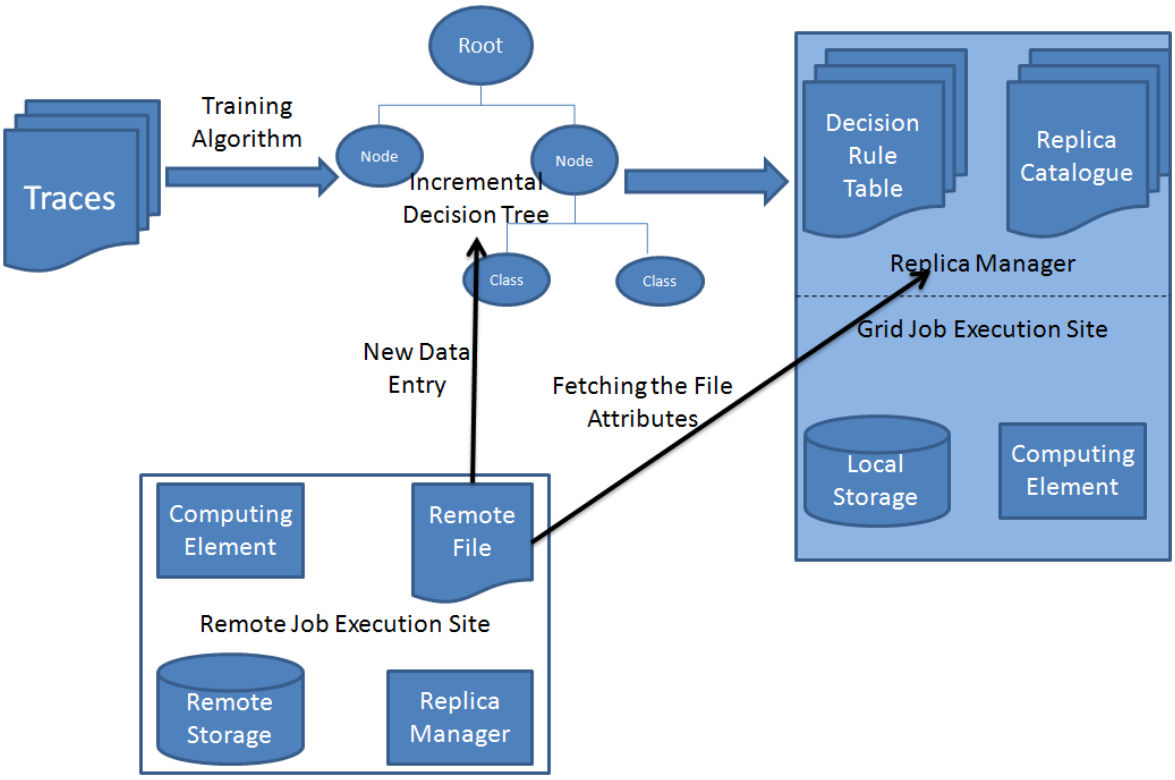


Figure 6.3 Grid components of the predictive file replication

If a local SE still has enough space for new replicas, the requested files are always replicated since the jobs would be repeatedly executed many times in future. Once the local SE is already fulfilled, replica manager will first fetch the characteristics of the files and check the decision rule table converted from the decision tree model. If the predicted rank of future popularity is “higher”, the replica manager will then replicate this file to the local storage. Regardless the prediction results, decision tree will always take the new file requests as the new data entries to update itself. If the predicted rank of future popularity of the new replica is “lower”, the local CE will look up the replica

catalogue to find the remote replica that can be accessed within shortest time by using remote I/O.

6.2.1 Replica Selection Based on Access Cost

We suppose that each file in local storage elements has a certain access frequency rank, which is determined by the trained predictive model. Assuming that each file is represented as an n-tuple form:

$$F_i = (t_i, n_i, i_i, att_{i1} ... att_{ij}, R_i)$$

Where t_i is the time stamp at which the request was filled in the resource broker, n_i is the logical name of the requested file, i_i is the index number of the file, att_{ij} are the attributes of the requested file such as owner id, group id and so on, and R_i is the predicted access frequency rank.

Replacing an existing file replica is based on the prediction results that decide whether or not the requested file's rank is higher than the lowest ranked replica in the local storage elements. In order to obtain the replica in the shortest time, the network link capability and the distance of the remote sites holding the replicas are considered. The link capability is represented as the bandwidth among nodes and the distance can be simulated by adopting the network access delays. The overall access cost of accessing a remote file replica is modelled as following:

$$C = d + \frac{B}{S}$$

Where d is a fixed cost of initiating a message from one site to another, known as access delay, B is the number of bits transmitted over the network and S is the transmission rate. As we want to find the replica offering smallest C value, the site storing the replica should provide enough information such as file size and the queue length.

6.3 Simulation configuration

The simulation is based on OptorSim simulation environment. OptorSim specifies Grid

topologies, Grid jobs, Grid files, file access patterns and scheduling algorithms in its configuration file to enable flexibility of emulating various replication strategies.

Grid Topology-In the simulation, the European Data Grid (EDG) topology was adopted as the simulation testbed. EDG testbed contains eighteen Grid nodes distributed in seven countries. It was developed mainly for High Energy Physics, Earth Observation and Biomedical applications. The topology below shows the structure of EDG testbed and the link capability among nodes.

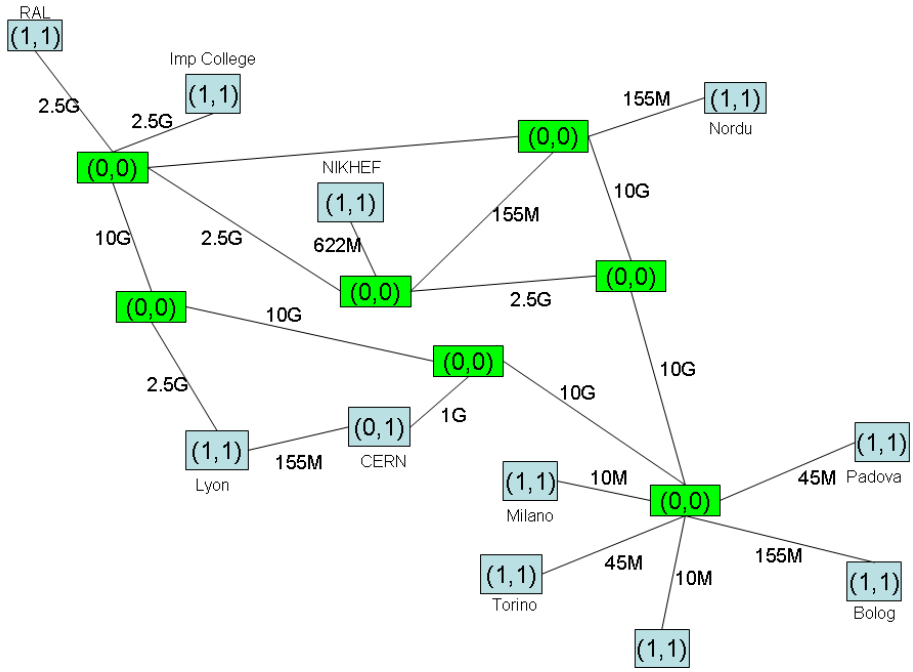


Figure 6.4 The network topology and configuration for the EU Data Grid testbed

Figure 6.4 shows the topology and network connectivity among nodes of the EDG testbed. Each node may contain one computing element and one storage element represented in a paired value. For example, node CERN contains zero computing elements and one storage element and thus is represented as (0, 1). The nodes containing zero computing elements and zero storage elements are treated as network routers shown as the green rectangles. The link capabilities between nodes are represented by the network bandwidth shown in the above figure.

Simulated Jobs-The simulated workloads are imported from the three network file system traces, DEAS03, EECS03 and CAMPUS, so that we can evaluate correlations

between real file characteristics and files' popularity. The jobs are defined as a set of file requests generated from same high level applications or sessions. A time slice window (30 minutes) is specified to avoid discontinuous file requests of different high level applications.

The Grid jobs are described in a job table along with a file table. The following example shows a Grid job configuration.

```
#
# File Table
#
#name, size (MB), index, mode, type, UID, GID
#
\begin {file table}
File1 1000  1    180   1    18b34    6
File2 1000  2    180   1    18ad4    6
File3 1000  3    180   1    18aa2    6
File4 1000  4    7ff   2    0        0
File5 1000  5    5c0   2    18a89    18a89
File6 1000  101  5c9   1    18aa2    18ac2
File7 1000  102  1c0   2    18aee    18ad5
File8 1000  103  180   1    18aa2    6
File9 1000  104  180   2    18c09    18b3b
\end
```

Figure 6.5 A typical file table of EU data Grids configuration

```
#
# Job Table
#
# job name, files required by it
#
\begin {job table}
Job1 File1 File2 File3 File4 File5
Job2 File6 File7 File8 File9
\end
```

```
#
#CE Schedule Table
#Site, jobs it will run
#
\begin {CE schedule table}
2 job1 job2
7 job1 job2
\end
```

```
#
#Job Selection Probability
#
\begin {job selection probability}
job1  0.8
Job2 0.2
\end
```

Figure 6.6 A typical job table of EU data Grids configuration

As we can see from the above example, the files for executing a job are pre-assigned before the simulation starts. Note that pre-assigning jobs to a particular CE does not bond the jobs permanently during the scheduling process. Differently, it only defines the CEs that are willing to run the relative jobs. In a similar manner, the Grid jobs are also described with their selection probability.

In Figure 6.5, the first column of the file table denotes unique file names followed by their size in MB and indexes. The third column describes the file modes while the other three columns represents file type, user id and group id respectively. Figure 6.6 shows the job table, CE schedule table and the job selection probability table. In the job table, the files needed to be executed follow the job name that requests these files. In addition, the CE schedule table represents the Grid sites that are able to run the jobs.

During the simulation, various numbers of jobs are simulated for certain purposes. However, which job should be chosen for a particular CE during the run-time is critical to the simulation results. The job selection probability specifies the probability that certain jobs will be selected by relative CEs. According to the job selection probability, we can imagine that the simulation results may vary from time to time due to the random choices of the jobs by CEs. This is true since the length of a job can be different from other jobs. In addition, the size of files involved in jobs can also lead to different job time. To overcome this problem, a simulation under a specific configuration should run enough times to work out the average results. An alternative is to regulate the length of jobs and the size of files so that the simulation results would not greatly differ. In my experiment, the results are obtained by calculating the average from 10 simulations for a certain configuration.

Compared with the original workloads contained in OptorSim, the imported workloads do not imply data intensive manipulation since the imported workloads contain more real file characteristics but much smaller file size. However, in simulation environments, it is possible to regulate the file size at a fixed value so that the imported workloads fit in with data intensive Grid simulations. In the simulations, the size of every file is all set to 1 GB.

Access Pattern-The access patterns are also taken into account as they can greatly influence the efficiency of job executions. In OptorSim, access patterns are defined as the patterns of file accesses during the job executions. Three types of file access patterns are simulated along with the replication strategies.

-Sequential Files are requested in a fixed order. Sequential access pattern does not change the original order in the file list of a Grid job so that each file is accessed without considering the access locality that can be exploited to reduce overall access time.

-Gaussian random walk Successive files are selected from a Gaussian distribution centred on the previous file. As many phenomena have been observed to obey Gaussian distribution (also called normal distribution), some workloads may also follow or approximately follow Gaussian distribution. Specifically, Gaussian random walk access pattern assumes that only a small portion of files hold high frequency and low frequency while most of files in the workloads hold medium access frequency around the mean value.

-Zipf File request distributions are similar with web service requests distributions, which sometimes imply Zipf-like pattern. Zipf access pattern assumes that only a small portion of files contribute to the most file accesses. The Zipf distribution has been observed to be common in many types of system workloads ranging from web servers, web proxies and file systems.

Scheduling Method-Scheduling algorithms are executed by resource brokers based on the current Grid resource status. As OptorSim is a two staged simulation environment, one can purely evaluate replication strategies based on fixed scheduling algorithms. In this thesis, we discuss the simulation with following three scheduling algorithms:

-*Random*: Jobs are scheduled to a random CE. CEs are randomly chosen according to a random number generator based on uniform distribution.

-***Queue Length*:** Jobs are scheduled to the CE with shortest job queue. If multiple CEs have same length of the job queue, it will randomly choose one according to the random number generator.

-***Queue Access Cost*:** Jobs are scheduled to the CE where the sum of file access costs and total job access costs are the smallest.

6.4 Simulation Results

The simulations contained 1000 jobs submitted in 5-second intervals. Each replication strategy was simulated 10 times for each scheduling algorithm and access pattern. Three performance indices, Mean Job time, CE Usage and Effective Network Usage (ENU), are examined to evaluate these three replication strategies. Before simulation starts, all master copies of files are initially stored in CERN, which is site 8 in EDG testbed. Allocating all master copies in single site is to guarantee the fairness of initially accessing files from different sites. In addition, the average file processing time was set to 1 second. However, according to the experience from the experiments, altering various file processing time only linearly scales the simulation results.

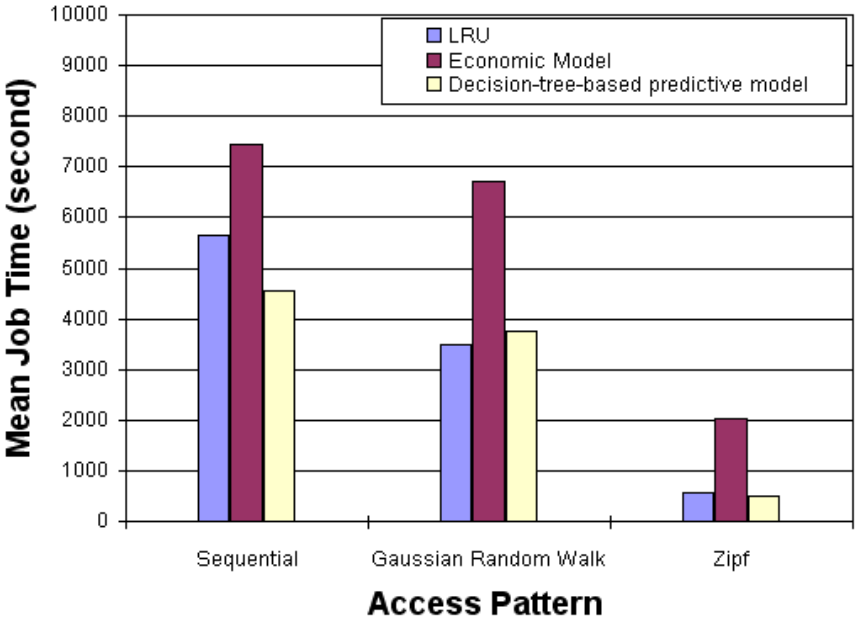


Figure 6.7 Mean job time of three replication strategies under Queue Access Cost scheduling algorithm

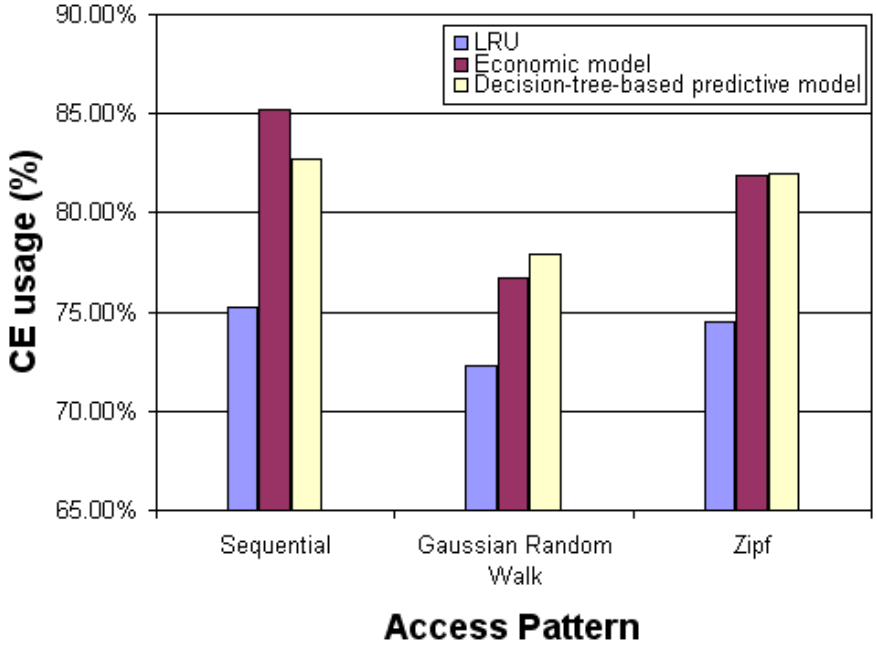


Figure 6.8 CE usages of three replication strategies under Queue Access Cost scheduling algorithm

Figure 6.7 and 6.8 show the simulation results for the three replication strategies under the Queue Access Cost scheduling algorithm. It can be seen that the access pattern Zipf offers the shortest mean job time for all three strategies. Since Zipf access pattern assumes that a small number of files contribute to the most of file accesses. Thus, the files with higher ranks based on their frequency will have greater chance to be accessed than files with lower ranks. Consequently, those files will be most likely found in the local storage because they could already exist in the local storage due to their frequency ranks. Taking into account the skewness of the workloads used in the simulation, the mean job time benefits from the file access distribution which obeys Zipf-like pattern. Figure 6.7 shows that the Economic model is not as efficient as the other two strategies in terms of mean job time. In addition, the Decision-tree-based predictive model slightly outperforms LRU under Zipf access pattern.

Figure 6.8 illustrates the CE usages of the three replication strategies under the same

configuration with Figure 6.7. It shows that the Economic model and the Decision-tree-based predictive model have relative higher CE usages than LRU. For the Decision-tree-based predictive model, it is believed that the extra computing overhead is caused by the prediction of file popularity. Similarly, the Economic model also spends extra computing overhead by introducing the calculation for the file value which is taken as the indicator whether or not to replicate a file.

As we have discussed in Section 6.2, the LRU based replication arbitrarily creates replicas without considering their potential popularity, which could lead to longer job time and higher effective network usage. On the other hand, the accuracy of files' popularity prediction is critical for the proposed predictive file replication strategy. For example, if a potentially frequent file is considered in-frequent, it will not be replicated to the other sites. However, this file will actually be demanded by many different sites due to its increasing popularity in future. Hence some of remote sites will be starving due to the access latency.

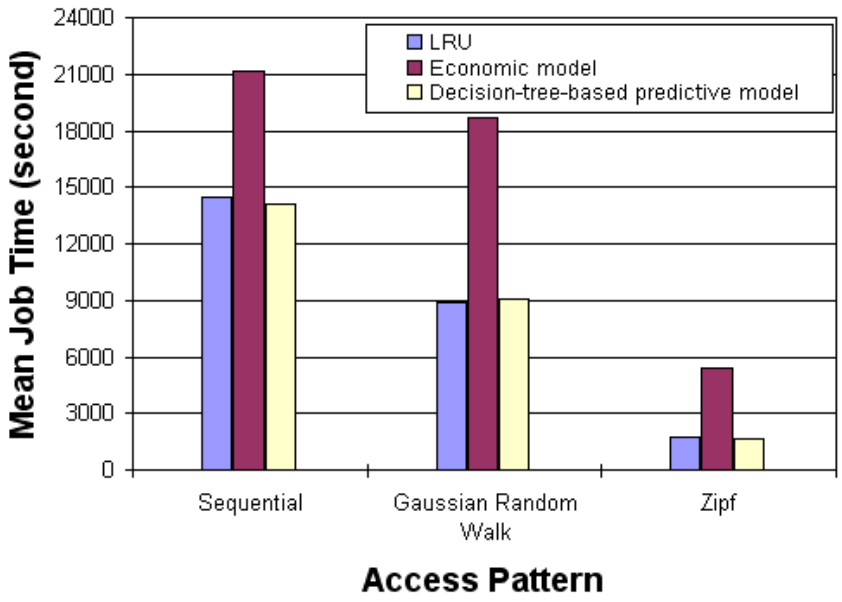


Figure 6.9 Mean job time comparison of three replication strategies under Random scheduling algorithm

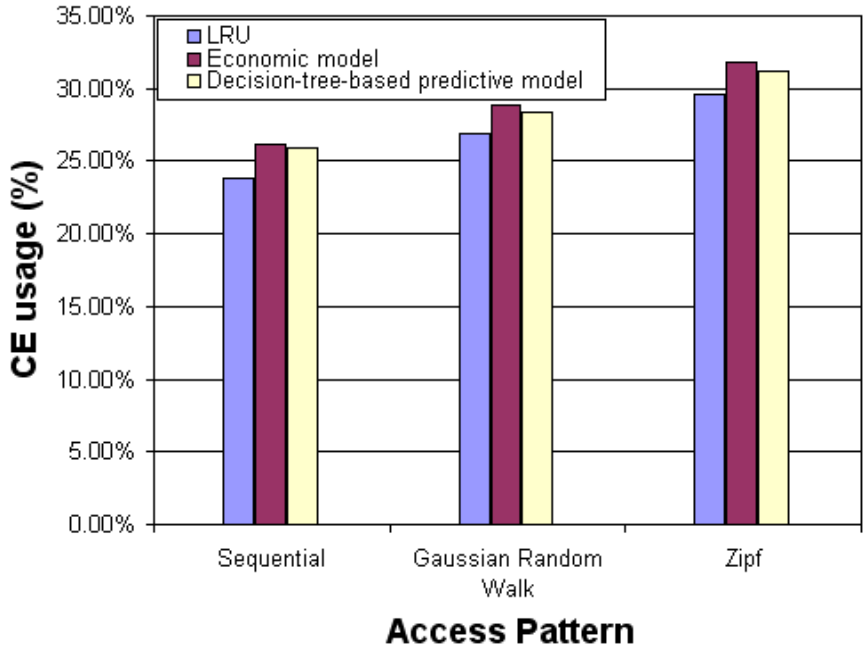


Figure 6.10 CE usages of three replication strategies under Random scheduling algorithm

Similar with the Queue Access Cost scheduling algorithm, it can be seen from Figure 6.9 and Figure 6.10 that the Zipf access pattern always produces the best results than other two access patterns. In addition, it is noticed that scheduling algorithms can dramatically influence the mean job time and CE usage. Since Queue Access Cost is the sum of file access cost and job access cost, the mean job time under this scheduling algorithm is unsurprisingly the shortest. Figure 6.7 shows that the Queue Access Cost also generates the highest CE usages for all three replication strategies. This is due to the compromise made for the sites with high network connectivity and the sites with low network connectivity. Firstly, it ensures that the sites with high network connectivity will not be overloaded. On the other hand, sites with low network connectivity are guaranteed not to be idle. With such a balance, CEs are greatly utilized by the Queue Access Cost scheduling algorithm.

From Figure 6.9, 6.10, 6.11 and 6.12, we can see that Zipf access pattern offers the shortest mean job times and highest CE usage for all strategies. This can be explained by the access distribution of the workloads used in the simulation, which in consequence leads to the best results compared with other access patterns. According

to the investigations [48] [49], most workloads of and web servers and data hosting servers exhibit Zipf access pattern, that is, some of files are intensively accessed while others are rarely accessed.

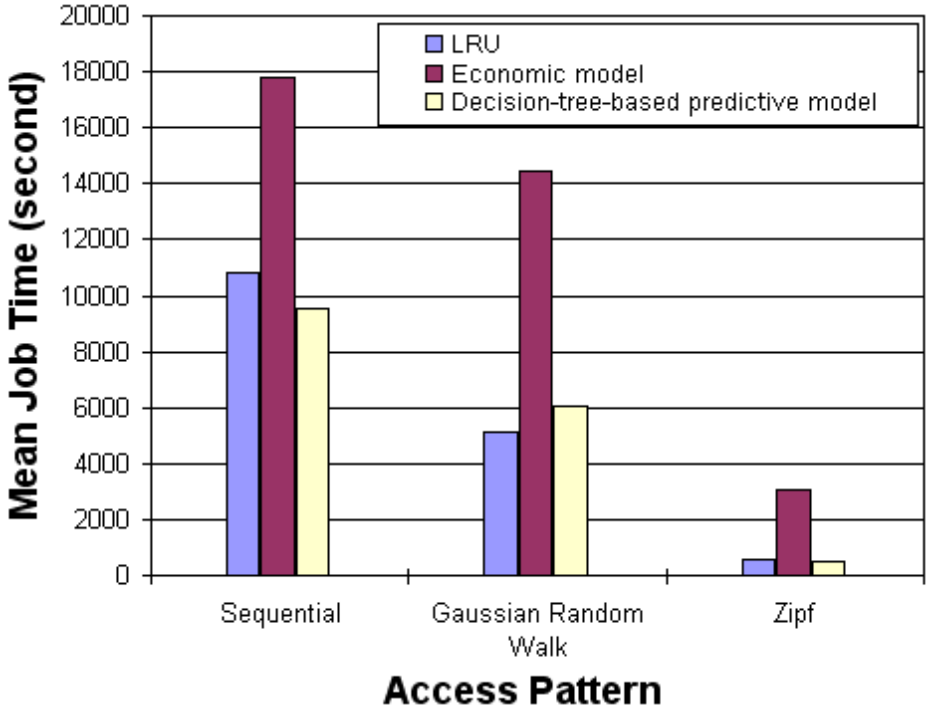


Figure 6.11 Mean job time of three replication strategies under Queue Length scheduling algorithm

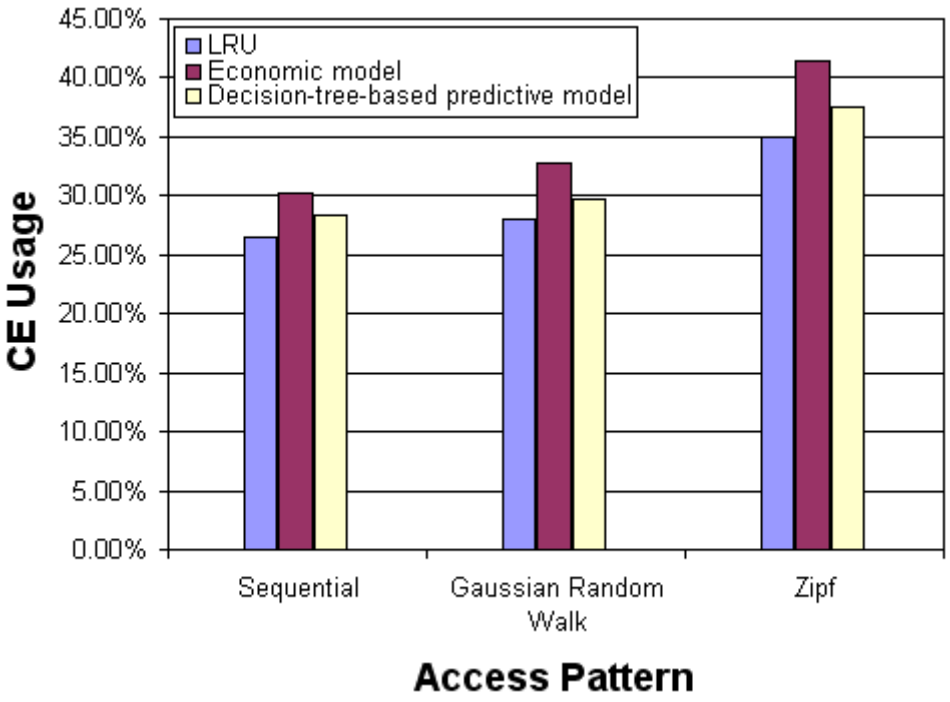


Figure 6.12 CE usages of three replication strategies under Queue Length scheduling algorithm

Conclusion can be drawn by comparing the mean job time and CE usages under all these three scheduling algorithms, each of which involves three access patterns. First of all, the proposed Decision-tree-based model outperforms the LRU and Economic model in terms of the mean job time given Sequential and Zipf access patterns. This is due to the advanced awareness of the potential popular files during the job execution. On the other hand, the proposed strategy always maintains high CE usages, which are caused by the extra computing overhead for predicting files' popularity. In addition, it is noticed that the Queue Length scheduling algorithm brings better balance than the other two scheduling methods between mean job time and CE usage. Further more, due to the existence of the trace skewness in the workloads Zipf access pattern can dramatically improve the job execution efficiency of all three replication strategies. In the following analysis, we compare the effective network usage (ENU) under Queue Length scheduling algorithm.

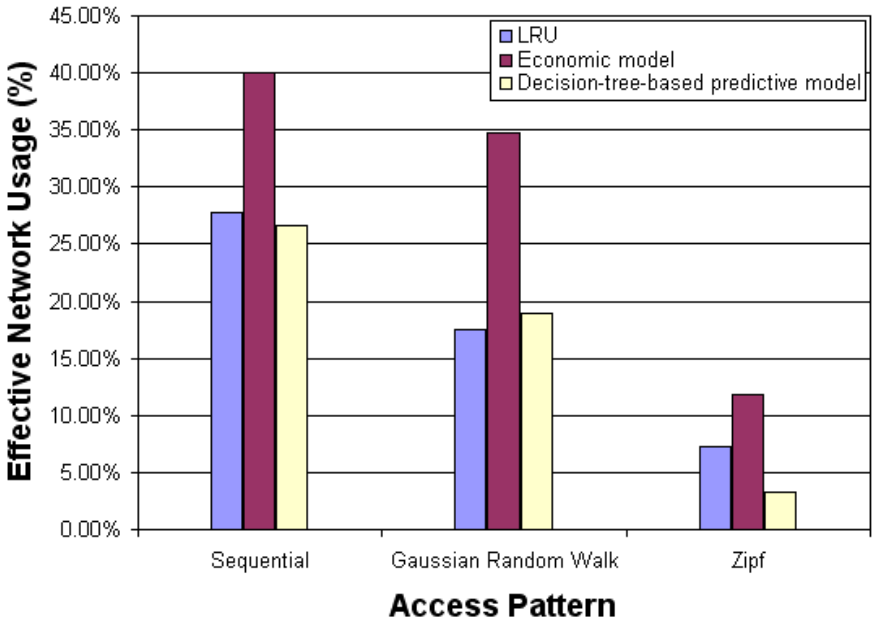


Figure 6.13 Effective network usages of three replication strategies under Queue Length scheduling algorithm

The effective network usage illustrates the bandwidth consumption during the job execution. Figure 6.13 represents the effective network usages of all three replication strategies given the Queue Length scheduling algorithm. As the Economic model employs the reverse auction mechanism to message other Grid sites for finding the "cheapest" replicas, the messaging overhead leads to relative higher effective network usages than the others. The Decision-tree-based predictive model maintains similar effective network usage compared with LRU. This can be explained with the definition of the effective network usage that is the equivalent to:

$$ENU = \frac{the\_number\_of\_remote\_accesses + the\_number\_of\_replicas}{the\_number\_of\_remote\_accesses + the\_number\_of\_local\_accesses}$$

As the proposed predictive model reduces the number of replicas while the remote accesses are then added, ENU would not decrease unless the most of replicas are found in the local storage such like Zipf-like access pattern.

Figure 6.14, 6.15 and 6.16 compare the performance of all three replication strategies with various workload skewness ranging from 20/80 to 20/40, which denote that the skewness is tuned from the 80% of total accesses contributed by 20% files to the 40% of total accesses contributed by 20% of files.

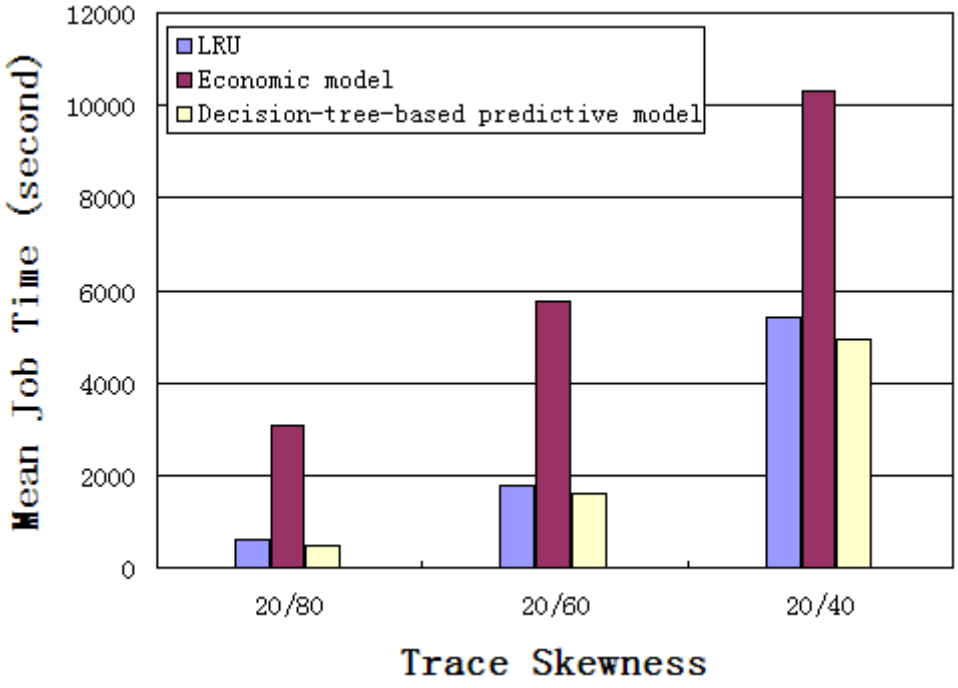


Figure 6.14 Mean job time of three replication strategies under Zipf access pattern and Queue Length scheduling algorithm

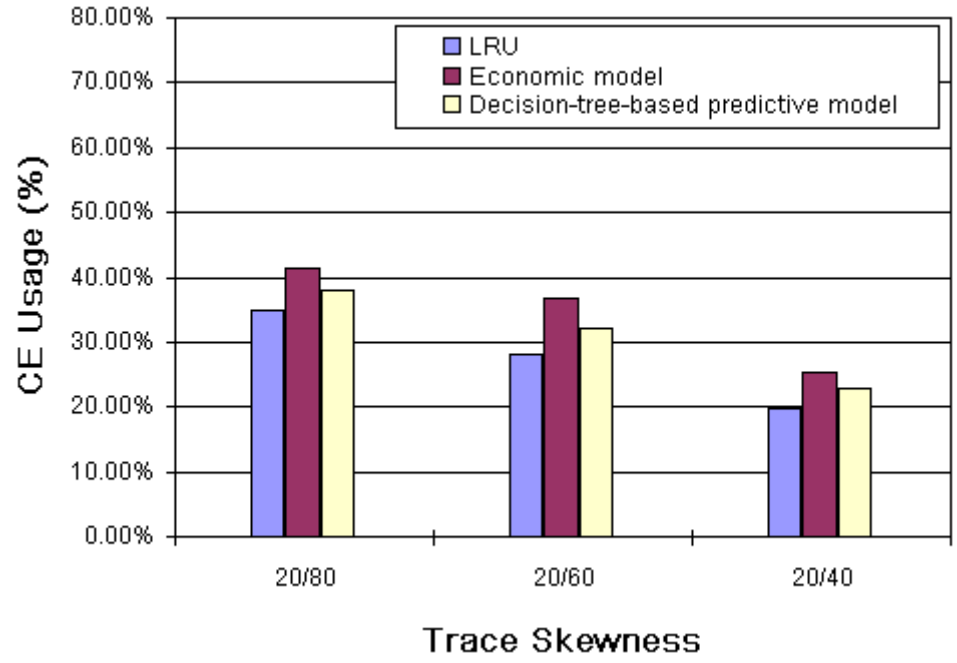


Figure 6.15 CE usages of three replication strategies under Zipf access pattern and Queue Length scheduling algorithm

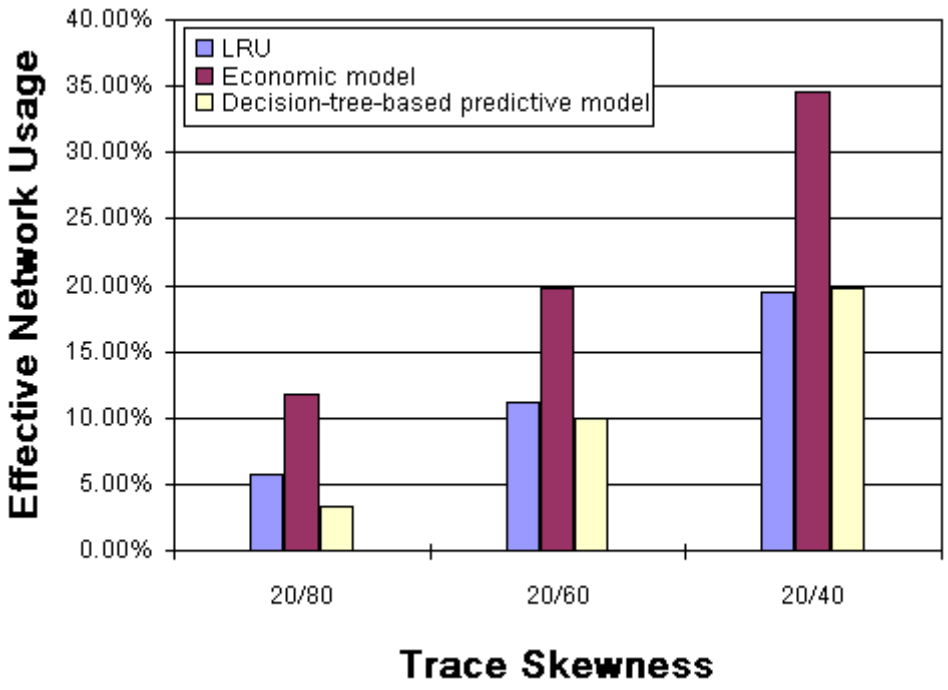


Figure 6.16 Effective network usages of the three replication strategies under Zipf access pattern and Queue Length scheduling algorithm

It is clear that the trace skewness dramatically affects the performance under Zipf access pattern. Reducing 20% skewness will approximately increase mean job time by 3 times and 3.4 times for LRU and the predictive model respectively. Meanwhile, the Economic model also gains twice the mean job time. The above three figures reveal that the trace skewness only linearly affects the three performance indices under the Zipf access pattern.

Moreover, the mean job time is also simulated with different average storage capacity given the Zipf access pattern and the Queue Length scheduling algorithm. Figure 6.17 and 6.18 show that the mean job time scales down with the increase of the average storage capacity. Since the larger storage capacity can accept more replicas which will be locally accessed in future, a great number of remote file I/Os would be saved. If each site has unlimited storage capacity, LRU will outperforms the other two strategies in terms of mean job time given a number of repeatedly executed jobs. Also, when the average storage capacity is tuned to 100GB, the mean job time of each strategy no longer scales down as the storage capacity is already large enough for replicating all the

requested files.

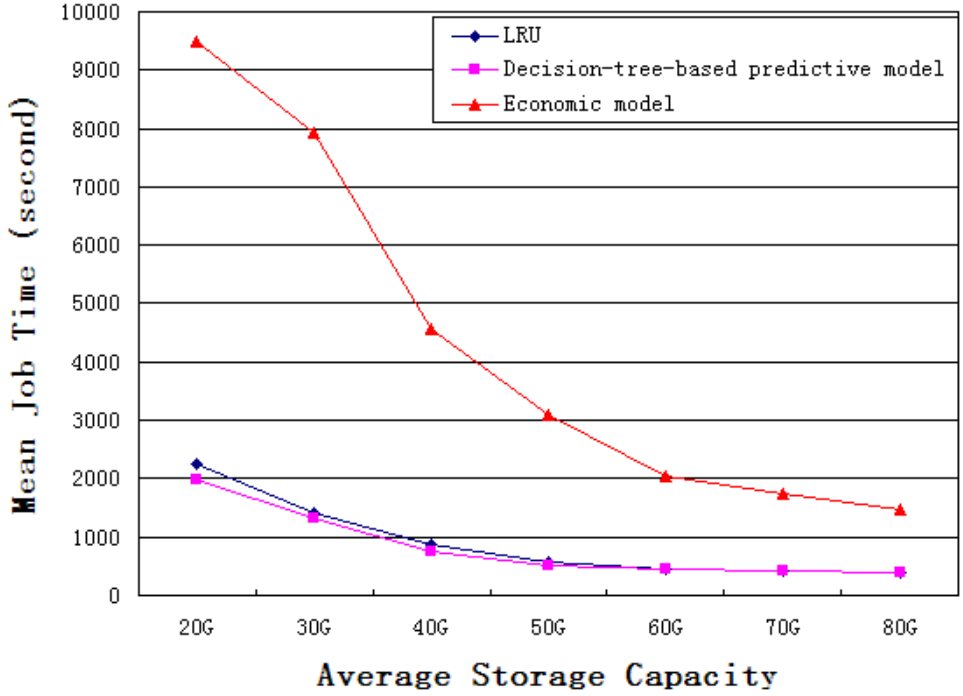


Figure 6.17 Mean job time of all three strategies given various average storage capacity

Zipf	**Performance**	**LRU**	**Economic model**	**Predictive model**
	Mean Job Time(s)	595(-11%)	3081(-83%)	530
	CE Usage	35%	41.40%	37.60%
	ENU	7.30%	11.76%	3.30%
Gaussian random walk		**LRU**	**Economic model**	**Predictive model**
	Mean Job Time(s)	5137(+17%)	14431(+58%)	6077
	CE Usage	28.10%	32.80%	29.70%
	ENU	17.5	34.70%	18.90%
Sequential		**LRU**	**Economic model**	**Predictive model**
	Mean Job Time(s)	10808(-12%)	17775(-46%)	9553
	CE Usage	26.50%	30.20%	28.30%
	ENU	27.80%	40%	26.65%

Table 6.2 The performance summary of the three strategies given various access patterns under the Queue Length scheduling algorithm

Table 6.2 shows the summarized performance of the three strategies under the Queue Length scheduling algorithm. Given the network file system workloads described in

Section 6.1, the proposed replication strategy outperforms LRU by 11% and 55% in terms of the mean job time and effective network usage respectively given the Zipf access pattern. In addition, it shows that the proposed replication strategy also outperforms the Economic model by 83% and 72% accordingly.

Summary

Data Grids deal with computational problems involving large volume of data such as scientific research data, climate monitoring data, satellite images, and sensor network data and so on. With the development of Grid infrastructures, software acting as the basic interfaces for future Grid application development is being intensively studied. However, compared with Grid middleware development, Grid resource optimization strategies are far behind studied and discussed.

In Grid environments, file replication strategies are critical to overall performance of large-scale data intensive applications. However, due to the dynamism of the Grids, file replication decisions are always made by monitoring the change of files' popularity. Although promptly replication can avoid the increase of access latency in future, the burden of the replications and the current accesses to the relative files may conflict each other and hence increase the access latency. Ideally, advanced file replications can smooth the access latency if the changes of files' popularity can be predicted. In this case study, it proposes a predictive file replication strategy based on forecasting files' future popularity to address the problem.

The real-system-trace-based simulations were conducted under European Data Grid simulation environment OptorSim. Having simulated the three replication strategies, it is clear that the predictive replication strategy outperforms the LRU and Economic model under sequential and Zipf access patterns. In addition, the Queue Length scheduling algorithm brings the balance between mean job time and resource usage. It is also noticed that no strategy delivers the best performance results in every circumstances. In order to choose a good replication strategy, trace skewness, storage capacity and the maximal computing power have to be considered.

As being a policy generator, the decision tree based predictive model is supposed to link file's own characteristics and its' future popularity and pass the rules to replication manager to decide when and where to create replicas. With advanced evaluation of files, access latency caused by geographically distributed resources can be smoothed.

Chapter 7 Conclusion

Storage system issues have been the challenging problems due to the widening gap of storage I/O systems, as evidenced by the growing number of researching efforts engaged in smoothing access latency of file caching and replication. This thesis inspects the storage system issues from the perspectives of data mining technology and raises the heuristic of exploiting file access frequency to enable the advanced and intelligent file system caching and replication. The ability of discovering unseen useful information enables data mining technology viable for improving storage system performance where large amounts of data stay undiscovered.

Data mining methodologies may be applied to various application areas where intensive data are automatically generated. Four typical data mining methodologies are reviewed and studied generally. The reason of choosing these four data mining techniques is that the related works of applying data mining to file and storage systems are mainly based on these four candidates. In order to echo literature review and create solid comprehension of the two cross-domain areas: data mining and storage system, studying the four adopted data mining techniques is essential.

Through inspecting the related works and the file and storage system issues, this thesis raised the hypothesis: whether we can make advanced file allocation and replication. A series statistical experiment supported the hypothesis via examining the association between file's own attributes and its future popularity. Next, a predictive model is proposed to evaluate the detailed contribution given by each attribute and forecast the future popularity.

The proposed model: An incremental decision tree based predictive model is capable of modelling the future file usage in the form of file access frequency class. Compared with the related works, the proposed model has two main advantages. Firstly, the file's future access frequency is predicted instead of supervisory and monitoring methods.

Therefore, the predictive model enables the advanced file usage evaluation that can be used to smooth the access latency of storage systems before the performance degrades. Secondly, the model is incrementally updated to feature time series data streams, which may require model re-construction for the static predictive models. The accuracy of the model is evaluated through N-fold cross-validation showing that the model can predict file access frequency with sufficient accuracy given three network file system workloads.

Two application scenarios are presented in Chapter 5 and Chapter 6. The first application employs the proposed model to predict access future frequency of files which are then determined to be stored in the hierarchical storage devices. A new caching hierarchy is also developed to serve flash memory conducted file caching. The potential of NAND flash memory adopted can contribute to storing large volumes of active files for read-dominating workloads. The reason that only read-dominating workloads can take advantages is because the life cycle of NAND flash memory is limited by the number of erase operations, which is normally one million times. Both analytical and experimental studies have suggested the improvement range is 2-5 compared with the base case (two tired caching structure without flash memory).

Another application: predictive file replication on the Data Grids is discussed in Chapter 6. Instead of monitoring the file usage threshold to determine files' popularity, the proposed incremental decision tree model is supposed to predict files' future popularity in data intensive Grid environments. The predicted file access frequency is taken as the indicator to initially determine the creation of file replicas and their relative location. The new replication strategy is compared with the most widely adopted LRU and the recently proposed Economic model in aspects of mean Grid job time and computing element usage. The simulation results show that the predictive replication strategy outperforms LRU and Economic model in some specific circumstances.

This thesis contributes to, firstly of all, a transaction-filtering algorithm that is capable of reducing database scanning overhead. The method can be universally adopted for data mining tasks where multiple database scans are required, especially for

multidimensional association rule mining. Secondly, this thesis inspects file and storage system issues from data mining perspectives and then introduces a predictive model to forecast files' future popularity. Finally, the proposed predictive model is deployed in two application scenarios (network file systems and Data Grids) to optimize file caching and replication respectively. Both scenarios benefit from the reduced access latency gained by the advanced file popularity prediction.

Some Thoughts and Future Works

In addition to data mining in file system level, it is believed that low level system behaviours are also exploitable to deliver system optimization. Similar with file systems, low level system behaviours are captured by low level traces reflecting time series data block requesting information such as time stamp, physical address, operation mode (read or write) and even its high level session. With the capability of discovering unseen information of data mining, some potential useful patterns of low level system behaviours are expected to be uncovered.

■Firstly, in the disk level of storage systems, intensive block requests occur between the cache and disk. Similar to file system level, disk level requests reflect low level system behaviours recorded by disk level traces. C-Miner in literatures mines the correlations among blocks from disk level traces where the block requesting information is stored. However, C-Miner mines the frequent block access sequences without considering the spatial locality of these correlated blocks, which could be used to further enhance caching efficiency. In fact, as we have discussed, spatial locality vitally influences the caching efficiency as sequential blocks and random blocks are two extreme spatial locality cases. We believe that sequentially correlated blocks are supposed to be treated differently with randomly correlated blocks and partially sequentially/randomly correlated blocks. This is because the sequentially correlated blocks can be fetched into cache much quicker from disks (no need to reposition disk head) than fetching randomly correlated blocks or partially randomly/sequentially correlated blocks.

Therefore, verifying spatial locality along with access locality of blocks can help us

better determine the priority of block access sequences to be remained in the cache or prefetched from the disk. For example, the mined frequent sequential block sequences should have lower priority than frequent random block sequences to be kept in the cache. The Following figures illustrate the idea.

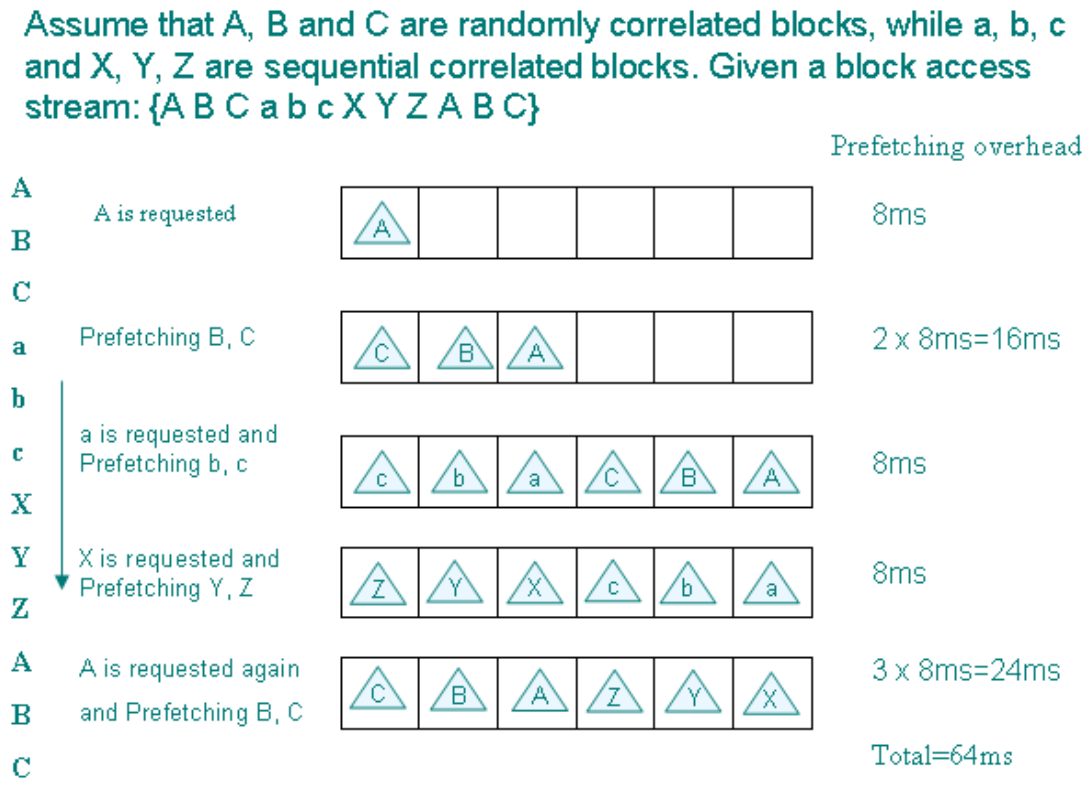


Figure 7.1 Caching and prefetching correlated blocks without considering spatial locality

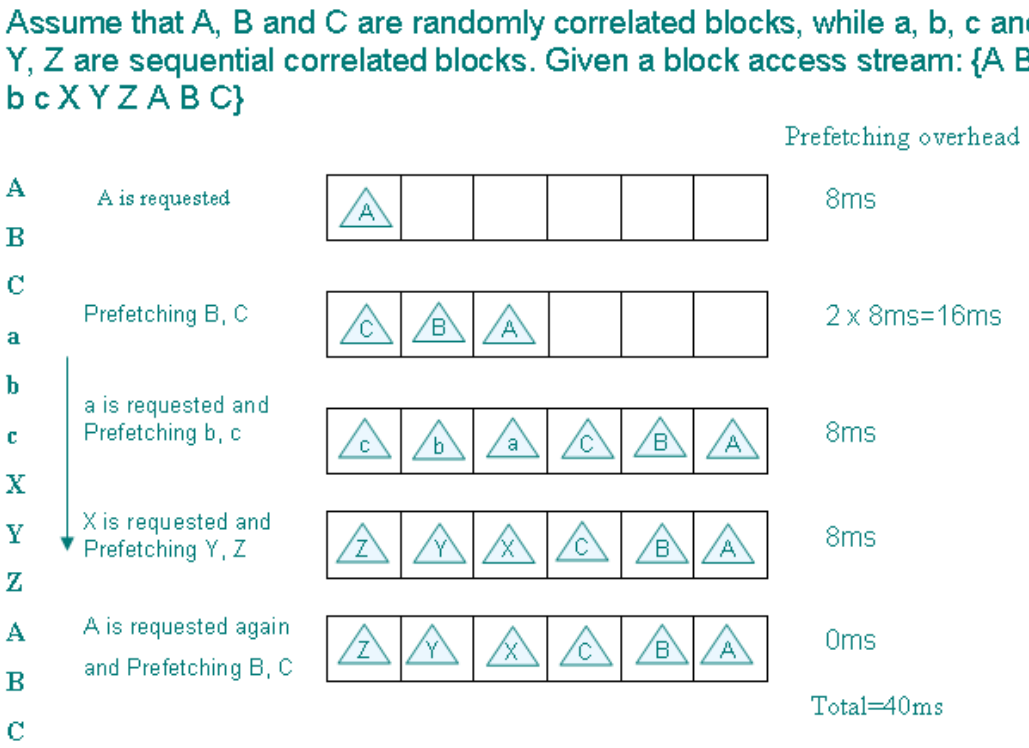


Figure 7.2 Caching and prefetching correlated blocks by considering the priority supported by blocks' spatial locality

The above figures show how the caching efficiency can be further improved by utilizing the spatial-locality-based caching policy. In Figure 7.1, spatial locality is not taken into account during the block access streaming. Instead, adopting spatial-locality-based prefetching can dramatically reduce the overall data access time shown in Figure 7.2.

The heuristic combines both the access locality and spatial locality of low level block streams. The frequent sequence analysis is supposed to find out the desired frequent block sequences with pre-determined threshold. This study should include the design of caching and prefetching policies, block access sequence analysis and the study of the impact given by the cache size. In my future work, a new disk caching policy is expected to be designed to serve intelligent data block management.

■Besides I/O latency, power management of hard disks is another emerging issue to storage research community. In computer systems, it had been observed that a hard disk can consume more than one fifth of the total power [50], [51]. Moreover, it is noticed

that an increasing portion of power consumed by hard disks can be predicted in near future [52].

With the capability of discovering unseen regularities offered by data mining methodologies, we believe that advanced awareness of disk idle periodicity can help us make better power management of hard disks. Let us consider such an example. Suppose we have an observed disk access stream shown in Figure 7.3 representing the disk requests recorded over a certain period.

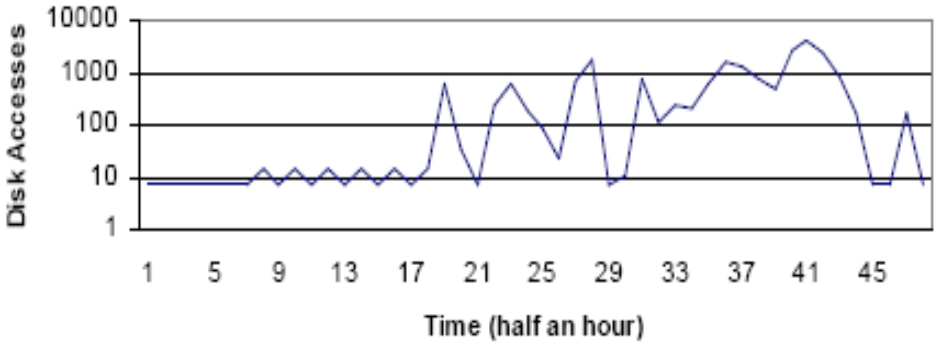


Figure 7.3 An example of observed disk access streams

As we can see from the above graph, disk access exhibits bursting pattern over the entire period. In fact, disks save its power by shutting down disk platters and arms once no disk requests waiting in the queue. Whenever a request is detected, it starts to spin up in order to serve the data requesting. However, shutting down the disk when no request is in the queue does not always save energy in case that another disk request is going to arrive in a reasonably short time. This observation is based on that frequent mechanic movements such as disk spinning down/up are the largest parts of power consumed of disks. Therefore, it is necessary to identify the disk idle time that is long enough to make the system confident to shut down the disk.

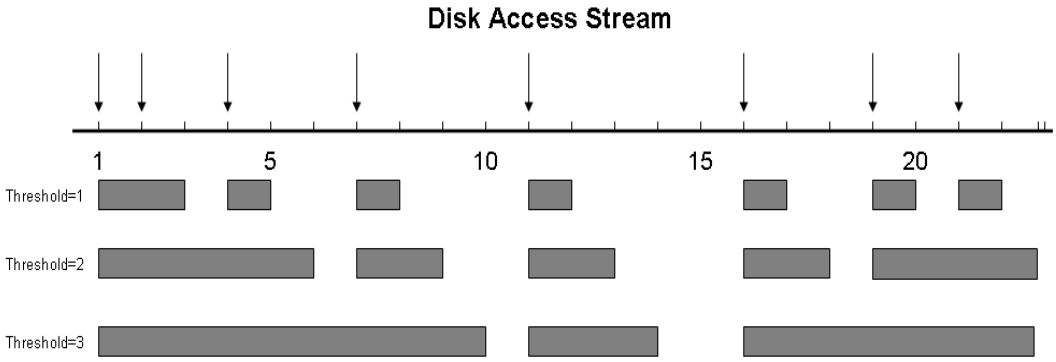


Figure 7.4 Clustering disk accesses to distinguish disk idle time

Ideally, disk requests can be clustered together to help identify the idle periodicity. The graph above shows the clustering of disk requests with various given thresholds which are used to separate clusters into independent sessions. The idle period is defined as the interval between the last request of the last session and the first request of the next session. We noticed that the size of thresholds can generate sessions with various lengths which could in consequence change the idle periods.

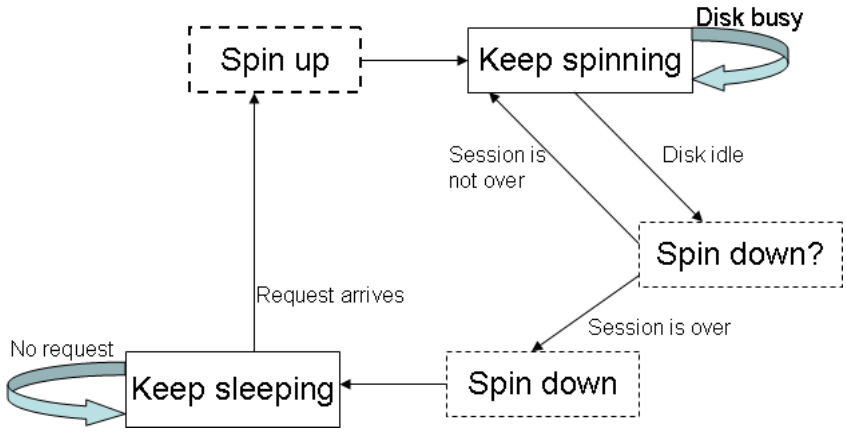


Figure 7.5 A demonstration of disk power management cycle

An accurate idle periodicity prediction model is needed to identify the long idle time that is worthy to let disks shut down. Data mining technique is expected to learn the historic disk requests patterns and extract the model serving accurate prediction. However, challenges such as threshold configuration, long idle period definition and requests bursting pattern identification need to be solved carefully.

Final thoughts

I believe that data are generated by the order of nature, where principles, rules and patterns function regularly. Wherever data exist, truths are hidden behind. By mining, we can discover the hidden information more efficient and accurate. Yet, it is our responsibility to find the gold nuggets from a pile of dirt.

The Candidate's Publication list During the PhD Study Period (Oct-2005, Oct 2008)

1. C. Liao, Frank Wang, N. Helian, S. Wu, Y. Deng, "Fast Flash Memory Caching Based on File Access frequency", *in proceedings of IADIS International Conference Applied Computing, 2008.(* The outstanding paper Award)

2. C. Liao, Frank Wang, N. Helian, "A DATABASE PREPROCESSING APPROACH FOR ARM", *in proceedings of IADIS International Conference e-society, 2008.*

3. C. Liao, Frank Wang, Sining Wu, M.M Rashid, "A General Survey of Knowledge Discovery in File and Storage Systems", *in proceedings of Cranfield Multi-strand conference, 2008.*

4. Frank Wang, C. Liao, N. Helian, Chris Thompson, S. Wu, Y. Deng, V. Khare & A. Parker, "Accelerating Linux/Windows File Systems by Predicting Access Frequency", *in poster sessions of UK e-science all hands meeting, 2007.*

5. C. Liao, Frank Wang, "Filtering Transaction to Speed Up Association Rule Mining", *in PhD forum of the 24th British National Conference of Database, 2007.*

6. Frank Wang, N. Helian, S. Wu, Y. Deng, V. R. Khare, C. Liao, R. Yates, P. Fairbairn, J. Crowcroft, J. Bacon, M. A. Parker, Z. Xu, Y. Guo "Parallel Streaming: Tenfold Accelerations of Office/Database/Web/Media Applications over the Internet/Grids", in *poster session of Super Computing, 2007(SC07).*

7. Frank Wang, Y. Deng, N. Helian, V. Khare, S. Wu, C. Liao & A. Parker, "Speeding

Up a Magnetic Disk by Clustering Frequent Data", *IEEE TRANSACTION ON MAGNETICS, Volume 43, Issue 6, June 2007 Page(s):2295 - 2297, 2006.*

8. YuHui Deng, Frank Wang, Na Helian, Sining Wu, Chenhan Liao, "Dynamic and scalable storage management architecture for Grid Oriented Storage devices", *Parallel Computing 34(1): 17-3, 2008.*

9. C. Liao, Frank Wang, S. Wu, N. Helian, Wen Zhang, "A Predictive File Replication Strategy on the Grids", *abstract accepted for the First International Conference on Parallel, Distributed and Grid Computing for Engineering to be held in Pécs, Hungary, 6-8 April 2009*

Bibliography

[1] Finding Quatations, 2006. [Online]. Available: http://thinkexist.com/quotation/. [Accessed: June 2008].

[2] J. Han, et al, "Mining Frequent Patterns without Candidate Generation," *In Proceedings Of Conf. on the Management of Data, 2000, ACM Press, New York, NY, USA.*

[3] Cross Industry Standard Process for Data Mining, 2007. [Online]. Available: http://www.crisp-dm.org/. [Accessed: June 2007].

[4] Successful Data Mining Applications, 2005. [Online].
Available:http://www.kdnuggets.com/polls/2005/successful_data_mining_applications.htm.[Accessed: June 2008].

[5] Cello traces from HP labs, 2001. [Online].
Available: http://tesla.hpl.hp.com/public_software/. [Accessed: April 2007].

[6] Daniel Ellard, Michael Mesnier, En0 Thereska, G. R. Ganger, and Margo Seltzer, "Attribute-Based Prediction of File Properties", *Harvard Computer Science Group Technical Report TR-14-03, December 2003.*

[7] N. Allen, "Don't waste your storage dollars: what you need to know," Research note, Gartner Group, March, 2001.

[8] J. Gray, "A conversation with Jim Gray," *ACM Queue, Vol. 1, No.4, ACM, June 2003.*

[9] E. Lamb, "Hardware spending sputters," *Red Herring, June, 2001, pp 32–33.*

[10] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *In Proceedings of the 20th VLDB Conferenc,.1994, pp.487-499.*

[11] J.S. Park, et al, "An Effective Hash-Based Algorithm for Mining Association Rules," *In proceedings Of ACM-SIGMOD Int. Conf. Management of Data,1995, pp. 175-186, San Jose, CA.*

[12] A. Savasere, et al, "An Efficient Algorithm for Mining Association Rules in large Databases," *In Proceedings of 21st VLDB 1995,, pp. 432-444.*

[13] H. Toivonen, "Sampling Large Databases for Association Rules," *In Proceedings of*

the 22nd VLDB Conf., 1996, pp. 134-145.

[14] M. Z. Ashrafi, et al, "ODAM: An Optimized Distributed Association Rule Mining Algorithm," *IEEE Distributed System Online 1541-4922, IEEE Computer Society, Vol. 5, No. 3, 2004.*

[15] P. Tang, et al, "Mining Web Access Patterns with First-Occurrence Linked WAP-Trees*," *In Proceedings of Software Engineering and Data Engineering,2007, Las vegas, Nevada.*

[16] IBM Quest Data Mining Project, "Quest Synthetic Data Generation Code," 1997. [Online]. Available: http://www.almaden.ibm.com/software/projects/iis/hdb/projects/data_mining/datasets/syndata.html, [Accessed: 15th June 2006]

[17] S. Brin, et al, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," *ACM SIGMOD Conference on Management of data,1997, pp. 255-264.*

[18] R. Agrawal and J. Shafer, "Parallel Mining of Association Rules, *IEEE Transactions on Knowledge and Data Engineering," Vol. 8, No. 6, 1996, pp. 962-969.*

[19] T. Brijs, "Retail Market Basket Data Set," 2001. [Online]. Available: http://fimi.cs.helsinki.fi/data/retail.pdf. [Accessed: 16th June 2006]

[20] T. Brijs, "Retail Market Basket Analysis: A Quantitative Modeling Approach," Ph.D. dissertation, Faculty of Applied Economics, Limburg University Center, Belgium, 2002.

[21] K. Geurts, "Traffic Accidents Data Set," 2006. [Online]. Available: http://fimi.cs.helsinki.fi/data/accidents.pdf. [Accessed: 16th June 2006]

[22] Belgian Institute for Traffic Safety (BIVV) and National Institute for Statistics, Year report on Traffic Safety 2000 (CD-ROM), BIVV v.z.w., Brussels, 2000.

[23] T. M. Kroeger and D. D. E. Long, "The Case for Efficient File Access Pattern Modeling," *in Proceedings of the Seventh Workshop on Topics in Operating systems, 1999.*

[24] Z. Li, et al, "Mining Block Correlations in Storage Systems," *In proceedings of the 3rd USENIX Conference on File and Storage Technologies, 2004, pp. 173-186.*

[25] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li, "Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling," *ACM Transactions on Computer Systems, Vol. 14, No .4, 1996, pp.*

311–343.

[26] Mendel Rosenblum and John K. Ousterhout, “The Design and Implementation of a Log-Structured File System,” *ACM Transactions on Computer Systems, Vol. 10, No. 1, 1992, pp. 26–52.*

[27] Keith Muller and Joseph Pasquale, “A High Performance Multi-Structured File System Design,” *In Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP-91), 1991, pp. 56–67, Asilomar, Pacific Grove, CA.*

[28] J. Griffioen and R. Appleton, “Performance measurements of automatic prefetching,” *in Proceedings of IEEE Parallel and Distributed Computing Systems, 1995, pp. 165–170.*

[29] Glen and G. Langdon and Thomas M. Kroeger and Thomas M. Kroeger and Thomas M. Kroeger, “Predicting File System Actions from Reference Patterns,” *Technique Report of Department of Computer Engineering, University of California, 2000.*

[30] X. Yan, J. Han, and R. Afshar, “CloSpan: Mining closed sequential patterns in large datasets,” *In Proceedings of Int. Conf. Data Mining (SDM’03), 2003, San Fransisco, CA.*

[31] James T. McClave, Frank H. Dietrich II, and Terry Sincich, Statistics, London: Prentice Hall, 1997.

[32] D. Muntz and P. Honeyman, "Multi-level Caching in Distributed File Systems", *Proceedings of the USENIX Winter 1992 Technical Conference, 1992, pp.305—313.*

[33]Darryl L. Willick and Derek L. Eager and Richard B. Bunt, "Disk Cache Replacement Policies for Network Fileservers", *In Proceedings of International Conference on Distributed Computing Systems, 1993, pp.2-11*.

[34] IBM Company, “IBM Mainframe Introduction”, 2006. [Online]. Available: http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro.html.
[Accessed: June 2007]

[35] Daniel Ellard, Jonathan Ledlie, Pia Malkani, and Margo Seltzer, “Passive NFS Tracing of Email and Research Workloads,” *In Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST’03), March 2003, pp 203–216, San Francisco, CA.*

[36] T. M. Mitchell, Machine Learning, New York: McGraw-Hill, 1997.

[37] Utgoff, P, “ID5R: An Incremental ID3,” *In Proceedings of the Fifth International Conference on Machine Learning, 1988, pp 107-120.*

[38] eeglosary, “neural networks: Definition and Recommended Links,” 2004. [Online]. Available: http://www.eeglossary.com/neural-networks.htm. [Accessed: June 2008].

[39] Frank Wang, Y. Deng, N. Helian, V. Khare, S. Wu, C. Liao & A. Parker, “Speeding Up a Magnetic Disk by Clustering Frequent Data”, *IEEE Transactions on Magnetics, 2006, eg13*.

[40] M. Carman, F. Zini, L. Serafini, and K. Stockinger, “Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid,” *In Int.Workshop on Agent based Cluster and Grid Computing at CCGrid, Berlin, Germany, May 2002. IEEE-CS Press.*

[41] Cameron, D.G, et al, “Evaluating scheduling and replica optimisation strategies in OptorSim”, *In Proceedings of in Proceedings of Fourth International Workshop on Grid Computing, 17 Nov. 2003.*

[42] K. Ranganathan and I. Foster, “Identifying Dynamic Replication Strategies for a High Performance Data Grid,” In *Proceedings of the Int. Grid Computing Workshop, Denver, CO, USA, November 2001*.

[43] K. Ranganathan and I. Foster, “Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications,” *In Int. Symposium of High Performance Distributed Computing, Edinburgh, Scotland, July 2002*.

[44] P. Crosby. EDGSim. [Online]. Available: http://www.hep.ucl.ac.uk/˜pac/EDGSim/. [Accessed: February 2008].

[45] R. Buyya and M. Murshed, “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,” *The Journal of Concurrency and Computation: Practice and Experience, May 2002, pages 1–32, Wiley Press.*

[46] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, “OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies,” *Int. Journal of High Performance Computing Applications*, 2003, *Vol. 17, No.4.*

[47] WP2 Optimization Team, “OptorSim, a Replica Optimiser Simulator,” [Online]. Available: http://cern.ch/edg-wp2/ optimization/optorsim.html. [Accessed: February

2008].

[48] M. F. Arlitt and C. L. Williamson, “Web Server Workload Characterization: The Search for Invariants,” *In ACM Sigmetrics Int. Conf. on Measurements and Modeling of Computer Systems, Philadelphia, PA, USA, May 1996.*

[49] P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation,” *In ACM Sigmetrics Int. Conf. on Measurements and Modeling of Computer Systems, Madison, WI, USA, July 1998.*

[50] F. Douglis, P. Krishnan, and B. Marsh, “Thwarting the power hungry disk,” *In USENIX Winter Conference, 1994, pp 293–306.*

[51] K. Li, R. Kumpf, P. Horton, and T. Anderson, “A quantitative analysis of disk drive power management in portable computers,” *In USENIX Winter Conference, 1994, pp. 279–292.*

[52] J. R. Lorch and A. J. Smith, “Software strategies for portable computer energy management,” *IEEE Personal Communications, June 1998, Vol. 5, No.3, pp. 60–73.*

[53] F. Douglis, et al, “A Comparison of Two Distributed Systems: Amoeba and Sprite”, *ACM Transactions on Computer Systems, 1991, volume 4, pp 261-275.*

[54] A. Schuster and R. Wolff, “Communication-Efficient Distributed Mining of Association Rules,” *In Proceedings Of ACM SIGMODE Conference, Management of Data, ACM Press, 2001, pp. 473-484.*

[55] B. Lee, et al, “Semantic Data Mining of Short Utterance,” *IEEE Transactions On Speech And Audio Processing, 2005, Vol. 13, No. 5.*

[56] M. S. Chen, et al, “Data Mining: An Overview from a Database Perspective,” *IEEE Transactions on Knowledge and Data Engineering, 1996, Vol. 8, No. 6.*

[57]S. Brin, et al, “Dynamic Itemset Counting and Implication Rules for Market Basket Data,” *In Proceedings of ACM SIGMOD Conference on Management of data, 1997, pp. 255-264.*

[58] T. Shintani and M. Kitsuregawa, “Hash-Based Parallel Algorithms for Mining Association Rules,” *In Proceedings of Conf. Parallel and Distributed Information Systems, IEEE CS Press, 1996, pp. 19-30.*

[59] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, New York: Morgan Kaufman Publishers, ISBN 1-55860-489-8, 2001.

[60] D. Agrawal and C. Aggarwal, “On the Design and Quantification of Privacy Preserving Data Mining Algorithms,” *In Proceedings of 20th ACM Symposium on principles on Principles of Databases systems (PODS), 2001.*

[61] Carl Staelin and H. Garcia-Molina, “Clustering active disk data to improve disk performance,” *Tech. Rep. CS-TR-283-90, Dept. of Computer Science, Princeton Univ., Princeton, N.J. Sept. 1990.*

[62] Daniel Ellard, Jonathan Ledlie, and Margo Seltzer, “The Utility of File Names,” *Technical Report TR-05-03, Harvard University Division of Engineering and Applied Sciences, 2003.*

[63] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. “Informed Prefetching and Caching,” *In ACM SOSP Proceedings, 1995.*

[64] Jonathan Ledlie, SOS project, 2003. [Online].

Available: http://www.eecs.harvard.edu/sos/index.html. [Accessed: July 2006].

[65] Daniel Ellard and Margo Seltzer. “New NFSTracing Tools and Techniques for System Analysis” *In Proceedings of the Seventeenth Annual Large Installation System Administration Conference (LISA’03), October 2003, pp 73–85, San Diego, CA.*

[66] M. F. Arlitt and C. L. Williamson, “Web Server Workload Characterization: The Search for Invariants,” *In ACM Sigmetrics Int. Conf. on Measurements and Modeling of Computer Systems, Philadelphia, PA, USA, May 1996.*

[67] The European DataGrid Project, 2003. [Online].

Available: http://www.edg.org. [Accessed: June 2007]/

[68] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, P. Millar, K. Stockinger, and F. Zini, “Evaluation of an Economy- Based File Replication Strategy for a Data Grid,” *In Int.Workshop on Agent Based Cluster and Grid Computing at CCGrid2003, Tokyo, Japan, May 2003. IEEE-CS Press.*

[69] M. Carman, F. Zini, L. Serafini, and K. Stockinger, “Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid,” *In Int.Workshop on Agent based Cluster and Grid Computing at CCGrid 2002, Berlin, Germany, May 2002. IEEE-CS Press.*

[70] K. Ranganathan and I. Foster, “Identifying Dynamic Replication Strategies for a High Performance Data Grid,” *In Proceedings of the Int. Grid Computing Workshop,*

Denver, CO, USA, November 2001.

[71] G. K. Zipf. *Selected Studies of the Principle of Relative Frequency in Language* . Cambridge, MA.: Harvard University Press, 1932.

[72] I. Foster and C. Kesselman, "A Data Grid Reference Architecture," GriPhyN 2001-6.

[73] The Globus Project, [Online].
Available: http://www.globus.org. [Accessed: April 2006].

[74] L. Capozza, K. Stockinger, and F. Zini, "Preliminary Evaluation of Revenue Prediction Functions for Economically-Effective File Replication," *CERN Geneva, Switzerland, Tech.Rep. DataGrid-02-TED-020724, July 2002.*

[75] C.-H. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *In International Conference on Computer-Aided Design, 1997, pp. 28–32, 1997.*

[76] Gregory R. Ganger and M. Frans Kaashoek, "Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files," *In USENIX Annual Technical Conference, 1997, pp. 1–17.*

Appendix

The Source code of the Transaction-Filtering ARM Implementation

**********Itemsets combination.h**********
****This file establishes the necessary functions for combining unique potential items to form candidate itemsets**********

```
//Itemsets combination.h

#ifndef __COMBINATION_H__
#define __COMBINATION_H__


// Non recursive template function
template <class BidIt>

bool next_combination(BidIt n_begin, BidIt n_end,
BidIt r_begin, BidIt r_end)
{

  bool boolmarked=false;
  BidIt r_marked;

  BidIt n_it1=n_end;
  --n_it1;


  BidIt tmp_r_end=r_end;
  --tmp_r_end;
  BidIt tmp_r_begin=r_begin;
  --tmp_r_begin;

  for(BidIt r_it1=tmp_r_end; r_it1!=tmp_r_begin ; --r_it1,--n_it1)
  {
    if(*r_it1==*n_it1 )
    {
      if(r_it1!=r_begin) //to ensure not at the start of r sequence
      {
        boolmarked=true;
        r_marked=(--r_it1);
        ++r_it1;//add it back again
        continue;
      }
      else // it means it is at the start the sequence, so return false
        return false;
    }
    else //if(*r_it1!=*n_it1 )
    {
      //marked code
      if(boolmarked==true)
      {
        //for loop to find which marked is in the first sequence
```

```
        BidIt n_marked;//mark in first sequence
        for (BidIt n_it2=n_begin;n_it2!=n_end;++n_it2)
          if(*r_marked==*n_it2) {n_marked=n_it2;break;}


        BidIt n_it3=++n_marked;
        for  (BidIt r_it2=r_marked;r_it2!=r_end;++r_it2,++n_it3)
        {
          *r_it2=*n_it3;
        }
        return true;
      }
      for(BidIt n_it4=n_begin; n_it4!=n_end; ++n_it4)
        if(*r_it1==*n_it4)
        {
          *r_it1=*(++n_it4);
          return true;
        }
    }
  }

  return true;//will never reach here
}



// Non recursive template function
template <class BidIt>

bool prev_combination(BidIt n_begin, BidIt n_end,
BidIt r_begin, BidIt r_end)
{

  bool boolsame=false;
  BidIt marked;//for r
  BidIt r_marked;
  BidIt n_marked;


  BidIt tmp_n_end=n_end;
  --tmp_n_end;
  BidIt tmp_n_begin=n_begin;
  --tmp_n_begin;

  BidIt r_it1=r_end;
  --r_it1;

  for(BidIt n_it1=tmp_n_end; n_it1!=tmp_n_begin ; --n_it1)
  {
    if(*r_it1==*n_it1)
    {
      r_marked=r_it1;
      n_marked=n_it1;
      break;
    }
  }

  BidIt n_it2=n_marked;
```

```
  BidIt tmp_r_end=r_end;
  --tmp_r_end;
  BidIt tmp_r_begin=r_begin;
  --tmp_r_begin;

  for(BidIt r_it2=r_marked; r_it2!=tmp_r_begin; --r_it2,--n_it2)
  {
    if(*r_it2==*n_it2 )
    {
      if(r_it2==r_begin&&*r_it2!=*n_begin)
      {
        for(BidIt n_it3=n_begin;n_it3!=n_end;++n_it3)
        {
          if(*r_it2==*n_it3)
          {
            marked=r_it2;
            *r_it2=*(--n_it3);

            BidIt n_it4=n_end;
            --n_it4;
            for(BidIt r_it3=tmp_r_end; r_it3!=tmp_r_begin
&&r_it3!=marked; --r_it3,--n_it4)
            {
              *r_it3=*n_it4;
            }
            return true;
          }
        }
      }
      else if(r_it2==r_begin&&*r_it2==*n_begin)
      {
        return false;//no more previous combination;
      }
    }
    else //if(*r_it2!=*n_it2 )
    {
      ++r_it2;
      marked=r_it2;
      for(BidIt n_it5=n_begin;n_it5!=n_end;++n_it5)
      {
        if(*r_it2==*n_it5)
        {
          *r_it2=*(--n_it5);

          BidIt n_it6=n_end;
          --n_it6;
          for(BidIt r_it4=tmp_r_end; r_it4!=tmp_r_begin &&r_it4!=marked;
--r_it4,--n_it6)
          {
            *r_it4=*n_it6;
          }
          return true;
        }
      }
    }
  }
```

```
  return false;//Will never reach here, unless error
}



// Recursive template function
template <class RanIt, class Func>

void recursive_combination(RanIt nbegin, RanIt nend, int n_column,
              RanIt rbegin, RanIt rend, int r_column,int loop,Func func)
{

    int r_size=rend-rbegin;

    int localloop=loop;
    int local_n_column=n_column;

    //A different combination is out
    if(r_column>(r_size-1))
    {
     func(rbegin,rend);
     return ;
    }
    ////////////////////////////////

    for(int i=0;i<=loop;++i)
    {

        RanIt it1=rbegin;
        for(int cnt=0;cnt<r_column;++cnt)
        {
          ++it1;
        }

        RanIt it2=nbegin;
        for(int cnt2=0;cnt2<n_column+i;++cnt2)
        {
          ++it2;
        }

        *it1=*it2;

        ++local_n_column;

        recursive_combination(nbegin,nend,local_n_column,
                rbegin,rend,r_column+1,localloop,func);
        --localloop;
    }

}

                                #endif
```

**********Transaction-Filtering ARM.cpp**********
*****The following code implements the transaction-filtering ARM**********

```
#pragma warning(disable: 4786)

#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <string>
#include "combination.h"
#include <map>

using namespace std;
//display itemsets stored in a vector
void display(vector<string>::iterator begin, vector<string>::iterator
end)
{
    for(vector<string>::iterator it=begin; it!=end; it++)
    {
        cout<<*it<<" ";
    }
     cout<<endl;
}
//write the itemsets into the output file
void vec_write(vector<string>::iterator begin, vector<string>::iterator
end, ofstream& output_file)
{
     for(vector<string>::iterator it=begin; it!=end; it++)
    {
        output_file<<*it<<" ";
    }
    output_file<<endl;
}

int main()
{
    char file_name1[100];
    char file_name2[100];
    vector<string> v;
   //the vector for storing unique items
    vector<string> v_unique;
     //the vector for storing candicate itemsets
    vector<string> v_combination;
    map<string, unsigned int> unique_item_index;
    map<vector<string>, unsigned int> temp_results;
    string str;
    string str1;
     //the counter of the row in the input files
    unsigned int row_counter=0;
     //the counter for the unique items in transaction database
    unsigned int unique_item_counter=1;
     //pre-determined minimum support
    double min_sup;
    //x is the maximal length of the expected itemsets
   int x;
    cout<<"enter the target file name:\n";
    cin>>file_name1;
    ifstream target_file(file_name1);
    cout<<"enter the results file name:\n";
```

```
    cin>>file_name2;
    cout<<"input min_sup [?]%\n";
    cin>>min_sup;
    ofstream results_file(file_name2);
    if(target_file.fail())
    {
      cerr<<"error opening target file\n";
    }
     //file scanning loop
    while(!target_file.eof())
    {
        //parsing fileds in a row
        getline(target_file, str,'\n');
        int temp=str.size();
        int j=0;
        for(int i=0; i<temp;i++)
        {
         if(str[i]==' ')
         {
          str1=str.substr(j,i-j);
          if(!binary_search(v_unique.begin(),v_unique.end(),str1))
          {
            v_unique.push_back(str1);
             sort(v_unique.begin(),v_unique.end());
            pair<string, unsigned int> p(str1, unique_item_counter);
            unique_item_index.insert(p);
          }
          else
          {
           unique_item_index[str1]++;
          }
          // store fileds in a vector
          v.push_back(str1);
          j=i+1;
         }
        }
        sort(v.begin(),v.end());
        display(v.begin(),v.end());
        v.clear();
         row_counter++;
    }
    target_file.close();
    v_unique.clear();
    cout<<endl;
   //listing the scanning results for unique items
    results_file<<"Unique items listed below:\n";
    cout<<"Unique items listed below:\n";
    cout<<endl;
    for(map<string, unsigned int>::iterator
miter=unique_item_index.begin(); miter!=unique_item_index.end();
miter++)
    {
      cout<<"["<<miter->first<<"]"<<"["<<miter->second<<"]"<<endl;

results_file<<"["<<miter->first<<"]"<<"["<<miter->second<<"]"<<endl;
    }
    cout<<endl;
    results_file<<row_counter<<" rows scanned\n";
```

```
    cout<<row_counter<<" rows scanned\n";
    double sup_min=min_sup/100*row_counter;
     results_file<<"The expected sup_min is"<<" "<<sup_min<<endl;
    cout<<"The expected sup_min is"<<" "<<sup_min<<endl;
    cout<<endl;
    results_file<<"The frequent 1 itemset are:\n";
    cout<<"The frequent 1 itemset are:\n";
     //listing all frequent size-one itemsets
     for(map<string, unsigned int>::iterator
miter1=unique_item_index.begin(); miter1!=unique_item_index.end();
miter1++)
    {
      if(miter1->second>=sup_min)
     {

results_file<<"["<<miter1->first<<"]"<<"["<<miter1->second<<"]"<<endl;
      cout<<"["<<miter1->first<<"]"<<"["<<miter1->second<<"]"<<endl;
       v_unique.push_back(miter1->first);
     }
    }
    cout<<endl;
    cout<<"input the expected length of frequent itemsets\n";
    cin>>x;
    vector<string> v_unique_gen;
    for(vector<string>::iterator viter=v_unique.begin();
viter!=v_unique.begin()+x; ++viter)
    {
      v_unique_gen.push_back(*viter);
    }
     unsigned int counter=0;
   //self-joint to form candidate itemsets untill no itemsets can be
generated
    do
    {

    vec_write(v_unique_gen.begin(),v_unique_gen.end(),results_file);
        pair<vector<string>, unsigned int> p(v_unique_gen, counter);
        temp_results.insert(p);
    }

while(next_combination(v_unique.begin(),v_unique.end(),v_unique_gen.be
gin(),v_unique_gen.end()));
    v_unique_gen.clear();
     for(vector<string>::iterator viter1=v_unique.begin();
viter1!=v_unique.begin()+x; ++viter1)
    {
      v_unique_gen.push_back(*viter1);
    }
    ifstream target_file1(file_name1);
     //file scanning loop
    while(!target_file1.eof())
    {
        getline(target_file1, str,'\n');
        int temp1=str.size();
        int j1=0;
        for(int i1=0; i1<temp1;i1++)
        {
         if(str[i1]==' ')
```

```
        {
         str1=str.substr(j1,i1-j1);
         v.push_back(str1);
         j1=i1+1;
        }
       }
       sort(v.begin(),v.end());
       display(v.begin(),v.end());
       vec_write(v.begin(),v.end(),results_file);
       do
       {
        display(v_unique_gen.begin(), v_unique_gen.end());
      vec_write(v_unique_gen.begin(),v_unique_gen.end(),results_file);
//verifying itemsets to see if they already exist in the filter
if(includes(v.begin(),v.end(),v_unique_gen.begin(),v_unique_gen.end())
)
        {
         temp_results[v_unique_gen]++;
         results_file<<"found\n";
         cout<<"found\n";
        }
        else
        {
         results_file<<"not found\n";
         cout<<"not found\n";
        }

       }
while(next_combination(v_unique.begin(),v_unique.end(),v_unique_gen.be
gin(),v_unique_gen.end()));
        v_unique_gen.clear();
       for(vector<string>::iterator viter=v_unique.begin();
viter!=v_unique.begin()+x; ++viter)
       {
         v_unique_gen.push_back(*viter);
       }
        v.clear();
   }
   cout<<endl;
   do
   {
   display(v_unique_gen.begin(),v_unique_gen.end());
   }
   while(next_combination(v_unique.begin(),v_unique.end(),v_unique_gen
.begin(),v_unique_gen.end()));
   v_unique_gen.clear();
    //display frequent size-k itemsets
   for(map<vector<string>, unsigned int>::iterator
miter2=temp_results.begin(); miter2!=temp_results.end(); miter2++)
    {
     if(miter2->second>=sup_min)
     {

      cout<<miter2->second<<endl;
     }
    }
   return 0;
}
```

The EDG Testbed Configuration File

```
# The network configuration for the EU Data Grid Testbed
# no of CEs, no of SEs, SE sizes, site vs site bandwidth
#
#                Imp  Coll     UK      Swed    NIKHEF   Holl    Germ    Fran    Lyon
CERN     Swis    Italy   Padova  Bolog   Catan   Torino  Milano  RAL     Nordu
1 1 80000 00000.0 02500.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 02500.0 00000.0 10000.0 00000.0 02500.0 00000.0 10000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 02500.0 00000.0
0 0000000 00000.0 10000.0 00000.0 00000.0 01550.0 10000.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 01550.0
1 1 33000 00000.0 00000.0 00000.0 00000.0 00622.0 00000.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 00000.0 02500.0 01550.0 00622.0 00000.0 02500.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 00000.0 00000.0 10000.0 00000.0 02500.0 00000.0 00000.0 00000.0 00000.0
10000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 00000.0 10000.0 00000.0 00000.0 00000.0 00000.0 00000.0 02500.0 00000.0 10000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 50000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 02500.0 00000.0 01550.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 1 100000000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 01550.0 00000.0
01000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 10000.0 00000.0 01000.0 00000.0
10000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
0 0000000 00000.0 00000.0 00000.0 00000.0 00000.0 10000.0 00000.0 00000.0 00000.0 10000.0
00000.0 00450.0 01550.0 00100.0 00450.0 00100.0 00000.0 00000.0
1 1 63000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00450.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 30000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
01550.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 30000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00100.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 50000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00450.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 50000 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00100.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 50000 00000.0 02500.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
1 1 70000 00000.0 00000.0 01550.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0 00000.0
```

The OptorSim Configuration File Used in the Simulations

```
#
# This file contains all the parameters required for OptorSim
# using the Properties class to store this information.
#
grid.configuration.file =
/home/optorsim/optorsim-2.0/examples/edg_testbed_grid.conf
job.configuration.file =
/home/optorsim/optorsim-2.0/examples/edg_testbed_jobs_attributes.conf
bandwidth.configuration.file =
/home/optorsim/optorsim-2.0/examples/edg_testbed_bandwidths.conf
number.jobs = 1000
#
# The choice of the scheduling algorithms for the RB is:
# (1) random
# (2) queue length
# (3) access cost for current job
# (4) access cost for current job + all queued jobs
scheduler = 4
#
# The categories of users available are:
# (1) Simple - wait job delay between submitting jobs
# (2) Random - wait uniform random time between 0 and 2 * job delay
# (3) CMS DC04 Users
users = 1
#
# The choice of optimisers is:
# (1) SimpleOptimiser - no replication.
# (2) LruOptimiser - always replicates, deleting least recently
#created file.
# (3) Decision-tree-based predictive model - always replicates, deleting
#least frequently file based on the decision rule table.
# (4) EcoModelOptimiser - replicates when eco-model says yes, deleting
#least valuable file.
# (5) EcoModelOptimizer Zipf-like distribution
optimiser = 2
#
# automatically multiplied by scale factor
#
dt = 1000000
#
# The choice of access pattern generators is:
# (1) SequentialAccessGenerator - Files are accessed in order.
# (2) RandomAccessGenerator - Files are accessed using a flat random
#        distribution.
# (3) RandomWalkUnitaryAccessGenerator - Files are accessed using a
#        unitary random walk.
# (4) RandomWalkGaussianAccessGenerator - Files are accessed using a
#        Gaussian random walk.
# (5) RandomZipfAccessGenerator - Files are accessed using a
#         Zipf distribution
#
access.pattern.generator = 1
# Shape parameter for Zipf-like distribution > 0
shape = 0.85
#
```

```
# The job set fraction is the number of files accessed per job.
#
job.set.fraction = 1
#
# The initial file distribution is either "random" in which case the
# master files are distributed randomly to SEs or numbers separated
# commas (e.g. 8,10,15) giving the sites where all the masters
# should be evenly distributed.
# (in edg testbed site 8 is CERN)
#
initial.file.distribution = 8
#
# Do we want to fill all the SEs with replicas?
#
fill.all.sites = no
#
# The scale factor is used to speed up the simulation by reducing the
# file sizes. If it is too small the timing will be inaccurate.
#
scale.factor = 1
#
# The job delay is the interval in ms between the RB submitting each job.
# Automatically multiplied by scale factor
#
job.delay = 5000
#
# The random seed for deciding which jobs are chosen can be random or
# fixed.
#
random.seed = no
#
# The maximum queue size is the maximum number of jobs the JobHandler
# will keep in its queue.
#
max.queue.size = 200
#
# The time (in ms) it takes each file to be processed (this is
# divided by the number of worker nodes on the site.)
# Automatically multiplied by scale factor
#
file.process.time = 1000
#
##################
# Auction stuff #
##################
#
auction.flag = no
# actually the number of sites contacted
hop.count = 50
timeout = 500
timeout.reduction.factor = 0.4
#
# Outputs auction information to auction.log. Can slow the
# simulation down a little bit.
#
auction.log = no
#
#############################
```

```
# BandwidthReader stuff     #
#############################
# flag to switch background traffic on or off
background.bandwidth = no
#
# The directory in which your background bandwidth data files are stored
data.directory = examples/bw_data/edg_testbed/
#
# The datafile to use when no other background data are available.
# For EDG, lyon_to_cern_ave.numbers is good; for UK dl_to_ncl_ave.numbers
is good.
default.background = lyon_to_cern_ave.numbers
#
# The time of day used as starting point. Should be in hours, with minutes
after
# the decimal point e.g. 22.5 for 22:30, and must be on the hour or half-hour.
time.of.day = 0.0
#
#############
# GUI stuff #
#############
#
# Options to use the GUI and histogram browser
#
gui = yes
histogram.browser = yes
#
# The file with the map information
#
map.info = examples/gui/europe.coords
#
##############
# Statistics #
##############
#
# Level of statistics to be printed out at the end of the simulation
# (1) None
# (2) Simple - only stats for the whole grid
# (3) Full - full stats for all elements on all sites
#
statistics = 2
#
##############
# Time Model #
##############
#
# use advanced grid time (yes) or not (no)
#
time.advance = yes
#
# end
#
```