Data mining on grids.

Maarten Altorf

maltorf@yahoo.com

Universiteit Leiden

August 2007

Contents

1 Introduction

Data mining is a hot topic. More and more companies, government institutions and educational institutions gather large amounts of data each day with the intention to find useful information. For extracting useful information and patterns out of this data data mining is used. But with the ever growing databases with data it takes more and more time to mine the data; many different organizations are working together, such as university research groups, data is more spread out over multiple locations. Most of these organizations have good network connections and sending over small amounts of data or information is not a problem, however sending over a dataset as large as a Terabyte might be a problem. For this type of problems computational grids are used: there is much more computer power if a thousand computers are used instead of just a few and this can reduce time for the data mining process. Another advantage of grids is that they can cope with distributed data. In this thesis we would like to test grid enabled data mining applications, and see if these applications can speed up the data mining process as well as deal with data distributed among multiple sites.

The thesis is organized as follows. Section 2 gives an introduction on data mining. It describes knowledge discovery in databases, and in more detail the data mining process including the tasks, models and algorithms. Section 3 describes grid computing and the Globus toolkit, a grid computing framework. In section 4 we discuss several grid enabled data mining application. The following applications are taken into consideration: GridMiner, WekaG, Weka4WS, DataMiningGrid and GridWeka 2. Section 5 is used to describe an algorithm that we grid enabled, the Depth-First Apriori frequent itemset algorithm. The last section contains the conclusions.

2 Data mining

Lets start with the word data: data literally means "that which is given" and it refers to raw facts, measurements, numbers, etc. We would like to transform the data into something that is more useful: information (knowledge and concepts). Information is the result of processing, manipulating and organizing data in a way that it creates meaning and hopefully knowledge for the person who receives it. **Data mining** can therefor be defined as: "The nontrivial extraction of implicit, previously unknown and potentially useful information or patterns from data"[10]. This data resides most of the time in very large databases, a **database** is an organized collection of information records stored on a computer so that its contents can easily be accessed, managed and updated. The data mining definition begins with nontrivial extraction which means that we extract information from the data in an intelligent way and not in a way that we end up with obvious information. It goes on by saying that the information has to be implicit which means that we are looking for information that is not explicitly stated in the data. We are looking for previously unknown information because most of the time it is useless to look for something we already know. Finding previously known patterns can help to improve the confidence of the algorithm and can also be useful when you are not completely sure about the information you gathered before. And last but not least we are hoping to find potentially useful information, we are not looking for useless information that we do not want, but sometimes it is not clear beforehand if the information that will be found will be useful.

Companies, governments, research and educational institutions collect vast amounts of data in many different areas. Areas like marketing, business processes, medical environments, consumer behavior, astronomy and many others. Data mining looks for patterns in these different kinds of data. Let us take the consumer behavior as an example. Most people are familiar with the concept of bonus cards, discount cards, air-miles cards, club cards. Companies collect this information using these cards because they want to know a lot: what you buy, when you buy it, where you buy it, etc. With all this data these companies are hoping to get some useful information which they can use to improve their products, advertisement and sales. Useful information for these companies can be: which products are sold most, in which part of the country do people buy one particular kind of product, which products are often sold together, what are the results of advertisements and discounts, etc. Getting this useful information can be done by applying data mining techniques on the raw data to look for interesting patterns. How this can be done is explained in the rest of the chapter.

2.1 KDD

Data mining algorithms are not the only thing involved in the data mining process. There is a lot of work to be done before and after the actual mining of the data. Most of the time the actual data mining part is done in 10% of the

total time of the process, the other aspects take up the rest of the time. Because the process is more extensive than only data mining the process is often refered to as knowledge discovery in databases (KDD).

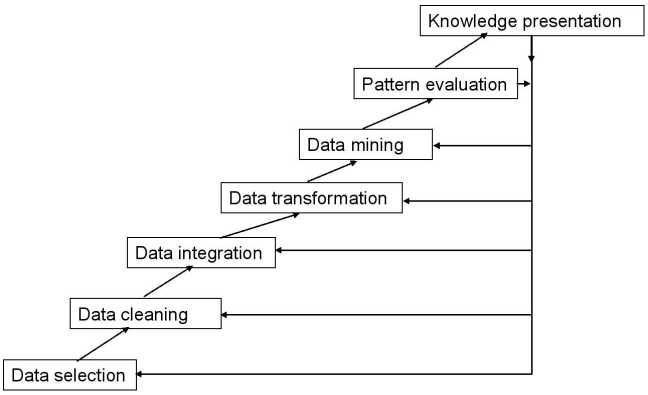


Figure 1: Knowledge Discovery Process

Figure 1 shows the different tasks in the KDD process. Note that there are arrows back from Pattern Evaluation and from Knowledge Presentation: it means that at the end of the process when you see the results, it might be necessary to go back and change some of the steps. For example if the results are not very interesting, then one could change the data mining algorithm and see if another one finds something more valuable.

We next discuss the steps of the KDD process in more detail:

- **Data selection** is the step where relevant data for the analysis task is selected from the data source to form the subset that is to be mined. In many cases it is not necessary to use all the variables available, blindly including all the columns can lead to incorrect models. First take a look at the variables and what they mean and then a better judgment can be made whether or not to include it in the analysis. For example when predicting age the variable date of birth is the perfect predictor but it is not a very useful predictor. If the data source is very large it might be necessary not to use all the variables and all the rows, otherwise it would take too much time to do the analysis.

- **Data cleaning** is the process of altering the raw data in order to make it as clean as possible. Clean data depends on the features relevant for the study, so for multiple studies one data set can be clean in different ways. Data cleaning can take between 60% and 80% of the total time of the KDD process [11]. The process copes with:

  - Errors in the data, e.g. values are entered with errors, names are spelled incorrectly, noise (outliers) in the data.

  - Duplicated data with inconsistencies. The first case is data that is entered repeatedly and possibly with different values. The second case is when a real world entity is entered twice but with different values, this problem is not easy to detect because it looks like two different entities.
  - Missing values in the data where values are expected.
  - Heterogeneities in data, this arises when two or more systems are brought together to form one system. In one case the meaning of the same data is different in various systems. The other case is when the structure or schema of the systems that are put together are different.
  - Irrelevant data that is to be denied when the data mining algorithms are applied to the data. Thus removing data from the data set which the user does not want to use for the experiment.

- **Data integration** is the stage in which multiple, often heterogeneous, data sources are integrated into one common source.
- **Data transformation** is the stage where the data is transformed to the appropriate format for the analysis, e.g. numbers can be transformed from integers to reals.
- **Data mining** is the step where the actual data mining is done. Intelligent and non-intelligent(simple) methods are applied to extract patterns from data source(s); more information will be given in the next section.
- **Pattern evaluation** is the phase where the interestingness of discovered patterns is identified and measured. There are some concrete methods available like measuring the support and the confidence of a pattern. One weakness of these objective measurements is the lack of (human) background knowledge, the so called domain knowledge. It is possible that by a concrete measurement a pattern does not look interesting while in fact it is an interesting pattern if a user with domain knowledge looks at it.
- **Knowledge presentation** and visualization are at the end of the KDD process. In this stage knowledge is presented to the users, and good data visualization is important for the user to be able to understand and interpret the outcomes of the process. Presentation and visualization can be done in many different ways; for some studies one way of presenting is useful while for other studies a different way of presenting the data is desirable.

2.2 Data mining tasks

We list a number of different data mining tasks:

- **Summarization** data mining maps data into groups with some common description and characteristics. This can be used as a first step of data

mining. First gather some simple information like averages, standard deviation, frequency distribution etc. Then use this information to continue the analysis with some other task.

- **Clustering**, a more advanced method of summarizing data, divides a data source into groups that are not known beforehand (unlike Classification). The goal of clustering is to determine the intrinsic grouping in a data set, in other words to find groups that are different from each other and members within one group to be similar. Training data is used to build or train a model, clustering is a form of Unsupervised learning, the training data does not specify beforehand what we are trying to learn. Clustering is used to visualize the data, to get a feeling of how the data looks like. Clustering can be used as a stand alone data mining task, but it can also be used as the data preprocessing step on which to build predictive data mining models.

- **Predictive data mining.** There are two techniques that are often used for predictive data mining: one is classification and the other is regression. Classification is a technique that can be used to predict in what class a case falls. Regression is a technique that can be used to predict what value a variable will have. Predictive data mining is a form of supervised learning, the training data has to specify what we are trying to learn. The prediction is based on a model, and the model is generated/build by an algorithm. The algorithm builds up a model using training data of which the value or class is already known. A test data set is used to test the model; the test data set is dependent of the training data. The resulting models are used to predict data of which the class or value is not yet known. The most common models used for prediction are decision trees, neural networks, general linear models, classification rules and Instance based learning.

- **Link analysis** is an approach to find relationships among data. Market basket analysis is a common example in link analysis. Link analysis in the market basket case means searching for relationships in data of shops like supermarkets or (online) bookshops. A relationship that could be found is "If people buy bread they also buy butter" or "If people buy beer and chips they also buy peanuts". These rules are examples of rules that can be found by association discovery. The other link analysis technique is sequence discovery, this technique looks for sequential patterns in data, e.g. shopping patterns, phone call patterns and DNA sequence patterns. In the market basket analysis case a sequence means the subsequent purchase of a product given a previous buy. For example buying a dvd player following the purchase of a tv.

- **Outlier analysis** is the discovery of rare or unusual events or objects in data. It searches for objects that do not comply to the general behavior of the data. These events or objects can be considered as noise in some

data sets, but they can be important in rare events analysis such as fraud detection. The general approach for finding these outliers is the use of distribution and deviation analysis.

2.3 Models and Algorithms

We next address the question of how a data mining technique can get useful information out of a big pile of data. There are several kinds of data mining techniques but basically all try to find patterns and regularities that frequently occur. Next we will describe a number of output representations, each with an example algorithm.

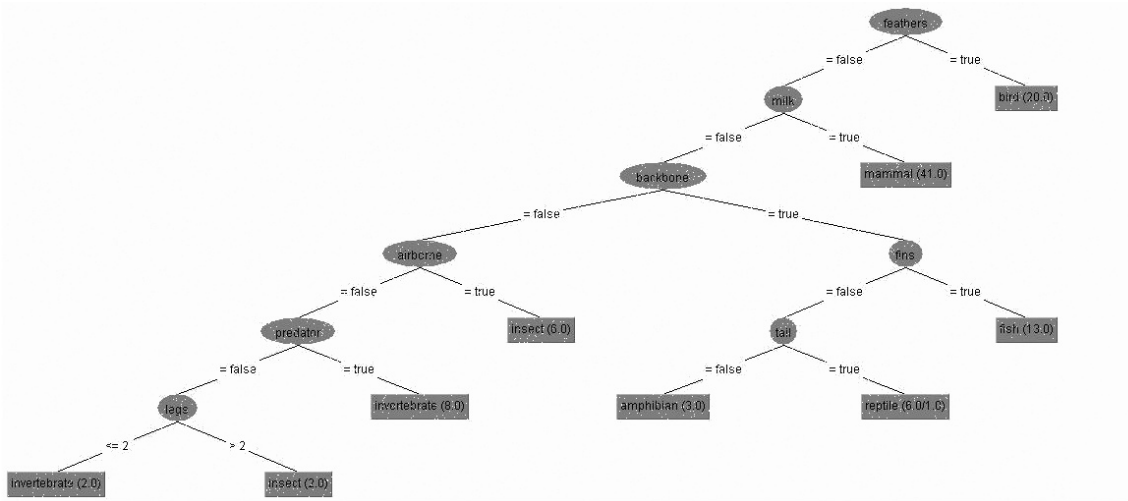


Figure 2: A decision tree (made with Weka J48 zoo)

Decision trees are a way of representing the output as a series of rules that lead to a class or value. Decision trees can be used for classification as well as for regression. The internal nodes of a tree test the attributes; an attribute can be tested with a constant value and they can be tested with other attributes. These internal nodes can test nominal and numeric values. With nominal values it usually branches for each value of the attribute. For numeric values it is common to branch into two ways, greater or less than a value. Generally the nominal attributes are tested only once in a branch while numeric attributes can be tested more often to make a further split. The leaf nodes of a tree represent the class or the value of all the instances that reach that leaf. To reach a leaf an instance needs to start at the root of the tree and go down the tree according to the nodes it passes. Examples of algorithms that build up a tree are J48, CART, C5.0 and CHAID.

Classification rules make predictions according to rules. A rule set may look like this:

If a then b
If c and d then e

The preconditions of a rule is a set of tests just like the internal nodes of a decision tree. And the conclusion of the rule is a class or value just like the leaves in a decision tree. It is easy to transform a decision tree into a set of rules, you generate one rule for each leaf. The other way round is more complicated because trees are not good at separating two rules. Here is an example of this complication.

If a and b then x
If c and d then x

These rules can be translated into a tree but not in a straightforward and efficient way. This is because a tree has to start with an attribute to compare, lets say a. If a is not true the tree can go on with the second rule, if a is true is must test b. If b is true then x is true, but if b is not true then c & d has to be tested just as in the first breach of the root node. This way there arises a tree with two identical subtrees, see figure 3.

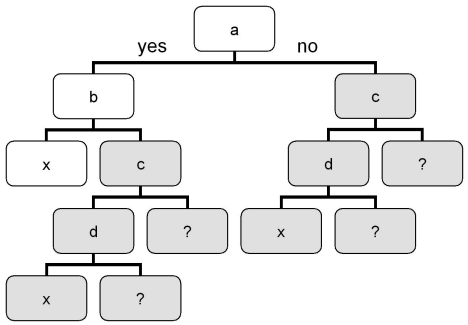


Figure 3: Two identical subtrees

Translating this tree into rules one obtains the following rule set; as one can observe there is one extra rule regarding the original rule set.

If a and b then x
If a and not b and c and d then x
If c and d then x

Examples of classification algorithms are Bayes, OneR and ZeroR [9].

Association rule discovery is a form of link analysis. Association rules are almost the same as classification rules except that they can predict attributes instead of classes. An association rule tells us something about the association between two or more attributes; it can predict one or more attributes. Here are some examples of association rules:

If milk and butter then cheese and bread

If chocolate custard then whipping cream

If Harry Potter 1 and Harry Potter 2 and Harry Potter 3 then Harry Potter 4 and Harry Potter 5

Rules in one association rule set do not express similar regularities in the data set. We can filter out the more interesting and useful rules by calculating the support and the confidence and setting threshold for these values. The support of a rule is the number of instances it predicts correctly. The confidence is the proportion of instances it predicts correctly given the correct preconditions. Apriori is the best known association rule algorithm that finds rules above a given support and confidence, we will discuss the Apriori algorithm in more detail in chapter 5.

Instance bases learning is completely different from the methods we have seen before. This learning does not compute or produce a model; the instances themselves form the model. Instance base learning is a way of lazy learning, it starts the work when it is needed, the moment there is a new instance. Opposed to eager learning what the other methods of representation do, they produce the model as soon as the data has been seen. Instance base learning searches for the training instance that mostly resembles the new instance and then assign the class or the value of that instance to the new one. Which training instance mostly resembles the new one is computed using a distance metric like the standard Euclidean distance. The distance can be computed for one or more attributes. If the attributes are nominal the distances are not so obvious. What is the distance between a bird and a fish? Frequently the distance between two nominal attributes is 0 when they are the same and 1 if they are different. Examples of instance based learning algorithms are IB-K and K-nearest neighbor [9].

Clusters represent the data in a form of a diagram with the instances divided into clusters. Figure 4 shows a simple case in which each instance is assigned precisely one cluster. Some clustering algorithms allow instances to be assigned to one or more clusters; this output is represented as a Venn diagram, see figure 5. In a Venn diagram each ellipse represents one cluster. As you can see there are instances such as a and k that are assigned to only one cluster, some are assigned to two such as i and j and then there are instances that are assigned to all three clusters such as d.

There are also algorithms that give each instance a probability measure of membership for each cluster. This means that an instance can have a probability of 0.5 to belong to one cluster, 0.3 to a second cluster and so on. Dendrograms are generated by the last sort of clustering algorithms, figure 6 is an example of a dendrogram.

Dendrograms show the order of the clusters hierarchical. The algorithms divides the instances first in a few clusters, and goes on by dividing these clusters into smaller cluster. Instances that are in one cluster on the bottom of the dendrogram are very similar while instances that are in different clusters in the top of the dendrogram are most different from each other. Examples of clus-

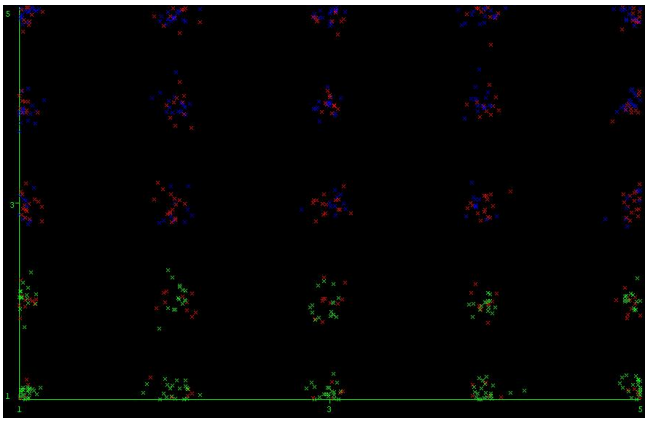


Figure 4: Example of a cluster output (Weka, k-means, 3 clusters, balance-scale dataset)

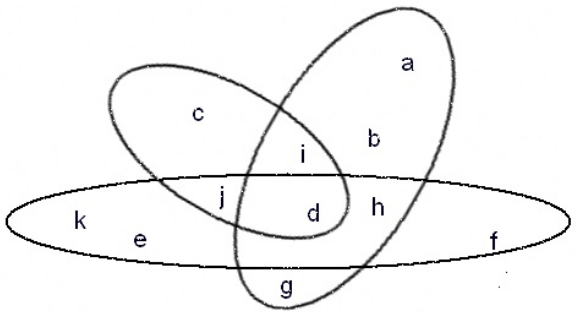


Figure 5: Example of a Venn diagram

tering algorithms are K-means, Kohonen self organizing maps and Hierarchical clustering.

Neural networks are inspired by the human brains in that it mimics inter neuron connection strengths known as synaptic weights. A neural network acquires knowledge trough learning. We discuss here only a so-called feedforward architecture. The input training data with the corresponding desired output is put on a neural network over a number of epochs(iterations); this enables the network to learn the classification or value of the training data. Each epoch the networks synaptic weights are adjusted to optimize the network. The final goal of the network is to correctly classify or assign the correct value to instances of

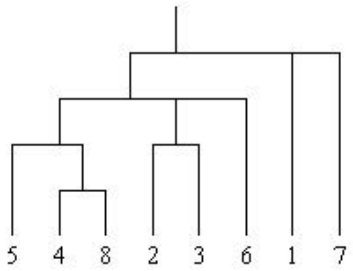


Figure 6: Example of a dendrogram

which the class or value is unknown. Figure 7 is an example of a feedforward, multilayer neural network made within Weka.

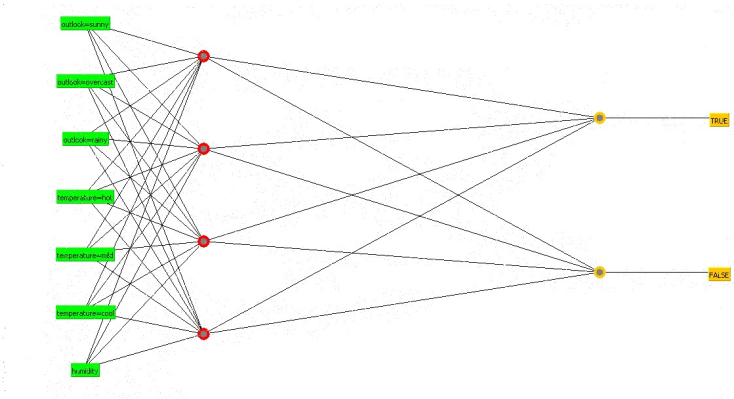


Figure 7: Example of a neural network (Weka, weather data set)

3 Grid computing

The idea behind grid computing is that one can plug one's computer into the wall and have access to computational and data resources without knowing where they are or who owns these resources. The term grid is stemming from the field of electricity network which provides a power grid one can use by plugging a power cable into the wall socket, this way getting the electricity that is needed without knowing where it comes from. Grid computing, simple stated, is taking distributed computing to the next level. So first a short definition of distributed computing followed by the definition of grid computing. Distributed computing means dividing tasks among multiple computer systems instead of doing the tasks on one centralized computer system. Distributed computing is a subset of grid computing, grid computing encompasses much more. Grid computing provides coordinated sharing of geographically distributed hardware, software and information resources, this sharing is highly controlled defining clearly what is shared, who is sharing and the conditions of the sharing, it provides a service oriented infrastructure and uses standardized protocols to accomplish this sharing. The set of individuals or organizations defined by these sharing rules is called a Virtual Organizations (VO)[1]. The essence of grid computing is captured by the following three point check list of Foster [1]:

1. A grid is a system that coordinates shared resources that are not under centralized control. It coordinates and integrates resources that are in different domains and together they form a VO.

2. A grid is a system that uses standard, open, general-purpose protocols, interfaces, API's and tools that address fundamental issues such as authentication, authorization and resource access. By using these standards it is easy to be adopted by other application.

3. A grid is a system that delivers nontrivial qualities of service. Which means that the system delivers various qualities of service, such as security, response time, throughput and availability so that the utility of the combined systems is greater than the sum of its parts.

A grid needs to have some basic functionalities in order to be qualified as a grid by Foster's checklist:

- **Resource discovery and information collection & publishing**. Resource publishing is the act of letting the grid know that a particular resource is online and that it can be used. When a client request service from the grid the resource discovery process locates the resources that satisfy the resource requests of the users. To locate these resources the discovery process uses the so called resource catalogues or registries which contain information about all the available resources. Most of the information in these catalogues is metadata, data about data. In grid computing metadata is the information that describes a resource when it is published; the

minimum information that must be specified of a resource in the metadata is its name, physical location and ownership.

- **Data management** on and between resources. The grid must provide ways to manage the resources across multiple, heterogeneous environments and must ensure that the catalogues are always available and up to date. Data management must also take care of backing up and recovering of data sources without loss of service [13].
- **Process management** on and between resources. This is the creation, monitoring, execution and management of processes; a process is the use of the grid by a user.
- **Process and session recording/accounting**. Recording all processes and sessions can be important for accounting purposes, especially when there is a charge for using the grid, either for storing, accessing, doing computations or other functionalities of the grid.
- **Security** mechanisms underlying the above functionalities. General security issues include protection against hackers and viruses, authentication, authorization, privacy, data integrity (backups) etc.

The architecture of grids is often described in terms of layers. The lower layers being the computers and the networks and the higher layers being more focused on the user and the applications.

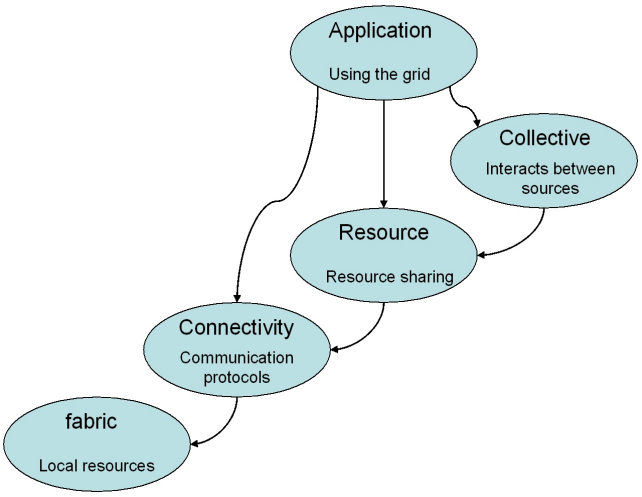


Figure 8: Architecture of the grid

Figure 8 shows the grid architecture in layers. These layers will be described below (starting from the lower levels).

The **fabric layer** provides the local resource specific operations on resources that are shared on the grid, e.g. computational, network, catalogues and storage resources.

The **connectivity layer** provides the core communication and authentication protocols. It enables save and reliable data exchange between fabric layer resources.

The **resource layer** enables resource sharing, it builds on the connectivity layer to control and access resources. It uses information protocols to obtain information about resources, and it uses management protocols to negotiate access to shared resources.

The **collective layer** coordinates interactions between multiple sources, it ties multiple sources together. E.g. it can combine data sources from multiple sites into one virtual data source, it can perform computations on multiple sites and return the results back to the user.

The **application layer** holds the application of the user, this layer uses the connectivity layer, the resource layer and the collective layer to perform grid operations in virtual organizations.

The connectivity, resource and collective layer are often called the middleware layer and are implemented by middleware software. Middleware is software that connects software components or applications, it is often used for complex, distributed applications and can be seen as the intelligence that brings all the elements together. A very well known implementation of grid middleware is the Globus Toolkit.

3.1 Globus Toolkit

Globus is a community of organizations, users and developers that work together on the use and development of fundamental technologies behind grids and the associated documentation. The Globus Toolkit (GT)[14] is an open-source software toolkit for developing service oriented grid infrastructures and applications. The GT is one of the most widely used toolkits for developing grid applications, some of the more extensive projects are Teragrid, Open Science Grid, LHC Computing Grid and China Grid. The first GT was designed to meet the demands of Virtual Organizations, especially in science. The newer version of the GT is not only designed for science, as commercial applications are becoming more important nowadays. The GT is designed to make applications be able to combine resources from distributed sources, resources like computers, storage, data, sensors, software, services, networks etc. Combining resources is motivated by the fact that it is not always possible to replicate resources locally, e.g. a large volume of data that is needed is located on a different geographic location then where the user is working, a scientist needs to do experiments on remote equipments, a scientist needs to do some computation for which he needs computer power much more than what is available at his geographic location. These examples show that every application has its own requirements but that there are some functions that grid application frequently need, which are stated in the previous section. It is important for interoperability reasons

that these implementations are widely adopted. Interoperability means that different systems are able to communicate, e.g. a relational database and an object database. Globus implements these common functions and uses an open source model to encourage contributions and adoptions.

3.2 Techniques

OGSA The Open Grid Services Architecture defines grid services as an extension of web services for a standard model to use grid resources [17]. Every resource is represented as a grid service: a Web Service that conforms to standard conventions and supports standard interfaces. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network, focusing on simple internet based standards such as the Simple Object Access Protocol (SOAP) and the Web Services Description Language (WSDL). OGSA provides a well defined set of basic interfaces for the development of interoperable grid systems. OGSA is an implementation of the service oriented architecture (SOA) model within the grid context [17]. SOA is a programming model to build flexible, modular and interoperable applications. The emphasis of grid computing shifted from intensive computing tasks to data intensive tasks. That is one of the reasons that OGSA-DAI was created, it is the database access and integration service. It allows data sources to be accessed via web services and it makes it possible to integrate data from various sources.

WSRF, Web Service Resource Framework The WSRF defines a standard specification to merge grid and web technology, this way building a bridge between the grid and the web[17]. OGSI was the predecessor of WSRF, it was accepted by the grid community but not by the Web Service community because it does not work well with existing Web Services, and therefore a new standard was developed. WSRF defines specifications to access and managing stateful resources using web services[18]; a stateful resource is a resource that can keep track of its state for multiple clients. The Globus Toolkit 4 contains Java and C implementations of WSRF, while GT3 contained OGSI implementations.

4 Overview of existing data mining applications on the grid

Data mining on Grids is an interesting and very promising research area. Data mining is already in a mature phase and more and more companies and scientists are using it on ever growing data sets. Grid technology is also getting to a more mature phase, the two techniques combined is a promising research area. Why data mining on a grid? One reason is that data mining is used on very large data sets and the time to execute a data mining technique on such a large set is taking more and more time, the execution time can become less when using a grid to do the computations on instead of a single computer. The second reason is when data mining is done on data sets residing on different geographical locations, a grid can be used to integrate the data sources into one virtual data set and then perform the data mining algorithm on it. There have already been studies in this new research area and there have been some programs developed that can perform data mining tasks on grids. Here is a list of the requirements for an application for distributed data mining on the grid [5].

- The application must be based upon an open architecture such as OGSA, this makes the application more extensible for the community.
- The application must be able to cope with distributed data, large data sizes, highly dimensional data sets (each item has many features) and heterogeneity data sets (data sets with different formats or schema's).
- It must be compatible with existing grid infrastructure. The (higher levels of the) application must use the basic grid techniques to be compatible with other grids.
- The application must be open to new data mining techniques and algorithms, so that it is easy to integrate new ones.
- Scalability of the application in terms of the number of users that can use the application and the resources at the same time, e.g. the number of nodes on which the computations are done.
- End users of the application do not have to worry or know any details about the underlying techniques of the grid, the network and the physical locations of the data sources.
- Security is a very important issue in the area of data mining, because very often the data that is mined is sensitive, personal and expensive data, e.g. data of patients in a hospital. The basic grid architecture takes care of many security issues such as how to send data over a network in a secure way. But the application has to take care of more application/subject specific security measurements. Such as filtering sensitive data, one user can see one part of the data and another user can not see that part of the data, and authorization, which user is authorized to work with which data sets.

- It must support OLAP (On Line Analytical Processing) and data warehousing. OLAP is an approach to quickly (within seconds) answer analytical questions; it involves large amounts of diverse data. OLAP is used to tell you what happened and data mining answers the question why it happened and then can be used to predict the future. Data mining and OLAP together can be used to answer the what and why questions. A data warehouse is a single database that integrates data from different sources, and is designed to support management information, analysis and answering business questions. It is a decision support database and it is not the same as the operational database of an organization. Often OLAP operations are done on the data warehouse instead of the normal database. Data mining, data warehousing and OLAP are complementary methodologies and can often strengthen one another.

4.1 GridMiner

The GridMiner application [4, 5, 6, 7, 8] is made for the development and runtime execution of data mining processes and data mining preprocessing on grids. It is a Service oriented grid application that integrates all aspects of the data mining process: data cleaning, data integration, data transformation, data mining, pattern evaluation, knowledge presentation and visualization. Goal of the GridMiner application: an easy to use tool for an expert data miner to ease the process of data mining on a grid system.

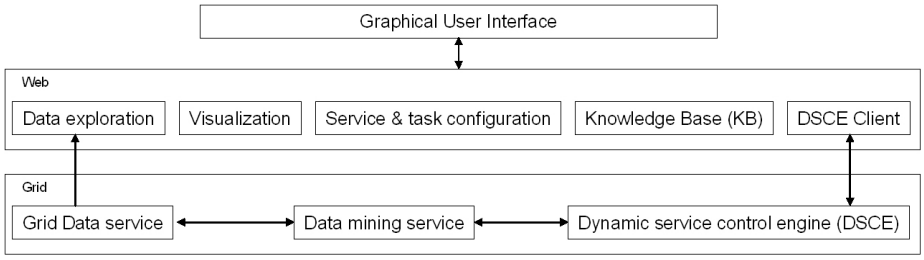


Figure 9: Global Architecture of the GridMiner application

In this chapter we will describe the GridMiner application according to figure 9. This figure shows the architecture of the application divided in three layers: the user interface, the web and the grid layer. The most important and interesting layer is the grid layer; we will describe this layer first and proceed with the other layers.

Grid Layer

The grid layer takes care of the execution of the data mining algorithms, the data preprocessing and the OLAP services. The execution is directed by the workflow engine and is supported by services such as the mediation service, security service and the file and database access service. These and more services, see figure 10, will be discussed in the remainder of this section.

The **workflow** service is an important part of the GridMiner application. A workflow is a component that combines different KDD activities into one job. It is used to ease the KDD process, especially for long running and complex jobs. Figure 11 is an example of such a workflow.

The GridMiner **Orchestration** service acts as workflow engine that executes the steps of the workflow sequential or in parallel[5]. The service is responsible for the creation of workflows, the handling of failures and interacting with optional components such as the resource broker and replica management.

The Dynamic Service Control Engine (DSCE) processes the workflow according to the DSCL file. DSCL is the Dynamic Service Control Language and it is used for the description of workflows, DSCL is easy to use and based on XML. This is an example of such a file:

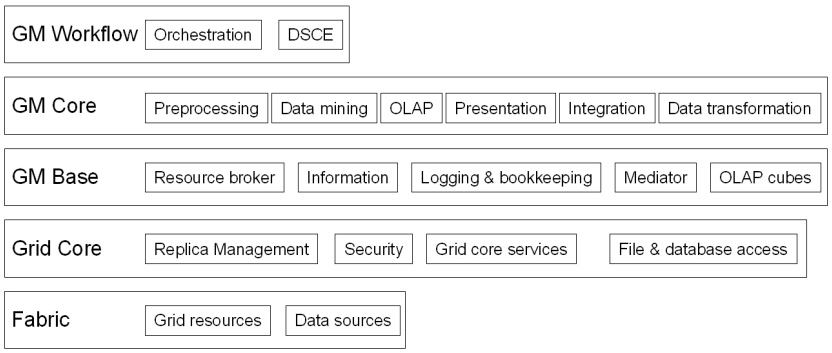


Figure 10: The grid layer of the GridMiner Architecture.

```
<dscl>

<variables>
<variable name="PERFORM_DOCUMENT">
<value>
<gds:gridDataServicePerform>
<gds:sqlQueryStatement name="myQuery">
<gds:expression>select * from test</gds:expression>
<gds:webRowSetStream name="myQueryOutput"/>
</gds:sqlQueryStatement>
</gds:gridDataServicePerform>
</value>
</variable>
<variable name=PERFORM_RESULTS"/>
</variables>

<composition>
<sequence>
<createService activityID="START"
factory-gsh=http://localhost:89/ogsa/services/ogsadai/GDSF/>
<invoke activityID=DAI001" operation="perform">
<parameter variable="PERFORM_DOCUMENT"/>
<result variable="PERFORM_RESULTS"/>
</invoke>
</sequence>
</composition>
```

The **GridMiner Core** is implemented on top of middleware [7], in this case the service oriented Globus Toolkit.

- GMPPS, GridMiner PreProcessing Service. Data mining preprocessing

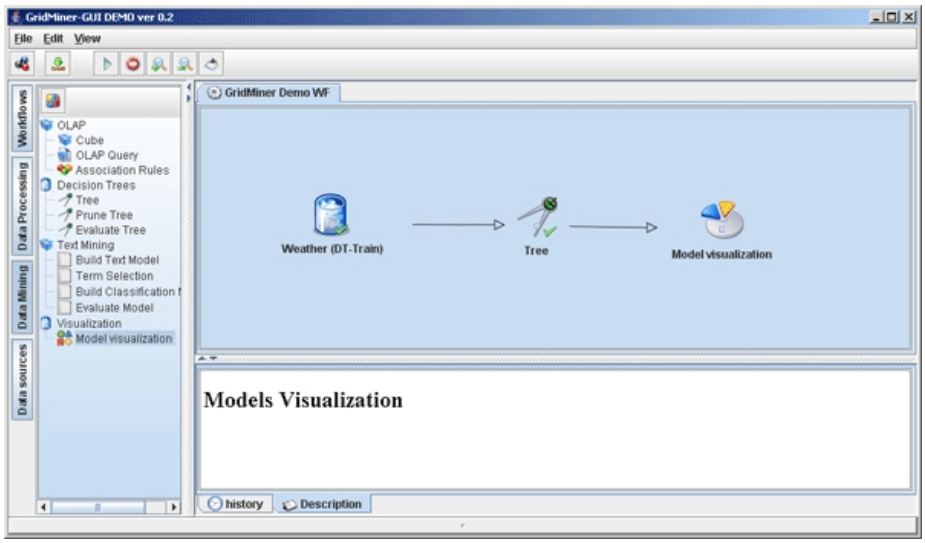


Figure 11: Screenshot of the GridMiner GUI with an example workflow.

tasks are all the tasks before the actual data mining task such as data selection, cleaning and integration. See 2.1.

- GMDMS, GridMiner Data mining service, performs several data mining tasks
  - Each data mining service is implemented as a grid service specified by OGSA.
  - The input is in the XML WebRowSet[1] (javax.sql.rowset.WebRowSet) format. WebRowSet is used to communicate with a database and can be used to easily transform data from databases to XML files and vice versa.
  - All the results are in the Predictive Model Markup Language (PMML). PMML is an XML based language developed by the Data mining Group[2], it provides a way for applications to share models. This means a user can use one application to make a model and output it in PMML, and use another application to visualize and analyze the model.
  - Currently implemented data mining services:
    - Sequential Clustering Service (SimpleKMeans)
    - Sequential Sequence Service (SPADE)
    - Distributed Decision Rules Service (SPRINT)

[1] http://www.forteach.net/Java/JDo/200611/33688.html
[2] http://www.dmg.org

- GMOMS, GridMiner OLAP Mining Service.
  - Parallel OLAP
  - Sequential Association Rule Mining in OLAP cubes
- GMPRS, GridMiner Presentation Service, can present the model in different representations such as charts, trees and association rules.
- GMDIS, GridMiner Data Integration Service (see) [8]. The integration service is responsible for the secure, reliable and efficient data transfers within a grid environment.
- GMDT, GridMiner Data Transformation, is responsible for transforming data from one format to another.

GridMiner Base

- GMRB, GridMiner Resource Broker, receives a request with the specifications of a resource. The service responds with the Grid Service Handles (GSH) that matches the requirements.
- GMIS, GridMiner Information Service, collects and combines information about all the available grid services and provides a way to query this information.
- GMLB, GridMiner Logging and Bookkeeping, is a service that collects all information about jobs, resource reservation and allocations as well logging error messages.
- GMMS, GridMiner Mediation Service, is used to integrate data from multiple data sources into one virtual data source. A virtual data source is a uniformly organized data source of which the subsets, the individual data sources, still reside at their original site.

  There are different ways in which multiple data sources can be partitioned[5]. If the mediator has to combine several data sources it is possible that they are horizontal partitioned, vertical partitioned or both horizontal and vertical partitioned.
  - Horizontal. The different data sources use the same column formats and the difference between the data sources is that they have different sets of rows. The task of the mediator is to do a **union** on the data sources so that the data sources are virtually put together (underneath each other).
  - Vertical. The database tables that has to be combined do not have the same attributes but they do have a common attribute. The task of the service is to **join** these tables into one virtual table(data source).

The GridMiner mediation service is based on the Grid Data Mediation Service (GDMS), which itself is based on the Grid Data Services: OGSA-DAI[6].

- GMCMS, GridMiner Cube Management Service, is the service that creates OLAP cubes. A typical workflow of an OLAP process is as follows: preprocessing, cube management, OLAP mining service, presentation. In the process cube management creates the cubes and the OLAP mining service is used to perform data mining on these cubes.

The **Grid Core** is implemented by middleware, in the GridMiner case by the service oriented Globus Toolkit. This means that these services are not a part of the research of the GridMiner project.

- Replica Management is used for the management, creation and deletion, of file copies (replicas). Replicas are created for several reasons such as better performance on the new storage location and better availability on the new location for a specific site. Replica selection is responsible for finding the replica that is best used in a specific case, e.g. find the closest replica available.
- Security, as said before, is very important and is largely implemented by the grid middleware.
- Grid core services implemented by OGSA.
- File and database access service. The access service is concerned with accessing data sources, which can be relational databases, XML databases, raw files or other formats, and providing meta-data and mechanisms for querying those data sources. Meta-data is information about the actual data such as the query language that has to be used, the physical location of the data, states, workload, etc. This access service is implemented by reusing the implementation of the OGSA-DAI services [6]. A wrapper is used to query the data sources [5].

The **Fabric** layer is responsible for local, resource specific operations.

- Grid resources, such as distributed hardware and software.
- Data sources, such as databases or data files.

Web layer

The Web Layer is the layer between the Graphical User Interface (GUI) and the grid, it consists of the following:

- Knowledge Base (KB) is used to store and share all information needed by the other components in the process of knowledge discovery, it is made up of the following components:

  - Ontologies: describing data mining domain, data sources and activities. The ontologies are written in OWL[3]. Owl is a web ontology language and is written specifically for the world wide web.
  - Metadata: information about the data.
  - Rules: discovered rules of the data mining process.
  - Facts: explicit knowledge generated from the discovered rules.
  - Central registry: information about services and their locations and informations about users and projects

- Service configuration is a set of web applications that makes it possible for the users to interact with the GridMiner application. It consists of the following procedures:
  - It is used for configuring services such as selecting the data mining application.
  - Setting up of the input parameters.
  - Prepare the workflow parameters in a DSCL document.

- The Dynamic Service Control Engine Client is the bridge between the grid and the Web environments. The DSCE client is used for the following:
  - To start up the engine, DSCE.
  - Control the execution of the process.
  - Sending messages from the services to the client, such as intermediate results.

- Data exploration and visualization are the other processes that are part of the web layer. These are responsible for the throughput from the grid to the GUI for the data.

Graphical User Interface

The GUI acts as the visualization layer. Allowing the user to interact with the system in the following way:

- Interactively construct workflow descriptions at a high abstraction level.
- Visualizing data mining results.
- The GUI is based on Java Web Start, which is platform independent and light weighted. There is no real work done here, everything is delegated to the other layers.

[3] www.w3.org/2004/OWL/

Design characteristics

- GridMiner resource specification language (GM-RSL) is the language to describe all the different types of resources. The first prototype of the GMRSL is based on hard wired XML for simplicity and to deliver a quick prototype of the GridMiner application.
- The GridMiner data mining service (GMDMS) is a container that delegates the actions to an implementation of the function. This way it is easy to extend the functionality of the application. The input to this container is an XML configuration file. The output can be any desirable format, even multiple output formats is possible. But it is desirable to use one standard format like PMML.

Prototype

At the time of writing the developers of the GridMiner application developed a prototype of the program with the following characteristics:

- It uses the Globus toolkit 3.0.
- Completely OGSA 1.0 compliant
- Written in Java.
- The messages that are send are SOAP over http.
- API's are available that can be used by data mining techniques for common operations.
- The framework can be deployed in a Jakarta Tomcat servlet with predeployed Globus Toolkit 3.0 or in a Globus Toolkit 3.0 standalone container.

4.2 WekaG

WekaG [15] is an application that performs data mining tasks on a grid, it extends the open source data mining toolkit Weka [9]. WekaG implements a vertical architecture called Data mining Grid Architecture (DMGA), which is based on the data mining phases: preprocessing, data mining and post-processing. The application implements a client/server architecture. The server side is responsible for a set of grid services that implement the different data mining algorithms and data mining phases. The client side interacts with the server and provides a user interface which is integrated in the Weka interface (without modifying Weka itself). WekaG is implemented to include at least the following features: coupling data sources, authorization access to resources, discovery based on metadata, planning and scheduling tasks and identifying the available and appropriate resources.

With the first prototype only the Apriori algorithm is implemented to be used on the grid, but unfortunately there are no performance or efficiency issues and results available. The prototype of the application uses the Globus Toolkit 3 as the grid middleware, GridFTP to transfer data to the server nodes, the Open Grid Service Infrastructure (OGSI) which is the predecessor of WSRF which is included in GT4. As future work the application will be extended by all the data mining algorithms included in Weka, also implementation of grid services to provide more data mining services and at last the performance will be evaluated [15].

4.3 Weka4WS

Weka4WS [16] is an application that extends Weka to perform data mining tasks on WSRF enabled grids. The first prototype of Weka4WS has been developed using the Java WSRF library provided by GT4. The goal of Weka4WS is to support remote execution of data mining algorithms in such a way that distributed data mining tasks can be concurrently executed on decentralized nodes on the grid, exploiting data distribution and improving performance. Each tasks is managed by a single thread and therefor a user can start multiple tasks in parallel, taking full advantage of the grid environment. There are three types of nodes in Weke4WS: local nodes (or user nodes) with Weka4WS client software, computing nodes that provide the Weka4WS Web Services and storage nodes which provide access to data to be mined. Data can be located on the local, the computing and the storage nodes as well as on third party nodes. If the data that is to be mined is not on a computational node it can be uploaded using the GT4 data management services (GridFTP). The other steps in the process, data-preprocessing and visualization, are still executed locally.

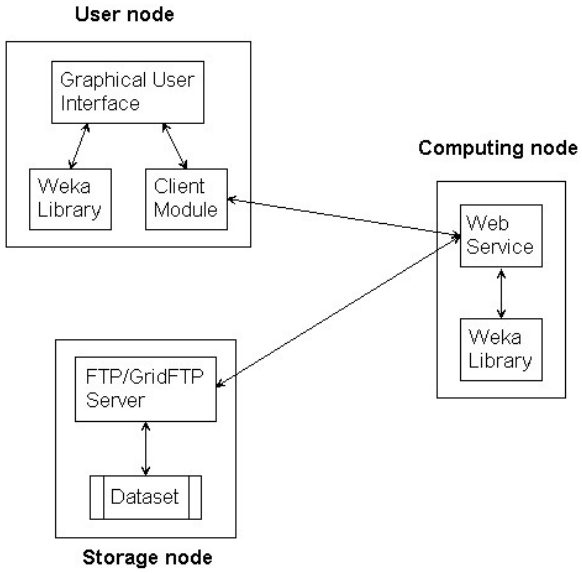


Figure 12: Weka4WS architecture

The Graphical User Interface of Weka is extended with a method to do remote data mining tasks. If a user wants to perform local data mining tasks it can click the start button on the local pane, and the task is done as usual using the local Weka Library. If the user wants to execute remote data mining tasks it can choose the available remote Web services and use the remote start button to begin the process. The Client Module of the user node acts as an intermediary between the GUI of the user and the Web Services of the computing node. The

results of both the local and the remote data mining tasks are visualized in the standard output pane. The computing nodes use the WSRF web service to expose all the data mining algorithms on that node provided by the Weka Library. All data mining algorithms provided by the Weka toolkit are exposed as Web Service and can be easily deployed.

Execution Steps

This section describes the steps Weka4WS performs to execute a data mining task on the grid [18]. In this example we assume that the data to be mined resides on the user node and not on the computing node.

The first step is the creation of a resource. The client module invokes a createResource operation, so it sends a message to the web service of the computing node. This resource is used to maintain the state (as property of the resource) of the subsequent clustering analysis. In the end the web service returns an endpoint reference back to the client.

The next step is a notification subscription. Whenever the model changes or has been computed the web service will send a notification to the client module with the value of the model resource property, which represents the result of the data mining task.

The third step is the task submission step. In this step the client module invokes the data mining operation with its arguments, among which are the url of the data set, the name of the algorithm to be used and the arguments of the algorithm.

Next is the file transfer step, if the dataset is already located on the computing node this step is skipped. The file transfer in managed by the GT4 Reliable Field Transfer (RFT) which resides on the computing node. The transfer is done by the GridFTP server running both on the user and on the computing node.

Next is the most important step, it is the data mining step. The data mining task is started by the web service which invokes the appropriate data mining algorithm that is located in the Weka library on the computing node. The result of the computation is stored in the model property of the resource which was created in step 1.

The sixth step is the notification of the result. Whenever the model property has been changed the new value is notified to the client module. This allows for asynchronous delivery of the results.

The final step is the destruction of the resource. The client module invokes the destroy operation which deletes the resource that was created in the first step.

Performance analysis

This performance analysis was carried out by the developers of Weka4WS [18]. They evaluated the execution times of the different steps of the process. The main goal was to analyze the overhead of the WSRF mechanisms: resource

creation, notification subscription, task submission, result notification and resource destruction. The other steps in the process are the file transfer and data mining step, which are the ones that are likely to be the largest steps. The data sets used range from 2 to 10 MB, they performed a classification with the J48 algorithm and a 10 fold cross-validation. The task was executed in two network scenarios. One is the local area grid with a high bandwidth and a low round-trip time (RTT is the time it takes for a package to travel from one computer to another computer over a network, and back), the other is a wide area grid with a lower bandwidth and a higher round-trip time. The results of the analysis:

- Local area grid
  - Small data size: Most of the execution time is consumed by the data mining task itself. About 9% percent is taken up by the WSRF mechanisms; the tranfer time is negligible.
  - Large data size: Almost all of the time is consumed by the data mining tasks. Less then half a percent is taken up by the WSRF mechanisms and the file transfer task.
- Wide area grid
  - Small data size: The data mining task and the file transfer both take up around 47%. The WSRF mechanisms take up about 6% of the total time.
  - Large data size: The data mining task again consumes most of the time, around 83%. File transfer time is around 16% and the WSRF mechanisms take up almost nothing.

The conclusion of the analysis is that the WSRF mechanisms with large data sets take up an almost negligible amount of time and with smaller data sets just a small percentage of time. Small data sets, smaller then around 8MB, should be computed on the node on which this data set resides. Otherwise a lot of overhead is generated by the transfer of the files. Our conclusion is that the grid must only be used if the data mining task takes up most of the time, or if the dataset resides on another node then the user node. The major reasons to execute data mining tasks on the grid are the increasingly large data sets, the faster networks and the distribution of data around the world. For future work we would like to test this program (and the other ones) on much larger datasets, on a network like the www because we are looking at grids in the future with nodes all around the world, and not only at one location.

4.4 DataMiningGrid

The DataMiningGrid[4] system is developed for generic and sector independent data mining interfaces and tools to be exploited on the grid. The DataMiningGrid consortium comprises of five members: Fraunhofer Institute, Daimler Crysler, University of Ulster, University of Ljubljana and the Technion Institute of Technology. The project composed the following requirements: handling of massive and distributed data, distributed operations, data privacy and security, user friendliness and resource identification and metadata. Following these requirements the main objectives of the project are the development of grid interfaces that could be used by data mining tools, a user friendly workflow editor for configuration, text mining and ontology learning services, a testbed with some demonstrator applications and the last objective is to develop all of this with emerging grid standards.

At the moment of writing the software is in the beta testing stage and it should be possible to test the DataMiningGrid software as an external user. There are two ways of doing this. One is joining the testbed and the other is an end user installation. To join the DataMiningGrid testbed the user will need a machine, preferably linux, but it can also be a windows machine on which core GT4 services will have to be installed. For that a fully qualified DN (distinguished name) is also needed. Depending on the experience of the user the set up procedure to join the test bed can take anything from a day to a week. To join the test bed the DataMiningGrid group also needs an official letter from the organization of the user. Other questions concerning the testbed are if the organizations network is behind a firewall and if the user is able to open a range of ports. The other option to test the software is by end-user installation. In order to actually use the DataMiningGrid system it is not necessary to join the test bed. The end-users installation takes about 30 minutes all together (first the user has to obtain a DataMiningGrid certificate). The DataMiningGrid would prefer the user to through both steps, testbed and end user installation, and provide them with feedback, especially comments and feedback on the installation and other instructions.

Joining the testbed was not possible because some files were missing and we started to get the end user installation working first. Here is a short description on how to get the client side of the DataMiningGrid working on a windows machine. First install **Java**[5] if it was not installed yet and set Java variables correct:

set JAVA_HOME=C:\java
set PATH=C:\java\bin

[4] www.datamininggrid.org
[5] java.sun.com

Next install **Ant**[6] and set the Ant variables as below; also add the line *C:\ant \bin* to the system variables → Path.

set ANT_HOME=C:\ant

You need to install **Triana**[7] and correctly set the path. Triana is a graphical environment that the DataMiningGroup uses for its interaction with the user. It is used as a workflow composer, which can build, execute and manage large and complex data mining processes in a grid environment. Here is how to correctly set the path if Triana is installed on C:

set TRIANA=C:\triana

After these actions unzip the DataMiningGrid zip file in the home directory of the Triana application. A certificate is needed to get access to the Kanin server of the Fraunhofer Institute. How to get a certificate is stated on the website of the DataMiningGrid. This certificate has to be put in a .globus directory, this directory has to be created by the user. And a certificate from the certificate authority (CA) has to be put in the subdirectory certificates of the .globus folder. To run the application go to the directory of triana/bin and run triana.bat. After the startup of Triana there will be a folder DataMiningGrid on the left of the screen. To start up do a CredentialsGenerator from DataMiningGrid → security and then run the ApplicationExplorer. However, after a lot of hard work we did not get it working because of firewall restrictions both on our side and on the Fraunhofer institute.

[6] ant.apache.org
[7] www.trianacode.org

4.5 GridWeka 2

Another grid enabled data mining application is GridWeka2. The difference between the original Weka and GridWeka2 is the option to run cross validation in parallel and distributed over several machines. Here is the way to run GridWeka2 on the DAS-2 [8].

1. To work on the DAS-2 it is sometimes convenient to first reserve some nodes before using them. This can be done with the next command when you are logged on to one of the DAS-2 fileservers:

   preserve -e node100.das2.liacs.nl:7 900

   This will reserve seven nodes on the DAS-2 for 900 second, only available nodes will be selected. It is not necessary to do a reservation, but when the DAS-2 is very busy it is more convenient.

2. The next step is the startup of the server(s). If you have reserved some nodes these nodes can be used to be running a server. On the DAS-2 the reserved nodes can be checked with the command *preserve -llist*. The startup of the servers can be done with the next command:

   rsh node108.das2.liacs.nl java -classpath /home/username/gridweka2/ GridWeka2.jar weka.ucd.WekaServer 6700 3

   This will start one server on the node 108 on port 6700. This server will be able to process three concurrent requests at the same time.

3. Before starting GridWeka2 it is necessary to tell GridWeka2 where the servers of the grid are located. This is done by creating a *servers.csv* file within the same directory as where GridWeka2 is located. This file should look like this:

   node100.das2.liacs.nl,6700,-,-

   node101.das2.liacs.nl,6700,-,-

   node102.das2.liacs.nl,6700,-,-

   In this case there are 3 servers (node100.das2.liacs.nl, node101.das2.liacs.nl, node102.das2.liacs.nl) each with the same port number (6700). The last two options are empty, but will be used in the future.

4. To start the client of GridWeka2 type the following line in your command prompt(windows) from the directory where GridWeka2.jar is located:

   java -cp GridWeka2.jar weka.gui.GUIChooser

   This will open the Weka GUI chooser, which is the same as the original Weka GUI chooser. GridWeka2 does not work in command line interface (CLI) but it works in both the Explorer as well as the Experimenter mode. Furthermore in these two modes it works in the same way as the original Weka for as long as the user is concerned. GridWeka2 uses the servers.csv

[8] http://www.cs.vu.nl/das2/

Datasets	Algorithms	Computers	Time
zoo, lymph	J48, KNN(1)	3 nodes	4:01
zoo, lymph	J48, KNN(1)	2 nodes	3:36
zoo, lymph	J48, KNN(1)	1 node	3:49
zoo, lymph	J48, KNN(1)	laptop	0:09
zoo	J48	3 nodes	1:15
zoo	J48	2 nodes	0:50
zoo	J48	1 node	0:49
zoo	J48	laptop	0:03
Waveform-5000	J48	3 nodes	2:58
Waveform-5000	J48	2 nodes	2:43
Waveform-5000	J48	1 node	4:34
Waveform-5000	J48	laptop	18:58

Table 1: Experiments with GridWeka2

file to connect to the available servers to perform the cross validation in parallel.

Experiments

Table 4.5 shows the experiments we have done with the GridWeka2 experimenter. All experiments were done with 10 repetitions and 10 fold cross-validation. Three data sets were used zoo, lymph and waveform, the sizes of these data sets are respectively 15 kb, 22 kb and 1 mb. The experiments were done on one laptop (800mhz, 512 mb) or on the same laptop in combination with 1, 2 or 3 nodes of the DAS-2. The DAS-2 is a Distributed ASCI Supercomputer; it consists of five cluster among which is the cluster at the university of Leiden. The cluster at the university of Leiden has 32 nodes each with 2 1-Ghz Pentium III processors, 1.5 GB RAM and a 80 GB harddisk.

The results from the experiments show us one important issue of data mining on grids. When the data set used is not big enough there will not be any improvement on execution time. On the contrary if the data set is very small it is much faster to execute the data mining on one single computer. That is because the overhead of transferring the data and the results is too big with smaller sets. But as you can see when a data set is used of 1 mb, the Waveform-5000 data set, the speedup when using 1, 2 or 3 nodes is significantly. And because it is not uncommon to have data sets larger then 1 mb the GridWeka2 can significantly improve the execution time.

We also tried the algorithms J48, KNN(1), NaiveBayes, ZeroR and Simple Linear Regression at the same time on the dataset Waveform-5000 with 1, 2 and 3 nodes but unfortunately both the DAS-2 and the notebook ran out of memory.

The adjustments made to Weka do not make GridWeka2 a grid enabled

application (cf. three point check list). Each server of GridWeka2 has to be set up manually. It is not the case that when one server is set up on the grid that the server looks for available nodes on the grid. The server only runs on one node and if more then one node wants to be used the other ones has to be set up manually as well. This is not what a grid enabled application should look like. The name, GridWeka2, suggests that it is a grid enabled application but on the contrary it is an application that can use multiple computing sources in parallel.

5 Grid enabled frequent itemsets algorithm

By the absence of a good test of a grid enabled data mining application we have decided to do some work ourselves. The idea was to rewrite a data mining algorithm to have it work on a grid. We took a depth-first implementation (the so called Apriori algorithm) from dr. W. Kosters of the Liacs (Leiden Institute of Advanced Computer Science) and used MPI (Message Passing Interface) to make it grid-enabled.

5.1 Apriori Algorithm

The Apriori algorithm is an association rule algorithm that finds frequent itemsets in a data set. A set is a frequent itemset if the set is contained in at least the threshold number of transactions in the data set. The threshold is called the minimal support and should be known in advance. Thus the Apriori looks in a data set of transaction for sets of items that are often together in one transaction. Take a look at the following data set of transactions. There are transactions with 3 and 4 items.

1. a, c, d
2. b, c, d, e
3. a, b, e
4. a, c, d, e
5. a, c, d

As one can see the set {c, d, e} has a support of 2. This set is called a 3-itemset because it contains three items. The support is two because there are two transactions containing this set, namely itemset 2 and 4. The 1-itemsets {a} and {c} both have a support of 4. The minimal support is used to narrow down the outcome of the algorithm and to find only the bigger sets. In this example if the minimal support would be 3 the itemsets that pass this threshold would be: {a}, {c}, {d}, {e}, {a,c}, {c,d} and {a, c, d}.

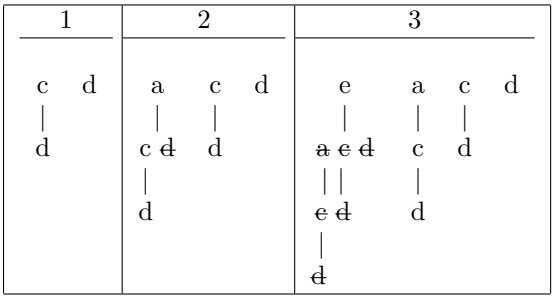
Now lets look at how the Apriori algorithm works and finds these frequent itemsets. The Apriori algorithm is based on the fact that all subsets of a frequent item set are also frequent itemsets. The algorithm generates the frequent itemsets in a breadth first way. This means that it first finds all the frequent 1-itemsets, next all the frequent 2-itemsets, all 3-itemsets, etc. Until the generated itemsets do not contain any more frequent itemsets. Apriori uses the frequent k-itemsets to generate the frequent k_{+1}itemset. The next table is the flow of the frequent itemset generation with the Apriori algorithm.

1-itemsets	support	frequent 1-itemsets	generated 2-itemsets	support
a	4	a	a, c	3
b	2		a, d	3
c	4	c	a, e	2
d	4	d	c, d	4
e	3	e	c, e	2
			d, e	2

frequent 2-itemsets	generated 3-itemsets	support	frequent 3-itemsets
a, c c, d	a, c, d	3	a, c, d

5.2 Depth-first Apriori

The Depth-First (DF) Apriori [21] algorithm is a form of the Apriori algorithm but instead of building the itemsets breadth first is does it depth first. Which means that it does not generate layer by layer. The trees below show the flow of the depth-first Apriori algorithm.



The algorithm works as follows. First find all the 1-itemsets in the data set. Remove the itemsets with a support lower then the minimal support and arrange the items by increasing frequency. In our example this means: e, a, c, d. We start with the last item and work backwards, say we have itemsets i_1 .. i_n. We start building the tree by putting i_n as a node of the tree, at this point this is the only node of the three. The next step is putting i_{n-1} next to the top node and copy the old tree beneath this new node. For our example this means we put the c next to the d and put the old tree, which consist of only d, beneath the c. After this step we count the support of each new itemset. In our case this means the itemset {c, d}, which is frequent and therefor stays. If the itemset {c, d} was not frequent we would have pruned it by removing the d beneath the c. This will lead us to the first tree and we begin again by putting i_{n-2} next to the top nodes and the old tree beneath it. Until all the 1-itemsets have been placed in the tree.

5.3 DF-Apriori using MPI

MPI stands for Message Passing Interface and is used for parallel computing. We intend to use MPI to make the DF-Apriori algorithm a parallel algorithm so that is can be used on a grid. The algorithm uses both domain decomposition and functional decomposition. In domain decomposition data are divided into smaller sets of data and are mapped on multiple processors. In the case of data mining algorithms this means to divide the dataset into smaller datasets and perform data mining processes on each of them in parallel. Functional decomposition is used to allocate tasks to slave processes and managed by a master process. In data mining this can be beneficial because it is not useful to perform data mining only on the smaller data sets, we want to combine the results of the smaller sets into a final result. Thus in the new algorithm we use a mix of domain and functional decomposition which are both implemented using MPI. MPI consists of a library of functions that you can include in your code to communicate between processes. The communication between the different processes is done by message passing which in turn is done by library calls. Four classes of calls can be distinguished. The first class of calls is used for initialization, starting communication, identifying the number of processors, creating subgroups of processors and managing the communication. The second is used for point-to-point communication, this is the communication between two processors. Communication consists of a **send** and a **receive**, without one of them the communication will fail. The third library class provides communication and synchronization among groups of classes. And the last class is used for the creation of complicated data structures.

We used MPI to make the DF-Apriori algorithm grid enabled. As said before domain and functional decomposition is implemented to accomplish this. We will now describe the algorithm in more detail and describe the techniques used to make it grid enabled. The algorithm starts with initializing basic MPI commands.

```
MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
MPI_Comm_size (MPI_COMM_WORLD, &ps);
```

MPI_Init is the initial mpi command, rank gives every processor a rank number and size determines the number of processors the in MPI_COMM_WORLD group.

The algorithm starts of with four steps of preprocessing. The first pass it computes the minimal and maximum item numbers. Later in the algorithm an array is used to keep track of the frequency of all the 1-itemsets. Say that the items in the dataset start from 10000, it is more efficient to have the array start from that point instead of having the first 10000 array elements containing nothing. This function reads the entire dataset and computes the minimum and maximum item numbers. MPI is used to have the computation done in parallel.

Every processor computes the minimum and maximum item number of a small part of the dataset. After it has computed this it will send its result back to the master process, which is realized with the following command:

MPI_Reduce (&min_itemnr, &mini, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

The reduce command is used to perform a global operation on all processors of a group. Here the reduce actions will cause all the group members to send their minimal item number to the master process (0). The master process will reduce all the numbers to one with the operator that is given, in this case it is MPI_MIN which computes the minimum. The following action is to send these numbers back to all the processors which is accomplished by the following command:

MPI_Bcast (&mini, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);

It broadcasts the minimal item number from the master process to all the other members of the group. The same is done with the maximum item number.

In the second pass of the dataset the number of frequent 1-items is counted. Each processor computes the frequency of a part of the dataset. For example, assume we have three processors and each of them has computed the following frequency of their part of the dataset.

Proc / Item	1	2	3	4	5	6	7	8	9	10
1	1	0	1	3	7	0	1	6	6	4
2	2	0	0	1	2	10	7	6	0	3
3	3	0	0	1	3	1	2	0	1	7

After the processors have computed their data it is send to all other processors, each processor receives all other data and adds it to their own. Which ends up in the situation where each processor has the following data:

Item:	1	2	3	4	5	6	7	8	9	10
Frequency:	6	0	1	5	12	11	10	12	7	14

Now the frequent 1-items are computed with a minimal support of 6. This is also done in parallel, followed by a reduce and a bcast. So in the end each processor ends up with the number of frequent items which is 7.

The third pass is used to compute the number of relevant transactions which are the ones that contain at least two frequent 1-itemsets. Again this is done in parallel. Each processor computes the number of relevant transactions of a part of the dataset. A reduce adds these computed numbers and a bcast sends them back to each of them.

Next is the sorting step, which sorts the 1-items with respect to the support and renumbers them. The sorting step is used for efficiency purposes, the fastest execution time is achieved when the items are sorted with increasing frequency [21]. The frequency list that was computed previously is split among the pro-

cessors. Each processor starts with sorting its list. At this point each processor has a sorted list and these ones should be combined and sorted again. We have solved this issue as follows:

```
while number of remaining processors (n) > 1 do
    if number of remaining processors = EVEN then
        ReceivingProcessor = 0
        for i = n to n/2 do
            Processor i sends to ReceivingProcessor
            ReceivingProcessor++
        end for
    end if
    if number of remaining processors = ODD then
        ReceivingProcessor= 0
        for i = n to n/2 do
            Processor i sends to ReceivingProcessor
            ReceivingProcessor++
        end for
        Processor n/2 sends to Processor n/2 -1
    end if
    Each processor that receives a list combines it with its own list and sorts
    it directly using mergesort
    Number of remaining processors n is n/2
end while
```

Following this the master processor has the sorted list and broadcasts it to all the other processors.

The next step of the algorithm is to build the FP tree. This FP tree is later used to count the frequent items. The whole file is read again in this phase. Each processor, except for the master processor, reads a part of the dataset and computes the number of frequent 1-itemsets in each transaction. If a transaction contains two or more frequent items then this transaction is send to the master processor. The master processor adds these transactions to the FP tree. This sounds easier then it is, because the master processor has to know whether there are remaining transactions or not (all transactions are already added to the tree). So we implemented this as follows: all processors send a finished message to a second master processor, as soon as this second master processor has received all finish messages it will send a finish message to the primary master processor. We used MPI_Probe for the master processor to check whether the incoming message is a finished messaged or a new transaction message.

MPI_Iprobe(MPI_ANY_SOURCE, 1 or 2, MPI_COMM_WORLD, &flag, &status);

If it is neither of them, which means that there is no message, then it will loop until there is a message. If the message is a finished message the master

processor will stop with the receiving loop and will continue with the next step of the algorithm. And if the message is a receive message it will add the transaction to the FP tree and send a continue message back to the sender to communicate that it continue with the next transaction. A sending processor will not continue its loop through the transactions until it receives this continue message. This mechanism makes sure that the processors that send the transactions to the master processor will not send too much. Which, on the processor side, can cause a memory problem. And this in turn can result in loss of data.

The last step of the algorithm is the counting step. This step builds the Depth-First tree and counts the frequent itemsets. The building is done using the ordered 1-itemsets. This step of the algorithm works as described as before: adding an itemset, constructing the tree, counting the frequencies, pruning the tree and copying the tree. We did not parallelize this step of the algorithm.

5.4 Experiments

We implemented the new algorithm on the DAS-2 using MPI. The source code is included in Appendix A. Compiling the code is done with the following command:

mpiCC -Wall -O3 -o fim_all current.cc

MpiCC is the MPI compiler for c code, fim_all is the target application and current.cc is the sourcecode. To start the application on the DAS-2 a script must be submitted to the DAS-2. Such a script looks as follows:

```
#!/bin/sh
#$ -S /bin/sh
#$ -cwd
#$ -N fim_all
#$ -l h_rt=11:55:00
#$ -pe mpi 15

PROG=fim_all
ARGS=""

/usr/local/mpich/mpich-gm-gcc/bin/mpirun \
    -np $NSLOTS -machinefile $TMPDIR/machines $PROG $1 $2 $3
```

This script states the number of processors to be used, in this example 15. It also states the program's name, which mpi to use and the parameters. The parameters are the programs name and three other arguments which are stated in the command to submit the script to the DAS-2. The next is example of such a submit:

qsub start.sge retail 12 uit

Qsub is the submit command and start.sge is the starting script as shown before. Input of the program are the dataset, minimal support and output filename, in the example these are respectively the retail dataset, 12 and uit.

Next are the tables with data of the experiment. There are five columns. Processor is the number of processors used, Or means original. This is the original algorithm and it is also submitted to the grid but is uses only one processor. Minsup is the minimal support. Reading is the time used for the first phase of the algorithm. FP is the time to build the FP-tree. And counting is the time it takes to build the Depth-first tree and to count the frequent items in this tree. In some cases there there is no outcome, this is stated in the tables with an X. There are also custom dataset used for the experiments, these are generated with a java application. For example Custom 12000-1000 is a custom dataset with 12000 attributes and 1000 transactions.

Processor	Minsup	Reading	FP	Counting
3	12	2	12	20
4	12	1	11	21
5	12	1	11	21
6	12	1	12	20
10	12	1	12	21
15	12	2	13	21
Or	12	2	8	19

Table 2: Retail dataset, approximately 4 Mb.

Processor	Minsup	Reading	FP	Counting
Or	12	12	7	X
2	12	13	24	X
3	12	13	12	X
4	12	13	9	X
5	12	13	8	X
10	12	12	6	X
15	12	13	6	X

Table 3: Accidents dataset, approximately 34 Mb.

Processor	Minsup	Reading	FP	Counting
Or	2	11	6	X
5	2	10	4	X
10	2	10	3	X
15	2	12	4	X
Or	4	12	5	X
5	4	10	4	X
10	4	10	3	X
15	4	12	3	X
Or	977	9	4	19
5	977	11	2	19
10	977	10	3	19
15	977	12	4	18

Table 4: Custom dataset with 12000 attributes and 1000 transaction, approximately 300 Mb.

Processor	Minsup	Reading	FP	Counting
Or	4	28	14	X
3	4	24	12	X
10	4	28	8	X
15	4	26	7	X
Or	2	26	15	X
10	2	25	8	X
15	2	25	7	X
Or	1500	25	12	X
5	1500	26	8	X
10	1500	25	7	X
15	1500	29	8	X
Or	2450	24	9	2
5	2450	25	7	2
10	2450	25	7	1
15	2450	25	7	2

Table 5: Custom dataset with 12000 attributes and 2500 transaction, approximately 750 Mb.

The results of the experiments with the retail dataset show us that using a grid for data mining is not useful when a relatively small dataset is used. The time it takes to read the dataset using only one node or using 15 nodes is equal. Computing the FP-tree takes 50% more time when doing this on more then one node. These negative results can be caused by a bad algorithm or by the extra time it takes to send the data and the intermediate results over the grid.

Processor	Minsup	Reading	FP	Counting
Or	4	20	8	X
5	4	20	7	X
10	4	20	6	X
15	4	20	6	X
Or	500	21	9	X
5	500	21	6	X
10	500	20	5	X
15	500	20	5	X
Or	990	20	6	X
5	990	20	7	X
10	990	19	7	X
15	990	19	7	X

Table 6: Custom dataset with 22000 attributes and 1000 transaction, approximately 625 Mb.

Processor	Minsup	Reading	FP	Counting
Or	12	82	13	X
5	12	24	X	X
Or	100	58	13	X
5	100	28	X	X
10	100	23	X	X
15	100	X	X	X
Or	400	25	9	X
5	400	24	7	X
10	400	24	6	X
15	400	X	X	X
Or	497	24	8	0
5	497	24	7	0
10	497	23	7	0
15	497	X	X	X

Table 7: Custom dataset with 50000 attributes and 500 transaction, approximately 715 Mb.

When looking at the experiments done on the accidents dataset it shows us a reading time which is equal for each number of processors used. It also shows us that the time used for building the FP-tree decreases when increasing the number of processors except for the fact again that using only one node is still almost as fast as when using 15 nodes. The reading time in the experiments with the custom datasets, 12000-1000 12000-2500 22000-1000, show no decrease when using more nodes. The FP phase shows an increase in speed of maximum

50% in case of a small minimal support and almost no increase in case of a large minimal support. For the biggest dataset, custom 50000-500, it is shown that the reading time for small minimal supports could be decreased with a maximum of 3.4 times. This increase in speed opposed to the other experiments with almost no increase is caused by the following. This dataset has a large amount of attributes, which means that there is more work to be done because there are more different frequent items. The increase in speed is decreasing when the minimal support increases. As before this can be explained by the fact that if the number of frequent items decreases the amount of work and the speedup also decreases. It is a pity that we can not use bigger datasets on the Das-2 then approximately 750 Mb. Some experiments were partially unsuccessful, in the table denoted by an X. We assume that these failures were caused by limitations of the Das-2 because even the original algorithm is not finished on some occasions. In the end of the experimenting phase using 15 nodes did not work anymore. We assume again that this is caused by the Das-2, maybe some nodes are not running anymore.

Concluding our experiments with the grid enabled Apriori algorithm, when using relatively small datasets there is no use for a grid. Grids should be used when dealing with large enough datasets, in the order of gigabytes. The algorithm can be further optimized for better performance by analyzing the time each phase of the algorithm uses. Also much of the same work is done on all sites. These flaws are probably caused by us when trying to make the algorithm work instead of trying to make it work in an optimal way. Future work can be spend on optimizing the algorithm for (much) better performance.

6 Conclusion and Future work

In this thesis we have described the work that has been done on grid enabling data mining. Also we have described our own work on grid enabling one data mining algorithm.

The GridMiner application is a well documented and structured application. It integrates all the data mining processes and it supports OLAP. Thanks to the workflow service it should be an easy to use application; a scientist does not need to know about all the technical details. GridMiner is able cope with distributed data (virtual data sources). It is based on OGSA and the Globus toolkit which are open and extensible architectures. Globus also takes care of most of the security issues. Furthermore the application can also create OLAP cubes. According to the list at the beginning of chapter 4 the GridMiner application is a very complete application. The only problem is that we can not download the program and test it ourselves.

WekaG is not as well documented as the GridMiner application. It has not yet finished being developed and therefor it could not be tested. The positive side is that the application uses open architectures such as OGSI and the Globus Toolkit. It can cope with distributed dated thanks to coupling data sources. As it is an extension of the open source Weka tool it can be further extended with data mining techniques and algorithms when needed. WekaG also implements authorization access to resources, combined with the security measurements of the Globus toolkit it forms the important security issues. Unfortunately it does not support OLAP and there is little known about the scalability of the application.

The third application we looked at was Weka4WS. We wanted to try this program on the Das-2 because it makes use of the newest Globus toolkit and all algorithms of the original Weka can also be executed on the grid. Furthermore this application is open source just as WekaG, because Weka is open source, so it can be extended when needed. Weka4WS can also cope with distributed data, it can be located on either three kinds of nodes and even on third party nodes. To run Weka4WS on a grid it has some requirement. First a list of programs need to be running on each node, among which is Globus Toolkit 4. Problem is that on the Das-2 the GT 4 is not installed yet. Another issue with the Das-2 and Weka4Ws is that on every machine there should be web services running and the Das-2 does not allow web services. This way it is impossible to test the program on the DAS-2.

The DataMiningGrid from the Fraunhofer institute began their research by describing the requirements for a grid enabled data mining application. These requirements strongly resemble the requirements stated at the beginning of chapter four. An advantage of the DataMiningGrid is the use of Triana to compose workflows. These workflows can be used to ease and to minimize the time needed for the kdd process. Unfortunately there are not many specifications of the program and as described before the application could not be tested.

GridWeka2 is the last grid enabled data mining application we looked at.

This was finally a program that we could test ourselves. As stated in chapter 4.5 this is not a grid enabled application according to the three point check from Foster [1]. Also it does not satisfy all the requirements stated at the beginning of chapter 4. The application is not based on an open architecture, the users do need to know about the details, security is not taken care of and OLAP is not supported. Ultimately it is a program that can benefit of multiple processors but it is not a grid enabled data mining application.

The FP-tree Apriori algorithm that was made grid enabled is not a grid enabled data mining application that we wished to test. But to do some real work and test a grid enabled algorithm was also a challenge. As it worked out we were partly satisfied, although it did not speed up the time to execute the algorithm that much. It probably was not the best algorithm to grid enable. Too much time was spend on minor details because the implementation of the original algorithm was really difficult to understand. It was not easy to grid enable this complicated algorithm, the reason for this besides the complicated algorithm was the way to test it on the Das-2. If there was a flaw in the implementation is was hard to find it in the source code because of the use of multiple processors, it often led to 'hanging' of the program at some seemingly random place. Looking back it would have been more efficient to first make state diagrams to make to sure there was no deadlock or other multi processor misapprehension. For future work there are steps of the algorithm that need to be grid enabled and there is plenty of room for improvement and fine tuning on the parts that were already grid enabled. More work can be done on the grid enabled applications. Some of the programs need to be tested and some can be tested more thorough full.

Appendices

A Depth-First Apriori MPI sourcecode

```
#define input_filename argv[1]
// (absolute) value of minsup:
#define macro_minsup atoi(argv[2])
// name outputfile:
#define output_filename argv[3]

#include <iostream>
#include <fstream>
#include <cstdio>
#include <climits>
#include <cstdlib>
#include <ctime>
#include "mpi++.h"
using namespace std;

const int MAXDEPTH = 100; // maximal depth of trie (for printing)

int minsup;                // minimum support
int mini,maxi,ps,my_rank;
class initialcounts
{
  public:
    initialcounts (char *inputdata);

    static int *itemsorder;
    static int *itemsorder2;

    static int *items_frequency;
    static int *items_frequency2;

    static int *ranking;
    static int min_itemnr, max_itemnr;
    static int number_transactions, number_freq_items, lines, last;

 private:
    void first_pass ( );
    void second_pass ( );
    void third_pass ( );
    void initialsort ( );
    void swap(int*, int*);
    void sort(int[], int, int,int[]);
    void mergesort(int[], int, int);

    char *infilename;
    int *init_items_frequency,*init_items_frequency2 ;
};

int *initialcounts::items_frequency = NULL;
int *initialcounts::items_frequency2 = NULL;

int *initialcounts::itemsorder = NULL;
int *initialcounts::itemsorder2 = NULL;

int *initialcounts::ranking = NULL;
int initialcounts::number_transactions = 0;
int initialcounts::lines = 0;
int initialcounts::number_freq_items = 0;
int initialcounts::min_itemnr = 0;
int initialcounts::max_itemnr = 0;
int initialcounts::last = 0;
```

```
// constructor
initialcounts::initialcounts (char *inputdata)
{
  infilename = inputdata;
  first_pass ( );
  second_pass();
  third_pass();
  initialsort();
}//initialcounts::initialcounts


// computes minimal and maximal item number that occur in the database;
// if these are known in advance, this function can be easily adapted
// function reads whole file!
void initialcounts::first_pass ( )
{
  int itemnr;
  bool first = true;
  ifstream fin (infilename);
  if ( ! fin )
    cout << "No_such_filename" << endl;
  char c;
  int pos;

 MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
 MPI_Comm_size(MPI_COMM_WORLD,&ps);

  do
   {
   do // per regel
        {
     if(lines % ps == my_rank )
      {
            fin.get (c);
       itemnr = 0;
            pos = 0;
             while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
            {
              itemnr = 10*itemnr + (int)(c) - (int)('0');
              pos++;
               fin.get (c);
             }//while
             if ( pos )
            {
        if ( first )
                  max_itemnr = min_itemnr = itemnr;
        first = false;
               if ( itemnr < min_itemnr )
                   min_itemnr = itemnr;
               else if ( itemnr > max_itemnr )
                  max_itemnr = itemnr;
             }//if
    }//if
    else
     {
      fin.get (c);
       while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
      {
        fin.get (c);
      }//while
     }//else
   } while ( c != '\n' && ! fin.eof ( ) );
     lines++;
    } while ( ! fin.eof ( ) );

  lines--;  // Because of the do/while it goes one to far.
```

```
  fin.close ( );

  MPI_Barrier(MPLCOMM_WORLD);
  MPI_Reduce(&min_itemnr, &mini, 1, MPI_INT, MPI_MIN, 0,MPLCOMM_WORLD);
  MPI_Reduce(&max_itemnr, &maxi, 1, MPI_INT, MPI_MAX, 0,MPLCOMM_WORLD);

  MPI_Barrier(MPLCOMM_WORLD);
  MPI_Bcast(&mini,1,MPI_INTEGER,0,MPLCOMM_WORLD);
  MPI_Bcast(&maxi,1,MPI_INTEGER,0,MPLCOMM_WORLD);

  MPI_Barrier(MPLCOMM_WORLD);
  min_itemnr = mini;
  max_itemnr = maxi;

  MPI_Barrier(MPLCOMM_WORLD);
  MPI_Barrier(MPLCOMM_WORLD);
}//initialcounts::first_pass


void initialcounts::second_pass ( )
{
  int k=0;
  int nfi=0;  // local number of frequent items
 MPI_Status status;  // Return status of the receive
 ifstream fin (infilename);

  if ( ! fin ) cout << "No_such_filename" << endl;
  int itemrange = max_itemnr-min_itemnr+1;  // for optimilization, if the first item is item 238734, then
                                            // you start with that instead of an array from 0.
 init_items_frequency2 = new int[itemrange]; // Array with all the items and their frequencies, array[67]=
                                             // 12 means, item 67(or minimum item number + 67) appears
                                             // 12 times in the dataset
  init_items_frequency = new int[itemrange];

  for ( k = 0; k < itemrange; k++ )
  {
   init_items_frequency2[k] = 0;
   init_items_frequency[k] = 0;
  }
 char c;
 int item, pos;
  k = 0;
  do
 {
   do
    {
     if(k % ps == my_rank ){    // each processor computes: #lines/#processors transactions.
        fin.get (c);
        item = 0;
       pos = 0;
        while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
       {
         item = 10*item + (int)(c) - (int)('0');         // Ascii renumbering
          pos++;
          fin.get (c);
        }//while
        if ( pos )
        {
         init_items_frequency[item-min_itemnr]++;
        }
     }//if
     else
      {
       fin.get (c);
        while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
       {
         fin.get (c);
        }//while
```

```
      }
    } while ( c != '\n' && ! fin.eof ( ) );
    k++;
  } while ( ! fin.eof ( ) );


  fin.close ( );
  MPI_Barrier(MPI_COMM_WORLD);

  int buff[(itemrange*sizeof(int) + MPI_BSEND_OVERHEAD)*ps];
  MPI_Buffer_attach(buff, (itemrange*sizeof(int) + MPI_BSEND_OVERHEAD)*ps);

        // Send the frequency array to all the other processors, not yourself
        for(int i = 0; i<ps;i++){
   if ( my_rank !=i ){
     MPI_Bsend(init_items_frequency, itemrange, MPI_INT,i,0,MPI_COMM_WORLD);
   }
    MPI_Barrier(MPI_COMM_WORLD);
  }

  MPI_Barrier(MPI_COMM_WORLD); //Wait, otherwise you can erase your own computated data.

  // Receive all the frequencies from the other processors and add them to your own frequency.
  for (int source = 0;source<ps;source++){
    if ( my_rank !=source ){
      MPI_Recv(init_items_frequency2, itemrange, MPI_INT,source,0,MPI_COMM_WORLD,&status);
      for(int i = 0; i<itemrange;i++){
        init_items_frequency[i] = init_items_frequency2[i] + init_items_frequency[i];
     }
    }
  }

  MPI_Barrier(MPI_COMM_WORLD);

  int size = (itemrange*sizeof(int) + MPI_BSEND_OVERHEAD)*ps;
  MPI_Buffer_detach(buff, &size);

  for ( k = 0; k < itemrange; k++ )
  {
   if(k % ps == my_rank )       // Split the counting in ps pieces
    {
     if ( init_items_frequency[k] >= minsup )
     {
      nfi++;  // Number of frequent items at one processor
     }
   }
  }
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Reduce(&nfi, &number_freq_items, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD); // Sum the nfi from all
                                                                                // the processors

  MPI_Bcast(&number_freq_items,1,MPI_INTEGER,0,MPI_COMM_WORLD); // Broadcast the number_freq_items

  MPI_Barrier(MPI_COMM_WORLD);

  if(0  == my_rank )printf ("Number of frequent items: %d\n", number_freq_items);
}//initialcounts::second_pass

void initialcounts::third_pass ( )
{
  number_transactions = 0;
  int nt = 0;  // a local value of number of transactions
  ifstream fin (infilename);
  if ( ! fin )
    cout << "No such filename" << endl;
  char c;
  int item, pos, items_in_trans,l=0; //l for line
```

```
  do
  {

    items_in_trans = 0;
    do
    {
    if(l % ps == my_rank ){    // each processor computes: #lines/#processors transactions.
      fin.get (c);
      item = 0;
     pos = 0;
      while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
     {
       item = 10*item + (int)(c) - (int)('0');
       pos++;
        fin.get (c);
      }//while
      if ( pos && init_items_frequency[item-min_itemnr] >= minsup )
       items_in_trans++;
   }
   else
    {
    fin.get (c);
      while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
     {
       fin.get (c);
      }//while
    }
 } while ( c != '\n' && ! fin.eof ( ) );

 l++;

  if ( items_in_trans >= 2 )
      nt++;

  } while ( ! fin.eof ( ) );

  MPI_Barrier(MPLCOMM_WORLD);
  MPI_Reduce(&nt, &number_transactions, 1, MPI_INT, MPI_SUM, 0,MPLCOMM_WORLD);
 MPI_Barrier(MPLCOMM_WORLD);
  MPI_Bcast(&number_transactions,1,MPI_INT,0,MPLCOMM_WORLD);
 MPI_Barrier(MPLCOMM_WORLD);


if(0  == my_rank )  printf ("Number of relevant transactions: %d\n", number_transactions);
if(0  == my_rank )  printf ("Number of lines: %d\n", lines);
  fin.close ( );
}//initialcounts::third_pass

void initialcounts::swap(int *a, int *b)
{
  int t=*a; *a=*b; *b=t;
}
void initialcounts::sort(int arr[], int beg, int end, int order[])
{
  if (end > beg + 1)
  {
    int piv = arr[beg], l = beg + 1, r = end;
    while (l < r){
      if (arr[l] <= piv)
        l++;
      else
      {
        r--;
        initialcounts::swap(&arr[l], &arr[r]);
        initialcounts::swap(&order[l], &order[r]);
      }
    }
    l--;
```

```
    initialcounts::swap(&arr[l], &arr[beg]);
    initialcounts::swap(&order[l], &order[beg]);

    initialcounts::sort(arr, beg, l, order);
    initialcounts::sort(arr, r, end, order);
  }
}

// sort items with respect to support - and renumber
void initialcounts::initialsort ( )
{
  MPI_Status status;  // Return status of the receive

  int itemrange = max_itemnr-min_itemnr+1;
  ranking = new int[itemrange];

  for (int k = 0; k < itemrange; k++)
  {
    ranking[k] = -1;
  }

  itemsorder = new int[number_freq_items];
  itemsorder2 = new int[number_freq_items];
  int itemsorderTemp[number_freq_items];

  items_frequency = new int[number_freq_items];
  items_frequency2 = new int[number_freq_items];
  int items_frequencyTemp[number_freq_items];

  for (int r = 0; r<number_freq_items;r++)
  {
    items_frequency[r] = -1;
    items_frequency2[r] = -1;
    items_frequencyTemp[r] = -1;

    itemsorder[r] = -1;
    itemsorderTemp[r] = -1;
    itemsorder2[r] = -1;
  }


  int start = 0, end = 0;    // Used for the start and end of the transactions one processor gets to do.
  int last = -1;             // The item after the last item.

  // Each processor takes a part of the array: init_items_frequency
  start = itemrange/ps * my_rank;  // Number of the transaction where to start for this processor
  end = (itemrange/ps * (my_rank+1)) -1;  // Which is the last transaction for this processor
  if(my_rank == ps-1) end = itemrange-1;  // For the last processor to do the last few that were not
                                          // taking before because of rounding down.
  int i =0;

  // Put the frequent items in a items_frequency array, and keep the position with the itemsorder2 array
  MPI_Barrier(MPLCOMM_WORLD);

  for (int x=start; x<=end; x++)
  {
    if(init_items_frequency[x]>= minsup)
    {
      items_frequency2[last+1]=init_items_frequency[x];
      itemsorder2[i]=x + min_itemnr;  // + min_itemnr is a patch, dont knwo precisely why.
      i++;
      last++;
    }
  }

  // Sort the items_frequency, sorting for the first time
  initialcounts::sort(items_frequency2, 0, last+1,itemsorder2);
```

```
int psa = ps; // Processors still available

// Send the items_frequencies2  and itemsorder2 and receive them in items_frequencies and itemsorder
while (psa >1)
{
  int x,y,source;
  // -- SENDING --EVEN -- //
  if (psa %2 ==0)
  {
    for ( x=0; x < psa/2; x++)
    {
      if (psa-x-1==my_rank)
      {
        MPI_Send(items_frequency2, number_freq_items, MPI_INT,x,psa,MPI_COMM_WORLD);
        MPI_Send(itemsorder2, number_freq_items, MPI_INT, x, psa+10, MPI_COMM_WORLD); // Send to x with
                                                                                      // flag -psa-1
        psa=0;
      } // if
   } // for

   // -- RECEIVING -- MERGING -- SORTING -- //
   for ( source=psa; source > psa/2; source--)
   {
     if (psa-source==my_rank)
     {
       MPI_Recv(items_frequency, number_freq_items, MPI_INT,source-1,psa,MPI_COMM_WORLD,&status);
       MPI_Recv(itemsorder, number_freq_items, MPI_INT,source-1,psa+10,MPI_COMM_WORLD,&status);
       x = 0;
       y = 0;
       bool done = false;
       while (not done)
       {
         if(x<number_freq_items)
           if(y<number_freq_items)
             if(items_frequency[x]!=-1)
               if(items_frequency2[y]!=-1)
                 if (items_frequency[x] > items_frequency2[y])
                 {
                   items_frequencyTemp[x+y] = items_frequency2[y];
                   itemsorderTemp[x+y] = itemsorder2[y];
                   y++; // Till end of y
                 }
                 else
                 {
                   items_frequencyTemp[x+y] = items_frequency[x];
                   itemsorderTemp[x+y] = itemsorder[x];
                   x++; // Only till the end of x
                 }
               else
               {
                 items_frequencyTemp[x+y] = items_frequency[x];
                 itemsorderTemp[x+y] = itemsorder[x];
                 x++; // Only till the end of x
               }
             else if(items_frequency2[y]!=-1)
             {
               items_frequencyTemp[x+y] = items_frequency2[y];
               itemsorderTemp[x+y] = itemsorder2[y];
               y++; // Till end of y
             }
             else done = true;
           else if(items_frequency[x]!=-1)
           {
             items_frequencyTemp[x+y] = items_frequency[x];
             itemsorderTemp[x+y] = itemsorder[x];
             x++; // Only till the end of x
           }
```

```
            else done = true;
          else  if(y<number_freq_items)
          {
            if(items_frequency2[y]!=-1)
            {
              items_frequencyTemp[x+y] = items_frequency2[y];
              itemsorderTemp[x+y] = itemsorder2[y];
              y++; // Till end of y
            }
            else done = true;
          }
          else done = true;
        }
      }
    }
    // Put the items_frequencyTemp into the items_frequency2
    for(int x = 0; x < number_freq_items; x++)
    {
      items_frequency2[x] = items_frequencyTemp[x];
      itemsorder2[x] = itemsorderTemp[x];
    }
    psa = psa/2;  // After sending and receiving there are only half of the processors for the next round
  } // if
  else {  // ODD number of processors left

    // SEND: last to 0  last-1 -> 1 last -2 -> 2
    for ( x=0; x < (psa-1)/2; x++)  // if psa =13 then 6 processors do this round.
    {
     if (psa-x-1==my_rank)
     {
        MPI_Send(items_frequency2, number_freq_items, MPI_INT,x,psa,MPLCOMM_WORLD);
        MPI_Send(itemsorder2, number_freq_items, MPI_INT,x,psa+10,MPLCOMM_WORLD);
        psa=0;
     } // if
    } // for

    // SEND THE ODD ONE: The odd one with psa = 9 is [0,1,2,3,4,5,6,7,8]
4 and is send to 3
    if ((psa-1)/2==my_rank)
    {
      MPI_Send(items_frequency2, number_freq_items, MPI_INT, my_rank-1, psa,MPLCOMM_WORLD);
      MPI_Send(itemsorder2, number_freq_items, MPI_INT, my_rank-1, psa+10,MPLCOMM_WORLD);
      psa=0;
    }

    // -- RECEIVING -- MERGING -- SORTING -- //
    // First for the normal ones, after this for the odd one
    for (source=psa; source > (psa+1)/2; source--)
    {
     if (psa-source==my_rank)
      {
        MPI_Recv(items_frequency, number_freq_items, MPI_INT,source-1,psa,MPLCOMM_WORLD,&status);
        MPI_Recv(itemsorder, number_freq_items, MPI_INT,source-1,psa+10,MPLCOMM_WORLD,&status);
        //mergesort
       x = 0;
        y = 0;
        bool done = false;

        while (not done)
        {
         if(x<number_freq_items)
           if(y<number_freq_items)
             if(items_frequency[x]!=-1)
               if(items_frequency2[y]!=-1)
               {

                 if (items_frequency[x] > items_frequency2[y])
                 {
```

```
                items_frequencyTemp[x+y] = items_frequency2[y];
                itemsorderTemp[x+y] = itemsorder2[y];
                y++; // Till end of y
              }
              else
               {
                items_frequencyTemp[x+y] = items_frequency[x];
                 itemsorderTemp[x+y] = itemsorder[x];
                 x++; // Only till the end of x
               }
            }
            else
            {
              items_frequencyTemp[x+y] = items_frequency[x];
               itemsorderTemp[x+y] = itemsorder[x];
               x++; // Only till the end of x
             }
          else if(items_frequency2[y]!=-1)
             {
              items_frequencyTemp[x+y] = items_frequency2[y];
              itemsorderTemp[x+y] = itemsorder2[y];
              y++; // Till end of y
            }
            else done = true;
        else if(items_frequency[x]!=-1)
           {
            items_frequencyTemp[x+y] = items_frequency[x];
             itemsorderTemp[x+y] = itemsorder[x];
             x++; // Only till the end of x
           }
          else done = true;
      else   if(y<number_freq_items)
        {
          if(items_frequency2[y]!=-1)
          {
            items_frequencyTemp[x+y] = items_frequency2[y];
            itemsorderTemp[x+y] = itemsorder2[y];
            y++; // Till end of y
          }
          else done = true;
        }
         else done = true;
     }// while

     // Put the items_frequencyTemp into the items_frequency2
     for(int x = 0; x < number_freq_items; x++)
     {
      items_frequency2[x] = items_frequencyTemp[x];
      itemsorder2[x] = itemsorderTemp[x];
    }
  } // if
} // for

 // Receive the ODD one if there is an odd number of processors left
source = (psa-1)/2;
if (source-1==my_rank)
{
  MPI_Recv(items_frequency, number_freq_items, MPI_INT,source,psa,MPI_COMM_WORLD,&status);
  MPI_Recv(itemsorder, number_freq_items, MPI_INT,source,psa+10,MPI_COMM_WORLD,&status);
  //mergesort
  // -- RECEIVING -- MERGING -- SORTING -- //
  x = 0;
  y = 0;
  bool done = false;
  while (not done)
  {
    if(x<number_freq_items)
        if(y<number_freq_items)
```

```
            if(items_frequency[x]!=-1)
              if(items_frequency2[y]!=-1)
                if (items_frequency[x] > items_frequency2[y])
                 {
                  items_frequencyTemp[x+y] = items_frequency2[y];
                  itemsorderTemp[x+y] = itemsorder2[y];
                  y++; // Till end of y
                 }
                 else
                  {
                   items_frequencyTemp[x+y] = items_frequency[x];
                    itemsorderTemp[x+y] = itemsorder[x];
                    x++; // Only till the end of x
                  }
              else
              {
                items_frequencyTemp[x+y] = items_frequency[x];
                 itemsorderTemp[x+y] = itemsorder[x];
                 x++; // Only till the end of x
               }
            else if(items_frequency2[y]!=-1)
               {
                items_frequencyTemp[x+y] = items_frequency2[y];
                itemsorderTemp[x+y] = itemsorder2[y];
                y++; // Till end of y
              }
              else done = true;
          else if(items_frequency[x]!=-1)
             {
              items_frequencyTemp[x+y] = items_frequency[x];
               itemsorderTemp[x+y] = itemsorder[x];
               x++; // Only till the end of x
             }
             else done = true;
        else   if(y<number_freq_items)
          {
            if(items_frequency2[y]!=-1)
            {
              items_frequencyTemp[x+y] = items_frequency2[y];
              itemsorderTemp[x+y] = itemsorder2[y];
              y++; // Till end of y
            }
            else done = true;
          }
           else done = true;
       }
      // Put the items_frequencyTemp into the items_frequency2
       for(int x = 0; x < number_freq_items; x++)
       {
        items_frequency2[x] = items_frequencyTemp[x];
        itemsorder2[x] = itemsorderTemp[x];
       }
     }

  psa = (psa-1)/2;
  } // else

} // while

// Send the items_frequency2 & itemsorder2 from processor 0 to the rest
itemsorder = itemsorder2;
items_frequency = items_frequency2;

MPI_Bcast(items_frequency,number_freq_items,MPI_INTEGER,0,MPI_COMM_WORLD);
MPI_Bcast(itemsorder,number_freq_items,MPI_INTEGER,0,MPI_COMM_WORLD);

for(int w = 0; w<number_freq_items; w++)
{
```

```
    ranking[itemsorder2[w]-min_itemnr] = w;
  }

  MPI_Barrier(MPLCOMM_WORLD);
}//initialcounts::initialsort



//=============================================================
//
// FP BUILDING

template <class T>
struct FPtreenode {
  unsigned short info;
  T count;
  unsigned short mark;
  FPtreenode<T> *child;
  FPtreenode<T> *brother;
  FPtreenode<T> *nextsame;
  FPtreenode<T> *father;
};

template <class T>
class FP
{
  public:
    FPtreenode<T> *FProot;
    FPtreenode<T> **layer;
    FP ( ) { FProot = NULL; }
    void buildFP (char *infilename);
    void receiveTransactions();
  private:
    void updateFPtree (bool *next_transaction);
};

template <class T>
void FP<T>::receiveTransactions()
{
  MPI_Status status;  // Return status of the receive
  bool done = false;
  bool *next_transaction=NULL;
  int flag=0;

  next_transaction = new bool[initialcounts::number_freq_items];
  layer = new FPtreenode<T>*[initialcounts::number_freq_items];

  for (int i = 0; i < initialcounts::number_freq_items; i++ )
  {
    layer[i] = NULL;
  }

  FProot = new FPtreenode<T>;
  FProot->info = 0;
  FProot->child = FProot->brother = FProot->father = FProot->nextsame = NULL;
  FProot->count = 0;
  int buffer2[1];
  buffer2[0] = 1;

  do
  {
    MPI_Iprobe(MPI_ANY_SOURCE, 2, MPLCOMM_WORLD, &flag, &status);
    if (flag==1)
    {
      done =true;
    }

    MPI_Iprobe(MPI_ANY_SOURCE, 1, MPLCOMM_WORLD, &flag, &status);
```

```
    if (flag==1)
    {
      MPI_Recv(next_transaction, initialcounts::number_freq_items, MPI_CHAR, MPI_ANY_SOURCE, 1,
               MPI_COMM_WORLD, &status); // Should be a boolena instead of a char, but mpi_bool(ean)
                                         // does not exists
      updateFPtree(next_transaction);
      MPI_Send(buffer2,1,MPI_INT,status.MPI_SOURCE,1,MPI_COMM_WORLD);
    }
  }while (!done);
} // receiveTransactions

// push next_transaction into FP-tree
template <class T>
void FP<T>::updateFPtree (bool *next_transaction)
{
  int art;
  FPtreenode<T> *ptr = FProot;
  FPtreenode<T> *kid;
  FPtreenode<T> *prev = NULL;
  for ( art = initialcounts::number_freq_items-1; art >= 0; art-- )
  {
    if ( next_transaction[art] )
    {
      kid = ptr->child;
      if ( kid )
      {
        while ( kid && kid->info != art )
        {
          prev = kid;
          kid = kid->brother;
        }//while
        if ( kid )
        {
          kid->count++;
          ptr = kid;
        }//if
        else
        {
          prev->brother = new FPtreenode<T>;
          if ( ptr == FProot )
            prev->brother->father = NULL;
          else
            prev->brother->father = ptr;
          ptr = prev->brother;
          ptr->count = 1;

          ptr->mark = ( ptr->info = art ) + 1;;
          ptr->child = ptr->brother = NULL;
          ptr->nextsame = layer[art];
          layer[art] = ptr;
        }//else
      }//if
      else
      {
        ptr->child = new FPtreenode<T>;
        if ( ptr == FProot )
          ptr->child->father = NULL;
        else
          ptr->child->father = ptr;
        ptr = ptr->child;
        ptr->count = 1;

        ptr->mark = ( ptr->info = art ) + 1;
        ptr->child = ptr->brother = NULL;
        ptr->nextsame = layer[art];
        layer[art] = ptr;
      }//else
    }//if
```

```
  }//for
}//FP::updateFPtree

// reads the entire datafile from file inputdata and
// puts it into an FP-tree
// reads whole file (for the fourth time)!
template <class T>
void FP<T>::buildFP (char *infilename)
{
  int pos, item, items_count, i;
  unsigned short newitemnr;
  char c;
  bool *next_transaction = new bool[initialcounts::number_freq_items];
  layer = new FPtreenode<T>*[initialcounts::number_freq_items];
  for ( i = 0; i < initialcounts::number_freq_items; i++ )
    layer[i] = NULL;
  FProot = new FPtreenode<T>;
  FProot->info = 0;
  FProot->child = FProot->brother = FProot->father = FProot->nextsame = NULL;
  FProot->count = 0;

  int line = 0;

  int buffer2[10];

  MPI_Status status;
  ifstream fin (infilename);
  if ( ! fin )
    cout << "No_such_filename" << endl;

  do
  {
    if((line % (ps-1)) == (my_rank-1) )
    {
      for ( int column = 0; column < initialcounts::number_freq_items; column++ ) {
        next_transaction[column] = false;
      }
      items_count = 0;
      do
      {
        fin.get(c);
        item = 0;
        pos = 0;
        while ( ( c >= '0' ) && ( c <= '9' ) && ! fin.eof ( ) )
        {
          item = 10*item + (int)(c) -(int)('0');
          pos++;
          fin.get (c);
        }//while

        if ( pos && initialcounts::ranking[item-initialcounts::min_itemnr] >= 0 )
        {
          newitemnr = initialcounts::ranking[item-initialcounts::min_itemnr];
          next_transaction[newitemnr] = true;
          items_count++;
        }//if
      } while (c != '\n' && ! fin.eof ( ) );

      if ( items_count >= 1 )  // perhaps 2, but does it really matter?
      {
       MPI_Sendrecv(next_transaction, initialcounts::number_freq_items, MPI_CHAR,0,1, buffer2, 10,
                    MPI_INT,0 , 1, MPI_COMM_WORLD, &status);
      }
    } // if lines
    else
    {
      fin.get (c);
      while ( c != '\n' && ! fin.eof ( ) )
```

```
      {
        fin.get (c);
      }//while
    }
    line++;
  }while( ! fin.eof ( ) );

  int * HIER = NULL;
  HIER  = new int[1];

  if(my_rank != 1)MPI_Send(HIER, 1, MPI_INT,1,12,MPLCOMM_WORLD);

  if(my_rank == 1)
  {
    int klaar = 1;
    MPI_Status status;  // Return status of the receive
    while(klaar<ps-1)
    {
      MPI_Recv(HIER, 1, MPI_INT,MPI_ANY_SOURCE,12,MPLCOMM_WORLD,&status);
      klaar++;
    }
  }

  if(my_rank == 1)
  {
    MPI_Send(next_transaction, initialcounts::number_freq_items, MPI_CHAR,0,2,MPLCOMM_WORLD);
// STOP
  }

  fin.close ( );
  delete [ ] next_transaction;
}//FP::buildFP



//=============================================================
//
// COUNTING

template <class T>
struct bucket
{
  unsigned short itemvalue;
  T count;
  T aux;
  unsigned short number_followers;
  bucket *next;
};

template <class T>
class trie
{
  public:
    trie ( ) { };
    trie (initialcounts & initialdata);
    void build_up (FP<T> & theoriginaltree);
    void printtrie (char *outputdata);
  private:
    int length_count[MAXDEPTH];
    bool *frequent;
    void fpcount3 (unsigned short *nodes, bucket<T> *trienode, unsigned short number_buckets);
    void makeaux0 (bucket<T> *root, unsigned short number);
    FILE *outfilename;
    struct bucket<T> *root;
    unsigned short triesize;
    unsigned short k;
    T thevalue;
    unsigned short *thearraypointer;
```

```
    int results[MAXDEPTH];
    unsigned short *follows;
    struct bucket<T> **roots;
    void copying (bucket<T> *p, bucket<T> *q, unsigned short number_q_buckets);
    void printout (int depth, bucket<T> *trienode, unsigned short number_buckets);
    void dotheFPtree (FPtreenode<T> *itsroot);
};

// constructor
template <class T>
trie<T>::trie (initialcounts & initialdata)
{
  triesize = initialcounts::number_freq_items;
  root = new bucket<T>[triesize];
  frequent = new bool[triesize];
  thearraypointer = new unsigned short[triesize+1];
  *thearraypointer = 0;
  thearraypointer++;
  follows = new unsigned short[triesize];
  roots = new bucket<T>*[triesize];
  for ( unsigned short itemnr = 0; itemnr < triesize; itemnr++ )
  {
    root[itemnr].count = 666;
    // to avoid problems with short's;
    // this particular count-field is never used anymore (I hope)
    root[itemnr].itemvalue = itemnr;
    root[itemnr].number_followers = 0;
    roots[itemnr] = root[itemnr].next = NULL;
    follows[itemnr] = 0;
  }//for
}//trie::trie

// do the FP-counting, already knowing the relevant frequent 2-itemsets
// now the deeper levels
// number_buckets > 0, nodes contains a sentinel 0 at the start
template <class T>
void trie<T>::fpcount3 (unsigned short *nodes, bucket<T> *trienode, unsigned short number_buckets)
{
  unsigned short x = trienode->itemvalue;
  do
  {
    while ( *nodes != x )
      if ( *nodes > x )
      {
        trienode++;
        if ( ! ( --number_buckets ) )
          return;
 x = trienode->itemvalue;
      }//if
      else
 if ( ! *--nodes )
   return;
    trienode->aux += thevalue;
    if ( ! *--nodes )
      return;
    if ( trienode->number_followers )
      fpcount3 (nodes, trienode->next, trienode->number_followers);
    trienode++;
    if ( ! ( --number_buckets ) )
      return;
    x = trienode->itemvalue;
  } while ( true );
}//trie::fpcount3

// traverse the FP-tree
template <class T>
void trie<T>::dotheFPtree (FPtreenode<T> *itsroot)
{
```

```
  itsroot = itsroot->child;
  while ( itsroot )
  {
    if ( itsroot->mark == k )
    {
      if ( frequent[*thearraypointer = itsroot->info] )
      {
  if ( follows[*thearraypointer] )
  {
   thevalue = itsroot->count;
    fpcount3 (thearraypointer, roots[*thearraypointer],
        follows[*thearraypointer]);
 }//if
 thearraypointer++;
  dotheFPtree (itsroot);
  thearraypointer--;
      }//if
      else
        dotheFPtree (itsroot);
    }//if
    itsroot = itsroot->brother;
  }//while
}//trie::dotheFPtree

// build trie out of FP-transactions
template <class T>
void trie<T>::build_up (FP<T> & theoriginaltree)
{
  FPtreenode<T> *goingup;
  FPtreenode<T> *globpointer;
  T *counttwoitemsets = new T[triesize];
  int cnt, i, suppo;
  k = triesize - 2;
  while ( true )
  {
    globpointer = theoriginaltree.layer[k];
    for ( i = k+1; i < triesize; i++ )
      counttwoitemsets[i] = 0;
    while ( goingup = globpointer )  // note single =!
    {
      suppo = globpointer->count;
      while ( ( goingup = goingup->father ) && goingup->mark > k )
      {
        counttwoitemsets[goingup->info] += suppo;
        goingup->count = suppo;
        goingup->mark = k;
      }//while
      while ( goingup )
      {
        counttwoitemsets[goingup->info] += suppo;
        goingup->count += suppo;
        goingup = goingup->father;
      }//while
      globpointer = globpointer->nextsame;
    }//while
    cnt = 0;
    for ( i = triesize-1; i >= k+1; i-- )  // in this order!
      if ( counttwoitemsets[i] < minsup )
      {
        root[i].aux = 0;
        frequent[i] = false;
      }//if
      else
      {
        root[i].aux = counttwoitemsets[i];
        frequent[i] = true;
        makeaux0 (roots[i], follows[i]);
        cnt++;
```

```
      }//else
    if ( cnt )
    {
      dotheFPtree (theoriginaltree.FProot);
      roots[k] = root[k].next = new bucket<T>[cnt];
      follows[k] = root[k].number_followers = cnt;
      copying (roots[k], root+k+1, triesize-k-1);
    }//if

    if ( k == 0 )
      break;
  k--;
  }//while
}//trie::build_up

// make all necessary aux-fields 0
template <class T>
void trie<T>::makeaux0 (bucket<T> *root, unsigned short number)
{
  for ( int j = 0; j < number; j++ )
  {
    root->aux = 0;
    if ( root->number_followers && frequent[root->itemvalue] )
      makeaux0 (root->next, root->number_followers);
    root++;
  }//for
}//trie::makeaux0

// copy trie structure from q to p
template <class T>
void trie<T>::copying (bucket<T> * p, bucket<T> *q,
            unsigned short number_q_buckets)
{
  short temp;  // how many buckets does p need?
  short i;
  for ( int source = 0; source < number_q_buckets; source++ )
  {
    if ( q->aux >= minsup )
    {
      p->count = q->aux;
      p->itemvalue = q->itemvalue;
      temp = 0;
      for ( i = q->number_followers-1; i >= 0; i-- )
        if ( q->next[i].aux >= minsup )
     temp++;
      p->number_followers = temp;
      if ( temp > 0 )
      {
        p->next = new bucket<T>[temp];
        copying (p->next, q->next, q->number_followers);
      }//if
      else
        p->next = NULL;
      p++;
    }//if
    q++;
  }//for
}//trie::copying

// print resulting trie and frequency of each pattern length
template <class T>
void trie<T>::printtrie (char *outputdata)
{
  int k;
  for ( k = 0; k < MAXDEPTH; k++ )
    length_count[k] = 0;
  outfilename = fopen (outputdata, "w");
  fprintf (outfilename, "(%d)\n", initialcounts::lines);  // empty set
```

```
  printout (1, root, triesize);
  int lpl;
  for ( lpl = MAXDEPTH-1; lpl >= 0 && length_count[lpl] == 0; lpl-- )
    ;
  if ( initialcounts::lines >= minsup )
    printf ("1\n");
  for ( k = 1; k <= lpl; k++ )
    printf ("%d\n", length_count[k]);
  fclose(outfilename);
}//trie::printtrie

// do the printing
template <class T>
void trie<T>::printout (int depth, bucket<T> *trienode,
            unsigned short number_buckets)
{
  for ( int i = 0; i < number_buckets; i++ )
  {
    results[depth] = trienode[i].itemvalue;
    length_count[depth]++;
    for ( int j = 1; j <= depth; j++ )
      fprintf (outfilename, "%d ", initialcounts::itemsorder[results[j]]);
    if ( depth > 1 )
      fprintf (outfilename, "(%d)\n", trienode[i].count);
    else
      fprintf (outfilename, "(%d)\n", initialcounts::items_frequency[i]);
    if ( trienode[i].number_followers > 0 )
      printout (depth+1, trienode[i].next, trienode[i].number_followers);
  }//for
}//trie::printout


// main program
int main (int argc, char *argv[ ])
{
  long int time1, time2, c;
  minsup = macro_minsup;
  time1 = time (&c);

  MPI_Init(&argc, &argv);
  initialcounts initialdata (input_filename);
  time2 = time (&c);

  printf ("Execution time - reading: %ld\n", time2-time1);

  MPI_Barrier(MPLCOMM_WORLD);

  if ( initialcounts::items_frequency[initialcounts::number_freq_items-1] >= USHRT_MAX )
  {
    time1 = time (&c);
    printf ("Building (big) FP-tree starts ...\n");
    FP<int> theFPtree;
    if (my_rank != 0)
    {
      theFPtree.buildFP (input_filename);
    }
    else
    {
      theFPtree.receiveTransactions();
    }
    printf ("FP-tree completed\n");
    time2 = time (&c);
    printf ("Execution time - FP-phase: %ld\n", time2-time1);
    time1 = time (&c);
    printf ("Counting (big) starts ...\n");
    trie<int> ourbigtrie (initialdata);
    ourbigtrie.build_up (theFPtree);
    time2 = time (&c);
```

```
    printf ("Execution time - counting: %ld\n\n", time2-time1);
    if ( argc > 3 )
    {
      ourbigtrie.printtrie (output_filename);
    }
  }//if
  else {
    time1 = time (&c);
    FP<unsigned short> theFPtree;
    if (my_rank != 0)
    {
      theFPtree.buildFP (input_filename);
    }
    else
    {
      theFPtree.receiveTransactions();
    }

    MPI_Barrier(MPI_COMM_WORLD);

    if(my_rank==0)
    {
      printf ("FP-tree completed\n");
      time2 = time (&c);
      printf ("Execution time - FP-phase: %ld\n", time2-time1);
      time1 = time (&c);
      printf ("Counting (small) starts ...\n");
      trie<unsigned short> oursmalltrie (initialdata);
      oursmalltrie.build_up (theFPtree);
      time2 = time (&c);
      printf ("Execution time - counting: %ld\n\n", time2-time1);
      if ( argc > 3 )
      {
        oursmalltrie.printtrie (output_filename);
      }
    }//if
  }//else

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();

  return 0;
}//main

\
```

References

[1] Foster. What is the Grid? A Three Point Checklist, July 2002.

[2] Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In Int'l. J. High-Performance Applications. Vol. 15, no. 3, 2001, pp 200-222.

[3] M. Cannataro, D. Talia. The Knowledge Grid: Designing, building, and implementing an architecture for distributed knowledge discovery. In Communications of the ACM. Vol. 46, no. 1, 2003, pp 89-93.

[4] P. Brezany. GridMiner - a Framework for Data Integration & Knowledge Discovery on Computational Grids. Slides, April 2005, Vienna.

[5] P. Brezany, J. Hofer, A Min Tjoa, A. Wöhrer. GridMiner: An Infrastructure for Data Mining on Computational Grids. In Proceedings of the APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch, Queensland Australia, October 2003.

[6] P. Brezany, I. Janciak, A. Wöhrer, A M. Tjoa. GridMiner: A Framework for Knowledge Discovery on the Grid - from a Vision to Design and Implementation. Cracow Grid Workshop, Cracow, December 2004.

[7] J. Hofer, P. Brezany. DIGIDT: Distributed Classifier Construction in the Grid Data Mining Framework GridMiner-Core. In Proceedings of the Workshop on Data Mining and the Grid (DM-Grid 2004) held in conjunction with the 4th IEEE International Conference on Data Mining (ICDM'04), Brighton, UK, November 1-4, 2004.

[8] G. Kickinger, J. Hofer, P. Brezany, A.M. Tjoa. Grid knowledge discovery processes and an architecture for their composition. In Proceeding of Parallel and Distributed Computing and Networks, Innsbruck, Austria, February 2004.

[9] I. H. Witten, E. Frank. Data Mining: Practical Machine Learning Techniques with Java Implementations. Morgan Kaufmann, San Francisco, CA, 2000.

[10] W. Frawley, G. Piatetsky-Shapiro, C. Matheus. Knowledge Discovery in Databases: An Overview. AI Magazine, 213-228, 1992.

[11] P. Jermyn, M. Dixon, B.J. Read. Preparing Clean Views of Data for Data Mining.

[12] Two Crows. Introduction to Data Mining and Knowledge Discovery. ISBN: 1-892095-02-5. 2005

[13] D. Pearson. Data requirements for the Grid. 2002

[14] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[15] M.S. Prez, A. Snchez, P. Herrero, V. Robles, J.M. Pea. Adapting the Weka Data Mining Toolkit to a Grid Based Environment, Lecture Notes in Computer Science, Volume 3528, Jan 2005, Pages 492 - 497.

[16] D. Talia, P. Trunfio, O. Verta. Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids. Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005), Porto, Portugal, October 2005, LNAI vol. 3721, pp. 309-320, Springer-Verlag, 2005.

[17] D. Talia. Grid-Based Distributed Data Mining Systems, Algorithms and Services. In Proceedings of the 9th International Workshop on High Performance and Distributed Mining (HPDM), April 2006.

[18] D. Talia, P. Trunfio, O. Verta. WSRF Services for Composing Distributed Data Mining Applications on Grids: Functionality and Performance. In Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2006), Vol. 3980:1080-1089 of LNCS, Springer-Verlag, Glasgow, UK, May 2006.

[19] P. Brezany, I. Janciak, A. Min Tjoa. GridMiner: a Fundamental Infrastructure for Building Intelligent Grid Systems. In Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, 2005.

[20] G. Kickinger and J. Hofer and A. Tjoa and P. Brezany. Workflow Management in GridMiner. In Proceedings of the 3rd Cracow Grid Workshop, Cracow, Poland, October 2003.

[21] W. Kosters and W. Pijls. Apriori, A Depth First Implementation. In Proceedings of the FIMI Workshop Of Frequent Itemset Mining Implementation, Melbourne, Florida, USA, November 19 2003.