

**A STUDY OF
DISTRIBUTED CLUSTERING OF
VECTOR TIME SERIES ON THE GRID
BY TASK FARMING**

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Industrial Engineering

In

The Department of Industrial & Manufacturing Systems Engineering

by
Arun Nayar
B.Tech., University of Kerala, India, 1999
May 2005

Table of Contents

ABSTRACT	iii
1. INTRODUCTION	1
2. DATA MINING	3
2.1. Neural Networks	3
2.2. Decision Trees	9
3. DISTRIBUTED DATA MINING	16
3.1. Agent Based Distributed Data Mining	17
4. GRID COMPUTING	21
4.1. Distributed Data Mining on the Grid	24
4.2. Condor	27
4.3. Taskfarming	29
5. CLUSTERING OF VECTOR TIME SERIES	34
5.1. Vector Time Series	34
5.2. Fuzzy Clustering	35
5.3. Interpolation	37
5.4. Cluster Validity Index	37
6. DATA DESCRIPTION	40
6.1. Model Generated Data	40
6.2. Battle Simulation Data	41
7. DESCRIPTION OF FACILITIES.....	43
8. METHODOLOGY	44
9. RESULTS AND DISCUSSION	50
9.1. Results	50
9.2. Discussion	56
10. CONCLUSIONS AND FUTURE DIRECTION	59
REFERENCES	61
APPENDIX: SUPPLEMENTARY DATA	65
VITA.....	67

Abstract

Traditional data mining methods were limited by availability of computing resources like network bandwidth, storage space and processing power. These algorithms were developed to work around this problem by looking at a small cross-section of the whole data available. However since a major chunk of the data is kept out, the predictions were generally inaccurate and missed out on significant features that was part of the data. Today with resources growing at almost the same pace as data, it is possible to rethink mining algorithms to work on distributed resources and essentially distributed data. Distributed data mining thus holds great promise. Using grid technologies, data mining can be extended to areas which were not previously looked at because of the volume of data being generated, like climate modeling, web usage, etc. An important characteristic of data today is that it is highly decentralized and mostly redundant. Data mining algorithms which can make efficient use of distributed data has to be thought of. Though it is possible to bring all the data together and run traditional algorithms, this has a high overhead, in terms of bandwidth usage for transmission, preprocessing steps which have to be to handle every format the received data. By processing the data locally, the preprocessing stage can be made less bulky and also the traditional data mining techniques would be able to work on the data efficiently. The focus of this project is to use an existing data mining technique, fuzzy c-means clustering to work on distributed data in a simulated grid environment and to review the performance of this approach viz., the traditional approach.

1. Introduction

“At least 500,000 people are estimated to die of cancer in the US every year [1].”

“The US defense budget for 2004 was above \$4 billion [2].”

These numbers hold a great deal of significance to the man on the street. The above numbers have been mentioned because it is obvious that a decrease in them has a direct impact on the well being of the nation. In this respect, the numbers have a greater meaning; they are “data”. Any study on analyzing or reducing these numbers needs a mechanism to read through the entire data and discover knowledge in them. This is the primary goal of data mining. It seeks to discover patterns in the data, which could be cause and effect, association or any other relationship. Several different data mining techniques have been developed over the years, some based on mathematical models like neural networks, genetic algorithms, etc, and others based on diagrammatic and logical representations of data like decision trees, fuzzy clustering and Boolean functions.

In today’s world, data is being constantly generated, be it scientific research, market analysis, medical reports, customer surveys, etc. The data is mostly decentralized and can occur in different formats. Hence an interesting challenge would be to combine this data in some standard format for mining purposes. Another possible approach that I have decided to pursue as part of this research is to process the data in situ and combine the results in a central repository. The advantages of the second approach are that raw data is normally bulky and might also have sensitive information. By processing in-situ only broad characteristics in the data will be transmitted, which would be less bulky to

transmit over the bandwidth. Also for data occurring in different formats it is less expensive to process it locally as it is enough to maintain one version of the application.

Vector time series is prevalent in different areas. It is computationally expensive and time consuming to analyze such time series data. Several methods have been proposed to reduce the time series data by identifying similar series and eliminating them, which makes the resultant data suitable for any analysis. Most of these reduction techniques use some kind of partitioning method to identify patterns or clusters in the data.

In the following sections, a brief overview of all the concepts involved in this research is summarized. Review of literature pertinent to each section is mentioned. Following this, the proposed methodology is outlined in detail. The results are then presented and discussed. Conclusions based on the results are presented along with some of the issues that were faced in this new methodology. The thesis is rounded out by pointing out some future directions for this research.

2. Data Mining

Data mining [3] refers to the set of techniques which are used to analyze data, usually available in extremely large quantities and derive meaningful information out of it, which may be put to some kind of practical use. These techniques are based on statistics and logic. Data mining techniques are classified as supervised or unsupervised. In supervised “learning”, the target variable is known and the learning technique attempts to identify the relation of unknown data samples to the target variable. In unsupervised learning, the target variable is not known, and we seek to identify discernible patterns in the data that may be inferred as some kind of useful information.

Data mining finds patterns and relationships in data by using sophisticated techniques to build models. There are two main kinds of models in data mining: *predictive* and *descriptive*. Predictive models can be used to forecast explicit values, based on patterns determined from known results. For example, from a database of customers who have already responded to a particular offer, a model can be built that predicts which prospects are most likely to respond to the same offer. Descriptive models describe patterns in existing data, and are generally used to create meaningful subgroups such as demographic clusters

In the following sections, different data mining techniques are described in detail.

2.1. Neural Networks

Neural networks grew out of research in artificial intelligence; specifically, attempts to mimic the fault-tolerance and capacity to learn of biological neural systems by modeling the low-level structure of the brain. The basic unit of a neural network is a

neuron. A neuron is made up of three elements, input, threshold and output. An output will only be generated if the sum of inputs exceeds the threshold value. Weights are used as filters for the input values. Given an input, the output can be changed by altering the threshold value and/or the weights. The ability to change the output to reflect the input is important; this ability gives the neuron the capability to learn. The neuron can be taught by setting the weights to an initial value, then giving the neuron some desired inputs and then adjusting the weights so that the desired output is obtained.

As an example consider a simple adaptive neural network called a perceptron;

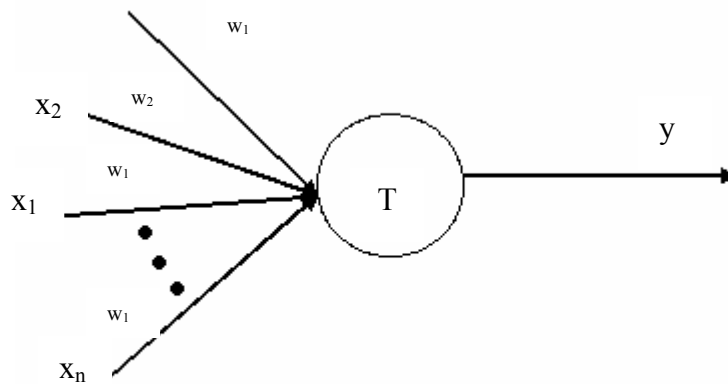


Figure 1: A simple neural network with n weighted inputs and a single output

The following are the variables used in the calculations:

Inputs - x_1, x_2, \dots, x_n . These could be real numbers or Boolean values depending on the problem.

Output – y and is Boolean.

Weights - w_1, w_2, \dots, w_n are weights of the edges and are real valued.

T is the threshold and is real valued.

W - Weighted input, $w_1 x_1 + w_2 x_2 + \dots + w_n x_n$

If $W > T$ then $y = 1$, else $y = 0$

The perceptron is trained to respond to certain inputs with certain desired outputs. Weights are determined so that when applied to the inputs the outputs obtained are as close as possible to the desired outputs. After the training period, the perceptron should be able to give reasonable outputs for any kind of input. Given a new input for which it was not trained, the perceptron will try to find the best possible output depending on how it was trained.

Frank Rosenblatt [4] first mentioned the use of the single layer perceptron in supervised learning. Widrow and Hoff [5] developed the ADALINE, an acronym for ADAPtive LInear NEuron, which minimizes the output error using a non-linear optimization technique called the gradient descent, where the error function is moved incrementally in the search space in order to obtain a minimum. After each training case is presented, the correction to be applied to the weights is proportional to the error. This method is also known as the delta rule. The correction is calculated *before* the thresholding step.

Most modern neural networks are of the multi layered type. McClelland and Rumelhart [6] developed the network architecture that is popularly known as a *multi-layered* perceptron (MLP). The back propagation algorithm is a modification of the delta rule in which extra *hidden layers* are added.

In Figure 3, the algorithm for a simple back propagation neural network with a single layer is given. The algorithm involves two steps namely:

- i. Training the neural network;

The weighted sum of the input variables is calculated by assigning arbitrary weights. The activation function is then used to obtain an approximate output.

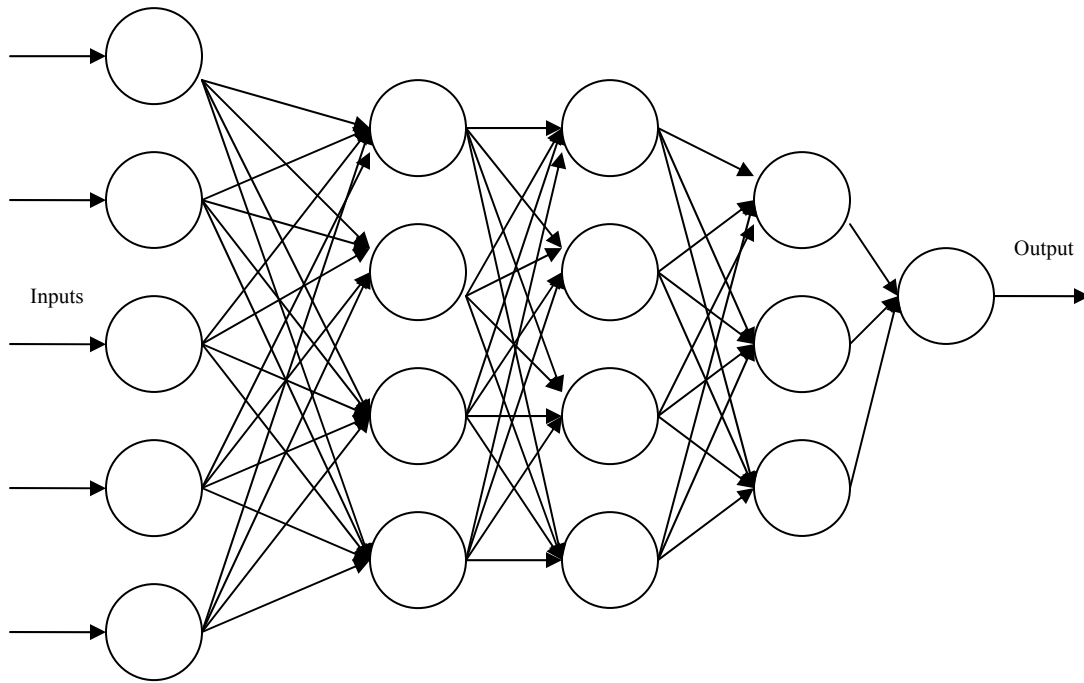


Figure 2: A multilayer perceptron with 5 layers and 5 inputs.

INPUT: ε – Error limit
 E – error
 W – weight
 V – Variance of the target output
 p – number of patterns
 η – random number between 1 and p
 σ – activation function, either linear or sigmoidal
 G – gradient
 μ – step size
 t – desired output value

Procedure:

```

while E/V >  $\varepsilon$ 

    G = 0

    for  $\eta = 1$  to p
        /* forward propagate */
        for i = 1 to m

             $z_i = \sigma(\sum W_i x_i)$ 
        end for
    end for

    /* backward propagate */
    for i = 1 to m

         $y_i = z_i (1 - z_i)(t_i - z_i)$ 

    end for

    /* accumulate the gradient */
     $G_{ij} = G_{ij} - y_i x_j$ 

    /* Update weights */
     $W_{ij} = W_{ij} - \mu G_{ij}$ 
  
```

Figure 3: Pseudocode for a single layer back propagation neural network

ii. Adjusting the weights by backpropagation

This output is then compared to the original output and the error determined.

This is then used to adjust the weights. The entire procedure is repeated till the error falls below a pre-determined limit.

The back propagation algorithm works as follows:

1) Given a number of examples with known outputs, one of them is selected at random and applied to the network. The network produces an output based on the current state of the weights assigned.

2) The output is compared to the known output of the example and a mean-squared error value is calculated.

3) The error value is then propagated backwards through the network, and small changes are made to the weights in each layer. The weight changes are calculated to reduce the error signal for the case in question.

4) The whole process is repeated for each of the example cases, then back to the first case again, and so on. The cycle is repeated until the overall error value drops below some pre-determined threshold or when a specified number of iterations are completed.

Fahlmann [7] developed the *quick propagation* algorithm, which was a modification of the back propagation method. In this the gradient of the errors for all the training cases was averaged and the weights were updated only at the end. Jacobs [8] developed another variant of the back propagation method known as delta-bar-delta which also calculated the average gradient error for all cases. It also included a learning rate, which was also updated according to whether the gradients were positive or negative.

Another popular MLP method is the radial basis function method. Broomhead and

Lowe [9] developed the radial basis function network in which the classification of points was based on their radial distance from the center. This method has the advantage of having only one hidden layer in the model.

The above methods are all based on supervised learning. The term "supervised" learning is usually applied to cases in which a particular classification is already observed and recorded in a training sample, and it is required to build a model to predict those classifications (in a new testing sample).

As opposed to this, in unsupervised learning, the outcome variable of interest is not directly observed. Instead, the objective is to detect some "structure" or clusters in the data that may not be easily observable. Only after identifying certain clusters can they be classified based on subsequent research. The main advantage of using unsupervised neural networks is that they do not require target values for their outputs. Kohonen [10] developed the self-organizing feature map (SOFM) network, which was primarily designed for unsupervised learning. SOFM networks are trained using an iterative algorithm. The algorithm learns to recognize patterns in the input data and allocates them to appropriate nodes in the output array, called 'bins'. Each bin represents a specific pattern. The nodes in the output array are arranged such that the neighboring bins represent very similar patterns and bins that are well separated represent very different patterns. The clusters are thus identified and then used for classification.

2.2. Decision Trees

A decision tree is a graphical representation of a procedure for classifying or evaluating an item of interest. A decision tree represents a function that maps each element of its domain to an element of its range, which is typically a class label or

numerical value. Each node of a decision tree corresponds to a test on one attribute of the input data, and each edge represents a subset of the values of the previous node's attributes. Classification of an object with a decision tree begins at the root and a corresponding test is performed. Depending on the outcome it follows down the edge that has the appropriate value and performs the test of the next node. This is repeated till a leaf of the tree is reached and the value of its classification label is returned.

Quinlan [11] first applied decision trees for learning classification procedures in chess endgames. The ID3 algorithm uses the concept of information gain based on entropy to decide the structure of the decision tree. The ID3 algorithm has the following requirements:

- Feature-value description - the same features must describe each example and have a fixed number of values.
- Predefined classes - an example's features must already be defined, that is, they are not learned by ID3.
- Discrete classes - classes must be sharply delineated. Continuous classes broken up into vague categories such as a metal being "hard, quite hard, flexible, soft, quite soft" are suspect.
- Sufficient examples - since inductive generalization is used (i.e. not provable) there must be enough test cases to distinguish valid patterns from chance occurrences.

To decide which attribute is to be used to split the given sample set at each node, the ID3 algorithm uses the information gain due to each attribute by calculating the entropy that is a measure of the information contained in an attribute.

Given a collection S of c outcomes

Entropy(S) = $- \sum p(I) \log p(I)$ where $p(I)$ is the proportion of S belonging to class I .

Once the entropy is determined, the information gain is calculated by finding the difference between the total entropy and the average entropy due to each value of the attribute.

Gain(S, A) is information gain of example set S on attribute A is defined as

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_v \left(\frac{|S_v|}{|S|} \right) * \text{Entropy}(S_v)$$

where:

S_v is each value v of all possible values of attribute A

S_v = subset of S for which attribute A has value v

$|S_v|$ = number of elements in S_v

$|S|$ = number of elements in S

For each attribute, the gain is calculated and the highest gain is used in the decision node.

As an example consider a decision model to determine whether the weather is amenable to playing baseball. Historic data of observations over a period of two weeks is available to build the model (table 1).

The target classification is "should we play baseball?" which can be yes or no.

The weather attributes are outlook, temperature, humidity, and wind speed. They can have the following values:

outlook = { sunny, overcast, rain }

temperature = {hot, mild, cool }

humidity = { high, normal }

wind = {weak, strong }

Table 1: Sample data for building a decision tree using ID3 algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

To determine the attribute that would be the root node for the tree, the gain is calculated for all four attributes.

The classification of these 14 examples are 9 YES and 5 NO. For attribute Wind, there are 8 occurrences of Wind = Weak and 6 occurrences of Wind = Strong. For Wind = Weak, 6 of the examples are YES and 2 are NO. For Wind = Strong, 3 are YES and 3 are NO. Therefore,

$$\begin{aligned} \text{Gain}(S, \text{Wind}) &= \text{Entropy}(S) - (8/14) * \text{Entropy}(S_{\text{weak}}) - (6/14) * \text{Entropy}(S_{\text{strong}}) \\ &= 0.940 - (8/14) * 0.811 - (6/14) * 1.00 \\ &= 0.048 \end{aligned}$$

$$\text{Entropy}(S_{\text{weak}}) = - (6/8) * \log_2(6/8) - (2/8) * \log_2(2/8) = 0.811$$

$$\text{Entropy}(S_{\text{strong}}) = - (3/6) * \log_2(3/6) - (3/6) * \log_2(3/6) = 1.00$$

Similarly the gain is calculated for the other attributes.

$$\text{Gain}(S, \text{Outlook}) = 0.246$$

$$\text{Gain}(S, \text{Temperature}) = 0.029$$

$$\text{Gain}(S, \text{Humidity}) = 0.151$$

Outlook attribute has the highest gain; therefore it is used as the decision attribute in the root node.

Since outlook has three possible values, the root node has three branches (sunny, overcast, rain). The next question is "what attribute should be tested at the sunny branch node?" Since outlook has already been used at the root, the decision is based on the remaining three attributes: Humidity, Temperature, or Wind.

$$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\} = 5 \text{ examples from Table 1 with outlook = sunny}$$

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = 0.970$$

$$\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = 0.570$$

$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = 0.019$$

Humidity has the highest gain; therefore, it is used as the decision node. This process goes on until all data is classified perfectly or we run out of attributes.

The decision tree obtained as the result of this algorithm is shown in Figure 4.

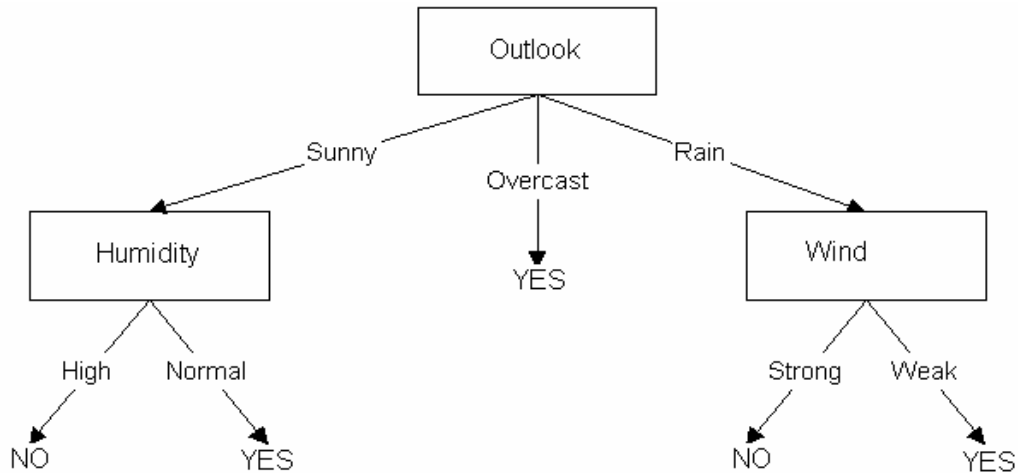


Figure 4: Decision tree to determine whether the weather is amenable to playing baseball

The following rules are generated from the decision tree;

IF outlook = sunny AND humidity = high THEN playball = no

IF outlook = rain AND humidity = high THEN playball = no

IF outlook = rain AND wind = strong THEN playball = yes

IF outlook = overcast THEN playball = yes

IF outlook = rain AND wind = weak THEN playball = yes

Quinlan [12] developed the C4.5 algorithm that was a modification of the ID3 algorithm. C4.5 introduced a number of extensions of the original ID3 algorithm. The algorithm was able to classify data which had unknown attribute values or which had

continuous values. The algorithm also incorporates post pruning after the induction of trees to improve the accuracy of classification.

Quinlan developed the FOIL algorithm, which induced trees by learning for known examples and extensional background knowledge. The general method of FOIL is to search the hypothesis space by adding one rule by another, and by constructing each rule literal by literal, guided by a heuristic measure, similar to information gain. The search for a hypothesis fitting the data is reduced to parts of the search space with highest information gain, which does not necessarily lead to optimal results.

Breiman, *et al.* [13] developed the CART algorithm, which was the first to consider a non-linear split based on a combination of the attributes of the sample data. The algorithm decides the splitting rule after doing a brute search of all possible splits in the given data and choosing the one that best separates the classes contained in the parent node. The algorithm used post pruning to improve the accuracy of the tree.

Clark and Niblett [14] developed the CN2 algorithm, which induced trees from a set of examples and used the concept of entropy and significance testing to decide the best split at each node. The algorithm tests for significant differences in the distribution of positive and negative examples covered by a rule and the overall distribution of positive and negative examples and rejects insignificant rules.

Cohen [15] developed the GROW algorithm which used a combination of pre and post pruning to induce a decision tree. The algorithm first generates a theory that overfits the training data. This intermediate theory is then augmented by generalizations of all its clauses. The augmented theory is then pruned till no further clause that improves the predictive accuracy on the pruning set can be found

3. Distributed Data Mining

Most of the existing data mining techniques were originally developed for centralized data and so were developed in a manner feasible to run on a single machine. But as the amount of data has grown and also become heterogeneous and available at multiple sites, it has become necessary to modify these algorithms. Extracting patterns from highly distributed datasets is known as distributed data mining. Distributed data mining is expected to perform partial analysis of the data at individual sites and then send the outcome as a partial result to other sites where it may be aggregated to the final result.

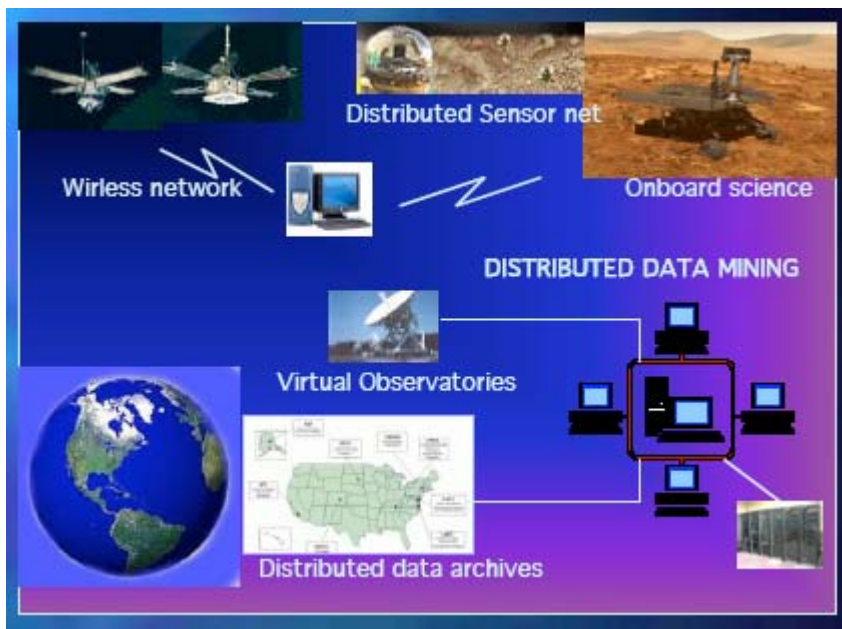


Figure 5: Distributed data mining in practice

To maintain sufficient generalization to incorporate the diverse range of data to be found on the distributed sources, distributed data mining employs a “meta-learning” technique. The models are computed by applying independent classifiers at the local sites. The base classifiers are then integrated by a separate learning process. Several methods for this integration have been studied and used, including techniques that

combine the set of models in some linear fashion, and techniques that employ referee functions to arbitrate among the predictions generated by the base classifiers.

Some popular distributed mining techniques include distributed association rules, distributed clustering, Bayesian learning, regression and compression. A very popular technique used in distributed data mining is the use of mining agents. These are semi-intelligent programs, which run as a layer above the mining algorithms and handle the communication, data selection, resource discovery and other useful coordination activities.

3.1. Agent Based Distributed Data Mining

Agent based distributed mining [16] has been especially attractive because of the following features:

- i. The agents are sufficiently modular to handle access to the underlying data source, which is autonomous of the system, data or model.
- ii. Agents reduce the amount of human intervention and supervision in running a data mining process.
- iii. Agents may be used to dynamically discover and select relevant data sources according to given criteria and support any OLAP and business data warehousing application.
- iv. Agents can be migrated to process data locally at each site in a system to reduce network and application server load. They may then return with or send pre-selected data back to the originating server.
- v. Agents may use a combination of different mining algorithms to operate on suitably complex applications. Based on the type of data retrieved from different

sites and the mining tasks to be performed, it will select the appropriate technique.

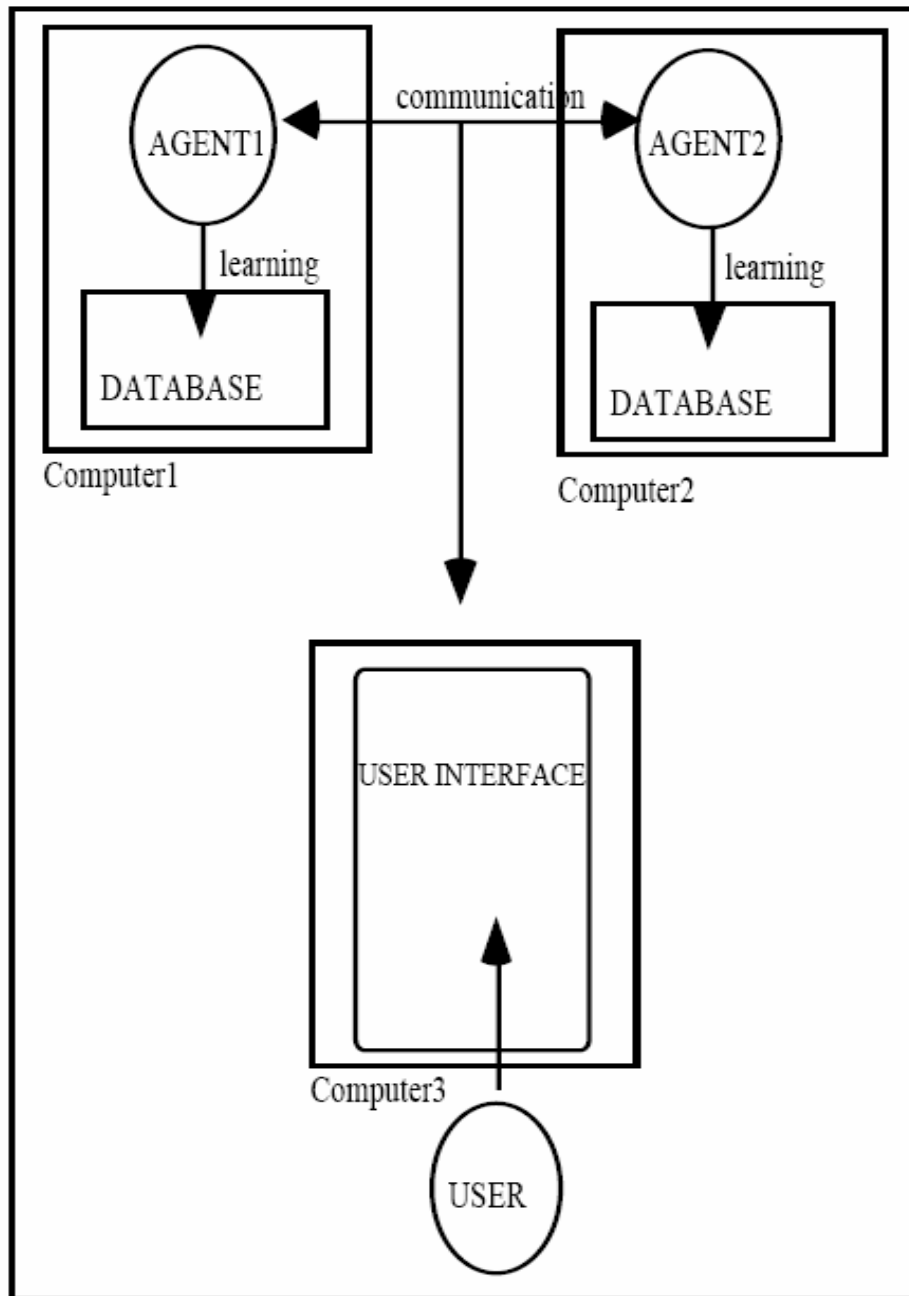


Figure 6: Distributed flow diagram for an agent based learner

Several commercial tools have been developed which make use of one or more of the above methods.

JAM [17] is a distributed agent-based data mining that uses meta-learning technique. Local databases of financial institutions are mined to develop local patterns,

which are then combined to develop final patterns. Each data site is assumed to consist of a local database, learning agents, and configuration modules, which are responsible for communication and distributing computing between the different sites.

PADMA [18] is an agent based distributed mining architecture. It works in a distributed environment based on cooperative agents. Partial data cluster models are first computed by local agents at different sites. All the local models are then collected to a central site that performs a second-level clustering algorithm to generate a global model. The individual local agents perform hierarchical clustering in text document classification and information visualization on the web.

Papyrus [19] is a Java based system, which addresses wide area distributed data mining over clusters of heterogeneous data sites and meta-clusters. The system supports different mining models like decision trees, clustering, etc. Agents distribute data among the local sites and return results to the central node, which produces the final result.

BODHI [20] is a hierarchical agent based distributed learning system. It uses local learning methods to build models at each site and then moves these models to a centralized location. The models are then combined to build a meta-model whose inputs are the outputs from the various models.

Kensington architecture [21] is a distributed component environment located on different nodes in a generic unsupervised network like the internet. The architecture provides different components like user-defined, Application servers and higher-level servers. The data analysis algorithm is implemented as Enterprise java beans components.

PaDDMAS [22] or Parallel and Distributed Data Mining Application Suite, is an extension of the Kensington architecture and implements additional features like support for third party components and a XML interface to hide the component implementation.

Distributed data mining architecture has also been developed to handle distributed data, which is heterogeneous.

Distributed data mining techniques are all concerned with issues like data security, access to remote resources, resource discovery and management. As such they can make use of the infrastructure provided by the grid-computing framework to handle communication between remote heterogeneous resources, data movement and security issues. In the next section, we define grid computing and the protocols available that are of interest to the distributed mining community.

4. Grid Computing

Grid computing [23] is used to refer to a distributed infrastructure that promotes large-scale resource sharing in a dynamic multi-institutional setup, also known as “virtual organizations”. In order to establish a computational grid, several institutions pool their resources. Global policies for the virtual organization are established that identify the roles and responsibilities of the participating entities. Applications containing sufficiently parallel subtasks can take advantage of the Grid to co-allocate a large number of distributed resources in parallel. To provide uniform resource sharing, a few protocols have been developed which define how distributed system elements interact with each other to achieve a specified behavior. Other elements of the Grid architecture are services which need to be present on each of the resources to handle various requests and information exchanges, an Application Programming Interface (API) for users to write high-level applications that can run on the grid by abstracting access to the underlying services and protocols, and a Software Development Kit which will help users extend the functionality provided by the Grid architecture.

The Grid architecture provides among other features the following:

Resource management [24] – In a distributed environment, it is important to locate and allocate computational resources, provide authentication, create processes in the remote resources, etc. Resource management is concerned with the handling of five important challenges:

- Site autonomy: Since different resources are owned and operated by different organizations, there is not likely to be commonality in acceptable use policy,

- scheduling policies, security mechanisms, etc.
- Heterogeneous substrate: Different sites may have different local resource management system. Even when same system is used, different configurations and local modifications can lead to significant differences in functionality.
 - Policy extensibility: Since meta-computing applications are derived from a wide range of domains with their requirements, it is necessary to support new domain-specific management structures without requiring changes to code at each installed site.
 - Co-allocation: Applications have resource requirements that may be satisfied only by using resources simultaneously at several sites. There is a need for specialized mechanisms for allocating multiple resources, initiating computations on those resources and monitoring and managing those computations.
 - Online control: A negotiation mechanism is needed to match application requirements to resource availability, when requirements and resource characteristics change during execution.

The Globus [25] project which aims to build a unifying architecture for grid computing implements resource management with a set of tools and specifications that take into account the above challenges. An extensible language specification, Resource Specification Language has been developed to handle online control and policy extensibility. Resource Managers provide a well defined interface diverse local resource management tools and policies. Resource brokers handle the mapping between high-level application requirements and requests to individual resource managers.

Information Services and Monitoring [26] – Sharing relationships in a grid environment may be static or dynamic but in most case, the participants are not aware of the resources contributed by all the participants in the virtual organization. Information services, which are capable of initial discovery and ongoing monitoring of the existence and characteristics of resources, services, computations and other entities are an important part of the Grid system.

An Information Services system as implemented by the Globus [27] project has the following features:

- Superscheduler routes computational requests to the best available computer in a Grid computing environment. The scheduler chooses based on information such as system configuration and dynamic information like instantaneous load and time of availability.
- Replica selection service hands out the best copy of files that are replicated on multiple storage systems based on information such as system configuration, storage capacity, instantaneous loads and network traffic on the storage system.
- Application adaptation agent monitors a running application and external resource availability and modifies application behavior and the resource consumption.
- Troubleshooting service monitors Grid resources looking for anomalous behaviors such as excess load or extended failure of critical services.
- Performance diagnosis tool is invoked by the user when anomalous behavior is detected and discovers what information sources are associated with an application and its resources.

Security [28] – In a Grid environment, applications need access to a number of processes on each participating resource. Also different participating sites might have different access mechanisms and local security policies. The security system provides authentication solutions that allow a user, the processes that comprise the application and computation and the resources, to verify each other's identity, and at the same time also apply any local policy without any change. The basic features that are provided by the security system as implemented in the Globus project are:

- Single sign-on: A user is able to authenticate just once when he starts the computation, further communication is handled automatically across resources and processes.
- Protection of credentials: User credentials such as passwords, pass-phrases, certificates must be protected from lose or corruption.
- The system is interoperable with the local security policy on the individual sites.
- Uniform credential infrastructure: Credentials are implemented as X509 certificates that are authorized by a signing authority. Certificates signed by trusted authorities may be used across multiple sites.

In the next section, attempts to incorporate distributed data mining and the Grid are summarized and also the issues and concerns of running distributed data mining on the Grid.

4.1. Distributed Data Mining on the Grid

New research into distributed data mining is trying to capitalize on the features provided by the Grid to encompass geographically separated computing resources to perform mining on extremely large datasets. Several attempts are currently underway to

establish a standard framework for running mining algorithms on the Grid.

Knowledge Grid [29][30] is a high level system that has been developed for providing grid based knowledge discovery services. The architecture of the system is composed of two layers, the Core layer that interfaces with the basic and generic Grid middleware services, and the High-level layer, which provides the user a set of services for the design and execution of knowledge discovery applications.

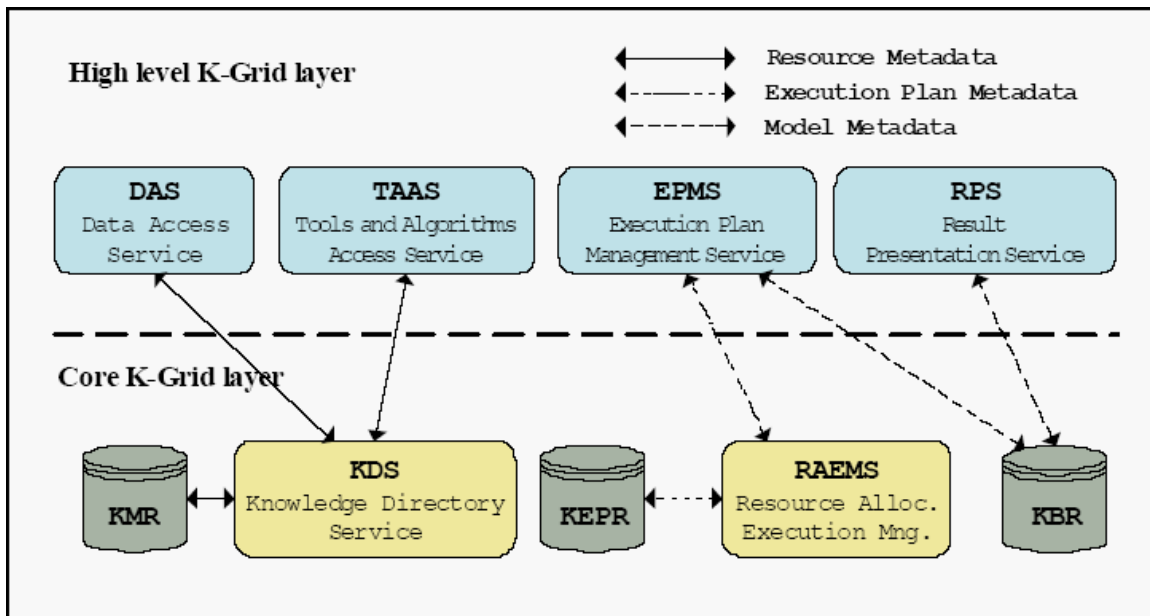


Figure 7: Layer decomposition of services in the Knowledge Grid

Figure 7 shows the general layout of the Knowledge Grid system and its main components and interaction patterns. The High-level layer includes services to compose, validate and execute a distributed or parallel data mining computation. These include the Data access layer for searching, selecting, transferring and delivering the data to be mined, Tools and Algorithms Access service for searching and selecting the data mining tools for the algorithms, Execution Plan management service for defining the structure and workflow of the application and the Results Presentation service to display and visualize the models extracted from the data.

The Core layer consists of two services; Knowledge Discovery Service, which manages metadata describing the Grid resources also including the data repositories, algorithms and tools necessary for the computation, and the Resource Allocation and Execution Management service that matches the execution plan schema to the available resources.

Unlike other applications on the Grid, running distributed mining on the grid presents certain unique issues:

- Very few sites can afford to keep enough disk space to allow the sudden and unpredictable arrival of a large dataset.
- Moving large quantities of data between the sites may choke the available bandwidth on the network causing other applications to halt.
- Politics may prevent data from moving across international boundaries thereby reducing the total availability of resources required for the computation.

Other attempts have been made to build a distributed mining framework as an application running over the grid middle layer thus extending the functionalities provided by the Grid. Du and Agarwal [31] have shown how such a framework can be used to analyze distributed data by a simple k-neighbor search. This framework also includes compiler and language support for building other data mining applications.

Grid Enabled Distributed Data Mining and Conversion of Unstructured Data [32] is yet another initiative to develop a framework for distributed data mining on the grid. A fuzzy based pattern matching technique is being developed to identify errors, both natural and deliberate in data from different sources. The architecture on which the distributed mining technique is built is based on the Open Grid Services Architecture (OGSA).

While the current focus of the community is to develop and extend a new framework entirely for distributed mining on the Grid, several generic tools and frameworks exist which have been build upon the underlying Grid infrastructure and provides developers with an easy interface to implement their applications on the Grid. The focus of this thesis is to make use of such easily available and easy to implement tools to build a distributed mining application and also to demonstrate the ability to adapt other existing mining applications to run on the Grid, thereby leveraging the features provided by the Grid and also utilizing the massive computational resources made available. In the next section we describe the job scheduling software and the task-farming paradigm that will be used for running the application on the Grid.

4.2. Condor

Condor [33] is a specialized workload management system for performing computationally intensive tasks. Among its many features, Condor provides a job-queuing mechanism, scheduling policy, priority scheme, resource monitoring and management. The success of Condor as a popular scheduling system is the fact that it is able to make use of the idle CPU cycles to run its jobs. Condor also provides a fault-tolerant system for long running jobs.

Condor is also integrated and leverages the use of Grid protocols [34] like Grid Security infrastructure (GSI) and Grid Resource Allocation and Management (GRAM). These protocols provide a secure and easy access to remote resources that are not available through simple network connections. The primary difficulty with accessing remote resources belonging to different sites originates from the following reasons:

- Heterogeneous resources, which require different mechanisms for authentication,

authorization, scheduling, hardware architecture, operating systems, etc.

- User is not aware of the characteristics of the remote resources.
- Keeping track of the status of the different elements of the computation involves tedious bookkeeping.

Condor-G [35], the grid enabled component of Condor, is able to submit jobs to remote Grid resources after authentication, and is able to monitor the job by tapping into the job management module on these resources. The Condor-G system addresses the above issues by separating the entire problem into three subsets:

- Accessing remote resources require that they speak standard protocols for resource discovery and management.
- Job management is handled by a robust workload management agent that is responsible for job submission, job monitoring and error recovering.
- Remote execution environment is provided by the use of a mobile sandboxing technology that is tailored for execution on a remote node.

Glidein is another important mechanism provided by Condor to extend job execution to the Grid. Glidein is a mechanism by which one or more Grid resources may temporarily join a local Condor pool. During the period of time when the added resource is part of the local pool it may be used to run jobs just like any other machine on the pool. A prerequisite for glidein is to have a valid user certificate which is accepted on the Grid resource. Glidein uses a security mechanism provided by Globus to authenticate the user who is requesting access to the Grid resource. Since most Grid resources employ some kind of queuing system, glidein also provides a way to specify the queue where these jobs

would be run. The advantage of using glidein is that any remote resource may be accessed by Condor to run the jobs thereby providing the notion of ubiquitous computing.

An important feature that is necessary because of the nature of this application is job dependency. Condor provides a very easily accessible feature whereby one job may start only after another job has been successfully completed and makes use of the output of the completed job as its input. Condor implements job dependency by the use of a directed acyclic graph (DAG).

A directed acyclic graph (DAG) [36] can be used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies. Condor finds machines for the execution of programs, but it does not schedule programs (jobs) based on dependencies. The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for Condor jobs. DAGMan submits jobs to Condor in an order represented by a DAG and processes the results. An input file defined prior to submission describes the DAG, and a Condor-submit-description file for each program in the DAG is used by Condor.

4.3. Task-farming

The application is to be designed in a way to make use of paradigms associated with grid computing like task farming. In task farming [37], the grid resources are divided as masters and workers. The role of the master resource is to drive the application by preparing the data, decomposing the problem into small tasks and distributing the tasks to the workers. These tasks are monitored and steered as required by the master process. Once these tasks are completed, more tasks can be assigned to the workers. The

master then collects the individual partial results from each worker to produce the final result of the computation. This is known as task farming. Task farming in a grid environment has to overcome several issues:

- i) Permission to create slave tasks on remote resources,
- ii) Authentication/Authorization on the remote resource to start and stop new tasks,
- iii) Management of queues on remote resources so that the slave tasks are completed in a suitable timeframe,
- iv) Application of firewall rules so that remote resources can communicate with other remote resource and the master process,
- v) Providing sufficient feedback mechanisms between the master and slave to reliably spawn new tasks, to divert new tasks in case a resource becomes unavailable and rollback changes in case of failure and continue runs to completion.

Several programming paradigms have been discovered and used to develop parallel programs which make use of distributed computing resources like Master-Worker, Single Program Multiple Data, Data Pipelining, Divide and Conquer and Speculative Parallelism. Of these the Master-Worker paradigm is especially attractive because it can be easily adapted to run on a Grid environment. The important characteristic of a Grid environment is the dynamic nature of resource availability. The Master-worker paradigm is highly suitable in this respect because communication between the master and workers happen only at the beginning and at the end of a task process. This means that master-worker applications show weak synchronization between

the masters and workers, so if and when a resource becomes available in the grid, it can be assigned as new tasks to participate in the application.

The MW [38] is a software framework developed to allow users to quickly and easily parallelize scientific computations using the master-worker paradigm on the grid. The MW provides abstract C++ classes for communication and resource management. These classes may be extended to incorporate any new Grid software toolkit. The advantage of using the MW application is that it provides a seamless integration with existing Grid infrastructure directly or indirectly through a very powerful distributed job scheduler system with which it is tightly integrated. The MW application is implemented as a set of 3 classes:

MWDriver: This class is responsible for reading in the data needed for the computation and breaking it down as input for each worker. The driver is also responsible for packing the results from the individual workers into the final global result for the computation.

MWTask: This class is responsible for communicating the individual input data to the workers and then retrieving the results from worker and passing them onto the driver.

MWWorker: This is where the actual task is executed. The worker unpacks the data that was received from the driver and performs the computation on it. The results are then sent back to the MWTask and request is made for further tasks if they exist.

The Cactus TaskFarmer[39] is another task-farming application that is build around Cactus, the scientific computation toolkit. It proposes a 3 level hierarchy as an alternate to the master-slave model as shown in figure 8:

Level 1: The Task Farm Manager (0), a.k.a. TFM (0), farms out tasks to remote resources on the Grid and was the Master in the traditional Master/Slave architecture.

Level 2: A Task Farm Manager (1), a.k.a. TFM (1), is started on a queue for each remote resource assigned a task.

Level 3: The specific computational task. This level corresponds to the Slave in the three-level model.

A Task Farm Manager (TFM) is a component that communicates with other TFMs to request for tasks, send feedback about the status of assigned tasks, and create parameter files to be passed for input, etc. The Task farm Manager uses two other generic modules which store information about the location of the executables, the minimum tasks that have to be assigned to a remote resource, how many tasks are required and requirements of the task viz., memory and processor requirements.

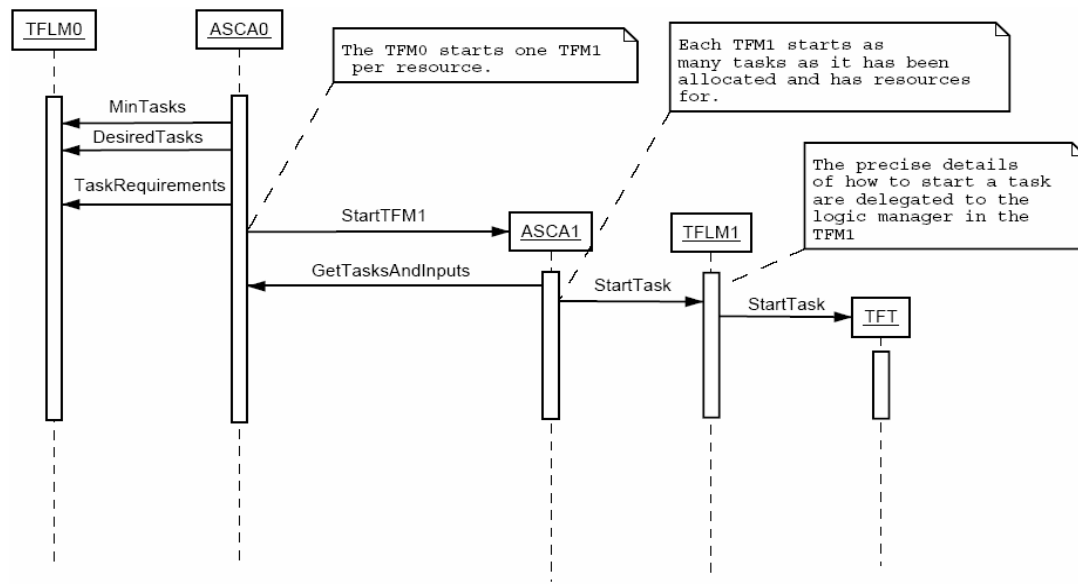


Figure 8: Task flow in the Cactus task farmer

AppLeS [40], Application-Level Scheduling system makes use of scheduling agents for parallel meta-computing applications. The agents are developed on a case-by-case basis and the user's application is mapped to these agents. The agent determines the schedules by considering the requirements of the application, and the predicted load and availability of resources at the time of scheduling. To monitor the performance and loads

on each resource, the system makes use of the Network Weather Service, a distributed system that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval. The service operates a distributed set of performance from which it gathers readings of the instantaneous conditions.

Netsolve [41] is an agent based client-server system, which provides the user with a set of APIs to solve complex scientific problems. One such API is a task-farmer, which provides an interface for farming applications that can be decomposed by a single bag of tasks. The implementation is limited to calling a function, which takes as argument the computation to be performed and the number of times the task needs to be performed. There is no feedback mechanism to aggregate the results of the individual tasks or spawn more tasks when the initially assigned task is completed.

In the next section, the core of the mining application is discussed. The data mining technique used here is fuzzy clustering based on vector time series.

5. Clustering of Vector Time Series

5.1. Vector Time Series

Time series data analysis is important in different areas like science, business, medicine, etc. This analysis can be very time consuming especially for vector time series. To reduce the computational burden some form of preprocessing is required to group similar time series into subsets and then analyzed. Swift *et al* [42] have performed a comprehensive study on a framework to reduce high-dimensional time series to low-dimensional time series, which are relatively independent of one another based on the correlation between the variables. A few selected works on clustering of time series data are reviewed in the following.

Kosmelj and Batalgelj [43] modified the *relocation clustering procedure* which was developed for static features, for the clustering of time varying data. To measure the dissimilarities between various series, a general model was first introduced by incorporating the time dimension. A specific model was then developed based on compound interest to determine the time dependent linear weights.

Owsley *et al.* [44] developed a method of automatic clustering of vector time sequences by generalizing a common vector clustering method, i.e., the generalized Lloyd Algorithm. It uses hidden Markov models to define the clusters and attempts to find a set of models which best describe the data.

Ramoni *et al.* [45] applied an agglomerative clustering procedure after replacing the vector time series by first order Markov chains represented by transition probability

matrices. It starts by assigning each set of transition matrices to a separate cluster and iteratively merges them till a stopping criterion is reached.

Liao [46] developed a two-step procedure for clustering continuously varying time series data based on partitioning. The first step applies a partitioning algorithm to the time stripped data in order to convert the multivariate time series into univariate time series. The univariate time series takes on discrete values regardless of the partitioning method. The second step uses another partitioning method to convert the univariate time series into a number of clusters. The distance between two discrete-valued univariate time series is expressed as transition probability matrices. Of all the partitioning methods available in literature the most commonly used is the fuzzy clustering method described in the next section.

5.2. Fuzzy Clustering

Fuzzy clustering [47] is a data mining technique based on the Fuzzy Set theory. In traditional set theory, a set is a crisp collection of objects, meaning the objects are either members of the set or not. In other words the membership of an object in a set is either 0 or 1. However, many things in reality are not so easily classifiable. The weather in a particular week, for example, may have days that were clear, days that were overcast or days that were a combination of the two. The fuzzy set theory assigns a membership value between 0 and 1 as to the likelihood of the object to be in a particular set. The more the membership value leans towards 1, the more is the likelihood of the object to be in that set. This concept is used in fuzzy clustering to determine clusters of data points that are likely to be part of a group, or exhibiting similar characteristics.

Fuzzy c-means clustering [48] is the most popularly used fuzzy clustering method today. This technique was originally introduced by Jim Bezdek in 1981. The main idea behind this method is the minimization of an objective function, which in normal cases is chosen to be the total distance between all patterns or groupings from their respective cluster centers. An iterative procedure is formulated to compute the solution and starts by choosing arbitrary initial cluster memberships or centers. The two main steps involved in this algorithm are;

- i) Distribution of the objects among the chosen clusters; and
- ii) Updating the cluster centers for the new distributions.

Each iteration of the algorithm alternates between these two steps until the value of the objective function cannot be reduced anymore.

To solve the fuzzy c-means model, the following algorithm has been developed. To run this procedure the number of clusters, c , and the weighting coefficient, m , must be specified. The algorithm has the following steps:

- 1) Choose c ($2 \leq c \leq n$), m ($1 < m < \infty$), and ε (a small number for stopping the iterative procedure). Set the counter $l = 0$ and initialize the membership matrix, $U^{(l)}$.

- 2) Calculate the cluster center, $v_i^{(l)}$ by using

$$v_i = \frac{\sum_{k=1}^n (\mu_{ik})^m x_k}{\sum_{k=1}^n (\mu_{ik})^m}, \quad i = 1, \dots, c.$$

- 3) Update the membership matrix $U^{(l+1)}$ by

$$\mu_{ik} = \frac{\left(\frac{1}{\|x_k - v_i\|^2} \right)^{1/(m-1)}}{\sum_{j=1}^c \left(\frac{1}{\|x_k - v_j\|^2} \right)^{1/(m-1)}}, \quad i = 1, \dots, c; k = 1, \dots, n.$$

using

if $x_k \neq v_i^{(l)}$. Otherwise, set $\mu_{jk} = 1$ (0) if $j = (\neq) i$.

- 4) Compute $\Delta = \| U^{(l-1)} - U^{(l)} \|$. If $\Delta > \epsilon$, increment l and go to Step 2. If $\Delta \leq \epsilon$, stop.

The data needs to be preprocessed before it can be used by the data-mining algorithm. The preprocessing techniques to be used in this application are linear, spline and cubic spline interpolation methods.

5.3. Interpolation

Linear interpolation [49] is the simplest method of getting values at positions in between the data points. The points are simply joined by straight-line segments. Each segment (bounded by two data points) can be interpolated independently. The parameter μ defines where to estimate the value on the interpolated line; it is 0 at the first point and 1 at the second point. For interpolated values between the two points μ ranges between 0 and 1. Values of μ outside this range result in extrapolation.

Spline interpolation [50] is a piecewise polynomial interpolation methods which connects points by means of smooth curves. Since not all points can be connected by a single polynomial function, piecewise interpolation curves are generated for different parts of the data and then combined to form a wavelet. In this case a generic quadratic polynomial would be used for the interpolation.

Cubic spline interpolation [51] is an extension of spline interpolation where the piecewise polynomial functions are cubic in nature. This leads to a better fit with the interpolated data. They produce a curve that appears to be seamless.

5.4. Cluster Validity Index

Determining the optimal number of clusters in unsupervised clustering is an important problem. Several validity indices are available in literature and a few of them are described in this section.

The PBM-index is defined as follows:

$$PBM(K) = \left(\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right)^2,$$

where K is the number of clusters. Here,

$$E_K = \sum_{k=1}^K E_k,$$

such that

$$E_k = \sum_{j=1}^n u_{kj} \|x_j - z_k\|$$

and

$$D_K = \max_{i,j=1}^K \|z_i - z_j\|.$$

n is the total number of points in the data set, $U(X) = [u_{kj}]_{K \times n}$ is a partition matrix for the data and z_k is the center of the k th cluster. The objective is to maximize this index in order to obtain the actual number of clusters.

The Davies – Bouldin index: This index is a function of the ratio of the sum of within-cluster scatter to between-cluster separation. The scatter within the i th cluster is computed as

$$S_{i,q} = \left(\frac{1}{|C_i|} \sum_{x \in C_i} \{\|x - z_i\|_2^q\} \right)^{1/q}$$

and the distance between cluster C_i and C_j is defined as

$$d_{ij,t} = \left\{ \sum_{s=1}^p |z_{is} - z_{js}|^t \right\}^{1/t} = \|z_i - z_j\|_t.$$

$S_{i,q}$ is the q th root of the q th moment of the points in cluster i with respect to their mean, and is a measure of the dispersion of the points in cluster i . Specifically, $S_{i,1}$, used in this article, is the average Euclidean distance of the vectors in class i to the centroid of class i . $d_{ij,t}$ is the Minkowski distance of order t between the centroids that characterize clusters i and j . Subsequently we compute

$$R_{i,qt} = \max_{j, j \neq i} \left\{ \frac{S_{i,q} + S_{j,q}}{d_{ij,t}} \right\}.$$

The Davies–Bouldin (DB) index is then defined as

$$DB = \frac{1}{K} \sum_{i=1}^K R_{i,qt}.$$

The objective is to minimize the DB index for achieving proper clustering.

The Xie – Beni index: This is a fuzzy clustering index. We mention it here briefly. The generalized version of this index is given by

$$S = \frac{J_m}{n * (d_{min})^2},$$

where J_m is the sum of squared errors objective function for fuzzy clustering and is given by

$$J_m(U, Z) = \sum_{j=1}^n \sum_{k=1}^K (u_{kj})^m \| \mathbf{x}_j - \mathbf{z}_k \|^2,$$

where $1 \leq m < \infty$. Here, U is a partition matrix, $U = [u_{kj}] \in R^{Kn}$. u_{kj} is interpreted to be the grade of membership of x_j in the k th cluster. Z is the set of cluster centers, i.e., $Z = \{z_k\} \in R^n$.

d_{min} is the minimum inter cluster distance. The minimum value of S in the hierarchy corresponds to the number of clusters presents in the data set.

6. Data Description

This section describes the data used for this research.

6.1. Model Generated Data

The Varmasim procedure of SAS was used to generate vector continuous time series data sets. Only bi-variate models were considered for ease of visualization. A bi-variate (K=2) stationary VARMA(1,1) time series was generated using the following formula.

$$y_t - \mu = \Phi(y_{t-1} - \mu) + \varepsilon_t - \Theta\varepsilon_{t-1}$$

where

$$\Phi = \begin{pmatrix} 1.2 & -0.5 \\ 0.6 & 0.3 \end{pmatrix}, \Theta = \begin{pmatrix} -0.6 & 0.3 \\ 0.3 & 0.6 \end{pmatrix}, \mu = \begin{pmatrix} 10 \\ 20 \end{pmatrix}, \Sigma = \begin{pmatrix} 1.0 & 0.5 \\ 0.5 & 1.25 \end{pmatrix}.$$

The stationary VMA(1), and VAR(1) series were generated by resetting the Φ and Θ variables of the above model to zero, respectively. Thirty series of 100 data points each from each of the above three models were generated. Figure 9 shows selected sampled generated from the three models.

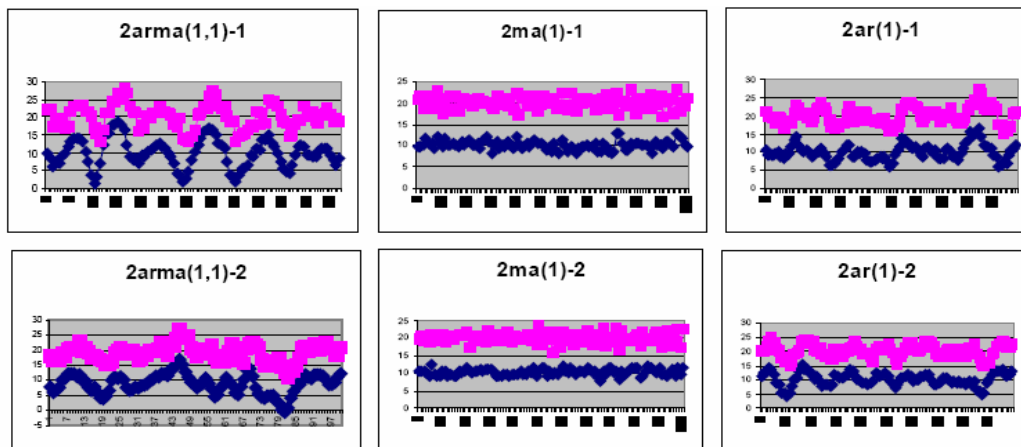


Figure 9: Visualization of model generated data

6.2. Battle Simulation Data

Battle Simulation time series was derived from data capture by running the OneSAF combat simulation software based on a battle scenario specially designed for the study [52]. The data was collected using a modified version of the Killer – Victim Scoreboard method, which was developed for collecting static feature data. The raw data collected were processed into time series by arranging them in the order of time stamps. It may be noted that since the collection mechanism used in this method is event triggered, the time stamps are not uniformly spaced. For each simulation run a total of five time series based on five indicators of battle states were obtained. They include:

- i. Relative territory Ownership
- ii. Relative firepower strength
- iii. Relative ammunition support
- iv. Relative fuel support
- v. Relative firing intensity

Each value of the time series is the relative indicator for the entire battle force because we are concerned with the development of an offense action plan. Random sample of eighty-five vector time series of varying lengths from eighty-five simulation runs is chosen. Figure 10 shows selected samples from some of the simulation runs.

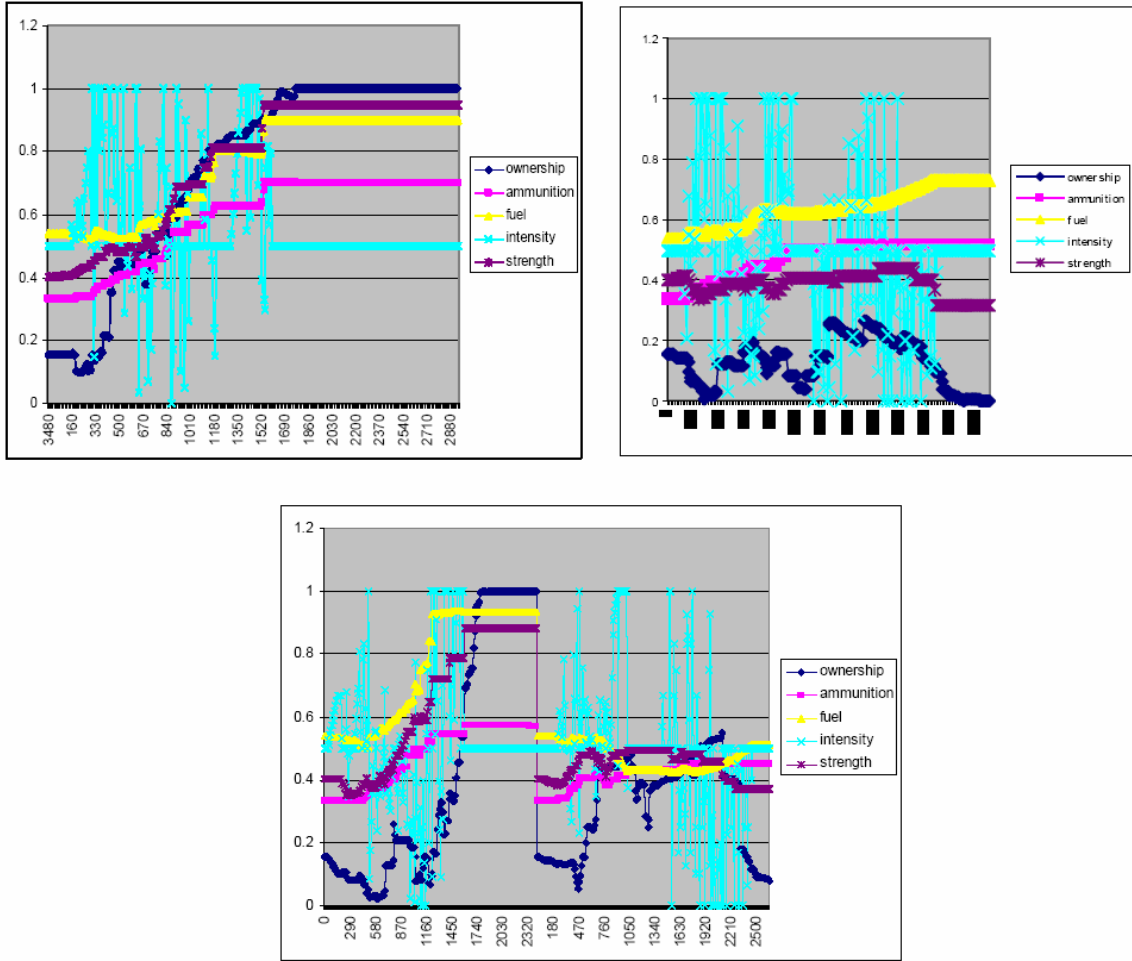


Figure 10: Visualization of battle simulation data

7. Description of Facilities

The infrastructure and resources for this research was provided by the Center for Computation and Technology, Louisiana State University.

These include

- Superhelix, a 256 processor BCVC cluster running Redhat linux enterprise edition.
- 6 dual processor Xeon work station running Redhat linux enterprise edition.
- The GUMBO Grid set of eight Pentium three class machines running Redhat linux enterprise edition.
- Condor installation is available on the GUMBO Grid and the workstations.
- Grid protocols necessary for accessing remote grid resources has been implemented in the globus tool kit, which is installed on the GUMBO Grid and Superhelix.
- Security certificates for authentication to the remote Grid resources are provided by NCSA and CCT.

8. Methodology

The two-step procedure proposed by Liao is modified to run on larger data sets and across distributed resources. The proposed methodology is then applied on the two different data sets described above. The data derived from the VARMA time series has profiles with equal length while data obtained from the battle simulation has profiles with unequal length. More importantly, the number of clusters and which cluster a series is in are known for the model-generated data, but not for battle simulation data.

The proposed procedure involves the following steps.

- Running the fuzzy clustering program to obtain discrete state outputs for converting multivariate real valued time series into univariate discrete time series.
- Compile the clustering results from individual works using the relational fuzzy c -means algorithm.
- For each univariate discrete time series compute the transition probability.
- Use the fuzzy clustering program to cluster the set of univariate discrete time series based on the transition probability.
- Compare the accuracy of the results with a serialized version of the same algorithm.

Fuzzy clustering of the initial input data is done using the MW task-farming framework. The large data file is passed as input to this program. The entire data set is split into smaller subsets based on the number of workers or machines available to run the clustering algorithm. Each individual resource would independently run the fuzzy clustering algorithm and classify the subset passed to it according to a pre-defined

validity index. For our procedure we use the PBM-Index which has been found to be the superior to other well known indices. It also passes the centers of the optimal clusters found by the clustering algorithm. Since the nature of the data is not known beforehand, and it has also been verified that the optimal number of clusters returned from each individual worker is different, some mechanism is required to normalize the optimal number of clusters across all workers.

In the second step, a relational fuzzy clustering algorithm based on Euclidean distances is used. This algorithm is modified from the Non-Euclidean Relational Fuzzy clustering (NERF) which was developed to identify clusters from the dissimilarity data for various groups. The dissimilarity is obtained as the relative distance between each group. This data is used to reduce or enhance the number of groups, so that groups whose dissimilarity is not significant are reduced to a single group without any loss in information. In this step, we obtain an optimal cluster number as the optimal cluster number that was returned by the maximum number of workers. Thus we use the distance of the centers from each worker to classify the centers into the optimal number of clusters. Once we know which cluster each center belongs to all corresponding time series would also belong to these clusters. The discrete state outputs are then relabeled according to the new cluster number obtained.

For each univariate discrete time series, the corresponding transition probability matrix is computed and serves as the input for measuring the distance between two time series. This is then passed on as input to another clustering program, which identifies the clusters or the state values for the original data set.

Since we are operating on extremely large volumes of data it is important that the whole procedure is as automated as possible. To achieve this Condor mechanism of Dagman is used to specify the dependencies between each individual steps in the procedure. The DAG representation for the proposal method, as required by the Condor is given below:

```
Job A submit_mwfile
Job B submit_fcmr
Job C submit_tpn
Job D submit_fcms
PARENT A CHILD B C D
PARENT B CHILD C D
PARENT C CHILD D
```

Each job corresponds to a step in the algorithm and the bottom half denotes the dependencies between the jobs. In this case Job A precedes Job B, which in turn precedes Job C. Job D is run after Jobs A,B and C are completed.

Data from the battle simulation has to be handled differently because they contain profiles with unequal length. In the proposed procedure this is taken care of by allowing the MW application to pass a specified number of time series to each individual worker. In the last step, the cluster number is compared to the original model from which the data was generated in the case of the model-generated data. The accuracy of this procedure is computed according to the number of time series whose final cluster value matches the model from which it came. Since we used three different models the clustering in the second step was required to give three clusters. In the case of the battle simulation data,

the clustering obtained at the end of the second step is to be analyzed to understand what each cluster in the output stands for. As a simplified case we propose a three-cluster output identified as:

- i. Case where “blue” (denoting the offensive force) won the simulation run.
- ii. Case where “blue” lost the simulation run.
- iii. Case where the result was a tie.

In this manner, simulation runs belonging to the same cluster may be identified as victory or loss conditions.

The entire procedure may be summarized in the following algorithm:

- i. Process input file by arranging the data according to their time stamps and normalizing it to a scale of one using any of the interpolation methods described above (linear is used in this research). The first column of the input file denotes the time stamp, which may be in a predefined uniform scale of tens or hundreds. The next five columns show the relative values of the five battle indicators. In the case of the model generated data the preprocessing stage involves combining the time series data of all individual runs. The first column in this case also denotes the time stamp while the next two columns denote the two variables on which the models are based.
- ii. The MW framework is used to build a form of the fuzzy clustering algorithm. The framework provides three separate classes, one for reading in the data and setting up the initial tasks, one for communicating between the master and the workers, and the last one for executing the clustering programs on the individual workers. The driver class splits the input data into a specified number of subsets, which are

- based on the number of tasks and the number of workers available to the application. It passes pointers to the start and end row of each subset along with the task number and the number of series contained in the subset. The task class provides mechanism for packing this data from the driver and unpacking it at each individual worker. After the clustering algorithm is run on the data at each individual worker the results in the form of cluster (state) values and cluster centers are packed by the task class and unpacked on the master. The driver then collects these individual outputs and returns two output files, which are required as inputs for the next step.
- iii. The relational fuzzy clustering algorithm is used to reduce higher number of clusters into equivalent low numbered clusters. Since the outputs from the individual workers are not uniform this method is employed to reduce the number of clusters for the entire dataset into a single set of values. The cluster centers obtained from the previous step are used to re-label the state values for the univariate time series.
 - iv. A simple program to compute the transition matrices of the univariate discrete time series is run on the newly labeled data set. The matrices corresponding to each run is arranged as a linear vector with $n \times n$ data points, with n corresponds to the optimal number of clusters obtained at the end of the previous set.
 - v. The transition probability vectors correspond to the original data sets they are of uniform length and hence the first step may be reapplied on this data set, this time on a single workstation because the size of the data set has been greatly reduced in the previous two steps. In this step we decide the optimal number of clusters to

be three in both cases. Each run corresponds to a state variable, which identifies the model it came from or in the case of the battle simulation data one of the three states: win, loss, or tie.

- vi. To compute the accuracy of the proposed model the cluster value obtained is compared to the model from which the run came from, if known. Otherwise, validation is done visually.

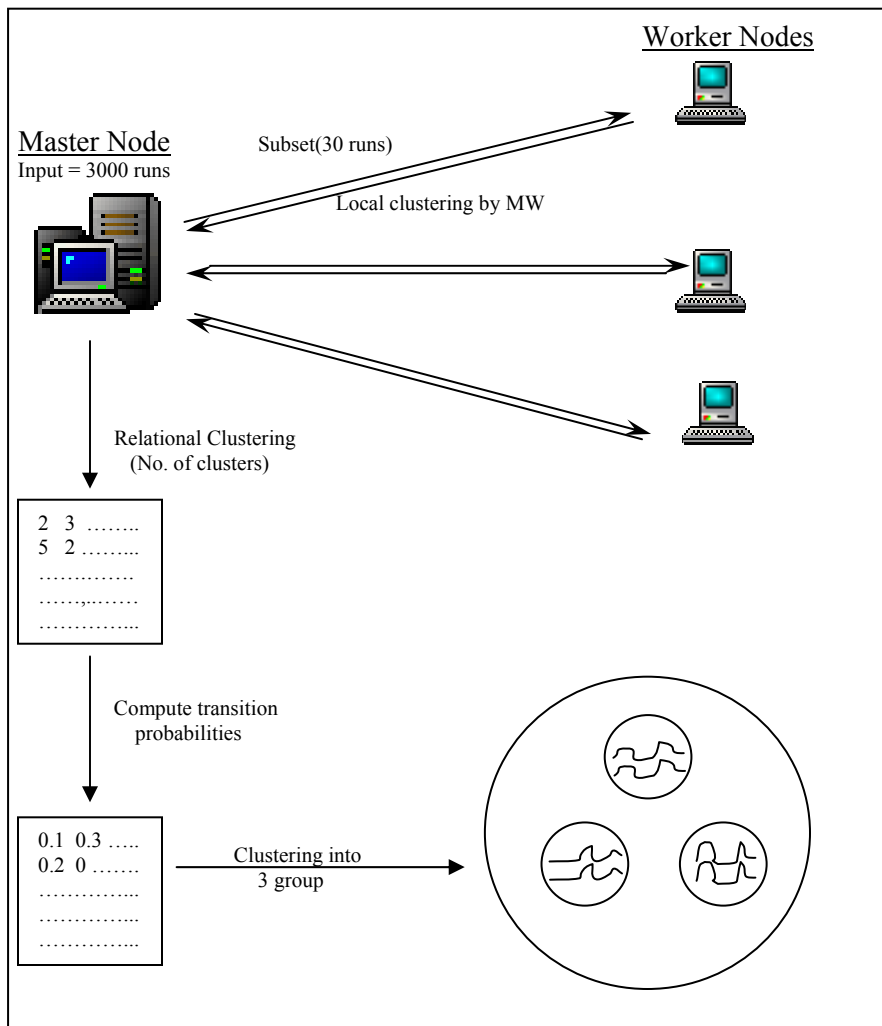


Figure 11: Graphical representation of the proposed methodology

9. Results and Discussion

9.1. Results

The results of the model-generated data are presented first, for which the ground truth is known. The accuracy and time taken for completion of the entire program were computed and plotted against the number of series used in each execution. In each execution of the program, the following parameters were varied:

- i. Number of series in the input file
- ii. Number of optimal clusters in the first step
- iii. Number of workers available for the first step
- iv. Number of series sent to each worker.

Tables 2 and 3 list the performances of the distributed clustering algorithm and single-station clustering for using different numbers of clusters in Step 1, and different numbers of workers, respectively.

Similar results of using different numbers of series are given in the Appendix as Tables 4-7.

Table 2: No. of optimal clusters vs. accuracy for 3000 series

No. of Clusters in Step 1	Distributed Clustering on the Grid			Single Station Run	
	Accuracy	No of workers	Time Taken (minutes)	Accuracy	Time Taken
2	64.8	45	26	84.6	246
3	65.6	45	26	82.7	277
4	47.4	45	28	65.3	257
5	72.6	45	26	69.6	291
6	67.2	45	27	81.2	315
7	77.5	45	26	76.9	312
8	74.2	45	28	77.6	307
9	69.3	45	27	68.3	294

Table 3: No. of workers vs. accuracy and time taken

No. of Clusters in Step 1	No. of Workers	Accuracy	Time Taken
7	1	79.3	287
7	5	67.6	114
7	10	76.7	78
7	15	72.2	56
7	20	76.9	48
7	25	78.2	44
7	30	66.3	37
7	35	76.5	31
7	40	67.3	19
7	45	76.4	13

The above results are used to plot the performance of the algorithm as a relation between the number of optimal clusters in the first step to the accuracy of the model (Figure 13), relation between the number of workers used in the first step and the accuracy of the model (Figure 15). To study the effects of using distributed resources to run the whole program, a comparison with the original method, which was run on a single workstation is used. The original method was slightly modified to automate the entire procedure so it can handle any size of data. Relation between the time taken for the execution of the program and the number of optimal clusters is also plotted to find whether a specific number may improve the speed of the algorithm (Figure 14).

Similarly, the effect of number of workers in the time of execution is plotted in Figure 16.

The intermediate results for the model generated data after the univariate time series is presented in Figure 12. Each series corresponds to the bivariate time series shown in Figure 9.

In the case of the battle simulation data, there are no ground rules to decide the accuracy of the proposed model. Hence we rely on a visual inspection of the

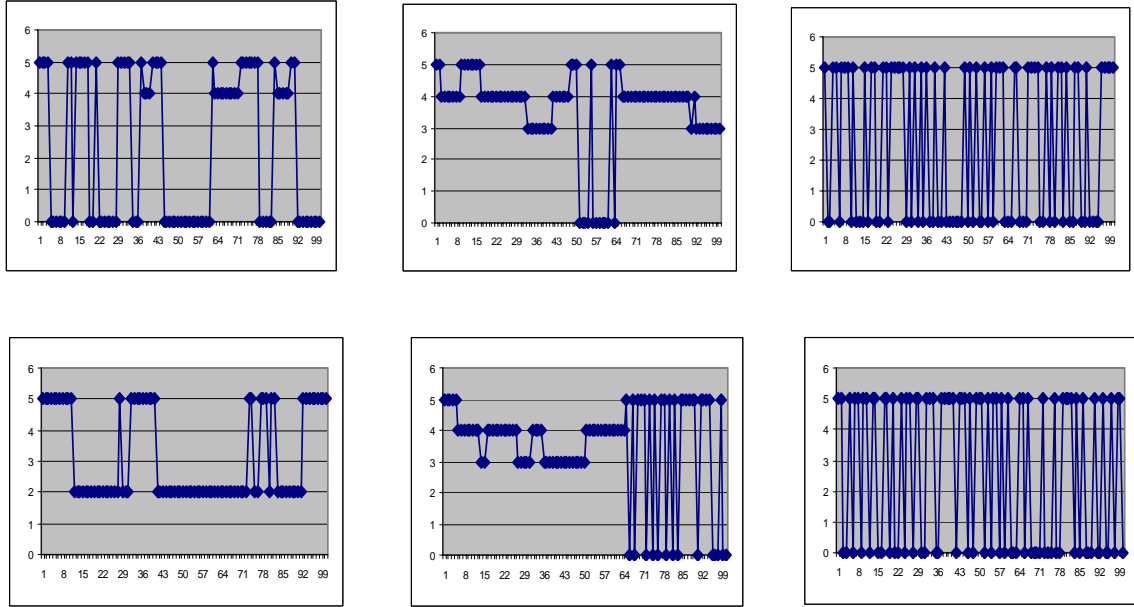


Figure 12: Univariate series for the model generated data

clustering results to decide if the classification of the data by the model is reliable.

For the execution of the program on the battle simulation data, the following parameters were varied:

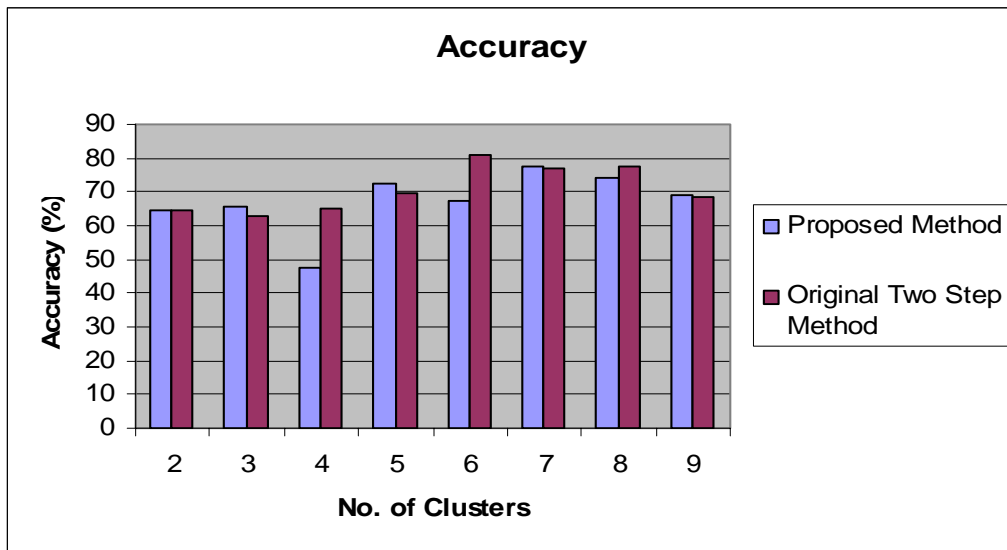


Figure 13: Accuracy vs. number of clusters for proposed and original methods

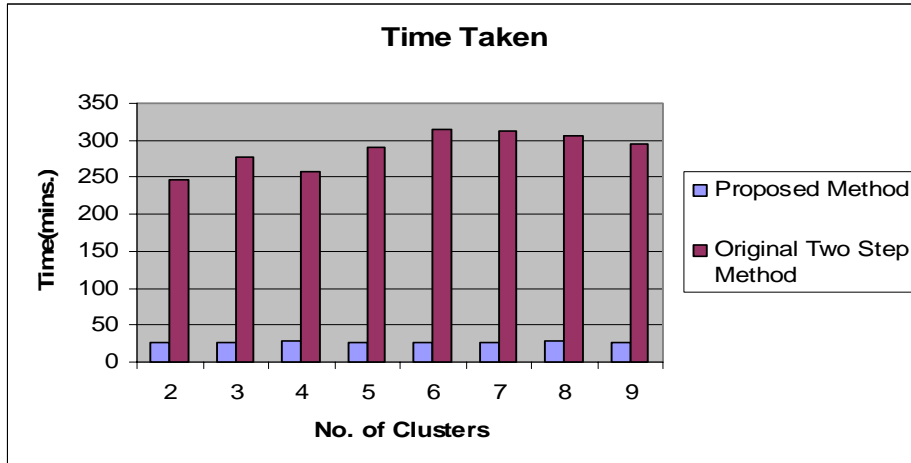


Figure 14: Time of execution vs. number of clusters for proposed and original methods

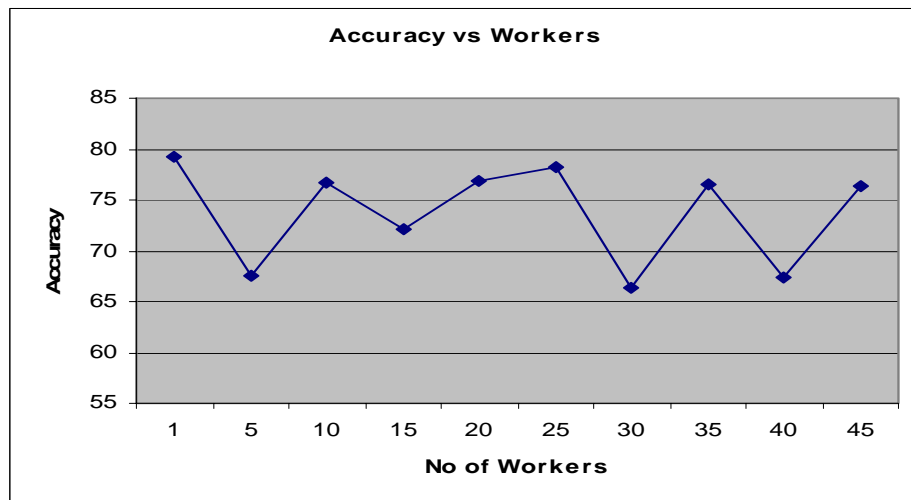


Figure 15: Accuracy vs. number of workers for proposed and original methods

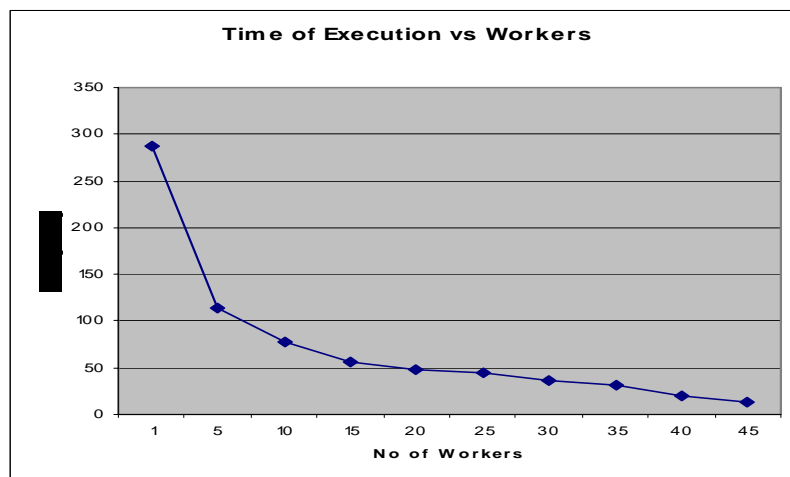


Figure 16: Time vs. number of workers for proposed and original methods

- i. Number of series in the input file
- ii. Number of optimal clusters in the first step
- iii. Number of workers available for the first step

The main differences between running the program for the battle simulation data from the model generated data may be summarized as follows:

- i. Since each run is of a different length, we need to capture the total data points in each run and also the total number of rows being sent to each worker.
- ii. To keep track of number of runs in each run to each worker, another array is introduced.
- iii. The resultant array size is not predetermined and has to be dynamically set according to the data returned from each worker.

The univariate time series obtained at the end of the relational clustering procedure is reliable. We plot individual time series, which are classified as belonging to the same cluster against time. In the following figures we show these results for each of the three clusters, which may be classified as “win”, ”loss” and “tie” clusters.

By visually inspecting the above clusters, we notice that in figures 17, 18 and 19; most of series in each cluster seem to match a pattern as the data points in the series make transitions between the different state values. It may be observed that while some series in

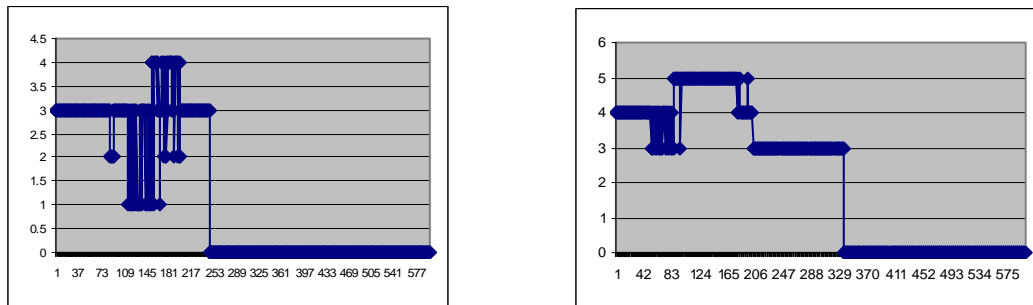


Figure 17: The “win” cluster

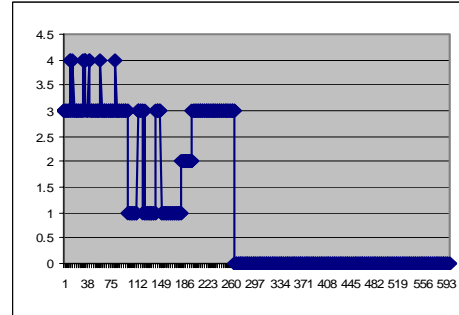
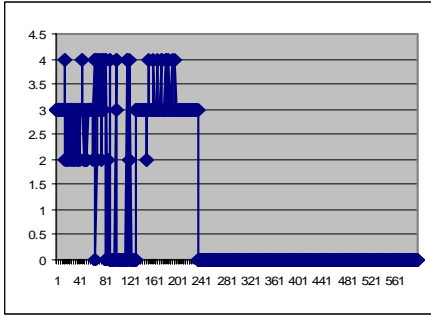


Figure 18: The “tie” cluster

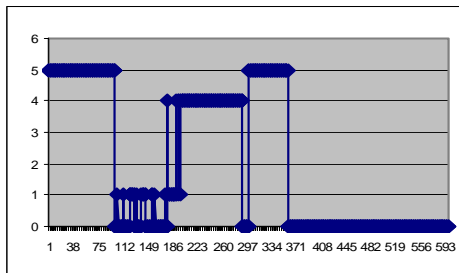
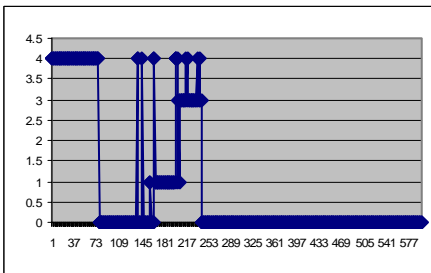


Figure 19: The “loss” cluster

each cluster may appear different from the rest, they still don't match the patterns, which describe the other two clusters. For example in the “tie” cluster, one series does not match the transition patterns shown by the remaining four but a visual inspection suggests that it does not match the dominant patterns in the “win” cluster and the “loss” cluster. These series may be termed as outliers as they don't fit into any of the three clusters.

Misclassified series are those on visual inspection appear to belong to a different cluster than the one it was clustered by the algorithm. It may be argued that two of the series in the first “loss” cluster actually belong to the “win” cluster and this may be used to determine the accuracy of the model.

However, assigning crisp accuracy values based on the above-described visual inspection is not entirely a reliable procedure until we have some information about the outcome of each series. A similar conclusion was reached in the original two-step procedure and this suggests that the proposed method does not lose any additional

information by using the distributed clustering and recombining the results. This is an encouraging sign as to the reliability of the proposed technique viz., the original two step procedure and every argument validating the use of the two-step procedure applies to this proposed method.

9.2. Discussion

Tables 3 and 4 summarize the performance of the algorithm when an input file of 3000 bivariate time series is used. By varying the number of clusters used in the first step, we test the performance of the algorithm with respect to accuracy and time of execution. We know that the number of optimal clusters at the end of the algorithm has to be 3 in the case of the model generated data, and since we assume three possible outcomes for the battle simulation data (win-loss-tie) the optimal number of clusters in this case is also 3. However using any of the validity indices tested in the original procedure yields an optimal number in both cases to be different from 3. Hence to maintain accuracy, we define optimal number of clusters to be 3 and then classify each individual run into these three clusters.

It may be observed from figure 13 that, the proposed method comes quite close to the original procedure in terms of accuracy. The splitting of data and running local clustering algorithms and then combining them using a new optimal cluster number obtained by a relational clustering based on the distances of the respective cluster centers, does not seem to reduce the performance of the overall procedure. This suggests that the distribution and combining methodology proposed here match the profiles of the input data. However this result may be application specific and other time series data may not behave in the same manner.

It may also be seen that the accuracy of the proposed model is highest when the optimal number of clusters is chosen to be 7; this indicates that with higher cluster number more information characteristic of the input data is captured by the model thus resulting in higher accuracy. In the case of the original procedure also this seems to hold thereby proving the hypothesis.

From the figures 14 and 16, it is obvious that using a distributed mechanism to run local clustering algorithms on subsets of the data on each individual worker and then combining them is much faster than running on single machines. To avoid the performance differences due to faster machines or network bandwidth, both the single machine procedure and proposed distributed procedure were run on the same physical computing resource. The multiprocessor cluster allows each individual worker to run in parallel without the need to wait for other workers to complete or begin execution. This allows the system to truly reflect the performance of the proposed method in terms of speed of execution.

An important result that we expected to observe is the effect of varying the number of workers on the accuracy, thereby deciding an optimal number of workers necessary for the program. However the nature of the MW framework is such that irrespective of the total number of workers, jobs are scheduled on the workers in batches of a predefined number. This is an inherent defect in the MW code, as it is not able to scale according to the dynamic availability of workers. Hence, it was observed that even if workers were available, only a predefined number were loaded with jobs while the rest remained in idle state. This problem was only overcome by specifying a number much higher than the actual number of workers available through Condor.

Most Grid resources use some form of queuing system and glidein is the mechanism by which these machines may be told execute jobs in a specified queue. However, we observed that when the queues were already loaded, the number of workers reported by Condor were not all available for jobs to run. This caused the MW framework to run into execution errors resulting in core dumps. From the above two issues it is obvious that a tradeoff is required as to the optimal number of workers necessary to run this program.

It was also observed that the time taken for execution was not directly proportional to the number of workers. This may be explained by the internal mechanisms by which Condor negotiates between the master and worker.

When the grid resources are loaded by external jobs, the workers go into the queues on the resource this also affects the running time of the proposed algorithm. The most obvious solution here is to use a dedicated Grid resource or to run the algorithm on the local Condor pool without relying on external resources. For this the local Condor pool must consist of a significant number of machines corresponding to the number of workers being chosen in the MW application and the pool is supported by high bandwidth network for the data movement.

10. Conclusions and Future Direction

We propose in this research a fully automated method to cluster multivariate time series data by distributing the input data among distributed computing resources and then collecting and combining the results from each of them. The method uses frameworks that are available to develop applications for the Grid. The entire application was built by extending the original procedure, which runs on a single machine. While core of the application remains the same, mechanisms to read in and transport the input data has to be redesigned using the MW framework.

There are several advantages to using a Grid framework to perform the data distribution and execution. Following are some of key points that were observed while running this new method:

- iv. Important system level functions like communication, resource handling and parallel programming mechanisms are handled by the framework.
- v. Unlike many other distributed mining algorithms, this new approach does not require a dedicated set of resources to run on.
- vi. Different data sets obtained from similar multivariate time series may be applied to this algorithm with a little modification in terms on structure and format.
- vii. Fuzzy clustering was used for the basic partitioning but could be replaced by any other partitioning technique available in the literature. It would require the least amount of coding to incorporate any other partitioning method into this proposed method.

- viii. The workers used in this application may be heterogeneous in nature. Condor automatically will discover the best resources to send the jobs so that faster machines would be loaded with more jobs.

This research work started as a way to automate the clustering of multivariate time series data so that large sized data sets may be clustered. Current research work suggested that the optimal way to work on large data sets and achieve significant gains in terms of performance and accuracy was to use some form of distributed mechanism. Knowledge of the MW framework and other task arming approaches allowed us to take advantage of the Grid framework to extend our application to the Grid. However as we observed, there is sufficient scope for work to be done to improve the quality of results obtained from this new procedure. Some of the things, which may be taken up as future work are summarized below:

- i. MW is very closely tied in with Condor system. So it is necessary to explore possibilities of integrating with other task farming solutions and grid frameworks like GAT, APPLES. This way the performance of each framework may be compared and the most optimal one suited to this application may be chosen.
- ii. Real world data like the battle simulation data may not have information pertaining to the optimal cluster number. Hence it is based on the user's experience to decide what the optimal cluster must be. Several validity indices are available in literature but it was observed that none of them were suited to this data set. Hence it is necessary to decide a method to determine the optimal number of clusters based on previous knowledge.

References

1. *Cancer Statistics for 2004*, American Cancer Society, 2004 (Link: http://www.cancer.org/docroot/STT/stt_0.asp)
2. *The US Defense Budget*, Thomas Crosby media, 2004
3. *Data Mining*, J.A. Bunge and D.H. Judson, Cornell University and US Census Bureau
4. *On the Convergence of Reinforcement Procedures in Simple Perceptrons*, Rosenblatt, F., Technical Report VG-1196-G-4, Cornell Aeronautical Laboratory Report, Buffalo, NY, 1960.
5. *Adaptive Switching Circuits*, Widrow B., Hoff, M.E., IRE WESCON Convention Record, part 4, pp 96-104, 1960.
6. *An Interactive Activation Model of the Effect of Context in Perception: Part 1*, McClelland, J. L. and Rumelhart, D. E., Psychological Review, vol. 88, pp 375-405, 1981.
7. *An Empirical Study of Learning Speed in Back-propagation Networks*, Fahlmann, CMU-CS-88-162, 1988.
8. *Increased Rates of Convergence Through Learning Rate Adaptations*, Jacobs, R.A., Neural Networks, Vol. 1, No. 4, pp 295-307, 1988.
9. *Multivariable Functional Interpolation and Adaptive Networks*, Broomhead, D.S. and Lowe D, Complex Systems, vol. 2, pp 321-355, 1988.
10. *Analysis of a Simple Self-organizing Process*, Kohonen, T., Biological Cybernetics Vol. 44, pp135-140, 1982
11. *Induction of Decision Trees*, Quinlan, J. R., Machine Learning, Vol. 1 , pp81-106, 1986.
12. *C4.5: Programs for Machine Learning*, Quinlan, J. R., Morgan Kaufmann, San Mateo CA, 1993.
13. *Classification and Regression Trees*, Breiman L., Friedman, J.H., Olshen, R.A., Stone, C.J., Wadsworth & Brooks Pacific Grove, CA, 1984
14. *The CN2 Induction Algorithm*, Clark, P. and Niblett, T., Machine Learning, vol. 3, pp 261-283, 1989

15. *Efficient Pruning Methods for Separate-and-conquer Rule Learning Systems*, Cohen W.W., In Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp 988-994, Chambéry France, 1993
16. *Agent-based Distributed Data Mining: The KDEC Scheme*, M. Klusch, S. Lodi, G. Moro, Intelligent Information Agents, LNAI 2586, pages 104-122, 2003.
17. *A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms*, IEEE Trans. PAMI, PAMI-2(1), 1-8. Reprinted in: *Fuzzy Models for Pattern Recognition*, ed. J.C. Bezdek and S.K. Pal, IEEE Press, Piscataway, NJ, 1993, 130-137.
18. *A fuzzy C-Means Variant for the Generation of Fuzzy Term Sets*, Liao, T.W., A.K. Celmins, and R.J. Hammell II, *Fuzzy Sets and Systems*, 135(2),2003, 241-257.
19. *The Greenhouse Project Page on Linear Interpolation*. A project of the National Science Foundation Center for Graphics and Scientific Visualization Link: <http://www.cs.brown.edu/stc/outrea/greenhouse/nursery/interpolation/>)
20. *B-Spline Interpolation of Data with Regular Sampling*, P. Thevenaz, Swiss Federal Institute of Technology Lausanne, Mathematical notebook, 2000. (Link: <http://library.wolfram.com/infocenter/MathSource/1055/>)
21. *The Cactus Framework and Toolkit: Design and Applications*, T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E.Siedel, J. Shalf, *Vector and Parallel Processing- VECPAR 2002, 5th International Conference*, Lecture notes in Computer Science, Springer(2003).
22. *Mining of Vector Time Series by Clustering*, W. Liao, Submitted for publication.
23. *Chapter on Cubic Spline Interpolation*, A Cuyt and L. Wuytack, *Non-linear methods in Numerical Analysis*, Amsterdam: North Holland, 1987.
24. *Globus: A Metacomputing Infrastructure Toolkit*, Ian Foster and Carl Kesselman, *International Journal of Supercomputer Applications*, 11(2):115-128,1997.
25. *The Grid: Blueprint for a New Computing Infrastructure*. Ian Foster and Carl Kesselman editors, Morgan Kaufmann Press, 1998.
26. *Condor-G: A Computation Management Agent for Multi-institutional Grids*, James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC, Pages 7-9)*, San Francisco California August 2001.
27. *An Enabling Framework for Master-worker Applications on the Computational Grid*, Jeff Linderoth, Sanjeev Kulkarni, Jean-Pierre Goux and Michael Yoder, In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43-50, Pittsburg, Pennsylvania, August 2000.

28. *Resource Management Through Multilateral Matchmaking*, Rajesh Raman, Miron Livny and Marvin Solomon, In Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), pages 290-291, Pittsburg PA, August 2000.
29. *Condor- A Distributed Job Scheduler*, Todd Tannenbaum, Derek Wright, Karen Miller and Miron Livny, In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001
30. *Condor Manual*, Condor Team, Available from <http://www.cs.wisc.edu/condor/manual>, 2001
31. *Metacomputing and the Master-Worker Paradigm*, Jeff Linderoth, Jean-Pierre Goux, and Michael Yoder, Preprint ANL/MCS-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory, February 2000.
32. *Condor and the Grid*, Douglas Thain, Todd Tannenbaum, and Miron Livny, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003. ISBN: 0-470-85319-0
33. *Jam: Java Agents for Meta-learning over Distributed Databases*, Stolfo S.J, Prodromidis A.L, Tselepis S, Lee W, Fan D. W, Chan P.K, Proceedings of the third international conference on knowledge discovery and data mining, pages 74-81, AAAI Press, Aug 1997.
34. *Papyrus: A System for Data Mining over Local and Wide Area Clusters and Super Clusters*, Bailey S.M, Grossman R.L, Sivakumar H, Turinsky A.L, Technical report, University at Illinios at Chicago.
35. *Scalable, Distributed Data mining An Agent Based Application*, Kargupta H, Hamzaoglu I, Stafford B, Proceedings of knowledge discovery and Data mining, Aug 1997.
36. *Collective Data mining: A New Perspective Towards Distributed Data Mining*, Kargupta, H, Park B, Hershberger D, and Johnson E, *Advances in Distributed and Parallel knowledge discovery*, MIT/AAAI Press, 1999.
37. *An Architecture for Distributed Enterprise Data Mining*, Chattratichat J, Darlington J, Guo Y, Hedvall S, Kohler M, and Syed J, HPCN, Amsterdam, 1999.
38. *Advances in Knowledge Discovery and Data Mining*, Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R, AAAI Press/MIT Press 1996.
39. *The Knowledge Grid*, M. Cannataro, D. Talia, *Communications of the ACM*, 46(1), 89-93, 2003.

40. *A Data Mining Toolset for Distributed High-Performance Platforms*, M. Cannataro, A. Congiusta, D. Talia, P. Trunfio, Proc. 3rd int. Conference Data mining 2002, WIT Press, Bologna, Italy, pp 41-50, September 2002.
41. *Mining Large Data Sets on Grids: Issues and Prospects*, Talia D, Skillicorn D, Computing and Informatics, pages 347-362, Vol. 21 2002.
42. *A Resource Management Architecture for Metacomputing Systems*, K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.
43. *Security for Grid Services*, V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, to appear June 2003.
44. *A Security Architecture for Computational Grids*, I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.
45. *Grid Resource Management*, J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds). Kluwer Publishing, Fall 2003.
46. *Communication Services for Advanced Network Applications*, J. Bresnahan, I. Foster, J. Insley, S. Tuecke, B. Toonen. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications 1999, Las Vegas, Nevada, June 28-July1, 1999. Volume IV, pp1861-1867.
47. *What Size Net Gives Valid Generalization?* , Baum, E. and Haussler, D., Neural Computation vol. 1, No.1, pp 151-160, 1989.
48. *Developing Distributed Data Mining Implementations for the Grid Environment*, Wei Du and Agarwal, G., Proceedings of the 2nd IEEE/ACM Symposium on Cluster Computing and the Grid, 2002.
49. *Identifying Battlefield Metrics Through Experimentation*, Heilman, E., Proc 7th International Command and Control research and technology symposium, 2002.
50. *Cross-sectional Approach for Clustering Time Varying Data*, Kosmelj, J. and Batamelj, V., Journal of Classification, 7, 99-109, 1990
51. *Undertanding and Projecting the Battle State*, Liao, T.W., Bolt, B., Forester, J., Heilman, E., Hansen, C., Kaster, R.C., and O'May, J., 23rd Army Science Conference, December 2-5, 2002.

Appendix: Supplementary Data

Time taken and accuracy measurements for the different number of series that were tested for the model generated data

Table 4: No. of optimal clusters vs. accuracy for 2850 series

No. of Runs	2850					
No. of Clusters	Accuracy	No of workers	Time Taken	Single Station Run	Accuracy	Time Taken
2	67.3	45	25		83.4	238
3	75.3	45	22		81.4	244
4	61.3	45	26		78.5	236
5	74.2	45	27		81.6	247
6	69.1	45	28		83.4	277
7	72.6	45	24		78.5	248
8	73.2	45	31		83.4	265
9	77.6	45	33		81.4	237

Table 5: No. of optimal clusters vs. accuracy for 2700 series

No. of Runs	2700					
No. of Clusters	Accuracy	No of workers	Time Taken	Single Station Run	Accuracy	Time Taken
2	66.4	45	21		84.1	247
3	64.3	45	24		77.4	278
4	68.4	45	25		74.4	235
5	71.1	45	24		67.3	227
6	78.9	45	22		63.3	257
7	66.7	45	26		78.4	243
8	68.3	45	24		81.4	236
9	74.7	45	27		79.1	228

Table 6: No. of optimal clusters vs. accuracy for 2550 series

No. of Runs	2550					
No. of Clusters	Accuracy	No of workers	Time Taken	Single Station Run	Accuracy	Time Taken
2	61.3	45	23		74.2	266
3	57.8	45	25		65.3	214
4	68.9	45	19		83.4	232
5	76.4	45	25		63.8	247
6	74.3	45	23		78.6	244
7	82.1	45	21		79.3	226
8	66.5	45	26		81.4	235
9	72.4	45	24		70.3	237

Table 7: No. of optimal clusters vs. accuracy for 2400 runs

No. of Runs	2400					
No. of Clusters	Accuracy	No of workers	Time Taken	Single Station Run	Accuracy	Time Taken
2	68.9	45	25		76.3	215
3	66.3	45	22		72.4	208
4	71.3	45	23		77.1	217
5	78.4	45	26		75.4	204
6	74.4	45	21		79.1	197
7	61.3	45	24		76.5	235
8	68.1	45	22		70.2	212
9	72.3	45	25		67.8	216

Vita

Arun Nayar received his Bachelor of Technology degree in mechanical engineering from the College of Engineering, University of Kerala in 1999. He joined the graduate program in industrial engineering at Louisiana State University in the United States in August 2002. He worked towards his Master of Science in Industrial Engineering degree under the guidance of Professor Warren Liao. He is a co-author of the publication titled “Of Death, Taxes and the Design of a Task Farming Architecture for Grid Computing without sacrificing Application Design” which has been submitted to the Journal of Association for Computing Machinery for review.