Priority Based Cloud Scheduling Using Analytical Hierarchy Process With Multi-Layer Perceptron

Tribhuvan University
Institute of Science and Technology

Dissertation

Submitted To
Central Department of Computer Science and Information Technology
University Campus, Tribhuvan University, Kirtipur, Kathmandu, Nepal

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science and Information Technology

Submitted By
Santosh Ghimire
Roll No: 03 (2012-2014)

Under the Supervision of
Mr. Jagdish Bhatta

December, 2015

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

Santosh Ghimire

Date: December 29, 2015

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Santosh Ghimire** entitled **Priority Based Cloud Scheduling using Analytical Hierarchy Process with Multi-Layer Perceptron** be accepted as in fulfilling partial requirements for the completion of Master's Degree of Science in Computer Science and Information Technology. This is an original work in Computer Science to best of my knowledge.

Mr. Jagdish Bhatta
University Campus,
Central Department of Computer Science and Information Technology,
Institute of Science and Technology,
Kirtipur, Kathmandu, Nepal.

(Supervisor)

Date: December 29, 2015

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Letter of Approval

We certify that we have read this dissertation work and in our opinion it is appreciable, fully adequate and satisfactory, in scope and quality, as a dissertation in the partial fulfillment of the requirements of Master's Degree of Science in Computer Science and Information Technology.

Evaluation Committee

Mr. Nawaraj Poudel
Head of Department
Central Department of Computer Science and Information Technology
University Campus, Kirtipur
Tribhuvan University, Nepal

Mr. Jagdish Bhatta
Supervisor
Central Department of Computer Science and Information Technology
University Campus, Kirtipur
Tribhuvan University, Nepal

(External Examiner)

(Internal Examiner)

ACKNOWLEDGEMENT

First of all, I want to express my sincere gratitude to my respected teacher as well as my dissertation supervisor, **Mr. Jagdish Bhatta**, Lecturer, Central Department of Computer Science and Information Technology (CDCSIT), Tribhuvan University for his wholehearted cooperation, encouragement and strong guidelines throughout the preparation of this work. With his enthusiastic presentation of new problems and ideas of possible solutions, he always managed to provide me with the necessary motivation. With this regard I wish to extend my genuine appreciation to respected Head of Central Department of Computer Science and Information Technology, **Mr. Nawaraj Poudel** for his kind help, encouragement and constructive suggestions.

I consider my pleasant duty to express my sincere gratitude to all the people who supported and encouraged me involving directly or indirectly to complete this thesis. I am also obliged to all respected teachers and staffs of CDCSIT for their willing co-operation to bring this thesis work in tangible form.

I have given my best effort to make this thesis work complete and error free. However, I am always looking forward to the suggestions from the readers which will improve this work.

I am especially beholden to **TU-CAS-Thesis Grants Committee** and **Chinese Academy of Sciences (CAS)** for conceding my thesis work and providing me **TU-CAS-Thesis Grants (212/072/073)** as well as bestowing me wonderful opportunity to undergo every academic routes and accomplish this dissertation.

Santosh Ghimire
Roll No: 03 (2012-2014)
(CDCSIT, TU)
December 29, 2015

ABSTRACT

Priority of jobs is an important issue in scheduling because some jobs should be serviced earlier than other those jobs can't stay for a long time in a system. Assigning priorities also helps to preempt task in the middle of execution. Since priorities often vary according to users with respect to various job scheduling parameters in cloud such as bandwidth, completion time, memory, etc., often a model that makes decision to schedule with respect to multi-criteria is relevant. Analytical Hierarchy Process (AHP) is a multi-criteria, multi-attribute decision making model that combines the criteria weights and the options scores, thus determining a global score for each option, and a consequent ranking. Using AHP, the decision making scheduling problem can be decomposed into hierarchy of comprehended sub problems, each of which can be analyzed independently. However, different users can have conflicting preferences. So, there might be inconsistent elements in the reciprocal pairwise comparison matrix created on the process of solving scheduling problem by using AHP. In order to solve matrix inconsistency problem as well as to tackle the situation of missing entries in the comparison matrix which must be filled in accordance to decision maker's judgments, a Multi-Layer Perceptron neural network model can be used to handle such scenarios after training them. Final result is the priority based schedule of tasks that can be scheduled in the datacenter with fixed resources which is typically suggested by neural network in course of deriving a PVS vector.

Categories and Subject Descriptors:

• Networks~Cloud computing • Computing methodologies~Neural networks

Keywords:

Cloud Scheduling, Analytical Hierarchy Process, Neural Networks, Multi Layer Perceptron

TABLE OF CONTENTS

CHAPTER 3
RESEARCH METHODOLOGY

CHAPTER 4
SCHEDULING IN CLOUD WITH AHP AND MLP

CHAPTER 5
IMPLEMENTATION AND TESTING

CHAPTER 6
ANALYSIS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
CI	Consistency Index
CR	Consistency Ratio
ML	Machine Learning
MLP	Multilayer Perceptron
PVS	Priority Vector of Schedulable Tasks
RI	Random Index
VM	Virtual Machine

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Cloud computing is the fastest new paradigm for delivering on demand services over internet and can be described as internet centric software. While managing resources in a cloud computing environment, scheduling can be made in different layers of the service stacks. *Scheduling in the application (software) layer* is to schedule the virtual or physical resources to support software and user applications, tasks and workflows, etc. with optimal QoS and efficiency. *Scheduling in the virtualization (platform) layer* focuses on mapping virtual resources onto physical resources with optimal load balance, energy conservation, etc. *Scheduling in the deployment (infrastructure) layer*, which also attracts worldwide attention, is concerned with optimal and strategic infrastructure, outsourcing, service placement, multi-cloud centers, partnering, data routing and application migration, etc. First, when scheduling resources in the application layer, challenges not only come from deadline and budget constraints of the cloud user, but they also come from the cloud providers in that the resources need to be utilized with balance or maximal rate. Therefore, this category can be further subdivided into i) Scheduling for user QoS, ii) Scheduling for provider efficiency and iii) Scheduling for negotiation. Secondly, when scheduling in the virtualization layer, challenges include how to schedule virtual machines (VMs) onto physical machines (PMs) efficiently with load balance, energy conservation, and/or cost effectiveness. Therefore, this category comprises i) Scheduling for load balance, ii) Scheduling for energy conservation and iii) Scheduling for cost effectiveness. Finally, scheduling in the infrastructure layer includes i) Scheduling for service placement, ii) Scheduling for partner federation and iii) Scheduling for data routing.

The dynamic and heterogeneous nature of resources and tasks in cloud computing is a major hurdle to be overcome for overall response time and better performance when designing scheduling policies. Various strategies have been proposed to address the challenges of task scheduling in cloud. Among them, priority based scheduling are amongst concerns because some jobs should be serviced earlier than other jobs that can't stay for a long time in a system. Priorities may vary according to users and they can also be assigned according to various

parameters like bandwidth, completion time, memory, etc. Generally, higher priorities are given for most urgent tasks. But task priorities can also be assigned according to the desired consumption of resource provided by the cloud service providers. For example higher priorities can be given to those resources for which frequent access from cloud users are desired for their maximum utilization. Besides, assigning priorities also helps to preempt task in the middle of execution since higher priorities tasks must be serviced before lower ones.

Among priority based schedulers, the decision making problems are widely handled using Analytic Hierarchy Process (AHP) due to its multi-criteria, multi-attribute approach which was introduced by Thomas Saaty [1]-[4]. Using AHP, the decision making problem can be decomposed into a hierarchy of more easily comprehended sub-problems, each of which can be analyzed independently. The main advantage of the AHP is its ability to rank choices in the order of their effectiveness in meeting conflicting objectives. AHP generates a weight for each evaluation criterion according to the decision maker's pairwise comparisons of the criteria. The higher the weight, the more important the corresponding criterion. Next, for a fixed criterion, the AHP assigns a score to each option according to the decision maker's pairwise comparisons of the options based on that criterion. The higher the score, the better the performance of the option with respect to the considered criterion. Finally, the AHP combines the criteria weights and the options scores, thus determining a global score for each option, and a consequent ranking. The global score for a given option is a weighted sum of the scores it obtained with respect to all the criteria. The AHP is a very flexible and powerful tool because the scores, and therefore the final ranking, are obtained on the basis of the pairwise relative evaluations of both the criteria and the options provided by the user.

AHP introduces a concept of pairwise ranking of alternatives (tasks) based on particular criteria (resources), users often have to create a comparison matrix in which tasks are relatively compared with each other in a typical scale. As a rule of thumb, Saaty has given this scale in the range {1, 2.....9}. 1 being the least important and 9 being the highest [1]-[4]. Although AHP supports multi-criteria and multi-attribute decision making model, this approach often suffers from inconsistency in comparison matrix since different users could have conflicting preferences, there might be inconsistent elements in the reciprocal pairwise comparison matrix [5], [6], [7]. Hence, some way for improving inconsistent comparison matrix to consistent one have to be introduced for better decision. Furthermore, in this approach, users typically have to provide comparison with large number of alternatives as the number of alternatives increase.

Users may even leave some comparisons without judgment. In this study, multilayer perceptron (MLP) artificial neural network that uses back-propagation strategy for error calculation in improving final output can be chosen in order to improve the comparison matrix consistency and handle the missing entries of the comparison matrix at the same time [5], [6].

1.2 THESIS ORGANIZATION

The thesis is organized into seven chapters.

Chapter 1 covers the introduction of the need of priority based scheduling approach in cloud using AHP paradigm.

Chapter 2 illustrates problem definition, objectives and relevant background study about the cloud computing, analytic hierarchy process and multi-layer perceptron with backpropagation algorithm required for the dissertation.

Chapter 3 elucidates research methodology with expounding reviews of previous literatures on several priority based approaches in cloud and using them to improve using MLP as a pronounced ML technique.

Chapter 4 explicates the AHP scheduling problem in association with the MLP in cloud. The detail construction of the comparison matrices as well as improving the consistency of the comparison matrix is described with proper algorithmic flow.

Chapter 5 is focused on implementations as well as testing issues.

Chapter 6 confers a depth analysis on the generated solution with sample test cases.

Finally, Chapter 7 concludes the dissertation with future recommendations.

Appendix section at the end presents the custom sample codes for the generated solution.

CHAPTER 2
PROBLEM DEFINITION AND BACKGROUND STUDY

2.1 PROBLEM DEFINITION

Consider a set of d resources R = $\{r_1, r_2, r_3, \ldots . r_d\}$ available in the cloud and a set of m tasks T = $\{t_1, t_2, t_3, \ldots t_m\}$ to be scheduled. Each task requests a resource with a determined priority. The scheduling problem is to assign each individual task to an optimal resource.

Priority of each resource is compared with other *d-1* resources respectively with respect to scheduling goal. This comparison is stored in a matrix of dimension $d \times d$. Next, priority of each task is compared with other *m-1* tasks separately with respect to an individual particular resource. These comparisons are stored separately in the matrix, of dimension $m \times m$.

Given these consistent matrices, the AHP problem is to compute a priority vector. However, these comparison matrices constructed from user's judgment is often inconsistent. Matrix must be consistent and consistency ratio (C.R.) as proposed by Saaty must be less than 0.1 in order to achieve acceptable priority vector for scheduling. Moreover, as the number of alternatives or criteria for comparison grows, user must provide $\frac{n(n-1)}{2}$ comparisons for n elements which is quite cumbersome. Therefore, a MLP neural network model is applied that can improve consistency as well as adapt in the situation of handling missing entries of the comparison matrix at the same time so that effective schedule can be generated.

2.2 OBJECTIVES

The objectives of this study are to

1. Implement and analyze AHP paradigm for scheduling prioritized tasks in cloud according to their priority.
2. Resolve the incomplete and inconsistent comparison matrix using multi-layer perceptron for generating efficient schedule of tasks in cloud.

2.3 BACKGROUND STUDY

2.3.1 CLOUD COMPUTING

After years in the work and 15 drafts, the National Institute of Standards and Technology's (NIST) working definition of cloud computing, the 16th and final definition has been published as The NIST Definition of Cloud Computing (NIST Special Publication 800-145) [8]. According to the official NIST definition,

> "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

The NIST definition lists five essential characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion and measured service. It also lists three "service models" (software, platform and infrastructure), and four "deployment models" (private, community, public and hybrid) that together categorize ways to deliver cloud services.

Cloud computing thus refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. The line between "low-level" infrastructure and a higher-level "platform" is not crisp. Data center hardware and software is what we call a cloud. When a cloud is made available in a pay-as-you-go manner (which is often described as "Converting capital expenses to operating expenses" i.e. CapEx to OpEx) to the general public, it is called as a public cloud; the service being sold is utility computing. The term private cloud is used to refer to internal data centers of a business or other organization, not made available to the general public. From a hardware provisioning and pricing point of view, three aspects are new in cloud computing [9]:

(1) The appearance of infinite computing resources available on demand, quickly enough to follow load surges, thereby eliminating the need for cloud computing users to plan far ahead for provisioning.

(2) The elimination of an up-front commitment by cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.
(3) The ability to pay for use of computing resources on a short-term basis as needed (for example, processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful.

Some of the recent and major cloud computing examples are Amazon EC2, Google AppEngine, Microsoft's Azure, etc. There are some compelling use cases that favor utility computing over conventional hosting. A first case is when demand for a service varies with time. For example, provisioning a data center for the peak load it must sustain a few days per month leads to underutilization at other times. Instead, cloud computing lets an organization pay by the hour for computing resources, potentially leading to cost savings even if the hourly rate to rent a machine from a cloud provider is higher than the rate to own one. A second case is when demand is unknown in advance. For example, a web startup will need to support a spike in demand when it becomes popular, followed potentially by a reduction once some visitors turn away. Finally, organizations that perform batch analytics can use the "cost associativity" of cloud computing to finish computations faster: using 1,000 EC2 machines for one hour costs the same as using one machine for 1,000 hours [9].

2.3.2 ANALYTIC HIERARCHY PROCESS

Analytic Hierarchy Process (AHP), as devised by Prof. Saaty is a comprehensive framework which is designed to cope with the intuitive, the rational and the irrational when multi-objective, multi-criterion and multi-actor decisions are made with or without certainty for any number of alternatives [1]-[4]. It organizes the basic rationality by breaking down a problem into its smaller constituent parts and then calls for only simple pairwise comparison judgments to develop priorities in each hierarchy.

There are three basic principles of AHP which one can recognize in problem solving: 1) Decomposition, 2) Comparative Judgments and 3) Synthesis of Priorities [10]. The decomposition principle calls for structuring the hierarchy to capture the basic elements of the problem. The principle of comparative judgment calls for setting up a matrix to carry out

pairwise comparisons of the relative importance of the elements in a level with respect to the elements in the level immediately above it. This matrix is used to generate a ratio scale. Finally, the principle of synthesis of priorities is used to generate the goal or composite priority of the elements at the lowest level of the hierarchy.

AHP is also based on three relatively simple axioms [11]. First, the *reciprocal axiom* requires that if $P_c(A, B)$ is a paired comparison of elements A and B with respect to their parent element C, representing how many times more the element A possesses a property than does element B, then $P_c(B, A) = \frac{1}{P_c(A,B)}$. For example if A is 5 times larger than B, then B is one-fifth as large as A. Second, the *homogeneity axiom* states that the elements being compared should not differ by too much in the property being compared. If this is not the case, large errors in judgment could occur. When constructing a hierarchy of objectives, one should attempt to arrange elements in clusters so that they do not differ by more than an order of magnitude in any cluster. (The AHP verbal scale ranges from 1 to 9). Third, the *synthesis axiom* states that judgments about or the priorities of the elements in a hierarchy do not depend on lower level elements. This axiom is required for the principle of hierarchic composition to apply.

2.3.3 USABILITY OF NEURAL NETWORK IN CLOUD SCHEDULING

Neural Networks are useful in prediction of future demands based on historic demands that involve along with several scheduling constraints. Neural network based predictors are used to forecast the requirements of resources (CPU, Memory, GPU, Disk I/O, bandwidth, load prediction, power consumption estimation, etc.) for an application or overall system. The literatures in scheduling using neural networks have considered the impact of task scheduling patterns on resource management and vice versa, helping to monitor those systems periodically [5,6]. Neural Network learn about task-resource mapping and comparisons and schedulers supported by ANN have been shown to achieve better results under the heterogeneity and large-scale simulated systems [12]. In such cases, ANN module will be an external component of the system that monitors the scheduling and task execution processes. The neural network learns from the observed failures of the machines and generates the task-machines mapping suggestions. Those “suggestions” are considered then as sub-optimal schedules and are sent to the schedulers as possible alternate solutions of the scheduling problem. Neural Networks,

therefore are convinced candidates in addressing both task scheduling and resource management optimization problems in a unique system.

2.3.3.1 Multi-Layer Perceptron

Multi-Layer Perceptron are the feed-forward neural networks which comprises of single input and output layer but the number of hidden layers is unlimited. They are called feed-forward because the nodes in the input layer activate only nodes in the subsequent hidden layer, which in turn activate only nodes in the next hidden layer, and so on until the nodes of the final hidden layer, which innervate the output layer. Every node of a particular layer is connected to every node of the subsequent layer, but this need not be the case. The layers are labeled in reverse alphabetical order and the convention is made that the i^{th} node is in the i^{th} layer, which has a total of I nodes. So is the case for j^{th} and k^{th} layers.

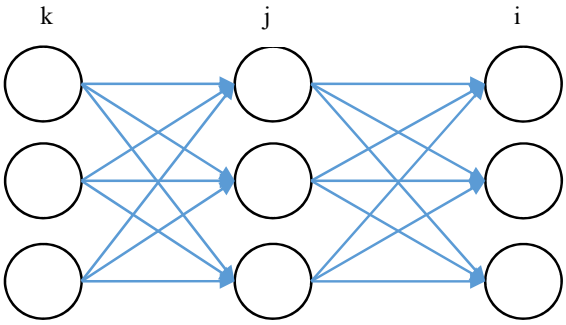


Fig 1: Multi-Layer Perceptron

2.3.3.2 The McCulloch-Pitts Neuron

A single McCulloch-Pitts (MP) neuron is a very simple neuron that transforms the weighted sum of its inputs via a function, usually non-linear, into an activation level, alternatively called the "output".

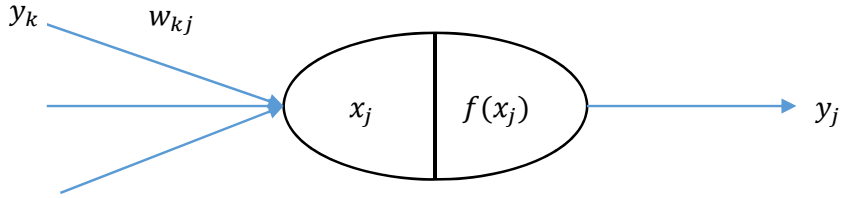


Fig 2: McCulloch-Pitts neuron takes a weighted sum (x_j) of the inputs (y_k) and passes it through the activation function $f(\cdot)$ to produce the output (y_j).

$$x_j = \sum_{k \epsilon K_j} w_{kj} y_k \text{ ----------(1)}$$

$$y_j = f(x_j) \text{ ----------(2)}$$

Where, K_j is the set of nodes from the k^{th} layer which feed node j. There are several types of activation functions $f(\cdot)$ that can be used. For e.g.

Linear,

$$f(z) = \beta z \text{----------(3)}$$

Threshold,

$$f(z) = \begin{cases} 1 & z \geq \theta \\ 0 & z < \theta \end{cases} \text{ ----------(4)}$$

Sigmoid,

$$f(z) = \frac{1}{1+e^{-\gamma z}} \text{----------(5)}$$

Radial Basis as in e.g. the Gaussian,

$$f(z) = \exp\left\{-\frac{(z-\mu)^2}{\sigma^2}\right\} \text{----------(6)}$$

Here, $\beta, \theta, \gamma, \sigma$ and μ are the free parameters which control the "shape" of the function.

2.3.3.3 Choice of Sigmoidal Function

In the derivation of the backpropagation algorithm, the sigmoidal function is largely used because its derivative has some nice property. For simplicity, the parameter γ is assumed to be unity.

Taking the derivative of sigmoidal function by application of "quotient rule",

$$
\begin{aligned}
f(z) &= \frac{1}{1+e^{-z}} \\
\frac{df(z)}{dz} &= \frac{(1+\ e^{-z})\frac{d(1)}{dz} - 1 \times \frac{d(1+e^{-z})}{dz}}{(1+e^{-z})^2} \\
&= \frac{0-(-e^{-z})}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} \times \frac{e^{-z}}{1+e^{-z}} \\
&= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) \\
&= f(z)\big(1-f(z)\big)
\end{aligned}
$$

$$\therefore \frac{df(z)}{dz} = f(z)(1-f(z)) \text{ ----------(7)}$$

This differential property of sigmoidal function is efficient and practically easy to deploy during execution. Other activation functions such as hyperbolic tangent, can be chosen which are also differentiable and have faster convergence rate than sigmoidal function.

2.3.3.4 Training Neural Network using Backpropagation Algorithm

Backpropagation is an iterative algorithm, which means that the weights are not changed all at once but rather incrementally. It is also a supervised learning algorithm attempts to minimize the error between the actual outputs i.e., the activation at the output layer – and the desired or "target" activation, in this case by changing the values of the weights in the network. The total error is the sum of these errors for each output node. Further, since negative and positive errors are not supposed to cancel each other out, these differences are squared before summing. Finally, this quantity is scaled by a factor of ½ for convenience.

$$E = \frac{1}{2}\sum_{j=1}^{J}\big(t_j - y_j\big)^2 \text{ ----------(8)}$$

This equation applies only when the j^{th} layer is the output layer, which is the only layer for which error is defined. The weight change for a weight connecting a node in layer k to a node in layer j is

$$\Delta w_{kj} = -\frac{\alpha \partial E}{\partial w_{kj}} \text{----------(9)}$$

Here, α is a free parameter (the "learning rate") that is set prior to training; it lets us scale our step size according to the problem at hand. The negative sign indicates that the weight change are in direction of decrease in error. Now expanding partial derivative by the chain rule,

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial w_{kj}} \text{ ---------- (10)}$$

Since from equation 1, it can be recalled that x_j is the weighted sum of the inputs into the j^{th} node,

$$\frac{\partial x_j}{\partial w_{kj}} = y_k \text{ ---------- (11)}$$

For notational convenience, the first two factors can be treated as a single quantity, an "error term":

$$\delta_j = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j} \text{---------- (12)}$$

Also, equation 2 represents a sigmoidal function, the derivative of which can be calculated as shown in equation 7, such that:

$$\frac{\partial y_j}{\partial x_j} = y_j(1 - y_j) \text{ ---------- (13)}$$

Finally, the first partial derivative in the error term can be considered. When j is an output layer, this quantity is just the derivative of equation 8 with respect to y_j, i.e,

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j) \text{ ---------- (14)}$$

Therefore, overall equation 10 can be written as,

$$\frac{\partial E}{\partial w_{kj}} = -\big(t_j - y_j\big)y_j\big(1 - y_j\big)y_k \text{--------- (15)}$$

For the case where j is a hidden layer, $\frac{\partial E}{\partial y_j}$ is not quite as simple. Intuitively, it is needed to visualize how the error caused by y_j has propagated into the activations of the next (i^{th}) layer. Mathematically, this amounts to another application of the chain rule:

$$\frac{\partial E}{\partial y_j} = \sum_{i \epsilon I_j}\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial y_j} \text{---------- (16)}$$

The first two partial derivatives are from equation 12, just the error term for the next (i^{th}) layer δ_i. The final term is simply the derivative of equation 1 with respect to the input:

$$\frac{\partial x_i}{\partial y_j} = w_{ji} \text{ ---------- (17)}$$

Making these substitutions yields:

$$\frac{\delta E}{\delta y_j} = -\sum_{i \epsilon I_j} \delta_i w_{ji} \text{---------- (18)}$$

So that in the case of hidden layers,

$$\frac{\partial E}{\partial w_{kj}} = -\sum_{i \epsilon I_j} (\delta_i w_{ji}) y_j (1 - y_j) y_k \text{ ---- (19)}$$

A single formula can also be written by employing our definition of the error term as in equation 12:

$$\frac{\partial E}{\partial w_{kj}} = -\delta_j y_k \text{ ---------- (20)}$$

Combining all together, for a weight connecting a node in layer k to a node in the layer j, the change in weight is given by the equation:

$$\Delta w_{kj}(n) = \alpha \delta_j y_k + \eta \Delta w_{kj}(n-1) \quad \text{--- (21)}$$

Where:

α is the learning rate, a real value on the interval (0,1];

y_k is the activation of the node in layer k, i.e. the activation of the presynaptic node, the one upstream of the weight;

n and $n-1$ refer to the iteration through the loop (i.e. the "epoch");

η is the momentum, a real value on the interval [0,1); and

δ_j is the "error term" associated with the node after the weight, i.e. postsynaptic node, such that

If j is the output layer,

$$\delta_j = (t_j - y_j) y_j (1 - y_j) \text{---------- (22)}$$

If j is the hidden layer, then

$$\delta_j = \left(\sum_{i \epsilon I_j} \delta_i w_{ji}\right) y_j (1 - y_j) \text{ ---------- (23)}$$

CHAPTER 3
RESEARCH METHODOLOGY

3.1 RESEARCH METHODOLOGY

The study is based on qualitative, idiographic and analytical research methodology as it involves modelling and analysis of schedule of prioritized tasks using AHP paradigm in cloud. Also, the study involves in solution of improving the consistency and handling missing values of the comparison matrix. Primary data is generated randomly as per the requirement during the study. Empirical approaches are undertaken in generation of training data and test sets for training MLP, experimental control and manipulation of variables such as learning rate, number of hidden layers, choice of activation function, choice of error function etc.

3.2 LITERATURE REVIEW

Several task scheduling algorithms have been proposed by different researchers to decrease the computational complexity and increase performance in the cloud computation. Some well-known examples of such algorithms intended to be applied in cloud computing environment are Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution, Ant Colony Optimization (ACO), etc. [13], [14], [15]. Each of these algorithms estimate the completion and execution time of each submitted task on each available resource. Salman et al. [14] have shown that the performance of PSO algorithm is faster than GA in solving static task assignment problem for homogeneous distributed computing systems based on their test cases. Pandey et al. [15] have used PSO as it has a faster convergence rate than GA. Also, it has fewer primitive mathematical operators than in GA (e.g. reproduction, crossover, mutation, etc.), making applications less dependent on parameter fine-tuning. Moreover, using discrete numbers, particle's position can easily be co-related to task-resource mappings. Other algorithms such as Ant Colony and Differential Evolution are the variants of swarm and evolutionary approaches used in cloud scheduling [13].

Regarding priority based algorithms, a number of literatures have been introduced for cloud by researchers [16], [17], [18] in their study. Patel and Bhoi [18] have compared several priority based algorithms in grid and cloud. Some of them have only focused on parameters such as

cost, time and makespan while others have shown emphasis on resource utilization and speed based on their static and dynamic nature. Their work, however have focused on few criteria of jobs in scheduling.

Ghanbari and Othman [19] have proposed priority based multiple criteria decision making model for job scheduling in cloud based on the theory of AHP. They have shown that the hierarchical framework of cloud scheduling can be represented by placing cloud jobs in the alternative or criterion level and cloud resources in the attributes level with scheduling as an overall goal. Their work suggests to choose a job with maximum amount of priority value based on PVS (priority vector of scheduling jobs) and allocate it suitable resource. Improving the proposed algorithm in order to gain less finish time (makespan) is considered as their future work. Moreover, they have also indicated that the worst case complexity of their proposed algorithm depends upon the number of jobs and resources which is given by the equation:

$$d^{2.81} + d \times m^{2.81} \text{---------- (24)}$$

Where, *m* and *d* are the number of jobs and resources respectively and an assumption is made that a matrix multiplication takes approximately $m^{2.81}$ arithmetic operations (additions and multiplications).

Saaty et.al. [1]-[4] have demonstrated the detailed decomposition of decision making problem under AHP paradigm. Their work explains the construction of comparison matrix, the need and recommended {1...9} scale of relative importance for comparison, synthesis of priorities using geometric mean as the approximation to computation of eigenvectors. The study suggests that after finding the largest eigenvalue λ_{max} of the comparison matrix of n elements, the consistency index can be found by

$$C.I = \frac{\lambda_{max} - n}{n-1} \text{---------- (25)}$$

Also, the consistency ratio can be calculated by dividing the consistency index by random consistency number (*R.I*) of the same size matrix which is obtained as the empirical value after several observations.

$$C.R = \frac{C.I}{R.I} \text{---------- (26)}$$

This C.R value is tolerated for acceptable consistent solution if it is less than 10%. Moreover, Saaty [1] himself has shown that for a positive reciprocal matrix $\lambda_{max} = n$ is a necessary and sufficient condition for consistency and with inconsistency $\lambda_{max} > n$ always holds.

Peláez and Lamata [7] have also mentioned that *R.I.* is the average value of CI obtained from 500 positive reciprocal pairwise comparison matrices whose entries were randomly generated using the 1 to 9 scale.

Gomez, Karanik and Peláez [5] have introduced Multi-Layer Perceptron (MLP) neural network model which is able to complete the missing values and improve the matrix consistency at the same time. Their work presents the comparative evaluation of Revised Geometric Mean Method (RGM) and Connecting Paths Method (CP) for reconstructing comparison matrix in the case of missing values. However, these methods do not guarantee that the matrix consistency is improved. The authors have shown that the neural network-based model can improve performance when the number of missing elements increases, as this model has capability to fill those matrix positions with appropriate values learned from consistent matrix configurations. Due to the fact that hyperbolic tangent transfer function is defined in the interval [-1, 1], their work suggests to normalize the training patterns (inputs and desired outputs) in that interval before being presented to the network. To do that, the interval [-1, 1] is divided into seventeen uniform spaced values creating a correspondence between these values and the Saaty's scale values $\{\frac{1}{9}, \ldots \frac{1}{2}, 1, 2, \ldots 9\}$. In this way, normalization process consists of replacing every matrix value for its corresponding normalized value. When the training process has finished, the network is ready to recognize incomplete pairwise matrices. Finally, the obtained results are transformed into their original form and the missing value positions are filled with the corresponding transformed network outputs.

Hu and Tsai [6] have proposed back-propagation based neural network as the regression tool for prediction of single missing entry of the comparison matrix from the available elements. Their work also compares MLP with RGM (Revised Geometric Mean Method), CP (Connecting Path Method) and Characteristic Polynomial-Based Method for matrix reconstruction. However, the challenge of knowing the upper bound on the number of missing elements in the matrix remains.

Ergu, Kou, Peng and Shi [20] have proposed an induced bias matrix method for identification of inconsistent elements and further improving the schedule of tasks even if the consistency of the matrix is already under 0.1. Even so, the method lacks to provide directions in missing elements case as well as providing proper allocation of corresponding computing resources dynamically and accurately for all tasks.

CHAPTER 4
SCHEDULING IN CLOUD WITH AHP AND MLP

4.1 ANALYTICAL HIERARCHY PROCESS FOR TASK SCHEDULING IN CLOUD

Initially, scheduling based decision is viewed in the hierarchical fashion of subsequent criteria and its possible set of alternatives. Thereafter a comparison among the alternatives and criteria is performed using a comparison matrix in reference to Saaty's Scale. Those comparison matrices are made separately for tasks that are to be scheduled and resources available in the servers. Further improvements on the consistency of the comparison matrix is performed by MLP if required. Final output is the priority vector schedule of the tasks needed for execution. The respective steps are illustrated in subsequent sub steps.

4.1.1 DECOMPOSITION OF PROBLEM

AHP decomposes any decision making goal into following hierarchical structure of criteria and alternatives.

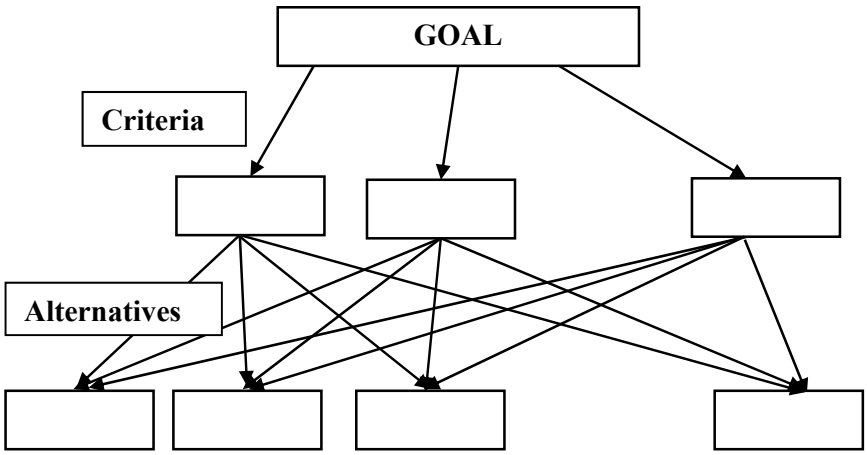


Fig 3: Decomposition of problem into hierarchy [4].

For scheduling problem in cloud, consider *d* resources and *m* tasks, this structure would resemble following figure.

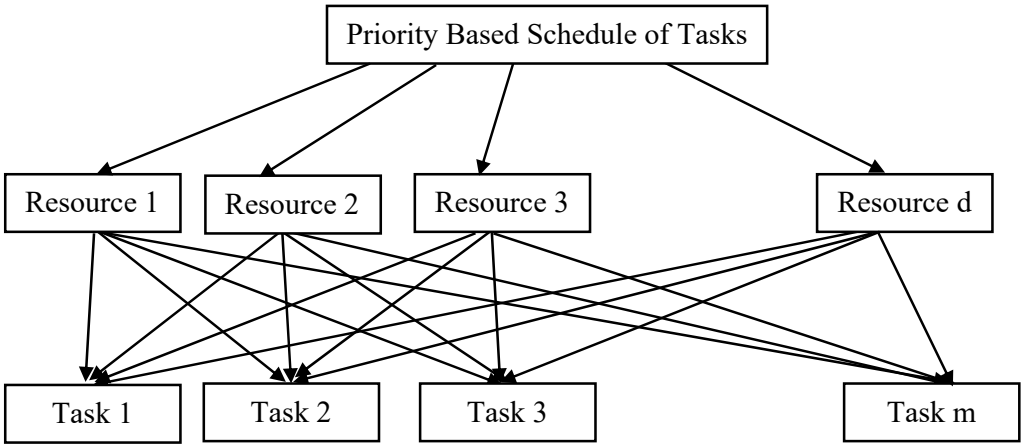


Fig 4: Decomposition of task scheduling problem using AHP paradigm [15].

4.1.2 CONSTRUCTION OF RESOURCE COMPARISON MATRIX

Each resource is compared with other *d-1* resources based on the fundamental scale of {1, 2...9}. Let this matrix be called A such that

Goal	Resource 1	Resource 2	Resource 3	.....	Resource d
Resource 1	a_{11}	a_{12}	a_{13}	....	a_{1d}
Resource 2	a_{21}				
Resource 3	a_{31}				
Resource d	a_{d1}			....	a_{dd}

Table 1: Comparison among resources with respect to scheduling goal.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ a_{21} & a_{22} & \dots & a_{2d} \\ . & . & \dots & . \\ a_{d1} & a_{d2} & \dots & a_{dd} \end{pmatrix} \text{---------- (27)}$$

Each resource is given weight $\omega = (w_1, w_2, \dots . w_d)$ such that

$$A = \begin{pmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \cdots & \frac{w_1}{w_d} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \cdots & \frac{w_2}{w_d} \\ \frac{w_d}{w_1} & \frac{w_d}{w_2} & \cdots & \frac{w_d}{w_d} \end{pmatrix} \text{---------- (28)}$$

Thus,

$$A = \begin{cases} a_{ij} = \frac{1}{a_{ji}}, & if\ i \neq j \\ 1, & if\ i = j \end{cases} \text{---------- (29)}$$

i.e.

$$A = \begin{pmatrix} 1 & a_{12} & \cdots & a_{1d} \\ \frac{1}{a_{12}} & 1 & \cdots & a_{2d} \\ \frac{1}{a_{1d}} & \frac{1}{a_{2d}} & \cdots & 1 \end{pmatrix} \text{---------- (30)}$$

Multiplying equation (30) with vector of weights ω,

$$A\omega = n\omega \text{---------- (31)}$$

But, the problem is that the precise value of $\frac{w_i}{w_j}$ for matrix A cannot be given, but can only be estimated from the fundamental scale. Therefore, our problem becomes,

$$A'\omega' = \lambda_{max}\omega' \text{---------- (32)}$$

Where, λ_{max} is the largest principal eigen value of A'. For perfectly consistent matrix, Saaty has shown that $\lambda_{max} = n$. The variance of error incurred in estimating λ_{max} is given by C.I equation 25. Matrix A is said to be of acceptable consistency if C.R given by equation 26 is less than 0.1, where R.I is obtained from the lookup table of same matrix size [4].

Intensity of Importance	**Definition**	**Explanation**
1	Equal importance	Two activities contribute equally to the objective
2 3	Weak Moderate importance	Experience and judgment slightly favor one activity over another

4 5	Moderate plus Strong importance	Experience and judgment strongly favor one activity over another
6 7	Strong plus Very strong or demonstrated importance	An activity is favored very strongly over another; its dominance demonstrated in practice
8 9	Very, very strong Extreme importance	The evidence favoring one activity over another is of the highest possible order of affirmation
Reciprocals of above	If activity *i* has one of the above non zero numbers assigned to it when compared with activity *j*, then *j* has the reciprocal value when compared with *i*	A reasonable assumption

Table 2: The fundamental scale [1]-[4].

4.1.3 CONSTRUCTION OF TASK COMPARISON MATRIX

Consider the priority vector constructed from matrix A be called δ which is of order $d \times 1$. Same process for computing task comparison matrix as well as priority vector with respect to each resource is carried out. Each tasks on the alternatives level are compared separately with respect to every resource among all tasks i.e., if there are m tasks, $m \times m$ matrices are created that compares m tasks with respect to a particular resource for all d resources.

$\boldsymbol{Resource_i}$ *(i=1 to d)*	Task 1	Task 2	.....	Task m
Task 1	t_{11}	t_{12}	....	t_{1m}
Task 2	t_{21}			
. .				
Task m	t_{m1}		....	t_{mm}

Table 3: Task comparison matrix w.r.t Resource *i* (T_i)

w_1
w_2
w_3
.
w_m

Table 4: Respective priority vector for each task comparison matrix (α_i).

Let, these matrices be called $T_1, T_2, \ldots\ldots, T_d$. Also, let the respective priority vectors associated to $T_1, T_2, \ldots\ldots, T_d$ be $\alpha_1, \alpha_2, \ldots \alpha_d$ each of order $m \times 1$. Let, a delta matrix be constructed that comprises every priority vectors of tasks.

i.e. $\Delta = (\alpha_1, \alpha_2, \ldots \alpha_d)$. ---------- (33)

This Δ matrix will be of order $m \times d$. Now a PVS (Priority Vector of Schedulable Tasks) vector is constructed such that

$$PVS_{m\times 1} = \Delta_{(m\times d)} \times \delta_{(d\times 1)} \text{---------- (34)}$$

Such constructed PVS vector will now be of order $m \times 1$. The task with highest priority is selected as the best task for accessing resource from this PVS vector [19].

4.2 IMPROVING CONSISTENCY OF COMPARISON MATRIX USING MULTI-LAYER PERCEPTRON

As the number of alternatives or criteria grows, the order of comparison matrix grows with the need of providing more comparisons from the user which is quite cumbersome if more number of elements have to be compared. For *n* number of elements, user must explicitly provide

$\frac{n\,(n-1)}{2}$ comparisons. To ease this issue, a Multi-Layer Perceptron (MLP) neural network can be implemented which not only improves the consistency of the matrix but can also handle some of the missing input values in the comparison matrix [5], [6].

Since, the lower triangular portion of the comparison matrix is just the reciprocal of the upper triangular part, MLP can be trained by just using the input from upper triangular portion of the comparison matrix. For example, the following reciprocal comparison matrix A of order 5, with two missing elements (represented by '–') and the corresponding neural network architecture would be:

$$A = \begin{pmatrix} 1 & a_{12} & a_{13} & - & a_{15} \\ \frac{1}{a_{12}} & 1 & a_{23} & - & a_{25} \\ \frac{1}{a_{13}} & \frac{1}{a_{23}} & 1 & a_{34} & a_{35} \\ — & — & \frac{1}{a_{34}} & 1 & a_{45} \\ \frac{1}{a_{15}} & \frac{1}{a_{25}} & \frac{1}{a_{35}} & \frac{1}{a_{45}} & 1 \end{pmatrix}$$

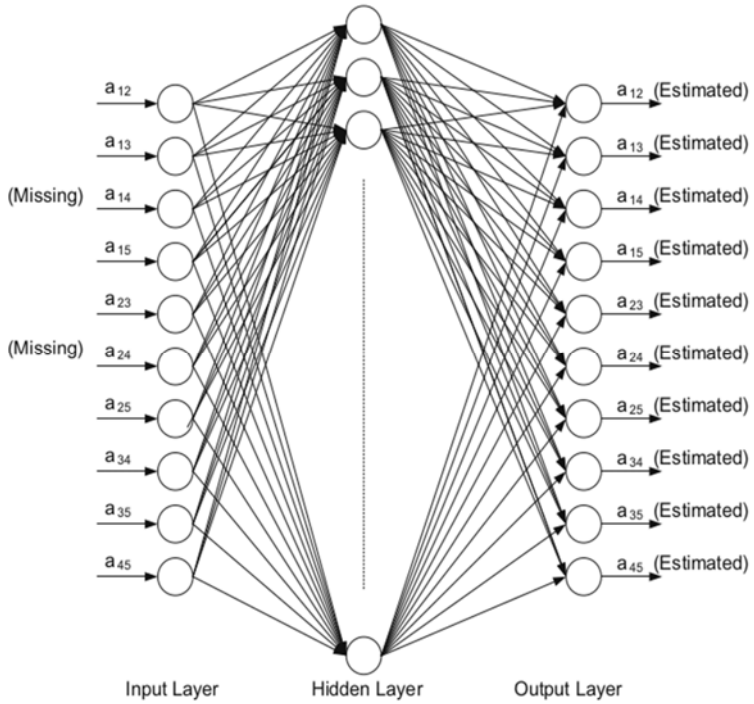
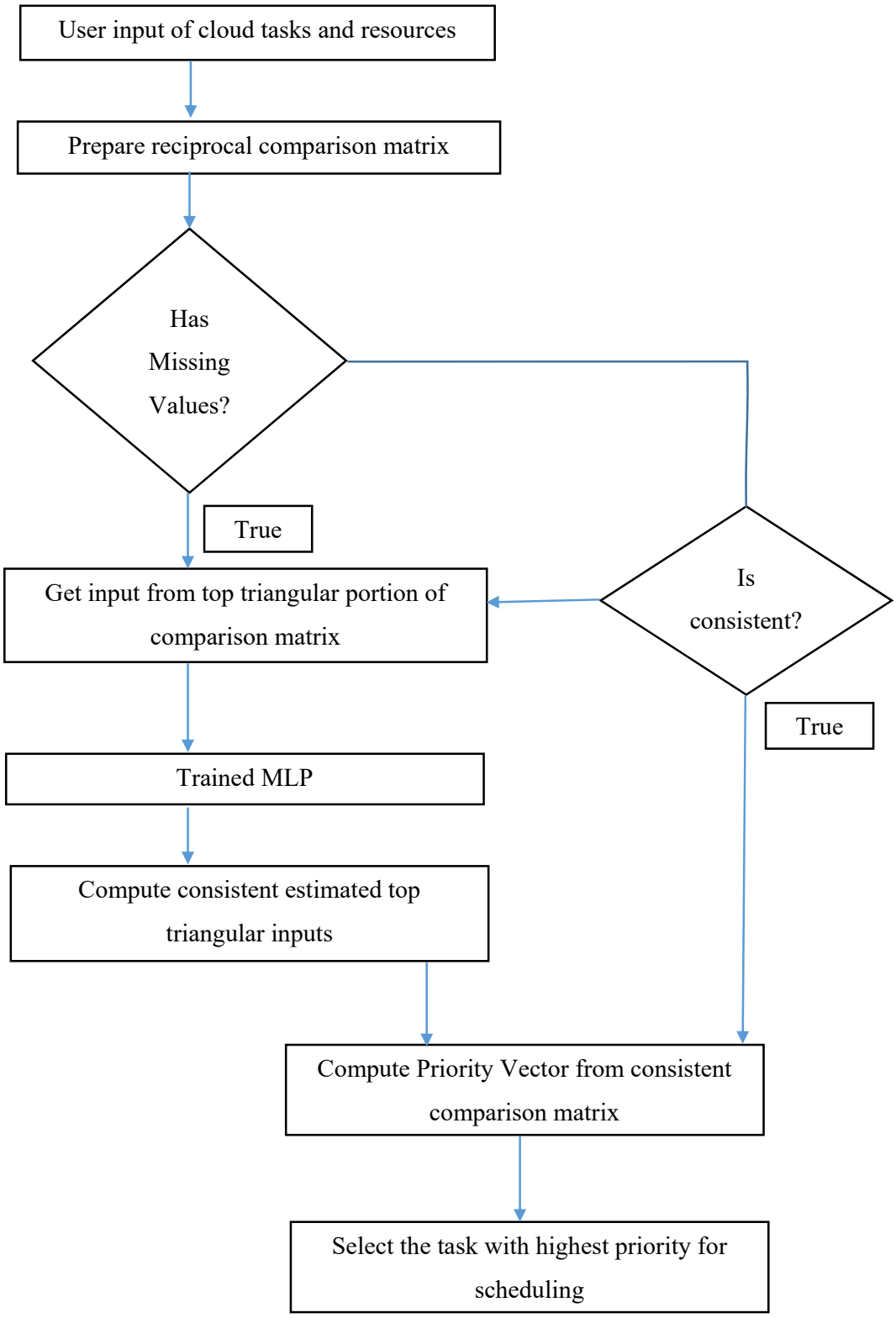


Fig 5: The corresponding neural network for missing value comparison matrix [5].

The MLP will be trained using Backpropagation learning algorithm. The MLP architecture contains neurons in layers: one input layer, one or more hidden layers and one output layer. In the learning process, the neural network adapts the connection weights, attempting to minimize the difference between the network output and the desired output. When an acceptable average training error is reached, the network is able to recognize incomplete patterns. Significant computational steps are explained mathematically in subsection 2.3.3.4.

4.3 ALGORITHMIC FLOW



CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 IMPLEMENTATION

The tools used for the implementation and its scenario with sample test cases and test data are described below:

5.1.1 TOOLS USED

The algorithm in the study was implemented using JAVA programming language. Entire cloud scheduling simulation was performed using Cloudsim 3.0.3 simulator. MLP based neural network was implemented using Neuroph 2.9 Java api. Math libraries for extending matrix implementations were taken from Java api provided by Apache Commons. Eclipse IDE was the default choice for coding purpose. The entire simulation was performed on x64-based PC with processor Intel(R) Core(TM) i3-3120M CPU @ 2.50GHz, 2 Cores, and 4 Logical Processors.

5.1.2 IMPLEMENTATION SCENARIO

Cloudsim package was first initialized with the default configuration. For this purpose, a single datacenter was created. A datacenter broker was made to submit cloudlets (tasks) as well as VMs (resources) to the datacenter. Then resource comparison matrix was created from the given set or resources. The consistency ratio of the matrix was then calculated using AHP. If the consistency ratio was found to be more than 10 percent, a matrix was revised by using neural network process. For training comparison matrix, randomly generated matrices using pseudorandom function which have C.R less than 0.1 were used. Same process was carried on for task comparison matrices with respect to each resource. Finally, PVS vector can be obtained that contained overall priority of tasks. The task with highest priority was then selected for execution in the Cloudsim environment.

5.1.3 SAMPLE MODULES

Class *AHPMLP.java* was used for cloud scheduling simulation. Method *NNProcess* received the comparison matrix as input. If missing elements were present, then the corresponding values were suggested by the neural network using Backpropagation Algorithm from Neuroph api. The matrix order, missing number of elements and number of training sample remained the choice for user to customize.

Class *MatrixManipulations.java* enabled further matrix processing of the comparison matrix for example, extracting upper triangular matrix data, normalizing and de-normalizing matrix data in the Saaty's scale range, computing matrix CR, calculation of Eigen Values and printing matrix data, etc.

The details are mentioned in the Appendix Section.

5.2 TESTING

Some empirical observations for the selection of parameters during the training process were made. For simulation purpose, hyperbolic tangent function was taken as the choice of activation function because of its fast convergence nature than sigmoidal. Value of learning rate α was taken to be 0.01. Maximum iteration epoch for the algorithm was taken to be 10000 and maximum allowable error was taken as 0.001. Default number of training sample was chosen as 30.

5.2.1 SAMPLE TEST CASES AND TEST DATA

Two separate test cases were prepared for the analysis which are discussed in detail in Chapter 6. First case includes the analysis of comparison matrix with CR > 0.1 and contained no missing elements. Second case includes the analysis of comparison matrix with missing elements.

For training neural network test data were generated using pseudorandom number generator Java function in the range of {1/9…1/2, 1, 2…9}. Default training size was 30 for training each inconsistent matrix. The upper triangular data portion of the matrices whose CR value was less than 0.1 were taken as the target vectors for training MLP.

CHAPTER 6
ANALYSIS

Two distinguishing cases are taken for the analysis of the inconsistency of the comparison matrix using MLP. First case presents the analysis of comparison matrix with CR > 10% and no other missing elements. Second presents the comparison matrix with the missing elements. For simplicity cases 3 resources and 4 tasks are taken under observation.

6.1 ANALYSIS OF COMPARISON MATRIX WITH CR > 0.1 AND NO MISSING ELEMENTS

A comparison matrix of 3 resources (VMs) with CR > 0.1 is chosen. It contained no any missing elements. The upper triangular portion of the matrix was fed to the trained neural network. Eventually, the trained network corrects the inconsistent matrix into consistent one using C.R. < 0.1. Following sample cases were observed:

6.1.1 SAMPLE OUTPUT 1 WITH NO MISSING ENTRIES IN RESOURCE COMPARISON MATRIX:

```
Starting CloudSim simulation...
Initialising...
3 resources (VMs) and 4 tasks (cloudlets) successfully created as desired.
Resource comparison matrix created:
|  1.000    3.000    5.000  |
|  0.333    1.000    8.000  |
|  0.200    0.125    1.000  |
CR:0.24137931034482743
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    0.250    0.111  |
|  4.000    1.000    1.000  |
|  9.000    1.000    1.000  |
CR:0.06379310344827573
```

Here, the comparison ratio of the matrix was reduced from 0.2413 to 0.06379 which was less than 0.1. MLP was thus effective in correcting CR in this case. Similar were the cases with 3 task comparison matrices consisting 4 tasks (cloudlets) with respect to each resource:

6.1.2 SAMPLE OUTPUT 2 WITH NO MISSING ENTRIES IN TASK COMPARISON MATRICES:

```
Following task comparison matrices are created :
```

```
=============TASK COMPARISON MATRIX (1)=================
|  1.000    0.500    4.000    0.167  |
|  2.000    1.000    4.000    0.250  |
|  0.250    0.250    1.000    0.200  |
|  6.000    4.000    5.000    1.000  |
CR:0.10518518518518512
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.111    7.000    0.500  |
|  9.000    1.000    0.500    1.000  |
|  0.143    2.000    1.000    0.250  |
|  2.000    1.000    4.000    1.000  |
New CR:1.0199999999999998
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.333    4.000    7.000  |
|  3.000    1.000    2.000    4.000  |
|  0.250    0.500    1.000    2.000  |
|  0.143    0.250    0.500    1.000  |
New CR:0.14629629629629612
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.500    0.125    5.000  |
|  2.000    1.000    0.143    3.000  |
|  8.000    7.000    1.000    8.000  |
|  0.200    0.333    0.125    1.000  |
New CR:0.12703703703703703
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    9.000    9.000    4.000  |
|  0.111    1.000    1.000    0.111  |
|  0.111    1.000    1.000    0.125  |
|  0.250    9.000    8.000    1.000  |
New CR:0.08481481481481484

=============TASK COMPARISON MATRIX (2)=================
|  1.000    0.500    0.333    5.000  |
|  2.000    1.000    0.500    7.000  |
|  3.000    2.000    1.000    9.000  |
|  0.200    0.143    0.111    1.000  |
CR:0.015555555555555486

=============TASK COMPARISON MATRIX (3)=================
|  1.000    0.200    1.000    0.500  |
|  5.000    1.000    0.500    2.000  |
|  1.000    2.000    1.000    3.000  |
|  2.000    0.500    0.333    1.000  |
CR:0.21703703703703714
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.125    0.167    3.000  |
|  8.000    1.000    7.000    8.000  |
|  6.000    0.143    1.000    2.000  |
|  0.333    0.125    0.500    1.000  |
New CR:0.22518518518518504
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.111    0.250    0.500  |
```

```
|  9.000    1.000    2.000    7.000  |
|  4.000    0.500    1.000    4.000  |
|  2.000    0.143    0.250    1.000  |
New CR:0.016666666666666642
```

In the above case, correction was made only for task comparison matrix 1 and 3 since task comparison matrix 2 already had C.R < 0.1.

6.2 ANALYSIS OF COMPARISON MATRIX WITH MISSING ELEMENTS

For missing elements, represented by -2, was taken as the input in the comparison matrix. Since the hyperbolic tangent activation function has the range [-1, 1], it was able to recognize the missing elements with any other number outside this range.

As in the above case, the elements from the upper triangular matrix was taken as an input to MLP. The MLP was successful enough to reduce the consistency less than 0.1 in these cases too.

6.2.1 SAMPLE OUTPUT 3 CONTAINING 1 MISSING ENTRY IN RESOURCE COMPARISON MATRIX ONLY

```
Starting CloudSim simulation...
Initialising...
3 resources (VMs) and 4 tasks (cloudlets) successfully created as desired.
Resource comparison matrix created:
|  1.000    3.000    5.000  |
|  0.333    1.000    -2.000  |
|  0.200    -2.000    1.000  |
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    9.000    9.000  |
|  0.111    1.000    0.500  |
|  0.111    2.000    1.000  |
CR:0.046551724137930885

Following task comparison matrices are created :

=============TASK COMPARISON MATRIX (1)=================
|  1.000    0.500    4.000    0.167  |
|  2.000    1.000    4.000    0.250  |
|  0.250    0.250    1.000    0.200  |
|  6.000    4.000    5.000    1.000  |
CR:0.10518518518518512
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.111    0.111    0.111  |
|  9.000    1.000    9.000    3.000  |
|  9.000    0.111    1.000    0.143  |
|  9.000    0.333    7.000    1.000  |
```

```
New CR:0.26148148148148165
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.250    2.000    0.111  |
|  4.000    1.000    7.000    0.200  |
|  0.500    0.143    1.000    0.111  |
|  9.000    5.000    9.000    1.000  |
New CR:0.07074074074074067

=============TASK COMPARISON MATRIX (2)=================
|  1.000    0.500    0.333    5.000  |
|  2.000    1.000    0.500    7.000  |
|  3.000    2.000    1.000    9.000  |
|  0.200    0.143    0.111    1.000  |
CR:0.015555555555555486

=============TASK COMPARISON MATRIX (3)=================
|  1.000    0.200    1.000    0.500  |
|  5.000    1.000    0.500    2.000  |
|  1.000    2.000    1.000    3.000  |
|  2.000    0.500    0.333    1.000  |
CR:0.21703703703703714
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.167    0.500    0.143  |
|  6.000    1.000    8.000    2.000  |
|  2.000    0.125    1.000    0.125  |
|  7.000    0.500    8.000    1.000  |
New CR:0.06703703703703705

Printing delta Matrix:
|  0.068    0.173    0.054  |
|  0.223    0.293    0.507  |
|  0.044    0.491    0.068  |
|  0.665    0.043    0.371  |

Printing PVS Matrix:
|  0.074  |
|  0.260  |
|  0.079  |
|  0.587  |
The selected job was : job 3
```

6.2.2 SAMPLE OUTPUT 3 CONTAINING SEVERAL MISSING ENTRIES IN TASK COMPARISON MATRICES ONLY

```
Starting CloudSim simulation...
Initialising...
3 resources (VMs) and 4 tasks (cloudlets) successfully created as desired.
Resource comparison matrix created:
|  1.000    3.000    5.000  |
|  0.333    1.000    8.000  |
|  0.200    0.125    1.000  |
CR:0.24137931034482743
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    0.111    8.000  |
|  9.000    1.000    9.000  |
```

```
|  0.125    0.111    1.000  |
CR:0.4310344827586207
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    8.000    0.167  |
|  0.125    1.000    0.111  |
|  6.000    9.000    1.000  |
CR:0.2758620689655171
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    1.000    2.000  |
|  1.000    1.000    4.000  |
|  0.500    0.250    1.000  |
CR:0.046551724137930885

Following task comparison matrices are created :

=============TASK COMPARISON MATRIX (1)=================
|  1.000    0.500    -2.000    0.167  |
|  2.000    1.000    4.000    -2.000  |
|  -2.000    0.250    1.000    0.200  |
|  6.000    -2.000    5.000    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    1.000    1.000    0.250  |
|  1.000    1.000    0.500    1.000  |
|  1.000    2.000    1.000    0.333  |
|  4.000    1.000    3.000    1.000  |
New CR:0.13814814814814821
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.500    0.500    5.000  |
|  2.000    1.000    3.000    6.000  |
|  2.000    0.333    1.000    7.000  |
|  0.200    0.167    0.143    1.000  |
New CR:0.07518518518518529

=============TASK COMPARISON MATRIX (2)=================
|  1.000    0.500    0.333    5.000  |
|  2.000    1.000    -2.000    7.000  |
|  3.000    -2.000    1.000    9.000  |
|  0.200    0.143    0.111    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    9.000    0.333    0.200  |
|  0.111    1.000    0.143    0.143  |
|  3.000    7.000    1.000    0.333  |
|  5.000    7.000    3.000    1.000  |
New CR:0.16592592592592606
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    1.000    7.000    2.000  |
|  1.000    1.000    6.000    3.000  |
|  0.143    0.167    1.000    0.200  |
|  0.500    0.333    5.000    1.000  |
New CR:0.027037037037037186

=============TASK COMPARISON MATRIX (3)=================
|  1.000    0.200    1.000    -2.000  |
|  5.000    1.000    0.500    2.000  |
|  1.000    2.000    1.000    3.000  |
```

```
|  -2.000    0.500    0.333    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.200    0.500    1.000  |
|  5.000    1.000    1.000    5.000  |
|  2.000    1.000    1.000    5.000  |
|  1.000    0.200    0.200    1.000  |
New CR:0.0396296296296297

Printing delta Matrix:
|  0.195    0.366    0.114  |
|  0.476    0.399    0.441  |
|  0.279    0.051    0.357  |
|  0.049    0.185    0.088  |

Printing PVS Matrix:
|  0.264  |
|  0.434  |
|  0.182  |
|  0.119  |
The selected job was : job 1
```

6.2.3 SCHEDULING IMPLEMENTATION INVOLVING MISSING ENTRIES IN BOTH RESOURCE AS WELL AS TASK COMPARISON MATRICES

```
Starting CloudSim simulation...
Initialising...
3 resources (VMs) and 4 tasks (cloudlets) successfully created as desired.
Resource comparison matrix created:
|  1.000    3.000    5.000  |
|  0.333    1.000    -2.000  |
|  0.200    -2.000    1.000  |
Training with MLP to correct CR of the resource comparison matrix...
|  1.000    0.125    1.000  |
|  8.000    1.000    9.000  |
|  1.000    0.111    1.000  |
CR:0.001724137931034293

Following task comparison matrices are created :

=============TASK COMPARISON MATRIX (1)=================
|  1.000    0.500    -2.000    0.167  |
|  2.000    1.000    4.000    -2.000  |
|  -2.000    0.250    1.000    0.200  |
|  6.000    -2.000    5.000    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.500    0.333    0.500  |
|  2.000    1.000    0.200    1.000  |
|  3.000    5.000    1.000    1.000  |
|  2.000    1.000    1.000    1.000  |
New CR:0.10148148148148148
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.167    0.500    0.500  |
|  6.000    1.000    4.000    6.000  |
```

```
|  2.000    0.250    1.000    1.000  |
|  2.000    0.167    1.000    1.000  |
New CR:0.017037037037037135

=============TASK COMPARISON MATRIX (2)================
|  1.000    0.500    0.333    5.000  |
|  2.000    1.000    -2.000    7.000  |
|  3.000    -2.000    1.000    9.000  |
|  0.200    0.143    0.111    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    7.000    0.250    8.000  |
|  0.143    1.000    0.250    7.000  |
|  4.000    4.000    1.000    9.000  |
|  0.125    0.143    0.111    1.000  |
New CR:0.2444444444444445
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.500    8.000    0.111  |
|  2.000    1.000    9.000    0.143  |
|  0.125    0.111    1.000    0.111  |
|  9.000    7.000    9.000    1.000  |
New CR:0.21407407407407417
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    0.111    0.111    0.125  |
|  9.000    1.000    2.000    4.000  |
|  9.000    0.500    1.000    5.000  |
|  8.000    0.250    0.200    1.000  |
New CR:0.12333333333333339
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    7.000    0.111    2.000  |
|  0.143    1.000    0.111    0.111  |
|  9.000    9.000    1.000    6.000  |
|  0.500    9.000    0.167    1.000  |
New CR:0.22296296296296308
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    1.000    5.000    2.000  |
|  1.000    1.000    2.000    8.000  |
|  0.200    0.500    1.000    8.000  |
|  0.500    0.125    0.125    1.000  |
New CR:0.32444444444444454
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    1.000    0.250    0.125  |
|  1.000    1.000    0.333    0.167  |
|  4.000    3.000    1.000    0.500  |
|  8.000    6.000    2.000    1.000  |
New CR:0.003703703703703625

=============TASK COMPARISON MATRIX (3)================
|  1.000    0.200    1.000    -2.000  |
|  5.000    1.000    0.500    2.000  |
|  1.000    2.000    1.000    3.000  |
|  -2.000    0.500    0.333    1.000  |
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
```

```
|  1.000    7.000    8.000    0.500  |
|  0.143    1.000    4.000    0.200  |
|  0.125    0.250    1.000    0.333  |
|  2.000    5.000    3.000    1.000  |
New CR:0.22037037037037027
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    9.000    0.143    9.000  |
|  0.111    1.000    0.111    0.200  |
|  7.000    9.000    1.000    9.000  |
|  0.111    5.000    0.111    1.000  |
New CR:0.31592592592592583
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    1.000    6.000    0.500  |
|  1.000    1.000    1.000    0.500  |
|  0.167    1.000    1.000    0.111  |
|  2.000    2.000    9.000    1.000  |
New CR:0.13555555555555543
Training with MLP to correct CR of the task comparison matrix...
Corrected Matrix after training
|  1.000    8.000    3.000    1.000  |
|  0.125    1.000    0.333    0.250  |
|  0.333    3.000    1.000    0.250  |
|  1.000    4.000    4.000    1.000  |
New CR:0.042222222222222175

Printing delta Matrix:
|  0.082    0.075    0.415  |
|  0.630    0.087    0.062  |
|  0.151    0.279    0.136  |
|  0.138    0.559    0.387  |

Printing PVS Matrix:
|  0.108  |
|  0.137  |
|  0.253  |
|  0.502  |
The selected job was : job 3

Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 3 to VM #0
0.1: Broker: Sending cloudlet 2 to VM #1
0.1: Broker: Sending cloudlet 1 to VM #2
0.1: Broker: Sending cloudlet 0 to VM #0
1.1: Broker: Cloudlet 2 received
1.1: Broker: Cloudlet 1 received
2.1: Broker: Cloudlet 3 received
2.1: Broker: Cloudlet 0 received
```

```
2.1: Broker: All Cloudlets executed. Finishing...
2.1: Broker: Destroying VM #0
2.1: Broker: Destroying VM #1
2.1: Broker: Destroying VM #2
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID     VM ID        Time     Start Time      Finish Time
    2         SUCCESS         2             1           1         0.1             1.1
    1         SUCCESS         2             2           1         0.1             1.1
    3         SUCCESS         2             0           2         0.1             2.1
    0         SUCCESS         2             0           2         0.1             2.1
Simulation finished!
```

6.3 RESULT

As the MLP was able to correct the CR of the inconsistent matrix in both the case, the final scheduling implementation was made by the algorithm as elucidated in chapter 4.

Above mentioned analysis in section 6.1 and 6.2 showed that the MLP was able to make matrix consistent even if the inconsistency is present in it. Moreover, it was a versatile solution that could adapt even in the case of missing entries. As the PVS matrix as a result of AHP in subsection 6.2.3 contained highest priority value of 0.502, task 3 (starting from index 0) was scheduled first and the datacenter broker sent cloudlet 3 to VM #0. Since 3 choices were present for resources (VMs), broker had the opportunity to schedule the cloudlets to different VM Ids. Similarly, other least prioritized tasks 2, 1 and 0 were respectively scheduled by the datacenter broker which showed the desired scheduling simulation.

Changes in the parametric value of α (learning rate) if taken less than 0.01, then step taken for convergence became small such that more time was taken for an algorithm to train the data. Whereas, there remained a chance to bypass the global optimum with more swing if the value of α was taken higher. Changes in the iteration epoch, maximum allowed error and default number of training sample directly affected the execution time for completion.

Hence, the use of Neural Network enabled us to make predictions on the missing values of the comparison matrices. More consistent matrices with corrected CR were obtained as the result of using trained MLP. In this way, the combination of AHP and MLP were useful for scheduling task execution in cloud.

CHAPTER 7
CONCLUSION AND FURTHER RECOMMENDATION

7.1 CONCLUSION

Cloud scheduling is the scheduling of virtualized resources and tasks over the large datacenter for different purposes such as energy aware considerations (minimizing power consumption), distributing tasks of large scientific data computation (big data) over the virtualized resources, fast and efficient execution than done in standalone computing environment, etc. Priority based scheduling are amongst concerns because some jobs should be serviced earlier than other jobs that can't stay for a long time in a system. Priorities may vary according to users for a particular task and they can also be assigned according to various parameters like bandwidth, completion time, memory, etc. Generally, higher priorities are given for most urgent tasks. But task priorities can also be assigned according to the desired consumption of resource provided by the cloud service providers. For example higher priorities can be given to those resources for which frequent access is desired from cloud users for their maximum utilization. Prioritizing tasks also helps to enable preemption in execution.

Analytical Hierarchy Process is effective for prioritizing cloud tasks and resources. Tasks are compared among themselves with respect to a particular resource, thereby producing a PVS vector, where the task with highest priority is then selected for execution in cloud. However, while comparing tasks and resources, there may remain some inconsistency or missing values in the comparison matrix given from users which can be successfully confiscated using trained multi-layer perceptron.

Thus, Final result of this dissertation is the consistent priority based schedule of tasks for cloud resources from the inconsistent user inputs of comparison matrix. Also, a versatile solution of multi-layer perceptron handling missing inputs and improving consistency ratio of the comparison matrix at the same time is devised.

7.2 FURTHER RECOMMENDATION

The possible number of missing entries in the comparison matrix remains a challenge as matrix becomes more sparse, the information that can be learned using neural network is lost. Also, as the dimension of the matrix increases, more time is taken by the neural network to learn the data. Therefore, some user restriction module can be implemented to limit the missing values entries from user input.

Also, dynamic monitoring of different states of resources whether they are busy or available, dead or alive and their auto updates can be accounted to make the scheduling system more influential. Migration of tasks from one cloud system to another is another open direction to adjoin the current system.

APPENDIX

SAMPLE CODE

File: AHPMLP.java

```
package com;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.linear.RealMatrix;
import org.apache.commons.math3.linear.RealVector;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.events.LearningEvent;
import org.neuroph.core.events.LearningEventListener;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.nnet.learning.MomentumBackpropagation;
import org.neuroph.util.TransferFunctionType;

public class AHPMLP {

    private static List<Cloudlet> cloudletList;
    private static List<Cloudlet> sortedCloudletList;
    private static List<Vm> vmlist;


    private static DecimalFormat df = new DecimalFormat(".###");
    private static int vmNo = 3;
    private static int cloudletNo = 4;


    //creating resource comparison matrix
    //input -2 for missing data here.
    private static double[] vmMatrixData = new double[]{3,5,-2};

    //creating task comparison matrices
    //input -2 for missing data here.
    //  private static double[][] cloudletMatrixData = new double[][]{
    //      {1/2.0, 4, 1/6.0, 4, 1/4.0, 1/5.0},
    //      {1/2.0, 1/3.0, 5, 1/2.0, 7, 9},
    //      {1/5.0, 1, 1/2.0, 1/2.0, 2, 3}
    //  };
    private static double[][] cloudletMatrixData = new double[][]{
```

```
        {1/2.0, -2, 1/6.0, 4, -2, 1/5.0},
        {1/2.0, 1/3.0, 5, -2, 7, 9},
        {1/5.0, 1, -2, 1/2.0, 2, 3}
    };



    private static int missingElementNoForResourceMatrix = 1;
    private static int[] missingElementNoForTaskMatrix = new int[]{2,1,1};
    private static int defaultnoOfTrainingSample = 30;

    //  private static int matrixOrder = 6;
    //  private static int missingNumberOfElements = 4;
    //  private static int noOfTrainingInputs = 30;
    //

    private static List<Vm> createVM(int userId, int vms) {

        // Creates a container to store VMs. This list is passed to the broker
        // later
        LinkedList<Vm> list = new LinkedList<Vm>();

        // VM Parameters
        long size = 10000; // image size (MB)
        int ram = 512; // vm memory (MB)
        int mips = 1000;
        long bw = 1000;
        int pesNumber = 1; // number of cpus
        String vmm = "Xen"; // VMM name

        // create VMs
        Vm[] vm = new Vm[vms];

        for (int i = 0; i < vms; i++) {
            vm[i] = new Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
            // for creating a VM with a space shared scheduling policy for
            // cloudlets:
            // vm[i] = Vm(i, userId, mips, pesNumber, ram, bw, size, priority,
            // vmm, new CloudletSchedulerSpaceShared());

            list.add(vm[i]);
        }

        return list;
    }

    private static List<Cloudlet> createCloudlet(int userId, int cloudlets) {
        // Creates a container to store Cloudlets
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

        // cloudlet parameters
        long length = 1000;
        long fileSize = 300;
        long outputSize = 300;
        int pesNumber = 1;
        UtilizationModel utilizationModel = new UtilizationModelFull();

        Cloudlet[] cloudlet = new Cloudlet[cloudlets];

        for (int i = 0; i < cloudlets; i++) {
            cloudlet[i] = new Cloudlet(i, length, pesNumber, fileSize, outputSize, utilizationModel,
utilizationModel,
                    utilizationModel);
            // setting the owner of these Cloudlets
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }

        return list;
    }

    private static Datacenter createDatacenter(String name){
```

```
// Here are the steps needed to create a PowerDatacenter:
// 1. We need to create a list to store one or more
//    Machines
List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
//    create a list to store these PEs before creating
//    a Machine.
List<Pe> peList1 = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into the list.
//for a quad-core machine, a list of 4 PEs is required:
peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

//Another list, for a dual-core machine
List<Pe> peList2 = new ArrayList<Pe>();

peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList1,
                new VmSchedulerTimeShared(peList1)
                )
        ); // This is our first machine

hostId++;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerTimeShared(peList2)
                )
        ); // Second machine


//To create a host with a space-shared allocation policy for PEs to VMs:
//hostList.add(
//      new Host(
//          hostId,
//          new CpuProvisionerSimple(peList1),
//          new RamProvisionerSimple(ram),
//          new BwProvisionerSimple(bw),
//          storage,
//          new VmSchedulerSpaceShared(peList1)
//      )
//  );

//To create a host with a opportunistic space-shared allocation policy for PEs to VMs:
//hostList.add(
//      new Host(
//          hostId,
//          new CpuProvisionerSimple(peList1),
//          new RamProvisionerSimple(ram),
```

```
        //          new BwProvisionerSimple(bw),
        //          storage,
        //          new VmSchedulerOportunisticSpaceShared(peList1)
        //      )
        //  );


        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x64";      // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
        double time_zone = 10.0;         // time zone this resource located
        double cost = 3.0;              // the cost of using processing in this resource
        double costPerMem = 0.05;        // the cost of using memory in this resource
        double costPerStorage = 0.1; // the cost of using storage in this resource
        double costPerBw = 0.1;          // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>();  //we are not adding SAN devices
by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList),
storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return datacenter;
    }

    private static DatacenterBroker createBroker(){

        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return broker;
    }

    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                "Data center ID" + indent + "VM ID" + indent + indent + "Time" + indent + "Start Time"
+ indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                Log.print("SUCCESS");

                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent +
cloudlet.getVmId() +
                        indent + indent + indent + dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent + dft.format(cloudlet.getExecStartTime())+ indent + indent +
indent + dft.format(cloudlet.getFinishTime()));
```

```
            }
        }

    }

    private static RealMatrix NNProcess(RealMatrix inputMatrix, int matrixOrder, int
missingNumberOfElements, int noOfTrainingSample ){
        int inputsize = matrixOrder * (matrixOrder - 1) / 2;
        int outputsize = inputsize;

        DataSet dataset = new DataSet(inputsize, outputsize);

        try {
            if(missingNumberOfElements > inputsize){
                throw new Exception("missing elements exceeded the input size.");
            }
            for (int i = 0; i < noOfTrainingSample; i++) {
                //step1 : preparing MLP input
                RealMatrix m = MatrixManipulations.createRandomInconsistentMatrix(matrixOrder,
missingNumberOfElements);
                double[] input = MatrixManipulations.extractUpperTriangularMatrixData(m);
                //MatrixManipulations.printMatrix(m);
                //System.out.println(Arrays.toString(input));
                double[] normalizedInput = MatrixManipulations.normalizeVector(input);
                //System.out.println(Arrays.toString(normalizedInput));

                //double[] denormalizedInput = MatrixManipulations.deNormalizeVector(normalizedInput);
                //System.out.println(Arrays.toString(denormalizedInput));
                //System.out.println("=====");

                //step 2 : preparing MLP output target vectors
                RealMatrix mTarget =  MatrixManipulations.createRandomInconsistentMatrix(matrixOrder,
0);
                double cr = MatrixManipulations.computeMatrixCR(mTarget);
                double min = cr;
                while (cr > 0.10) {
                    mTarget = MatrixManipulations.createRandomInconsistentMatrix(matrixOrder,0);
                    cr = MatrixManipulations.computeMatrixCR(mTarget);
                    if(cr<min) min = cr;
                    //System.out.println("CR:"+cr + "\t MIN:"+min);
                }
                double[] output = MatrixManipulations.extractUpperTriangularMatrixData(mTarget);
                //System.out.println(Arrays.toString(output));
                double[] normalizedOutput = MatrixManipulations.normalizeVector(output);
                //System.out.println(Arrays.toString(normalizedOutput));
                dataset.addRow(normalizedInput, normalizedOutput);

            } // endfor


            //neural network setup with momentum backpropagation
            MomentumBackpropagation mbpn = new MomentumBackpropagation();
            MultiLayerPerceptron mlp = new MultiLayerPerceptron(TransferFunctionType.TANH, inputsize,
30, outputsize);
            mlp.reset();
            mbpn.setMomentum(0.01);
            mbpn.setLearningRate(0.01);
            mbpn.setMaxIterations(10000);
            mbpn.setMaxError(0.001);
            mlp.setLearningRule(mbpn);
            mbpn.addListener(new LearningEventListener() {

                @Override
                public void handleLearningEvent(LearningEvent event) {
                    MomentumBackpropagation mbp =(MomentumBackpropagation)event.getSource();
                    //                  System.out.println(mbp.getCurrentIteration() + ". iteration |
Total network error: "
                    //                          + mbp.getTotalNetworkError());

                }
            });
            mlp.learn(dataset);
```

```
            // testing new input data now..

            //RealMatrix m1 =
MatrixManipulations.createRandomInconsistentMatrix(matrixOrder,missingNumberOfElements);
            //MatrixManipulations.printMatrix(inputMatrix);
            //System.out.println("Before CR:" + MatrixManipulations.computeMatrixCR(inputMatrix));
            double[] input1 = MatrixManipulations.extractUpperTriangularMatrixData(inputMatrix);
            //System.out.println(Arrays.toString(input1));
            double[] normalizedInput1 = MatrixManipulations.normalizeVector(input1);
            //System.out.println(Arrays.toString(normalizedInput1));
            mlp.setInput(normalizedInput1);
            mlp.calculate();
            //System.out.println("====================OUTPUT==============");
            double[] output1 = mlp.getOutput();
            //System.out.println(Arrays.toString(output1));
            double[] denormalizedOutput1 = MatrixManipulations.deNormalizeVector(output1);
            //System.out.println(Arrays.toString(denormalizedOutput1));
            RealMatrix m2 =
MatrixManipulations.completeMatrixWithLowerTraingularData(denormalizedOutput1, matrixOrder);
            //MatrixManipulations.printMatrix(m2);
            //System.out.println("After CR:" + MatrixManipulations.computeMatrixCR(m2));

            return m2;
        } catch (Exception e) {
            System.out.println("Exception in main.\n");
            e.printStackTrace();
            return null;
        }


    }

    public static void main(String[] args) {
        Log.printLine("Starting CloudSim simulation...");

        // creating 4 cloudlets and 3 vms
        try {
            // First step: Initialize the CloudSim package. It should be called
            // before creating any entities.
            int num_user = 1; // number of cloud users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false; // mean trace events

            // Initialize the CloudSim library
            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            // Datacenters are the resource providers in CloudSim. We need at
            // list one of them to run a CloudSim simulation
            @SuppressWarnings("unused")
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            //          @SuppressWarnings("unused")
            //          Datacenter datacenter1 = createDatacenter("Datacenter_1");

            //Third step: Create Broker
            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();


            //Fourth step: Create VMs and Cloudlets and send them to broker
            vmlist = createVM(brokerId,vmNo); //creating 3 vms
            //vmlist = createVM(brokerId,1);
            cloudletList = createCloudlet(brokerId,cloudletNo); // creating 4 cloudlets
            System.out.println(vmNo+" resources (VMs) and "+cloudletNo + " tasks (cloudlets)
successfully created as desired.");


            //creating resource comparison matrix
            RealMatrix vmMatrix =
MatrixManipulations.completeMatrixWithLowerTraingularData(vmMatrixData, vmNo);
            System.out.println("Resource comparison matrix created:");
```

```
            MatrixManipulations.printMatrix(vmMatrix);
            if(missingElementNoForResourceMatrix ==0)
                System.out.println("CR:"+ MatrixManipulations.computeMatrixCR(vmMatrix));
            //System.out.println("Lambda-Max :"+MatrixManipulations.lambdaMax);



            while(MatrixManipulations.computeMatrixCR(vmMatrix) > 0.1 ||
(missingElementNoForResourceMatrix > 0)){
                System.out.println("Training with MLP to correct CR of the resource comparison
matrix...");
                vmMatrix = NNProcess(vmMatrix, vmNo, missingElementNoForResourceMatrix,
defaultnoOfTrainingSample);
                MatrixManipulations.printMatrix(vmMatrix);
                System.out.println("CR:"+ MatrixManipulations.computeMatrixCR(vmMatrix));
                missingElementNoForResourceMatrix = 0;
            }



            //creating task comparison matrices
            RealMatrix[] cloudletMatrix = new RealMatrix[vmNo];

            System.out.println("\nFollowing task comparison matrices are created :");
            for(int i=0;i<vmNo;i++){
                cloudletMatrix[i] =
MatrixManipulations.completeMatrixWithLowerTraingularData(cloudletMatrixData[i],cloudletNo);
                System.out.println("\n=============TASK COMPARISON MATRIX
("+(i+1)+")================");
                MatrixManipulations.printMatrix(cloudletMatrix[i]);
                if(missingElementNoForTaskMatrix[i] == 0)
                    System.out.println("CR:"+ MatrixManipulations.computeMatrixCR(cloudletMatrix[i]));


                while(MatrixManipulations.computeMatrixCR(cloudletMatrix[i]) > 0.1 ||
(missingElementNoForTaskMatrix[i] >0)){
                    System.out.println("Training with MLP to correct CR of the task comparison
matrix...");
                    cloudletMatrix[i] = NNProcess(cloudletMatrix[i], cloudletNo,
missingElementNoForTaskMatrix[i], defaultnoOfTrainingSample);
                    System.out.println("Corrected Matrix after training");
                    MatrixManipulations.printMatrix(cloudletMatrix[i]);
                    System.out.println("New CR:"+
MatrixManipulations.computeMatrixCR(cloudletMatrix[i]));
                    missingElementNoForTaskMatrix[i] =0;
                }
            }




            //assuming all matrices are consistent now
            RealVector[] cloudletVector = new RealVector[vmNo];

            for(int i=0;i<vmNo;i++){
                cloudletVector[i] = MatrixManipulations.computeEigenVector(cloudletMatrix[i]);

            }
            int rows = cloudletNo;
            int cols = vmNo;
            double[][] deltaMatrixData = new double[rows][cols];
            RealMatrix deltaMatrix = MatrixUtils.createRealMatrix(deltaMatrixData);


            for(int i=0;i<cloudletVector.length;i++){
                deltaMatrix.setColumnVector(i, cloudletVector[i]);
            }

            System.out.println("\nPrinting delta Matrix:");
            MatrixManipulations.printMatrix(deltaMatrix);


            RealVector gamma = MatrixManipulations.computeEigenVector(vmMatrix);
```

```
            RealMatrix pvs =
deltaMatrix.multiply(MatrixUtils.createColumnRealMatrix(gamma.toArray()));

            System.out.println("\nPrinting PVS Matrix:");
            MatrixManipulations.printMatrix(pvs);

            System.out.println("The selected job was : job
"+(pvs.getColumnVector(0).getMaxIndex()+"\n"));


            double[] pvsData = pvs.getColumnVector(0).toArray();
            double[] sortedPvsData = pvsData.clone();
            Arrays.sort(sortedPvsData);

            sortedCloudletList = new LinkedList<Cloudlet>();

            for(int i=sortedPvsData.length-1;i>=0;i--){
                for(int j=0;j<pvsData.length;j++){
                    if(sortedPvsData[i] == pvsData[j]){
                        sortedCloudletList.add(cloudletList.get(j));
                    }
                }
            }



            broker.submitVmList(vmlist);
            broker.submitCloudletList(sortedCloudletList);


            // Fifth step: Starts the simulation
            CloudSim.startSimulation();

            // Final step: Print results when simulation is over
            List<Cloudlet> newList = broker.getCloudletReceivedList();

            CloudSim.stopSimulation();

            printCloudletList(newList);

            Log.printLine("Simulation finished!");


        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an unexpected error");
        }

    }

}
```

File: MatrixManipulations.java

```
package com;

import java.text.DecimalFormat;
import java.util.Random;

import org.apache.commons.math3.linear.EigenDecomposition;
import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.linear.RealMatrix;
import org.apache.commons.math3.linear.RealVector;

public final class MatrixManipulations {
    public static double lambdaMax = 0;
    // to represent fraction write 1/9.0
    // saaty's scale vector
    private static double[] SSV = new double[] { 1 / 9.0, 1 / 8.0, 1 / 7.0, 1 / 6.0, 1 / 5.0, 1 / 4.0,
1 / 3.0, 1 / 2.0,
            1, 2, 3, 4, 5, 6, 7, 8, 9 };
    private static DecimalFormat df = new DecimalFormat(".###");
```

```
    public static RealMatrix createRandomInconsistentMatrix(int order, int missingElementsCount) {
        double[][] matrixData = new double[order][order];
        for (int i = 0; i < order; i++) {
            for (int j = 0; j < order; j++) {
                if (i == j) {
                    matrixData[i][j] = 1;
                } else if (i < j) {
                    Random random = new Random();
                    // generates random int between 0 (inclusive) and
                    // 17(exclusive) ie 0-16 ie 17 elements.
                    int s = random.nextInt(17);
                    matrixData[i][j] = SSV[s];
                    matrixData[j][i] = 1 / matrixData[i][j];
                }
            } // end for j
        } // end for i

        // filling missing entries by -2
        int indexr[] = new int[missingElementsCount];
        int indexc[] = new int[missingElementsCount];
        for (int k = 0; k < missingElementsCount; k++) {
            Random random = new Random();
            int r, c;
            boolean flag;
            do {
                flag = false;
                r = random.nextInt(order);
                c = random.nextInt(order);
                //r should not be greater or equal to c index
                if(r>c || r==c) flag =true;
                else if (r < c) {
                    // r and c should not be in previous indexr and indexc list
                    for(int count=0;count<missingElementsCount;count++){
                        if(indexr[count]==r && indexc[count]==c) flag = true;
                    }
                }
            } while (flag == true);

            indexr[k] = r;
            indexc[k] = c;
            matrixData[r][c] = -2;
            matrixData[c][r] = -2;
        }

        return MatrixUtils.createRealMatrix(matrixData);
    }

    public static double[] extractUpperTriangularMatrixData(RealMatrix m) {
        double[][] matrixData = m.getData();
        int n = m.getColumnDimension();
        double[] data = new double[n * (n - 1) / 2];
        int count = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i < j) {
                    data[count] = matrixData[i][j];
                    count++;
                }
            }
        }
        return data;
    }

    public static RealMatrix completeMatrixWithLowerTraingularData(double[] inputData, int dimension)
{
        // creating square matrix with given dimension
        double[][] matrixData = new double[dimension][dimension];

        // count for traversing data
        int count = 0;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                if (i == j)
```

```
                matrixData[i][j] = 1;
            if (i < j) {
                matrixData[i][j] = inputData[count];
                count++;
            }
            if (i > j) {
                //checking missing elements in the inputData vector
                if(matrixData[j][i]!= -2){
                    matrixData[i][j] = 1 / matrixData[j][i];
                }
                else{
                    matrixData[i][j] = -2;
                }
            }

        }
    }

    return MatrixUtils.createRealMatrix(matrixData);

}

// normalizing saaty's scale in the range[-1,1]
public static double[] normalizeVector(double[] vector) {
    // double[] normalizedVector = new double[vector.length];
    for (int i = 0; i < vector.length; i++) {
        if (vector[i] != -2) {
            for (double value : SSV) {
                if (value == vector[i]) {
                    // linear conversion of range [1/9, 9] to [-1, 1]
                    // equation will be a*(1/9) + b = -1
                    // and another is a*(9)+b = 1
                    // solving these two equation will give a = 9/40 and
                    // b=-41/40
                    vector[i] = 9.0 / 40.0 * value - 41.0 / 40.0;
                } // end if
            } // end for
        } // end if
    }
    return vector;
}

public static double[] deNormalizeVector(double[] vector) {
    double[] deNormalizedVector = new double[vector.length];

    double difference = Double.MAX_VALUE;

    for (int i = 0; i < vector.length; i++) {
        difference = Double.MAX_VALUE;
        for (int j = 0; j < SSV.length; j++) {
            double val = 9.0 / 40.0 * SSV[j] - 41.0 / 40.0;
            if (Math.abs(val - vector[i]) < difference) {
                difference = Math.abs(val - vector[i]);
                deNormalizedVector[i] = SSV[j];
            }
        }
    }
    return deNormalizedVector;
}

public static double computeMatrixCR(RealMatrix m) {
    // doc get lambdaMax and n from matrix
    EigenDecomposition ed = new EigenDecomposition(m);
    int index = -1;
    int position = 0;
    double value = 0;

    // get the index of lambda-max
    for (double d : ed.getRealEigenvalues()) {
        index++;
        if (d > value) {
            position = index;
            value = Double.parseDouble(df.format(d));
        }
```

```
        }

        lambdaMax = value;
        //System.out.println("lambda-max :"+value);

        double lambdaMax = value;
        int n = m.getColumnDimension();

        // doc: compute consitency index C.I.
        double ci = (lambdaMax - n) / (n - 1);

        // doc: compute random index R.I from lookup table
        double ri = 0;
        switch (n) {
        case 1:
            ri = 0.00;
            break;
        case 2:
            ri = 0.00;
            break;
        case 3:
            ri = 0.58;
            break;
        case 4:
            ri = 0.90;
            break;
        case 5:
            ri = 1.12;
            break;
        case 6:
            ri = 1.24;
            break;
        case 7:
            ri = 1.32;
            break;
        case 8:
            ri = 1.41;
            break;
        case 9:
            ri = 1.45;
            break;
        case 10:
            ri = 1.49;
            break;
        case 11:
            ri = 1.51;
            break;
        case 12:
            ri = 1.48;
            break;
        case 13:
            ri = 1.56;
            break;
        case 14:
            ri = 1.57;
            break;
        case 15:
            ri = 1.59;
            break;
        default:
            ri = (1.98 * (n - 2)) / n; // ref:hamdy taha 'Operation Research'
                                       // 8th edition page 481-482
            break;
        }
        // doc: compute consistency ratio C.R.
        double cr = ci / ri;
        return cr;
    }

    public static void printMatrix(RealMatrix matrix) {
        for (double a[] : matrix.getData()) {
            System.out.print("|");
            for (double b : a) {
                b = Double.parseDouble(df.format(b));
```

```
                System.out.printf("  %.3f  ", b);
            }
            System.out.print("|\n");
        }
    }

    public static RealVector computeEigenVector(RealMatrix m) {
        EigenDecomposition ed = new EigenDecomposition(m);
        int index = -1;
        int position = 0;
        double value = 0;

        // get the index of lambda-max
        for (double d : ed.getRealEigenvalues()) {
            index++;
            if (d > value) {
                position = index;
                value = Double.parseDouble(df.format(d));
            }
        }

        RealVector v = ed.getEigenvector(position);

        // normalizing eigenvector
        double sum = 0;
        for (int j = 0; j < v.getDimension(); j++) {
            sum += v.getEntry(j);
        }

        for (int k = 0; k < v.getDimension(); k++) {
            v.setEntry(k, Double.parseDouble(df.format(v.getEntry(k) / sum)));
        }

        // System.out.println(v);
        return v;
    }

}
```

REFERENCES

[1] T. L. Saaty, "A scaling method for priorities in hierarchical structures," *Journal of mathematical psychology*, vol. 15, no. 3, pp. 234-281, 1977.

[2] T. L. Saaty, *What is the analytic hierarchy process?* Berlin Heidelberg: Springer, pp. 109-121, 1988.

[3] T. L. Saaty and K. P. Kearns, *Analytical planning: The organization of system*.: Elsevier, vol. 7, 2014.

[4] T. L. Saaty and L. G. Vargas, *Models, methods, concepts & applications of the analytic hierarchy process*.: Springer Science & Business Media, vol. 175, 2012.

[5] J. A., Gomez-Ruiz, M. Karanik, and J. I. Peláez, "Estimation of missing judgments in AHP pairwise matrices using a neural network-based model," *Applied Mathematics and Computation*, vol. 216, no. 10, pp. 2959-2975, 2010.

[6] Y. C. Hu and J. F. Tsai, "Backpropagation multi-layer perceptron for incomplete pairwise comparison matrices in analytic hierarchy process," *Applied Mathematics and Computation*, vol. 180, no. 1, pp. 53-62, 2006.

[7] J. I. Peláez and M. T. Lamata, "A new measure of consistency for positive reciprocal matrices.," *Computers & Mathematics with Applications*, vol. 46, no. 12, pp. 1839-1845, 2003.

[8] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[9] M. Armbrust *et al.,*"A view of cloud computing," *Communications of the ACM,* vol 53, no. 4, pp. 50-58, 2010.

[10] P. Harker and L. Vargas, "The theory of Ratio Scale Estimation: Saaty's Analytic Hierarchy Process," *Management Science*, vol 33, no. 11, pp. 1383-1403, 1987.

[11] E. Forman and S. Gass, "The Analytic Hierarchy Process – An Exposition," *Operation Research,* vol. 49, no. 4, pp. 469-486, 2001.

[12] D. Grzonka *et al*. "Artificial Neural Network support to monitoring of the evolutionary driven security aware scheduling in computational distributed environments," *Future Generation Computer Systems*, 2014.

[13] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informatics Journal*, 2015.

[14] Salman *et al*., "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems,* vol. 26, no. 8, pp. 363-371, 2002.

[15] S. Pandey, L. Wu, S. M. Guru and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 400-407, 2010.

[16] S. Bansal, B. Kothari, and C. Hota, "Dynamic task-scheduling in grid computing using prioritized round robin algorithm," *International Journal of Computer Science Issues(IJCSI)*, vol. 8, no. 2, 2011.

[17] L. Yang, C. Pan, E. Zhang, and H. Liu, "A new Class of Priority-based Weighted Fair Scheduling Algorithm," *Physics Procedia*, vol. 33, pp. 942-948, 2012.

[18] S. Patel and U. Bhoi, "Priority Based Job Scheduling Techniques In Cloud Computing: A Systematic Review," *International Journal of Scientific & Technology Research*, vol. 2, pp. 147-152, 2013.

[19] S. Ghanbari and M. Othman, "A priority based job scheduling algorithm in cloud computing," *Procedia Engineering*, vol. 50, pp. 778-785, 2012.

[20] D., Kou Ergu, G., Peng Kou, and Shi Y., "The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 835-848, 2013.

BIBLIOGRAPHY

[1] B. Golden, E. Wasil, P. Harker and J. Alexander, *The Analytic hierarchy process*. New York: Springer-Verlag, 1989.

[2] C. Bishop, *Neural networks for pattern recognition*. Oxford: Clarendon Press, 1995.

[3] C. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[4] H. Taha, *Operations research, an introduction*. New York: MacMillan Pub. Co., 1992.