IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727Volume 8, Issue 3 (Jan. - Feb. 2013), PP 01-08
www.iosrjournals.org

Realization of web hotspot rescue in a distributed system

Debashree Devi
Dept. Of Computer Science & IT, Assam DonBosco University, Guwahati-07, India

Abstract: *Web hotspot is considered as a serious problem in case of distributed systems. When the load in a website is suddenly increases, the situation is termed as web hotspot. This kind of situation can seriously degrade the performance of the website. One of the well known for this problem is DotSlash autonomic rescue system. In this paper we are trying to realize this web hotspot problem in a distributed system environment. Basically we need to distribute the load efficiently in various nodes so that the critical period can be handled. A cluster based approach is taken where the concept of content replication is used for efficient load distribution. Then a genetic algorithm based approach is proposed where we introduce a fitness function to find the best solution among a no of solutions (i.e. the nodes) with an aim to reduce delay and to increase the throughput.*

Keywords: *Autonomic resource computing, distributed system, Clustering, DotSlash rescue system, Genetic algorithm, Web hotspot.*

I. Introduction

From last few years, autonomic resource computing [1] became an emerging trend. It enables the computing systems to adapt to the changing environment without any human intervention i.e. .the computing systems can manage their resources autonomously in a way to satisfy the changes in the environment. In case of distributed systems, manual control and management of resources is not only time-consuming, costly and error-prone but also difficult to perform in some situations [3]. So there is a need for a self-managing system that can implement the self*-properties [2] i.e.it can self-monitor, self-configure, self- optimize, self-heal and self-protect itself for changes in the working environment. The evaluation of an autonomic system depends on how much it can adopt the self*- properties.

Web hotspot [3], also known as flash crowds is a serious problem in case of web-applications running distributed systems. Web hotspot problem occurs as a result of sudden increase in the request rate that a web site is experiencing. This kind of situation generally occurs when a no of users accessing the same website requesting the same task. As a result the load increases to a very high. Though the increase in the load lasts only for a short period of time, but it can seriously degrade the performance of the website.

Some of the solutions [3] to handle the problem of web hotspot; researchers are seeking solutions such as:- extending the server capacity,
-addition of more network bandwidth to the distributed system,
-by applying various caching mechanisms,
-reduction of content complexity under heavy load situation,
- Replication of contents and redirecting client requests etc.

To handle web hotspot, it is hard to predict about the increase in load like when it is going to increase, for how long it will last. So supplying of extra resources is not an efficient solution to this problem. That is the reason we need an autonomic system that can dynamically extend a website's capacity when needed, so that it can handle sudden increase of load.

When go for handling web hotspot problem, we have to face three major problems [3]:

1) Discovering of needed resources and then their allocation to the respective clients dynamically.
2) How to make the web hotspot handling process autonomic in order to quickly react to the sudden increase in load and hence maintain the website's availability even during a critical period.
3) How to overcome the bottlenecks in the web server infrastructure.

For solving the problem, websites both with static content and dynamic content, are considered. For static content website, the most common bottleneck is the access network bandwidth. And for dynamic content website, different applications may have different bottlenecks. We have to adopt different mechanisms to address different bottlenecks in case of dynamic content web page.

The remaining of the paper is organized as follows: section 2 describes the detailed related work done. Section 3 gives the proposed model architecture. Section 4 gives the parameters to be considered for evaluation of the proposed model. Section 5 describes the proposed AI-based approach towards the solution of the web hotspot problem. Section 6 covers the figures and tables given as the content and section 7 concludes the paper.

II. Related Work

Web hotspot being a serious problem as it degrades the quality of the website. To find a solution and then deploying the technique to find the result is very important.

Manual control on this whole process would surely affect the website quality, so we have to find an autonomic system that can find a solution immediately and appropriately.

DotSlash autonomic rescue system, given by Weibin Zhao [3] provided a solution to this problem. In order to solve the problem, DotSlash [3] enables the web site to create san distributed web server system on the fly, adaptive to the changing environment.

For web sites experiencing a request load that can't be solved by using a single server, then it is a common method to use multiple servers. Generally every distributed web server has a capacity planning and hence a fixed no of server is available to provide services. It works well only when the request load is relatively consistent.

But to solve problem like web hotspot where the request load suddenly increases and it is difficult to predict the peak load, we refer to dynamic allocation server capacity from a globally distributed server pool.

In the design model of DotSlash autonomic rescue system, a cost effective mechanism was applied to handle the increase request load. According to it, different web sites can form a mutual-aid community of web servers, so that in case of critical period it can use the spare capacity of other web sites in the community. In this design model, a web site is consists of a fixed no of origin servers, but also has a changing set of rescue servers taken from other sites. Based on the request load, the web server can allocates and releases rescue servers. To define the **origin servers**, it is the server which takes rescue services from other servers. And the **rescue server** is the one which provide rescue to other web servers.

In DotSlash overview, a web server is in one of the following states at any time:
-in SOS state when it takes rescue services from other servers,
- in RESCUE state when it provides rescue services to others,
- in NORMAL state otherwise.
These three states of a web server is mutually exclusive, i.e. when a web server is providing rescue services to others, it is not allowed to take rescue from other web servers at that time. This condition makes the DotSlash system simple to use and robust to maintain the scalability.

For offloading of client requests from origin servers to the rescue servers, DotSlash rescue system uses the HTTP Redirect and DNS Round Robin techniques. These two techniques are considered as the most common techniques to redirect client requests to different locations.

To illustrate the working of DotSlash rescue system, the whole process can be described simply by the following steps:

2.1 Dynamic virtual hosting

Dynamic virtual hosting enables a rescue server to serve the content of its origin server. A rescue relationship is established between the origin server and the rescue server to carry out the whole process. The rescue server assigns a unique virtual host name to the origin server; by using which the origin server perform HTTP redirect. For assigning virtual host name to the origin server, the rescue server generates it by adding sequence number components to its configured name, e.g. *vh<seqnum>host.name* for *host.name* and the <*seqnum*> increases with every single request.

In the DotSlash system, a rescue server works as a reverse caching proxy towards its origin server. As a result of it, an online on-demand policy is adapted. According to it a document is sent from its origin sever to the rescue server only when there is a request for the document and a cache miss occurs at the rescue server.

2.2 Request Redirection

Request redirection involves how client requests are transferred from the origin server to the rescue server. In DotSlash design model, the request redirection consists of two aspects:
a) Mechanisms for transferring client requests from the origin server to the rescue server,
b) Policies for selecting the appropriate rescue server.
Generally HTTP redirect and DNS round robin are the two common mechanisms for request redirection.

2.3 Workload Monitoring

When the load changes occur in a website, workload monitoring enables the web server to quickly react to the load changes. To determine the traffic in the network, the DotSlash design overview refers to monitor the outbound HTTP traffic in the web server. A no of parameters including lower & upper threshold for network utilization, maximum data rate, real data rate, real redirect data rate, allowed redirect data rate, redirect probability, network utilization, reference network utilization are defined to carry out this process.

2.4 Rescue Control

Rescue control enables a web server to manage its resource utilization by using rescue actions those are initiated autonomously based on the load condition. Generally to control the use of multiple resources, overload control actions are initiated based on some *rules.* Such as overload actions are initiated ***if*** any resource has high load. Again ***if*** the resources are lightly loaded *then* the under-load actions are initiated. DotSlash design overview used the DSRP (DotSlash rescue protocol) for executing the rescue control. DSRP is an application based request-response protocol that uses simple line text messages as communication medium.

2.5 Service Discovery

Service discovery allows dynamic collaboration of web servers and to learn about each other. In the DotSlash design overview, it used the Service Location Protocol (SLP) for service discovery. This SLP is flexible, light weight and powerful protocol.

The architecture for DotSlash autonomic rescue is shown in Fig.1 [3].
As shown in the Fig.1 DotSlash rescue system can be implemented in two parts:
Mod_dots and **Dotsd**. The web server used was the Apache as it is open source. The **Mod_dots** is a module of Apache which can be used for supporting different DotSlash functions like Client request processing etc. **Dotsd** is used to accomplish DotSlash function like service discovery, rescue control and management, dynamic DNS update. Both Mod_dots and Dotsd can access the control data structure through shared memory. BIND is the DNS server software used to set up a DNS domain “dot-slash.net”. All rescue servers can register their virtual host name in this domain by using dynamic DNS update. To setup a DotSlash service registry, mSLP DA is used, so that a track of all web servers can be maintained.

The DotSlash design overview we discussed so far is the DotSlash base system. The DotSlash base system is used for the static content website i.e. to handle web hotspot in case of static content website; this rescue system is very much efficient. But to handle web hotspot in case of dynamic content website, DotSlash have to overcome the bottlenecks in the application server and database server.

Handling of web hotspot in dynamic content website is a challenging task, because of two reasons:
1) Since generation of dynamic content web site consumes more CPU cycles, it is likely to be overwhelmed by web hotspot.
2) The different caching mechanisms that we can use for static content cannot be applied to dynamic content website.

So to handle web hotspot for dynamic content website, the concept of dynamic script replication was introduced. It allows the replication of requested scripts from the origin server by the rescue server and the caching those scripts locally at the rescue server.

Generally the typical dynamic content web architecture can be modelled in three-tier structure containing a web server, an application server and a database server, as shown in Fig.2 [3].

The web server is responsible for processing of the client requests through HTTP redirect and DNS round robin. The application server implements the business-logic and the database server is responsible for storing of the content. For generating dynamic content, several techniques can be used including PHP, Active Server Pages (ASP), Java Server Pages (JSP), Java Servlets, and Enterprise Java Beans (EJB) etc.

For discussing web hotspot for dynamic content, we can use a common configuration as LAMP (Linux, Apache, MySQL, and PHP). The motivation behind the concept of dynamic script replication is that running of scripts consumes a more no of CPU cycles and this CPU utilization often acts as a bottleneck in case of dynamic content website. We can simply define the working of dynamic script replication in a 8-steps process, as shown in the Fig.3.

For simplicity let us assume the origin server as www.origin.com and it can be denoted as S_o. And the rescue server as the www.rescue.com and can be denoted as *Sr.*
Now, the client C takes the following steps to retrieve the request *http://www.origin.com/serch.php?name=x* [3].
1) C makes an HTTP request to S_o.
2) S_o sends the HTTP redirect to C.
3) C makes an HTTP request to S_r.
4) S_r makes an HTTP request to S_o in case there is a cache miss for the script file search.php
5) S_o sends the script file to S_r and the S_r caches the file locally.
6) S_r runs the script file to access the corresponding database.
7) S_r gets the query result from the database.
8) S_r sends the query result to C.

By applying this dynamic script replication, the web hotspot problem is solved and the bottlenecks at the web server are successfully overcome. A distributed system architecture is followed so that it can be implemented to fulfil our requirements.

III. Load Distribution In Distributed System

Fig. 4 is showing the working of a simple load distribution process in a distributed system.
It is consists of three types of entities: **master**, **client** and **slave** [4].
Every computer that is taking part in the system is termed as the **Slave** [4]. The processing of user request is done in the slaves. It accepts the computational requests from the masters and then based on its availability, access the data from the data server and then upload the accessed data to the master. The communication medium for inter-slave communication is a massage passing interface.

The **master** maintains a database for slaves containing the resource availability, capacity, configuration of each of the slave. After getting the client request, first it checks the slave lib to know which the slaves available are at that time to process the request. According to the data, it sends a simple test massage command to each of the free/idle slaves. Based on the response, it performs the necessary read-write actions or redirects the requests so that the required data can be accessed [4] [7]. The requests are submitted by users through the client to the master. The scheduling of the received requests is done by the master so that two processes can't overlap.

The client acts as an interface between the user and the system [7]. User requests are sent to the master through clients to process it. For execution of clients two modes are there to follow; one is the batch mode and the other is the interactive mode [4].

In batch mode [4] the requests are sent as a project file which contains what are the resource required, steps to execute the request etc. The client can detach from the master and again in later can get linked to it to receive the processed request.

In the interactive mode [4], the user can dynamically create the requests, and can communicate with the already processed requests as well as with the currently processing requests. This kind of interactivity added an advantage to the system.

For storing the contents of different websites, the model provides a shared disk space as a local data server. This data server stores contents of different websites, executable input/output file for accessing of project file, data for request processing etc. Any web server or services like **ftp** which provides downloading of files and then uploading it, can be taken as the data server.

IV. Cluster Based Load Distribution

4.1 Clustering:

Clustering is a technique by which we can categorize objects into some groups based on some properties common to them. The goal of clustering is to determine the intrinsic grouping in an unlabelled set of data.

4.2 Cluster based approach

In the cluster based approach for load distribution, the nodes are categorized as strong cluster and weak cluster based on their weight vector which consists of the following parameter [5]:

1) Available capacity
2) CPU speed
3) Memory size
4) Access latency.

4.3 Content replication algorithm

Content replication can be considered as one of the technique to improve the scalability of the system by load distributing through multiple servers. As the presence of a replica of the request (e.g. a web page) in the near proximity of a client, reduces the rate of access latency [6].

For content replication, the contents are categorized to either Class I/ Strong Cluster and Class II/ Weak class based on their pattern of data accessing.

The most frequently accessed contents are clustered as Strong Class and the ones which are less frequently accessed are ranked as Weak class [5]. Then frequently accessed contents are replicated to more copies and requests/ queries are only broadcasted to the strong clusters. Hence this approach achieved low bandwidth consumption, reduced latency, reduced maintenance cost and fast query processing.

The question of how many replicas to be assigned to each request and how to place them in appropriate location, Replica placement algorithm is introduced to manage this issue.

Here it is assumed that every node stores one copy of its own content which it serves as a response to the client request. Moreover it also has enough disk space to store K replicated copies of content of other nodes. An authoritative *origin server* is defined which stores the update about the object, for every content replication. The object copy stored at the origin server is called as the *origin copy* and the other copies located at any remaining server are called the *replica*.
The replica placement algorithm consists of two steps:

a) Clustering the nodes

At first for each node, N_i, i=1, 2 ,......n , let
BW_i denotes the available bandwidth
SP_i denotes the CPU speed
AL_i denotes the access latency
MZ_i denotes the memory size.

1. The weight vector of node, Ni can be calculated as:
   $Wi=(BW_i + SP_i + MZ_i)/ AL_i$
2. Now a weight vector W= $\{S_i, W_i\}$ is formed where S_i denotes the node ids and W_i denotes their respective weights, sorted in ascending order.
3. Let $\{S_k\}$ be the set of strong clusters (0<= k<n), satisfies the condition $W_k >= \beta$, where β represents the minimum threshold value for weight.
4. Let the set $\{W_j\}= \{N_i\}- \{S_k\}$, denotes the set of weak clusters, on satisfying the condition $W_k < \beta$.

b) Replica placement

Let QS be the query server which stores the query / requests of each client.
Besides it stores the cluster information about each node along with their node id, either S(strong) or W(weak).

1. Let at time T_k, the set $\{Q_m\}$ denotes the query set generated by m- clients, for q-form, q=(n_{id}, c_{kwd}); where n_{id} denotes the node id of the client and c_{kwd} denotes the keyword of the requested content.
2. The query set is registered in the QS.
3. The requested content of the queries are categorized as either Class 1 or class 2, based on their access frequencies; i.e.

A query Q_j, for j<m, is categorized as Class 1
If $n(Q_j)>= A_{min}$
Else
Qj is in Class 2; where A_{min} denotes the minimum access threshold frequency.

4. Then the QS assigns the Class 1 contents to the Strong cluster nodes and the Class 2 contents to the weak cluster nodes respectively.
5. Then the QS transmits the replication pattern information to the origin server.
6. Based on the pattern information from QS, the OS performs the replica placement to different files and location. For every node the weight value, W_i is stored along with the content.
7. Then OS then broadcasts the replication information to the respective clients in the given format $\{N_{id}$, C_{id}, c1, c2.....}; where N_{id} denotes the node id, C_{id} denotes the cluster id and c1, c2...are denoting content database ids.

This is a simple replica placement algorithm, ensuring the delivering of replicas to appropriate locations.

V. Genetic Algorithm Based Load Distribution

Genetic algorithm is basically a search algorithm based on a natural selection and natural genetics. It combines the exploitation of past results with the exploration of new results to get an optimal solution [8]. By using survival of the fittest technique and a random but structured information exchange, genetic algorithm can implement innovative search and adaptation like human brain.

The structure of genetic algorithm can be described as a loop consists of a selection, followed by a sequence of crossover and mutation. In a loop the rate of crossover and mutation are fixed. The loop continues until it meets some stopping condition like execution time, optimal result etc.

Genetic algorithm based load distribution technique can be executed in seven steps [8]:
Let P= $\{p_1, p_2, p_3......p_M\}$ denotes the set of nodes/ computers contributing to the system.
Let T= $\{t_1, t_2, t_3.....t_N\}$ denotes the set of requests that have to be accessed.
Let F is a linear matrix where the element f_i (1<=i<=N) € F, is the node, targeted to send the request for processing.
Now our aim is to send each request, t_i € T, to f_i € P so that the total execution time can be minimized and fast processing of requests can be achieved.

Now the load at the server node can be given by:
Load (server) = $\sum a_{ij}$ (1<= i< no of requests already assigned to nodes) + $\sum a_k$ (1<= K < no of newly arrived requests).
Then execution time can be given as the maximum finishing time required accessing all the requests at the server.
i.e. Execution time = max (load (server))
To determine node utilization, i.e. how much a particular node, pi is involved in request processing, is given by:
Utilization (p_i) = load (server) / Execution time
So the average node utilization can be given by:
Avg. Utilization =
$\sum$ [utilization (p_i) for 1<= i <= no of available nodes] / total no of nodes.
Now going for the seven steps to be followed for execution, they are –

1) String representation

The string representation of individuals is called as the phenotype. In this GA based approach for load distribution, the string representation is an array of **n×m** matrix, where n is the no of requests and m is the no of nodes connected to the system.

2) Initial population

The population size is basically problem dependent. Here each solution, **i** is generated as every request, i is assigned to one of the nodes. This process is iterated till each of the requests gets assigned to some node.

3) Fitness function

The fitness function is the objective function, providing to check the fitness of every individual and also control the reproduction process. In load distribution context, factors such as execution time, node utilization are considered. By taking into account this objective, the fitness function for request T is given by:
f(T)= Avg. Utilization / execution time ; i.e. a fitter solution has a higher node utilization and less execution time.

4) Reproduction

It basically depends on the fitness value of the strings. The motive is that the string with higher fitness value will have a greater chance of transmitting to the next generation. As a result the searching time for the algorithm is reduced and the best solution can be obtained soon.

5) Crossover

It causes to exchange portions of strings between two or more strings. It also helps in finding the best solution. In the context of load distribution, if a request **t_i** is processed at one node and another request tj get processed at another node, then for a new request, any of the two nodes is chosen randomly and the new request is assigned to that node.

6) Mutation

It is basically used to change the genes in the chromosome. In the load distribution context, for a request, **t_i** to be accessed, any of the nodes is chosen at random and the request is assigned to the node.

7) Stopping condition

In this load distribution approach stopping conditions are can be:
- When a minimum execution time is obtained.
- All the requests get accessed.

VI. Figures And Tables

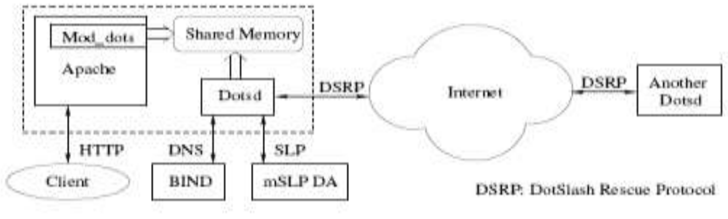

figure 1: dotslash software architecture

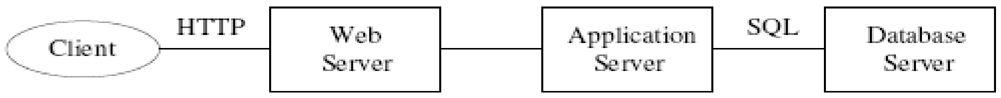

figure 2: three tier architecture for dynamic content website

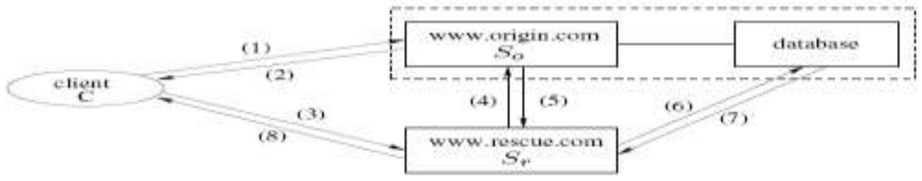

figure 3: working of dynamic script replication

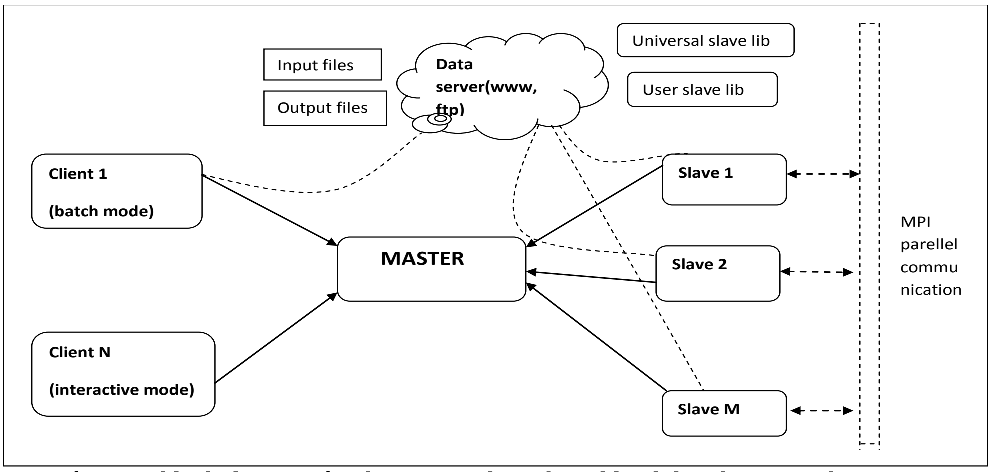

figure 4:block diagram for the proposed GA based load distribution architecture

VII. Conclusion

In this report, at first we have defined the problem of web hotspot in a web server community and have analysed the probable solutions to the problem. A simple overview of the DotSlash autonomic rescue system showed an efficient way to solve the problem.

For realization of web hotspot in distributed system we introduced two approaches: one is cluster based and another one is the Genetic algorithm based.

The cluster based approach describes how we can implement the concept of content replication for load distribution. In the GA based approach, the process is executed through seven steps, by defining a fitness function to evaluate the best solution.

Our future work involves the implementation of both the approaches in distributed system environment. The cluster based approach will be compared with the GA based approach and a comparison will be done. In my next paper a simulation study will be performed for both the approaches, considering the parameters- throughput, delay and for different population size and for a no of generations.

Throughput (bits/ sec): It defines the rate of successfully accessed data per second in the network.

Delay (sec): It is the time taken for accessing a client request and then successfully delivering accessed information to the client.

On the basis of the result, we can conclude which one will be an efficient approach for web hotspot realization in distributed system.

The implementation of master node may be inconvenient in some situations. We like to implement the idea of peer-to-peer (P2P) computing, where a no of master nodes are connected and can communicate with each other. Thus a single master node can act as a client to another master and as master to other clients as well, according to the situation.

References

[1] Jeffrey O. Kephart & David M. Chess, "The Vision of Autonomic Computing ", IBM Thomas J. Watson Research Centre ,*Published IEEE Computer Society, 2003.*

[2] Julie A. McCann, Markus Huebscher, "Evaluation issues in Autonomic Computing", Department Of Computing, Imperial College, London.

[3] Weibin Zhao, "Towards Autonomic Computing: Service Discovery and Web Hotspot Rescue", COLUMBIA UNIVERSITY, 2006.

[4] Zoran Constantinescu, "Towards an Autonomic Distributed Computing System", Norwegian University of Science and Technology, Norway, 2003.

[5] S.Ayyasamy1 and S.N. Sivanandam, "Cluster Based Replication Architecture for Load Balancing in Peer-to-Peer Content Distribution", *International Journal of Computer Networks & Communications (IJCNC) Vol.2, No.5, September 2010.*

[6] Stephanos, Routsellis-Theotokis and Diomidis Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies", *ACM Computing Surveys, Vol. 36, No. 4, December 2004, pp. 335–371.*

[7] Y. Jayanta Singh , Yumnam Somananda Singh, Ashok Gaikwad and S.C. Mehrotra, "Dynamic management of transactions in distributed real-time processing system", *International journal of Database Management system (IJDMS), Vol.2,No.2,May 2010.*

[8] Vinay Harsora and .Apurva Shah, "A Modified GeneticAlgorithm for Process Scheduling in Distributed System", *IJCA Special Issue on "Artificial Intelligence Techniques - Novel Approaches & Practical Applications" AIT, 2011.*

[9] Brighten Godfrey Karthik Lakshminarayanan Sonesh Surana Richard Karp Ion Stoica, "Load Balancing in Dynamic Structured P2P Systems", *IEEE INFOCOM* 2004.

.