EAI Endorsed Transactions
on Scalable Information Systems

Research Article EAI.EU

Selective lookup and intercommunication in grid (SLIG) adapting the distributed spanning tree to grid computing

J. Amudhavel [1,*], V.Agalya[2] , T. Dhivya[2], V.Vijayakumar[2], S. Keerthana[2], A. Dhamayanthi[2] and B.Bhuvaneswari[3]

[1]Department of CSE, KL University, Andhra Pradesh, India
[2]Department of Computer Science and Engineering, SMVEC, Pondicherry, India
[3]Department of Computer Science, Pondicherry University, Pondicherry,India.

Abstract

Computing consists of a network of heterogeneous computers, from which a virtual super computer is essentially formed. It displays immense potential as the various resources across large networks can be pooled to service many and be utilized by many, using the Internet from around the world. The potential for parallel CPU processing is one of the most attractive features of a grid. A perfectly scalable application will finish five times faster if it uses five times the number of processors. Application software as required by the users of the grid. Thus, the structure can be represented in layers, as implied by the grouping of grid components. Hardware, the bottom layer, would then contain a large number of heterogeneous resources and would be accessed by a limited number of users to ensure data privacy. The next layer would then consist of application software and tools that are useful for the users and which are domain-specific. In this research we analyzed the distributed and high-performance system in grid computing to provide the efficient resource discovery and message broadcasting. The Distributed Spanning Tree (DST's) implementation is altered and adapted to achieve better server load and message load distribution by a selective search and look-up mechanism in this proposal. In addition, a fault tolerance mechanism is also expressed in this contribution, as part of the DST's adaptation, such that if the system which is providing the service fails or leaves the grid environment, then the backup site will immediately take up the execution and recover the task.

Keywords: Grid computing systems, Distributed spanning tree, Resource selection.

Received on 06 February 2018, accepted on 17 May 2018, published on 19 June 2018

doi: 10.4108/eai.19-6-2018.154826

1. Introduction

The Computing capability has advanced exponentially in the last few years and has become a widespread demand in various disciplines, from the natural sciences to the humanities, with no end in sight. Networks play a critical role in latest technologies as well, with the rise of distributed computers and remote networked computing technologies, for serving high-performance applications and more. Thus grid computing was born, to take computing to the next level.

Grid Computing Systems: The main goal of a grid computing system is to create the illusion of a simple yet large and powerful self-managing virtual computer through a network of heterogeneous systems sharing resources. The potential for parallel CPU processing is one of the most attractive features of a grid. A perfectly scalable application will finish five times faster if it uses five times the number of processors. Other important grid computing characteristics include collaboration and resource pooling among a wider audience. Users of the

*Corresponding author. Email: info.amudhavel@gmail.com

grid can be arranged dynamically into a number of virtual organizations, each with different requirements. Additionally, sharing in the grid environment is not limited to only files but also includes other resources such as equipment, software, services and more. Thus, grid computing has convinced many governments to engage in significant investments with the view to develop pervasive grid computing infrastructures. Grids can theoretically be of any size and be scalable. Larger grids may have a hierarchical or some other type of topology. That is, computers locally connected together using a LAN may form a cluster of machines, which in turn form a hierarchy.

Furthermore, large-scale grids may span several different administration domains. Some of the initiatives with regards to grid computing include the *D-Grid* initiative in Germany, *Grid'5000* in France, *DAS* in the Netherlands, *PL-Grid* in Poland, *NAREGI* in Japan, *Open Science Grid* and *TeraGrid* in the USA [1]. Despite the initiatives mentioned above, a successful grid computing environment is still yet to be developed with a full plethora of users actively participating within the system. This is due to issues such as job scheduling, resource selection and management, fault tolerance and more, which pose a problem still today, inhibiting grid systems from full-fledged implementation. The intent of this paper is to propose a new mechanism that improves upon some of these areas. To enhance the performance of grid computing systems, we propose a selective search and discovery method, namely, Selective Lookup & Intercommunication in Grid (SLIG). This mechanism uses the Distributed Spanning Tree (DST) as its key foundation. It aims to implement an adaptation of the DST to a grid computing environment by introducing selective discovery based on resource availability, along with fault-tolerance and request recovery to the resource lookup. More details of this approach are provided in later sections of the paper. The following sections present how various researchers around the world have attempted to solve these issues and describe our own proposed model.

The paper is organized as follows: After the initial general description of a grid computing system and elaborates on the architecture of grid computing systems, followed by an exploration of the applications of a grid in Section. Section 2 deals with literature survey of some of the research presented by various academics worldwide, while Section 3 describes the existing system and its structure. Section 4 expounds on the proposed model along with its system design and architecture, after which Section 5 examines the system requirements. Finally, we conclude the paper in Section 6.

2. Related works

Distributed Spanning Tree Structures: One of the most significant breakthroughs is delivered by Sylvain Dahan et al [2] in their new, innovative topology, namely the Distributed Spanning Tree (DST). This structure can be applied to overlay networks, reducing message load and traffic, as well as eliminating bottlenecks, which are frequently seen in tree-based and hierarchical topologies. DST does this by combining the advantages of a tree topology with that of a graph topology, such that every computer acts as a leaf and every computer can act as the root of its own spanning tree. The DST structure is made up of nodes, which are each a complete graph of its children. By applying this new topology to any type of distributed network, including grid systems, the need for a master node is eliminated. Furthermore, message load is distributed across the network, with decreased chances of bottlenecks occurring. The distributed spanning tree takes advantage of the local-area inexpensive communications available to lower levels and makes use of long distance intercommunication judiciously. This structure can be easily realized due to its straightforward implementation, with a routing table being the only data structure needed. This structure has a few limitations, namely there's no recovery method in case of fault-occurrence when servicing a request, nor are resources grouped together semantically to achieve higher success rates with lower number of queries during a resource look-up. Furthermore, there is no defined method or mechanism to assess neither the server availability nor the processor availability of each computer.

Saeed Ebadi and Leyli Mohammad Khanli [3] proposed a new distributed and hierarchical mechanism for service discovery in a grid environment, which makes use of a layered architecture loosely based on the Distributed Spanning Tree. It contains the layers: client and service layer, institution layer, organization layer, domain layer, and the newly suggested root layer. The root layer is introduced to facilitate communication and transfer requests between domains, which had initially been a weakness of the layered architecture prior to their proposal. This mechanism, furthermore, ensures fault-tolerance through continuous service discovery such that several instances of the requested service are found, and pointers to these instances are made available in the institution layer. Along with fault-tolerance and recovery, another advantage offered by this distributed and hierarchical mechanism is its semantic grouping of resources, which reduces the number of queries required for a specific resource search. However, this approach poses speed issues as the lists of resources maintained at each layer increase broadly at every layer, resulting in very slow service and resource look-ups. Furthermore, there is a restriction placed upon the number of clients maintained by each institution, along with the number of institutions maintained by each organization, and so on, with a limit also on the number of domains that can be possibly present in the architecture.

A hybrid policy for fault tolerant load balancing in grid computing environments is presented by Balasangameshwara and Raju [4] in which the numerous unpredictable factors of a grid - namely the heterogeneity, varying network bandwidths, communication delay, resource availability, and unequal processor capabilities - are taken into account. This policy consists of an architecture of components that each takes care of a specific task, producing seamless execution and resource management within the grid environment. For example, a grid scheduler manages the jobs placed in the job queue and handles load balancing and site selection, collecting information from the sites about CPU utilization, CPU capability, and remaining memory. The load balancing decision maker processes the list of candidate sites and decides whether the job should be executed on a local or remote site. The fault detector and fault manager monitor the state of sites and use the proposed distributed fault tolerance policy to manage any failures via passive job replication and rescheduling. Thus, this hybrid policy offers optimal resource management and provides better fault tolerance but does not consider security in its proceedings.

Another resource management approach is introduced by B.T. Benjamin Khoo et al [5] called Multiple Resource Scheduling (MRS) algorithm. It employs a dimensions concept, presenting a computation and data perspective to the approach. It efficiently administers the resources in the grid into a minimal execution schedule by means of a Virtual Map, finding the best-fit resources for a job request while considering the different requirements of the job. Along with the Virtual Map concept, the concept of Resource Potential is used to determine the execution overheads and costs of communication between resource sites. Merits of this approach include improvement of performances up to 50% and excellent resource management and resource selection by using job fragmentation. Areas of improvement include introducing more dimensions to the approach, such as Quality of Service and economic considerations, and expanding the algorithm to take into account latency information. María Botón-Fernández et al [6] state an Efficient Resources Selection (ERS) model, achieving self-adaptivity in grid computing environment applications. This model works based on evolutionary computation algorithms. ERS produces improved application execution times and better throughput of the grid by avoiding resources that may be overloaded and/or inefficient during resource selection.

All these works utilize tree or graph topologies. However, as mentioned in the work by Sylvain Dahan et al, the innovative topology of the DST overcomes the drawbacks of traffic congestion, message overload and bottlenecks present in tree topologies. Therefore, although DST has only been applied to overlay networks so far, it would be immensely beneficial to apply it to grid computing systems and accelerate the intercommunications between the various computers and processing systems participating on the grid network. Nevertheless, DST also has its own disadvantages, namely no mechanism for fault-recovery nor for selective resource look-up based on semantic resource groupings. Thus, our proposed model involves re-designing the DST's routing table to include and indicate processor availability such that smart resource selection can be achieved, maximizing the load distribution and effective processor utilization within the grid. Furthermore, a recovery mechanism such as the job replica approach is looked to be implemented so that the selective look-up discovers a set of appropriate sites and uses one as the primary site for the job execution and another as a backup site, ensuring fault-tolerance.

3. Preliminaries

Distributed Spanning Tree (DST) optimizes flooding and search algorithms to get better performances. The idea behind this approach originates from the difference between tree and graph arrangements. In tree topologies, merely 2n number of messages for querying is needed for n nodes or computers. However, one of the biggest weaknesses behind tree structures is that they have high chances of experiencing bottlenecks due to the single access point nature of the nodes as each message can only reach the child node through the corresponding parent node. That is, congestion occurs in trees. Conversely, though graph topologies do not experience bottlenecks, they require more messages to be sent between the nodes. Therefore, both tree and graph topologies suffer drawbacks.

Furthermore, in the tree topology, intermediate nodes receive and forward messages to their children, while leaf computers simply wait to receive the messages, resulting in an imbalance of functions carried out by the various nodes. It is possible to overcome these disadvantages by creating a distributed spanning tree in which every computer acts as a root, as a leaf, and as intermediate nodes. The root node would then be distributed inherently among all the computers. Thus, "each computer is a leaf, and each non-leaf node is distributed through its children, and each computer is the root of its own spanning tree."

Structure of DST: DST is considered to have three different levels, namely: the Logical Level, Interconnection Level, and the Topological level. *Logical Leve***l:** Every leaf is a computer, and each parent node is the complete graph of its children. This is the DST's fundamental concept. The formation of these particular parent nodes constitutes the different stages of the DST (Figure 1) Thus, the root is distributed among the entirety of the child nodes.

Interconnection Level: In stage 1 of the DST, the corresponding logical level links are realized on the interconnection level by connecting leaf computers of a

parent node together such that they produce a complete graph. Then, in the higher levels, if there is a logical level link between two non-leaf nodes A and B, then "every computer that is a descendant of A opens a TCP/IP link with one random computer that is a descendant of B and vice versa."[2] Therefore, every child computer of node A can directly communicate with node B, and the same is true in the reverse. These TCP/IP links at each of the stages can be seen in Figure. 2. The DST's performance can be improved by considering how the computer pairs are established at each of the stages of the DST.

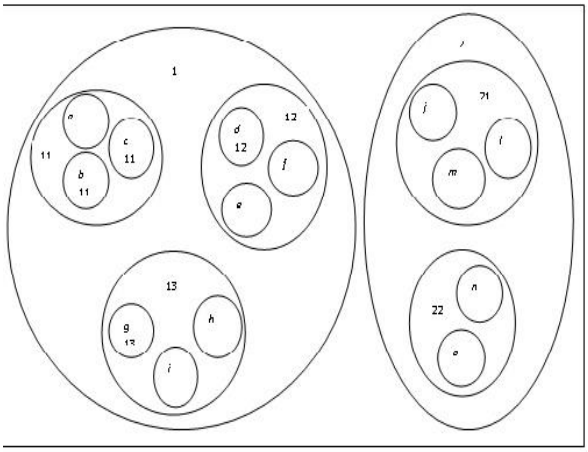

Figure 1. DST with nodes and names

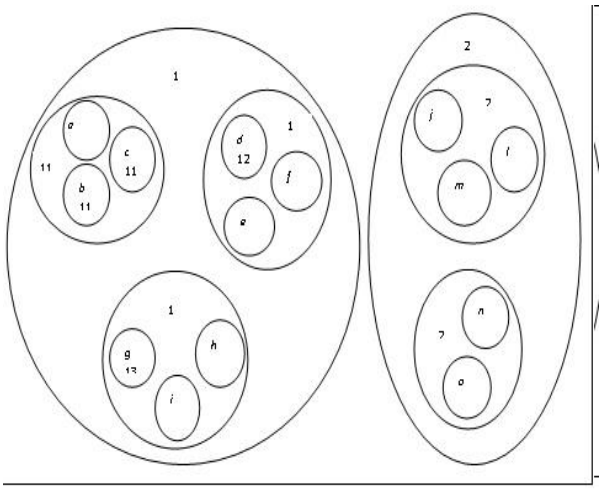

Figure 2. DST with nodes, names and IP

Topological Level: LANs are grouped to form sites such as university campuses or MANs and they make up the lower levels of the network hierarchy. These sites may be aggregated to form autonomous systems, which are then, in turn, linked together to produce the worldwide Internet. Message transfers become more expensive and less efficient the further up in the hierarchy of networks, from LANs to all the way up to autonomous systems and the Internet. DST groups together computers belonging to the same LAN into lower level nodes, thus efficiently utilizing local-area communications and reducing long-distance communications.

Implementation of this structure occurs on the interconnection level where all the TCP/IP links are made. A routing table is present in every computer, which manages these TCP/IP links. Two naming notations are first defined: Every leaf in the DST is a computer, having a unique IP address. IP addresses are represented by alphabets. For example, a, b, and c. At each stage of the DST, each computer that is a descendant of a node knows a representative computer for each of the brothers of that particular node. The routing table presents in each computer stores this knowledge and thus represents the local view of that computer in the DST.

An example of computer *e*'srouting table is shown in Table 1.Each row of the routing table contains data corresponding to different stages in the DST. The particular stage of the DST represented by each row is indicated by the first column of the table. Then, for the corresponding stage, the index of the child containing the current computer *e* is given in the second column. Note that the name of the leaf, i.e. the name of the current computer *e*, can be obtained by reading the second column top-down. The next few following columns specify which computer is used as the representative for each child at that stage so that when the computer *e* wants to contact a brother, it sends a message to the representative mentioned in its routing table

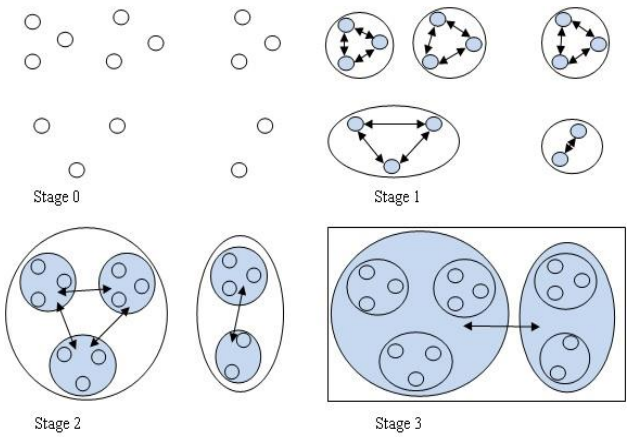

Figure 3. Logical level of DST

Similarly, computer m's routing table is displayed in Table 2.Hence, the only data structure needed to put the DST in place is the routing table at each computer.

Merits of DST: DST offers many advantages over the conventional network topology. The first being that no master node is needed, thus eliminating single points of access, which in turn reduces bottlenecks and congestion within the network. With the number of multiple paths present between each node in the DST, message load is also distributed across the various communication links

within the network. Furthermore, DST uses local area, inexpensive communications at lower levels, saving cost and time. And one of the major benefits is that the DST is easy to implement, requiring only a routing table to be present at each leaf node.

COMPUTER E'S ROUTING TABLE

Position		Representatives		
Stage	*Index*	*1*	*2*	*3*
3	1	*e*	*m*	-
2	2	*c*	*e*	*i*
1	2	*d*	*e*	*f*

COMPUTER M'S ROUTING TABLE

Position		Representatives		
Stage	*Index*	*1*	*2*	*3*
3	2	*c*	*m*	-
2	1	*m*	*o*	-
1	2	*j*	*m*	*l*

Drawbacks of DST: One of the disadvantages faced with the DST structure is that resources are not grouped together semantically. This may lead to more queries than is needed for a resource look-up. Another drawback is that there is no defined method to assess the server or processor availability of a computer in order to effectively utilize the nodes within a grid environment and thus maximize processor utilization. Furthermore, no recovery methods are indicated in case of any failures of servicing leaf nodes.

4. Proposed work

Formulationn of SLIG: Though grid computing is thought to have immense potential and powerful enough to create a virtual super computer, the implementation and the mechanics of such a system are still not clearly defined. One of the performance disadvantages is that the various heterogeneous processors and computers, including local storage areas, do not have high-speed network connections. In many cases, the issue of trust also arises as the participating nodes on the grid may abuse the access being granted by interfering with other operations within the system, as well as disturbing stored data, breaching secure, private and confidential information.

Furthermore, during each resource request, factors such as application availability, server load, location of processed data, and others must be considered to find the best-suited server. Thus, the connections between the nodes in a grid system are costly. In current middleware systems, tree based architecture is used to achieve communication within a grid system. A filter may be implemented in the tree structure to achieve the best-suited server look-up, where each server delivers its parent node an inventory of its applications. This way, each parent forwards its descendants' inventory of applications to its own parent. Thus, request queries need not be sent to branches not containing the desired applications. This tree architecture allows for better scalability in consideration of request frequency. However, it also has the disadvantage of being overloaded by request queries and look-ups, increasing the chances of a bottleneck occurring at intermediate nodes.

To overcome the above-mentioned disadvantages, a better suited topology other than a tree structure may be utilized, namely the DST. After all, the client doesn't need to contact the best server on the network; it only needs to approach a good server. This new topology is suitable for linking self-organized networks, eliminating bottlenecks, and splitting the message load across various nodes in the grid. Another major drawback in the tree structure is that though the search is initiated by the root node, and the message is redirected through the transitional nodes to the leaf nodes,

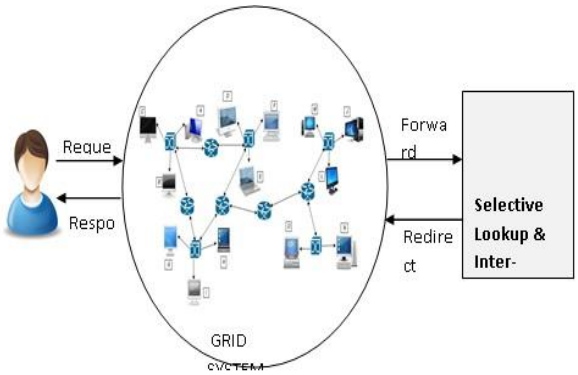

Figure 4. Architecture of selective lookup and interconnection of Grid

Since most of the computers are usually terminal nodes or leaves, an imbalance occurs as the leaves don't do anything in terms of message forwarding. However, the distributed spanning tree easily overcomes this as each of the nodes in the DST may act as a leaf, root, or as an intermediate node. Additionally, the root node is dispersed among all the computers; therefore, each computer may act as the root of its own spanning tree. To achieve inexpensive network connections between the nodes of a grid computing system, DST groups together computers in same LANs to form nodes at lower stages, encouraging more message transfers across a LAN than through sites or autonomous systems. This effectively takes advantage of the locally available efficient and cheap connections. To achieve the DST topology, the only data structure needed is a routing table in each computer's memory. Therefore, the distributed spanning tree is a well-suited

topology that can be used to maximize the efficiency of communication in a grid computing system.

TABLE I. PROCESSOR AVAILABILITY INDEX

Representative	Stage 1	Stage 2	Stage 3
a	50	57.7	55.6
b	47	57.7	55.6
c	76	57.7	55.6
d	43	47.3	55.6
e	76	47.3	55.6
f	23	47.3	55.6
g	76	61.7	55.6
h	86	61.7	55.6
i	23	61.7	55.6
j	76	65.3	62.9
l	86	65.3	62.9
m	34	65.3	62.9
n	87	60.5	62.9
o	34	60.5	62.9

However, DST itself has several weaknesses like no fault-recovery nor selective resource lookup mechanisms are present. With a selective resource search, the number of nodes queried for the needed resource can be reduced. Selective resource lookup can be implemented by grouping together resources semantically. However, to adapt the implementation to the grid environment, let's consider the characteristics of a grid which make it successful. One such characteristic is processor utilization or load distribution.

Uneven arrival patterns of resource requests and unequal computing capacities result in some nodes being overloaded while others get under-utilized. Thus, it can be said that a grid is successful if it evaluates well based on its processor utilization, where the load is equally spread among all the nodes on the grid, optimizing the utilization, throughput, and response of the grid as a whole. To achieve this fair distribution of load, where the difference between the heaviest-loaded node and the lightest-loaded node is minimized, we propose a selective resource discovery mechanism, namely SLIG. The architecture of this mechanism is shown below in Fig. 4. SLIG expounds on the DST, using a modified data structure that indicates processor availability to allow for smart and selective resource location. This reduces the message load while achieving fairer load distribution by assigning the arriving resource requests to those nodes which are under-utilized.

A fault recovery mechanism is broached in SLIG to overcome another one of the limitations in the original DST. It involves using a job replica approach such that the selective look-up discovers a set of appropriate sites within the grid and uses one as the primary site for the job execution, with another as the backup site. Thus, in case of a failure or an incapacitating issue with regards to the primary site, the client node can still be able to redirect the resource request to the backup site for successful execution. These variations on the original DST which comprise our new model, SLIG, are further seen in depth in the following section. Structure of SLIG: Expanding on the DST, SLIG uses one other data structure called a processor availability index. It is a global table which consists of and maintains values indicating the processor availability of each of the nodes, at each stage of the DST[7]. At stage one, the index values are calculated for each of the leaf computers based on their local resource usage as well as their resource utilization in service to remote requests. At higher stages of the DST, the index value of each logical node is determined by taking into account the index values of all the children of that node. That is, to make the SLIG mechanism's implementation simpler, the average of all the processor availability values of the children of a particular non-leaf node is found, and this resulting average is then assigned as the processor availability index for that particular non-leaf node. This process of finding the average index value for each non-leaf node is continued in the same way, regardless of however many number of stages there are within a DST.

The processor availability index varies in size depending upon the number of stages present in the DST. The first column within the table lists all the representatives or leaf computers present within the grid system. The other columns within the table each contain the index values representing the processor availability for each of the nodes at each stage. The number of columns present within the table that indicate the processor availability at each stage is directly proportional to the number of stages the DST contains.

An example of this processor availability index is displayed in Table 3. This table contains the processor availability values for each stage of the DST for the computers *a* through *o*. Since we are working with this sample set of computers, and the resulting DST contains three logical stages, the corresponding processor availability index also contains three columns, one for each of these stages. As shown in Table 3, at stage 1, the availability values for each of the individual leaf computers, *a* through *o*, is listed, where each of these values is calculated according to the respective individual systems based on their local resource utilization and resource employment to service remote requests. Then at stage 2, based on how the nodes are logically separated, the average processor availability value is found for each of the non-leaf nodes. Similarly, the same is done for the third stage.

Thus, at stage 2, since child nodes *a*, *b*, and *c* form the logical node labelled *11*, the average for the processor availability values of *a*, *b*, and *c* in stage 1 is found and assigned as the index value for all three child nodes, *a*, *b*, and *c* in stage 2. Likewise, the same process is done for the other nodes in stage 2, namely the nodes labelled *12*, *13*, *21*, and *22*. Then at stage 3, for instance, the average of the stage 2 index values of the children in the nodes *11*, *12*, and *13* is determined and allocated as the index value for the node labelled as *1* in stage 3. The same method is performed correspondingly for node *2* in stage 3. This index table needs to be maintained throughout the functional lifetime of the grid environment such that the values within the table are updated accordingly in each of the stages of the DST as both the local and remote resource requests are serviced in a dynamic fashion.

Working of SLIG: The mechanism behind SLIG works off of both the DST routing table and the global processor availability index. This selective resource discovery method is initiated when a resource request is received by any one of the leaf nodes within the grid. Due to the DST structure of the grid, this specific leaf node acts as the root of its own spanning tree. Let this particular leaf node be referred to as *R* from here on. The process of finding the best, suitable sites to service the request begins with *R* consulting its own DST routing table. The node *R* determines who its representatives are in its sibling nodes, starting at the highest stage *n*, initially. Once its representatives are found, *R* checks the processor availability index to ascertain which of these representatives have the higher index value, reflecting that computer's idleness or resource availability. This representative, let it be denoted as *P*, is then chosen to receive a copy of the request, forwarded from node *R*.

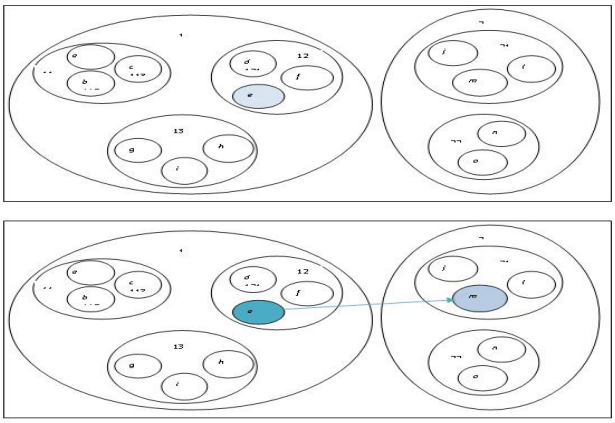

Figure 5. Stages of SLIG

Then, the next logical stage or level of the DST, *n-1*, is considered and node *P* refers its own routing table to determine its representatives at this level, finding the best representative by comparing the processor availability index of each and forwarding the copy of the request to the selected representative node. Thus, each of the successive lower stages of the DST is considered, and the process is repeated in the same manner for each of the subsequently selected nodes until the lowest stage of the DST, stage 1, is reached. At this level, the best suitable receiving sites are selected in the same way, yet at this point, a set of minimum two sites are chosen to ensure fault-recovery, where the first best site is assigned as primary and the other(s) may be delegated as secondary or backup sites. Hence, if the primary node fails, *R* can always redirect the resource request to the lastly discovered secondary site.

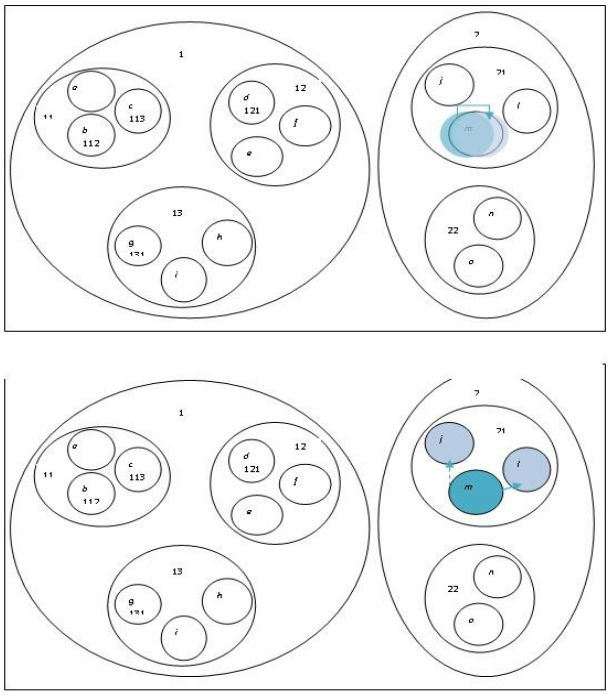

Figure 6. SLIG stages

Figure. 5 to Figure. 6 illustrates the working of SLIG, as explained just prior. As shown in Fig. 5, the leaf computer *e* receives a resource request and initiates SLIG by referring its routing table (see Table 1) to find its representatives at the highest level, i.e. at stage 3. Once its representatives are determined, namely itself and *m*, it checks the processor availability index (refer Table 3) to determine which of the representatives have the higher resource availability. Here, *m* has the larger index value at stage 3. Therefore, a copy of the resource request is forwarded from *e* to *m*, as shown in Fig. 6. Once forwarded, the SLIG mechanism considers the next stage, stage 2, of the DST. At stage 2, *m* refers its own routing table (see Table 2) to find its representatives at this stage, namely itself and *o*, before matching them with the processor availability index to find the best receiver, which is *m* itself. Thus, *m* forwards the copy of the request to itself, which can be seen in Figure. 6, and prompts SLIG to consider the next stage of the DST.

In the next level, stage 1 of the DST, *m*'s representatives are found to be *j*, *l*, and *m* itself. After

checking the processor availability index, it can be found that computer *l* has the highest resource availability, with computer *j* coming in second largest. Therefore, *l* is chosen as the primary site to service the resource request originally from computer *e*, while *j* is selected as a secondary backup site. This is shown in Figure. 6. Thus, SLIG ensures fault-recovery, in case of any failures with the servicing leaf nodes, along with selective resource discovery based on resource availability, to maximize performance of the grid computing system.

5. Experimental Evaluation and Analysis

A real-time grid computing system involves a network of computers and hardware devices like any other conventional interconnecting network. Thus, some of the basic hardware requirements, as would be seen typically are: To create a network, many computers are required. However, in the grid environment, these computers may be of heterogeneous quality and different in their processing capabilities and power. In SLIG, each computer acts as a leaf node on the grid. During operation, each node may form its own distributed spanning tree, redirecting resource requests to other leaf computers, along with servicing any received resource requests. Every single computer must have a Network Interface Card (NIC) embedded within as part of the hardware.

NIC is required to connect devices to each other within a network. Some of the other standard devices used to interconnect nodes within a network include switches, bridges, and routers. Thus, some of the hardware components required to build a real-world grid environment were seen. However, our contribution of a selective resource lookup mechanism, that is, adapting the DST to a grid environment, involves implementing the mechanism by means of a simulation. Therefore, some of the software requirements for constructing the simulation are expressed below. OMNeT++ is an open-source, portable, and an object-oriented network simulation framework available for academic and non-profit use. It allows for recreation of discrete events in a network. It contains a generic architecture which can be used to model wired/wireless communication networks, various network protocols, queuing networks, multiprocessors, distributed hardware systems, and other entities that communicate by message exchanges. It can also be used to validate architectures, and evaluate performance aspects of the modelled system. OMNeT++ provides the infrastructure and tools necessary for writing simulations. This framework's architecture permits models to be constructed from reusable modules and components. Gates (also called a sports) are used to connect modules with each other, and these modules may be linked together to create compound modules with unlimited depth of nesting. Module behaviours, along with the model topology, can be customized. Modules at the lowest level of hierarchy, called simple modules, encapsulate model behaviour and are programmed using the OMNeT++ simulation library, in C++.

OMNeT++ simulations can be executed in graphical user interfaces for demonstration purposes, as well as in command-line user interfaces for batch-execution. Parallel distributed simulations are supported also.

6. Conclusion

In grid computing systems, a large number of heterogeneous systems are interconnected with different network bandwidths and unequal processor capabilities. Using tree topology in grid environments leads to bottleneck and message overload issues. On the other hand, using graph topology leads to sending numerous messages to all other nodes to find the requested service. Therefore, to achieve better message load distribution and reduce bottlenecks in a grid computing environment, an adaptation of the Distributed Spanning Tree (DST) structure is proposed. This adaptation of the DST hopes to overcome fault-tolerance issues and improve resource management through job replica approach and selective resource look-up respectively. Thus, once the resource request is received, the selective lookup mechanism will find both primary and secondary sites suitable for job execution, ensuring request recovery in case of node failure, and effective processor utilization in the grid.

References

[1] Uwe Schwiegelshohn, Rosa M. Badia. (2010) Perspectives of grid computing. *Future Generation Computer Systems.*

[2] Sylvain Dahan, Laurent Philippe, and Jean-Marc Nicod. The distributed spanning tree structure. *IEEE Transactions on Parallel and Distributed Systems*. 2009 December.

[3] Saeed Ebadi, Leyli Mohammad Khanli. (2011) A new distributed and hierarchical mechanism for service discovery in a grid environment. *Future Generation Computer Systems*.

[4] Jasma Balasangameshwara, Nedunchezhian Raju. (2012) A hybrid policy for fault tolerant load balancing in grid computing environments. *Journal of Network and Computer Applications*.

[5] Benjamin Khoo, B.T. Bharadwaj Veeravalli, Terence Hung, C.W. Simon See. (2007) A multi-dimensional scheduling scheme in a grid computing environment. *Journal of Parallel and Distributed Computing*.

[6] María Botón-Fernández, Miguel A. Vega-Rodriguez, Francisco Prieto Castrillo. (2014) Self- adaptivity for grid applications. An Efficient Resources Selection model based on evolutionary computation algorithms. *Parallel Computing*.

[7] Luis Ferreira. (2003) Introduction to grid computing with globus. *ACM digital library*.