Decentralized Grid Management Model Based on Broker Overlay

Abdulrahman Azab[1], Hein Meling[1]

[1] Dept. of Computer Science and Electrical Engineering , Faculty of Science and Technology, University of Stavanger,
4036 Stavanger, Norway
{abdulrahman.azab, hein.meling}@uis.no

Abstract. Grid computing is based on coordinated resource sharing in a dynamic environment based on multi-institutional virtual organizations. Data exchange, and software component deployment, are challenging problems in the filed of Grid computing. This is due to the decentralization of Grid systems. Building decentralized Grid systems with efficient resource management and software component mechanisms is a need for achieving the required efficiency and usability of Grid systems. In this work, a decentralized Grid system model is presented. In this model, the system is divided into virtual organizations each controlled by a grid broker. An overlay network of Grid broker is responsible for global resource management and managing deployment of software components. Experimental results show that, the system achieves accepted performance with various loads of software components, and broker failures.

Keywords: Grid computing, Peer-to-peer computing, Virtual organization management.

1 Introduction

Grid computing is the computing paradigm which is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [1]. A virtual organization (VO) can be defined as a collection of IT resources in which each participant can acquire or provide IT services from/to other resources inside/outside the organization [2]. Cloud computing refers to services delivered through the Internet and the hardware resources and systems software in datacenters that provide those services [3]. Some researchers consider cloud computing as a new name of Grid computing. Actually there is no consensus on what a Cloud is [4]. It can be considered as the hardware resources and software provided by datacenters [3]. The main aspect in cloud computing is transparency, while in Grid computing is coordinated resource sharing. The common aim of both paradigms is to achieve rapid decrease in hardware cost and increase in computing power and storage capacities [4]. To build a distributed computing structure which fulfills the requirements of both Grid computing and Cloud computing is a big challenge. It requires implementing a multi-VO model in which regular participants don't have to

worry about the complex structure of the entire system. Instead, the system complexity should be transparent to the users who should establish connection with only one or very few nodes within the same VO.

Most of the famous Grid solutions (e.g. Condor [8], Globus [9], and BOINC [10]), implement single virtual organization model. The Grid is composed of a set of regular nodes and one or more nodes responsible for resource management and scheduling. SZTAKI [11] project is based on BOINC, but it supports inter-VO communication by permitting sub-Grids to be members of the main Grid system. In this technique, servers of sub-Grids will communicate with the server of the super-Grid for data and task exchange. But still, there is no base for communication between equal VOs. Alchemi [12], supports cross-platform communication with Alchemi based Grids and Grid systems built on different architectures (e.g. Globus). This communication is carried out through a cross-platform manager.

This paper presents a decentralized multi-VO Grid resource management model based on hybrid peer- to-peer communication [5]. The proposed model is designed to be tested on HIMAN [6, 7], a pure peer-to-peer computational Grid middleware. The Grid system is divided into a set of virtual organizations. Each VO contains a set of regular participants and one *Broker*. Rules of resource sharing within a virtual organization are well known by each participant and controlled and managed by what so called Grid resource Brokers. A Grid resource broker is responsible for receiving requests for grid resources, comparing the requirements in each request with the specifications of the available resources, and direct requests to suitable resources. Brokers from different domains construct a cooperative collection called, *Broker Overlay*, which represents the Cloud in the system. The idea is to provide each participant with the ability to offer and to claim computational resources. In addition, the complexity of the system is hidden from regular participants in the broker overlay, as each participant interacts only with the attached broker.

The rest of the paper is organized as follows: Section 2 describes the system model. Section 3 describes the resource information exchange mechanism. Section 4 describes the service deployment model. Section 5 describes the broker failure handling mechanism. Section 6 described the simulation model implemented for running the experiments. Section 7 describes the performed experiments and discussion of the results. Section 8 presents conclusions.

2 System Model

To build a computational Grid system as a collection of virtual organizations, two essential issues has to be taken into account. First, local resource management and task scheduling within each virtual organization (i.e. Intra-VO model). Second, global resource management and task scheduling (i.e. inter-VO model). The main aim is to fulfill the system requirements by allocating submitted computational tasks to suitable resources together with achieving load balancing locally, in the virtual organizations, and globally, in the Grid. The proposed model is based on global resource sharing based on collaboration of virtual organizations. Each virtual organization is set up as a

domain. Each domain consists of one domain controller (i.e. Broker), and a collection of regular nodes. Components of the grid system are defined as:

2.1 Service

A service is a software component which performs a specific task, and has four execution parameters: 1) Required CPU, the computational power required for running the service. 2) Required Memory, the memory size required for running the service. 3) Expiration Time, the amount of time to wait before the allocation is expired. 4) Creation Time, the time at which the service is created for allocation. 5) Deployment attempts, the maximum number of attempts to deploy the service before it is expired.

The Grid management system is responsible for deploying each Grid service on a suitable Grid node which available resource matches the requirements of the service. The need for the *expiration time* and the *deployment attempts* parameters will be described in sec. 4.

2.2 Regular Node

Regular nodes represent regular participants in the Grid. Each regular node can be a member of one virtual organization, and can submit and/or run a service. A regular node is also responsible for periodically sending information about the current available resource state of the node to the attached grid broker. Each regular node has two resource parameters: 1) Available CPU, which refers to the available computational power in the node, and 2) Available Memory space. Regular is equivalent to Peer in HIMAN, which contains two components: *Worker* (W), which is responsible for task execution, and *Client* (C), which is responsible for task submission[1] [7].

2.3 Broker

A broker node works as a virtual organization (i.e. domain) controller, and also can work as a regular node in addition. It is responsible for: 1) Allocating Grid services to suitable nodes[2], and 2) Storing information about the current state of local Grid nodes (i.e. in the local virtual organization), as well as global Grid nodes in other virtual organizations.

---

[1] In HIMAN, the client component is responsible also for task allocation [7]. In this model, it is carried out by the broker.

[2] A suitable node for a service is elected by performing a matchmaking process between the service requirements and the available resources of attached Grid nodes [13].

2.4 Virtual organization

A virtual organization is an overlay collection of Grid nodes, which may be allocated in different regions and members of different organizations. Each VO is composed of one Grid broker and a number of regular nodes. A VO is structured as an overlay star topology based network, so that; communication is between the broker and regular nodes. No local communication between regular nodes within the same virtual organization.

2.5 Broker Overlay

Broker overlay is the overlay network between brokers within which communication and data exchange between different virtual organizations is performed. In this model, four overlay communication topologies are implemented for the broker overlay: Ring, Hyper cube, Wire K out, and Fully connected. Based on the communication topology, each broker will have a number of neighbor brokers, those brokers with which direct communication can be established. Fig. 1 shows the structure of the Grid as a collection of virtual organizations. Each broker is a member of two overlay networks; the broker overlay and a virtual organization.

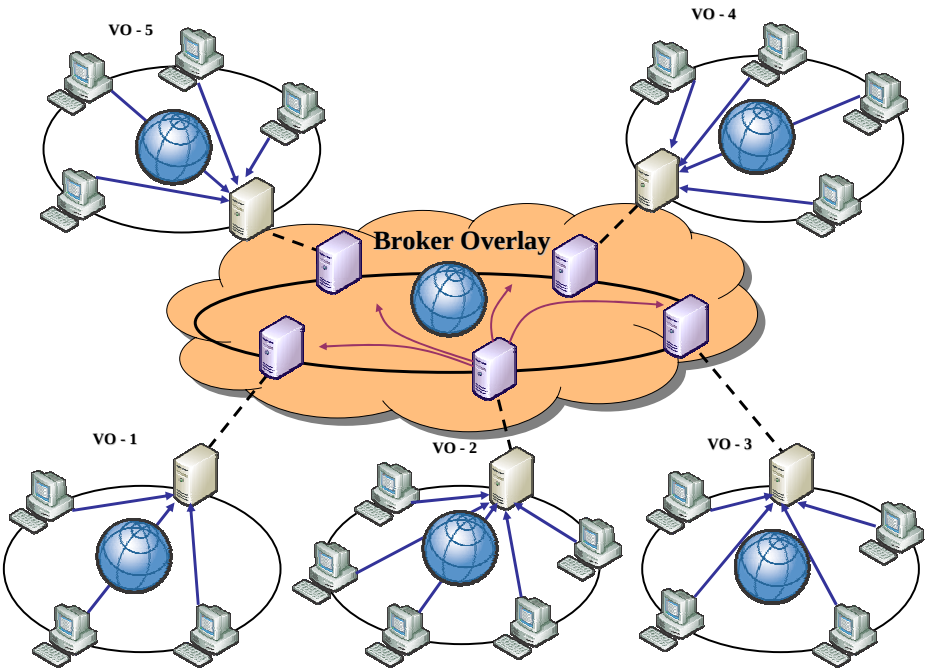


Fig. 1. Grid system structure

3 Resource Information Sharing

Resource information about any participating node is stored in a three field data block. The three fields represent: 1) Available CPU, 2) Available Memory, and 3) Time of last read. The third field, time of last read, is included to indicate if this read is too old so that it may not be dependable for deployment actions.

Each broker maintains a data set of resource information about all Grid nodes in the system. At each time unit, regular nodes in a virtual organization read the current resource state (i.e. available CPU, and available Memory) and send it along with the current time to the attached broker. Each time a broker receives a resource information block from a local node; it automatically removes the previously stored reading, and replaces it with the new one.

Brokers in the broker overlay also periodically exchange resource information. Each broker performs one exchange operation with a single neighbor broker[3] each time unit. The exchange operation is done by updating each resource information data set in each of the two brokers with the newest data blocks. The resource information exchange algorithm is described as follows:

```
//Start:
RS1 → Resource information data set of broker1
RS2 → Resource information data set of broker2

Declare RS as new DataSet;
Declare db1, db2 as new ResourceDataBlock;

// Loop among all Grid nodes:

For (int i = 0; i < Grid.size(); i++)
{
/* Retrieve information about Grid node(i) from datasets
of both brokers*/
      db1 = GetResourceInformationByNodeIndex(i, RS1);
      db2 = GetResourceInformationByNodeIndex(i, RS2);

// Compare the two blocks
      if (db1 != null && db2 != null)
      {
            if (db1.ReadingTime >= rs2. ReadingTime)
                  RS.add(db1);
            else
                  RS.add(db2);
         }
      else if (db1 != null)
         RS.add(db1);
      else if (db2 != null)
         RS.add(rs2);
}
// Update the data sets
```

[3] Neighbor brokers for a broker are those which it has direct communication with, according to the topology of the broker overlay.

```
Set RS1 = RS;
Set RS2 = RS;
```

4 Service Deployment Model

Deployment of Grid services on Grid nodes is done through Grid brokers. Brokers are responsible to deploy services to suitable nodes. Passing new services to brokers for deployment can be implemented in various ways: centrally (e.g. Deployment server accessed through web interface), directly to nodes, through the brokers by including a deployment portal in each broker, etc. In this work, deployment through the brokers is implemented. A Service deployer component is included in each node for forwarding services to the attached broker for deployment. The deployment model is depicted in fig. 2.

Each broker has a service deployment queue. When a service deployer sends a new service to a broker, it is automatically appended to the end of the service queue. Each time unit a broker picks the first service, placed in q0, from the queue and searches in the resource information data set for a suitable node which resource state matches the resource requirements of the service. The broker starts comparing resource state with resource requirements first among resource information blocks of the local nodes, in the local virtual organization. If no suitable resource found, the broker repeats the operation among resource information blocks of global nodes. If a global node matches, the broker passes the service to that node's broker with high priority, so that it will be placed at q0, else passes it to any neighbor broker. The deployment attempts parameter value of a service is decremented each time the service is transferred to a new broker queue. The algorithm is described in fig. 3.

4.1 Service validation parameters

Each time unit, a broker checks the *expiration time* and *deployment attempts* values for each service in the local service queue. For a service S:

```
If (S.waiting time < (Current time – S.Creation time) OR
S.deployment trials ==0)

      // Service S is expired
      Remove(S) from local service queue;
```

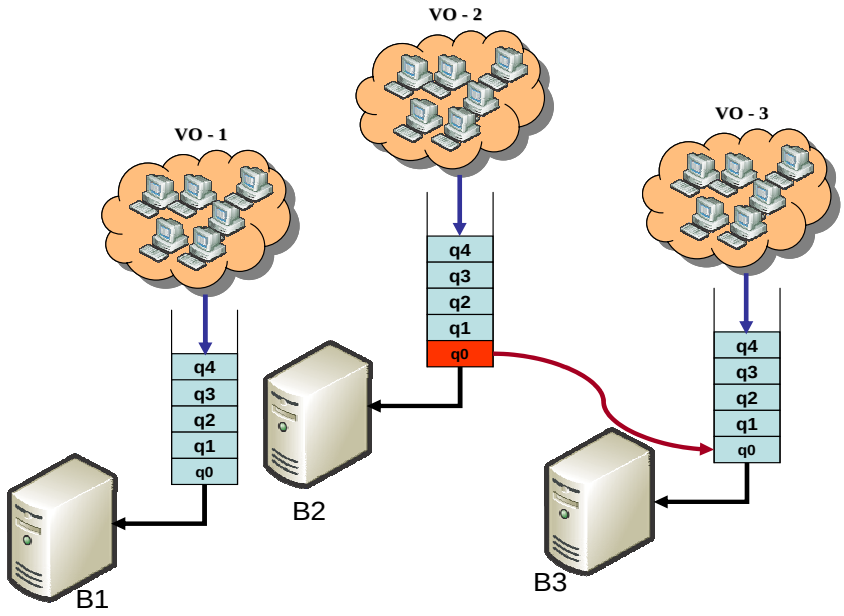


Fig. 2. Service deployment model

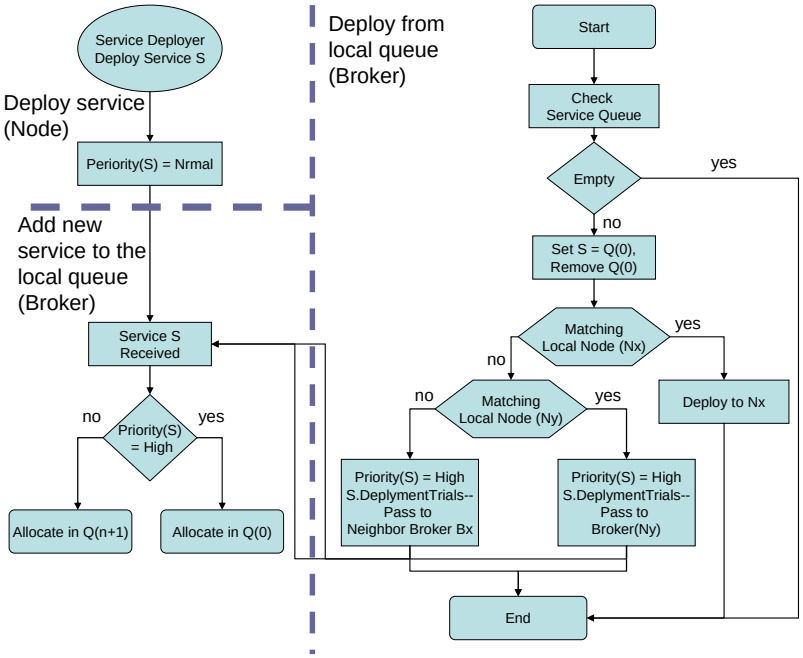


Fig. 3. Service deployment algorithm

5 Failure Handling

Two types of failure are considered: regular node failure, and broker failure. In this work focus is on broker failure. In a virtual organization, it is assumed that each regular node has direct communication only with the broker. In addition, each node in the Grid holds a list of information about all existing brokers in the broker overlay. This information is updated periodically in regular nodes through the attached broker.

When a broker failure occurs, a regular node can detect the broker failure when it attempts to send its resource information to the broker. In case of broker failure, all regular nodes in the local virtual organization of the failed broker will be detached from the Grid. Once a broker failure detected, a regular node sends a membership request to the first broker in the list. If the request is granted, the node will set the new broker as the attached broker, and add it as a neighbor, else repeat the request to the next broker in the list. Fig. 4 describes the algorithm implemented in regular nodes, together with resource information sending to the local broker.

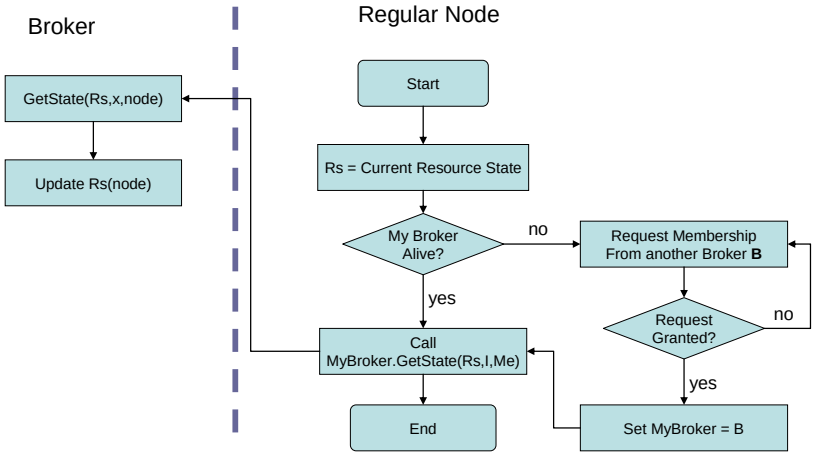


Fig. 4. Failure handling, and resource information sending algorithm

6 Simulation Model

The simulation model is built using PeerSim [14]; a Java-based simulation-engine designed to help protocol designers in simulating their P2P protocols. PeerSim supports both cycle-based and event-based simulation. This work is based on cycle-based simulation. Input parameters for the simulation engine are read from a configuration text file.

In this work, `GridNode` class is a source for node objects. `GridDeployer`, and `GridFailureControl` classes are included as a source for Control object to simulate service deployment and failure handling. Three cycle-driven Protocol classes [14] are also built: 1) `Grid CD Protocol`. Included in each regular node and is responsible for communicating with the attached broker and sends the resource

information in each simulation cycle. 2) `Deployment Protocol`. Included in each regular node and is responsible for responding to the deployment requests from the broker. 3) `Grid Broker Protocol`. Included in each broker node for performing the tasks associated with the broker (described in the previous sections). Fig. 5 describes the Grid simulation model and the communication between different protocols.

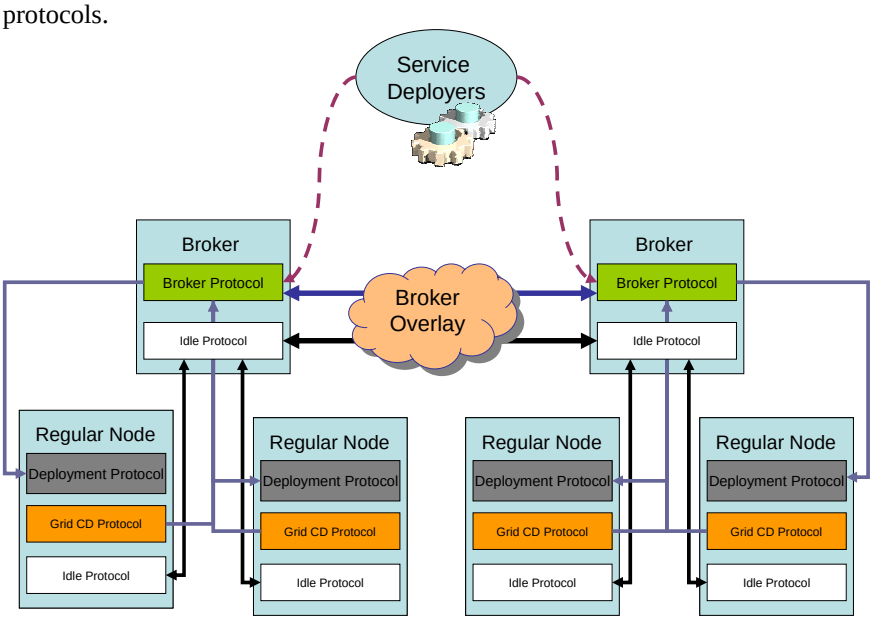


Fig. 5. Grid simulation model

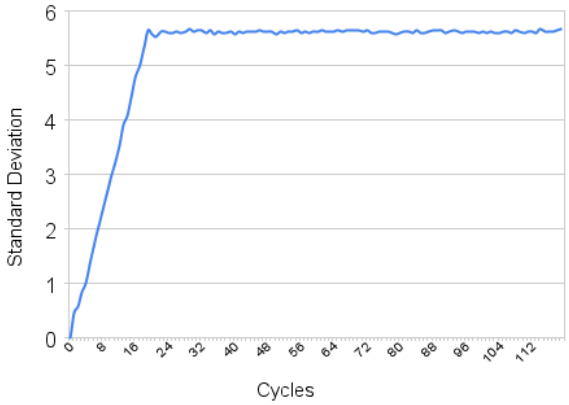
7 Performance Evaluations

To evaluate the performance of the proposed model, four performance benchmarks are used: 1) Validity of stored resource information, 2) Efficiency of service deployment, and 3) Impact of broker failure on resource information updating.

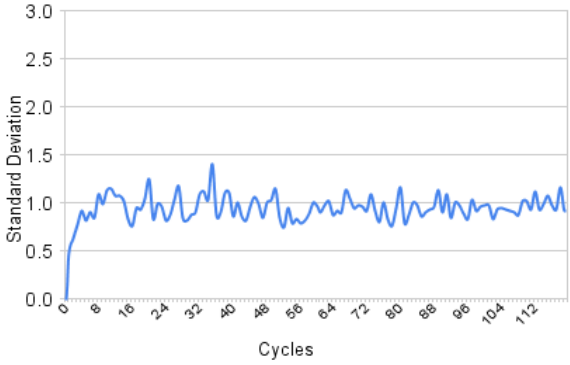
7.1 Validity of stored resource information

This benchmark is implemented through measuring the efficiency of the resource information exchange algorithm in keeping resource information up to date. The implemented methodology is to depict the deviation of the reading time values of resource information data blocks stored in the resource information data set, from the current cycle in a broker, with the simulation cycles. The results are read from one broker. Implemented topologies for the broker overlay are Ring and Fully connected. Total of 120 simulation cycles are used. Two experiments are performed with the following configuration: 1) Total network size of 100 nodes divided in 20 VOs. 2)

Total network size of 500 nodes divided in 100 VOs. The results are depicted for experiment 1 and experiment 2 in fig 6 and fig 7 respectively.

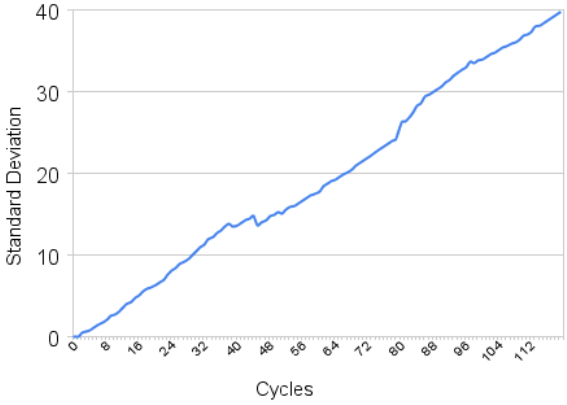


a) Ring broker overlay topology

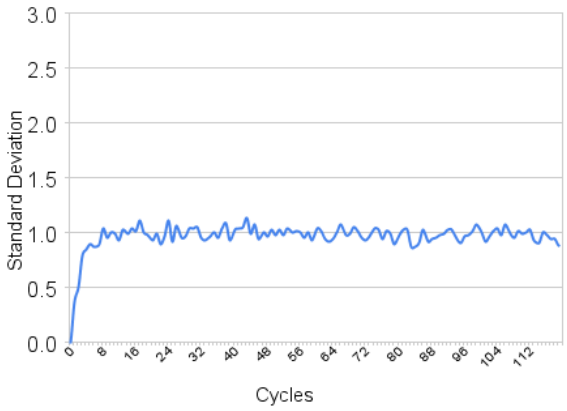


b) Fully connected broker overlay topology

Fig. 6. Deviation of the resource information reading time from the current cycle among simulation cycles for Network size of 100 nodes divided in 20 VOs.



a) Ring broker overlay topology



b) Fully connected broker overlay topology

Fig. 7. Deviation of the resource information reading time from the current cycle among simulation cycles for Network size of 500 nodes divided in 100 VOs.

In fig. 6, and fig. 7, it is clear that the deviation is much more less for the fully connected topology that for the ring topology. In addition, when the network size and the number of broker increased, in experiment 2, the deviation remains in the same level for fully connected topology, and incredibly increases for ring topology. This can be described that, in fully connected topology, a broker has all other brokers as neighbors with whom it can exchange resource information. This increases the chance to get more up to date data. In Ring topology a broker has only two neighbors. Increasing the number of brokers, the number of a broker neighbors increases for fully connected topology, but remains two for ring topology. This reduces the chance of reaching data stored in far brokers in ring topology, so, the deviation increases.

7.2 Efficiency of service deployment

This benchmark is implemented through measuring the efficiency of the service deployment algorithm in distributing services among available suitable nodes, using different broker overlay topologies. The network size is fixed to 500 nodes, and 100 virtual organizations. The implemented methodology is to depict the total number of waiting services, in broker queues, and the number of expired services with the simulation cycles. The results are collected from all brokers.

The main deployment method is: One broker periodical deployment. In this method, nodes of one VO deploy a number of services to the broker each specific number of cycles. The idea is to focus all the deployment traffic on one broker, as the worst case, to measure the efficiency of service exchange. Only the fully connected topology is tested with a total number of cycles of 1500. Two experiments are performed with the following configuration: 1) Total of 1500 services deployed as 10 services per 10 cycles. 2) Total of 3000 services deployed as 20 services per 10 cycles. The results are depicted for experiment 1 and experiment 2 in fig 8 and fig 9 respectively.

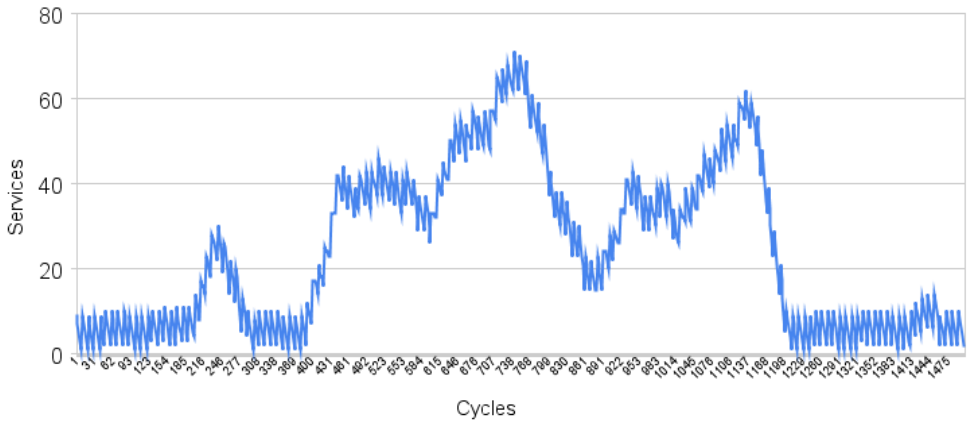


Fig. 8. Number of waiting services among simulation cycles for periodic deployment of 10 services per 10 cycles, and Fully connected broker overlay topology

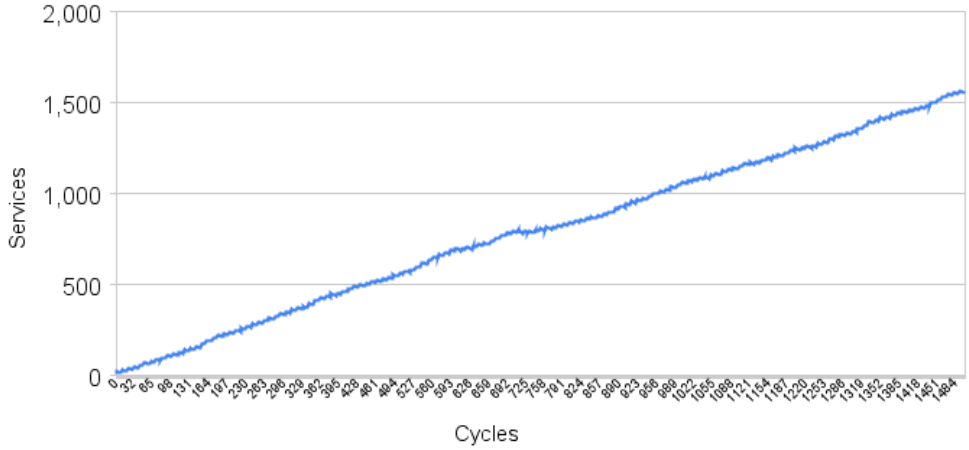


Fig. 9. Number of waiting services among simulation cycles for periodic deployment of 20 services per 10 cycles, and Fully connected broker overlay topology

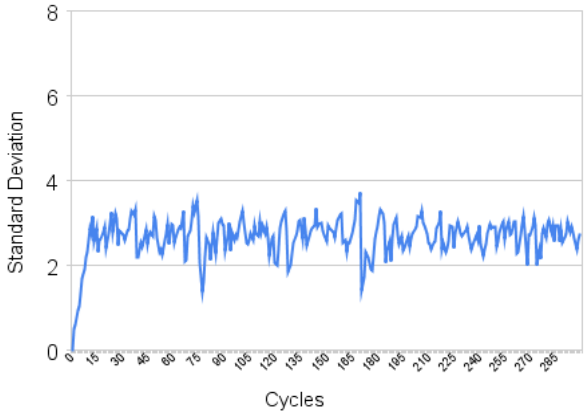
In fig. 8, it is clear that the system can satisfy the required performance. It is noticed that some bottlenecks can occur, but the system can recover. In fig. 9, it is clear that the system tends to be overloaded with services. It can be concluded that, in periodical deployment, the deployment ratio of 10 services/ 10 cycles (i.e. 1 Service/ cycle), is accepted and can be handled in a Grid system of network size >= 500, and 100 brokers with fully connected broker topology. If the ratio increased to 2 services/ cycle, the system, with the same network size, will reach overload.

7.3 Impact of broker failure on resource information updating

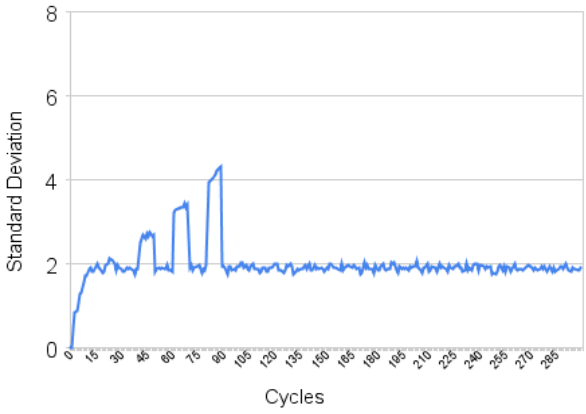
The aim of the experiments represented in this section, is to measure the impact of broker failures on the validity of stored resource information. Experiment 2 in Sec 7.1, is repeated with adding injected broker failures during the simulation. With the

existence of broker failures, it is expected that the deviation of the reading time values of resource information data blocks from the current cycle will increase in case of failure occurrence. The reason is that resource information of the regular nodes which have been attached to the failed broker, will remain old and not updated until they are attached to other brokers and start sending resource information blocks. In the following experiments, a new parameter is taken into account: *Data Age*, the maximum age, in cycles, of resource information in a broker resource data set. In each simulation cycle, the broker protocol checks the reading time of each block in the resource information data set. If the reading time of a block is < (Current time – Data Age), then, this block is removed from the data set. If a new block for the same node is received later, in an exchange operation, it is added to the data set. The following experiments are performed by varying the value of Data Age.

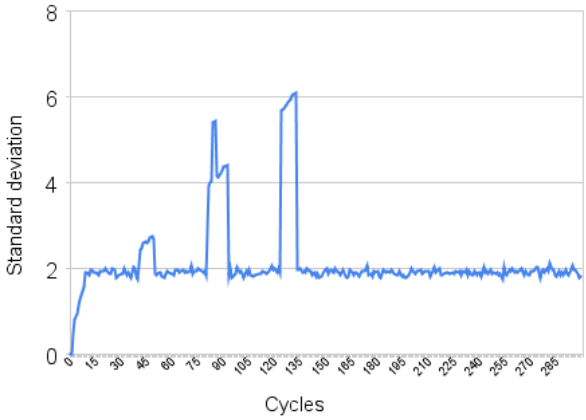
Four topologies are implemented: Ring, Fully connected, and Wire K Out (K = 60), and Hyper Cube. The network size is fixed to 500 nodes, and 100 virtual organizations. Number of simulation cycles is 300. Two experiments are performed with varying the total number of failures: 1) Data age of 10 cycles with 4 injected broker failures, and 2) Data age of 20 cycles with 8 injected broker failures. The results are depicted for experiment 1 and experiment 2 in fig 10 and fig 11 respectively.



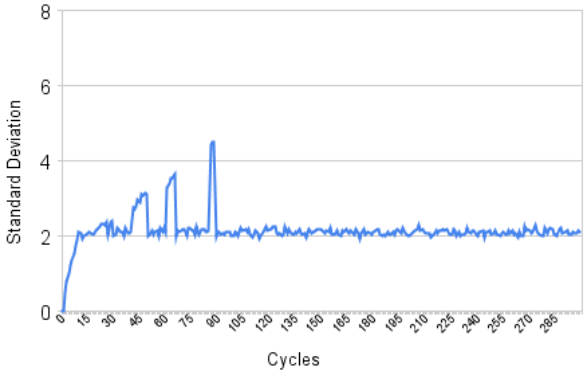
a) Ring broker overlay topology



b) Fully Connected broker overlay topology

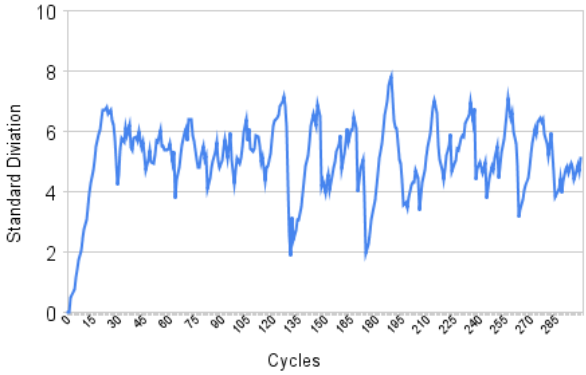


c) Wire K Out broker overlay topology, K = 60

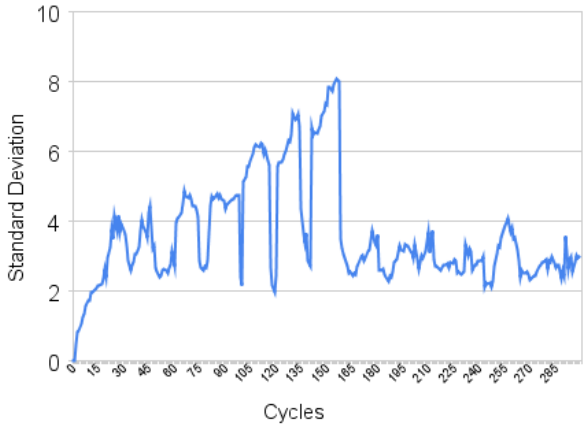


d) Hyper Cube broker overlay topology, K = 60

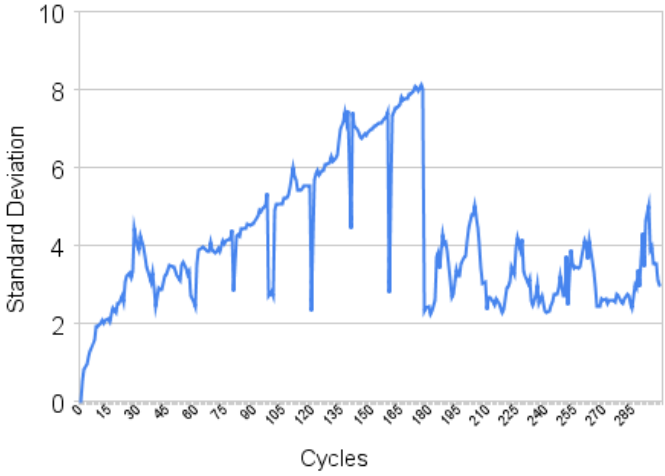
Fig. 10. Impact of failures on the deviation of the resource information for data age of 20 cycles with 8 injected broker failures



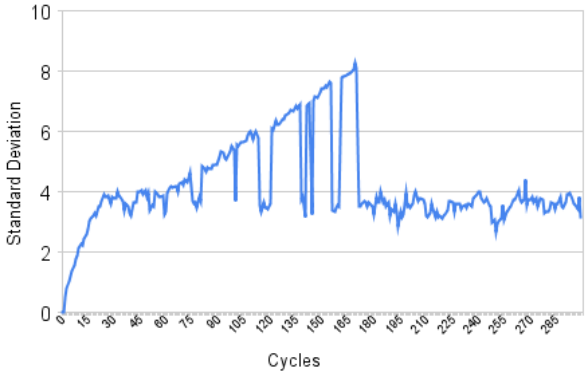
a) Ring broker overlay topology



b) Fully Connected broker overlay topology



c) Wire K Out broker overlay topology, K = 60



d) Hyper Cube broker overlay topology

Fig. 11. Impact of failures on the deviation of the resource information for data age of 20 cycles with 8 injected broker failures

In fig. 10, and fig. 11, it is clear that when the Data Age value decreases, the impact of failure decreases. This is because old data associated with unreachable nodes is periodically deleted from the resource information data sets. It is also clear that for Fully Connected, Wire K Out, and Hyper Cube topologies, the system can recover from failures and return to stable state. In case of Ring topology, the deviation has terrible variation and unstable. This can be described that, because of the lack of possible direct communications between brokers, it takes time for a broker to reach data stored in non-neighbor brokers.

It can be noticed that the magnitude of deviation caused by failure increases each time a new failure occurs, in Fully Connected, Wire K Out, and Hyper Cube topologies. This increase is not noticed in Ring topology. This increase can be described as follows: when a broker fails, all attached nodes attempt to join virtual organizations of other brokers. As the number of failures increases, the number of regular nodes attached to existing brokers also increases, So when a failure occurs then, the number of detached nodes will be larger than those in the previous failures, which causes increase in the number of old data blocks in brokers' data sets.

8 Conclusions

Grid simulation model which is built based on the concept of collaboration of virtual organizations has been presented. Global data exchange between virtual organizations has been implemented using the overlay network between brokers, based on different topologies. Four topologies for the broker overlay has been discussed and implemented. Two main algorithms have been described: resource information exchange algorithm, and service deployment algorithm. Performed experiments aimed at evaluating the performance of both algorithms with different broker overlay topologies. In addition, evaluating the performance of the resource information exchange algorithm in the existence of broker failures.

Results show that, the system can adapt to some extent to the service deploying load, and achieve required performance. Resource information exchange algorithm is efficient for the tested topologies, but in case of Ring topology, it biases to instability in case of failures, and slow in updating resource information data because of the lack of possible direct communications between brokers.

References

1. I. Foster, C. Kesselman, S. Tuecke.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: International J. Supercomputer Applications, 15(3), 2001.
2. I. Foster.: What is the Grid? A Three Point Checklist. In: GRIDToday, July 20, 2002.
3. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia.: Above the Clouds: A Berkeley View of Cloud Computing. In: Technical Report No. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences University of California at Berkeley, February 10, 2009.

4. I Foster, Yong Zhao, I Raicu, S Lu.: Cloud Computing and Grid Computing 360-Degree Compared. In: Grid Computing Environments Workshop, 2008. GCE '08, pp. 1-10 (2008).
5. David Barkai.: An Introduction to Peer-to-Peer Computing. In: Developer UPDATE Magazine Intel ®, Feb. 2000.
6. Kholidy, H.A. Azab, A.A. Deif, S.H. Enhanced "ULTRA GRIDSEC": Enhancing High Performance Symmetric Key Cryptography Schema Using Pure Peer To Peer Computational Grid Middleware (HIMAN). ICPCA 2008. Vol. 1, page(s): 26 – 31, 2008.
7. El-Desoky, A.E. Ali, H.A. Azab, A.A. A Pure Peer-To-Peer Desktop Grid framework with efficient fault tolerance. ICCES'07. page(s): 346 – 352, 2007.
8. Condor project. http://www.cs.wisc.edu/condor/.
9. The Globus toolkit. http://www.globus.org/toolkit/.
10. Open-source software for volunteer computing and grid computing. http://boinc.berkeley.edu/.
11. Scalable desktop Grid system, Peter Kacsuk, Norbert Podhorszki, and Tamas Kiss, In VECPAR 2006. 7th International meeting on high performance computing for computational science. Rio de Janeiro, 2006., Pages 1-13, 2006.
12. Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", Wiley Press, New Jersey, USA, June 2005.
13. Azab, A.A. Kholidy, H.A. An adaptive decentralized scheduling mechanism for peer-to-peer Desktop Grids. ICCES'08. page(s): 364 – 371, 2008.
14. PeerSim: A Peer-to-Peer Simulator. http://peersim.sourceforge.net/.