2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

ad hoc Grid: An Adaptive and Self-Organizing Peer-to-Peer Computing Grid

Pablo G. S. Tiburcio
Computing and Systems Department
Federal University of Campina Grande
Campina Grande, PB - Brazil
Email: tiburcio@dsc.ufcg.edu.br

Marco Aurélio Spohn
Computing and Systems Department
Federal University of Campina Grande
Campina Grande, PB - Brazil
Email: maspohn@dsc.ufcg.edu.br

***Abstract*—This paper presents an adaptive and self-organizing Peer-to-Peer (P2P) computing grid. The proposed solution, named *ad hoc Grid*, leverages on the *OurGrid* (OG) middleware, which is an open source P2P computing grid. OG requires some centralized administration, preventing it to be used for deploying spontaneous (ad hoc) computing grids. To make the P2P grid adaptive and self-organizing, new approaches were adopted for Peer discovery, failure handling and failure recovery. Grid nodes communicate via multicast: a set of grid members (e.g., nodes in the same local network) are logically connected to a single Peer, and they coordinate through a local multicast group; as for the Peers, they get to know each other through another multicast group, by exchanging periodical and on demand messages which allow them to coordinate and keep the grid connected. Instead of having a static and well defined role in the grid as in the OG architecture, any grid node might eventually take the Peer role. This way, the grid can be instantiated even with a single node, and grow as new nodes join the grid. *ad hoc Grid* is well suited for those willing to quickly deploy a computing grid without requiring any centralized administration, by just combining the computing power of distributed machinery of users willing to join the grid.**

***Keywords*-peer-to-peer computing grid; self-organizing systems;**

I. Introduction

Computing grids provide the means to harvest computing resources spread over a geographical area, and in some cases spanning different administrative domains [2]. There are many grid computing architectures [3], but this work focuses only on peer-to-peer (P2P) computing grids, which usually allow building decentralized grids.

OurGrid [1] (OG) middleware is targeted for building *free-to-join* P2P grids for bag-of-tasks applications (i.e., applications that run independent tasks). However, OG requires the intervention of an administrator to keep the system up and running. The *Core Peer* (a centralized server for discovering active Peers) is also a single point of failure. In addition to that, an administrator has to set up Peers and Workers (i.e., the entities that actually execute foreign tasks).

The main motivation for this work consists in providing a solution, named *ad hoc Grid*, for building self-organizing P2P computing grids. Given that OG provides the basic elements for running tasks on the grid (i.e., accounting and task scheduling), one can just focus on the grid infrastructure itself, identifying what is needed to make it independent of any centralized administration.

To make the grid self-organizing, the centralized Peer discovery server (i.e., Core Peer) role is played by a peer-to-peer multicast group. When Peers join a grid community, they actually join this multicast group, through which they also get to know details about the current active Peers. Given that in OG a single Peer may be responsible for several Workers in a single administrative domain (alternatively there might be a Peer for every local network), there is a local multicast group for bringing together all the entities under the administration of a single Peer.

The rest of this article is organized as follows. Section II presents a brief description of *OurGrid*. Section III presents an approach for building adaptive and self-organizing P2P computing grids, and a case study based on the OG middleware. Section IV presents an experimental evaluation of the proposed solution, and Section VI concludes this work.

II. *OurGrid*: a short description

OG [1] is a free-to-join P2P grid computing system. It is targeted for decentralized grid infrastructures, such as independent small to medium size labs which just want to cooperate by sharing their idle computing resources. OG is designed for running bag-of-tasks applications (i.e., applications made out of independent tasks which can run separately). To mitigate the impact of *free-riding*[1], OG adopts an autonomous accounting system named *Network-of-Favors* [4] which creates incentives for users to donate their idle computing resources.

There are four main components in the OG architecture (depicted in Figure 1): `Peer`, `CorePeer`, `MyGrid`, and `Worker` [2]. `MyGrid` is the user's broker when interacting with the grid. `MyGrid` talks to the local Peer which provides a list of host machines ready to receive tasks, then MyGrid is responsible for scheduling the tasks and monitoring them as well. The `Peer` is responsible for managing a set of `Workers` which are actually the entities that execute

[1] A free-rider is a user (or an entity) which just receives resources from other users (or entities) without giving anything back to the donators.

[2] Also called *UserAgent*.

978-0-7695-4108-2/10 $26.00 © 2010 IEEE
DOI 10.1109/CIT.2010.504

IEEE computer society

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

the tasks dispatched to the grid. As mentioned before, the `CorePeer` acts as a discovery server for gathering information about all the active Peers in the grid community.

III. An Adaptive and Self-Organizing P2P Computing Grid

Leveraging on the OG components, it is proposed a solution named *ad hoc Grid* for adaptive and self-organizing P2P computing grids. First of all, the peer discovery server (played by the Core Peer in the OG architecture) is replaced by a peer-to-peer multicast group. When a Peer joins a grid community, the Peer actually joins the P2P multicast group (see Figure 2). This way, there is no need for any centralized peer discovery service. Peers announce themselves periodically through this multicast group, and they can be queried as well.

A grid community is defined as a set of Peers interconnected through a multicast group (called peer-to-peer multicast group). Peers may belong to different administrative domains, but they must be able to communicate through multicast.

A grid site is defined as a set of members (i.e., grid application instances) under the guidance of the same Peer. It is assumed that there is at least one Peer per local network.

To make the system totally decentralized, the main OG elements (i.e., Peer, Worker, and MyGrid) are integrated into a single application. That is, the user still interfaces with MyGrid, but now the Peer and Worker can also be active in the same host. The Worker is always instantiated to allow the local user running her own tasks in the grid, and for executing foreign tasks when the local host is idle.

Depending when the application was started by the user, the application has to play the Peer role as well (Algorithm 1 presents the pseudo-code for the start-up procedure). To make things simpler, just one Peer is deployed in each local network. The first instance of the application to run on the local network becomes the local Peer. To find out if there is already a local Peer, the node joins a pre-defined multicast group (line 2 in Algorithm 1) - which address is derived from the local network address (alternatively, one could also rely on Zeroconf Multicast Address Allocation Protocol for assigning multicast addresses on demand) - used for communication with the local Peer. Two solutions for detecting a local Peer are presented (the pseudo-code in Algorithm 1 shows just one of these solutions):

- *Passive detection*: Active Peers are supposed to post to the local multicast group an announcement every t_{alive} seconds. In case the node does not hear from any active Peer after $n \times t_{alive}$ s, the node defines itself as the local Peer. Considering that communication in local networks is pretty reliable, one can expect that after n attempts any node waiting for an announcement will eventually receive at least one message from the Peer.

Algorithm 1: Start-Up.

```
        i       Member's IP address
Data:   peer    Local Peer
        LMG     Local Multicast Group
1  begin
     /* Join the local multicast group
        */
2    JoinGroup(LMG)
     /* Check for any Peer: proactive
        approach!                          */
3    for j=1 to n do
       /* Query the group, and wait for
          t_ack s                          */
4      peer = SendMessage(LMG,QueryPeer)
5      if peer != null then
6        break
7      end
8    end
9    if peer==null then
       /* No Peer in local network. This
          node should take the Peer
          role.                            */
10     peer = i
       /* Start the peer thread!          */
11     Peer.start()
       /* Just in case, announce its
          Peer status to the LMG.          */
12     SendMessage(LMG, PeerAnnouncement(i))
13   end
14   else
       /* Report to the local Peer by
          sending all the necessary info
          regarding this node.             */
15     SendPeer(peer, i.info)
16   end
     /* There is a Peer. Ready to start
        the Worker.                        */
17   Worker.start()
     /* Start thread for keeping track of local Peer!
        */
18   CheckPeer.start()
19 end
```

- *Proactive detection* (Algorithm 1, lines 3 to 8): After joining the local multicast group, a node queries the group for any active Peer. The node repeats the query n times spaced by t_{ack} s. In case the node does not receive any reply, it takes the Peer role for this local network sending a proper announcement via the local multicast group.

At any given time, if it happens to have more than one active Peer (check Algorithm 2 for the pseudo-code), the

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

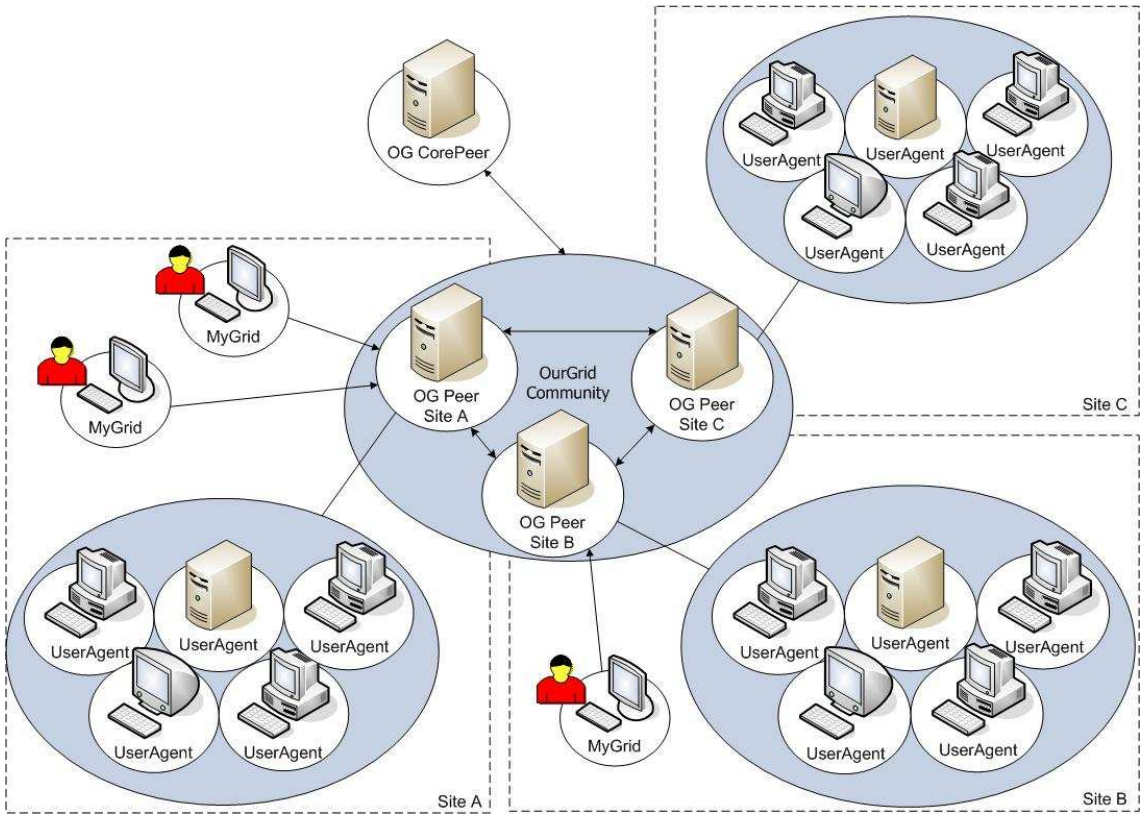

Figure 1. OurGrid architecture.

Peer with the largest IP address selects itself as the local Peer, announcing itself as the *de facto* Peer via the local multicast group.

A node whose status is defined as *non* Peer will have to report to the local Peer (line 15, Algorithm 1), which, on its turn, will gather information about all the current application instances (i.e., applications acting as Workers from the grid point of view). Considering that Peers can fail, Peers (see Algorithm 4 for Peer's pseudo-code) periodically post to the local multicast group the local site accounting information (i.e., regarding the Network-of-Favors). When the members detect that the local Peer is no longer active, the member with the largest IP takes over as the new Peer. The most recent accounting report is used for the new Peer to start over.

Peers from different sites interact with each other through the P2P multicast group (lines 2 and 3 in Algorithm 4). This way, instead of querying a central server (e.g., *Core Peer*), queries are posted to the P2P multicast group.

When a user submits tasks to the grid (via the MyGrid interface), the application asks the local Peer for available Workers (Algorithm 3 shows Worker's pseudo-code). The Peer on its turn, sends a request to all active Peers through the P2P multicast group (this is accomplished by the *P2P-MG* thread, line 5 in Algorithm 4). This is by itself an improvement, because multicast allows a single message to reach all the active Peers, while in OG an unicast message is sent to every Peer individually to achieve the same purpose. Once the Peer hears from other Peers, the local Peer sends back to the local requester the list of available Workers. After this point, the tasks are scheduled as in the original OG; that is, the application sends out the tasks to some or all the machines listed by the Peer.

A. Protocol Correctness

To show that the protocol is correct, one has to show that it is safe and live. While there is at least one user in the local network, there is also a Peer. Given that there is a Peer, it is guaranteed that there is at least one Peer member in the P2P multicast group. Considering that all Peers must join the P2P multicast group, all Peers are accessible to each other.

The protocol is live because it is deadlock free (i.e., loop free). Given that any new member of any site has to check if there is a Peer for that site, and that the checking waiting time is well defined and finite, two things can happen: (a) there is no Peer in this site, and the new member takes the Peer role; and (b) there is a Peer which the new member gets to know via a Peer advertisement. In case there is a Peer but

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

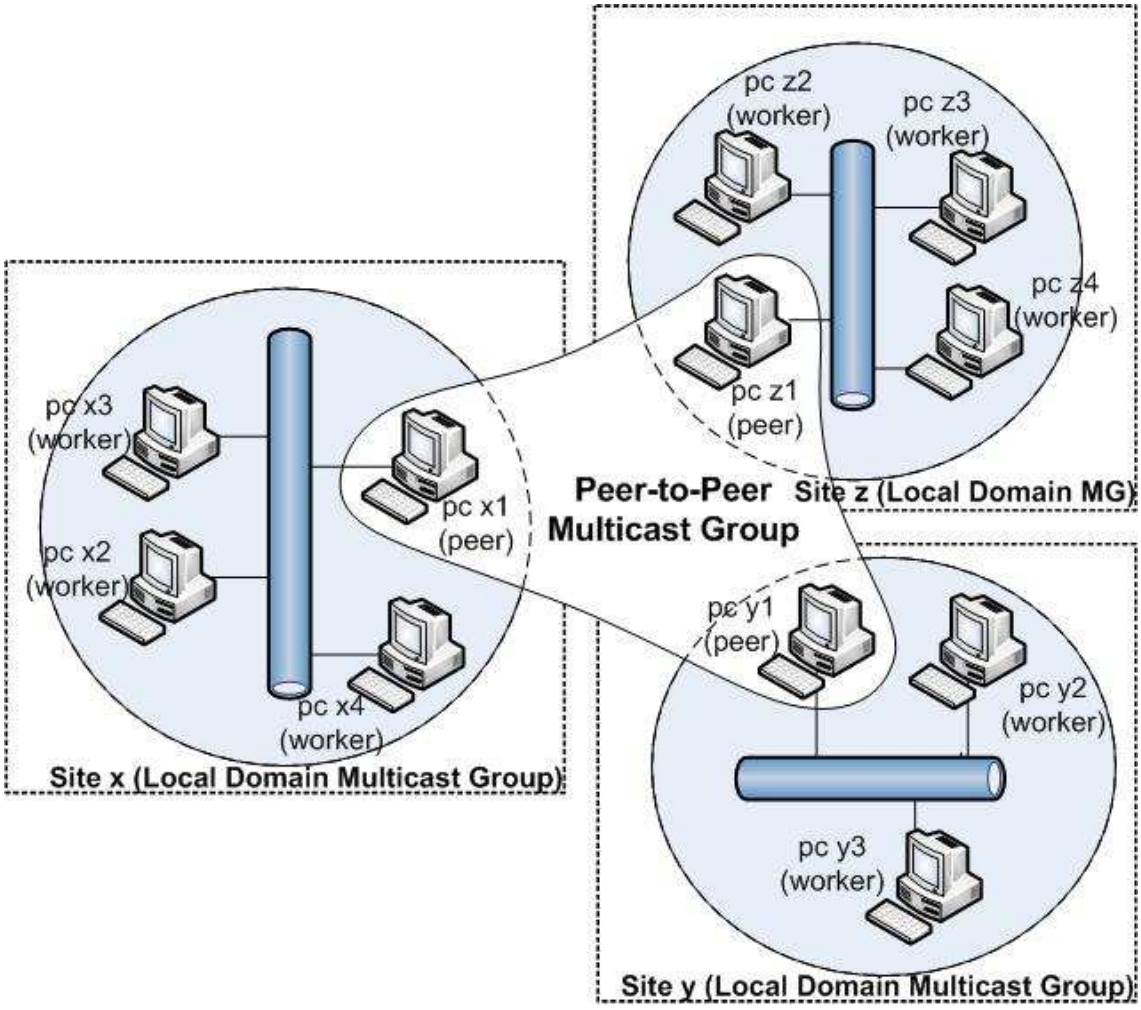

Figure 2. *ad hoc* Grid.

the new member does not get to know about it during the waiting time, more than one Peer might exist for a finite period of time after which the member with the largest IP takes over the Peer role.

IV. Experimental Evaluation

A network (as depicted in Figure 3) was set up for running the experiments. As for the workload, jobs that simulate *NodeWiz* (a distributed solution for resource discovery on the Internet) [5] are submitted to the grid, with each individual job consisting of five tasks (i.e., each task simulates a given scenario in NodeWiz).

A. Evaluating Grid Organization

To evaluate adaptiveness and self-organization, grid components were monitored during the execution of batches of jobs, and experiments were repeated ten times. The results show the average Peer discovery time and, eventually, the identification of more than one transient Peer per local group.

In order to automate the execution of experiments, it is employed a functional test module for the system. This module was implemented in Java and monitors the operations of each grid component, as follows:

- Monitoring when a node joins the local multicast group;
- Monitoring when a node joins the P2P multicast group;
- Checking the type of message received by the local multicast group, storing the grid component status, IP addresses, and the message timestamp;
- Checking the type of message received from the P2P multicast group, and identifying any inconsistencies on Peer assignments.

The first scenario under consideration consisted of a grid comprised of a single machine. Given that the node is standing alone, it took in average 10s for the machine to assume the Peer role, because the worst case in the proactive

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

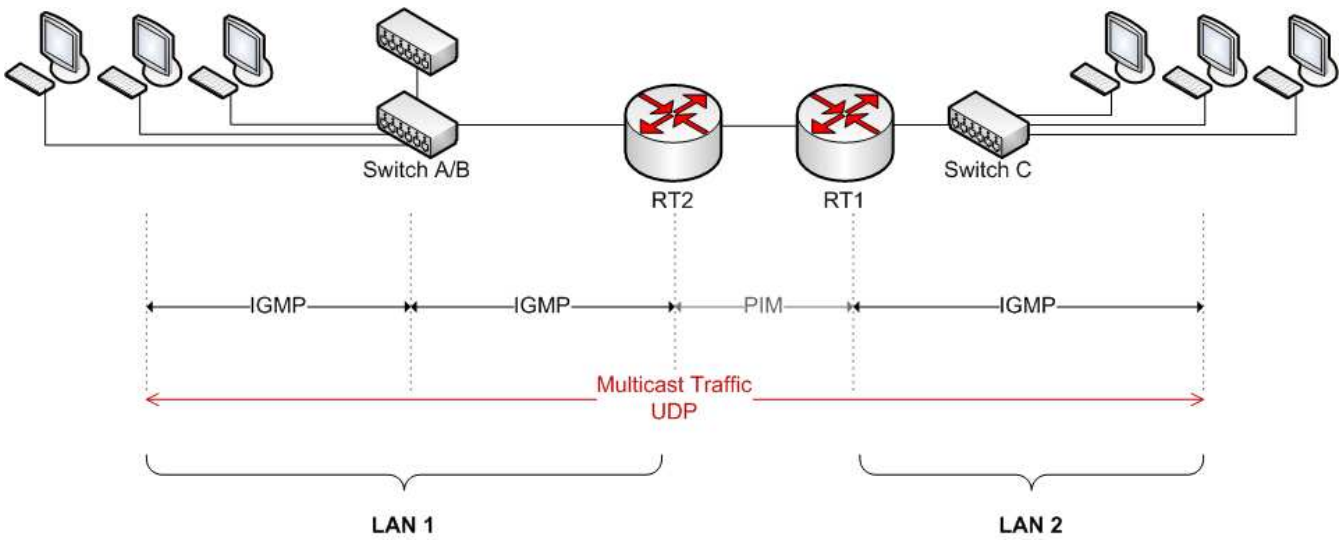

Figure 3. Network topology.

Peer discovery approach is $n * t_{ack}$, with $n = 3$ and $t_{ack} = 3$s. In addition to that, one has to account for the extra time for starting up the Worker, Mygrid, and the Peer process itself. Despite the t_{ack} parameter had been defined as a constant, it could well be computed dynamically taking, for example, a weighted moving average over round trip time (RTT) measures.

The second scenario was comprised of two machines, each instantiating the *ad hoc* Grid at different times. The average Peer discovery time for the second instance was $1s$. Peer recovery was evaluated by stopping the execution of the Peer process in one of the grid machines. The remaining machine identifies the Peer failure, and later assuming its role. As each machine locally maintains the list of Workers and the contability for the network of favors, such data is retrieved without the need for querying the local multicast group.

The ad hoc Grid was also evaluated when two local grid hosts start simultaneously. In this case, for a while the two instances assume the Peer role, but the machine with the lowest IP gives up to the other instance after on average $2s$.

In the last scenario, a given set of grid machines start at random periods of time. It is noticed that the transient state of multiple simultaneous Peers in the grid is very unlikely. Nevertheless, after a finite period of time the instance with the highest IP takes the Peer role, and the remaining ones act just as Workers.

B. Performance

When evaluating the ad hoc Grid performance, three scenarios were taken into account. In the first scenario, the grid configuration is kept the same throughout all the grid lifetime, and there is no failures. In the second scenario, faults are injected into the grid components, by choosing at random one grid component once every 10 minutes, and bringing the instance back to grid after 5 minutes after the failure. In the last scenario, faults are injected more often, by choosing one victim once every 1 minute.

For all scenarios, the grid is comprised of 25 machines on network LAN1, and by 5 machines on network LAN2. Having established the P2P grid, four jobs are submitted to the grid every 20 minutes from network LAN1, and two jobs are submitted every 20 minutes from network LAN2. The number of jobs and the time interval in which they are submitted to the grid have been chosen so that for the first scenario all the jobs have plenty of time to be executed, allowing to account for the control overhead introduced by the ad hoc Grid. As for the second and third scenarios, they were designed to capture the impact of automatic grid maintenance on the overall grid performance.

Each scenario underwent ten trials of one hour each, obtaining the statistics for the average running time, control overhead (mainly multicast messages), and Peer availability. Therefore, each job was executed 180 times for each scenario.

1) Computing Performance: The ad hoc Grid computing performance is compared against OG. The observed average job running time on OG was 731.6s, while it was 742.42s for the ad hoc Grid, representing an increase of 1.48% compared to the running time for OG. As for the second scenario, it is noticed an increase of 1.64%. In the last scenario, the ad hoc Grid presented average running time of 778.6s, representing an increase of 6.43% compared to OG, mainly due to the increased instability in the ad hoc Grid (it is worth mentioning that faults were not introduced during the OG tests).

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: CheckPeer *thread.*

```
Data: i      Member's IP address
      LMG    Local Multicast Group
      peer   Local peer ID
1  begin
2    while true do
       /* Once t_rep s Peer is supposed to
          send local site info via the
          LMG. n − 1 report loss
          tolerance.                        */
3      ScheduleTimer(Peer-Report, n × t_rep)
4      if no report received OR more-than-one-peer
       then
5        if i is the largest ID member AND peer!=i
         then
           /* Select itself as Peer        */
6          peer = i
           /* Start the peer thread!       */
7          Peer.start()
           /* Just in case, announce
              itself to the LMG.            */
8          SendMessage(LMG,
           PeerAnnouncement(i))
9        end
10       else
           /* In case there is another
              winner, stop local Peer!
              */
11         if more-than-one-peer AND peer == i
           then
12           Peer.stop();
13         end
14       end
15     end
16   end
17 end
```

Algorithm 3: Worker *thread.*

```
Data: i      Member's IP address
      peer   Local peer ID
1  begin
2    while true do
       /* Wait for a new task              */
3      Worker.accept(task)
4      ret=task.run()
5      if ret=success then
         /* After running the task
            successfully, send
            accounting report to Peer.
            */
6        SendMessage(peer, report)
7      end
8    end
9  end
```

Algorithm 4: Peer *thread.*

```
Data: i      Member's IP address
      LMG    Local Multicast Group
      P2P-MG P2P Multicast Group
1  begin
     /* Join the P2P multicast group     */
2    JoinGroup(P2P-MG)
     /* Send Peer status to all Peers
        via the P2P-MG                    */
3    SendMessage(P2P-MG, peer.info)
     /* Once every t_rep s sends a local
        site report (i.e., accounting
        info and list of active
        members)!                         */
4    Schedule(LMG, PeerReport, t_rep)
     /* Start thread for sending/receiving queries
        to/from other Peers!              */
5    Peer.start(Thread.P2P-MG)
     /* Start thread for accepting queries from
        local members!                    */
6    Peer.start(Thread.LMG)
7  end
```

2) Control Overheard: The next step was to measure the control overhead resulting from the exchange of multicast messages among grid members. Results are presented for the measured control overhead between networks LAN1 and LAN2 (i.e., mainly due to multicast message exchange among Peers), and the control overhead in each local network (i.e., mainly due to the communication among local multicast group members).

Internetwork Overhead: Figure 4 presents the control overhead resulting from multicast transmissions passing through router RT1 (see Figure 3). These results represent the exchange of multicast packets between the Peers located at LAN1 and LAN2 respectively. The average packet transmission was 11.73 packets/s, with a peak of 13.24 packets/s at the beginning of the experiments, because there is all the extra IGMP related messages once nodes join a new multicast group, and that takes place at the beginning of the grid deployment. The multicast traffic accounted for at router RT2 was the same as for router RT1, given that RT2 is there just to connect the two networks together.

Intranetwork Overhead: Figure 5 shows that there were in average 8.43 multicast packets/s being transmitted in LAN1, while there were in average 1.49 multicast packets/s in LAN2. There were more packets in average in LAN1 because it has 25 machines while there are just 5 machines

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

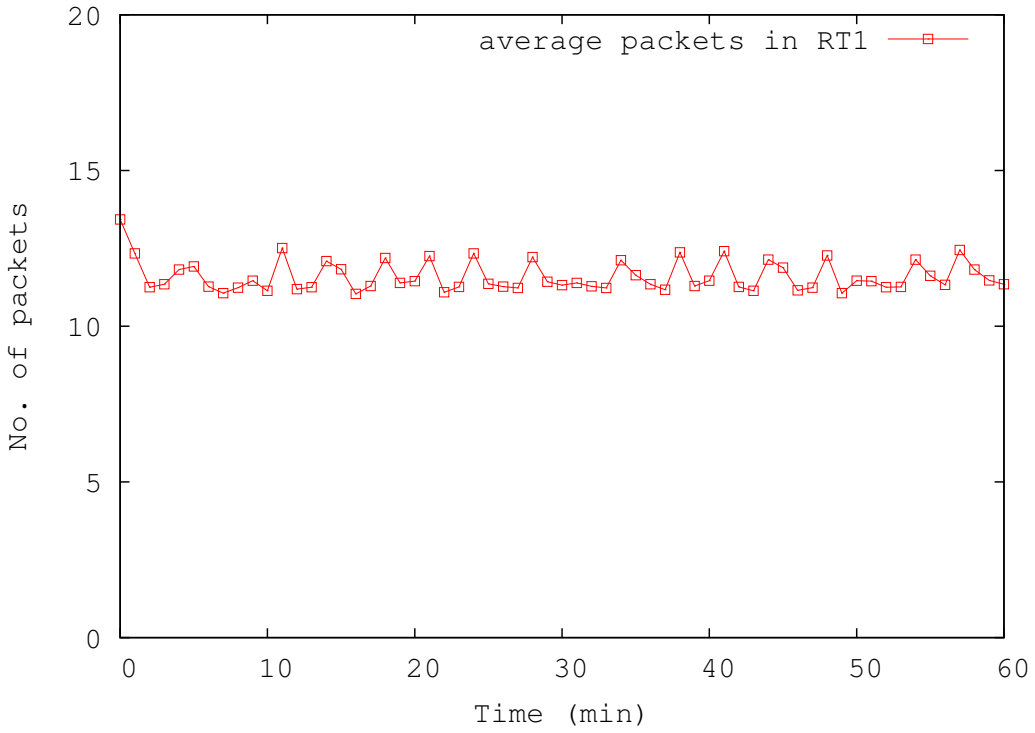

Figure 4. Internetwork control overhead.

in LAN2. Therefore, each machine in both networks transmit one multicast packet once every three seconds (i.e., or one could say each machine transmit 0.33 multicast packets/s in average). Assuming multicast packets of 2 KBytes, the average multicast throughput incurred by each grid node is approximately 660 Bytes/s.

V. Related Work

The Organic Grid [6] is a grid computing system based on autonomous scheduling by mobile agents in a P2P network. It is a large-scale desktop computing grid which allows organizing computing resources for different types of applications.

Mobile agents encapsulate the computing power, molding the grid behavior. These agents communicate to keep an overview of the system, adapting to any changes in the environment. The agents adopt algorithms inspired in biological systems, allowing agents to schedule their jobs independently depending on the available computing resources, aiming at maximizing the overall use.

The grid is organized following a tree structure that is constantly modified to adapt to new conditions of the system. Thus, each agent represents a node in the tree, and when a node receives a request for any of its available resources, the requesting agent is set as a child of the provider node in the tree. The resulting topology of the network is a tree where the machine that first started the grid is the root.

Each node in the tree can have a maximum number of child nodes, which are selected based on their performance. Thus, if a node has extrapolated the maximum number of child nodes, the slowest node is removed. The performance of a child node is measured by the time interval between two reporting results.

The Organic Grid deployment relies on some pre-defined bootstrap nodes (called friend list), which are nodes that an agent must first contact when joining the grid. Therefore, one can observe that the initial grid configuration still relies on a pre-defined initial state. In our solution, Peers have their multicast group as a *rendezvous point*, through which they can interact with all other Peers in the grid community.

The ICEN [7] grid middleware is based on a service-oriented architecture (SOA) that enables its users to share idle resources. Services are made available following three basic actions:

- Announcement: The service provider offers the service to service brokers;
- Discovery: consumers contact service brokers for discovering the required service;
- Interaction: consumers and service provider interact while the service is made available.

The ICEN middleware has an architecture similar to OurGrid, where the service provider can be compared to the Worker which provides its resources to the grid, the service

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.

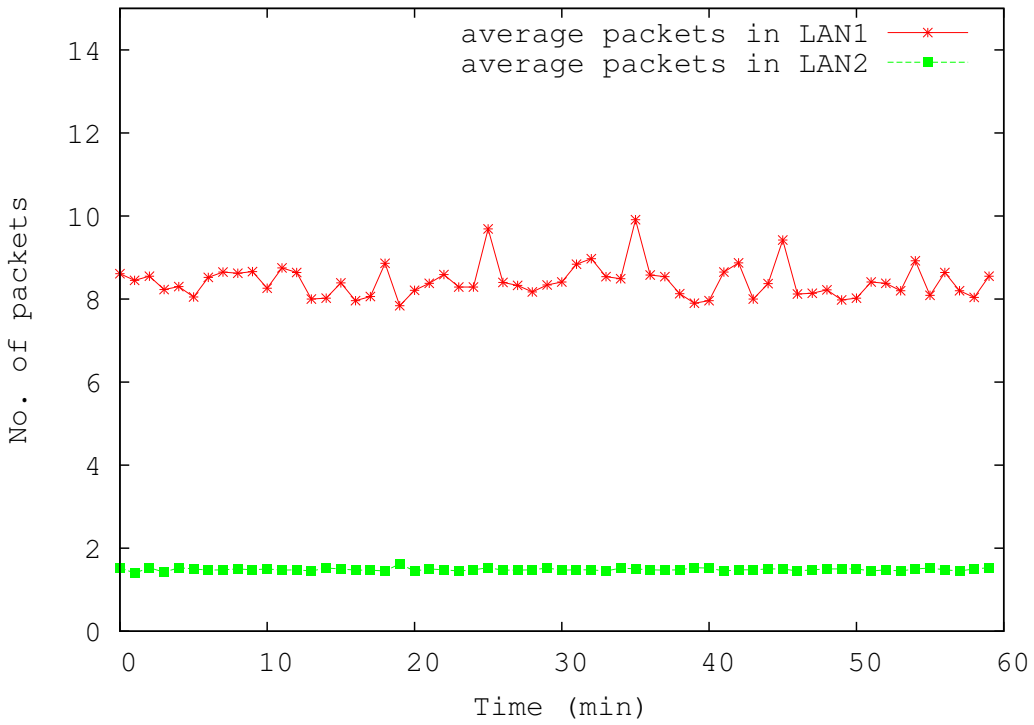

Figure 5. Intranetwork control overhead.

consumer and broker can be compared to the MyGrid, and the interaction between consumers and providers in the OurGrid is coordinated by the Peers. Both ICEN and OurGrid do not provide means for automatic deployment and service migration, which are essential for an ad hoc grid.

VI. Conclusions

This paper presented a solution, named *ad hoc Grid*, for deploying adaptive and self-organizing peer-to-peer computing grids. A P2P multicat group acts as a *rendezvous point* for all Peers, providing all the necessary information for Peer discovery. There is just one Peer per local site, which is managed through a local multicast group. All the non Peer members in the same site are managed by their local Peer. To guarantee that the local site survives to a Peer failure, Peers periodically advertise their accounting records to all local members. In case a Peer fails, the current member with the largest IP takes over as the new Peer. Leveraging on the *OurGrid* middleware, the ad hoc Grid has been able to provide the same grid computing services but without the need for any centralized administration.

References

[1] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, Miranda Mowbray. Miranda: *Labs of the World, Unite!!!*. Journal of Grid Computing, Volume 4, Issue 3, p. 225-246, 2006.

[2] Ian Foster, C. Kesselman: *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Second Edition, 2004.

[3] Heba Kurdi, Maozhen Li, Hamed Al-Raweshidy: *A Classification of Emerging and Traditional Grid Systems*. IEEE Distributed Systems Online, vol. 9, no. 3, art. no. 0803-o3001, 2008.

[4] Nazareno Andrade, Francisco Brasileiro, Walfredo Cirne, Miranda Mowbray: *Discouraging Free-riding in a Peer-to-Peer CPU-Sharing Grid*. Proceedings of 13th IEEE International Symposium on High-Performance Distributed Computing, pages 4-9, 2004.

[5] Sujoy Basu, Lauro Beltrão Costa, Francisco Brasileiro, Sujata Banerjee, Puneet Sharma, Sung-Ju Lee: NodeWiz: Fault-tolerant grid information service. Peer-to-Peer Networking and Applications. Springer New York, 2009.

[6] A. J. Chakravarti, G. Baumgartner, M. Lauria: The organic grid: self-organizing computation on a peer-to-peer network. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, vol. 35, no. 3, p. 373-384, 2005.

[7] Nathalie Furmento, Jeffrey Hau, William Lee, Steven Newhouse, John Darlington: Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI. Lecture Notes in Computer Science, vol. 3165, p. 90-99, 2004.

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:45:50 UTC from IEEE Xplore. Restrictions apply.