Future Generation Computer Systems 29 (2013) 1583–1595

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

CycloidGrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems

Toktam Ghafarian [a,*], Hossein Deldari [a], Bahman Javadi [c], Mohammad H. Yaghmaee [a], Rajkumar Buyya [b]

[a] Department of Computer Engineering, Ferdowsi University of Mashhad, Iran

[b] Cloud Computing and Distributed Systems Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

[c] School of Computing, Engineering and Mathematics, University of Western Sydney, Australia

ARTICLE INFO

Article history:
Received 23 January 2012
Received in revised form
31 July 2012
Accepted 22 August 2012
Available online 31 August 2012

Keywords:
Peer-to-Peer computing
Volunteer computing
Resource discovery
Proximity-aware scheduling
Bag of tasks application

ABSTRACT

Volunteer computing which benefits from idle cycles of volunteer resources over the Internet can integrate the power of hundreds to thousands of resources to achieve high computing power. In such an environment the resources are heterogeneous in terms of CPU speed, RAM, disk capacity, and network bandwidth. So finding a suitable resource to run a particular job becomes difficult. Resource discovery architecture is a key factor for overall performance of peer-to-peer based volunteer computing systems. The main contribution of this paper is to develop a proximity-aware resource discovery architecture for peer-to-peer based volunteer computing systems. The proposed resource discovery algorithm consists of two stages. In the first stage, it selects resources based on the requested quality of service and current load of peers. In the second stage, a resource with higher priority to communication delay is selected among the discovered resources. Communication delay between two peers is computed by a network model based on queuing theory, taking into account the background traffic of the Internet. Simulation results show that the proposed resource discovery algorithm improves the response time of user's requests by a factor of 4.04 under a moderate load.

1. Introduction

Volunteer computing in which volunteers donate processing and storage resources is an attractive cost efficient platform for running scientific projects with significant computational requirements (tens or hundreds of TeraFLOPs) [1]. Some of these projects are the SETI@home [2], Folding@home [3], EDGeS [4], DEGISCO [5] and EDGI [6] and Climate@home [7] projects.

Some of the popular volunteer computing systems are BOINC [8,9], condor-like grid system [10], Entropia [11], XtremeWeb [12], Aneka [13], and SZTAKI [14]. Peer-to-Peer (P2P) based volunteer computing (VC) systems represent a decentralized, self-organized and scalable environment for running applications such as PastryGrid [15], BonjourGrid [16], ShareGrid [17], Condor-Flock P2P [18], and Self-Gridron [19].

A fundamental challenge in this large, decentralized and distributed resource sharing environment is efficient discovery of resources described by a set of attributes such as CPU speed, memory, and operating system type. Another challenge comes from these complex environments characterized by large scale and geographically scattered resources. In such an environment, millions of heterogeneous resources are scattered across geographically distributed nodes, therefore a resource discovery architecture with proximity-aware features has great impact on overall performance of the system [18]. There are many resource discovery algorithms in P2P-based volunteer computing systems [20–24], but none of them considers a model in computing communication overhead.

The main contribution of this work is to propose a proximity-aware resource discovery architecture considering quality of service (QoS) constraints of requests, and communication overhead in P2P-based VC systems. In a previous work [25], we presented an architecture for resource discovery in P2P-based VC systems. This architecture is called CycloidGrid, and it distributes a load between peers based on QoS constraints of requests, round trip time (RTT), and current load of resources (for more information refers to Section 3). In this research we focus on a network model based on queuing theory to compute more realistic communication overhead considering the background traffic of the Internet.

The proposed resource discovery algorithm is composed of two stages. In the first stage, resources are selected based on QoS constraints of requests and current load of peers. The QoS constraints can be selected from CPU speed, RAM requirements,

* Corresponding author. Tel.: +98 9151250187.
E-mail addresses: tghafarian@yahoo.com, ghafarian@stu-mail.um.ac.ir (T. Ghafarian), hd@ferdowsi.um.ac.ir (H. Deldari), b.javadi@uws.edu.au (B. Javadi), hyaghmae@ferdowsi.um.ac.ir (M.H. Yaghmaee), rbuyya@unimelb.edu.au (R. Buyya).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.08.010

hard disk requirements, operating system, and processor model. In the second stage, the algorithm selects a resource with higher priority to communication delay among the discovered resources in the first stage. To compute the communication delay between two peers in the system, each connection is modeled as a GI/GI/1 queue. Background traffic of the Internet is considered in the model to generate a more realistic communication delay. In summary, we have the following contributions in this paper:

- We provide a resource discovery architecture in P2P-based volunteer computing systems considering QoS constraints of request and communication overhead.
- We adapt the analytical model for computing communication delay between two peers. The background traffic of the Internet is considered during computation of communication delay between two peers.
- We evaluate the proposed resource discovery architecture under realistic workload models and different numbers of peers to show the scalability of the system.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 discusses the CycloidGrid environment, including the application model, architecture, and churn management. Section 4 presents an analytical model for computing communication delay in CycloidGrid and resource discovery policy in this architecture. Section 5 describes the performance evaluation of the proposed resource discovery architecture under a realistic workload model. Conclusions and future directions are presented in Section 6.

2. Related work

There are several research works that have investigated resource discovery based on load balancing, QoS constraints of request, and proximity-aware features in P2P-based volunteer computing systems. These research works can be divided into two categories: in the first category, the resources discovery approach focuses on load balancing and QoS constraints of requests. Some works in this category consider load balancing and QoS constraints of requests and some of them only consider one of them. The second category focuses on proximity-aware features in resource discovery algorithm. Some of the works on this category consider QoS constraints of request too. In the first category, we highlight the following works:

Kim et al. [20] proposed an approach for load balancing in a resource discovery algorithm in P2P-based desktop grid systems. The resource discovery algorithm has been considered as a routing problem in the CAN [26] space. It searches for a node whose coordinate in all dimensions satisfies or exceeds the QoS requirements of the request. The matchmaking algorithm fairly distributes jobs between capable resources based on aggregated load information along each dimension of the CAN overlay network.

Abdullah et al. [21] suggested a dynamic, self-organizing model for resource discovery in ad hoc grids. In this work, three types of agent, named customer, producer, and matchmaker are introduced. The whole identifier space has been divided into zones, which have a dedicated matchmaker. The matchmaker uses a continuous double auction to perform resource allocation and looks for matches among producers and consumers according to QoS requirements of the request. Load balancing in this system is performed between matchmakers. The authors defined a mechanism to calculate the matchmaker workload (TCost) based on the number of request/offer messages to be processed in the ad hoc grid. TCost with a threshold has been applied for dynamic segmentation and de-segmentation in the ad hoc grid and for balancing a load between different matchmakers.

Lazaro et al. [22] proposed a decentralized resource discovery algorithm that meets QoS constraints of request in P2P-based VC systems. Three main agents (worker, client, and matchmaker) are defined in the system. A worker sends advertisements to multiple matchmakers in the system. When a client needs resources, it asks the matchmaker, which searches among advertisements in order to find possible matches. In this work only the QoS requirements of a request are studied, but load balancing is not considered.

Di et al. [27] presented a decentralized scheduling algorithm for dynamic load balancing in a self-organized desktop grid environment. A dynamic Newscast model has been organized in their work as an unstructured P2P overlay. In this research, each peer gathers load information of its neighbors based on the epidemic gossip protocol. The average load level on participating nodes is used to distinguish overloaded and underloaded nodes in the system. A node is in a load balanced state if its current load is close to average load level, otherwise it is improved by migrating any process into it or out of it. The autonomous scheduler designed on each node performs process migration to balance the workload in the system. The system decreases migration overhead by considering process workload and bandwidth between two relative nodes. The QoS constraints of a request are not considered in their work.

Perez-Miguel et al. [28] suggested a decentralized computing system based on a P2P network. Cassandra [29] is used as a P2P storage system in their work. Cassandra keeps several values, columns for each key. The columns are gathered into different sets called ColumnFamily or SuperColumnFamily. A distributed queue system is used to manage the execution of jobs over its resource pool in a FCFS manner. When a node submits a new job into the system, its related data is inserted into one ColumnFamily called Works. The related files of this job are uploaded into another ColumnFamily called FileStorage. After that, the job is queued into the waiting queue of the Queues SuperColumnFamily. When the job is completed, the results are stored in Cassandra. Each node has a scheduling process to look for the jobs to run. The scheduling process can look up the Queues SuperColumnFamily in time order until it finds a desired job. In their research work, QoS constraints of requests are considered in the resource discovery policy, but load balancing and proximity-aware features are neglected. Also the system is evaluated under a stable situation without any node failures.

Ferretti [30] argued that a simple gossip protocol can be applied as an efficient resource discovery policy on top of an unstructured P2P overlay network. An analytical model based on complex network theory is suggested in his work. Each peer stores local knowledge about itself and its neighbors which are connected to it directly. When a node receives a query, it takes into account its neighbors and selects a subset of them. This subset is selected with regard to their resource items. If their resource items satisfy the query, they are selected. Also other neighbors are selected to spread the query through the P2P overlay. The suggested analytical model is capable of estimating the number of nodes receiving the query, the number of query hits, and the number of resource items can be discovered. QoS constraints of requests are considered in the resource discovery algorithm, but load balancing and proximity-aware features are ignored.

In the second category there are a few research works that consider proximity-aware features in the resource discovery algorithm. Some of these works are:

Mastroianni et al. [24] suggested a super-peer based resource discovery algorithm for P2P-based volunteer computing systems. Their algorithm consists of two phases: job-assignment and data-download phase. In the job assignment, a job manager generates a number of job's adverts based on a job's QoS requirements and sends them to the local super-peer and some of the other super-peers in the system. Workers generate a job query and this query travels the network to find a matching job's advert. In the data-download phase, the worker sends a data query and downloads

a data file from the data centre. In this work, the data file of each job is downloaded taking into consideration the distance and available bandwidth to decrease the communication delay in the system.

Merz et al. [31] proposed a self-organizing super-peer overlay network with a Chord [32] ring core that interconnects the super-peers. Authors managed their proposed overlay with a proximity-aware distributed algorithm. They improved Chord's ID assignment by allocating IDs to joining peers in a delay optimized fashion. The Chord's message routing scheme is modified to reduce end-to-end delay for both uni-cast and broadcast messages. A network coordinate is used for RTT estimates. The resource discovery algorithm in this work is implemented based on routing in the proposed P2P network and reduces the number of hops required to deliver a message.

OurGrid [33] introduced a peer-to-peer based resource sharing environment through which labs with idle resources donate them to labs with computational demands. This system considered the Bag of Tasks application. The scheduler of OurGrid considers the QoS constraints of a task such as storage size requirements, operating system, and architecture. A storage affinity metric is used in this system for data reutilization. The storage affinity of the task to a site is the number of bytes within the task input dataset that are already stored in the site. By using the storage affinity metric and replication, this system has tried to decrease the communication overhead and turnaround time of the jobs in the system.

Butt et al. [18] described a P2P-based flocking scheme that self-organizes the distributed Condor pools into a Pastry [34] overlay network. This architecture locates nearby resource pools in the physical network for flocking. In this system, a locality-aware routing table of Pastry is suggested. This system guarantees, if willing Condor pools are available nearby, that jobs will not be shipped across long distances in the network proximity space.

Chmaj et al. [35] proposed a P2P-based computing system. The two types of element are defined in their system as node and tracker. The nodes are regular machines that do computations on data block and swap result blocks with another nodes. However, the tracker is a central node that contains the source blocks and the location of available result blocks in the system. Each node sends a request to the tracker to get a data block, and then it can do computation on it. After that, the availability of its result block is informed to the tracker. When any node needs a result block, it can send a request to the tracker and get the locations of the desired result block. Then the node can download it from one of these, based on a decision policy. Their system has a central resource discovery mechanism, because the tracker has information of all available nodes and sends the data block to them on its request. All nodes get the source block from the tracker, but exchanging the result block between them is done directly over the P2P overlay. This system considers QoS constraints of requests such as CPU speed. Also their system takes into account proximity-aware features. Each node can download the desired result block based on link speed or the number of hops. Load balancing is neglected in their work.

3. CycloidGrid: Proximity-aware resource discovery architecture in peer-to-peer based volunteer computing systems

In this section CycloidGrid is discussed in detail. CycloidGrid is an architecture for resource discovery in P2P-based volunteer computing systems. Routing of requests in this architecture is done by Cycloid [36] as the P2P overlay. This P2P network is discussed in the following section briefly.

3.1. Cycloid

Cycloid [36] is a constant-degree structured P2P overlay with $n = d \cdot 2^d$ nodes, where d is a dimension. It takes a time complexity $O(d)$ hops for the lookup request with $O(1)$ neighbors per node. Each node in the Cycloid is presented by a pair of indices $(k, a_{d-1}a_{d-2}\ldots a_0)$, where k is a cyclic index and $a_{d-1}a_{d-2}\ldots a_0$ is a cubical index. The cyclic index is an integer number from 0 to $d-1$ and the cubical index is a binary number ranging from 0 to 2^d-1. All nodes are classified into some clusters in this P2P overlay. However, all clusters are ordered by their cubical indices mod 2^d on a large cycle, while, inside each cluster, nodes are ordered by their cyclic index mod d. The largest cyclic index in a local cycle is called a primary node of the local cycle. For more detail on Cycloid readers can refer to [36].

3.2. Resource and application models in CycloidGrid

Each resource in CycloidGrid refers to any volunteer resource in VC systems (e.g. desktop, laptop, tablet computers, smart phones and servers) [37]. These volunteer resources are connected through the Internet and owned by public (volunteers). Also, they have intermittent Internet connectivity or they are permanently connected [38]. These resources are heterogeneous in terms of CPU speed, RAM, hard disk size and network bandwidth. Resource and peer are used interchangeably in this paper.

The Bag of Tasks (BoT) application model is used in this research. So, each job consists of independent parallel tasks. Each application is assigned to one resource (peer) in this research. Because some of resources in VC systems have less connectivity [38] (e.g. wireless connection), many tasks are assigned at once to keep the resource busy until the next connection.

3.3. CycloidGrid architecture

Three types of node are defined in CycloidGrid. These nodes are called *reporting* node, *host* node and *client* node. The reporting node is responsible for keeping resource attributes of each peer in the system. The host node has two roles in the system. It runs associated jobs and acts as a scheduler when it receives a lookup request. So each host node is a component of the distributed scheduling system in CycloidGrid. The client node has a request for running a job, and sends a lookup request for a BoT application execution. This node keeps executable code of a BoT application, input files and generated output files.

Each resource in the system is described by a set of attributes. These attributes are CPU speed, the amount of RAM, available hard disk size, operating system, and processor model. The classification of resource attributes in this architecture is done by a decision tree (DT). A decision tree is a tree-structured plan of a set of attributes to test in order to predict the corresponding cluster. In the decision tree, a non leaf node is labeled with an attribute and the arcs out of a node are labeled with possible values for that attribute. The leaves of the tree are labeled with the classification. The proposed DT is illustrated in Fig. 1. These attributes are static and do not change during the life time of a resource. Four attribute values are selected in each level. This number of attribute values is a trade off between the number of clusters and covers various values for these attributes. Consequently, the number of clusters in the DT is $4^5 = 1024$ clusters. Information of all resources with similar attribute values is gathered in the same cluster of the DT. The clusters of DT occupy several clusters (the first 1024 clusters) in CycloidGrid and are called *reporting clusters*.

The remaining clusters are called *host clusters*. Consequently, we have two types of cluster in CycloidGrid: reporting clusters and host clusters. CycloidGrid has a cluster-based structure, so

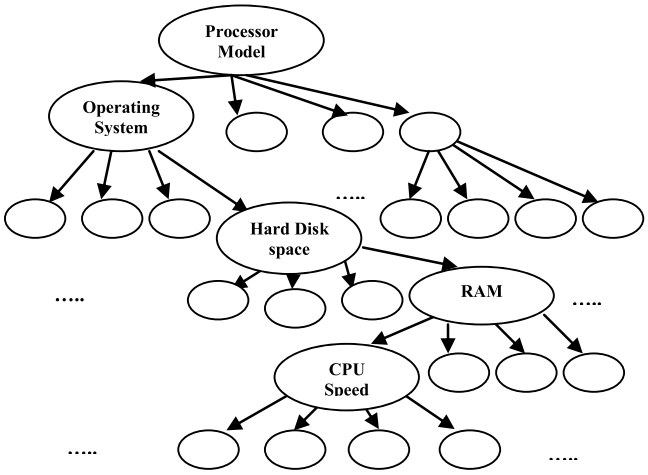

Fig. 1. Decision tree for classification of resources.

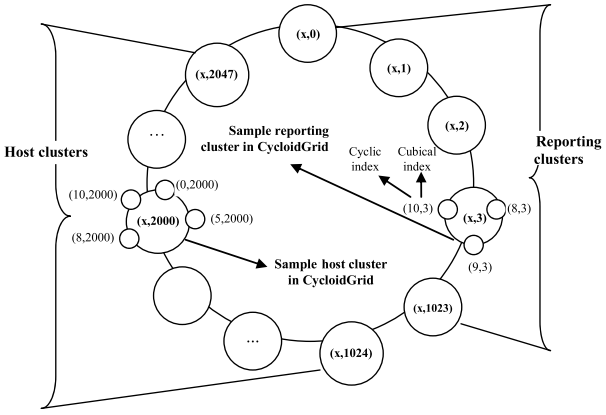

Fig. 2. The organization of clusters in the 11 dimensional CycloidGrid.

the Cycloid P2P overlay with hierarchical structure is preferred to other P2P systems in this research. Although, it is applicable on other P2P overlays.

As we mentioned earlier, the system has three types of node in the system. The reporting clusters keep reporting nodes and the host clusters contain host/client nodes. Each reporting cluster contains three reporting nodes with similar resource attributes. In fact we have a replication factor 3 in each reporting cluster. The authors in [23] showed that we can obtain 99.9% of data availability using a replication factor of 3 in P2P based opportunistic grids. One of these reporting nodes is called *primary reporting node* and has the largest cyclic index in the corresponding cluster and the other ones are called *replica reporting nodes*. Replica nodes have a snapshot of the primary node's resource attributes. When a primary node leaves the system one of the replica nodes can take the role of the primary node based on an election procedure. The role of primary and replica nodes is discussed in detail in the following section. The organization of clusters in CycloidGrid is shown in Fig. 2.

3.4. Churn management in CycloidGrid

When a node joins the system, it can be used either as reporting node or host node. At first the resource attribute values of a new node are imported into the DT. The DT determines the responsible reporting cluster to keep its resource information. If its reporting cluster is empty (inactive reporting cluster) or the number of reporting nodes in this cluster is lower than the replica factor, this node joins as a reporting node; otherwise it joins the system as a host node. The reporting cluster becomes active in the system when there is related resource information to keep. Most of the time the number of active reporting clusters in the system is very much smaller than 1024 clusters. Fig. 3 shows the percentage of active reporting nodes with increasing number of peers in the system.

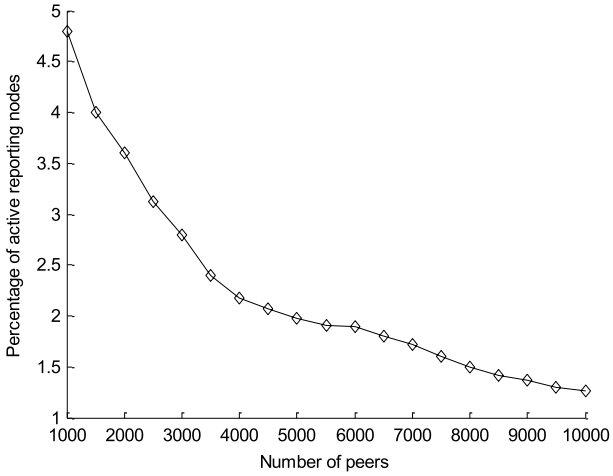

Fig. 3. The percentage of active reporting nodes versus number of peers in the system.

If a node joins as a reporting node and its reporting cluster is empty, it will be a primary node in its reporting cluster; otherwise it will be a replica node in its reporting cluster. If a node joins as a host node, it will get a node identifier by consistent hashing of its IP address. Then, it should report resource information to the primary node of its reporting cluster.

The primary node of each reporting cluster periodically sends a request to the host node belonging to its cluster. Those host nodes respond to this request by sending the current queue length. Then the primary node updates resource information with the current queue length and deletes unavailable resources from its resource information. After that, it sends a snapshot of the resource information to the replica nodes. So the replica node refreshes resource information with the last resource information periodically.

Primary or replica nodes can respond to the lookup request sent by host nodes. In the time-interval between two consecutive updates, a replica node responds based on the last snapshot of its cluster.

When a node leaves the system, the behavior of the system is different for a host node from a reporting node. If a host node leaves the system, all the waiting jobs in its queue should be rerun by another active host node in the system. This situation is recognized by the client nodes of these failed jobs, since a client node sends heartbeat messages to the host node running its job.

If a reporting node leaves the system, the behavior of a system depends on the reporting node type (primary node or replica node). If a replica node leaves the system, the primary node of its reporting cluster recognizes this situation. The primary node sends a snapshot of its resource information to the replica nodes periodically. If it gets no acknowledgment from the replica node, it selects an active host node randomly. The host node changes its role from host node to reporting node. All of the waiting jobs on this new reporting node will run, but it will accept no new jobs from now on. The new replica node gets a snapshot of the primary node and acts as a replica node after that.

If a primary node leaves the system, replica nodes of this cluster can discover it, because replica nodes periodically receive a snapshot from primary node. If a replica node recognizes this situation, it will start an election. The election procedure is

discussed in the following section. After an election is finished, the winner among the replica nodes accepts the primary node's role. The system is recovered from primary node disappearing by two time-intervals between getting two consecutive snapshots. In the first time-interval, any replica node waits for next snapshot. When the first time-interval is over and the replica node has not got any snapshot, an election is started in the second time-interval. So, at the end of second time-interval, the system is recovered. The system can guarantee that reporting clusters always have primary and replica nodes, because the leaving of these nodes from the system is replaced by the leaving of an active host node as soon as this situation is recognized.

3.4.1. Election procedure

Distributed election runs in each replica node in two steps. In the first step, a replica node sends an election start message to other replica nodes in its reporting cluster. Each election start message has a timestamp with the cyclic index of its replica node. In the second step of the distributed election, each replica node receives some election start messages. If a replica node gets the election start message with a timestamp higher than its cyclic index, it leaves the election and waits for the election end message from a new primary node. However, if a replica node receives election start messages with a lower timestamp of its cyclic index, it will be a winner of the election and will send an election end message to other replica nodes. In other words, a replica node with higher cyclic index always wins the election. The election procedure is done in the time-interval between two consecutive updates of the primary node's resource information.

3.5. Security issues

Reporting clusters contain a primary node and replica nodes. These nodes keep resource information and their corresponding identifier in their index. So, each reporting cluster is susceptible to an index poisoning attack [39] in P2P systems. This kind of attack can be done by inserting a massive number of bogus records into the index of reporting nodes. In such an attack an attacker can have two roles in CycloidGrid. If it is a host node, it can send its resource information with a non-existent identifier to the primary node of its corresponding reporting cluster. Its corresponding reporting cluster is determined by the decision tree. The inserted bogus record will be removed from the primary node index after the first update, because the primary node checks the availability of each resource along with the current queue length in its index periodically and it removes unavailable resources from that.

If an attacker gets the reporting node role, it can be the replica or primary node of its reporting cluster. In the case of a replica node, it can respond to the lookup request with the dummy identifier of a non-existent resource. Replication with majority voting policy, widely used in BOINC [8], can be applied to overcome this situation. In this policy, a lookup request is sent to two replicas in each reporting cluster. If the results of them are the same it is accepted, otherwise the lookup request is sent to a third replica. This is repeated until two results are the same or some limit number of lookup requests is reached.

An attacker can get primary node role if it is the first node in its reporting cluster or wins the election. In this case a modified version of policy proposed by Liang et al. [39] can be used. In this policy, it is assumed that each replica node keeps a reputation value for its primary node. The reputation is initialized to zero, and increased during the lifetime of the primary node. As we discussed in Section 3.4, the primary node sends a snapshot of its resource information to replica nodes periodically. This snapshot includes any update on its index, such as new added resources or removed resources from its index. When a replica node gets the snapshot, it checks the validity of this snapshot. So it selects a subset of newly added resources based on the reputation value of its primary node and checks their availability. If the reputation value is zero (new primary node), all of new added resources are checked for availability. The size of this subset is decreased with increasing reputation value of the primary node. Then, for the selected subset of resources a *poison level* is computed. The poison level of any snapshot is defined as the fraction of unavailable newly added resources to all of the new resources in the selected subset of this snapshot. If the poison level is higher than a threshold, the replica node determines the primary node as an attacker. The high poison level shows that there are a massive number of bogus records in the received snapshot. This threshold is determined based on the churn rate in the system, because after primary node updates its index, some of nodes may leave the system and are unavailable. If the replica node recognizes a primary node as an attacker, it sends a deny message to all of the remaining replica nodes in its cluster. If any replica node gets two or more deny messages, a new election is started and the previous primary node is blacklisted. The reputation of the primary node is increased in each replica node with the inverse of its poison level after getting each snapshot. This policy can be applied for a non-colluding attacker.

If an attacker submits bad results after running its job, some kind of sabotage-tolerance mechanism [40,41] in volunteer computing systems can be used in the client nodes. The implementation of this security issue is left for the future.

4. Resource discovery algorithm in CycloidGrid

In CycloidGrid, each request (job) containing some tasks is executed within a single peer. Each request has the following characteristics:

- Number of independent tasks
- Estimated duration of each task
- QoS constraints of this job in terms of minimum CPU speed, RAM, hard disk requirements, operating system, and processor model. The number of QoS constraints varies between zero to five.

Each request is served in the system within two stages. During the first stage, a subset of resources is advertised by the reporting nodes. In this stage, each queried reporting node selects a resource with shorter queue length and more CPU speed within a subset of resources satisfied the QoS constraints of this request. In the second stage, the proximity of advertised resources in the previous stage to the requester is considered. So, the final resource for running the request is selected with higher priority to communication overhead and lower priority to the criteria considered in the first stage. The computation of a communication delay and resource discovery algorithm is discussed in detail in the following sections.

4.1. Analytical queuing model to compute communication delay in P2P-based volunteer computing systems

Since the interconnection network plays a critical role in the overall performance of P2P-based volunteer computing systems, a more realistic approximation of communication delay in these systems is very important. In this section, each connection between two peers is modeled by a network model based on queuing theory. The store & forward flow control mechanism is considered for each connection. With store & forward flow control, each peer along a route between two peers waits until a packet has been completely received (stored) and then forwards the packet to the next peer. The distributed hash table (DHT) of Cycloid is used for routing a request between two peers.

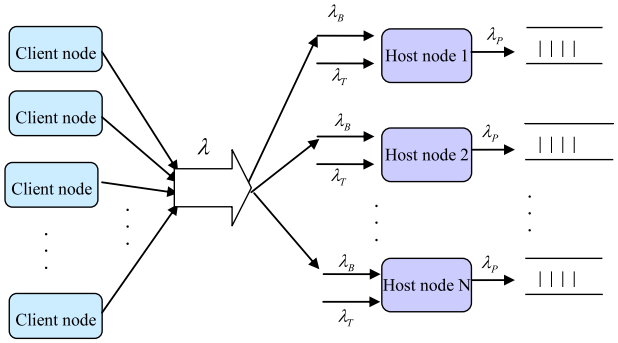

Fig. 4. Queuing model for computing communication delay.

In this analytical model each connection between two peers is modeled by a GI/GI/1 queue. According to [42] the average message latency or communication delay ($\bar{L}$) between two peers consists of three parts; the average waiting time at the source peer's queue ($\bar{W}$), the average network latency ($\bar{T}$) and the average time for a request to reach the destination peer ($\bar{R}$)

$$\bar{L} = \bar{W} + \bar{T} + \bar{R} \tag{1}$$

In this model, we assume each peer in the P2P-based VC systems receives two types of traffic, as shown in Fig. 4. The first part of traffic is the background traffic of the Internet in volunteer computing systems. The second part of traffic is some portion of traffic loaded to the system by client nodes (workload). The background traffic of the Internet is assumed to have Weibull distribution [43] with inter-arrival rate λ_T and variance $\sigma^2_{I_T}$. In this model, BoT requests arrive into the system from the client nodes with inter-arrival rate λ and variance σ^2_I. These requests are redirected to the host nodes. The host node is selected among the active host nodes randomly; however the probability of receiving any of these requests by any host node (peer) in the system is $P_{=}\frac{1}{N}$, with N equal to the number of peers in the system. Therefore each peer receives BoT requests by inter-arrival rate λ_B as follows:

$$\lambda_B = P\lambda = \frac{\lambda}{N} \tag{2}$$

and its variance can be computed by Wald's equation [44]:

$$\sigma^2_{I_B} = \frac{\sigma^2_I P + (\lambda)^{-2}(1-P)}{(P)^2}. \tag{3}$$

As we discussed earlier the input traffic at each peer is composed of two parts. The inter-arrival rate λ_P and variance $\sigma^2_{I_P}$ of a compound distribution at each peer can be computed as follows [45]:

$$\lambda_P = \frac{\lambda_T \lambda_B}{w_T \lambda_B + w_B \lambda_T} \tag{4}$$

$$\sigma^2_{I_P} = w_T \sigma^2_{I_T} + w_B \sigma^2_{I_B} + w_T \left(\frac{\lambda_P - \lambda_T}{\lambda_P \lambda_T}\right)^2 + w_B \left(\frac{\lambda_P - \lambda_B}{\lambda_P \lambda_B}\right)^2, \tag{5}$$

where w_T, w_B are mixing parameter and $w_T + w_B = 1$.

We consider the following service time for each connection between two peers based on the store & forward flow control mechanism [42].

$$S_P = 0.5\alpha_{net} + F\beta_{net}, \tag{6}$$

where α_{net} is the network latency, and β_{net} is the inverse of bandwidth along the link between two adjacent peers based on the routing algorithm in the P2P network. F is a flow size transmitted between two peers.

In Eq. (1), the average of network latency [42] is equal to:

$$\bar{T} = S_P. \tag{7}$$

The average waiting time at the source peer's queue can be computed by the following equation [46]:

$$\bar{W} = S_P + \frac{C^2_{I_P} - C^2_{S_P}}{2((S_P)^{-1} - \lambda_P)}, \tag{8}$$

where $C^2_{I_P}$ and $C^2_{S_P}$ is the squared coefficient of variance for inter-arrival time and service time at the source peer's queue. The variance and coefficient of variance for service time equal to zero based on [42].

$$C^2_{I_P} = \sigma^2_{I_P} \lambda^2_P \tag{9}$$

$$\bar{W} = S_P + \frac{\sigma^2_{I_P} \lambda^2_P}{2((S_P)^{-1} - \lambda_P)} \tag{10}$$

and

$$\bar{R} = \sum_{i=s+1}^{d} RTT_i. \tag{11}$$

Eq. (11) estimates the average time for the flow to reach the destination peer as a summation of RTTs along the route between peers next to the source node and destination peer based on the routing algorithm in the P2P overlay. The Vivaldi algorithm [47] is used to predict the RTT between two peers in the system. Vivaldi is a simple, adaptive, and decentralized algorithm which computes a synthetic coordinate for the Internet host. In this algorithm, the synthetic coordinate is computed in some coordinate space for each Internet host. The Euclidean distance between two hosts' synthetic coordinates is used to predict the RTT between them in the Internet. In this algorithm, each node i adjusts its coordinate after getting some RTT samples of remote nodes in the network. At first it initializes its coordinate (x_i) at the origin of coordinate space. Whenever a node i communicates with a remote node j, it adjusts its coordinate in a few steps. First of all, this node measures the actual RTT (rtt) to the remote node j and gets the remote node's current coordinates x_j. The error in its current prediction to the remote node j is calculated based on Eq. (12).

$$e = rtt - \left\| x_i - x_j \right\|. \tag{12}$$

This node moves toward or away from a remote node j based on the magnitude of error e. So, node i adjusts its coordinate by moving a fraction of distance to the remote node j based on the following equation.

$$x_i = x_i + \delta \times e \times u(x_i - x_j). \tag{13}$$

$u(x_i - x_j)$ is a unit vector and determines the direction of movement. δ is a time-step and it is controlled the rate of convergence. Each node movement decreases the node's error with respect to one remote node in the system. Little by little, the nodes converge to the coordinates that predict RTT well by communicating with other nodes in the system continually. In fact, each new node can find a good coordinate for itself after getting a small number of RTT samples from remote nodes.

If the flow size between two peers is small, such as sending a lookup request, the Euclidean distance between two synthetic coordinates will be a good estimate with low overhead for the average communication time for the flow between two peers. In the case of a large flow size (input/output dataset), RTT in Eq. (11) is replaced with S_P as is shown in Eq. (14) to give a more realistic estimate along the route between peers next to the source node and destination peer.

$$\bar{R} = \sum_{i=s+1}^{d} S_{P_i}. \tag{14}$$

4.2. Proposed resource discovery algorithm in CycloidGrid

Fig. 5 illustrates a scenario in which a peer is selected for running a BoT request. Resource discovery in the proposed architecture is done in two stages. The first stage includes the following steps:

A client node sends a lookup request for its request to an active host node in the system (1). The active host node is selected randomly. The selected host node is called the *injection node*, and acts as a scheduler for this request. In fact, the injection node has two queues. One queue belongs to the lookup request and another one belongs to the jobs which should be executed on this node. The analytical model discussed in Section 4.1 is applied on the first queue. Each injection node has a copy of the decision tree discussed in Section 3.3 and uses it to find which reporting clusters can be useful to search according to the QoS constraints of this request. In this phase, reporting clusters with attribute values higher than the minimum QoS requirements (such as a reporting cluster with more speed and RAM) will be found in addition to reporting cluster with the minimum QoS constraints. This is done since host nodes with the minimum QoS requirements may be overloaded, while the host nodes with higher values may be underloaded.

As mentioned in Section 3.3, each reporting cluster has one primary and two replica nodes with the same resource attributes. In this phase, the injection node selects closer nodes among the reporting nodes in each selected reporting cluster. In fact the injection node computes a communication delay based on Eq. (1) between itself and reporting nodes in each selected reporting cluster and selects two closer ones. To consider security issues the lookup request is sent to these two reporting nodes (2). It is sent to the third one if the results of these reporting nodes are different. Finally, in the case of inconsistent results, this reporting cluster is discarded.

Each reporting node searches in its index to find a resource to satisfy the QoS constraints of this request. The reporting node balances a load among its resources. A ranking algorithm is done in each reporting node. In this ranking algorithm, resources get a rank based on the current queue length, CPU speed and ticket parameter. Ticket is a parameter assigned to each resource in the reporting node locally. When a resource is selected for any request, the ticket parameter is incremented. In fact, the role of this parameter is controlling a convergence to the lowest load resource in the time slice between two consecutive update of queue length. Ticket is reinitialized to zero after each update. The rank of each resource (r_j) is computed by a weighting function as follows:

$$r_j = \frac{w_1 l + w_2 s + w_3 t}{w_1 + w_2 + w_3}. \quad (15)$$

l is a current queue length, s is a scaled value of inverse CPU speed and t is the ticket value for this resource. w_1, w_2, w_3 are weighting coefficients and are selected with respect to the importance of these factors. After computing the rank of each resource, a resource with the lowest rank is selected (3) and the address of selected resource along with its rank are sent to the injection node (4).

The second stage of resource discovery algorithm is done at the injection node, based on Algorithm 1. The injection node selects a resource with lower priority to minimum rank and higher priority to minimum communication delay. In order to optimize these two parameters, Algorithm 1 is used. In this algorithm in Step 1–3 the communication delay ($\bar{L}_j$) is computed for each resource j based on Eq. (1). $\bar{L}_{ij}$ equals the communication delay between the injection node and a resource j and $\bar{L}_{cj}$ equals the communication delay between the client node and the resource j. In Step 4–10, the resource with minimum communication delay is selected and in Step 11–17 the resource with minimum rank is selected. If the selected resource in these two stages is the same, the algorithm returns this resource. Otherwise the resource with next minimum rank is selected until half of the resources are chosen. If half of the resources are selected and none of them is the same as the resource with minimum delay, the resource with next minimum communication delay is selected and this process continues until these two minimums overlap.

The selected resource in the injection node (5) is called the *run node* and the job's profile is sent to the run node. The job is added to the run node's queue in FIFO order and waits for execution (6). When a job is finished, its result is returned back to the client node (7).

Algorithm 1: second stage of resource discovery algorithm in CycloidGrid

Input: r_j rank of all recommended resource j at the first phase, $1 \le j \le N$
(*N=number of peers*)

Output: index of selected resource ($1 \le k \le N$)

1. **foreach** *resource j* do
2. $\bar{L}_j = \bar{L}_{ij} + \bar{L}_{cj}$
3. **end**
4. min ← max Value
5. **foreach** *resource j* **do**
6. **if** ($\bar{L}_j <$ min) **then**
7. min ← $\bar{L}_j$
8. min Delay ← j
9. **end**
10. **end**
11. min ← max Value
12. **foreach** *resource j* **do**
13. **if** ($r_j <$ min) **then**
14. min ← r_j
15. min Rank ← j
16. **end**
17. **end**
18. **if** *(minDelay==minRank)* **then**
19. *k= mindelay*
20. **return** k
21. **end**
22. **else**
23. **if** half of resources are not selected **then**
24. find next minimum rank and goto 11
25. **else** find next minimum communication delay and goto 4
26. **end**

5. Performance evaluation

In order to evaluate the performance of the proposed resource discovery algorithm, we implemented CycloidGrid simulator as a discrete event simulator. CycloidGrid is written in Java and it is an extended version of Cycloid simulator [36] to emulate the P2P-based volunteer computing systems.

The physical network in CycloidGrid is emulated by the Brite topology generator [48]. A physical network with n computers, which are connected by the Waxman model and different link bandwidth, are generated by the Brite topology generator. The bandwidth between two nodes is 10 Mbps to 1 Gbps with uniform distribution [49,9]. The Vivaldi algorithm [47] is used to compute the synthetic coordinate of each node in the physical network to predict RTT between them. A 2-dimensional Euclidean model with height vector is used in the Vivaldi algorithm in this research.

Xtremlab trace [50] is used to emulate resources in the CycloidGrid simulator. The trace is exported from the BOINC database, and its information is collected by a client or server in BOINC.

Two performance metrics are considered in this research. The first one, related to the overhead of the system, is the average number of messages exchanged in the system per each job. The second one, related to the response time of requests, is the average response time (*ART*). As we mentioned in Section 3.2, each BoT

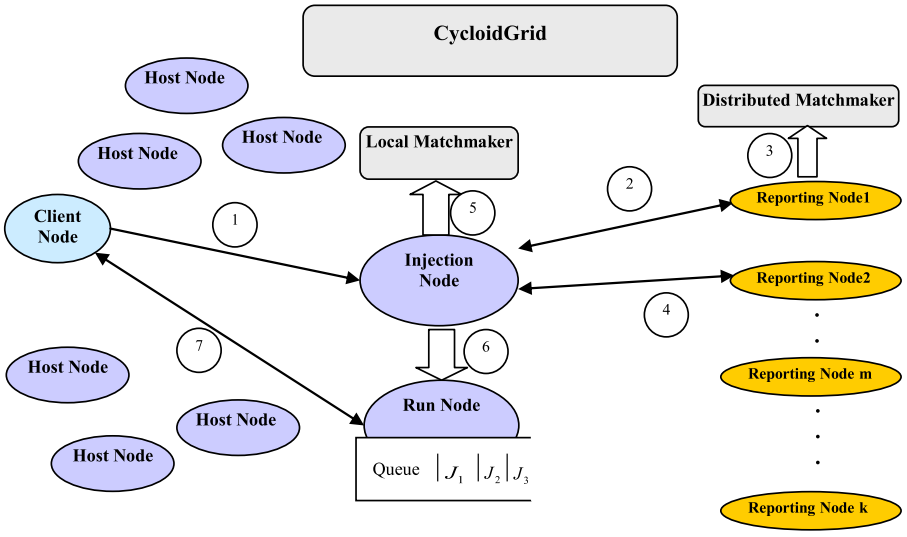

Fig. 5. Resource discovery architecture for CycloidGrid.

application is assigned to one resource. The *ART* of *R* given requests is defined as follows:

$$ART = \frac{\sum_{j=1}^{R}\left(w_j + \sum_{k=1}^{l_j} d_{k_j}\right)}{R}, \tag{16}$$

where w_j is the waiting time of request j, l_j is the number of tasks in request j and d_{k_j} is the weighted run time of each task k in request j. The weighted run time of each task is a scaled down value of run time on a computer with higher speed. The waiting time of request j is computed by the following equation:

$$w_j = \bar{L}_{ci} + \max(\bar{L}_{ir'}) + \bar{L}_{ir} + \max\left(\bar{L}_{cr}, \sum_{l} d_l\right). \tag{17}$$

In the above equation $\bar{L}_{ci}$ is a communication delay between a client node and an injection node. $\bar{L}_{ir'}$ is the communication delay between the injection node and each of the selected reporting nodes in step (2) of Fig. 5. The maximum communication delay $\bar{L}_{ir'}$ is added to the wait time of the job because the injection node contacts selected reporting nodes in parallel. $\bar{L}_{ir}$ is the communication delay between the injection node and a run node, and $\bar{L}_{cr}$ is the communication delay between the client node and the run node for sending the input data. Therefore, in the last term, the maximum of the communication delay between the client node and the run node and summation of the run time for the waiting tasks in the run node's queue is added to the wait time of the request j. All communication delays are computed based on Eq. (1).

5.1. Workload model

The workload model for experiments is based on Grid Workload Archive [51]. Based on this model, a number of BoT requests are generated. The inter-arrival time and request size have a Weibull distribution while the request duration follows a Normal distribution. These distributions with their parameters are listed in Table 1. As in the volunteer computing systems the task duration is large, the power of two for task duration is considered.

Each BoT request may have some QoS constraints in terms of minimum CPU speed, RAM, hard disk size, operating system, and processor model. Some requests have no QoS constraints while some of them have one or five constraints. These QoS constraints are different for each request.

Table 1
Input parameters for the workload model.

Parameters	Distribution/value	Reference
BoT inter-arrival time	Weibull ($5 \le \alpha \le 9, \beta = 4.25$)	[51]
No. of tasks	Weibull ($\alpha = 2.11, \beta = 1.76$)	[51]
Task duration	Normal ($2.73 \le m \le 9.5, \sigma = 6.1$)	[51]
Inter-arrival time of peer churn	Poisson ($0.66 \le \tau \le 4.83$)	
Internet inter-arrival time (heavy traffic)	Weibull ($\alpha = 0.3, \beta = 0.15$)	[43]
Internet Inter-arrival time (medium traffic)	Weibull ($\alpha = 0.06, \beta = 0.15$)	[43]
Internet flow size	Pareto ($\alpha = 3, \beta = 1.05$)	[43]

We generate the workload for 1 day, where 2.5 h is considered as a warm-up phase to avoid bias before the system reaches the steady-state. Each experiment is performed on each of these workloads separately. For the sake of accuracy, each experiment is carried out several times by using different workloads, and the average of results is reported. In all the reported results the coefficient of variance is less than 0.05 ($CV < 0.05$). The number of resources is equal to 1000 and 3000 peers with heterogeneous computing speeds.

In order to generate different workloads, we modified two parameters one at a time. So to change the inter-arrival time, we modified the first parameter of the Weibull distribution (the scale parameter α) as shown in Table 1. So, the number of jobs increases from 10 000 (i.e. $\alpha = 9$) to 19 000 (i.e. $\alpha = 5$). Also to have requests with different duration, the mean of the normal distribution changes from 2.73 to 9.5, as mentioned in Table 1. The average task duration in the BoT application changes from 44.71 to 126.35 s.

Peer churn is modeled with a Poisson distribution [20] with the average inter-arrival time (τ) varying from 0.66 min to 4.83 min, as presented in Table 1. However, when the average inter-arrival time is 4.83 min or 0.66 min, respectively, 10% or 70% of peers leave the system, while some nodes join the system.

We consider the background traffic of the Internet follows the Weibull distribution [43], as shown in Table 1. The background traffic is considered in two modes: heavy traffic for the working hours and medium traffic for non-working hours. Also the Internet flow size follows the Pareto distribution according to [43]. The mean of the Pareto distribution is considered as the flow size for the Internet traffic.

We assume that each BoT request has an input file. Various studies on scheduling BoT requests adopt a measure to express the

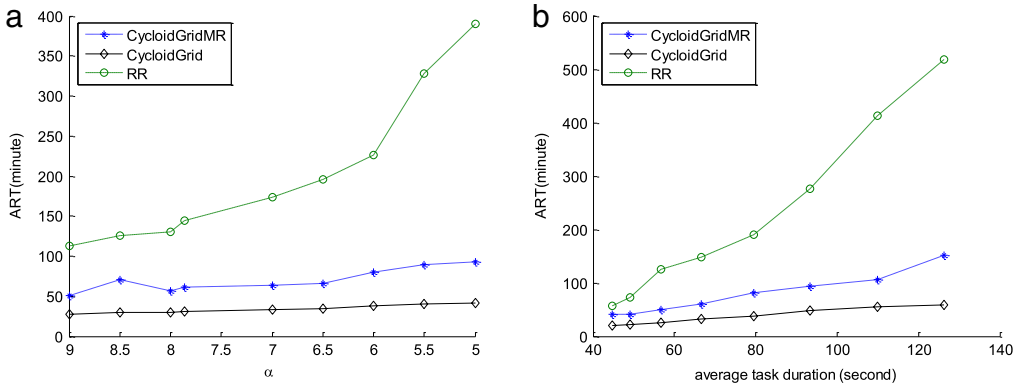

Fig. 6. Average response time resulting from different policies with 1000 peers and heavy background traffic of the Internet. The simulations are carried out by modifying (a) the α parameter in inter-arrival time, (b) the average duration of task in BoT.

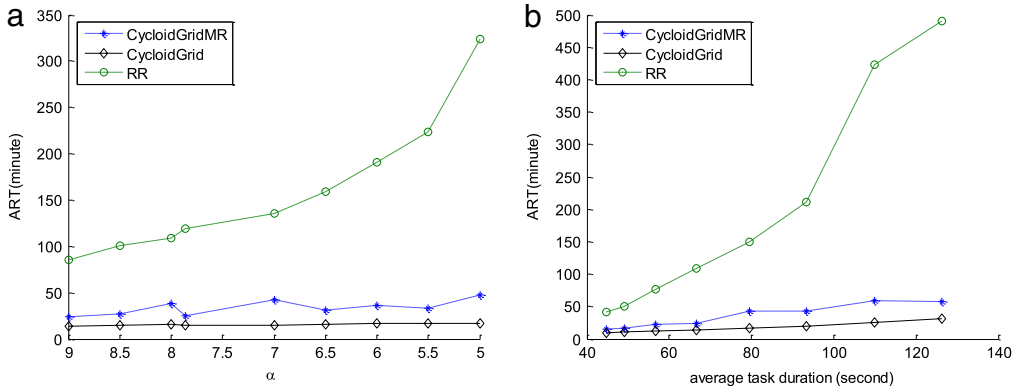

Fig. 7. Average response time resulting from different policies with 1000 peers and medium background traffic of the Internet. The simulations are carried out by modifying (a) the α parameter in inter-arrival time, (b) the average duration of task in BoT.

ratio between the communication and computation costs [52]. It is called communication-to-computation ratio (CCR). So, the file size of each BoT request is considered as CCR times the computation time. It is worth noting that in this study, we concentrate on a BoT application in which both the execution time and communication time are important factors for the job response time. We do not consider the case of negligible communication time (CCR very close to zero). So, balanced BoT requests with computation time and communication time by CCR = 2 are considered.

5.2. Baseline policies

We evaluate the proposed policy against two other policies as follows:

- Round-Robin (RR): The round-robin strategy is used as benchmark strategy for scheduling BoT request in similar studies [52].In this policy, each reporting node in the system selects resources with a round-robin strategy based on QoS of requests (stage 1 of resource discovery algorithm Section 4.2). If a resource does not satisfy the QoS constraints, the next resource is examined in the deterministic sequence. Also, we consider that the injection node selects the target run node randomly among selected host nodes in stage 1 (stage 2 of resource discovery algorithm in Section 4.2).
- CycloidGridMR : in this policy, we consider that resources are selected in each reporting node with the same ranking algorithm of CycloidGrid (stage 1 of resource discovery algorithm in Section 4.2), but in stage 2 of the resource discovery algorithm, the resource with minimum rank is selected.

5.3. Simulation results

The simulation results for ART versus arrival rate and average task duration are shown in Figs. 6–10 for different policies. In these figures, the average task duration is kept in the medium size (66.55 s) for ART versus arrival rate. Also, the inter-arrival time is kept in the medium size (i.e. $\alpha = 7.86$) for ART versus average task duration.

In Fig. 6, there are 1000 peers in the system and the system is relatively static and no peer joins or leaves during the simulation. The background traffic of the Internet is considered heavy according to Table 1. As we expected, by increasing the arrival rate, the ART dramatically increases. CycloidGrid and CycloidGridMR strategies can control the ART by distributing the load evenly in the system. Meanwhile, RR approaches the saturation point exponentially. CycloidGrid marginally surpasses the CycloidGridMR with an improvement factor of 9%, 8% in Fig. 6(a) and (b), respectively. The improvement of CycloidGrid in Fig. 6(a) and (b) with respect to RR is 46% and 37%, respectively.

In Fig. 7, we decrease the background traffic of the Internet to medium traffic according to Table 1. The system is kept in the static state. The improvement factors of CycloidGrid with respect to CycloidGridMR and RR are 6% and 46% in Fig. 7(a) and 4%, 36% in Fig. 7(b).

In Fig. 8, we increase the number of peers to 3000 in the system, while the system is kept in the static state. The background traffic of the Internet is considered heavy in this figure. As is shown, CycloidGrid still achieves a better performance with respect to CycloidGridMR and RR, with improvement factors of 18%, 43% in Fig. 8(a) and 13%, 29% in Fig. 8(b). Fig. 9 shows the simulation results for 3000 peers, in a static environment with

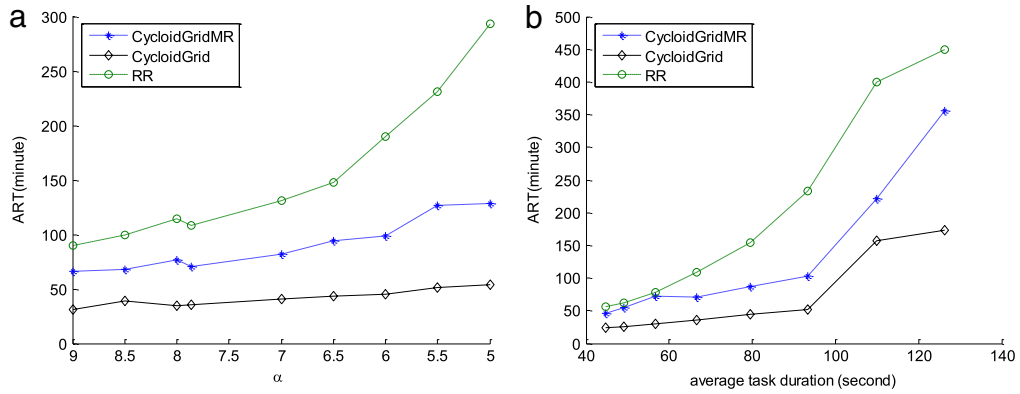

Fig. 8. Average response time resulting from different policies with 3000 peers and heavy background traffic of the Internet. The simulations are carried out by modifying (a) the α parameter in inter-arrival time, (b) the average duration of task in BoT.

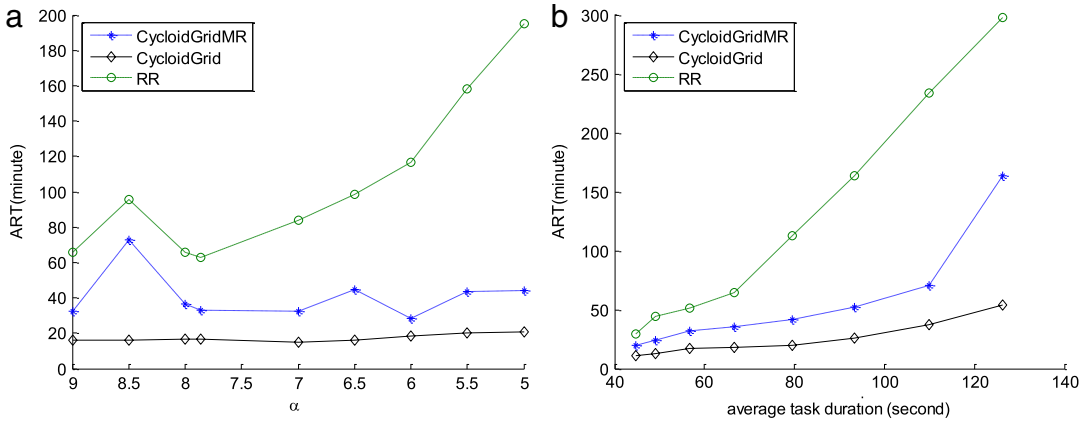

Fig. 9. Average response time resulting from different policies with 3000 peers and medium background traffic of the Internet. The simulations are carried out by modifying (a) the α parameter in inter-arrival time, (b) the average duration of task in BoT.

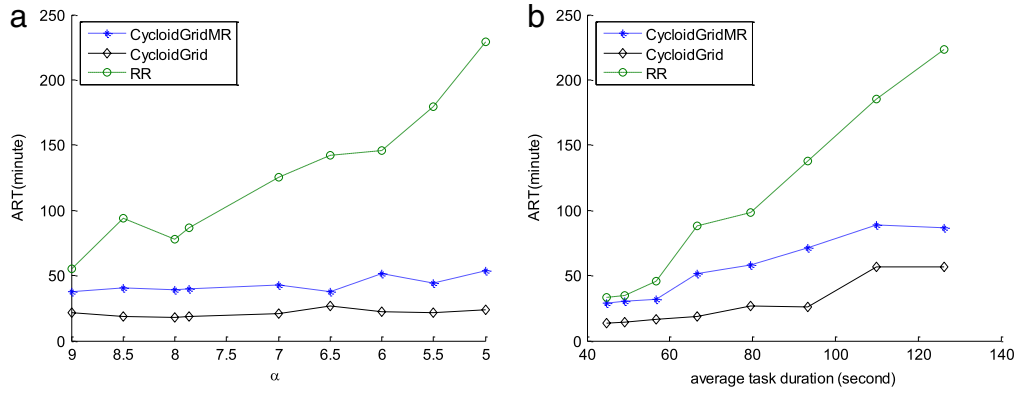

Fig. 10. Average response time resulting from different policies with 3000 peers, medium background traffic of the Internet, and dynamic environments. The simulations are carried out by modifying (a) the α parameter in inter-arrival time, (b) the average duration of task in BoT.

medium background traffic. The CycloidGrid improvement factor is 13%, 48% in Fig. 9(a) and 10%, 34% in Fig. 9(b) with respect to CycloidGridMR and RR, respectively.

In Fig. 10, the number of peers is the same as the previous simulation, but peers join or depart from the system with the average inter-arrival time $\tau = 2.38$ min. In this scenario, after the initial joining of 3000 peers into the system, some peers leave while some peers join the system. The departure rate of peers is 20% of all peers in the system. When a peer leaves the system, all the assigned jobs on the leaving node are reassigned to another peer. CycloidGrid surpasses CycloidGridMR and RR, with the improvement factor of 10%, 49% in Fig. 10(a) and 13%, 36% in Fig. 10(b). In this scenario, the performance of CycloidGridMR and RR decrease more than CycloidGrid compared to the same scenario with a static environment. When a peer leaves the system, some of the jobs on this node should be reassigned. If the system is in the balanced state, the leaving peer does not have a noticeable impact on the average response time. However, in the system with an unbalanced state, a leaving busy node has more influence on the average response time. CycloidGrid selects resources by taking into account the communication overhead and current queue length. The communication overhead parameter gives a kind of

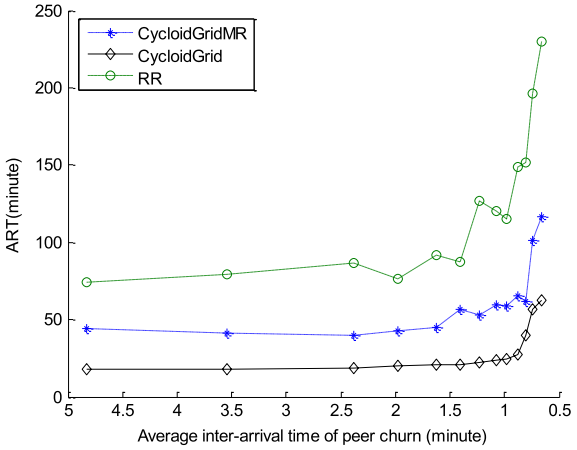

Fig. 11. Average response time resulting from different policies with 3000 peers, medium background traffic of the Internet. The simulations are carried out by modifying the average inter-arrival time of peer churn.

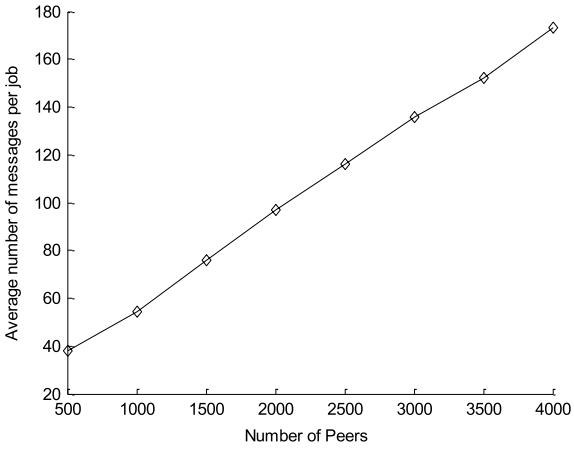

Fig. 12. Average number of messages exchanged in the system per job.

randomness to this policy, and avoids the orientation of the system to the lower queue length resources. This leads to a more balanced system compared to other policies.

CycloidGrid has better performance than CycloidGridMR, especially in heavy background traffic. According to Algorithm 1, CycloidGrid gives a higher priority to communication overhead than resource's queue length to select the run node, while CycloidGridMR only decides based on the resource's queue length. Heavy background traffic decreases λ_P (based on Eq. (4)), and increases the impact of the communication delay in comparison with medium background traffic.

If the number of peers is increased and the background traffic of the Internet remains unchanged, according to Eq. (2), (4) λ_B, λ_P decreases. This reduction has a positive impact on the communication delay. So, CycloidGrid with a higher priority to communication delay has better performance in comparison to CycloidGridMR. RR selects a run node in the second stage of resource discovery randomly, so an increase in the number of peers or background traffic of the Internet does not have a great impact on its performance.

Fig. 11 presents ART for 3000 peers with medium background traffic. The average inter-arrival time of peer churn varies from 0.66 min to 4.83 min. However, from 10% ($\tau = 4.83$ min) to 70% ($\tau = 0.66$ min) of all peers (with step of 5%) leave the system while some nodes join the system. In this figure, the inter-arrival time and average task duration of BoT job are kept in the medium size ($\alpha = 7.86$, avg. task duration $= 66.55$ s). As illustrated in this figure, CycloidGrid has stable behavior with decreasing average inter-arrival time till 0.88 min. At this point, 55% of peers leave the system. After that, it slowly approaches the saturation point. In fact CycloidGrid works well under moderate churn. CycloidGridMR starts to oscillate when the average inter-arrival time equals 1.41 min, with 35% leaving of peers, and then increases exponentially. RR tends to oscillate when the average inter-arrival time is 1.98 min, with 25% leaving of peers, and approaches the saturation point exponentially.

Fig. 12 shows the overhead of CycloidGrid through the average number of messages exchanged in the system per each job versus number of peers. These messages are measured in the dynamic environment with the average inter-arrival time $\tau = 1.63$ min in such a way that 30% of peers leave the system while some nodes join the system. The average number of messages per job includes the quota of a job from all of the messages exchanged between primary reporting nodes and host nodes for updating the queue length, the quota of a job from total messages exchanged for election, and the messages are exchanged among client node, injection node, reporting nodes and run node to find a suitable resource for this job. As shown in this figure, the number of messages increases linearly with increasing number of peers in the system.

Dynamism of the system can affect the performance of CycloidGrid and CycloidGridMR, because when nodes depart the system, the information used for load balancing would be stale during two consecutive update of the resource's queue length. It affects CycloidGridMR more than CycloidGrid, since CycloidGridMR relies on queue length for load balancing, whereas CycloidGrid uses proximity in addition to queue length. The effect of proximity on randomness in resource discovery policy causes CycloidGrid to be more resistant to churn than CycloidGridMR.

6. Conclusion

In this paper, we propose a proximity-aware resource discovery architecture in P2P-based volunteer computing systems. We consider a request arriving into the system as the Bag of Tasks, where each request may have QoS constraints such as minimum CPU speed, RAM, hard disk requirements, operating system, and processor model. The resource discovery algorithm has two phases. In the first phase, it takes into account the load balancing and QoS constraints of requests, whereas in the second phase, proximity-aware features are considered and the resource is selected with higher priority to the communication delay. Each connection between two peers is modeled with a network queuing model, and a more realistic communication delay is computed with respect to background traffic of the Internet. We compare the performance of the proposed resource discovery algorithm with two other baseline policies. The results of the experiments indicate that CycloidGrid significantly decreases the average response time of the system with improvement factors of 10.4% and 40.4% on average with respect to CycloidGridMR and RR. The proposed resource discovery architecture is scalable and can tolerate an increasing number of peers without decreasing the performance.

As part of future work, we intend to consider distributed parallel queues and a knowledge-free approach for implementing the load balancing policy instead of getting the current queue length of each peer. A knowledge-free approach does not need any information about the current peer's status, and can decrease the overhead of the system. Another interesting extension would be using Cloud resources in some of the peers. Some jobs have QoS requirements that could not be satisfied by the available resources of the VC systems, thus Cloud resources can be used in order to handle the QoS requirements of these applications.

Acknowledgments

This project was partially supported by Iran Telecommunication Research Centre (ITRC). The authors would like to thank Rodrigo N. Calheiros for useful discussions.

References

[1] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D.P. Anderson, Cost-Benefit analysis of cloud computing versus desktop grids, in: Proceedings of IEEE International Symposium on Parallel & Distributed Processing, IPDPS, 2009, pp. 23–29.
[2] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@home: an experiment in public-resource computing, Commun. ACM 45 (2002) 56–61.
[3] A.L. Beberg, D.L. Ensign, G. Jayachandran, S. Khaliq, V.S. Pande, Folding@home: lessons from eight years of volunteer distributed computing, in: Proceeding of IEEE International Symposium on Parallel and Distributed Processing, IPDPS, 2009, pp. 1–8.
[4] G. Fedak, H. HE, O. Lodygensky, Z. Balaton, Z. Farkas, G. Gombas, P. Kacsuk, R. Lovas, A.C. Marosi, I. Kelley, I. Taylor, G. Terstyanszky, T. Kiss, M. Cardenas-Montes, A. Emmen, F. Araujo, EDGeS: a bridge between desktop grids and service grids, in: Proceeding of The Third ChinaGrid Annual Conference, ChinaGrid, 2008, pp. 3–9.
[5] DEGISCO project, http://degisco.eu/.
[6] EDGI project, http://edgi-project.eu/.
[7] P. Guinnessy, Climate@home, Phys. Today 56 (2003) 38.
[8] D.P. Anderson, BOINC: a system for public-resource computing and storage, in: Proceeding of Grid Computing, Grid, 2004, pp. 4–10.
[9] A. Elwaer, A. Harrison, I. Kelley, I. Taylor, Attic: a case study for distributing data in BOINC projects, in: IEEE International Parallel & Distributed Processing Symposium, IPDPS, 2011, pp. 1863–1870.
[10] D.H.J. Epema, M. Livny, R.V. Dantzig, X. Evers, J. Pruyne, A worldwide flock of condors: load sharing among workstation clusters, Future Gener. Comput. Syst. 12 (1996) 53–65.
[11] A. Chien, B. Calder, S. Elbert, K. Bhatia, Entropia: architecture and performance of an enterprise desktop grid system, J. Parallel Distrib. Comput. 63 (2003) 597–610.
[12] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri, O. Lodygensky, Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid, Future Gener. Comput. Syst. 21 (2005) 417–437.
[13] X. Chu, K. Nadiminti, C. Jin, S. Venugopal, R. Buyya, Aneka: next-generation enterprise grid platform for e-science and e-business applications, in: IEEE International Conference on e-Science and Grid Computing, E-SCIENCE, 2007, pp. 10–13.
[14] A.C. Marosi, G. Gombas, Z. Balaton, P. Kacsuk, T. Kiss, SZTAKI desktopgrid: building a scalable, secure platform for desktop grid computing, Making Grids Work VII (2008) 365–376.
[15] H. Abbes, C. Cerin, M. Jemni, A decentralized and fault-tolerant desktop grid system for distributed applications, Concurrency Computat: Pract. Exper. 22 (2010) 261–277.
[16] H. Abbes, C. Cerin, M. Jemni, Bonjourgrid: orchestration of multi-instances of grid middlewares on institutional desktop grids, in: Proceeding of IEEE International Symposium on Parallel and Distributed Processing, IPDPS, 2009, pp. 1–8.
[17] C. Anglano, M. Canonico, M. Guazzone, The ShareGrid peer-to-peer desktop grid: infrastructure, applications, and performance evaluation, J. Grid Comput. 8 (2010) 543–570.
[18] A.R. Butt, R. Zhang, C.Y. Hu, A self-organizing flock of condors, J. Parallel Distrib. Comput. 66 (2006) 145–161.
[19] E. Byun, H. Kim, S. Choi, S. Lee, Y.S. Han, J.M. Gil, S.Y. Jung, Self-gridron: reliable, autonomous, and fully decentralized desktop grid computing system based on neural overlay network, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA, 2008, pp. 569–575.
[20] J.S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, A. Sussman, Trade-offs in matchmaking job and balancing load for distributed desktop grids, Future Gener. Comput. Syst. 24 (2008) 415–424.
[21] T. Abdullah, L.O. Alima, V. Sokolov, D. Calomme, K. Bertels, Hybrid resource discovery mechanism in ad hoc grid using structured overlay, Lect. Notes Comput. Sci. 5455 (2009) 108–119.
[22] D. Lazaro, J.M. Marques, X. Vilajosana, Flexible resource discovery for decentralized P2P and volunteer computing systems, in: Proceeding of Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, 2010, pp. 235–240.
[23] R.Y. de Camargo, F. Kon, Design and implementation of a middleware for data storage in opportunistic Grids, in: Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGRID, 2007, pp. 23–30.
[24] C. Mastroianni, P. Cozza, D. Talia, I. Kelley, I. Taylor, A scalable super-peer approach for public scientific computation, Future Gener. Comput. Syst. 25 (2009) 213–223.
[25] T. Ghafarian-M, H. Deldari, M.H. Yaghmaee-M, Proximity-aware resource discovery architecture in peer-to-peer based volunteer computing system, in: Proceedings of IEEE 11th International Conference on Computer and Information Technology, CIT, 2011, pp. 83–90.
[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in: Proceedings of ACM SIGCOMM, 2001, pp. 161–172.
[27] S. Di, C.L. Wang, D.H. Hu, Gossip-based dynamic load balancing in a self-organized desktop grid, in: Proceedings of The 10th High-Performance Computing Asia, HPCAsia, 2009, pp. 85–92.
[28] C. Perez-Miguel, J. Miguel-Alonso, A. Mendiburu, High throughput computing over peer-to-peer networks, Future Gener. Comput. Syst. (2011) http://dx.doi.org/10.1016/j.future.2011.08.011.
[29] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, SIGOPS Operating Systems Review 44 (2010) 35–40.
[30] S. Ferretti, Gossiping for resource discovering: an analysis based on complex network theory, Future Gener. Comput. Syst. (2012) http://dx.doi.org/10.1016/j.future.2012.06.002.
[31] P. Merz, S. Wolf, D. Schwerdel, M. Priebe, A self-organizing super-peer overlay with a chord core for desktop grids, Lect. Notes Comput. Sci. 5343 (2008) 23–34.
[32] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for Internet applications, in: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM, 2001, pp. 149–160.
[33] W. Cirne, F. Brasileiro, N. Andrade, L.B. Costa, A. Andrade, R. Novaes, M. Mowbray, Labs of the world, unite!!!, J. Grid Comput. 4 (2006) 225–246.
[34] A. Rowstran, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proceeding of Middleware, in: Lect. Notes Comput. Sci., 2218, 2001, pp. 329–350.
[35] G. Chmaj, K. Walkowiak, A P2P computing system for overlay networks, Future Gener. Comput. Syst. (2010) http://dx.doi.org/10.1016/j.future.2010.11.009.
[36] H. Shen, C. Xu, G. Chen, Cycloid: a scalable constant-degree p2p overlay network, Perform Evaluation 63 (2006) 195–216.
[37] D.P. Anderson, Emulating volunteer computing scheduling policies, in: Proceeding of 2011 IEEE International Parallel & Distributed Processing Symposium, IPDPS, 2011, pp. 1839–1846.
[38] D. Kondo, D.P. Anderson, J. McLeod VII, Performance evaluation of scheduling policies for volunteer computing, in: Proceedings of Third IEEE International Conference on e-Science and Grid Computing, e-Science, 2007, pp. 415–422.
[39] J. Liang, N. Naoumov, K.W. Ross, The index poisoning attack in P2P file sharing systems, in: Proceeding of International Conference on Computer Communications, INFOCOM, 2006, pp. 1–12.
[40] L.F. Sarmeta, Sabotage-tolerance mechanism for volunteer computing systems, Future Gener. Comp. Syst. 18 (2002) 561–572.
[41] F. Araujo, J. Farinha, P. Domingues, G. Cosmin Silaghi, D. Kondo, A maximum independent set approach for collusion detection in voting pools, J. Parallel Distrib. Comput. 71 (2011) 1356–1366.
[42] B. Javadi, J.H. Abawajy, M.K. Akbari, A comprehensive analytical model of interconnection networks in large-scale cluster systems, Concurrency Computat: Pract. Exper. 20 (2007) 75–97.
[43] N. Basher, A. Mahanti, C. Williamson, M. Arlitt, A comparative analysis of web and peer-to-peer traffic, in: Proceeding of International World Wide Web Conference, WWW, 2008, pp. 287–296.
[44] S.M. Ross, Stochastic Processes, John Wiley and Sons, New York, 1997.
[45] H. Rinne, A Weibull Distribution: A Handbook, Chapman & Hall/CRC Press, Taylor and Francis Group, Florida, 2009.
[46] X. Guo, Y. Lu, M.S. Squillante, Optimal probabilistic routing in distributed parallel queues, ACM SIGMETRICS Performance Evaluation Review 32 (2004) 53–54.
[47] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a decentralized network coordinate system, in: Proceedings of The Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM SIGCOMM, 2004, pp. 15–26.
[48] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: an approach to universal topology generation, in: Proceedings of Ninth International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, 2001, pp. 346–353.
[49] M.S. Bouguerra, D. Kondo, D. Trystram, On the scheduling of checkpoints in desktop grids, in: Proceeding of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid, 2011, pp. 305–313.
[50] P. Malecot, D. Kondo, G. Fedak, XtremLab: a platform for observation and characterization Grids of PCs on the Internet, in: Proceeding of Parallel Meetings of The French (RenPar'17 / SympA'2006 / CFSE'5 / JC'2006), 2006.
[51] A. Iosup, O. Sonmez, S. Anoep, D. Epema, The performance of Bags-of-Tasks in large-scale distributed systems, in: Proceedings of The 17th International Symposium on High Performance Distributed Computing, HPDC, 2008, pp. 97–108.
[52] F.A.B. da Silva, H. Senger, Scalability limits of Bag-of-Tasks applications running on hierarchical platforms, J. Parallel Distrib. Comput. 71 (2011) 788–801.

Toktam Ghafarian received the B.S. degree and M.S. degree in computer engineering from Ferdowsi University of Mashhad, Iran in 1999 and 2002 respectively. She is now a Ph.D. student in the Department of Computer Engineering at Ferdowsi University of Mashhad, Iran. From May 2011 to December 2011, she was with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia as a visiting researcher. Her research interests include distributed and parallel computer systems, parallel programming, algorithmic skeletons for parallel genetic algorithms, P2P overlay networks, desktop grid computing.

Hossein Deldari received a B.S. degree in Physics from Ferdowsi University of Mashhad, Iran in 1970, his Masters degree in Computer Science from University of Oregon, USA in 1979 and his Ph.D. in Parallel and distributed systems from University of Leeds, England in 1995. His research interests include distributed and parallel computer systems, parallel programming, parallel algorithmic skeletons, parallel fuzzy genetic algorithms, grid/cluster and cloud computing, and multi-core architectures.

Bahman Javadi is a Lecturer in Networking and Cloud Computing at the University of Western Sydney, Australia. Prior to this appointment, he was a Research Fellow at the University of Melbourne, Australia. From 2008 to 2010, he was a Postdoctoral Fellow at the INRIA Rhone-Alpes, France. He received his M.S. and Ph.D. degrees in Computer Engineering from the Amirkabir University of Technology in 2001 and 2007, respectively. He was a Research Scholar at the School of Engineering and Information Technology, Deakin University, Australia during his Ph.D. course. He is co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems. He has received numerous Best Paper Awards at IEEE/ACM conferences for his research papers. He has served as a program committee of many international conferences and workshops. His research interests include Cloud and Grid computing, performance evaluation of large-scale distributed computing systems, and reliability and fault tolerance.

Mohammad Hossien Yaghmaee was born in July 1971 in Mashad, Iran. He received his B.S. degree in Communication Engineering from Sharif University of Technology, Tehran, Iran in 1993, and M.S. degree in communication engineering from Tehran Polytechnic (Amirkabir) University of Technology in 1995. He received his Ph.D. degree in Communication Engineering from Tehran Polytechnic (Amirkabir) University of Technology in 2000. He has been a computer network engineer with several networking projects in Iran Telecommunication Research Center (ITRC) since 1992. From November 1998 to July 1999, he was with Network Technology Group (NTG), C&C Media Research Labs., NEC Corporation, Tokyo, Japan, as a visiting research scholar. From September 2007 to August 2008, he was with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, USA as a visiting Associate Professor. He is the author of 3 books, all in the Farsi language. He has published more than 85 international conference and journal papers. His research interests are in Wireless Sensor Networks (WSNs), traffic and congestion control, high-speed networks, including ATM and MPLS, Quality of Services (QoS) and fuzzy logic control.

Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He also serves as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercializing its innovations in Grid and Cloud Computing. He has authored and published over 300 research papers and four textbooks. The books on emerging topics that Dr. Buyya edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999), Content Delivery Networks (Springer, Germany, 2008), Market-Oriented Grid and Utility Computing (Wiley, USA, 2009), and Cloud Computing: Principles and Paradigms (Wiley, USA, 2011). He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 52, gindex = 111, 14 500 citations).

Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and four IEEE conferences (CCGrid, Cluster, Grid, and e-Science). He has presented over 250 invited talks on his vision on IT Futures and advanced computing technologies at international conferences and institutions in Asia, Australia, Europe, North America, and South America. These contributions and international research leadership of Dr. Buyya are recognized through the award of the "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. Manjrasoft's Aneka technology for Cloud Computing developed under his leadership received the "2010 Asia Pacific Frost & Sullivan New Product Innovation Award".