J Supercomput (2010) 52: 149–170
DOI 10.1007/s11227-009-0281-x

A dynamic framework for integrated management of all types of resources in P2P systems

Mohsen Sharifi · Seyedeh Leili Mirtaheri · Ehsan Mousavi Khaneghah

Published online: 14 March 2009

Abstract Traditional Peer-to-Peer (P2P) systems were restricted to sharing of files on the Internet. Although some of the more recent P2P distributed systems have tried to support transparent sharing of other types of resources, like computer processing power, but none allow and support sharing of all types of resources available on the Internet. This is mainly because the resource management part of P2P systems are custom designed in support of specific features of only one type of resource, making simultaneous access to all types of resources impractical. Another shortcoming of existing P2P systems is that they follow a client/server model of resource sharing that makes them structurally constrained and dependent on dedicated servers (resource managers). Clients must get permission from a limited number of servers to share or access resources, and resource management mechanisms run on these servers. Because resource management by servers is not dynamically reconfigurable, such P2P systems are not scalable to the ever growing extent of Internet. We present an integrated framework for sharing of all types of resources in P2P systems by using a dynamic structure for managing four basic types of resources, namely *process*, *file*, *memory*, and *I/O*, in the same way they are routinely managed by operating systems. The proposed framework allows P2P systems to use dynamically reconfigurable resource management mechanisms where each machine in the P2P system can at the same time serve both as a server and as a client. The pattern of requests for shared resources at a given time identifies which machines are currently servers and which ones are currently clients. The client server pattern changes with changes in the pattern of requests for distributed resources. Scalable P2P systems with dynamically

M. Sharifi (✉) · S.L. Mirtaheri · E.M. Khaneghah
Iran University of Science and Technology, Teheran, Iran
e-mail: msharifi@iust.ac.ir

S.L. Mirtaheri
e-mail: mirtaheri@comp.iust.ac.ir

E.M. Khaneghah
e-mail: emousavi@comp.iust.ac.ir

reconfigurable structures can thus be built using our proposed resource management mechanisms. This dynamic structure also allows for the interoperability of different P2P systems.

Keywords P2P distributed systems · Integrated resource management · Resource sharing · Framework · Operating system

1 Introduction

Peer-to-Peer (P2P) systems have chosen the Internet as their underlying platform. This is because Internet as a network of networks is a context-based distributed system with unlimited and ever growing resources that may well be shared and are defined by Napster project at first time [1].

P2P systems have been defined in various ways by the P2P community. The Hewlett Packard P2P group has this definition: "*P2P is about sharing: giving to and getting from a peer community. A peer gives some resources and obtains other resources in return*" [2]. The Intel P2P working group defines P2P as "*the sharing of computer resources and services by direct exchange between systems*" [3]. David Anderson [4] calls SETI@home and similar P2P projects that do not involve communication as "*inverted client-server, emphasizing that the computers at the edge provide power and those in the middle of the network are there only to coordinate them*" [4]. Department of Information Technology at Harvard University defines P2P as: "*a method of file sharing over a network in which individual computers are linked via the Internet or a private network to share programs/files, often illegally. Users download files directly from other users' computers, rather than from a central server*" [5]. Shirkey defines P2P as: "*a class of applications that take advantage of resources (such as storage, processing cycles, and human presence) at the edges of the Internet*" [6]. PEPITO project group defines a P2P system as: "*a distributed system based on resource sharing by direct exchange between nodes*" [7]. Blanco et al. define P2P systems as: "*massively distributed and highly volatile systems for sharing large amounts of resources*" [8].

Given the above definitions, we believe in and define P2P systems in this paper from a different perspective: *a P2P system is a kind of distributed system that can potentially utilize all types of available and accessible resources on the Internet*. Being a distributed system, a P2P system consists of a collection of scalable number of computers that are geographically dispersed on the Internet and as a whole appear to each user as a single coherent system. Users transparently access remote resources and share their own resources with other users in a controlled way, hiding the fact that resources are physically distributed across multiple heterogeneous or homogenous computers, and offering their services according to standard rules that describe the syntax and semantics of those services. Note that it is necessary for P2P systems to be scalable [9].

Based on the above exigencies on distributed systems, P2P systems that are a kind of distributed system must pursue the connectivity goal of distributed systems by deploying appropriate resource sharing mechanisms to transparently connect users to all

types of resources. However, current P2P systems fall short of supporting the sharing of all types of resources and only focus on sharing of a particular type of resource with specific features. We claim this is not fair because other types of potentially available resources are denied from users in critical need of such free resources. On the other hand, history has taught us that operating systems have successfully served as managers of all four types of resources, namely *processes*, *memory*, *file*, and *I/O*, in standalone computers, as well as acting as a virtual machine to hide the intricacies of hardware. We believe that a resource manager for a distributed system like a P2P system can serve similar to an operating system by managing all four types of resources that are dispersed on the Internet, as well as serving as a virtual machine to hide the intricacies of resources including their distribution.

Another important shortcoming of existing P2P systems is that they follow a client/server model of resource sharing that makes them structurally constrained and dependent on dedicated servers (resource managers). Clients must get permission from a limited number of servers to share or access resources, and resource management mechanisms run on these servers. Because resource management by servers is not dynamically reconfigurable, such P2P systems are not scalable to the ever growing extent of Internet. We believe that P2P systems must use dynamically reconfigurable resource management mechanisms where each machine in the P2P system can at the same time serve both as a server and as a client. The pattern of requests for shared resources at a given time identifies which machines are currently servers and which ones are currently clients. The client server pattern changes with the changes in the pattern of requests. Such dynamic reconfigurable resource management mechanisms can administer the scalability problem of P2P systems well.

In this paper, we propose a framework for dynamic and distributed management of all four basic types of resources in P2P systems and believe that this will be the trend for next generation P2P systems to come. Evidence for this belief and its practicality is presented in the following sections of the paper.

The remaining sections of the paper are organized as follows. Section 2 presents related work. Section 3 defines the distributed degree criterion for P2P system implementations. Section 4 introduces our proposed resource management framework. Section 5 presents the formation of the structure of the framework, and Sect. 6 concludes the paper.

2 Related work

The main goal of a distributed system is to make it easy for all users to access remote resources and to share them with other users in a controlled way [9]. P2P systems are a kind of distributed system, and thus should be able to manage all kinds of resources (process, memory, file, and I/O) and share them between users. Resource management in P2P systems involves the discovery of other peers and their resources in a distributed environment, as well as routing information in all shapes and forms between peers. To perform these tasks, the resource management unit in a P2P system must adopt a particular resource sharing model. This model can be highly centralized such as in Napster, or highly distributed such as in Gnutella [2, 10]. Without arguing

the benefits and shortages of these two models, let us suffice here to state that we have used a hybrid model in our framework to take advantage of positive points of these two models and to avoid their shortages.

P2P systems have been classified into four groups in general based on their usage [2, 8, 11]

1. *Distributed Computing Systems*. Examples include SETI@home, Avaki [12] and Entropia [13]. These systems share the computing tasks assigned to the system and try to provide aggregate resources necessary to perform all tasks [2, 11].
2. *File Sharing Systems*. Examples include Napster, Gnutella [10], Freenet [14], Publius [15] and Free Haven [16]. These systems aggregate file resources and provide content storage and exchange [2, 17].
3. *Collaboration Systems*. Examples include Magi [18], Groove [19] and Jabber [20]. These systems allow application-level collaboration between users [2, 17].
4. *Platforms*. Examples include JXTA [21] and .NET My Services [22]. These systems provide a system software infrastructure in support of P2P applications [2, 17].

There are several implementations of P2P systems each of which manages a particular kind of resource. Most of them share file resources, some of them share processes and memory to attain high performance, and very few share only especial kinds of I/O. Each P2P system is tailored to specific kinds of resources In other words, there is no implementation that supports the sharing of all types of resources as it is routinely done in operating systems. This means that users should use disparate P2P systems if they want to share all types of distributed resources. But the problem is that the variety of P2P systems in use does not interoperate [11, 23, 24] because they may well differ in important features like architecture, sharing mechanisms, resource management techniques, communication models, and protocols. The resource sharing model in P2P systems is either pure or hybrid. In a pure model, there is no centralized server [25, 26]. In a data-sharing pure P2P system, all nodes are equal and no functionality is centralized. Examples of file sharing pure P2P systems are Gnutella and Freenet, where every node is a "servant" (both a client and a server), and can equally communicate with any other connected node. Pure P2P networks have become quite unpopular though because they generate a lot of traffic to keep the network up and running [10, 27]. These systems are faced with critical problems such as: (1) limited search scope, (2) dynamic nature of peers, and (3) lack of collaboration among peers [26].

In the hybrid resource sharing model, a server is approached first to obtain meta-information such as the identity of the peer on which some information is stored, or to verify security credentials. After that, the P2P communication is performed. Examples of the hybrid model include Napster, Groove, Magi, and iMesh [2, 26]. In the hybrid model, some nodes like global central server nodes have special functionality and can optimize the connections much better than it is done in other resource sharing models. However, scalability and legal issues might plague a centralized architecture. The hybrid model requires powerful servers to run management and search algorithms [27]. These servers are a bottleneck to the system.

There is also an intermediate resource sharing model that uses super peers; KaZaa [28] uses this model. Peers typically lookup information stored in super peers if they

cannot find it elsewhere [2, 25, 27, 28]. This model is similar to the model where clients and servers are not distinguished and any node can be a super peer or ordinary node depending on its capacity and availability [29]. The P2P community is evolving towards this model. This can be seen in the architectures of new P2P applications such as eDonkey and the FastTrack P2P stack, as well as in the moves toward the introduction of super peers into the Gnutella network.

There has been a clear need for Gnutella to evolve [27, 30]. In spite of popularity of hybrid architectures, mediated architectures have additional advantages. One advantage is the business case. Since in most cases home users are responsible for running the super peer, there is no need to invest in expensive server farms [27]. There are two layers in the control plane: one of normal peers connecting to the super peer in a client-server fashion, and one of super peer connected with each other via a pure P2P network.

Communication between super peers generates a lot of expensive interdomain traffic that can be reduced by intelligently building the overlay [27]. These systems have no fixed central directory server but super peer, which are dynamically assigned based on the computer resources of peers. Super peers are powerful peers that provide file indexing services to their connected peers known as nodes, with less computer resources. Unlike the servers in the centralized systems, the super peer keeps a file index of their connected nodes, reflecting only a partial view of the network. The file index records the information including the available files shared by nodes and their corresponding locations. When the nodes start searching, they first search for the files in the file index kept by the super peer. If results are found, they are returned to the nodes at once, so the search performance is better than that of the flooding algorithm [26].

Because of the advantages of super peers and also the special characteristics of the pure model, we have used a mixed model in our framework. Initially at $t = 0$ in the system, there is no super peer. In fact, no primary structure is initially assumed. As time passes, the pattern of requests forms the structure of system in terms of super peers and regions; regions are introduced in Sect. 5.1. The main difference is that this structure is formed dynamically and may well change upon arrival of new requests. This dynamicity allows for more efficient responses to existing requests.

In our framework, different types of super peers are available for each category of four types of resources and the regions of those super peers cooperate to respond to requests of a specific resource. It is quite possible that different policies are defined and imposed on super peers due to the requirements in that region. To allow super peers to control their numbers reduces the overhead caused by super peers' connections, and prevents the formation of big regions.

There have been also some researches to present frameworks for P2P systems with a different purpose than ours. Fries et al. [24] have presented a framework for resource management and information storage in P2P networks. Their framework aims at providing an easy to use and flexible model for P2P computing, encouraging more modularized application development, permitting the reuse of components and the use of graph structure to browse the contents of P2P systems. Lam et al. [31] from Texas A&M University have proposed an accountability management framework for resource sharing in P2P distributed computing systems, too. On the basis of

a generalized e-coin paradigm, they developed a management framework based on the distribution of resource credit/policy, exchange of service and service evidences, and accountability management.

To sum up, we present a new framework for integrated management of resources in P2P systems, which are formulated based on the definition and goals of distributed systems, while using existing best practices. For the first time, in our framework, users transparently get connected to resources to receive different facilities and services like distributed computing, distributed file sharing, distributed I/O sharing, and distributed storage.

3 Distributed degree criterion

A distributed system consists of a group of computers, each of which has an independent operating system and all are connected to each other through a computer network [9]. There is a system software in each of these computers that envisages the whole distributed system as a coherent system to its users. Each user is assumed to be able to connect to and transparently access all available resources in the distributed system as though connecting and accessing his/her own local resources. This definition of distributed systems is very general and covers all types of purposes for user resource connectivity. Users get connected to resources to receive different facilities and services like distributed computing, distributed file sharing, distributed I/O sharing, and distributed storage. There are some implementations of distributed systems, each of which is customized to provide only a particular service. We may classify these implementations into three groups, namely, *Clusters*, *Grids,* and *P2P systems*.

Unfortunately, most existing implementations of Grids, Clusters, and P2P systems had fallen short of fully supporting all features of distributed systems, mainly because of their design decisions or technology limitations, sacrificing some features in favor of other features. We believe that implementations of P2P systems are much closer to the concepts of distributed systems than implementations of Grids and Clusters, although current P2P systems are very special purpose and support the connection of users only to a particular kind of resource.

In order to become a truly distributed system, every P2P system must support and manage in an integrated and transparent way all four basic types of resources on the Internet, namely processes, memory, files, and I/O. This is to say that if a P2P system could connect users to all kinds of resources and could manage all of them in a fully transparent manner in a large scale environment like the Internet, we can be optimistic to become nearer to a real implementation of distributed systems. This cannot be achieved unless the management of any P2P system can establish cooperation between resources to dynamically support different distributed usages such as distributed computing and distributed file sharing.

As it was mentioned before, two approaches to resource sharing in large scale environments had been practiced, namely, pure and hybrid [2]. Pure systems are closer to the distributed system model while the hybrid ones are close to decentralized and centralized system models. We need a criterion to benchmark the closeness of any implementation of distributed systems to real distribution. We define a ratio called *dis-*

tribution ratio for this purpose for P2P implementations like Kazaa, Napster, Gnutela, and also our own proposed framework.

Definition 1 Distribution ratio represents the closeness and farness of any implementation to distribution. Distribution ratio of 1 represents full distribution, distribution ratio of 0 represents full centralization, and any value in this range represents the closeness and farness of any implementation to distribution. Distribution ratio is calculated as follows:

Distribution Ratio $= 1 - \{X\}$
where X is calculated according to the following algorithm:
$\{X = 0;$
IF there exists a server in the P2P system then

$$X = X + \left(\left(\frac{\textit{the number of servers in P2P system}}{\textit{the number of machines in P2P system}} \right) * Y_1 \right).$$

If there exists a coordinator in the P2P system then

$$X = X + \left(\left(\frac{\textit{the number of Coordinators in P2P system}}{\textit{the number of machines in P2P system}} \right) * Y_2 \right).$$

For each type of distribution transparency that exists in the P2P system,
$X = X + Z_n;$
$X = X + Y_3;$
$\}$
where Y_1, Y_2, Z_n and Y_3 in the above algorithm have the following connotations:

- Y_1 represents *the importance factor of server* machines in the P2P system. Its value is between 0 and 1.
- Y_2 represents *the importance factor of coordinator* machines in the P2P system. Its value is between 0 and 1.
- Z_n represents *the importance factor of the transparency type* to distributed systems. Its value is between 0 and 1.
- Y_3 represents the *constrained configuration factor* of the P2P system. Its value is between 0 and 1.

Note: attending to the above parameters have concept and have deal with each other, it s not possible that all of them have value of 1.

The above algorithm for calculating the distribution ratio estimates the amount of closeness to distributed systems' concepts in developing P2P systems. But it should be pointed out that the implementations of some concepts of distributed systems are in contradiction with the implementations of other concepts of distributed systems. For example, the implementation of a fully scalable P2P system limits the full support for all types of distribution transparencies on grounds of efficiency. That is why we have included limiting factors like scalability, reconfigurability, structural dynamicity, and importance percentage of transparency, in the above algorithm for management of resources in P2P systems. These limitations, in fact, correspond to the three notable

challenges in current P2P systems, namely *flexible resource management*, *interoperability of different P2P systems*, and *integrated management of different kinds of resources*.

Flexibility is driven by reconfigurability because reconfigurable systems are more amenable to unpredictable dynamic changes that are inherent in large scale environments like the Internet on which P2P systems operate. We show that the amount of Y_3 in our framework has the least effect on the P2P system, implying that the system is totally configurable.

The resource management algorithms in most existing P2P systems are run on either server or coordinator machines. Y_1 and Y_2 parameters expose the structure and the distribution degree of management algorithms in the P2P systems. The framework does not use server machines and Y_1 parameter is in fact omitted all together. Instead, dynamic coordinators are used in a way that no assumption is considered a priori for their existence in order to reduce the influence of Y_2 factor on the distribution rate.

4 Resource management

Given our different approach toward P2P systems described in previous sections, let us state our envisaged definition of framework as follows.

Definition 2 "A framework is an extensible structure for describing a set of concepts, methods, technologies, and cultural changes necessary for a complete product design and manufacturing process" [32].

In the light of above definition for framework, the proposed management framework presents new concepts and methods to manage resources as they are discussed in the following.

4.1 An integrated resource management

Resource management algorithms in current P2P systems are mostly designed based on and biased toward the very nature and specification of each resource they want to share, for example, low frequency of changes in case of music resource. In other words, their management structures and algorithms are designed for special purposes and use static structures thereof. This is one of the reasons why the interoperability of different types of P2P systems is made impractical. In the proposed framework, we try to design algorithms based on general resource categories.

Management in our framework is based on two operations: Request and Response. Request operation is considered as a process in a machine in a P2P system that represents a request for a resource that is not accessible in this machine locally; each request is first processed by the local operating system and in case the resource is unavailable locally, the request is processed globally in the P2P system using the Request operation.

We know that when a process in a local machine requests a resource, the local operating system on the machine responds to this request. Such requests by processes

are not specialized to the kind of resources. Learning and believing in this best practice, and contrary to existing P2P systems whose resource management are specialized on specifications of particular resources, we use the pattern of resource categories in the same way it is done in operating systems. An operating system views each computer as a machine containing four basic groups of resources: I/O, file, memory, and process. Each group has a specific role that is different from the roles of others based on the operating system's view point. Each member of a group has a general specification that is shared with other members in that group and it is this very feature that puts these as members of the same group. For example, all members of I/O group have the general specification that all of them are data input/output devices. So, we can design algorithms and protocols with this general property for managing the resources in this group. We show how our framework uses this type of categorization to support all types of resources available on the Internet.

The management of resources in the proposed framework for P2P systems uses the pattern of requests for four major groups of resources similar to operating system management patterns [33] where each existing resource belongs to one of these groups based on its role in the P2P system. The conditions for being a member in a group constitute the specification of that group and the algorithms for the management of each group of resources are designed based on these specifications. As previously stated, resources fall into one of the following four categories: *process resource, I/O resource, storage resource, and memory resource*. The nature of these four groups differs from one another. It is necessary for P2P systems to have an integrated management for all these four groups, as in operating systems. Utilizing this pattern has two important advantages:

1. Resource management pattern in P2P systems does not require any new additional concepts than those traditionally available in operating systems for the purpose of local resource management.
2. Most P2P systems run on personal computers whose resources are managed by their local operating systems based on four categories of resources. Generalization of this pattern of resource management and its application to a resource management pattern in P2P systems avoids disparities between P2P systems and their underlying platforms, resulting in a more unified management of resources on the whole, as well as removing the differences between the P2P systems themselves making them interoperable.

We map our resource management pattern to the pattern used in the set algebra and use membership pattern in the set algebra to have an efficient categorization of resources. The algebra of sets has a rich set of operations (e.g., union, intersection, and minus) that we can use to change our categorization to form proper subsets. Based on the set algebra, resources are categorized in different sets with a membership condition. The membership condition is very important to have accurate partitioning and then define appropriate algorithms for each set. Therefore, in the proposed framework, the request operation is managed based on above. The response operation in this framework defined as a process is ready to response to our request. Response operation is completely depended on our framework structure.

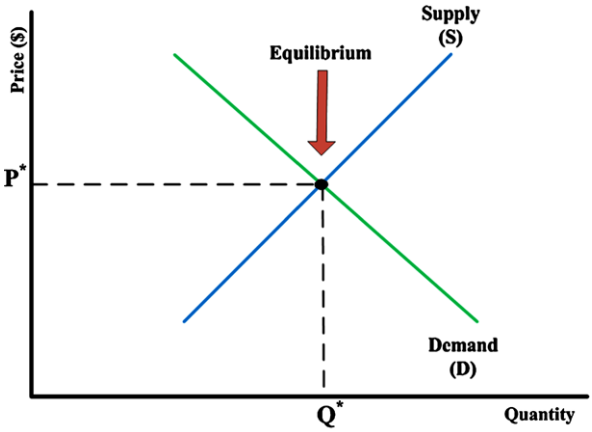

Fig. 1 Equilibrium point in supply and demand functions

5 System structure

When a process requests a resource from a local machine, the local operating system uses proper mechanisms to respond to the request. But the response operation in P2P systems is more complex and different researches have been carried out to present frameworks to handle requests effectively.

In traditional resource sharing systems, computer networks, and most of the P2P systems allowing sharing of files and processes, there are one or more machines responsible for managing available resources. When a process requests a resource, the request is processed by these machines. In case of positive response, the resource is made available to the process. These systems often work in a client/server model.

In our framework, there is no dedicated machine to process requests. To begin with, there is no difference between a P2P system and a computer networked system. Upon gradual arrival of requests by processes running on the P2P system, *regions* are formed. Regions are groups of machines that are configured to share a particular type of resource that is more available in this group than in other groups. Control over resources is not managed by servers in our framework. Instead, the pattern of requests for a given type of resource and the response capabilities of machines in the P2P system determine the machine or a group of machines that are configured to manage that particular type of resource.

5.1 Regions

Given our definitions for framework, distribution ratio and coefficient factor, we now briefly describe how a P2P system is dynamically formed. Our framework contains dynamic units named *regions*. Regions are formed based on the patterns of requests and responses (R.R.) in order to transparently and efficiently manage all four types of P2P system resources. R.R. operations behave similar to supply and demand functions in economic science. When supply and demand are equal, i.e., when the supply function and demand function intersect (as it is shown in Fig. 1), the economy is said to be at equilibrium. At this point, the allocation of goods is most efficient because the amount of supplied goods is exactly the same as the amount of goods being

demanded. Thus, everyone (individuals, firms, or countries) is satisfied with this economic condition. At the given price, suppliers sell their entire produced goods and consumers buy all their required goods [34].

Resource sharing by processes is an operation synonymous to supply, and request for resources by processes is synonymous to demand. Therefore, R.R. functions have a balance point at which they reach a stable state. Each region in our framework is formed by the pattern of R.R operations specializing in a particular functionality, and gradually reaches to a stable state as time passes and more R.R. operations are processed. For example, when a region providing computational power reaches to a stable state, it means that the region is the most efficient region in the P2P system to provide its functionality to all machines in the P2P system demanding this functionality.

Definition 3 Regions are the units of system operation in our framework. A region consists of a number of local or geographically dispersed machines that are logically grouped together to manage one type of resource (I/O, file, memory, or process).

A P2P system is dynamically structured by formation of regions. One machine in each region plays the role of group leader or *coordinator*. The ability of each machine to respond to resource requests determines to which group or domain a machine in the P2P system belongs. When a process requests access to a resource and the local machine cannot provide the resource, but there is another machine in the system that can provide the requested resource, a region is dynamically formed in the system. The formation of regions and changes in their existence and the number of machines in each region at each instant depends on the pattern of resource requests and also the capability of the group to service resource requests. Due to high frequency of changes in the number of requests, regions should dynamically adapt themselves to these changes. For example, if the number of requests in one region is abundant, the region may be broken into smaller regions, or if there is no request for a resource in one region, the region may well vanish.

Because of dynamicity of regions in our framework, P2P systems in our framework have a low *constrained structure percentage factor* (i.e., the Y_3 parameter introduced in Sect. 3). In other words, a P2P system in this framework has no priori structure when it starts operating. The system is structured based on the pattern of incoming requests and the capabilities of machines to respond to these requests. The system can reconfigure itself as the need arises by using available resources in capable machines.

Because each region is somehow structured to service a particular type of requests for resources, it can well be specialized with necessary governances. For example, a region whose machines are formed to respond efficiently to requests for processes, interprocess communication can be considered as a high performance clustered set of computers governed by high performance computing rules and policies.

5.2 Region formation algorithm

The region formation in our framework is based on the pattern of requests and responses, as it is illustrated through a scenario 1 in Fig. 2. Figure 2A shows a P2P

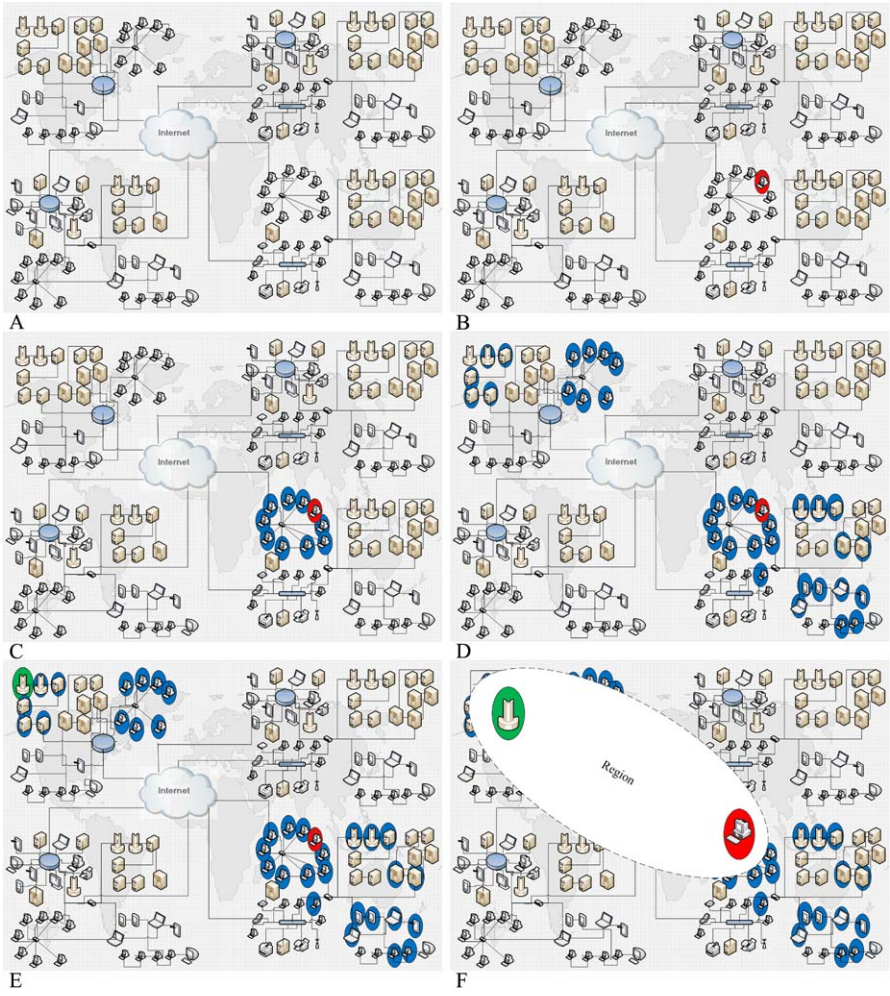

Fig. 2 Region formation process (scenario 1)

system that is quite similar to a network system with no priori structure; we call this state a *Mosaic state* in our framework. In two situations, the P2P system manager starts to work and the P2P system enters a new state we call it the *Jurassic state*. When there is a request for a resource that cannot be satisfied locally, we say a *Bang* has occurred in the system; Fig. 2B shows this case. Local operations are preferred than global operations. The second case is when the P2P manager sends a request message to other machines; the machines that the P2P manager is not active on them receive the request and active it and enter to Jurassic state. The P2P manager in a local machine has a unit called *Oasis* that has the important role of determining the location of resources in the P2P system.

We use the Oasis concept in the proposed resource management framework to keep the history of resource categorization patterns, using the operations running on

the machines. A part of memory space of each machine in a P2P system is reserved for saving the histories of four categories of resources. There are four Oasis spaces in general:

- File Management Oasis.
- Memory Management Oasis.
- I/O Management Oasis.
- Interprocess Communication Oasis.

The Oasis space contains meta-data on resources of the P2P system and it is protected by the P2P manager from access by other programs running on the same machine. The history of access, requests, responding machines, and routines for responding to requests on each of the major resources are kept inside this protected space. In other words, each machine has an Oasis space for each category of four resources. The Oasis space is initially formed in the memory of every local machine when the local machine cannot satisfy a local resource request.

The interesting point about Oasis space is that each machine has up-to-date historical information on the types of resources it had provided and the types of resource services it had from other machines. The study of the content of Oasis space identifies on which four major categories of resources a machine had been active most and on which resources is dependent most. This cached information can be used by a machine to decide if it can satisfy a local resource request by itself, or if not from which machine it can most probably get the service for that resource. The study of Oasis spaces determines the stability or otherwise of all machines in a P2P system. In other words, the P2P manager can use the content of Oasis spaces that are stored across machines locally in the peer community to determine the location of desired resources in a completely distributed manner.

The P2P manager uses the Oasis space related to the kind of a requested resource to obtain information on the machine having the resource most available than other machines. At system startup (i.e. $t = 0$), the Oasis spaces are empty. When a resource request cannot be satisfied locally in a machine, the P2P manager running on the machine sends a control message to its immediate neighboring machines (as it is shown in Fig. 2C). When neighboring machines receive this control message, their P2P managers are activated and check if they have the requested resources. If any machines in the neighborhood are not capable to provide the requested resources, they pass the request message to their own immediate neighboring machines as it is shown in Fig. 2D. The propagation of request message continues until at least one machine has the requested resources (Fig. 2E). At this time, the machine requesting the resource in the first place forms a region with one of the machines that responded to its request (Fig. 2F); the choice of a machine among more than one respondent machine is FIFO. Information on respondent machines and kind of their available resources is registered in the Oasis spaces of machines requesting the resources. Each region that is formed has a coordinator that is elected by machines in the region.

Let us now see how coordinators in regions are elected. Having formed a new group with a remote machine that provides the requested resource, one of the two machines (requester and the chosen respondent) is nominated and declared as the coordinator of the group or region. Selection is done through an election that is biased

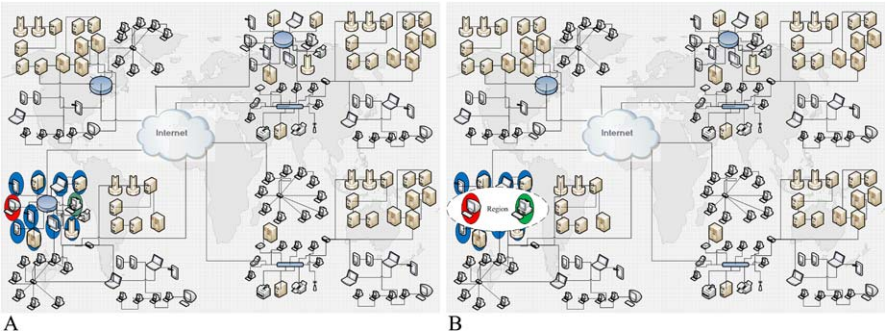

Fig. 3 Regions formation process (scenario 2)

towards the type of requested resource, the capability of machines and the frequency of changes in the requests. The machine with a higher CPU power is nominated if process resource is requested, and a machine with higher memory capacity or bus frequency is nominated if memory or secondary storage resource is requested. Upon arrival of new requests as it is shown in Fig. 3, new machines may join existing groups based on the *BarbaPapa Rule* (described later in Sect. 5.3) and each group may reorganize and start electing a new coordinator.

When a request for resource cannot be serviced by any machine in a P2P system, the request is abandoned on grounds of unavailability of resources and retried later when resources are freed.

A P2P system should reach to a stable state that we call it *Kertaseh State* in our framework. In this state of system, every region and machine reaches its stable state. In this state, regions apply special policies for their governance. As new requests for resources are raised, the regions may need to reorganize to service these new requests. In this period, the P2P system is in a transitory state we call the *Chaos State*. Any P2P system must be in a Kertaseh state before changing its state to Chaos state.

The following pseudo code shows more details of the management process of our framework:

```
If Local Process Requests a Global Resource Then
    If System State NOT Jurassic Then
        System State := Jurassic
        Activate Internal Daemons
          Create Requirement Table
          Load Control Mechanism
        Load Oasis Manager
        Create Oasis Table
        Start Timer Mechanism
            Do Until Found Resource or Time Over
        Send Request Message to Neighbors
    Else
      Load Oasis Manager
        If Oasis Table Search to Select Machines to Respond to the Request is
```

```
        Successful Then
          Send the Request Message to Them
      Else
          Do Until Found Resource or Time Over
          Send Request Message to Neighbors
   If any Response Exists then
          Call Region Creation Mechanism (based on Election Algorithms and
          Barbapapa Rule)
              Call Oasis Update Mechanism
   Else
       Abandon Request on grounds of unavailability of resources and Retry Later (If
       possible)
If a Request Message is Received Then
    If System State NOT Jurassic Then
            System State := Jurassic
            Activate Internal Daemons
              Create Requirement Table
              Load Control Mechanism
            Load Oasis Manager
            Create Oasis Table
   If Checking Mechanism Finds the Request Resource Successfully Then
          Call Responding Mechanism
   Else
       Load Oasis Manager
         If Oasis Table Search to Select Machines to Respond to the Request is
         Successful Then
             Send the Request Message to Them
       Else
            Do Until Found Resource or Time Over
            Send Request Message to Neighbors
   If Stability Status Checking Mechanism is True Then /// periodically is checked at
   system
     System State := Kertaseh
   Else
       System State := Chaos
End
```

5.3 BarbaPapa rule

To control the size of regions in our framework, we use a rule called the *Barba-Papa* rule. As we noted before, a major problem that arises in a P2P system that is structured based on our framework is when the number of machines in the system increases exponentially high. In such cases, the system ends up with four very big clouds, each of which represents the management area for one of the four resource categories. Over a long period of time, these four clouds to a great extent may overlap one another and only differ in their coordinators. To prevent the occurrence of such a

state, the size of regions need to be restricted to manageable sizes so that more than smaller sized regions replace four big regions.

The rule for enforcing size limitation in the framework is called the *BarbaPapas Collision Rule*. If a local request for a resource reaches a local coordinator and the coordinator cannot provide the resource by the machines in its region, the coordinator passes the request to its neighboring machines outside its region. One of the neighboring machines that have responded positively to the request is selected and continues to service the request. If neighboring machines are not able to satisfy the request, they propagate the initial request to their own neighboring machines likewise until either a machine ready to service the request is finally found or no machine is found at all. In the latter case, the request is passed to the coordinator that had been assigned statically to the requested resource. So finally, the machine that services the request is either a machine from another region that has voluntarily accepted to service the request or a coordinator related to that resource. In the first case, the machine that provides the service becomes a member of the region to which the requester belongs. But in the second case, the coordinator is placed exactly in the situation that directly starts to send the request to a related coordinator of that resource. In both of these two situations, the first coordinator will contact a machine that gives service, which is the coordinator itself.

Given the above descriptions, one question is whether these two coordinators can be combined into one? In other words, can one of them become the regional coordinator to coordinate requests of members of both regions? In fact, the coordinators that are connected can decide to combine based on their capabilities to respond to the requests, the number of requests they have responded to up to now, the number of their own requests responded by the other party, the size of their regions, the kind of resource they are managing, and their relevant *Flux*. But what is Flux?

Definition 4 $Flux_i$ is a parameter for studying the possibility of transferring the management of a resource i from one machine to another. If a requested resource has a *Flux* equal to 1 or bigger than 1, its control can be passed to another machine, otherwise its control cannot be passed to other machines. $Flux_i$ is calculated as follows:

IF resource i *is portable THEN*

$$Flux_i = \frac{\textit{the number of local requests for resource } i + \textit{ the number of region requests for resource } i}{\textit{total number of processes in the local machine}}$$

ELSE

$$Flux_i = 0$$

$Flux_i$ allows transfer of portable resources from their current locality only if the number of local requests for these resources does not exceed the number of requests for the same resources from other machines. For example, if there are 10 requests from 20 local processes in machine A for resource X and there are 2 external requests from remote processes, the $Flux_i$ is equal to 0.6, inhibiting the transfer of management of the resource X to other machines. Note that only portable resources can be transferred.

5.4 Region stability

To allow an unlimited number of machines to enter a given region dynamically is troublesome and inefficient because the grouped machines become saturated and spend most of their time on intra resource management instead of providing resource services to requests. We have thus proposed a rule for controlling the number of machines in regions; the rule is called *BarbaPapa*. We repeatedly simulated regions in different situations using this rule and found the optimal number of machines in each region to be from 10 to 10^2. This is a suitable threshold for the coordinator of a region to manage the resources in its region without getting too much involved in useless intra resource management.

A region that manages one of the four categories of resources (Φ) reaches a stable state when the density of related resource management operations in the region tends to 1. To find the status of region, the following variable is defined:

Definition 5 *Operation density variable* is defined in the structure of resource management in P2P systems in each region. It indicates whether the region is in an stable state or it is moving toward an stable state. It also shows for which type of resource the region had been formed.

Operation density in region $\beta =$

$$\frac{\textit{the percentage usage of a given } \Phi \textit{ operation in a seconds in region } \beta}{\textit{total usage time of all } \Phi \textit{ operations in a seconds in region } \beta}$$

where $\Phi = \{$*I/O, File, Memory, and IPC*$\}$
IF operation density in region $\beta = 1$ *THEN*
region β *becomes a repository for* Φ *operation*;

The coordinator in each region can determine the state of its region by calculating the operation density variable periodically. During formation of a region, a number of machines are busy creating the region and other machines in the P2P system are completely unaware of the region formation. The regions are completely dynamic and change as time passes, so other machines can use this variable to find out the type of resource a region is servicing.

If the operation density in a given region reaches 1, the region coordinator can change its state to *Repository State* for this resource. In repository state, the coordinator eliminates its current operations in that region and only performs operations to serve a particular type of Φ resource. At such stage, the region will adapt to perform a special operation and set some other special policies for the region based on the features of the resource that is more available in the region. Therefore, regions are formed in P2P system based on their functionality.

5.5 Machine stability status

The management of resources in P2P systems in our framework is structured in a way that always a machine in a region whose coordinator has the most ability to respond

to requests in that machine is selected. This way every machine and its enclosing region become specialized in servicing a particular type of resource, reaching closer to a stable state.

After some time called the *Mosaic time* in the framework, the Oasis of each resource in every machine is analyzed to find which machines had been most responsive to which requests of processes residing on this machine. This analysis may lead to the following cases:

1. There is no convergence on the number of machines in the Oasis of a group of machines. This means that none of the members in this group is inclined toward another member. If the group has n members, $(n/2) + 1$ members of the group must point to a special machine. Otherwise, the machine has not reached to a stable state in responding to requests which need resource and during its life time changes its membership position in different regions to reach stability.
2. In a special situation inside the members of an Oasis related to a resource Φ, either there is no element (information of machines can respond the request of resource Φ) or the number of elements according to the lifetime of machines is very few but there are more requests for resource Φ. In this case, we say the machine has autonomy in performing operations related to resource Φ. Its mean that in most of the time this machine can respond to local requests about resource Φ itself. Therefore, it can be one of the members of region or even this machine is a good choose in coordinator electing algorithm to be the coordinator of the regions that is formed for resource Φ.
3. Inside the number of machines in a group that creates Oasis there is a convergence. In fact, in most of the times a machine has responded to the requests of the machine about resource Φ, so the machine can be considered as a coordinator machine in the region in which Φ resource exists. This way we say the local machine under study is getting stable on resource Φ and by calculating the following formula we can have the responding density of machines to requests, related to resource Φ and according to that we can figure out the amount of system's stability on responding to requests related to resource Φ.

$$\Phi\ Oasis_{Density} = \frac{\textit{the number of } main_{index} \textit{ repeated in local machine}}{\textit{the number of operations that are written in Oasis}} * \frac{1}{\textit{the number of machines in region} - 1}$$

IF $\Phi\ Oasis_{Density}$ *tends to* 0 *THEN*
the local machine tends toward a stable state

Reaching stability in the structure of resource management of the system under study is very important. When a machine becomes stable, it clearly knows to which requests it can respond locally by itself and to which requests must respond globally by delegating them to other known machines that are responsive to such requests. It should be noted that stability depends on parameters that vary at run time. The higher

are the changes in configurable parameters in a P2P system, the longer takes the system to reach stability.

5.6 Discussion

We pursued two main objectives in the proposition of a new management structure for P2P systems: (1) to manage all four categories of resources in an integrated way, and (2) to introduce a flexible and dynamic structure that is also reconfigurable. The first objective was realized by following the pattern of management in operating systems wherein some requests in a machine are processed by the local operating system and the rest that cannot be processed locally are delegated to other operating systems of machines in the region to which this machine belongs to. The second objective was realized by the introduction of the distribution ratio for P2P systems that can determine how far these systems are close to real distributed systems. Parameters affecting the distribution ratio are as follows:

1. Servers.
2. Coordinator(s).
3. Configuration.
4. Transparency.

Given that our proposed resource management for P2P systems (1) does not use any central servers, (2) coordinators are not fixed but machines can change their roles with changes in the pattern of requests for resources in the whole system, (3) the configuration of system is not fixed and it is reconfigurable according to the patterns of requests for resources and responses to requests, and (4) processes are fully transparent in the sense that do not need to know the whereabouts of their requested resources, our proposed resource management has an acceptable distribution ratio.

6 Conclusion

Given the shortcomings of existing P2P systems where each one is specialized to share only a specific type of resource in the system, users had been forced to deploy different P2P systems to be able to share other types of resources as well, knowing that these systems do not interoperate. To get round this critical problem, we presented a general purpose P2P framework for integrated management of all types of resources in large scale environments, based on local resource management patterns that are traditionally used by operating systems for managing all types of resources. In other words, our proposed resource manager uses categorized resource patterns that operating systems use inside a machine for managing the resources of the machine. Resources of P2P systems were categorized into four major groups, namely memory, i/o, process, and storage, and machines in a P2P system were dynamically structured into separate regions specialized in providing a particular type of resource. Contrary to existing P2P systems that store meta-data about resources on statically known server machines, such meta-data in our framework is stored across machines locally in the peer community to determine the location of desired resources in a

completely distributed manner. Each machine in a P2P system can freely change its region based on the frequency and type of resources it services, leading to a dynamically reconfigurable and flexible system. A rule enforces size limits on each region to prevent the situation where a P2P system becomes a large system containing numerous numbers of machines located in only four system wide regions. Issues like reliability, scalability, deadlock, starvation, and timing complications that had not been considered in the proposed framework will be pursued in our future works.

References

1. Bengt C, Rune G (2001) The rise and fall of Napster—an evolutionary approach. In: The 6th international computer science conference on active media technology, Hong Kong, China, December 18–20, 2001
2. Milojicic DS, Kalogeraki V, Lukose R, Nagaraja K, Pruyne J, Richard B, Rollins S, Xu Z (2002) Peer-to-peer computing. Technical Report HPL-2002 P2Pwg
3. P2P Working Group (2008) P2P Computing Home Page. http://www.peer-to-peerwg.org/, 2008
4. Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) SETI@home: an experiment in public-resource computing. Commun ACM 45(11):56–61
5. Information Technology Department of Harvard University (2007) Home Page. http://hms.harvard.edu/hmsit/pg.asp?pn=security_glossary, 2007
6. Oram A (2001) P2P—Harnessing the power of disruptive technologies. O' Reilly & Assoc
7. Eugster P, Leifer J (2003) Peer-to-peer implementation and theory deliverable. Second progress report on formal models, Project number: IST-2001-33234, Deliverable No: D1.8, Responsible Partner: UCAM
8. Blanco R, Ahmed N, Hadaller D, Alex Sung LG, Li H, Soliman MA (2006) A survey of data management in peer-to-peer systems. University of Waterloo, Technical Report CS-2006-18
9. Tanenbaum A (2005) Distributed operating systems. Prentice Hall, New York
10. Ripeanu M, Foster I, Iamnitchi A (2002) Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Comput J 6(1):50–57 (Special issue on peer-to-peer networking)
11. Mauthe A, Hutchison D (2003) P2P computing: systems, concepts and characteristics. Praxis in der Information sverarbeitung & Kommunikation (PIK), 26(03/03). K.G. Sauer Verlag, Special Issue on P2P
12. Avaki Corporation (2002) AVAKI grid software: concepts and architecture. http://pompone.cs.ucsb.edu/~wenye/majorexam/Architecture/avaki.pdf, March 2002
13. Chien A, Calder B, Elbert S, Bhatia K (2003) Entropia: architecture and performance of an enterprise desktop grid system. J Parallel Distrib Comput 63(5):597–610
14. Clarke I, Sandberg O, Wiley B, Hong TW (2000) Freenet: a distributed anonymous information storage and retrieval system. In: Workshop on design issues in anonymity and unobservability, Berkeley, CA, USA, 2000, pp 46–66
15. Waldman M, Rubin A, Cranor L (2000) Publius: a robust, tamper-evident, censorship-resistant web publishing system. In: Proceedings of the USENIX security symposium, Denver, Colorado, USA, 2000
16. Dingledine R, Freedman M, Rubin A (2001) Free haven. In: Oram, A (ed) Peer-to-peer, harnessing the power of disruptive technologies, pp 159–187
17. Pourebrahimi B, Bertels KLM, Vassiliadis S (2005) A survey of P2P networks. In: 16th Annual workshop on circuits, systems and signal processing, ProRisc 2005, Veldhoven, The Netherlands, 2005
18. Bolcer G (2000) Magi: architecture for mobile and disconnected workflow. IEEE Internet Comput 4(3):46–54
19. Stanhope P (2002) Get in the Groove: building tools and peer-to-peer solutions with the Groove platform. Wiley, New York
20. Strom D (2001) Businesses embrace instant messaging. http://enterprise.cnet.com/enterprise/0-9534-7-4403317html. January 2001
21. Oaks S, Traversat B, Gong L (2002) JXTA in a Nutshell. O'Reilly Media, Inc.
22. Microsoft (2008) .NET Passport Technical Overview

23. Barkai D (2002) Peer-to-peer computing: technologies for sharing and collaborating on the net, 1st edn. Intel Press, Santa Clara
24. Friese T, Freisleben B, Rusitschka S, Southall A (2002) A framework for resource management in P2P networks. In: Proceedings of the international conference net object days 2002. LNCS, vol 2591. Erfurt, Germany. Springer, Berlin, pp 4–21
25. Hector BY, Molina G (2001) Comparing hybrid P2P systems. In: The 27th VLDB conference, Roma, Italy, 2001, pp 561–570
26. Kwok SH, Long Beach Chan KY, Cheung YM (2005) A server-mediated P2P system. ACM SIGecom Exch 5(3):38–47
27. Backx P, Wauters T, Dhoedt B, Demeester P (2002) A comparison of P2P architectures. In: Eurescom summit, Heidelberg, Germany, 2002
28. Liang J, Kumar R, Ross KW (2005) The KaZaA overlay: a measurement study. Comput Netw J (Special Issue on Overlays)
29. Singh K, Schulzrinne H (2004) P2P Internet telephony using SIP. In: New York metro area networking workshop. City University of New York, New York, NY, September 2004
30. Gnutella (2000) To the bandwidth barrier and beyond, Clip2 report. Available at http://www.clip2.com/gnutella.html, 2000
31. Lam TC, Liu JC (2003) On the evidence based P2P resource management in distributed computing systems. Technical Report, CPSC 681, Texas A&M University
32. Innomet Glossary (2008) http://www.innomet.ee/innomet/
33. Tanenbaum A, Woodhull AS (2006) Operating systems design and implementation. Prentice Hall, New York
34. Hülsmann JG (1999) Economic science and technology and neoclassicism. Q Austrian Econ 2(4):1–20

Mohsen Sharifi is Associate Professor of Software Engineering, currently chairing the Computer Engineering Department of Iran University of Science and Technology. He directs a distributed system software research group and laboratory. His main interest is in the development of distributed systems, solutions, and applications, particularly for use in various fields of science. He has developed a high performance scalable cluster solution comprising any number of ordinary PCs for use in scientific applications requiring high performance and availability. The development of a true distributed operating system is on top of his wish list. He received his B.Sc., M.Sc. and Ph.D. in Computer Science from the Victoria University of Manchester in the United Kingdom in 1982, 1986, and 1990, respectively.

Seyedeh Leili Mirtaheri is an active member of distributed systems research laboratory in the Computer Engineering Department of Iran University of Science and Technology. She received her M.Sc. in Computer Engineering (Software) from Iran University of Science and Technology in 2008 and received her bachelor's degree in Computer Engineering (Software) from Islamic Azad University, Iran, in 2005. Her research interests are in the areas of distributed and parallel systems, peer-to-peer computing, cluster computing, mathematics, and scientific computing.

Ehsan Mousavi Khaneghah is an active member of distributed systems research laboratory in the Computer Engineering Department of Iran University of Science and Technology. He received his M.Sc. in Computer Engineering (Software) from Iran University of Science and Technology in 2008 and received his bachelor's degree in Computer Engineering (Software) from Shahid Abbaspor University, Iran, Tehran, in 2005. His research interests are focused on distributed and parallel systems, peer-to-peer computing, cluster computing, mathematics, and scientific computing.