Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 21–34. http://www.scpe.org
ISSN 1895-1767

RESOURCE MANAGEMENT BASED ON GOSSIP MONITORING ALGORITHM FOR LARGE SCALE DISTRIBUTED SYSTEMS

FLORIN POP *

Abstract. The optimization of resource management in large scale distributed systems (LSDS) with the capability of self-organization is a complex process. LSDS are highly dynamic systems, with permanent changes in their configurations, as peers may join and leave the system with no restriction or control. This paper present the architecture for monitoring and resource management based on existing middleware solutions through the design of algorithms and methods inspired by natural models. The architecture is decentralized and it aims to optimize resource management in different types of distributed systems such as Grid, P2P, and Cloud. The important components considered for the architecture are: allocation of resources, task scheduling, resource discovery, monitoring resources and provide fault tolerance. As the system may have a large number of nodes, we need a scalable algorithm for monitoring process, able to guarantee a fast convergence no matter what the structure of the network is. In this context, gossip-based algorithms offer solutions for various topics in LSDS. The project aims to highlights the original obtained results in internationally scientific community. The paper presents the expecting results and discusses the performance evaluation of proposed architecture. Secondly, the paper presents a gossip-based algorithm for monitoring large-scale distributed systems and analyzes its efficiency in a simulated environment provided by OverSim.

Key words: Resource Management, Large Scale Distributed Systems, Self-organizing Systems, Optimization, Task Scheduling

AMS subject classifications. 68M14, 68M15, 68Q10

1. Introduction.

A distributed system consists of multiple independent computers that interact with each other in order to achieve a common goal and more importantly, it appears to its users like a single coherent system. Andrew S. Tanembaum and Marteen van Steen mention in [2] the four critical goals that should be met when developing a distributed system: first, it should provide an easy and efficient way to access its available resources, its users should not be aware of the fact that its resources are distributed across the network, it should be open and also, it should be scalable.

The property of scalability can have different meanings, depending on the component that is intended to scale. The possibility of adding new users and resources with minimal changes refers to size scalability, but if it is considered the locations of the system's nodes, it is necessary have to ensure geographic scalability. Finally, a system can also be scalable regarding its administration, so that independent administrative organizations can be easily managed and controlled.

Several performance problems may occur when trying to ensure distributed system scalability: users running centralized applications may overwhelm the unique server that handles their requests, centralized algorithms that overload the network with the huge number of messages sent, global synchronization issues for decentralized algorithms, unreliability of wide-area networks, conflicting policies for resource usage, management and security [2].

The resource management is a very important key in Large Scale Distributes Systems (LSDS). Distributed computing was developed in recent years in LSDS in response to challenges raised by complex problems solving and resource sharing in collaborative, dynamic environments. LSDS computing concerns large-scale interconnected systems and have the main purpose to aggregate and to efficiently exploit the power of widely distributed resources. In LSDS, load-balancing plays an essential role, in cases where one is concerned with optimized use of resources. A well balanced task distribution contributes to reducing execution time for jobs and to using resources, such as processors, efficiently, in the system. On the other hand, the problem of scheduling heterogeneous tasks onto heterogeneous resources is intractable, thus making room for good heuristic solutions. Concerning the platforms, heterogeneity refers to hardware, software, communication characteristics and protocols, network irregularities, etc.

Bio-inspired models have been used for the decentralized construction of Grid information systems with adaptive and self-organizing characteristics. In [18] A. Forestiero and collaborators present a self-organizing Grid *SO-Grid* that basically provides two functionalities: logical reorganization of resources inspired from the behavior of some species of ants which move and collect items within their environment and resource discovery based on the ants ability of searching food by following pheromone traces left by the others. Gossiping protocols

*Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: florin.pop@cs.pub.ro

are attractive bio-inspired techniques commonly used in large-scale distributed systems. Not only their robustness, flexibility and simplicity, but also their efficiency in spreading information within a group, make them very interesting for enhancing distributed systems with self-organizing capabilities. Moreover, Baker and Shostak described in [20] a gossiping protocol between ladies through telephones, protocol which was later applied for computers and networks. Even though most of the time, these algorithms are continuously processes, one of their major drawbacks is the high number of sent messages across the network. This transforms convergence into a critical property. Convergence refers to the number of rounds executed during the gossiping process in order to propagate the information within the whole group of peers. In the context of large-scale distributed systems, we must ensure that the algorithm designed for monitoring the peers maintains its scalability even when the size of the network is huge.

All the work presented in this paper is based on position paper [1]. We extend here the previous work with experimental results and experimental validation of monitoring gossiping algorithm for LSDS. The paper is structured as following. Section 2 presents an overview of Large Scale Distributes Systems and defines different metrics used for characterizing networks from a structural point of view. In Section 3 is presented the problem of Resource Management in LSDS and existing solutions. Section 4 presents Resource Management in LSDS. Monitoring process is presented in Section 5, describing a gossiping our protocol for the monitoring process of a peer-to-peer system. Section 6 describes system evaluation and experimental results. Section 7 highlights the conclusions and future work. Moreover we analyze and explain the results obtained for various test scenarios. In the end we draw the conclusions and we emphasize the impact of our research.

2. Overview of Large Scale Distributes Systems. The development of SORMSYS project (Resource Management Optimization in Self-Organizing Large Scale Distributes Systems) refers to many types of distributed systems, including Grid Computing Systems, Cloud Computing Systems, Massive Parallel Machine (MPM) and Peer-to-Peer Systems (P2P).

Grid Computing. Grid Computing represents a distributed paradigm dedicated to high performance computing that brings together resources from different organizations in order to facilitate the collaboration of a group of people. The major challenges that hide behind the concept of Grid are large-scale resource sharing coordination and the ability of developing innovative applications by using the advantages of a system with a high degree of heterogeneity. The key issue in a grid computing system refers to the notion of Virtual Organization (VO) which represents a group of people or institutions having a collaboration based on a set of rules that defines the terms and conditions of sharing direct access to computers, applications, other resources.

Ian Foster, Carl Kesselman and Steven Tuecke analyze in their work [3] the Grid problems and challenges and present an extensible architecture with respect to protocols, services, application programming interfaces (API) and software development kits [4]. Grid's architecture is composed of four layers that can be easily mapped into Internet layers (protocol architecture). Grid computing can be seen as an answer to drawbacks such as overloading, failure, and low QoS, which are inherent to centralized service provisioning in client-server systems. Such problems can occur in the context of high-performance computing, for example, when a large set of remote users accesses a supercomputer.

Centralized computing systems based on paradigms such as the client-server or the master-worker have to deal with many drawbacks including overloading of the central node, single point of failure, low performance and QoS. Domenico Talia and Paolo Trunfio describe Grid computing in [5] as a solution to all of these problems, especially in the context of high-performance applications when a large set of remote users gains access to a supercomputer.

Grid's primary goal is to ensure access to remote resources and for this purpose have been developed toolkits that provide secure services for submitting batch jobs or executing interactive applications on remote machines, but they also include mechanisms for efficiently sharing and moving data.

Resource discovery and management is an important point of interest in Grid technology. Current centralized models in which a certain node running a server application is responsible for storing and publishing information about a certain organization's node set are not capable to satisfy the requests coming from a more dynamic and large-scale distributed system. Considering the dynamic nature of a Grid environment, it is necessary to provide fault tolerance mechanisms based on decentralized P2P algorithms that do not involve critical failure points. Grid technology includes many different forms of computing:

- *Semantic and Service-Oriented Grid* - resources and services described using semantic data model.
- *Ubiquitous and Pervasive Grid* - the result of combination between mobile and wireless devices with

wired Grid infrastructure.
- *Data Grid* - the controlled sharing and management of large amounts of distributed data.
- *eScience Grid* - computing dedicated for scientific applications from biology, medicine, astronomy, physics, finance, weather forecast.
- *Enterprise Grid computing* - enterprise data centers, managed by a single business entity.
- *Autonomic Grid computing* - Grid systems with self properties.
- *Knowledge Grid* - manage knowledge resources used in VOs by interoperability among users, applications and resources.
- *Economy Grid* - development of economic or market based resource management and contributory systems.

Cloud Computing Systems. This section presents the Cloud Computing System with its architectural features and the most important characteristics that distinguish this paradigm from other computing technologies.

Lizhe Wang and Gregor von Laszewski study the anatomy of a cloud system in [6] and define it as a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way. All the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services defines the concept of cloud computing. Michael Armbrust and collaborators [7] mention three important aspects regarding cloud computing from a hardware point of view: the illusion of infinite computing resources available on demand, the elimination of an up-front commitment by Cloud users and the ability to pay for use of computing resources.

Mladen A. Vouk introduces cloud computing in [11] as the evaluated solution for on-demand information technology services and products based on virtualized resources and states the main properties of a cloud computing system: service oriented architecture, reduced information technology overhead for the end-user, great flexibility and reduced total cost of ownership [10]. As the user is the most important entity, a Cloud System has a clear hierarchy that groups users in different categories: system developers responsible with Cloud infrastructure, developers or authors of component services and underlying applications, domain personnel who integrates basic services into composite services and delivers them to end-users, users of simple and composite services. As in the case of Grid and P2P computing, a cloud system provides an integrated computing platform as a service, in a transparent way:

- *Hardware as a Service* - users can buy IT hardware or even an entire data center. Examples of flexible, scalable and manageable are represented by Amazon EC2, IBM's Blue Cloud project, Nimbus, Enomalism.
- *Software as a Service* - software applications are implemented as services and are provided to users across the Internet. In this manner, users are no longer required to buy software or to install and run the needed application on their local machines. An interesting example is Google's Chrome browser that offers a new desktop, through which applications can be delivered besides the traditional web browsing experience.
- *Data as a Service* - users connected to a certain network have share the opportunity of accessing data from multiple sources via services and manipulating in a transparent way, as it would be located on their local disk. A simple Web services interface used for storing and retrieving data is provided by Amazon Simple Storage Service. Other examples can be found at Google Docs, Adobe Buzzword, ElasticDrive.

Moreover, through cloud computing, users can subscribe to their favorite computing platforms with requirements of hardware configuration, software installation and data access demands, this feature representing the concept of Platform as a Service.

The main features for cloud computing are:

- User-centric interfaces - users are not required to learn new APIs and commands to access resources and services, like they have to do in the Grid systems case.
- *On-demand service provisioning* - users gain access to resources and services.
- *QoS guaranteed* - the computing environments can guarantee CPU speed, I/O bandwidth and memory size.
- *Autonomous system* - the system's management is transparent to its users
- *Scalability and flexibility* - computing clouds can be easily scaled regarding the geographic locations, hardware performance or software configurations, but they are also flexible to adapt to a large number

of users.

P2P Computing Systems. Peer-to-peer systems gained a significant importance in the last years due to their capability of cooperating and forming a network without the existence of a central point. They are distributed systems with no centralized control, where each node accomplishes the same functionality [8]. This decentralized nature makes them extremely robust against certain failures and also well-suited for high-performance computations or long-term storage.

In contrast with the Grid systems, P2P systems have been developed in open communities, in which security mechanisms are not required and do not involve authentication and content validation [9]. Users of P2P systems have common goals such as retrieving music from the Internet, so they only need protocols that offer anonymity.

Regarding the connectivity aspect, P2P systems prove to have another important difference from the Grids, as they are composed mainly of desktop computers connected to the network for a limited period of time and with reduced reliability [12]. Grids, on the other hand have powerful machines, statically connected through high-performance networks with high level of availability. Moreover, the number of nodes connected in a P2P network at a certain time is much greater than in a Grid which has a very restrictive access to resources due to certain accounting mechanisms.

Grid's major goal was to provide access and management to remote resources, fact which is not respected in P2P systems since they do not handle remote cycle's allocations and storage. A key element in P2P systems is the presence-management protocol that allows each node to periodically notify its presence, discovering, in the same time, its neighbors. Since in Grid environments resource discovery in is based mainly on centralized or hierarchical models, further implementations should be inspired from this P2P decentralized resource discovery model [13]. Even though fault tolerance is a very important aspect in a distributed environment, this issue remains unexplored in grid models and tools. As it has mentioned before, Globus is able to provide fault detection, but developers should implement fault tolerance at the application level. The solution comes from decentralized P2P algorithms, which do not have to handle centralized services with single points of failure.

P2P Computing appeared as the new paradigm after client-server and web-based computing. P2P systems became quite popular for file sharing among Internet users through Napster, Gnutella, FreeNet, BitTorrent and other similar systems. Besides its great impact over file sharing applications, P2P systems also have a strong contribution in high parallel computing: SETI@home and FightAIDS@home are parallel applications running on available nodes.

Differently from centralized or hierarchical models of Grid systems, in P2P systems, nodes (peers) have equivalent capabilities and responsibilities and can be both servers and clients. These systems are evolving beyond file sharing being thus the basis for the development of P2P applications.

Moreover, P2P systems have inspired the emergence and development of social networking for enabling human interaction at large scale, which are having a tremendous impact on today's information societies. Since the appearance of the P2P systems [14], new forms of such paradigm has appeared, including B2B (Business to Business), B2C (Business to Consumer), B2G (Business to Government), B2E (Business to Employee), etc.

Self-organization gains more and more importance with the fast expansion of the distributed systems. The permanent changes and unexpected events occurrences are extremely difficult to handle globally due to the very large size of the environment which can cover a huge number of different geographic locations. Moreover, traditional mechanisms for resource management and fault tolerance become inefficient, as centralized models will no longer represent viable solutions. In this context, the need of an flexible, robust, adaptive and self-organizing environment is obvious.

Coefficients used to characterize a network as a support for LSDS are important for systems self-organization. In assortative networks, well-connected nodes tend to join to other well-connected nodes, as in many social networks (for instance, Facebook). This means that an assortative network has the property that almost all nodes with the same degree are linked only between themselves. On the other hand, disassortative networks have the property that well-connected nodes join to a much larger number of less-well-connected nodes, fact which is typical of biological networks. Previous work in the field of assortativity present different methods for generating correlated networks with predefined correlations. In [22], R. Brunet and I.M. Sokolov propose a different approach based on link-restructuring, also called *rewiring process*, with either the goal of connecting nodes with similar degrees in order to obtain an assortative mixing, or the goal of connecting nodes having low degree with nodes having high degree in order to obtain a dissassortative mixing.

The clustering coefficient **C** [21] represents another basic metric for measuring the internal structure of a network. It relates to the local cohesiveness of the network and measures the probability that two verticies with

a common neighbor are connected. Considering an undirected network, a certain node N_i with k_i neighbors, there are:

$$\frac{k_i \times (k_i - 1)}{2}$$

possible edges between the neighbors.

The clustering coefficient of that node is given by the ratio of the actual number of edges E_i between neighbors and the maximal number:

$$\mathbf{C} = \frac{2 \times E_i}{k_i \times (k_i - 1)}$$

The global value of the clustering coefficient is obtained as the average cluster coefficient of all nodes. In the context of large-scale distributed systems, the last two metrics are very important, because the algorithms we design must be scalable with respect to the system's size and also they must ensure the robustness of the system, as failures may occur at any moment of time.

3. Resource Management in LSDS. Resource management must take into account additional issues such as resource consumer and owner requirements, the need to continuously adapt to changes in the availability of resources, etc. Based on this Grid characteristic, a number of challenging issues need to be addressed: maximization of system throughput and user satisfaction, the sites' autonomy (the Grid is composed of resources owned by different users, which retain control over them), scalability, and fault-tolerance.

The design process of a resource management system for a given system and implicitly of a scheduling algorithm must consider all aspects of the system and applications that will run in it. The resource management process in LSDS has the main function as scheduling. Basically, we have local and global scheduling. A *local scheduler* considers a single CPU (a single machine). *Global scheduling* is dedicated to multiple resources. Scheduling for distributed systems such as the Grid is part of the global scheduling class. For global scheduling three classes of scheduling algorithms have been designed: *centralized* (the scheduling decisions are made by one central module which runs the scheduling algorithm), *hierarchical* (based on the principle of work division) and *decentralized* (algorithms for cooperatively or independently working) [15].

Various strategies for resource management have been developed, in order to achieve optimized task scheduling in distributed systems. In the static scheduling model, every task is assigned only once to a resource. A realistic prediction of the cost of the computation can be made before the actual execution. The static model adopts a "global view" of tasks and computational costs. One of the major benefits is the easy way of implementation. On the other hand, static strategies cannot be applied in a scenario where tasks appear a-periodically, and the environment undergoes various state changes. Cost estimate do not adapt to situations in which one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failures, is so heavily loaded with jobs that its response time becomes longer than expected, or a new computing node enters the system. These changes are possible in Grids. In dynamic scheduling techniques, which have been widely explored in literature, tasks are allocated dynamically at their arrival. Dynamic scheduling is usually applied when it is difficult to estimate the cost of applications, or jobs are coming on-line dynamically (in this case, it is also called online scheduling). Dynamic task scheduling has two major components: one for system state estimation (other than cost estimation in static scheduling) and one for decision making. System state estimation involves collecting state information through Grid monitoring and constructing an estimate. Decisions are made to assign tasks to selected resources. Since the cost for an assignment is not always available, a natural way to keep the whole system healthy is by balancing the loads of all resources.

Another criterion used to classify the schedulers is the way they perform the state estimation of the system. Some of the schedulers attempt to predict the load on the resources in the future or the execution time of the jobs, others take into account only the present information. However, the available information is always partial or stale, due to the propagation delay in large distributed systems.

Some of the schedulers provide a rescheduling mechanism, which determines when the current schedule is re-examined and the job executions reordered. The rescheduling taxonomy divides this mechanism in two conceptual mechanisms: periodic/batch and event-driven on line. Periodic or batch mechanism approaches group resource request and system events which are then processed at intervals that may be periodically triggered

by certain system events. The other mechanism performs the rescheduling as soon the system receives the resource request [16].

The scheduling policy can be fixed or extensible. The fixed policies are system oriented or application oriented. The extensible policies are ad-hoc or structured. In a fixed approach, the policy implemented by the resource manager is predetermined. Extensible policy schemes allow external entities the ability to change the scheduling policy.

Genetic algorithms (GAs) have been widely used to solve difficult NP complete problems like scheduling problem. A genetic algorithm for scheduling independent tasks in Grid environment was developed. It can increase search efficiency with a limited number of iterations by improving the evolutionary process while meeting a feasible result. A fault tolerance-genetic algorithm for Grid task scheduling using check point was proposed. Another genetic algorithm based scheduler for computational grids is designed to minimize make-span, idle time of the available computational resources, turn-around time and the specified deadlines provided by users. The architecture is hierarchical and the scheduler is usable at either the lowest or the higher tiers. It can also be used in both the intra-grid of a large organization and in a research Grid consisting of large clusters, connected through a high bandwidth network.

Considering all the issues presented on the management of resources in different types of distributed systems and considering the remaining issues still unresolved, the motivation for the SORMSYS project is to design and develop a full decentralized architecture to optimize management of resources in large distributed systems. The need to optimize resources management is maintained by increasing the number of users and applications and their types. Thus, the main problem for the management of resources is to develop architecture with meta-scheduling capabilities aimed the resource dynamic comportment and heterogeneity. In terms of interdisciplinary, SORMSYS project considers self-organization distributed systems and use of nature-inspired techniques and methods for developing algorithms for resource allocation, task scheduling, resource discovery, resource monitoring and ensuring fault tolerance.

4. System Architecture. SORMSYS project reaches a scalable and flexible environment, able to detect complex events like distributed failures and to make a system converge to a desired state. A new approach for the LSDS infrastructure, represented by the *Virtual Middleware* (VM) is proposed. The VM is placed on top of the actual middleware and is responsible for the self-management and self-organization of the system.

Originality of the solution proposed by SORMSYS project is to use existing methods in various types of distributed systems (Grid, P2P, Cloud, MPM, Cluster), but also inspired by nature for the optimization process to realize the resource management architecture. The impact of the project in this context consists of the increase of resource availability while preserving scalability of large scale distributed systems, the increase of the maintainability of distributed systems using an architectural model based on a minimal set of functionalities, the extension of the methods for testing of the performances using simulation and real environment in the analysis of reliability, availability, safety and security for large scale distributed systems.

4.1. Components Analysis. The system's organization is a multi-layered one, with three major architectural components (see Figure 4.1):

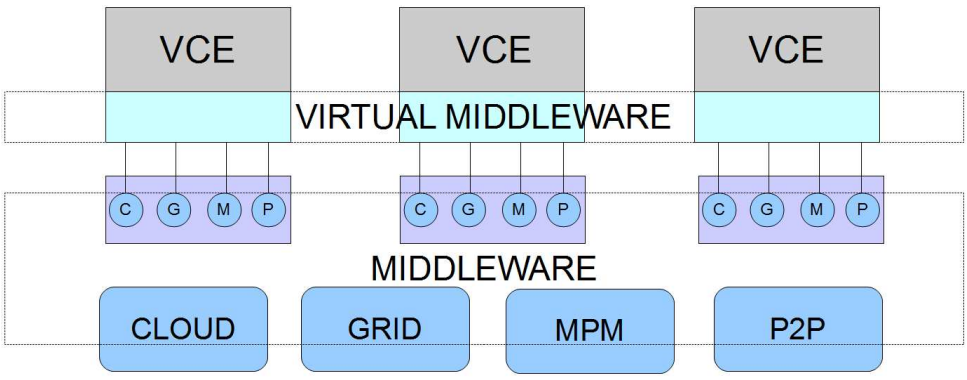

FIG. 4.1. *System Architecture.* ***C****loud,* ***G****rid,* ***P****2P,* ***M****PM*

- **Virtual Entity Manager**. The Virtual Controller Entity (VCE) is a piece of software running on each machine of the system that interacts with other similar entities in order to ensure an adaptive and self-organizing environment. Its implementation is based on bio-inspired and natural models such as ant computing and gossiping that are suitable in the context of a large-scale distributed system due to their good behavior in the presence of unexpected failures or other similar events.
- **Virtual Middleware** (VM). The VM is the layer that guarantees the collaboration and interoperability between VCEs, by providing communication channels and protocols necessary for their interactions. A complete description of this layer is offered in the next subsection.
- **Virtual Connectors**. The Virtual Connectors are responsible for providing the proper solutions for different types of distributed systems, according to their architectural structure and requirements: Grid, P2P, Massive Parallel Machine (MPM) or Cloud.

4.2. Virtual Middleware in SORMSYS. Middleware is a layer of software that connects users or applications working or running on different operating systems. It creates environments for developing systems that can be easily deployed on a large number of topologies and computing machines. This is achieved by providing necessary platforms and tools in order to monitor, validate, coordinate and manage resources, functions that actually ensure QoS for the running application and also users interoperability.

The VM of a large-scale distributed system is responsible for providing a self-organizing environment, with respect to any unexpected event that may occur. This VM can be viewed as an extension of the actual middleware, being mainly represented by the entities that control and monitor each node from the distributed system. The most important role of this layer is to ensure communication and interoperability between high level entities defined as VCEs that manage and control the whole system. Moreover, the services provided through decentralized gossip-based algorithms and other bio-inspired paradigms, prove the importance of the VM. Besides its strong connections with the agents running on top of it, the VM also provides connectors to the current available distributed systems technology: Cloud, Grid, P2P and MPM.

4.3. Resources Management in SORMSYS. A very difficult challenge in a large-scale distributed system is to realize the resources management. It is impossible to gather and maintain a detailed and up-to-date list of all nodes participating in any large computation. The solution comes from gossip-based algorithms that have an excellent behavior, proving very good convergence and resilience properties, in the present of continuous changes across the system.

Our first achievement in this research is a gossip based algorithm capable to monitor the nodes in terms of their performance, availability and reliability.

5. SORMSYS Monitoring. In the context of a large-scale distributed system, a major challenge is to discover each resource capabilities and to propagate them across the network. A resource's availability at a certain moment depends not only on its monitored parameters, but also it has to evaluate its neighbor's states. When running a distributed application, it is a critical requirement for the starting node to acknowledge the state of the resources from its neighborhood with respect to reliability and performance capabilities.

Gossip algorithms rely on simplicity, robustness and flexibility, features that make them suitable for a broad range of applications from data dissemination and aggregation [19] to overlay maintenance and resource allocation. Thus, the concept of gossiping is based on two key elements: repetition, as there is a continuous process of stochastic selection of two nodes and let them share information, and probabilistic choice which is involved in the selection process of the two nodes. The analogy with a real life behavior is obvious, as the information exchanged between two randomly chosen nodes can spread within a certain group. Moreover gossip-based information propagation can also be associated with the way a viral infection spreads in a biological population.

In the context of a large-scale distributed system, a major challenge is to be discover each node capabilities and to propagate them across the network. A node's processing availability at a certain moment depends not only on its parameters, but also it has to evaluate its neighbor's states.

When running a distributed application, it is a critical requirement for the starting node to acknowledge the state of the nodes from its neighborhood with respect to reliability and performance capabilities.

5.1. Nodes Evaluation. Each node is characterized by a set of parameters which describes its current state. The nodes are classified upon three key insights that are called the **ARP**-property:

1. **Availability** (A) - a node is considered available if it is not executing a task or if its current load is not too intensive. This property is also given as a percentage.
2. **Reliability** (R) - it is critical to have an evaluation of the nodes regarding their level of dependability, because unexpected failures must be avoided during a computation task. For this reason, the nodes are classified as following:
   - **Alive** - the node can be trusted and presents a very low risc of failure: $R = 1$ is the maximum value of this parameter indicates a great confidence in the node
   - **Dead** - the node is down and cannot be included in computation: $R = 0$ is the minimum value of this parameter and indicates a very low confidence in the node
   - **Transit** - a real value between 0 and 1 will indicate that a transit node is either overwhelmed due to some intensive computation, either in the process of recovering from a previous failure
3. **Performance** (P) - reflect the node's physical resources (CPU, memory, channels) and has three thresholds: *High*, *Medium* and *Low* given as real values between 0 and 1.

The node's evaluation is realized in respect to these three parameters by a function $f(A, R, P)$ which returns the ARP-value that is going to be associated with the node. There are several types of message that are going to be sent between system's nodes:

- **Gossip Request Message** (GRQ) - this message is sent by a node to its neighbors' with the goal of performing an evaluation of their ARP-property. This is always followed by a Gossip Reply Message sent to the initiator of the gossip.
- **Gossip Reply Message** (GRP) - this is the response that is received from neighbours with their parameters at a certain moment. At the first exchange it will contain only the records of the neighbor selected for gossiping.
- **Reliability Check Message** (RCM) - this message is sent by each node to its neighbors' in order to check if the node is still alive or not. It is useful for computing the reliability parameter from the ARP-property.

Both gossiping messages have the same layout, the only difference being a bit that indicates either a request or a reply - **GRx**. They also contain another two fields representing the peer that sent the message **SRC** and the round number registered at the moment when the message was created **RND**.

The last bytes store the gossip table **GT**, which is basically formed by the tuple $(Peer, ARP - value, Last\ Modified)$: the peer, its ARP-property and a bit that specifies if last round the recorded value has been modified. If the byte is unset, it is considered that the stored value is old and will have low priority during the process of solving conflicts between multiple GT. A proper field will also indicate the size of the transferred table.

5.2. Decentralized Monitoring Algorithm. In this paper is proposed a fully decentralized algorithm based on gossiping technique that ensures global monitoring in a distributed environment, with a large number of nodes.

The goal of this algorithm is to dynamically spread monitoring information about each node across the network, so that each node will estimate the ARP-value of the others.

Each node stores locally a *Gossip Table* that reflects its knowledge of the system in terms of availability, reliability and performance. The performance is measured according to the peer's physical resources and the availability is evaluated during the gossiping process. For reliability estimation, each node has to send a ping message at certain predefined *time samples* to each of its neighbors' and to update its current records from GT.

Peer-to-peer systems are highly dynamic systems, with permanent changes in their configurations, as peers may join and leave the system with no restriction or control. This makes monitoring an important element for several applications, esspecially for fault management: fault detection and fault recovery.

There are two major issues that occur in P2P systems, that represent important aspects for monitoring algorithm:

- first, as we consider large-scale distributed systems, the system's size, fact which influences the scalability and the efficiency of the algorithms that we design for managing the system.
  This gives the first requirement of any algorithm dedicated for such systems: it must be scalable with respect to the number of peers within the system.
- second, peer-to-peer systems are well-known for their dynamic behavior, as peers appear and disappear from the network all the time. This means, that the algorithm should be invariant to arbitrarly events.

Even though multiple nodes fail, or a considerable number of nodes join the system, the algorithm should be able to adapt itself in order to complete the same functions.

The algorithm is composed on several rounds, during which the nodes execute the following actions:

- *Step 1.* First, a certain node, randomly selects a gossiping partner, packs its knowledge existing in the gossiping table into a message labeled with GRQ and sends it to the selected peer.
- *Step 2.* For a predefined timeout, the node waits the reply from its partner. Based on the reply time, the initiator will evaluate the availability according to a certain pre-established threshold. In the mean time, the node can also receive other gossiping requests from its neighbors'.
- *Step 3.* When the timeout of current round expires, the node will have to handle the gossip tables received from its neighbors' during gossiping exchanges. It will solve the occurred conflicts, by storing in its local gossip table only the last modified records. After finalizing these updates, a new round can be performed.

The algorithm respects the scheme of a gossip algorithm, as it is based on both probabilistic choice for selecting the gossiping partner and on repetition, because several rounds are executed in order to ensure solution's convergence. This means that after a certain number of rounds, the monitoring information about each node is spread across the system and each node can evaluate the other peers in terms of availability, reliability and performance.

The number of rounds after the monitoring information from each peer have been propagated within the whole group of active represents the complexity of the algorithm.

Algorithm 5.2.1 Active Thread on Peer P_i

```
P_j := peer within the network
map := hash table < peer, load >
f := computes the load of the peer based on its local parameters

P_j := selectRandomPeer()
local = f(w_1, w_2, ..., w_n)
map.put(P_i, local)
sendGossipRequest(P_j, map)

msg = receiveGossip(Q)
if (msg.type == REPLY) then
   map.update(msg)
end if
```

Each peer maintains a monitoring table that reflects the peer's local knowledge of the system it belongs to. Initially, this table contains only the monitoring information of the peer, but after several gossiping rounds it will be populated with data from different peers of the system. Each peer of the network executes two threads. The active thread, in which the peer play the role of the initiator, as it is the one that chooses a gossiping partner and sends it a request for exchanging information (algorithm 5.2.1). The passive thread, in which the peer simply waits for incoming gossiping requests and replies with its local knowledge (Algorithm 5.2.2). After each round, if new values have been exchanged, the table is updated and based on the current information, a peer can estimate the global load of the network:

$$\textbf{load} = \frac{\sum_{P_i} map.get(P_i)}{N}$$

6. SORMSYS Evaluation and Use Cases for Experimental Tests. Considering the topology in Figure 6.5 and the initial ARP-values in Figure 6.1, the behavior and the evolution of the gossiping algorithm can be described as following:

- N_1 selects the only neighbor it has, node 2 and sends a gossip request, which will be followed by a reply that is going to add a new entry in the local table.

Algorithm 5.2.2 Background Thread on Peer P_i

```
P_j := peer within the network
map := hash table < peer, load >

msg = receiveGossip(Q)
if (msg.type = REQUEST) then
   sendGossipReply(Q, map)
   map.update(msg)
end if
```

PEER	ARP
1.	0.5
2.	0.6
3.	0.7
4.	0.3
5.	0.2
6.	0.2

FIG. 6.1. *Initial ARP Evaluation of Nodes*

- N_2 receives a request from N_1, adds a new entry according to the information it received and sends the reply message with its own data. It also has to initiate a gossip request, so let's assume that it selects N_3 to exchange information with.
- N_3 exchanges information with N_2 and sends a request message to a random neighbor: N_4. It will also receive requests from N_5 and N_6, so it will almost complete its table, except the first node.
- N_4 selects N_3, but they have already exchanged data, so no modifications appear.
- N_5 and N_6 send requests to N_3 and update their local tables with the replied data.

The algorithm converges towards a consistent solution after three rounds, when all peers will have information about each others. The complete evolution of the gossiping tables is illustrated in Figures 6.2, 6.3 and 6.4.

The peer number and its depth are colored to reflect the round they have been discovered by a certain node.

The overlay of the network in OverSim is randomly generated: each node that joins the system selects a random bootstrap node to connect with. The unique key associated with each node is determined based on the IP addresses as follows:

$$\mathbf{key} = IP\& \ (1 << 24)$$

After joining the overlay, each node sets up a timer that for scheduling a gossiping round. The gossips sent between peers are UDP messages of two types: REQUESTS and REPLIES.

After the timer expires, each node packs its current local knowledge and sends it through a gossip request message to a randomly selected peer within the system.

When receiving a gossip request, a node replies with its local knowledge and also updates its monitoring table with the information received. When receiving a gossip reply, a node will simply update its local knowledge.

For simulating multiple failures, we used a random churn model supported by OverSim.

Several tests were executed, with different network sizes and for each node was recorded the number of gossiping executed to reach a state where its local knowledge reflects the monitoring information of the whole network. Beside this, was encountered the total number of gossip rounds initiated by all peers and the average number of rounds after the algorithm converges to the desired state.

The number of gossips initiated by a node is represented by the number of rounds executed until its local monitoring table is completed.

As some of the peers reach the final state faster than other, we computed an average number of rounds using the weighted average.

Peer	Depth	Round 1	Round 2	Round 3
2.	1	0.6	0.6	0.6
3.	2	-	0.7	0.7
4.	2	-	-	0.3
5.	3	-	-	0.2
6.	3	-	-	0.2

Peer	Depth	Round 1	Round 2	Round 3
1.	1	0.5	0.5	0.5
3.	1	0.7	0.7	0.7
4.	1	-	0.3	0.3
5.	2	-	0.2	0.2
6.	2	-	0.2	0.2

FIG. 6.2. *Rounds evolution for nodes 1 and 2*

Peer	Depth	Round 1	Round 2	Round 3
1.	2	-	0.5	0.5
2.	1	0.6	0.6	0.6
4.	1	0.3	0.3	0.3
5.	1	0.2	0.2	0.2
6.	1	0.2	0.2	0.2

Peer	Depth	Round 1	Round 2	Round 3
1.	2	-	0.5	0.5
2.	1	-	0.6	0.6
3.	1	0.7	0.7	0.7
5.	2	-	-	0.2
6.	2	-	-	0.2

FIG. 6.3. *Rounds evolution for nodes 3 and 4*

Considering **N** the size of the network, **x** the dimensions of each group of nodes that complete the monitoring process in the same amount of time and **w** the number of rounds associated with each group of nodes.

$$\textbf{rounds} = \frac{\sum_{k}^{i=0} w_i \times x_i}{N}$$

For a network of 100 nodes, 3 of them reach the final state in 5 rounds, 55 in 6 rounds, 39 in 7 rounds and 3 in 8 rounds. The total number of gossips is 642 and the average number of rounds executed to ensure algorithm's convergence is 6.42.

In the case of 200 nodes, 6 rounds are required for 14 nodes to receive monitoring data from the whole system, 7 rounds for 141 nodes and 8 rounds for the rest of 45 nodes. The total number of gossips is 1431 and the average number rounds is 7.15.

With a network of 300 nodes, only 1 node completes the gossiping process in 6 rounds, 118 nodes in 7 rounds, 165 in 8 rounds and 16 nodes in 9 rounds. In this situation, he total number of gossips is 2296 and the average number rounds is 7.65.

For a network of 400 nodes, 130 of them reach the final state in 7 rounds, 250 in 8 rounds and 20 in 9 rounds. The total number of gossips is 3090 and the average number rounds is 7.72.

Considering a network of 500 nodes, 96 nodes received the complete set of monitoring data in 7 rounds, other 325 nodes in 8 rounds and the rest of 78 in 9 rounds. The total number of gossips is 3974 and the average number rounds is 7.95.

Finally, for a group of 1000 nodes, 7 nodes converged in 7 rounds, 485 nodes in 8 rounds, 488 nodes in 9 rounds and 20 nodes in 10 rounds. The total number of gossips is 8521 and the average number rounds is 8.52.

Figure 6.6 illustrates the evolution of algorithm's convergence for nodes belonging to networks of various sizes, while Figure 6.7 presents the average convergence rounds of the algorithm.

6.1. Local ARP Estimation. When selecting the node to become gossiping partner, the arbitrary selection will be eliminated and a more accurate approach is provided. For this, each node will associate to its neighbors a confidence parameter expressing the capacity of each subset of nodes dominated by each neighbor. The network can be viewed as a graph, so the previous statement is equivalent to: a node's confidence is calculated based on the capacity of each sub-graph determined by its children.

This value is important for scheduling a distributed application, because it estimates the capability of a certain set of nodes to execute a pool of tasks. In this context, the pre-defined *TIMEOUT* is now used to limit

Peer	Depth	Round 1	Round 2	Round 3
1.	3	-	-	0.5
2.	2	-	0.6	0.6
3.	1	0.7	0.7	0.7
4.	2	-	0.3	0.3
5.	2	-	0.2	0.2

Peer	Depth	Round 1	Round 2	Round 3
1.	3	-	-	0.5
2.	2	-	0.6	0.6
3.	1	0.7	0.7	0.7
4.	2	-	0.3	0.3
5.	2	-	0.2	0.2

FIG. 6.4. *Rounds evolution for nodes 5 and 6*

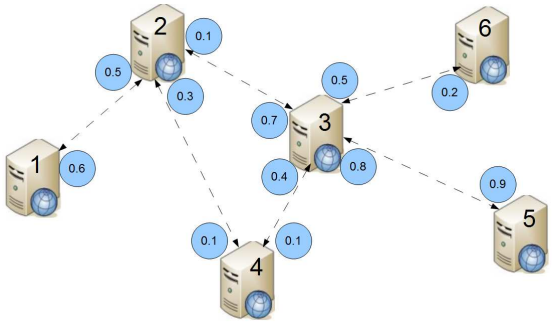

FIG. 6.5. *P2P Topology with Confidence Parameters*

the gossiping area for confidence computation. This is the reason to call it a local algorithm, because it returns a value that characterizes only a subset of nodes.

A node will locally store its confidence values associated for each connection it has with its neighbors. The *getConfidence(P, N)* function computes the confidence of a certain node N in one of its neighbors P according to the formula:

$$confidence(P,N) = \frac{1}{n}\sum confidence(child), child \neq P \tag{6.1}$$

6.2. Hybrid Monitoring. Now, considering the fact that some nodes might not be considered *interesting* from a distributed computation point of view, their neighbors should not choose them as gossiping partners. Thus, the probability of selecting a peer must be calculated according to the *confidence* that the node should have in that peer. The confidence was introduced above and is going to be the key element in the selecting phase of the monitoring algorithm.

This feature is related to ants behavior, as they are used to follow pheromone traces in their path towards food. In a similar way, the most persistent traces (the highest confidence) are going to indicate the selected peer.

Due to the fact that two bio-inspired approaches are combined, starting from a gossip-based model and adding ant colony features, proves that this algorithm is a hybrid one. It is relevant for optimizing the monitoring process, by focusing only on possible available nodes and ignoring those peers that are very unlikely to be able to participate in a distributed computing.

7. Conclusion. Developing the Virtual Middleware proposed by SORMSYS project highlight a new approach over the infrastructure of LSDS. SORMSYS project uses bio-inspired methods like gossip-based algorithms with ant colony features that are meant to ensure resource management and system monitoring. The advantages and the impact that gossip-based algorithms have over the LSDS will be analyzed. Moreover, the adaptive nature of the protocol, allows it to spread the information across the system, with no performance penalties, even when a high number of nodes join or leave the system. The excellent behavior of the monitoring algorithm is also strengthened by the results obtained in OverSim when monitoring systems of different sizes: from 100 to 1000 nodes.

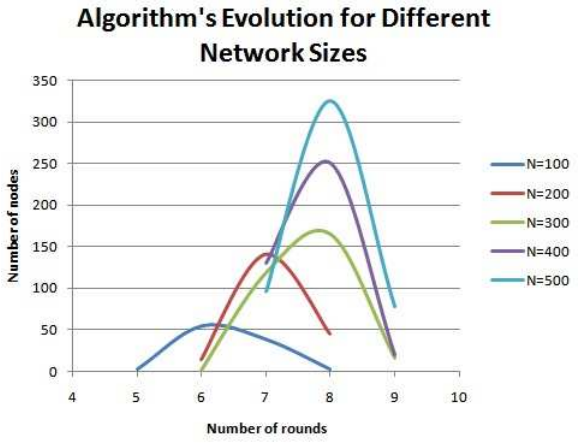

FIG. 6.6. *Number of rounds executed by nodes for different network sizes*

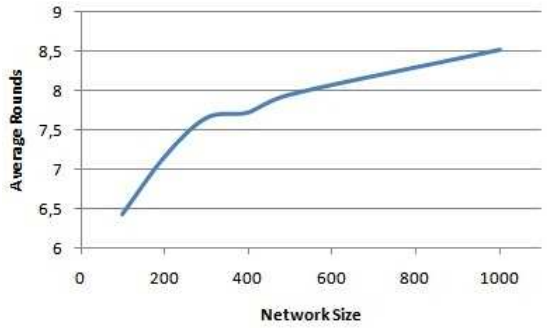

FIG. 6.7. *Average Rounds for Different Network Sizes*

Considering the fact that each node is limited as computing capacity in comparison with the system's size, the real power of computation is going to be the system as a whole, not the individual machines. In such an environment, centralized models based on master-worker or client-server paradigms are not compliant, as the huge number of requests coming from different peers will most probably overwhelm any central point considered. As a centralized model is not acceptable in the specified conditions, the future work will focus on fully-decentralized models that provide scalable solutions for monitoring and controlling the environment. The system will reach a high level of knowledge regarding its state in any node, so that it will be able to self-organize in order to deal with unexpected events. In this LSDS, it will be impossible to gather and maintain a detailed and up-to-date list of all nodes participating in any large computation. Gossip-based algorithms are known to exhibit a very good convergence and resilience properties in the present of constant unexpected events.

Acknowledgment. The research presented in this paper is supported by national project: "SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems", Project CNCSIS-PN-II-RU-PD ID: 201 (Contract No. 5/28.07.2010).

REFERENCES

[1] FLORIN POP. *SORMSYS: Towards to a Resource Management Platform for Self-Organizing Large Scale Distributed Systems*, Proc. of SYNASC 2010, the12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 23-26, 2010, pp: 534-540, ISBN:3-642-14798-4.

[2] A.S. TANENBAUM AND M. VAN STEEN. Distributed systems. Prentice Hall; 2 edition (Oct 12 2006), ISBN: 978-0132392273

[3] FOSTER, I., KESSELMAN, C., AND TUECKE, S.. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 3 (Aug. 2001), 200-222.

[4] PEREIRA, A. L., MUPPAVARAPU, V., AND CHUNG, S. M.. Role-Based Access Control for Grid Database Services Using the Community Authorization Service. IEEE Trans. Dependable Secur. Comput. 3, 2 (Apr. 2006), 156-166.

[5] Trunfio, P., Talia, D., Papadakis, H., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., Vlassov, V., and Haridi, S.. Peer-to-Peer resource discovery in Grids: Models and systems. Future Gener. Comput. Syst. 23, 7 (Aug. 2007), 864-878.

[6] Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., and Karl, W.. Scientific Cloud Computing: Early Definition and Experience. In Proceedings of the 2008 10th IEEE international Conference on High Performance Computing and Communications (September 25 - 27, 2008). HPCC. IEEE Computer Society, Washington, DC, 825-830.

[7] M. Armbrust, A. Fox, R. Grith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[8] Liben-Nowell, D., Balakrishnan, H., and Karger, D.. Analysis of the evolution of peer-to-peer systems. In Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing (Monterey, California, July 21 - 24, 2002). PODC '02. ACM, New York, NY, 233-242.

[9] Suomalainen, J., Pehrsson, A., and Nurminen, J. K.. A Security Analysis of a P2P Incentive Mechanisms for Mobile Devices. In Proceedings of the 2008 Third international Conference on internet and Web Applications and Services(June 08 - 13, 2008). ICIW. IEEE Computer Society, Washington, DC, 397-402.

[10] Wang, S. C., Yan, K. Q., Wang, S. S., and Huang, C. P.. Achieving high efficient agreement with malicious faulty nodes on a cloud computing environment. In Proceedings of the 2nd international Conference on interaction Sciences: information Technology, Culture and Human (Seoul, Korea, November 24 - 26, 2009). ICIS '09, vol. 403. ACM, New York, NY, 468-473.

[11] M.A. Vouk. Cloud computing - Issues, research and implementations. Journal of Computing and Information Technology, 16(4):235-246, 2008.

[12] Yu, J., Li, Z., Hu, J., Liu, F., and Zhou, L.. Structural Robustness in Peer to Peer Botnets. In Proceedings of the 2009 international Conference on Networks Security, Wireless Communications and Trusted Computing - Volume 02 (April 25 - 26, 2009). NSWCTC. IEEE Computer Society, Washington, DC, 860-863.

[13] Gehlen, G., Aijaz, F., Zhu, Y., and Walke, B.. Mobile P2P Web Services using SIP. Mob. Inf. Syst. 3, 3,4 (Dec. 2007), 165-185.

[14] Buchegger, S., Schiberg, D., Vu, L., and Datta, A.. PeerSoN: P2P social networking: early experiences and insights. InProceedings of the Second ACM Eurosys Workshop on Social Network Systems (Nuremberg, Germany, March 31 - 31, 2009). SNS '09. ACM, New York, NY, 46-52.

[15] M. Arora, S.K. Das, and R. Biswas. A decentralized scheduling and load balancing algorithm for heterogeneous Grid environments. In Proceedings of International Conference on Parallel Processing Workshops (ICPPW'02), Vancouver, British Columbia Canada, pages 499 - 505, 2002.

[16] Florin Pop, Ciprian Dobre, and Valentin Cristea. Performance analysis of Grid dag scheduling algorithms using monarc simulation tool. Proceedings of 7th International Symposium on Parallel and Distributed Computing (ISPDC'08), July 1-5, Krakaw, Poland, 2008.

[17] Forestiero, A., Mastroianni, C., and Spezzano, G.. So-Grid: A self-organizing Grid featuring bio-inspired algorithms.ACM Trans. Auton. Adapt. Syst. 3, 2 (May. 2008), 1-37.

[18] Forestiero, A., Mastroianni, C., and Meo, M.. Self-Chord: A Bio-inspired Algorithm for Structured P2P Systems. InProceedings of the 2009 9th IEEE/ACM international Symposium on Cluster Computing and the Grid (May 18 - 21, 2009). CCGRID. IEEE Computer Society, Washington, DC, 44-51.

[19] Jan Sacha, Jeffrey Napper, Corina Stratan and Guillaume Pierre. Adam2: Reliable Distribution Estimation in Decentralized Environments, 30th International Conference on Distributed Computing Systems (ICDCS 2010), Genoe, Italy, June 2009

[20] B. Baker and R. Shostak. Gossips and telephones. *Discrete Mathematics*, 2(3):191–193, 1972.

[21] B.H. Junker and F. Schreiber. *Analysis of biological networks*. Wiley-Interscience, 2008.

[22] R. Xulvi-Brunet and IM Sokolov. Construction and properties of assortative random networks. *Arxiv preprint cond-mat/0405095*, 2004.

Edited by: Dana Petcu and Alex Galis
Received: March 1, 2011
Accepted: March 31, 2011