Decentralized and Fair Mutual Exclusion Protocol in Peer-to-Peer Systems

Conference Paper · January 2008

CITATION
1

READS
416

1 author:

Atef Ahmed Obeidat
Al Balqa Applied University
24 PUBLICATIONS **123** CITATIONS

SEE PROFILE

Decentralized and Fair Mutual Exclusion Protocol in Peer-to-Peer Systems

Atef Ahmed Obeidat
Department of computer science
Novosibirsk State Technical University
Novosibirsk, Russia
e-mail: atefob@hotmail.com

Vasily Vasiliyevich Gubarev
Department of computer science
Novosibirsk State Technical University
Novosibirsk, Russia
e-mail: gubarev@vt.cs.nstu.ru

Ali Ahmad Al-yousef
Department of Computer Science
Huson College University
Huson, Jordan
e-mail: alimalkawi @yahoo.com

Abstract[1]

This paper presents decentralized mutual exclusion algorithm for dynamic Peer-to-Peer systems in which processes communicate by asynchronous passing messages and the resource has many replicas. In order to be practically useful, mutual exclusion protocol not only needs to be safe and live, but it also needs to be fair across clients. We propose a protocol to satisfy the fairness which organizes the replicas of the resource as a logical group and it always has a coordinator to keep the replicas consistent. The proposed protocol is based on the token approach. The proposed protocol presents solving of the mutual exclusion in distributed manner by maintaining the queue of requests kept in the replicas and requesting nodes with an efficient way that reduces the message overhead. We analytically prove the correctness of our algorithm and experimentally evaluate its scalability and efficiency by using simulations.

Keywords: mutual exclusion, election, fairness, distributed algorithm, peer-to-peer, distributed systems.

1. Introduction

The processes in Peer-to-Peer (P2P) systems share different types of recourses; the recourses can be either computational resources or data; it is necessary to avoid multiple simultaneous accesses to the resources mentioned above.

The distributed mutual exclusion (DME) protocols manage the access to a shared resource by single process at any time in distributed environment; in general the process enters its critical section (CS) to achieve mutual exclusion. It was first described and solved by Dijkstra in [1]. DME introduces some new requirements as summarized in [2-4].

There are many solutions for DME in dynamic P2P systems. These solutions belong to the field of distributed hash table (DHT) routing, as in [5, 6]. These solutions have many disadvantages (e.g. single-point of failure, unfairness, and the high message overhead).

A fairness condition for mutual exclusion in P2P systems called first-come-first-served (FCFS) is important for many systems (e.g. Banking systems, and ticket agent systems). In addition, their databases are generally replicated to enhance reliability or improve performance. A common requirement for replicated data is consistency. Informally, this means that when one copy is updated, we need to ensure that other copies are updated as well; otherwise the replicas will no longer be the same. These systems require consistency between the replicas of the databases all the time, and achievement of the fairness.

The existing mutual exclusion protocols (e.g. [5, 6]) do not satisfy these requirements. After crashing of the majority of the replicas at the same time, they may grant to the first new request. In this case, clients access the resource without order and may cause the violation of the mutual exclusion requirement (i.e. fairness).

We develop a new protocol in dynamic P2P systems in this paper in order to satisfy fairness, to achieve decentralization and reduce the message overhead. The basic idea is maintaining the queue of requests at all the requesting nodes and at all the replicas of the resource to achieve the decentralizing characteristics. We also propose a mechanism to address replica crashes and memory losses by electing a coordinator between replicas of specific resource to keep the consistency between them. After crashing, when any server re-enters the system, it must update its memory as the coordinator at the beginning; then it will be ready to give the grant for

Proceedings of the 10th International Workshop on Computer Science and Information Technologies CSIT'2008, Antalya, Turkey, 2008

the requests. In addition, the coordinator is the only one that can give the authority to a new client in order to access the shared resources. Initially, the token is situated at the coordinator of replicas, then it is exchanged between the owners of shared resource through the release messages and at the moment when there are no requesters else, it will return again to the coordinator waiting any new requesting to access the shared resource. The developed protocol presents a competitive level of performance and achieves the requirements of DME for providing efficient and reliable access to shared resources.

2. Related Literature

There are two different approaches in distributed systems to solve the mutual exclusion problem as it is described in the classification in [2]. The first approach is called token-based mutual exclusion [e.g. 11, 12], and the second one is called permission-based [e.g. 7, 8, 9, 10].

DME protocols of dynamic P2P systems tend to fall into two categories as detailed in the survey paper in [4]: they include the protocols in the method of DHT routing and the protocols in the method of semantic routing. There are many algorithms in the field of DHT routing. Reference [5] was the first protocol for providing DME in dynamic P2P DHT systems, it is called Sigma. It is implemented inside a P2P DHT and adopts queuing and cooperation between clients and replicas to enforce a quorum consensus. It has a high message overhead, and a central-point of failure and increased load point on a few nodes (i.e. replicas), when we assume that the number of requesters is very large. In addition, its service policy does not guarantee FCFS - this case happens when the replica fails. It returns to the system with fresh memory, and it may vote for any new requester since client requests can take arbitrarily long to arrive. Thus, Sigma can be best described as quasi -FCFS. The second, End-to-End (E2E) mutual exclusion protocol maintains the queue of requests at first N waiting nodes to use a particular resource, as in [6]. It relies on N nodes to maintain the queue of requests for the resource, leading to a less fault tolerant system due to a central point of failure and increased load on a few set of nodes because the system is dynamic and we can't expect the number of requester at the moment. E2E protocol also doesn't guarantee the service rule "first come first served" (FCFS) (i.e. fairness) as in Sigma. E2E protocol doesn't present more improvement in the message overhead than the others. The last protocol is Non-End-to-End (NonE2E) protocol, as in [6], the fundamental idea behind this protocol is to maintain a partial queue of requests at all the nodes in the quorum set rather than a complete queue at only the accessing node. Information maintained in these partial queues can be consolidated when the current node exits the critical section, to determine the next node in line to use this resource. It has disadvantages as others but it is more distributed.

In recent years, the token based fair K mutual exclusion algorithm for p2p systems using a single forest tree structure was proposed in ref. [13]. It achieves improvement in fairness characteristics by minimizing the speed of access time. The algorithm for solving DME problem is introduced in Ref. [14]. It is based on the multiple tokens ring approach. Each competing process generates a unique token and sends it as request to enter the critical section that travels along the ring. The process enters the critical section if it gets back its own token.

3. System Model

The protocol presented in this paper is based on a system model that represents dynamic P2P DHT systems. The basic entity in the system is called node (peer). We assume that each node has a unique *nodeId*. The nodes are connected over a P2P DHT that allows any node to route a message to any other. The number of nodes (clients) is unpredictable and may be very large. There may be high churn in the system – nodes may enter and leave the system anytime.

The shared resource corresponds to a set of nodes (i.e. replicas), they may be virtual (e.g. a file or a computational resource). We consider one from the active replicas as a coordinator, in order to keep the consistency between the replicas. The replicas for a resource are always available, but their internal states may be randomly reset due to a crash-recovery failure. They rejoin the network with the same previous number *nodeId*, and they recover refresh their memory according to the memory of the coordinator.

4. Proposed Protocol

The existing protocols present a good solution for the mutual exclusion problem in P2P systems, but they don't fully satisfy many requirements of this domain. The main disadvantage is single-point of failure. We will be able to achieve better distribution characteristics in proposed protocol by maintaining a complete queue of future requests at all the requesting nodes and at the replicas of the resource rather than at the current owner of the resource (or at the first N nodes in the queue) as it is in the case of E2E protocol [5], or complete queue at replicas of resource as it is in the case of Sigma protocol [6]. Another disadvantage is that the service policies are not FCFS at all the existing protocols. We can call them quasi-FCFS in case of failure of a replica. We can make the service policy be FCFS by specifying the responding to a new request by the coordinator. Message overhead will be lower than in other protocols because the protocol uses less number of operations than the existing protocols. As an example, the response message is from the coordinator of replicas instead of N messages from the replicas as in exiting protocols. The proposed mutual exclusion protocol is presented in fig. 1. In the following subsections, we will explain the proposed protocol in details.

4.1. Basis of the new protocol

The proposed protocol is based on message passing between the requesting nodes and the replicas. The replicas cooperate with the coordinator to achieve fair responding of the requests. The replicas of the resource are organized as a group. The group always has one replica that has been "elected" to serve as the central arbitrator (i.e. coordinator) to keep the integrity between the replicas of the resource and response on the new requests. Every replica has identifiers: Coordinator, and T_{coor} to assign the current coordinator and its *Start_Time*. Initially, the first created replica will be the coordinator.

When a replica receives an incoming message from the requesting nodes (e.g. REQUEST, RELEASE) or RECOVER messages from other replica, the first course of action is to determine if it is the coordinator or not. If replica is the coordinator, it sends the suitable answer to source of message (e.g. RESPONSE, UPDATE). On the other hand, if the replica finds that the coordinator is not active, it begins the election operation; otherwise, it just passes the message to coordinator.

The coordinator election

We proposed a simple and efficient algorithm to elect the next coordinator. It consists of appointing a replica-coordinator from an initial configuration in which all replicas are in the same state, that is, all are candidates and can become a coordinator. When any replica discovers that the current coordinator failed, it sends the message *IAMCOORDINATOR* to all active replicas. When the other replicas receive the message it simply appoints the new coordinator. The algorithm solves the problem of concurrent issuing electing using more than one replica by assigning the last one according to its timestamp. We use timing as proposed by Lamport in [15]. Fig.1 explains pseudo-code of the election operation.

Recovering the memory

After crashing, when the replica re-enters the system it in the beginning sends the message "RECOVER" to all active replicas. When the coordinator receives the message, it directly sends UPDATE message to the new replica. But if any other replicas receive the message, they firstly check if the coordinator is active or not. If the coordinator is not active, the replica must begin ELECTION operation in order to find a new coordinator. Then the replica completes the recover operation; it just passes the message to the coordinator, because just the coordinator can response to "RECOVER" message by sending UPDATE message to the source of the message. The recover operation is explained in fig.1.

```
Client side pseudo-code:
Timestamp=GETTIMESTAMP();//state variable saves initial node's clock.
//requsting:
incrementTimeStamp();
for each ri in replicas r
        Send (REQUEST,timestamp) to ri ;
// releasing:
IncrementTimeStamp();
for each ri in replicas r
        Send (RELEASE,timestamp) to ri ;
for each Ci in Queue q
        Send (RELEASE,timestamp) to Ci ;
onDeliver(from,msg){
switch msg.type is
  RESPONSE:
    if (msg.resource_avialabe) enterCriticalSection();
    else {queue=msg.queue;
        queue[R].insert(myId,msg.timestamp);}
  RELEASE:
    unLockResource(R);
    owner[R]=queue[R].removeFirst();
    if (owner[R]== myId) enterCriticalSection();
    if(owner[R]!=nill) lockResource(R);
  ADDREQUEST:
    Queue[R].insert(msg.sourc1e,msg.timestamp);
  PING: send(msg.from,myId)
}
Replica side pseudo-code:
Coordinator, Tcoor;  //state variables used in all replicas of resource.
Periodically:
        Send (PING) to Coordinator;
        Wait periodOfTime();
Active(Id){
         Send (PING) to  Id;
        Wait(periodTime)
        If  receive ok Return true; Else return false;}
Updating(){
for all active ri in replicas r
        Send (RECOVER,timestamp) to ri ;}
Election(){
   For all active ri in replicas r
        Send (IAMCOORDINATOR, timestamp) to ri ;}
onDeliver(from,msg){
  switch. Msg.type is
    IAMCOORDINATOR:
        If (msg.timeStamp> Tcoor ){
                Coordinator=messege.from;
                Tcoor =message.timeStamp;}
    RECOVER:// rebuild memory
     If ( received (msg) break;
     If (msg.from==Coordinator || !active (Coordinator))
        Election();//current coordinator failed
     If (myId==Coordinator) send(UPDATE) to message.from;
     Else  send (RECOVER, message.from) to Coordinator;
    PING: send (msg.from,Ok);
    REQUEST:
     if (received (msg) break;
      if(myId==Coordinator){ send(RESPONSE) to msg.from;
      if (!available(resource)){
        for each ri in replicas r
          Send (ADDREQUEST,msg.timestamp) to ri ;
        for each Ci in Queue q
          Send (ADDREQUEST,msg.timestamp) to Ci ;}
      }
      else if (active(Coordinator))  Send (REQUEST) to Coordinator;
        else{ Election();
              Send (REQUEST) to Coordinator;
        }}
```

Fig 1. The proposed mutual exclusion protocol

The requesting operation

Any node wanting to request a resource does so by issuing its request to the group of replicas. The set of responsible replicas for a given resource can be found

using the Peer-to-Peer Replica Location Services (P-RLS), as in [16]. The coordinator listens to such requests, processes and responds to them appropriately. Only the coordinator makes a decision regarding the resource request, it sends the responding message to requester; the message must contain the queue and the current owner of critical section. In addition, it sends *ADDREQUEST* message to all requesters that are waiting in the queue and to the entire group of replicas. The requesting node, upon receipt of this response, enters the critical section if it has been determined by the coordinator that the resource is available (i.e. the resource is free and queue is empty). All nodes (other than the node that has been granted access to the resource) simply mark the resource as "in use" and wait when the node exits the critical section to broadcast a release message before attempting to request the resource again. The operations of the protocol are described in details below:

1. Node A, wishing to enter the critical section, sends a REQUEST to the group of replicas requesting the use of resource R.
2. Node Q, which is currently acting as the coordinator for the group of replicas, receives the request for resource R from Node A. Node Q determines that resource R is available and locks resource R for Node A. At the same time, Node B issues a request for resource R as well. Before dealing with the next incoming message (which is Node B's request for resource R), Node Q sends its response (i.e. ADDREQUEST) to the requesters and to other replicas regarding the status of resource R (i.e. the response message contains status of resource, and queue of requests). A complete queue of requests is maintained at all requesters and replicas.
3. Node "A" receives the response from the coordinator and notices that the coordinator has granted access to resource R for Node "A" (i.e. queue of requests is empty and the resource is available). Node "A" enters the critical section and begins to use resource R. At the same time, Node B also receives the response and sees that resource R is used by Node "A". It marks resource R as "in use" and enters the request into its queue.
4. Upon finishing its work in the critical section, Node "A" issues a RELEASE message to the entire queue of requests and coordinator, indicating that it is done using the critical section. All nodes, including Node "B", mark resource R as being available again. Node "B" is now free to issue a request for resource R.
5. Steps 1-4 are repeated as nodes wish to access the critical section of the system.

4.2. The node's internal state

The node's internal state information includes its myId, the owner of the resource (i.e. Owner), the variable T_{Owner}, which stores the timestamp of the request; the list Queue, which stores the time stamped order of waiting clients (requesters). We use Lamport's logical clock to generate timestamp, as in [15]. The information in the clients and replicas is consistent because the protocol uses the Coordinator to coordinate between them.

4.3. Types of messages

The communication between the requesting nodes and replicas of the resource is done by the following types of messages:

1. Message type of *REQUEST*. It is sent by a client to replicas. If the message is received by the coordinator, the coordinator responds to the message by sending *RESPONSE* message with the status of the resource. If the resource is free, the coordinator locks the resource and informs the other replicas. But if the resource is busy, the coordinator adds the request to waiting queue and also sends *ADDREQUEST* message to all replicas and the nodes which have requests in queue. When any replica receives a REQUEST message, it pings the coordinator if it is active or not. If the coordinator is active the replica passes the request to it, else the current replica does the election.
2. Message type of RESPONSE. It is received by the requesting node from the coordinator. The message contains the status of resource, the queue of requests. As a result, a new node just waits until receiving RELEASE (token) message from previous owner. It may directly enter CS if the resource was unlocked.
3. Message type of ADDREQUEST. When a node receives this type of message, it adds the request into the queue, but if the queue is empty and the resource is free, the requester will be directly the owner.
4. Message type of RELEASE. It is routed from the current owner of the resource to waiting nodes in the queue. If the current client is the owner, it succeeds and gets the permission to enter CS directly. Someone else does win but not this element. This element does nothing because it knows it has already been registered in the queue. It will enter later and now it is just waiting.

4.4. Failure Handling

The proposed protocol works under failure-free environments. Any node in the system may crash and the nodes communicate via messages across unreliable channels. In the following, we will explain how the protocol may solve the different types of failures:

1. Failure of node currently presented in critical section. This case is solved by applying the principle leasing, as in [12]: the client is granted permission to enter CS for a specific time. When the lease expires, the next requester in queue will get the permission. The node can estimate the waiting time in queue in advance (i.e.

forecasting waiting time $t_w = l \times t_{cs}$, where l is the number of nodes in queue, t_{cs} - time in critical section)

2. Failure of requester or replica during different communications. This failure is detected by the "PING/ACK" mechanism between the sender and receiver.

3. The unreliable communication channel between nodes of the system will cause similar problems as well. The lost messages in all protocols are detected using a "PING/ACK" scheme, with constant overhead.

4. Failure of the coordinator discovered by any other replica when the replica needs to connect with it for updating memory or passing the requesting messages.

4.5. Correctness of the Proposed Protocol

Correctness of P2P DME protocols can be guaranteed if the requirements of safety, liveness, and ordering (fairness) are upheld, that is established through the following lemmas.

Lemma 1. *The proposed protocol ensures that node p can enter CS, if it satisfies one of the following cases:*

a. If it received the authority from the coordinator by the RESPONSE message.

b. *If it received the RELEASE "token" message, and it requested at the front of the queue (i.e. p=q[1]) where q is queue of waiting requests.*

Proof a: this case happens when a new client requests to enter CS, and if CS is free. At the beginning, the client doesn't have the queue of requests, so, as the responses arrive at client, it gradually forms the idea about its place in race. The coordinator answers the requester by sending RESPONSE message that contains the queue of waiting requesters and/or state of resource if it is available or not. If resource is available, then the client will be the owner and it can enter CS, else the client will wait and it will use CS according to case b.

Proof b: in this case, the client p has already got the queue of request and its request is already waiting its turn in the queue ***q***, which is ordered according to time-stamp. After the RELEASE "token" message is received, the first node in the queue *q* is granted to access the CS and others just wait. So, if the node p is at the front of q (i.e. *p=q(1)*), it will be the new winner to access CS.

Lemma 2. *The new protocol ensures mutual exclusion***.**
Proof 2*:* the replicas and requesting nodes of the resource R_i maintain a complete waiting queue ordered with time-stamp. Only the first element in queue can enter CS (see Lemma 1.b), and when it exits the critical section, it sends a release message to waiting clients in line, so the following requester can enter CS. Hence the lemma follows. For the new client, it may enter CS if it receives authority from the coordinator of replicas (see Lemma 1.a).

Lemma 3. *The protocol is deadlock-free.*
Proof 3*:* Deadlocks are avoided by using "ping/ack" mechanism and the principle of leasing.
Lemma 4. *The protocol is starvation-free.*
Proof 4: a node cannot wait indefinitely long because the requests are ordered by timestamp and all the requesting nodes and replicas maintain the requesting queue. Only the coordinator of replicas can give grant to enter CS.
Theorem 1. *The new protocol is correct.*
Proof theorem 1: Follows directly from Lemmas 2 – 4.

5. Experimental Results

The proposed and the existing protocols have been implemented over the FreePastry [17] implementation. The test-bench code starts off by creating a set of nodes, arranged according to the Pastry network topology. The simulation then runs for a certain number of rounds and derives the experimental results by over ten such runs in average. In the simulation, the concept of a round is introduced in order to describe the amount of work done by the nodes in the network, which includes a certain number of requests for resources being issued by a set of randomly chosen nodes. Also in each round, a node randomly releases 50% of its held resources.

5.1. Scalability

The main goal of the proposed protocol was to come up with an efficient and scalable method for achieving DME. Fig.2 presents a comparison of the proposed protocol with exiting protocols. Even though the plots are linear with respect to the number of nodes, the proposed protocol requires about 2 times fewer messages than the E2E protocol in average. The number of resources, replicas and requests per round are held constant at 65, 10, and 20 respectively; the number of rounds is 20.

5.2. High Churn Rate

Rate of churn is defined as the sum of nodes that are in transition (i.e., either leaving or joining the system per round). In our simulation with a number of nodes =2000, n/2 randomly chosen nodes are failed and n/2 new nodes are added to the system, where n is the desired churn rate. Requesters are not churned in our experiments. Fig. 3 shows the effect of increasing the rate of churn from 0 to 1000 on the overall message overhead. The new protocol experience is a linear increase in message overhead when the rate of churn increases.

6. Conclusion

In this work, we have proposed a new protocol for achieving DME in a P2P system in order to satisfy fairness and decentralization, in addition, to reduce the message overhead. We have presented a novel method which uses a coordinator between replicas of the resource to keep the consistency between them in order to satisfy the requirement "fairness". The protocol also presents a way for local decision-making. It is able to keep the load on the replicas relatively low even in the presence of a growing network and high churn rates. The protocol can

be easily configured to run on any DHT; and consequently provide a truly generalized solution for the addressed problem.

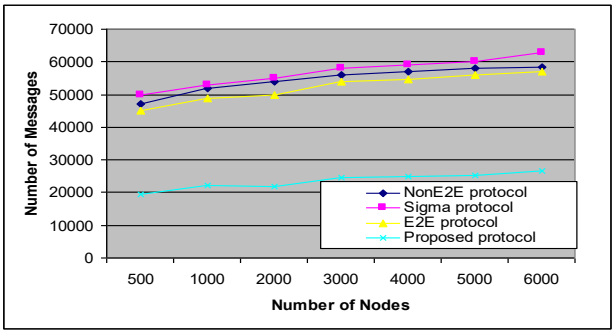

Fig 2.Comparison between the protocols: scalability, message overhead

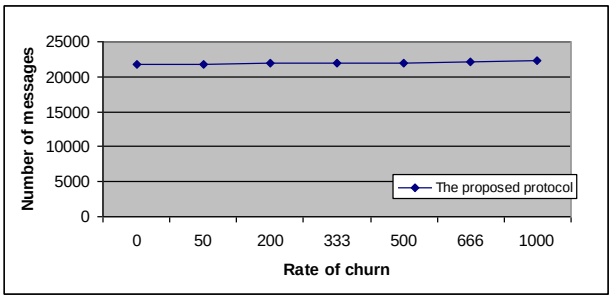

Fig 3.Effect of Churn on message overhead

References

1. E. W. Dijkstra, "Solution of a problem in concurrent programming control", *Communications ACM, vol. 8, 9 (Sept. 1965)*, p.569.
2. M. G. Velazquez. "A Survey of Distributed Mutual Exclusion Algorithms". Colorado State University, Technical Report CS-93-116.
3. M. Raynal, "A simple taxonomy for distributed mutual exclusion algorithms", *ACM SIGOPS Operating Systems Review*. Vol. 25,1991, pp.47-50.
4. A. A Обейдат., B. B. Губарев, "*Обзор алгоритмов распределенного взаимного исключения в динамических пиринговых системах"*, Сборник Научных Трудов НГТУ. vol. 2(48), 2007,pp. 63-68.
5. Shiding Lin, Qiao Lian, Ming Chen, and Zheng Zhang", A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems". *In 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, 2004.
6. Moosa Muhammad. "Efficient Mutual Exclusion in Peer-To-Peer Systems". University of Illinois at Urbana-Champaign, 2005.
7. D. Agrawal and A. El Abbadi," An efficient solution to the distributed mutual exclusion problem", in Proc. *8th ACM Symposium on Principles of Distributed Computing*, 1989, pp. 193-200.
8. M. Maekawa, "A √N algorithm for mutual exclusion in decentralized systems", *ACM Transactions on Computer Systems*, 1985 ,pp. 145-159.
9. G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks", *Communications of the ACM*, , 1981, pp. 9-17.
10. M. Singhal," A dynamic information structure mutual exclusion algorithm for distributed systems". *IEEE Transactions on Parallel and Distributed Systems* , 1992 ,pp. 121-125.
11. J. M. Helary and N. Plouzeau and M. Raynal, "A distributed algorithm for mutual exclusion in an arbitrary network", *volume 31 of Computer Journa*l, ,1988 ,pp. 289-295.
12. A. J. Martin, "Distributed mutual exclusion on a ring of processes", *Science of Computer Programming*, pp. 1985,256-276.
13. Korthikanti, Vijay Anand, Prateek Mittal, Indranil Gupta. "Fair K Mutual Exclusion Algorithm for Peer to Peer Systems", *Proc. International Conference on Distributed Computing Systems (ICDCS)*, 2008.
14. Md. Abdur Razzaque, Choong Seon Hong. Multi- "Token Distributed Mutual Exclusion Algorithm", *22nd International Conference on Advanced Information Networking and Applications (AINA 2008)* ,2008, pp. 963-970.
15. L. Lamport, "Time, clocks and the ordering of events in a distributed system", *in Communications of the ACM*, vol. 21, 1978, pp.558-565.
16. Cai, M., Chervenak, A., and Frank, M. "A peer-to-peer replica location service based on a distributed hash table", *Proceedings of the SC2004 Conference*, November 2004.
17. FreePastry. http://freepastry.rice.edu/