Efficient Mutual Exclusion in Peer-to-Peer Systems

Moosa Muhammad, Adeep S. Cheema, and Indranil Gupta

***Abstract*— Due to the recent surge in the area of Grid computing, there is an urgency to find efficient ways of protecting consistent and concurrent access to shared resources. Traditional peer-to-peer (p2p) applications such as Kazaa and Gnutella have been primarily used for sharing read-only files (such as mpegs and mp3s). This paper introduces two novel protocols, the End-to-End and Non End-to-End, for achieving mutual exclusion efficiently in dynamic p2p systems. The protocols are layered atop a distributed hash table (DHT), making them scalable and fault-tolerant. The burden of controlling access to the critical section is also evenly distributed among all the nodes in the network, making the protocols more distributed and easily adaptable to growing networks. Since the protocols are designed independent of any specific DHT implementation, they can be incorporated with any generic p2p DHT, depending on the application requirements. We present experiments comparing our implementations with existing mutual exclusion algorithms. The significant reduction in overall message overhead and better load-balancing mechanisms makes the proposed protocols very attractive in being used for current and future p2p and Grid applications.**

***Index Terms*— Distributed algorithms, Distributed computing, Resource management, Token networks**

I. Introduction

THE problem of mutual exclusion can be described as a collection of asynchronous processes, each alternately executing a critical and a non-critical section that must be synchronized so that no two processes ever execute their critical sections concurrently. It was first described and solved by Dijkstra in [2].

Even though mutual exclusion is a well-studied problem in distributed systems, it is not possible to directly adapt the proposed solutions into the p2p domain. This dilemma is caused by the differences in the underlying system models, one of which being the absence of a centralized index server to keep track of membership and to ensure consistency. Also, classical distributed algorithms use several rounds of all-to-all communication which is unscalable.

The three most important characteristics that all mutual exclusion protocols for p2p systems should demonstrate are:

- Scalability – Since 1000's of nodes can be actively participating at any given time, the protocol should be scalable with system size.
- Fault tolerant – Failure of nodes should be gracefully handled and should not pose a large background overhead or break the correctness of the protocol.
- Churn resistant – The dynamic nature of p2p systems should be taken into account when designing protocols. Typical p2p systems experience high churn rate, i.e., nodes join and leave the network at a high rate.

Resources that are being shared can be either computational resources or data, and access to them should be controlled in an efficient and completely decentralized manner. Since each resource can have multiple replicas, the problem in question becomes even more challenging. Access to that resource is controlled by its set of replicas.

The proposed mutual exclusion protocols combine token and quorum-based approaches, to provide efficient and reliable access to shared resources in dynamic p2p systems. The algorithms demonstrate good load-balancing characteristics and low message overhead, when acquiring access to a resource. They utilize a route-based scheme such that a quorum set is constructed for every replica, based on the route traversed by a request to reach that replica. The replica's quorum set thus comprises of every intermediate node lying on the DHT-based route between the requester and itself. This translates to O(log n) quorum set nodes per replica and O(log n) messages, where n is the number of nodes in the system, using an underlying DHT routing scheme like Chord [23] or Pastry [20]. The use of these quorum sets allow the protocols to scale well with system size since a large number of messages will be intercepted before reaching the replicas.

II. SYSTEM MODEL

The protocols presented in this paper are based on the following system model representing a dynamic p2p DHT:

- The basic entities in the system are called nodes (or peers).
- Each virtual resource (e.g., a file or a computational

Manuscript received June 3, 2005. This work was supported in part by the NSF CAREER grant CNS-0448246 and in part by NSF ITR grant CMS-0427089.

Moosa Muhammad was with the Computer Science Department, University of Illinois at Urbana-Champaign, USA. He is now with Motorola, Inc. (e-mail: mmuhamma@motorola.com).

Adeep S. Cheema was with the Computer Science Department, University of Illinois at Urbana-Champaign, USA. He is now with Microsoft, Inc. (e-mail: adeepc@microsoft.com).

Indranil Gupta is with the Computer Science Department, University of Illinois at Urbana-Champaign, USA (e-mail: indy@cs.uiuc.edu).

resource) corresponds to a set of nodes (i.e., replicas) that are responsible for granting access to that resource.
- A node can access a resource if and only if each responsible replica grants access to it.
- Nodes are connected over a p2p DHT that allows any node to route a message to any other node.
- 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 nodeId.
- The number of clients is unpredictable and can be very large. Clients are not malicious.
- There may be high churn in the system – nodes may enter and leave the system anytime.
- Clients and replicas communicate via messages across unreliable channels. Messages can be replicated, lost, but never forged.

III. RELATED WORK

A. Mutual Exclusion

Distributed mutual exclusion protocols tend to fall into two categories as detailed in survey papers [18] and [27]. These include token based protocols [17] and quorum based protocols [19] [24] [13], which intersect at completely centralized exclusion. This study proposes a hybrid between these two sets of protocols, with a competitive level of performance and adherence to general guidelines established for such protocols.

Previous work by researchers in the mutual exclusion domain have set certain performance thresholds and presented a few common tradeoffs. Maekawa's algorithm [13] improves upon Ricart and Agrwala's [19] completely distributed approach and Suzuki's algorithm for instance, by grouping nodes into overlapping sets (i.e., quorum sets) and reducing the number of messages from O(n) to O(√n). Certain assumptions on topology can be used to reduce this to O(log n) messages [17]. [15] shows how quorum based protocols can be analyzed for a random distribution of nodes. [11] presents an initial attempt at an algorithm for achieving mutual exclusion in dynamic p2p systems, but lacks a detailed analysis of their work.

B. P2P Systems

Several applications have been built and deployed on p2p DHTs, such as distributed file systems and various resource sharing overlays such as POST [14], PAST [21], SCRIVENER [16], and SQUIRREL [8]. In the physics and medical communities for instance, large pools of networked computing resources are needed to solve computationally intensive tasks. Some currently active projects within these domains include the Grid Physics Network [6] and the Human Proteome Folding Project [26]. This gives rise to the challenge of coming up with efficient resource management and sharing mechanisms that scale appropriately. Some previous protocols developed for this purpose include the Grid [3] and SHARP [5].

Prior research conducted in efficiently routing messages to the nodes holding a particular resource include, the Chord and the Pastry protocols. Both are examples of structured p2p DHTs and choose their neighbors intelligently to lower the latency and message cost of routing (i.e., lookups and inserts) to just O(log n). Pastry differs from Chord in that it arranges its nodes based on their geographical locality. Kelips [7] is another p2p DHT that makes a tradeoff by consuming greater amount of memory and constant background communication in order to reduce file lookup times and increase resistance to failures and churn.

C. Sigma Protocol

Sigma [11] is the only currently existing protocol for providing mutual exclusion in dynamic p2p systems. It is implemented inside a p2p DHT and adopts queuing and cooperation between clients and replicas, to enforce a quorum consensus. It utilizes the fact that nodes in the DHT collectively form a logical space that does not have holes, institute a set of logical replicas upon which a quorum consensus protocol grants access to the critical section. It deals with failures using a combination informed back-off and lease mechanisms.

IV. PROPOSED PROTOCOLS

The Sigma protocol is a step in the right direction, but it does not fully utilize the decentralized nature of p2p domain. It relies on the replica 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. Along with addressing these problems, the following proposed solutions also achieve better performance and load balancing characteristics, which play an integral part in p2p systems.

The two protocols below differ in terms of how well they conform to the End-to-End argument [22]. This metric helps us to find the best placement for a mutual exclusion mechanism, in a p2p system.

A. End-to-End Mutual Exclusion Protocol

In order to achieve better load-balancing characteristics, this protocol maintains the queue of future requests at the node that is currently in the critical section, instead of at the replica (as was the case in the Sigma protocol). It defines the quorum set for gaining mutually exclusive access to a responsible replica R to be every node in the path from the requesting node to R. Since all the nodes within the quorum set maintain information regarding the current owner of the replica, requests for a resource that is already being held by another node can be satisfied by any of them, by means of an ENQUEUE message being sent to the current owner. When the node has finished using the resource and leaves the critical section, it sends the queue of requests (i.e., token) to the node

who requested the resource the earliest. This can easily be determined from the request queue, which is kept sorted by timestamps of request messages received.

This protocol primarily focuses on providing mutual exclusion with a low message overhead, along with reducing the burden on the replicas, for controlling access to the critical section.

The internal state information maintained by each node is as follows:

- A set of request queues (one for each resource currently being accessed by this node). The queues keep track of requesters' nodeIds and are kept sorted by Lamport timestamps [10], in order to maintain fairness and avoid starvation.
- A replica list maintaining (Replica Id, Owner Id) pairs to keep track of which replica is held by (i.e., voted for) which node, for the active quorum sets that this node is part of. Next and Previous node pointers are also maintained for each replica list entry to aid during node failures.
- List of resources this node currently has access to.

The description of this protocol is as follows:

1. When a node wants mutually exclusive access to a particular resource, it sends requests to all the responsible replicas of that resource, using the underlying DHT routing mechanism. The set of responsible replicas for a given resource can be found using the Peer-to-Peer Replica Location Service (P-RLS) [1].
2. Intermediate nodes lookup their replica list for the intended resource id. If found, the REQUEST message is stopped being forwarded and instead an ENQUEUE message is sent directly to the node currently accessing the resource in context. Otherwise, the REQUEST message continues to be routed
3. Upon receiving the REQUEST message at the target node, that replica will check if it has voted already or not. A RESPONSE message is routed directly to the original sender of the REQUEST message. The RESPONSE message contains the id and timestamp of the replica's owner (i.e., the node this replica has voted for). If the replica has not voted before, this information would be that of the requesting node.
4. Whenever a requesting node receives a RESPONSE message, it checks to determine if it has received a majority of replica votes. If so, it sends an IAMWINNER message to all the replicas, declaring itself as the new owner of the resource. Next and Previous node pointers (part of the replica list entry) are updated as this message is routed to all the replicas. If nobody has accumulated enough votes in a round, the requesters send out a YIELD (i.e., RELEASE + REQUEST) message to each of the replicas that voted for it
5. When an ENQUEUE message reaches its destination (i.e., reaches the owner of the resource that is being requested), the owner adds the requester's id to its request queue.
6. In order to release a resource, a RELEASE message is routed beginning at the current owner of the resource and is targeted for all the responsible replicas for that resource. The intermediate nodes that this message traverses through depend on the Next pointers of each node's replica list entry. Clean-up occurs as this message is being routed to the replicas. Once the message reaches its target replica, owner information of that replica is reset.
7. When the owner of the resource is done using that resource and leaves the critical section, it sends a TOKEN message to the node next in line, by removing the head of the queue. The TOKEN message that is sent contains the remainder of the request queue.

Figure 1 shows a snapshot of the E2E Protocol, where Req1 is currently in the critical section and Req2's request to use the same resource is queued up at Req1. When Req3 sends out a request for the same resource, its request is intercepted by quorum set nodes (before reaching the replica), and therefore ENQUEUE messages are sent to Req1 and it adds Req3 to its queue. The three quorum sets that exist in this scenario are marked.

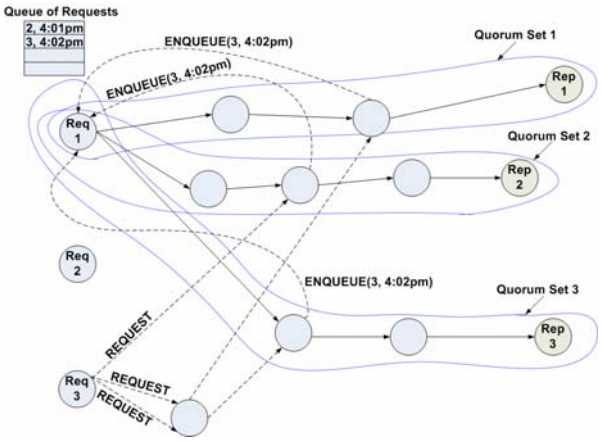

Fig. 1. A snapshot of the E2E Protocol

B. Non End-to-End Mutual Exclusion Protocol

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.

The message overhead is low, so is the burden of achieving mutual exclusion on the replicas (just like the E2E protocol). Furthermore, this protocol is more distributed and fault-tolerant, since the queue of requests previously being stored at one node is now split among the quorum set.

The internal state information maintained by each node is as follows:

- A replica list maintaining (Replica Id, Node Id, Replica Request Queue, Next, Previous) entries. Used to keep track of which replicas are held and subsequent requests for them, for the active quorum sets that this node is part of.
- List of resources this node currently has access to.

The description of this protocol is as follows:

1. When a node wants mutually exclusive access to a particular resource, it sends a REQUEST message to all the responsible replicas of that resource.
2. Intermediate nodes lookup their replica list for the intended replica id. If found then the request is queued locally on the node and a GRANTREQ message is sent to the requester, containing the current holder of the replica. Otherwise, their replica list is updated and the REQUEST message continues to be routed.
3. Upon receiving the REQUEST message at the target node (i.e., a replica), the replica will check if it has already granted access to the required resource or not. If it has, the request is queued on the replica itself. In either case, a GRANTREQ message is routed directly to the requester, specifying the holder of the replica (which will be the requester if the replica has not voted before).
4. In order to release a resource, RELEASE messages are routed to all the responsible replicas for that resource. All quorum set members update their replica list by deleting the respective entry from it and forward the message together with a list of requests, sorted by timestamp, seen so far by any of the nodes prior to them in the sequence from the holder to the replica. When this message reaches its target replica, it contains every request seen so far for that replica, and is merged with the local queue. The next owner of this replica is then determined from this newly assembled queue and GRANTREQ message is sent directly to that node.
5. When all GRANTREQ messages reach their destination, the requester checks if it has attained majority of the replica votes, to enter the critical section. If no requester receives majority of the votes, all of them propagate a YIELD message to the replicas. Otherwise only one of the requesters will receive the majority votes and can therefore access the resource.
6. The YIELD operation reflects the collaborative nature of this protocol and is used to reshuffle the queue. The fact that nobody wins indicates that contention has occurred. The YIELD message allows all requesters to try to acquire the resource again after a random time period. Typically, this self-stabilization process will quickly settle, as verified in [12].

Figure 2 shows the snapshot of the Non E2E protocol, where Req1 is currently in the critical section. Req2 and Req3 have issued REQUEST messages to also try to gain access to the same resource. Their requests are queued by the intermediate nodes of the respective quorum sets. When Req1 has finished using the resource and leaves the critical section, the partial queues will be merged (within each quorum set) to form the complete queue of requests at the replicas.

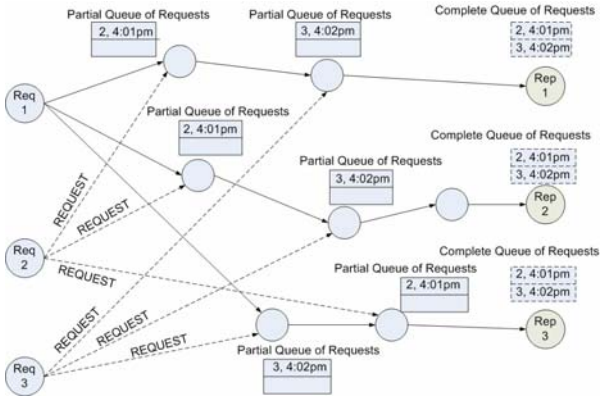

Fig. 2. A snapshot of the Non E2E Protocol

V. HANDLING FAILURES

The protocols described above work well under failure-free environments. The types of failures that are being considered in the analysis below include: (1) Failure of node currently present in the critical section (2) Failure of node(s) maintaining the queue of requests (3) Failure of intermediate nodes of the quorum set.

A. E2E Protocol

Failure of the node that is currently in the critical section and is therefore also holding the token (i.e., maintaining the request queue), can have a devastating effect on the entire system. A very clever scheme has been proposed to handle this kind of failure. Since the request queue maintains information for nodes that are waiting to use a particular resource, why not replicate and maintain the same queue of requests at the first n waiting requesters (based on timestamps of the requests submitted)? These n+1 nodes (including the node currently accessing the resource) can run a membership protocol within themselves, to propagate updates and also to determine if a particular node has failed. Whenever the request queue is updated, this change is propagated to the other n nodes of its membership. These updates are ordered by sequence numbers so that, in case of failure, we can use the most current copy of the request queue. “Ping” messages are sent periodically from n waiting nodes to the node holding the token, at a low frequency, for example 1 in every 10 minutes. This means that the failure of that node can be detected in just 10/n minutes. When the node holding the token fails, among the waiting nodes, the node with the highest sequence number will have the most updated copy of the queue. This node routes a message to all the replicas notifying itself as the owner of the resource. This failure detection/correction mechanism is tolerant to the failure of n waiting nodes, where n is a parameter that can be set in order to come up with the ideal tradeoff between extra bandwidth consumption and

fault-tolerance, for a particular application criteria.

Failure of intermediate nodes within a quorum set is detected using the Next and Previous pointers maintained by each node, as part of its replica list entry. In case of failure, queries are re-routed by the underlying Pastry layer using an alternate set of nodes.

B. *Non E2E Protocol*

Since the request queue is being maintained by all the intermediate nodes of the quorum set, the queue of requests would not be lost by a single node failure. In case a node in the path fails, its predecessor routes a new request message to the replica, creating a new quorum path, and its successor directs a cleanup message to clear the old quorum nodes from the failed node to the replica.

During a RELEASE message sequence, it can happen that the replica fails after receiving the partially assembled queue from its predecessor. The failure of the replica can be detected by the "Ping/Ack" mechanism between the replica and its predecessors. In such a case, an alive predecessor would act as the temporary token holder, where the new queue would be rebuilt.

VI. EXPERIMENTAL RESULTS

The E2E, Non E2E, and Sigma mutual exclusion protocols have been implemented over the FreePastry [4] implementation of the Pastry routing substrate. This allows us to accurately compare our proposed solutions against the Sigma protocol in various different network scenarios, using a virtual network simulation model with message-based communication.

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 averaging over ten such runs. 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 releases one of its already held resources, with a 50% probability. This means that the average holding time of a critical section is 2 rounds. Below are the default values for the simulation parameters, which will be used to generate the plots in the following sections:

# of nodes	# of resources	# of replicas per resource	# of rounds	# of requests per round
6500	65	10	20	20

Table 1: Default values for parameters

A. *Scalability*

One of our initial goals behind the proposed protocols was to come up with an efficient and scalable method for achieving mutual exclusion. Figure 3a presents a comparison of the three protocols. Even though all the plots are linear with respect to number of nodes, the two proposed protocols require 1.5 to 2.5 times less messages than the Sigma protocol, which is a big improvement.

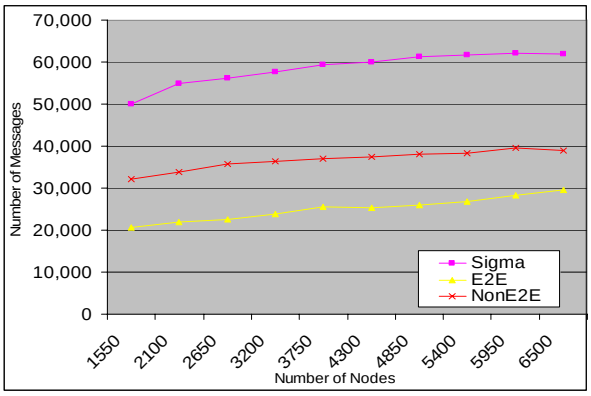

Fig. 3a. A lower increase in bandwidth consumption, as the network grows

Figure 3b shows the effect of increased network contention on the amount of bandwidth consumed. The number of nodes present is kept constant at 6500, but the rate of requests per round is steadily increased from 10 to 100. This is a nice measure as it demonstrates how the protocols would perform under heavy loads.

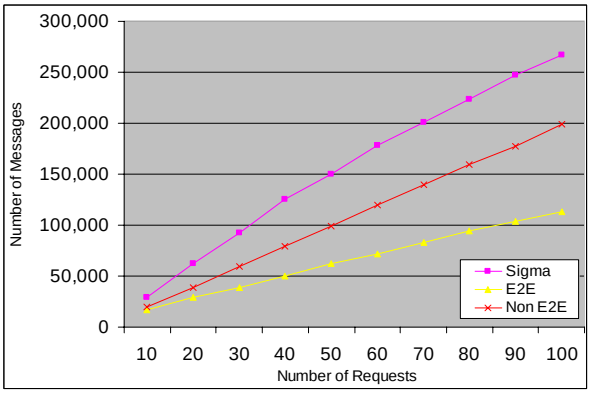

Figure 3b: Effect of increasing the number of requests issued per round on overall bandwidth consumption

B. *Overall Load Distribution*

In order to demonstrate the even load distribution of the proposed protocols, the following plots show the Cumulative Distribution Functions for them. The simulation parameters being used to generate the plots are shown in Table 2. Two CDF plots are generated for each protocol, which differ in the number of replicas that are responsible for granting access to a particular resource. The term load (as used in the plots below) refers to the number of messages that a node has to process.

# of nodes	# of resources	# of replicas per resource	# of rounds	# of requests per round
1000	10	10, 40	20	20

Table 2: Default values for parameters

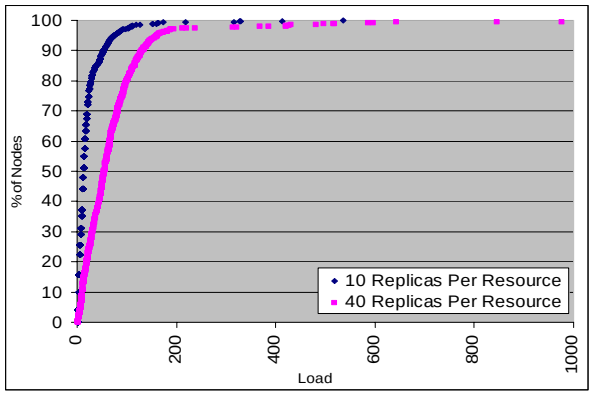

Figure 4a: In E2E, majority of the nodes have a load of 200 messages/round or less, except for a few outliers. By increasing the number of responsible replicas for a resource, the load on each node increases. This behavior is expected since the number of request/response messages exchanged between the replicas and the requesters would now be greater

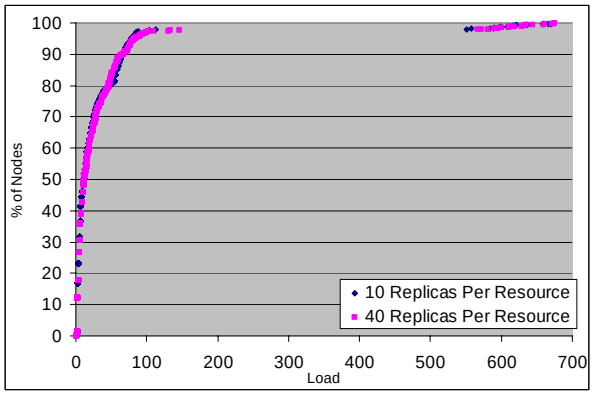

Figure 4b: In Non E2E, majority of the nodes experience a load of less than 180 messages/round. Unlike the E2E CDF plot, this one has a slightly larger number of nodes with loads in the range of 550-680 messages/round. Increasing the number of responsible replicas per resource does not have any effect on the load experienced by each node. This behavior is explained by the fact that in the Non E2E protocol, request messages are intercepted and stopped quicker than E2E (especially when number of replicas per resource is increased)

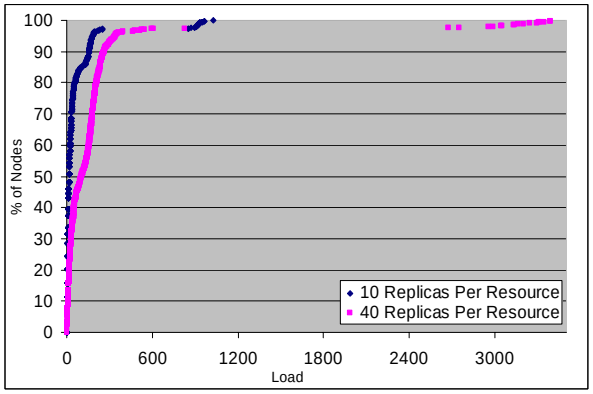

Figure 4c: In Sigma, majority of the nodes experience a load of 700 messages/round or less. Similar to the Non E2E CDF plot, this one has a slightly larger number of nodes with loads in the range of 2500-3500 messages/round. Increasing the number of responsible replicas per resource has similar effect, as was viewed for the E2E Protocol in Figure 4a

C. *High Churn Rate*

Rate of churn is defined as the sum of the number of nodes that are in transition (i.e., are either leaving or joining the system per round). In our simulation, n/2 randomly chosen nodes are failed and n/2 new nodes are added to the system, where n is the desired churn rate. The parameters used to generate the following plots are consistent with Table 1. Requesters are not churned in our experiments.

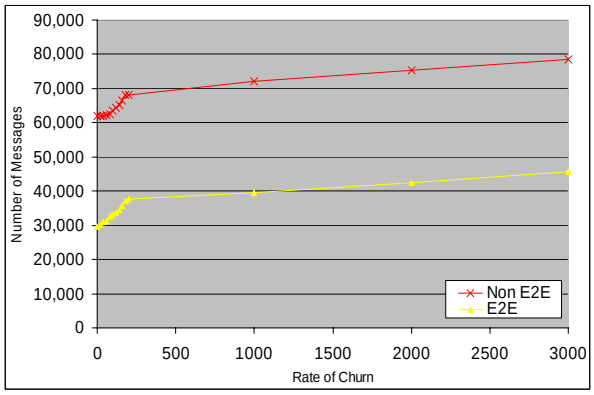

Figure 5: As the rate of churn is increased from 0 to 200, the change in message overhead is shown in this plot. Both the protocols experience a slightly higher increase in message overhead (at first), but then increase at a much lower rate

VII. CONCLUSION

In this work, a couple of protocols for achieving mutual exclusion in a p2p system were presented, one of which conformed more to the End-to-End argument. Both protocols can be easily configured to run on any DHT, therefore providing a truly generalized solution to the addressed problem. The purpose behind presenting two different approaches to the same problem was to present the inherent tradeoff that exists between the amount of bandwidth consumed and better load balancing and fault-tolerance level required. An educated decision, based on specific application needs, must be made in order to fully utilize the potential of the chosen protocol. The proposed protocols, through their truly novel quorum and token-based schemes, were able to keep the load on the replicas relatively low even in the presence of a growing network and high churn rates.

VIII. FUTURE WORK

An important future direction of this research would be to port some existing Grid or p2p applications to use these protocols for handling their mutual exclusion needs. The ideal goal would be to present a general set of APIs for our protocols so that future applications can make use of our algorithms in a modular fashion, without worrying about the low-level mutual exclusion details.

REFERENCES

[1] 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.

[2] Dijkstra, E.W., Solution of a Problem in Concurrent Programming Control, Communications ACM 8, 9 (Sept. 1965), 569.

[3] Foster, I., The Grid: A New Infrastructure for 21st Century Science, Physics Today, 2002.

[4] FreePastry. http://freepastry.rice.edu/FreePastry.

[5] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A., SHARP: An Architecture For Secure Resource Peering, SOSP 2003.

[6] GridPhyN, http://www.griphyn.org.

[7] Gupta, I., Birman, K., Linga, P., Demers, A., and Renesse, R., Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead, IPTPS, 2003.

[8] Iyer, S., Rowstron, A., and Druschel, P., SQUIRREL: A Decentralized, Peer-to-Peer Web Cache, PODC, 2002.

[9] Kazaa. http://www.kazaa.com.

[10] Lamport, L., Time, Clocks and the Ordering of Events in a Distributed System, Communications of the ACM 21, July 1978, 558-565.

[11] Lin, S., Lian, Q., Chen, M., and Zhang, Z., A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems, International Workshop on P2P Systems (IPTPS), 2004.

[12] Lin, S., Lian, Q., Chen, M., and Zhang, Z., A Practical Distributed Mutual Exclusion Protocol in Dynamic P2P Systems, Microsoft Research, Technical Report MSR-TR-2004-13.

[13] Maekawa, M., A √n Algorithm for Mutual Exclusion in Decentralized Systems, ACM Transactions on Computer Systems (TOCS). 1985.

[14] Mislove, A., Post, A., Reis, C., Willmann, P., and Druschel, P., POST: A Secure, Resilient, Cooperative Messaging System, USENIX HotOS IX, 2003.

[15] Naor, M., and Wieder, U., Scalable and Dynamic Quorum Systems, Principles of Distributed Computing (PODC), 2003.

[16] Ngan, T., Wallach, D.S., and Druschel, P., Enforcing Fair Sharing of Peer-to-Peer Resources, IPTPS'03, February 2003.

[17] Raymond, K., A Tree-Based Algorithm for Distributed Mutual Exclusion, ACM Transactions on Computer Systems (TOCS). 1989.7.1.

[18] Raynal, M., A Simple Taxonomy for Distributed Mutual Exclusion Algorithms, ACM SIGOPS Operating Systems Review. 1991.25.2.

[19] Ricart, G., and Agrawala, A.K., An Optimal Algorithm for Mutual Exclusion in Computer Networks, Communications of the ACM. 1981.

[20] Rowstron, A., and Druschel, P., Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, Middleware, 2001.

[21] Rowstron, A., and Druschel, P., Storage Management and Caching in PAST, A Large-Scale, Persistent Peer-to-Peer Storage Utility, ACM Symposium on Operating Systems Principles (SOSP), 2001.

[22] Saltzer, J.H., Reed, D.P., and Clark, D.D., End-to-End Arguments in System Design, ACM Transactions in Computer Systems 2, 4, November, 1984, 277-288.

[23] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., Chord: A Scalable Peer-to-Peer Lookup Service For Internet Applications, Special Interest Group for Data Communications (SIGCOMM), 2001.

[24] Suzuki, I., and Kasami, T., A Distributed Mutual Exclusion Algorithm, ACM Transactions on Computer Systems (TOCS). 1985.3.4.

[25] The Gnutella protocol specification v0.4, Document revision 1.2, www.clip2.com.

[26] The Human Proteome Folding Project, http://www.grid.org/projects/hpf.

[27] Velazquez, M.G., A Survey of Distributed Mutual Exclusion Algorithms, Technical Report CS-93-116, Colorado State University, 1993.