CS556: Distributed Systems
Spring 2011 – Panagiota Fatourou
Student Projects

General Project Deadline: June 17

Theory Projects

1. Distributed Counting
    a. J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *Journal of the Association for Computing Machinery*, 41(5):1020-1048, September 1994.
    b. M. D. Riedel and J. Bruck, "Tolerating Faults in Counting Networks", http://citeseer.ist.psu.edu/102978.html

2. Distributed Directory Protocols
    a. M. Demmer and M. Herlihy, "The arrow distributed Directory Protocol", pp.119-133, DISC 1998.
    b. D. Peleg and E. Reshef, "Low Complexity Variants of the Arrow Distributed Directory", *Journal of Computer and System Sciences*, Vol. 63, Issue 3, November 2001, pp. 474-485.
3. Structured P2P Systems I
    a. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", SIGCOMM'01, August 2731, 2001, San Diego, California, USA.
    b. A. Rowstron and P. Druschel, "*Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.
4. Distributed Storage
    a. A. Muthitacharoen., R. Morris, T.M. Gil, and B. Chen, "Ivy: A read/write Peer-to-Peer File System", Proceedings of 5th Symposium on Operating Systems Design and Implementation, Boston, MA, December 2002.
    b. J. Kubiatowicz, D. Bindel, Y. Chen, S. Cwerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: an architecture for global scale persistent storage", ASPLOS, pp. 190 -201, November 2000.
5. Interval Routing
    a. T. Eilam, S. Moran, and S. Zaks, "A Simple DFS-Based Algorithm for Linear Interval Routing", pp. 37-51, WDAG 1997.
    b. Bakker E. M., van Leeuwen, J., Tan R.B., "Linear Interval Routing", Algorithms Review 2, pp. 45-61, 1991.
6. Scheduling in Networks
    a. K. S. Lui and S. Zaks, "Scheduling in Synchronous networks and the greedy algorithm", pp. 66-80, WDAG 1997.
    b. T/L/ Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computer systems", Readings in Distributed Systems, T. Casavant and M. Singhal (eds.), IEEE Computer Society Press, 1994, pp. 31-51.

7. Replication
    a. Rivka Ladin, Barbara Liskov, Liuba Shrira: Lazy Replication: Exploiting the Semantics of Distributed Services. PODC 1990: 43-57
    b. Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, Michael Williams: Replication in the Harp File System. SOSP 1991: 226-238
8. Clock Synchronization
    a. T. Srikanth and S. Toueg, "Optimal Clock Synchronization", J. of the ACM, 34(3): 626-645, 1987.
    b. J. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization", Information and Computation, 77(1): 1-36, 1988.
9. Byzantine quorum systems
    a. D. Malkhi and M.K. Reiter, "Byzantine Querum Systems", Distibuted Computing, 11:203-213, 1998.
    b. D. Malkhi, M.K. Reiter, and A. Wool, "The load and availability of Byzantine quorum systems", ACM Symposium on Principles of Distributed Computing, p. 249-257, 1997.
10. Replicated DataBases
    a. A. El Abbadi and S. Dani, "A dynamic accessibility protocol for replicated data bases", Data and Knowledge Engineering, 6:319-332, 1991.
    b. E. El Abbadi and S. Toueg, "Maintaining Availability in partitioned replicated data bases", ACM Transactions on Database Systems, 14(2): 264-290, 1989.
11. Replicated Data
    a. D. K. Gifford, "Weighted Voting for Replicated Data", 7th Symposium on Operating Systems Principles, pp. 150-159, December 1979.
    b. S. Jajodia and D. Mutcler, "Dyanamic Voting", ACM SIGMOD Int. Conference on Management of Data, pp. 227-238, May 1987.
12. Dynamic Networks
    a. B. Awerbuch, I. Cidon, and S. Kutten, "Optimal maintenance of replicated information", 31st IEEE Symposium on Foundations of Computer Science, pp. 492-502, October 1990.
    b. S. Kutten and A. Porat, "Maintenance of a Spanning Tree in Dynamic Networks", 13th Symposium on Distributed Computing (DISC), pp. 342-355, 1999.
13. Self-stabilization
    a. L. Higham and Z. Liang, "Self-Stabilizing Minimum Spanning Tree Construction on Message-Passing Networks", 15th Symposium on Distributed Computing (DISC), pp. 194-208, 2001.
    b. T. Herman and T. Masuzawa, "Stabilizing Replicated Search Trees", 15th Symposium on Distributed Computing (DISC), pp. 315-329, 2001.
14. P2P Systems II
    a. J. Aspnes and G. Shah, "Skip Graphs", ACM Transactions on Algorithms, Vol. 3, No 4, November 2007.
    b. Gabarro, J., Martınez, C., and Messeguer, X. 1996. A Top-Down Design of a Parallel Dictionary using Skip Lists. Theoretical Computer Science 158, 1–2 (May), 1–33.
15. P2P Systems III

a. Castro, M., Druschel, P., Hu, Y. C., and Rowstron, A. 2002. Topology-aware routing in structured peer-to-peer overlay networks. In Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo), Schiper, A. Shvartsman, H. Weatherspoon, and B. Zhao, Eds. Lecture Notes in Computer Science, vol. 2584. Bertinoro, Italy, 103–107.

b. Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. 2001. A Scalable Content-Addressable Network. In Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM). San Diego, CA, USA, 161–172.

Paper Understanding - Reports

Each student should deeply understand the material presented in the papers s/he has undertaken. Most specifically, each student should:

- know the algorithms and the techniques presented in the paper;
- be able to answer to questions of the style «Why is each line of the code useful in the algorithms s/he will present and what could go wrong if any line was removed»;
- invest time on the algorithm s/he studies, devise his/her own bad scenarios of execution and understand how the algorithms cope with these scenarios;
- study/devise a big number of examples to deeply understand how the algorithms work; (it is these examples that give a concrete idea of how deeply the student has understood the technical part of the paper);
- invest some time to understand the high level idea of the analysis of the algorithms included in the paper;

Each student should provide an intuitive description of the algorithms, their correctness and their complexity in his/her report.

Reports

Your report should not exceed 12 full pages. Extra pages will be evaluated negatively. Your report must include the most important, according to your opinion, results of the papers you study. Also, you may show several examples to demonstrate in-depth understanding of your subject.

Your report should have approximately the following structure:

1. Introduction (approximately 1-2 pages): Abstract description of the problem and its significance. Abstract description of the results presented in later sections. A paragraph on the organization of the rest of the material in the report.
2. Model (approximately 2 pages): Presentation of the model (the system parameters, definitions needed to describe and analyze the algorithm, etc.).
3. Technical part of the paper (approximately 8 pages): In this section, we describe the algorithms, their correctness and their analysis in accordance with the instructions given above. In cases a paper presents a series of results, and there is not enough room for all of them, the student should choose which of them to include in the paper.
4. Epilogue: A summary and a description of open problems (if any) (at most 1 page).

Your report will be graded based on the following criteria:

1. how much it convinces that you understand the results presented in the studied papers in depth,
2. whether it is well-structured,
3. selected results you have decided to include (for instance, these must no be only the easier results presented in the papers)
4. how much it differentiates from the original papers,
5. how many examples you have studied (some of them should be of your own), and how well you explain the algorithms, techniques and proofs you have included,
6. how well written it is (the degree of formalism, how correct are the material you have included in it)
7. the extent to which all of these objectives are achieved without having circumvented the limit of 14 pages.

We do not present the results of the two papers in order. We rather try to discover similarities and differences between those results and structure the report so that related results are presented and discussed together. So, the final report is more kind of a survey where in addition to the presentation of the results, the different algorithms are compared and contrast. The structure of the report is very important.

Ideas for Programming Projects

You can choose any of the ideas presented below or you should suggest your own project. In either case, you should submit a project proposal of 2-3 pages providing more details on which project you are going to implement and describing the difficulties that you are going to face and solve in your project.

Project proposal due: May 16, 2011
Final Project due: June 17, 2011

1. Distributed Social Network

Implementation of some kind of a distributed social network application. Quoting from Wikipedia: “A social network is a social structure made up of individuals (or organizations) called "nodes", which are tied (connected) by one or more specific types of interdependency, such as friendship, kinship, common interest, financial exchange, dislike, or relationships of beliefs, knowledge or prestige”. All these are parameters specified by each node.

The general idea is the following. Each node must be able to join the social network and exit from the social network. During its presence to the network each node can specify or modify various parameters. The distributed system must be able to correlate nodes with common parameters and form “groups of relevance”. Exiting at any arbitrary time from the social network may cause a lot of problems and have several implications, according to the services you will choose to implement and the algorithms you selected to implement them. These problems or implications must be clearly identified and their solutions must be described.

You can choose the functionality of your distributed social network so that it takes into consideration the constraints of the project described below.

2. Library of Collaborating Tools

Implementation of a library of collaborating tools, such as:

- Distributed Calendar (like the google calendar but probably with enhanced functionality)
- Distributed Meeting Arrangement (like doodle but with enhanced functionality)
- Distributed Meeting and Sharing (like facebook). The nodes can chat with each other, form chat-rooms or chat-groups, send messages to multiple recipients, etc. When chatting with each other, it is required the messages to be displayed in the same order in both participants. The same is required when talking to chat-rooms or chat-groups.
- Distributed Editor (like recent version of emacs). Users can create and edit documents, which can be concurrently edited by several other users.

Choose **some** of the above items so that the result should be a comprehensive project.

3. Middleware

Implementation of middleware that provides **some** of the following functionality:

- Naming service. Obviously, each node has to define a “name” when it enters the network. This service requires implementing a mechanism that permits to some node to find the information of any other node by searching its “name” and this must be implemented in a distributed way.
- Group creation, node joining and exiting, and communication. Maintaining the groups of relevance in a distributed way, when nodes can join and exit the network arbitrarily, is challenging.
- Security service: The security service may include the following: (1) authentication of principals, (2) access control on the reception of remote method invocations, (3) security of communication between clients and servers, (4) facilities for non-repudiation, etc.
- Trading service: Allows the location of objects by their attributes, i.e., it is kind of a directory service. Its database contains a mapping from service types and their associated attributes onto remote memory references of objects. Clients make queries specifying the type of service required, together with other arguments specifying constraints on the values of attributes, and preferences for the order in which to receive matching offers.
- Transaction and concurrency control services.
- Persistent state service.
- Any other functionality you would like to suggest.

4. Publish-Subscribe System

Implementation of a publish-subscribe system. Some node wants to publicly share a piece of information (e.g., to announce an event) with any other member included in the groups of relevance it participates. Implement algorithms that efficiently diffuses the information (e.g., efficient broadcasting, or mobile agents, or gossiping protocols)

5. **Distributed Hash Tables**

   Implementation of a P2P system, structured or unstructured.

6. **Distributed Data Structures**

   Implementation of a library of distributed data structures for distributed memory machines.

7. **Consistency and Replication**

   Implementation of a quorum system where quorums are appropriately formed to guarantee consistency.

   A Revision Control System. It is used when a team of people participating to the same project want to concurrently update the same files. Changes may be identified by a number or letter code, termed the "revision". Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged. Revisions can be made either to independent files or to the whole project.

8. **Distributed Memory Allocator or Garbage Collection**

   A distributed memory allocator is a distributed algorithm that efficiently allocates memory requested by different nodes.

   The garbage collector attempts to reclaim memory occupied by objects that are no longer in use by the program. Distributed Garbage Collection, is a particular case of Garbage Collection, where references to some object can be held by remote clients. Mechanisms, e.g. leases, exist so that an object can be identified either as useful or as garbage.

You can choose to implement any of the projects described above and you are encouraged to propose and implement any other service you want (which is related to the topics of this course, i.e. it incorporates any class of distributed algorithms discussed during the lectures or the student's presentations). Your choice should be so that **at least** three of the following list of algorithms should be implemented in the project:

1. Broadcast and/or multicast through spanning tree construction
2. Mobile Agents
3. Gossiping Protocols
4. Mutual Exclusion
5. Leader Election
6. Distributed Snapshots
7. Timestamps
8. Resource allocation algorithms

9. Concurrency control
10. Resource discovery
11. Agreement protocols
12. Replication and consistency protocols
13. Distributed shared memory protocols

The implementation of your project should take into account issues like fault tolerance, performance, scalability, and other challenges that we have discussed in class.