

A Peer-to-peer System Assignment for Teaching Distributed Systems Concepts

Shivakant Mishra

Department of Computer Science

University of Colorado, Campus Box 0430

Boulder, CO 80309-0430, USA.

<http://www.cs.colorado.edu/~mishras>

Abstract *This paper describes a peer-to-peer system design and implementation assignment for a distributed systems course at an advanced undergraduate level, or a first-year graduate level. The assignment consists of four stages, and provides students a good, practical understanding of some of the key distributed computing concepts. These include issues of decentralization, managing distributed information, communication middlewares, heterogeneity, distributed fault tolerance and security, and performance evaluation. The assignment is easy to explain to the students, makes them understand the inherent limitations of a distributed system, and provides hands-on experience with some of the latest distributed computing technologies.*

Keywords: Distributed systems education, programming assignment, hands-on experience, peer-to-peer systems.

1 Introduction

An important challenge in teaching a distributed systems course at an advanced undergraduate level or a first-year graduate level is to design appropriate programming assignments that can provide a use-

ful, hands-on experience in a variety of distributed system technologies, and can be completed within a single semester. The key problem is how to design assignments that can provide students an insight into the complexities of distributed computing, as well as a hands-on experience with using the latest technologies.

One model for teaching distributed systems that is commonly followed consists of a number of different assignments. These assignments are typically independent of one another, and address different concepts of distributed computing. For example, a typical distributed system course may consist of three assignments: (1) design and implementation of a simple distributed application based on the client-server technology, (2) design and implementation of a distributed application using some well-known middleware, such as CORBA, DCOM, Java RMI, or Sun RPC, and (3) design and implementation of a relatively large distributed system such as a distributed file system, distributed shared memory system, or a group communication system. While the first two assignments provide students basic, distributed programming skills, the last assignment provides students a hands-on experience with addressing the com-

plex issues of fault tolerance, security, distributed storage, memory coherency, etc.

We used this model to teach distributed systems course in the past. We taught this course five times in the period from 1994 to 2000 using this model. In our experience, there are several problems associated with this model. First, there is no smooth transition from one assignment to the next. For example, students are generally not able to appreciate the usefulness of a newer technology, or relate the concepts learned in the previous assignments to the concepts being addressed in the following assignments. Second, because each assignment is independent, students have to typically repeat several, similar design and implementation techniques in all assignments. This repetition unnecessarily consumes useful time, and does not contribute much in the way of learning. Finally, this model does not provide students any experience with building a large distributed system. While the third assignment is intended for this purpose, it has to be typically squeezed in the last few weeks of the semester. This short period of time is generally not enough to convey any useful experience with building a large distributed system.

In this paper, we describe another model for structuring the assignments in a distributed systems course. We have followed this model in teaching the Distributed Systems course at the University of Colorado, Boulder in the Spring 2002 semester. The course consists of a single assignment that is divided into four stages. The goal of this programming assignment is to design, implement, and evaluate a peer-to-peer look up service for a distributed system. Peer-to-peer systems are distributed systems without any centralized control or hierarchical organization. Software running at each node is equivalent in functionality. These systems provide a basis for constructing a

large number of distributed (particularly Internet-based) applications. These include redundant storage, chat services, file and data sharing, and so on. Examples of peer-to-peer systems include [1, 2, 5, 3, 4].

2 Assignment Description

The peer-to-peer assignment is divided into four stages. The goal is to explore a different set of distributed computing concepts in each stage, provide a smooth transition from one stage to the next, and avoid any need for repetitions of design or implementation techniques.

2.1 Stage 1

The first stage of this assignment consists of implementing a simple peer-to-peer system that provides support for storing a large set of $\langle \text{Key}, \text{Value} \rangle$ pairs among a *group* of nodes. The system exports two operations to the clients, `Find()` and `Store()`. The `Find` operation searches for the value corresponding to a given key, and the `Store` operation stores a $\langle \text{key}, \text{value} \rangle$ pair in the system. Each group member is assumed to have knowledge of only a few other group members (less than 10%). Communication between different group members is done using an appropriate transport-level protocol, such as UDP or TCP.

There are several simplifying assumption that are made at this stage. For example, the membership of the group is assumed to be static, and there are no failure or security issues to be addressed. These assumptions are removed one by one as the assignment moves to the following stages.

The main goal of this stage of the assignment is to understand the inherent limitations of a distributed system, namely no

shared memory or common global clock. The focus is on algorithm design for efficient storage and retrieval of data in a distributed environment in which there is no centralized coordinator, and each node has only some partial (, and perhaps stale and inconsistent) information about the other nodes storing parts of the data set.

2.2 Stage 2

In the second stage, the prototype constructed in the first stage needs to be extended to include dynamic group membership in which new nodes may join the group at any time, and existing group members may leave the group either voluntarily, or by failing. The issues that need to be addressed in this stage include how store and search algorithms are affected by node joins and node failure/leave, and how node joins and node failure/leave is facilitated. This requires implementing failure detection mechanisms.

Another important issue addressed in this stage is robustness. The data stored at a node that fails is lost. So, in order to avoid the loss of data items due to node failures, the data items need to be stored at multiple locations. This allows students to experiment with different replication techniques, and techniques for replica distribution over a set of nodes. Finally, one requirement of the assignment at this stage is to measure the performance of the system. In particular, the performance index measured is average response time for the `Find` operation. The key concept that students learn here is what constitutes “average performance” in a distributed computation.

2.3 Stage 3

The goal of the third stage of the assignment is to have students experiment

with different types of communication middlewares. In this stage, students are required to use an appropriate communication middleware for facilitating communication and synchronization between different group members. Typical communication middlewares that students experiment with at this stage include CORBA, DCOM, Java RMI, and Sun RPC.

After implementing the prototype using an appropriate middleware, students measure the performance and compare this performance with the performance measured in the previous stage. This gives students insight into the performance issues of communication middleware.

2.4 Stage 4

Finally, in the fourth stage, the peer-to-peer system prototype implemented so far is extended to provide support for security. The security concepts to be implemented in this stage include privacy, integrity, and authentication. Students are required to implement mechanisms for ensuring the privacy and integrity of data items as they are transmitted over the network. In addition, only authenticated nodes can join a group and invoke `Find()` or `Store()` operations. While implementing these mechanisms, students use security tools such as Kerberos, triple-DES, and key management protocols in large groups.

3 Experience

We taught a distributed systems course using this assignment at the University of Colorado at Boulder in Spring 2002. Students in this course ranged from first-year M.S. students to Ph.D. candidates. Experience from these course has been very positive and we intend to repeat this assignment in

the future semesters.

Students really liked this assignment. In stage one, they came up with some very innovative techniques for distributing the $\langle key, value \rangle$ pairs over a set of 50 nodes to improve the performance of both `Find()` and `Store()` operations. There were about ten different techniques attempted by students here. More importantly, students were able to really appreciate the complexities of distributed computing. They learnt first-hand the issues that arise due to distribution and partial information.

The second stage was a useful lesson for students to address the issues of the dynamic nature of distributed services and possibility of partial failures. They were able to devise interesting techniques to deal with node failures, join, and leave. They experimented with different replication techniques that were covered in class.

The third stage of the assignment dealt with using different types of middlewares. The middlewares students used were CORBA, DCOM, and Java RMI. The most important lesson students learnt from this stage was that there is some performance price that applications have to pay in order to use these middlewares. Finally, the fourth stage allowed students to learn about some simple security mechanisms.

We plan to make several changes in this assignment based on this experience. First, we will move the security aspects of the assignment (currently stage 4) into earlier stages. One problem we faced was it was difficult to incorporate appropriate security mechanisms in the prototype at a late stage. The second change we plan to make is to extend the security issues to include support for intrusion tolerance. This will provide students with insights into the relation between techniques used for security, fault tolerance, and intrusion tolerance.

4 Conclusion

This paper describes a peer-to-peer system design and implementation assignment for a distributed systems course at an advanced undergraduate level, or a first-year graduate level. The assignment is designed to avoid any repetitions, provide hands-on experience with some of the popular distributed computing technologies, and covers some of the key distributed computing concepts. These include issues of decentralization, managing distributed information, communication middlewares, heterogeneity, distributed fault tolerance and security, and performance evaluation. Experience from teaching a distributed system course based on this assignment has been very positive.

References

- [1] Gnutella website.
URL: <http://gnutella.wego.com>.
- [2] Napster website.
URL: <http://www.napster.com>.
- [3] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the Symposium on Operating Systems Principles*, October 2001.
- [4] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage facility. In *Proceedings of the Symposium on Operating Systems Principles*, October 2001.
- [5] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM'01*, August 2001.