The Structure of a Project-Based Course on the Fundamentals of Distributed Computing

Prasun Dewan
Department of Computer Science
University of North Carolina
Chapel Hill, USA
dewan@cs.unc.edu

***Abstract*—We have developed a novel structure for a course on distributed computing suitable for juniors, seniors and graduate students that covers (a) use, design and implementation of state of the art IPC mechanisms, and (b) implementation and experimentation with state of the art consistency algorithms.**

Keywords- education, remote procedure call, Paxos, atomic broadcast, two-phase commit, non-blocking I/O, performance.

I. MOTIVATION AND PROBLEM

The motivation for teaching distributed computing is relatively straightforward. It is replete with concepts that are difficult to self-learn and are the foundation for both traditional fields such as distributed database, operating, and simulation systems, and relatively new areas such as distributed collaboration and machine learning, mobile computing, sensor networks, and IOT (internet of things). Yet, unlike related subjects such as programming languages, compilers, and algorithms, there is no generally accepted approach to teaching the fundamentals of this subject to undergraduate students. We describe the structure of a course we have developed to address this limitation.

II. SCOPE: CONSISTENCY AND IPC ABSTRACTIONS

The trickiest issue we faced arose from the fact that almost every area of non-distributed computing extends into distributed computing – it is difficult to imagine any computation that could not benefit from distribution. Which of these areas should one cover in a single course? Our answer was to focus on areas in non-distributed computing considered most fundamental – programming abstractions and algorithms.

Arguably, a deep understanding of programming abstractions requires knowledge of how they are designed and implemented. Thus, we covered use, design, and implementation of abstractions for distributed, that is, inter-process, communication (IPC).

Given that IPC is layered above network protocols, and any distributed application is also a concurrent application, we required, as prerequisites, courses on networking *or* operating systems, making this a course for juniors, seniors, and graduate students.

III. ALGORITHMS: CONSISTENCY/CONSENSUS

The course focused on algorithms for consistency and consensus rather than speed up, as the latter fall more in the domain of parallel computing. These included the (log—based) Two Generals' Problem, non-atomic and atomic broadcast, two-phase commit, and Paxos. Log-based algorithms were contrasted with state-based consistency mechanisms, and the applications of both kinds were given in replicating data centers (such as those provided by Amazon and Oracle), data sharing systems (such as Google Drive and Dropbox), and nested transactions.

IV. ABSTRACTIONS STUDY: IPC DESIGN SPACE

The course developed an IPC design space with several dimensions including blocking vs non-blocking, synchronous vs asynchronous, guarded vs non-guarded, serialization vs byte transfer, and remote assignment vs procedure call. Influential IPC mechanisms were placed in this space including Unix signals, pipes and sockets, Ada Rendezvous, and CSP Remote I/O.

V. ABSTRACTION USE: JAVA IPC

Arguably, it is important for all students, and especially undergraduates, to gets hands-on experience with concrete implementations to understand abstract concepts. This, in turn, raises the issue of the language used for implementing assignments. We chose Java for two reasons. First, it is statically typed, particularly important for distributed programs, which are relatively small but difficult to debug. Second, it provides four different, layered, state of the art IPC abstractions – non-blocking byte communication, blocking byte communication, serialized object communication, and remote procedure call (RPC). Our assignments involved the use of three of these layers – they ignored blocking byte communication, typically covered by networking courses that teach sockets.

VI. ABSTRACTION IMPLEMENTATION: GIPC

The use of Java allowed us to meet to our goal of developing assignments involving the use of state of the art IPC abstractions. Java, however, does not provide an easy way to substitute or customize layers. For instance, it is not possible to make its RPC layer – called RMI –use non-blocking I/O for byte communication or to change the

Funded in part by NSF grant OAC 1829752

algorithm used for object-graph serialization. To overcome this limitation, we created our own implementation of a Java IPC stack in a system called GIPC (Generalized/Group IPC), which is in the spirit of creating educational operating systems such as Xinu and MINIX.

VII. LAYERED ASSIGNMENTS → PROJECT

Our programming assignments were layered to form a semester-wide project, allowing students to better understand the relationship among and motivation for the concepts implemented. Successive assignments required students to implement/use additional IPC abstractions and consistency/consensus algorithms. Later assignments implicitly tested almost all aspects of previous assignments.

Students started with a single-process interactive simulation of Halloween trick and treat (Fig. 1), implemented by a student (Beau Anderson) as part a CS-2 course taught by the author. The first assignment required them to implement both non-atomic and atomic broadcast using Java NIO, and replicate the simulation using both of these algorithms. Which consistency algorithm was used was determined in each replica by a dynamic parameter, *consistency-choice*.

The next assignment required them to add a second mechanism for communication – Java RMI. Thus, the project now supported in each simulation a second dynamic parameter, *IPC-choice*, which determined if Java NIO or RMI was used for atomic and non-atomic broadcast. In addition, students were required to support replication of changes to two aspects of the meta-state: *consistency-choice* and *IPC-choice*. Thus, consistency-choice was recursively made consistent using a consistency algorithm!

The third assignment required them to add (a) GIPC to the set of supported IPC mechanisms, and (b) two-phase commit to the set of consistency mechanisms. With two-phase commit, a client could veto execution of a simulation command or changes to the IPC-choice and consistency mechanism.

Three subsequent assignments involved changing aspects of GIPC using the Factory design pattern. Serialization of physical structures controlled by inheritance-based class serializers was changed to serialization of logical structures controlled by delegation-based interface serializers. An explicit receive was added to the supported implicit, notification-based receive. Asynchronous procedure invocation was changed to synchronous invocation using the explicit receive. Serialization was made as extra credit assignment.

For the last assignment, a GIPC implementation of Paxos was provided to the students – they used it to expand the supported consistency mechanisms and create and understand corner cases illustrating the nature of and need for the three Paxos phases.

Thus, their final project used their own implementations of two-phase commit, atomic broadcast, and non-atomic broadcast together with the GIPC Paxos implementation to replicate regular and meta-state. Moreover, the GIPC-based algorithm implementations, in turn, used the students' implementation of remote procedure call, explicit receive, and serialization. The result was a comprehensive project of which, to the best of our knowledge, students were proud.

VIII. ASSIGNMENT DEMONSTRATION

As live demonstrations take a significant time to set up and run, for each assignment, students created videos explaining their implementations, demonstrating cases that created and prevented various forms of inconsistency, tracing the steps of their algorithms through the debugger and logs, and showing their performance results. As the simulation was interactive, inconsistencies were visualized (Fig 1.).

Performance was usually compared by running all processes on a single computer. The third assignment required them to also use separate virtual computers. Several Cloud infrastructure could be used to obtain virtual computers. We used NSF's CyVerse as we are currently funded by NSF to study and improve the training mechanisms supported by it. By using CyVerse, we exposed students to an important application of the studied distributed-computing concepts.

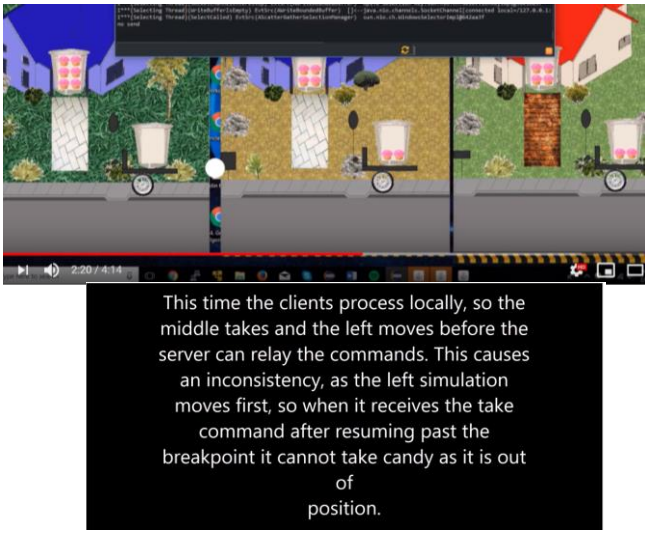

Figure 1. Snapshots from the video submission of a student (Andrew Huang) showing an inconsistency and its cause.

Assignment descriptions, and videos of the lectures along with PPTs from which they were created, were available from http://www.cs.unc.edu/~dewan/533/current/index.html The published material included background information on both concurrency and networking as few students satisfied both prerequisites It allowed the course to be flipped, focusing, during class time, on reviewing the material and answering open-book quiz questions collaboratively.

IX. SUMMARY AND FUTURE WORK

The course covered a wide range of fundamental topics in algorithms and abstractions. A carefully crafted project connected these concepts together. Students were provided educational scaffolding code in the form a customizable IPC-stack and a Paxos implementation. We are currently developing automatic checks for grading the assignments. The author can be contacted for more information.