Lecture 11: Shared Memory and Registers

Aurojit Panda

1 Why are we looking at this?

Thus far we have focused on processes that communicate by sending messages over an asynchronous network. It might appear strange that this last lecture looks at a very different model where processes communicate by writing to shared registers, and might lead you to ask why are we looking at this model so late in the semester. There are a few reasons for looking at this model today:

- First, it is a model used when describing several protocols. Some of this is because as the Attiya paper mentions some problems are easier when you can access values written by a process despite process failure.
- Second, and more practically, we are increasingly seeing systems where processes communicate using key-value stores or other storage systems.
  - Provide similar guarantees to registers, but also assume additional infrastructure in the form of these key-value stores.
  - Useful to understand the tradeoff offered by such designs.
- Third, this topic gives us an opportunity to talk about partial network connectivity, and maybe think a bit about how it impacts protocols.

2 Complete networks vs arbitrary networks

We are going to discuss this first since it is unrelated to most of the material we are going to be covering. Thus far in the semester we have assumed a fair, asynchronous network that connects all processes. As we have discussed this is a pretty weak set of assumptions, but it does mean that

- If process A and process B are both alive (i.e., neither has failed), and process A sends an infinite number of messages to B, then B must receive an infinite number of messages.

- More strongly, if processes A, B and C are all alive and process A sends an infinite number of messages to both B and C, then both B and C must receive an infinite number of messages.

The network we have been considering is thus "symmetric": all live processes can communicate with each other. Unfortunately, this is not always the case in real life, because the network itself is a distributed system that can exhibit weird pathologies. For instance, CloudFlare (`https://blog.cloudflare.com/a-byzantine-failure-in-the-real-world/`) had one of these less than a year ago. They are surprisingly common.

One obvious question is do they have an impact on the protocols we have looked at thus far? The unfortunate answer is yes, they do impact the availability (i.e., performance) of some of our protocols. You might want to consider how.

3 Shared Memory Registers

There is a very large family of registers out there that allow different numbers of readers and writers (single reader single write, multiple reader single writer, and multipler reader and multiple writer are the most common), and provide a variety of different consistency guarantees (atomicity, linearizability, causal consistency, eventual consistency, etc.). Following the Attiya paper we will focus on atomic single writer multi-reader register. Fortunately, it is easy to convert these to multi-reader multi-writer registers.

4 Constructing an Atomic Multi-Reader Single-Writer Register

See Figure 2 of the Attiya paper. Figure 1 (`communicate`) is really the same as reliable message delivery from lab 1.

5 Unbounded Messages and Achieving boundedness

he bounded emulator essentially tries to bound the number of labels alive in the system at a time, and use a function LABEL to generate new labels when necessary. The main challenge in doing so lies in determining when to "garbage collect" an existing label, i.e., when to decide that it is safe to

no longer consider the label.The paper does this by collecting labels from a majority, Lemma 5.3 shows this is sufficient.

A similar trick is used in practice in many protocols (e.g., in sliding window protocols) , though usually we impose requirements on the number of outstanding in-flight messages, which is harder in this context.

6 Dealing with Arbitrary Networks

Uses a technique similar to anti-entropy: get processes to forward messages to get around partial connectivity.

7 Using Failure Detectors to Characterize the gap

Easy to see that given registers we can emulate message passing (just use 2^n registers, one for each pair of processes, as a way to pass messages). Have seen that given message passing we can emulate registers, though seems to require an additional assumption that a majority of processes are alive and potentially requires more messages.

Last class we looked at failure detectors and how they allowed us to characterize the requirements for a protocol. The Delporte-Gallet paper asks what is the failure detector one needs for registers, and finds that what you need is really a quorum, i.e., this is the minimal requirement that we have. Compare this to what we found out last class for consensus.