CS514: Intermediate Course in Computer Systems

Lecture 26: March 26, 2003

Data replication is IMPOSSIBLE!

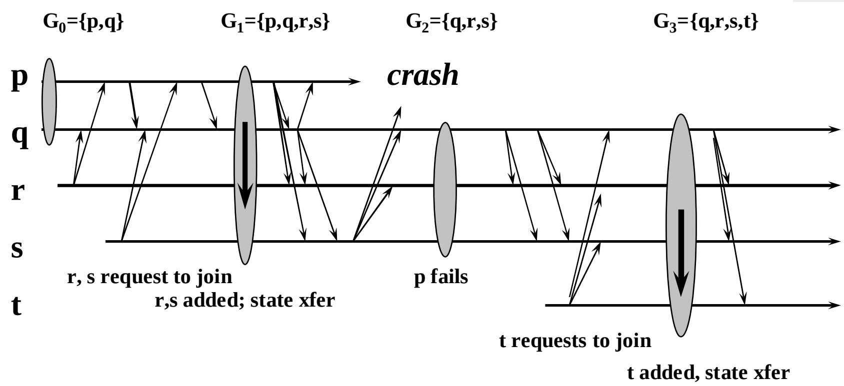
On Monday we looked at data replication….

CS514

- We discussed the virtual synchrony model
- And we explored ways of implementing it
  - A failure detection service
  - Multicast protocols, with varied ordering properties
  - Group membership, state transfer…
  - Other tools in a "toolkit"

Virtual Synchrony Model

CS514



... to date, the *only* widely adopted model for consistency and fault-tolerance in highly available networked applications

Some issues we ran into

CS514

- Scalability and performance
- Lack of vendor enthusiasm
  - More properly, vendors prefer not to expose the interfaces
  - These techniques are widely used in products of many kinds
- Complexity of user programming model (multiple forms of ordering, reliability guarantees, etc). Vsync model vs Paxos
  - Still simpler than working “from bits up”

But in fact DATA REPLICATION IS IMPOSSIBLE!

CS514

- Famous result from a paper in 1982
  - Fischer, Lynch and Patterson
  - Impossibility of Distributed Consensus with One Faulty Process
- How can we solve a problem and yet also prove that it is impossible to solve?

It depends on what you mean by impossible

CS514

- No, this isn't a Bill Clinton quote!
- Some options: "Impossible" means:
  - The problem can never be solved, not even once, ever.
  - Algorithms for solving the problem can be safe but not live.
  - The problem is just "damn hard" to solve

FLP result is of the second kind

CS514

- They define a specific runtime model
- In this specific model they offer a specific problem definition
- Then they show that any solution to the problem, in that model, is safe only if it isn't live

Safe? Live?

CS514

- Safe means "correct". A safe solution is one that only does the right thing
- Live means "always terminates". A live solution never goes into infinite loops or gets stuck in some way without solving the problem

Safe but not live?

CS514

- This means that if you give me a solution to the problem,
- … then I can show you a scenario
- … in which your solution thinks forever and never actually solves the problem

The FLP model

CS514

- They focus on *asynchronous systems*
- Such a system
  - Has processes that only fail by halting
  - Has a reliable network that never loses a message
  - But has no form of time whatsoever
  - Processes don't even run at the same the speed. Speed is not "meaningful"

Why do they use such a strange model?

CS514

- In fact there are two extreme models which the theory community use heavily
  - The asynchronous model
  - The synchronous model
    - All computation is in fixed-length rounds
    - Messages are delivered during the round. Clocks are perfectly synchronized…
- Each model represents an extreme

How to use a model

CS514

- The asynchronous model is strangely weak.
  - If something is possible, it will also be possible in the real world
  - But if something is impossible in that model, it may be possible in the real world. But if so, it will be possible because of something the real world "adds"
- The synchronous model is strangely strong
  - If something is impossible, it will also be impossible in the real world
  - But if something is possible, we may not be able to use it in the real world!

But moving on…

CS514

- The problem they look at is *consensus*, not replication
  - They suggest that consensus is needed in most useful systems
  - This seems to be true…

Consensus problem

CS514

- Given a set of *N* processes $\{p_0, \dots p_n\}$
  - Each has an input b_i, either 0 or 1
  - Could be all 0's, all 1's or a mix
- Job of our system is to pick a single value and "decide" on it
  - It needs to be a valid input someone really received
  - E.g. could be the majority value

… despite one faulty process

CS514

- And our algorithm must do this both if everyone is healthy, and if just a single process fails by halting

Why should this be hard?

CS514

- First, any process that decides on a value must pick the same value that everyone else will pick.
  - Even if a process fails, once it has decided, the system is “committed”
- And of course we don’t have a way to detect failures
  - In an asynchronous system a failure just looks like a very slow process

How does the proof work?

CS514

- Very counterintuitive. In fact the paper is extremely hard to read
- Basic idea is this
  - Imagine a case where half the processes get 0 and half get 1
  - Either value is fine… but we need to make sure everyone agrees on the value

How does the proof work?

CS514

- Initially the system is "bivalent"
  - It could decide on either 0 or on 1
- Later the system will be univalent
  - It will become inevitable that we all pick, say, 0
- Assume you have an algorithm for accomplishing this decision

How does the proof work?

CS514

- We set your algorithm up and let it run
- Look at the step when it is about to switch from bivalent to univalent
  - This step will be triggered by delivery of some sort of message or by some form of “timeout”
  - The model lacks timeouts but it does have a kind of “arbitrary action” that can mimic a timeout

How does the proof work?

CS514

- So we identify some step
  - Process p_i will decide if it receives input *m*
- Now we delay this step and let the rest of the system proceed

How does the proof work

CS514

- Recall that the system is still in a bivalent state. By definition it could still pick either 0 or 1
- Suppose that p_i was poised to pick 0
- Look at a run where the rest of the system gets ready to pick 1
- Now let p_i continue

Recall that your algorithm was correct

CS514

- Apparently, at this point p_i won't pick 0 after all!
  - If it does, it is inconsistent with everyone else and hence your algorithm wasn't correct after all
- So p_i has a bit of extra work to do…

They construct an infinite loop this way

CS514

- In practice, they are basically saying that
  - An infinitely smart adversary (say, Elizabeth Hurley)
  - … given total control over when messages are delivered
  - … could trick a system into endlessly changing its mind
- The system thinks forever and never decides on a value

Why does this establish impossibility?

CS514

- Recall that impossibility means different things to different kinds of people
  - For FLP, it means that the problem *can't always be solved*
- They are showing that any correct algorithm has at least one scenario where it loops forever

But how likely is such a scenario?

CS514

- They don't look at probabilities!
  - For them, an algorithm that always works "in practice" but still has one theoretically plausible scenario where it hangs still establishes impossibility
  - Perhaps, for your purposes, "very very unlikely" is good enough?

Back to virtual synchrony

CS514

- Virtual synchrony is subject to the FLP result!
  - There are situations under which it will be unable to make progress
  - Nonetheless, this confused people for many years…
- Paxos suffers the same issue
  - In fact Paxos is "less" able to guarantee progress than virtual synchrony

The Achilles Heel of Vsync?

CS514

- The problem is in failure detection
  - This can make mistakes
  - So it could mistakenly think that everyone is faulty
- Virtual synchrony can only survive failures of "less than half" the nodes
  - Else subject to split brain problems
- So mistakes can stall the protocol

What about Paxos?

CS514

- Issue is basically the same
- Needs majority agreement on each message
- So if a majority are inaccessible Paxos can freeze
- Again, apparent failures can cause endless delays in the protocol

Theoretician's revenge

CS514

- The theory community developed a solution guaranteed to make "optimal progress"
  - Based on a failure detector called <>W which is allowed to make mistakes
  - And a rather slow consensus protocol
- But <>W can't really be implemented
- And the sluggish performance makes this whole approach a non-starter
- Bottom line: we can't guarantee progress…

Summary?

CS514

- Data replication is complicated by an impossibility result
  - But the result revolves around what you mean by "impossible"
- As a practical matter, the result isn't very important
  - Doesn't lead to better practical options
- But practical issues seen previously *are* important. And the theoretical results further confuse industry and have hence contributed to slow uptake of replication