



CS514: Intermediate Course in Computer Systems

Lecture 25: March 24, 2003

Building a data replication protocol



Why replicate?

CS514

- We've seen many situations where replicated data arises in modern systems
- Yet modern architectures like Web Services lack replication primitives
 - The big exception is CORBA, which has a "fault-tolerance" standard!
- Today, look at how to support replicated data in such systems



At what level should we work?

CS514

- Could build our protocol over SOAP-RPC on HTTP
 - Advantage is that this plays to the standards
 - But our solutions involve lots of peer-to-peer style interactions and messages may be large
- UDP would also be appealing
 - Firewalls, NATs traditionally made this hard but with STUN the story is changing
- TCP also an option
 - Issue here is that it can break channels when we would rather that it didn't!
 - But could disable the "KEEPALIVE" option and take manual control of TCP connection liveness



Specifying the problem

CS514

- Before we set out to solve a problem we need to understand our goals
- What should be the goals of a data replication protocol?
 - Processes should be able to "join" into "process groups"
 - Also to leave. If they fail, automatically detected and group membership adjusted.
 - A joining process needs the "state" of the group to initialize itself



Concept of “group state”

CS514


- Assume your program is constructed in a modular manner
- Each group will have a role
 - E.g. the “state of ATC section x.y.z over Paris”, or “the list of flights awaiting clearance for takeoff”
 - The program will have a data structure to represent this
 - The state is basically the data in that structure



Concept of the group state

CS514


- Basically, we want to use replication to deliver updates to group members
 - If they start in the same state
 - And apply the same updates
 - Then they will stay in the same state
- Of course, different members may have different roles



Two examples

CS514

- ATC flight information
 - State might be a sort of picture of the air sector
 - Updates describe things planes are doing
 - Everyone updates the picture identically and so all consoles in a cluster have the equivalent state



Two examples

CS514

- Searching a very big database
 - Group members replicate the database
 - And they all know about the search requests
 - With k members, divide database equally into k parts
 - Process i will search the i 'th chunk of the database
 - Combine the k separate responses to get a single composite response
 - This can give a high degree of parallelism



State Machine Approach

CS514

- Leslie Lamport and Fred Schneider formalized this concept as the “State Machine Approach”
 - But they advocated a fairly expensive way of implementing state machines
 - At the core was a question of fault-tolerance
- How fault-tolerant should a group be?



Fault models

CS514

- Most benign is “fail stop” model
 - Network never fails. Processes fail by halting and notification is reliable and immediate
- Realistic model
 - Network can lose packets. Processes crash and halt but no notifications are sent
- Paranoid model (“Byzantine”)
 - Includes possibility of malicious behavior, corruption of packets, lying, cheating, spoofing, etc.
 - State Machine model of Lamport and Schneider was formulated in this very pessimistic failure model
 - This resulted in a hugely expensive solution

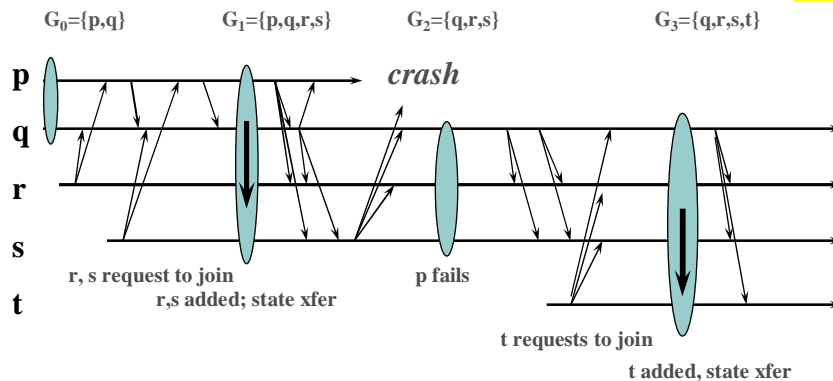
A modest goal

CS514

- Aim for the realistic failure model
- But seek to support the real needs of real applications
 - This leads to what we call the *virtual synchrony* execution model
 - Motivation can be traced to database *serializability* model (but we won't digress on that topic today)

Virtual Synchrony Model

CS514



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



Building such a system

CS514

- The system is structured into layers
 - Top layer is what the user sees
 - Often, a collection of software tools similar to what Visual C# might offer
 - E.g. a locking tool, a tool for performing an action fault-tolerantly, etc
 - Middle layer implements the virtual synchrony model
 - Lowest layer implements failure monitoring service



Failure monitoring service

CS514

- Basically similar to what Rimon discussed last Friday
 - It uses various methods to detect apparent failures of processes in the system
 - Notifies anyone who is interested
 - Is itself a fault-tolerant service
 - Kills a process or machine if it was mistakenly reported as having failed



Core primitives

CS514

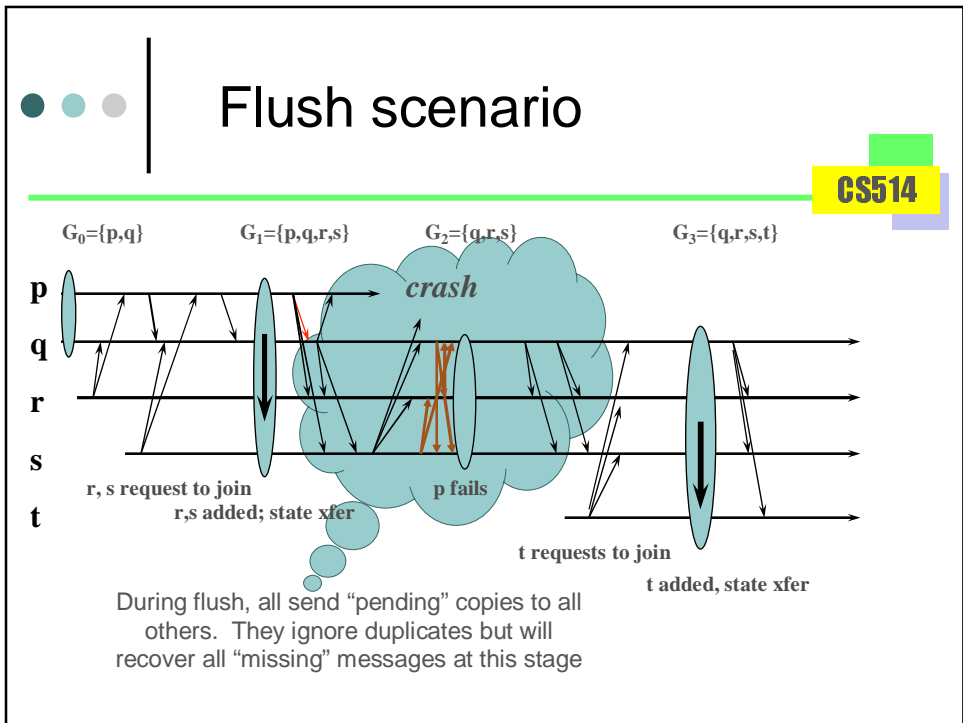
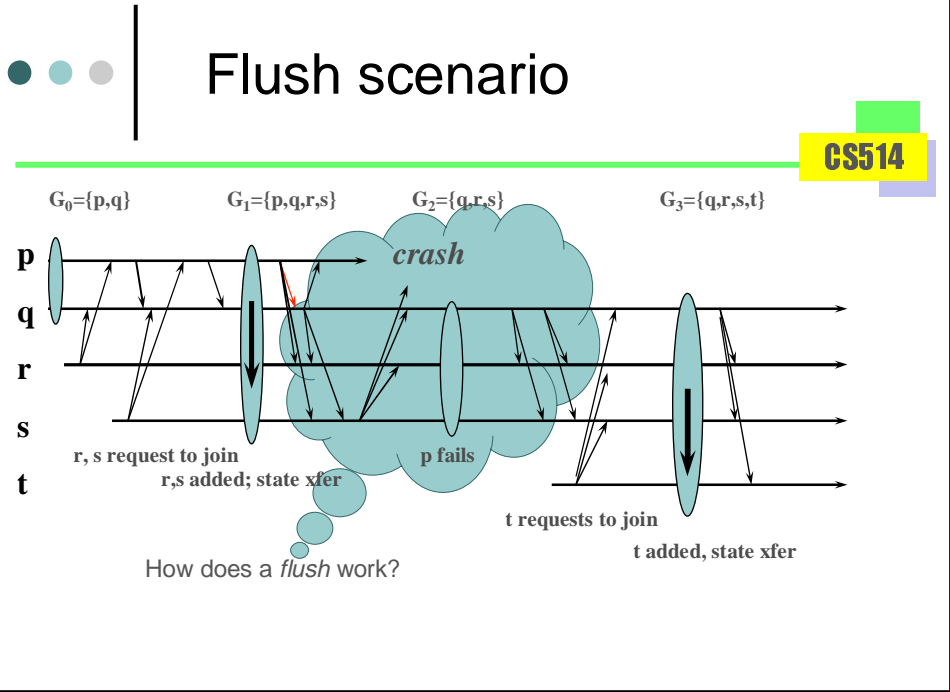
- Next we build a set of “multicast” primitives without worrying about crashes
- Sender calls `mcast(grp-id, data,)`
 - System look at membership of group
 - Sends a copy of data to each member
 - Waits for acks
 - Then notifies members of completion



Failures?

CS514

- Each receiver stashes a copy of the incoming messages
- When a member fails...
 - Each receiver looks in its stash
 - Retransmits copy to everyone who still seems to be alive
- We call this a “flush” mechanism. Notice the burst of all-to-all traffic when a crash occurs!





Group membership reporting

CS514

- Basically, the system can use multicast to send updates to the membership list
- Trick is to “order” regular data updates either before or after membership changes
- Goal is that everyone gets a message with the same idea of membership
 - Thought question: *why do we care?*



Ordering multicasts

CS514

- Suppose several members try to update the same piece of state
- If these updates are processed in different orders the copies could become inconsistent
- This motivates us to think about multicast ordering: the system can pick an order and enforce it



Types of ordering

CS514

- None: “mcast”
- In the order they were sent (looks at a single sender only): fbcast
- In “causal” order (like fbcast but if sender a transmits m_0 and b, receiving m_0 , sends m_1 , interprets m_1 as being “after” m_0). Called cbcast
- A system-wide “atomic” order: abcast



Implementing ordering?

CS514

- Easiest cases are mbcast, fbcast, cbcast: just need a timestamp on the message, or a vector of timestamps (one per sender) for cbcast.
- Harder is abcast.
 - Most systems use a sequencer process – a leader who assigns an ordering.
 - Some use more elaborate voting schemes
 - But no matter what, abcast will be slower than cbcast, fbcast or mbcast



What about state transfer

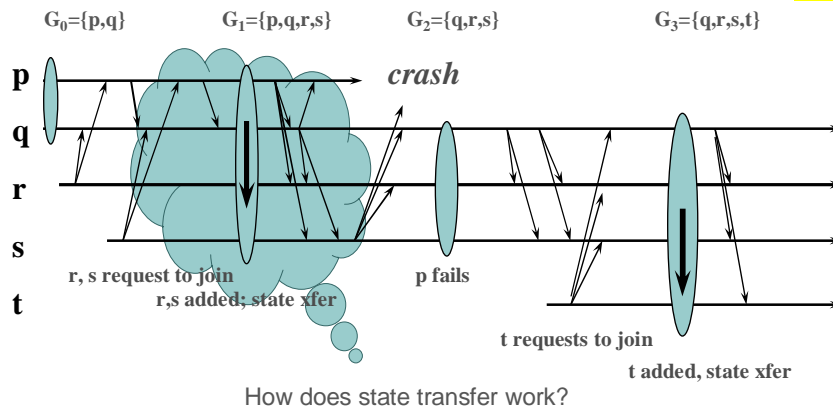
CS514

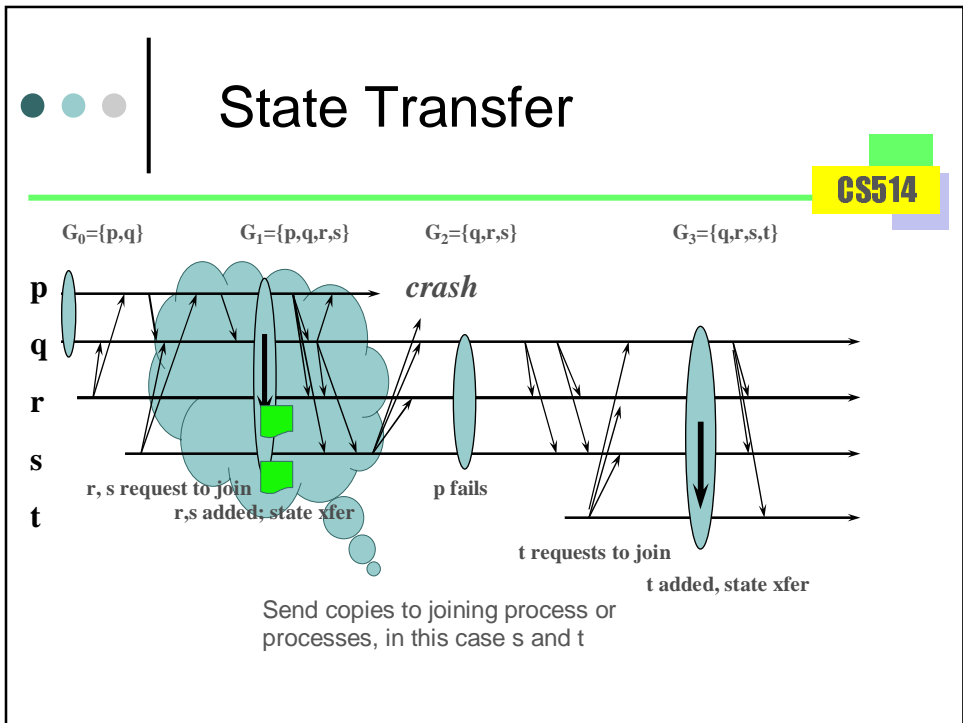
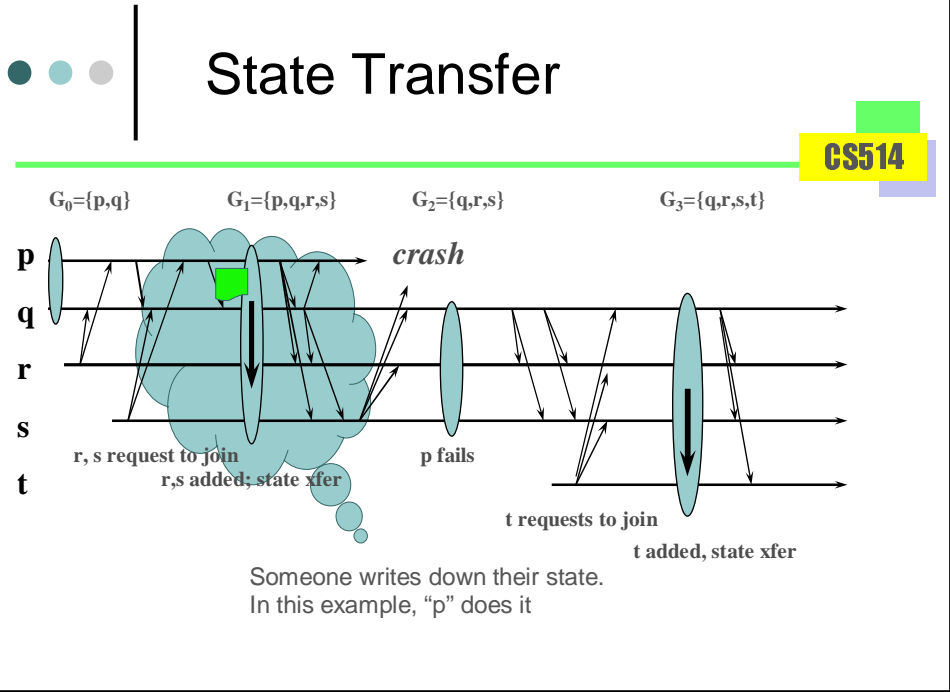
- This is the issue of initializing a joining member of a group
 - Some existing member writes down its state, like making a checkpoint
 - The new member reads this state in and constructs the data structures needed
 - Then the new guy can process updates like everyone else



State Transfer

CS514







State transfer tricks

CS514

- Issue is to ensure that state is not changing while being transferred!
 - Applying an update “twice” could leave joining process inconsistent
 - So could neglecting to apply an update at all!
- So state transfer must occur after all messages have been delivered in the “prior” group membership view and before reporting the new membership



Systems implementing virtual synchrony

CS514

- This is the model used by CORBA
- Many systems have implemented the model
 - At Cornell: Isis, Horus, Ensemble. All support “many” groups and have user-oriented “toolkits”, e.g. JavaGroups
 - At UCSD: Totem, Eternal. Later was basis of CORBA FTOL specification
 - At Hebrew University: Transis
 - May others: perhaps a dozen or more!



Other popular technologies

CS514

- Lately, Paxos is often discussed
 - Leslie Lamport built this as a relaxation of his State Machines approach
 - But Paxos is still quite costly
- Fundamental difference?
 - Paxos only delivers a message iff majority of group members acknowledge receipt of it
 - Idea is that such a message can never be forgotten if a crash occurs. But this slows things down dramatically!
 - Also, Paxos only supports abcast



Getting virtual synchrony to act like Paxos

CS514

- These systems normally support “flush the group” as a primitive you can call
- If you do a flush operation, everyone will have seen all the same messages that you have seen
- So all those messages now have the properties of a Paxos message



Pros and Cons...

CS514


- Paxos is extremely slow by comparison to virtual synchrony
 - In virtual synchrony we can reach 80,000 updates per second in a small group
 - Paxos normally runs at rates of 50 or 100 per second maximum
- But Paxos doesn't ask the user to "think" about reliability needs...



Where things get hard

CS514

- Real systems need to worry about
 - Scaling in numbers of groups
 - Supporting very high performance (lots of multicasts streamed at high speeds)
 - Rapid membership changes
 - Heterogeneous platforms
 - Security of the model and implementation



Experience with process groups?

CS514

- Ken wrote a paper on this
 - New York Stock Exchange – uses groups to manage distribution of quotes. But actual data flows on TCP
 - Swiss Exchange – replicates entire exchange!
 - French ATC system: Console clusters and flight plan updates
 - Florida Electric and Gas – large scale control
 - AEGIS warship for onboard communication despite “failures”...



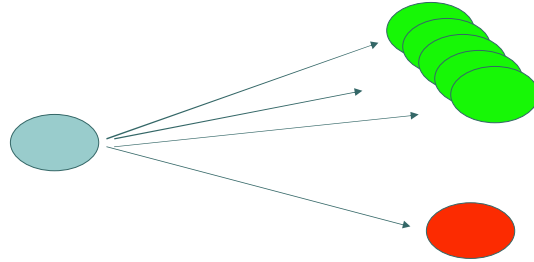
Overall findings

CS514

- Scaling is a real issue for many applications
 - Especially vast numbers of groups, large membership
- Lack of standards and commercial product support also a serious issue
- But vendors have hesitated to offer such a complex option to users...

Virtual Synchrony Scaling Issue

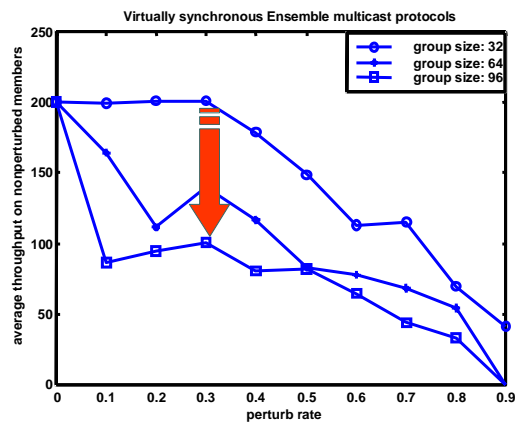
CS514



Group has many healthy members. But one is "perturbed" (slowed down) just a little bit...

Virtual Synchrony scaling issue

CS514





Summary?

CS514

- Almost two decades of experience with replication now
- Virtual synchrony and Paxos models are widely recognized and used
 - But they have fared poorly as stand-alone products in the market
 - Big OS vendors use similar tricks but hide them inside their systems. User's can't access the primitives