Too many seminal concepts

- Process pairs, primary-backup
- 2PC and 3PC, Quorums
- Atomic Transactions
- State machine replication
- RAID storage solutions
- Checkpoints, Message Logging
- Byzantine Agreement
- Gossip protocols
- Virtual synchrony model
- Paxos
- Zookeeper

Theory

- Consensus
- FLP
- ◊W: consensus + oracle

... Skepticism

- CATOCS
- CAP
• Too much for 25 minutes...
• Focus on state machine replication with crash failures
Fault-Tolerance via Replication: Rich History

• Early debate about the question itself
  • Some believed that the OS layer is the wrong place to offer strong properties...

• Today that debate has reemerged:
  • Some believe that the cloud can’t afford strong properties!
Theory
Basic questions

• What sort of system are we talking about?

• What do we mean by “failure”?

• What does “tolerating” mean?
Thinking of Fault-Tolerance in terms of Safety

• Consistent State: A system-specific invariant: Pred(S)

• S is fault-tolerant if:

\[ S \text{ maintains/restores } \text{Pred}(S) \text{ even if something fails} \]

• Normally, we also have *timeliness* requirements.
Principles from the theory side...

• FLP: Protocols strong enough to solve asynchronous consensus cannot guarantee liveness (progress under all conditions).

• If running a highly available database with network partition, conflicting transactions induce inconsistencies (CAP theorem).

• Need 3f+1 replicas to overcome Byzantine faults
Systems
Principles from the systems side...

• Make core elements as simple as possible
  • Pare down, optimize the critical path
  • Captures something fundamental about systems.

• Generalized End-to-End argument:
  • Let the application layer pick its own models.
  • Limit core systems to fast, flexible building blocks.

Gray: How do systems really fail?

• Studied Tandem’s “non-stop” platforms
  *Failures caused by bugs, user mistakes, poor designs.*
  *Few hardware failures, and nothing malicious.*

• Jim’s advice? Focus our efforts on the real needs

Tensions

Why aren’t existing OS mechanisms adequate?

Is fault-tolerance / consistency too complex or costly?

Do the needed mechanisms enable or impose models?
Do we need fault-tolerant replication?

• Not just for making systems tolerant of failures
  • Cloud computing: Provision lots of servers
  • Performance-limiting for many machine-learning systems

• So we agree, hopefully: replication is awesome!

• But is there a core OS mechanism here?
It comes down to performance and scalability

• As systems researchers, abstracted properties are...
  • Useful when designing and testing
  • Valuable tools for explaining behavior to users
  • Not obstacles: “Impossible” problems don’t scare us...

• Performance is a more fundamental challenge
  • Can fault-tolerance mechanisms be fast?
Existing core OS support: Inadequate

• IP multicast just doesn’t work…
  • Amazon AWS disables IPMC and tunnels over TCP

• TCP is the main option, but it has some issues:
  • No support for reliable transfer to multiple receivers
  • Uncoordinated model for breaking connections on failure
  • Byte stream model is mismatched to RDMA
... Higher-level replication primitives?

- Isis: In 1985 used state machine replication on objects
  - Core innovation was its group membership model, which integrates membership dynamics with ordered multicast.
  - Durability tools: help application persist its state

- Paxos*: Implements state machine replication (1990)
  - A durable database of events (not an ordered multicast)
  - Runs in “quasi-static” groups.

*Homework: First version of Paxos protocol?
Delays on the critical path: Isis

Original Isis Toolkit:
State machine replication of user-defined objects.
Durability was optional.

Oracle
- Uses quorums
- Outputs “Views”
- Bisimulates Paxos

Critical Path
- Asynchronous, pipelined
- Flush when view changes
- Only pay for properties used

Virtual Synchrony: Model + menu of choices
[Note: CATOCS controversy arose here...]

Paxos: Many optimizations, often via transformations like the Isis ones

But Paxos theory and formal methodology are very clean, elegant...
How does one speed such systems up?

• Start with simple, easily analyzed solution... Study the code
  • The critical paths often embody inefficiencies, like requesting total order for actions already in order, or that commute.
  • Often, synchronous events can be asynchronously pipelined

• Restructure critical paths to leverage your insights
  • Hopefully, the correctness argument still holds...

Pattern shared by Isis, Paxos, Zookeeper, Chain Replication, Zyzzyva, many others...
... Real systems informed by sound theory

• Isis: Widely adopted during the 1995-2005 period
  • French ATC system, US Navy AEGIS, NYSE...

• Paxos: Very wide uptake 2005-now
  • Locking, file replication, HA databases...
  • Clean methodology and theory appeal to designers
  • Corfu is the purest Paxos solution: robust logging
CATOCS: A case against consistent replication

• Too complex
• Violates End-to-End by imposing model on the user
• No matter what form of update order is supported, user won’t like it
• Ordering is just too slow, won’t scale
So were CATOCS claims true?

• Early replication solutions really were too slow.
  • Later ones were faster, but more complex.

• But CATOCS analysis of ordering was dubious.

• Yet... what about that missing low-level building block?
  • ... a puzzle (we’ll come back to it later)
The “consensus” family...

• Can transform one to another... optimizations driven by desired properties.

• For me, durability remains puzzling

  • Is the goal durability of the application, or of its “state”?
... a few winners:

- State Machine Replication, Paxos, ACID transactions
- Chubby, Zookeeper, Corfu
- Primary + Warm backup... Chain Replication
Meanwhile, along came a cloud!

Servers: 3-5 nodes

A cloud-hosted service could run on 5,000 nodes in each of dozens of data centers
... Cloud rebellion: “Just say no!”

- State Machine Replication, Paxos, ACID transactions
- Chubby, Zookeeper, Corfu
- Primary + Warm backup... Chain Replication
- Dynamo: Eventual consistency (BASE), NoSQL KVS

Conceptual Tools

Real Systems

ACID
Is consistency just too costly?

• **CAP**: Two of {Consistency, Availability, Partition-Tolerance}
  - Widely cited by systems that cache or replicate data
  - Relaxed consistency eliminates blocking on the critical path
  - CAP theorem: proved for a WAN partition of an H/A database

• **BASE** (eBay, Amazon)
  - Start with a transactional design, but then weaken atomicity
  - Eventually sense inconsistencies and repair them
... but does CAP+BASE work?

• CAP folk theorem: “don’t even try to achieve consistency.”

... meaning what?
  • “Anything goes”? “Bring it on?”

• Einstein: “A thing should be as simple as possible, but not simpler.”
... but does CAP+BASE work?

• **CAP folk theorem:** “*don’t even try to achieve consistency.*”

• **CAP + BASE are successful *for a reason:***
  • In the applications that dominate today’s cloud, stale cache reads have negative utility but don’t cause safety violations.
  • In effect a *redefinition*, not a rejection, of consistency
A fascinating co-evolution

• The cloud fits the need; the applications fit the cloud. At first, fault-tolerance wasn’t given much thought.

  • Jim Gray :  “Why do systems fail?”

  • Today:   Why don’t CAP+BASE systems fail?

• Could we apply Dijkstra’s theory of “self-stabilization” to BASE?

Future Shock: Disruption is coming

• Life and safety-critical cloud computing...
  • Smart power grid, homes, cities
  • Self-driving cars
  • Cloud-hosted banking, medical care

• Weakened consistency won’t suffice for these uses.
Homework (due date: SOSP 2017)

• Start with a clean slate (but do learn from the past)

• Embrace a modern architecture
  • Cloud-scale systems...
  • Multicore servers with NVRAM storage
  • RDMA (like Tesla’s “insane speed” button).

• Propose a new approach to cloud-scale consistency
Future Cloud...

• The O/S has been an obstacle... even embraced inconsistency.
  • The future cloud should embrace consistency.

• Key: Elegance, speed, support real needs of real developers

• Need a core OS building block that works, integrated with developer tools and IDEs that are easy to use.