Paxos: Agreement for Replicated State Machines

Brad Karp
UCL Computer Science

CS GZ03 / M030

2nd November, 2009

Review: Types of Distributedness

- NFS: distributed to share data across clients through filesystem interface
- Ivy: distributed to provide illusion of seamless shared memory across clients
- 2PC: distributed because different nodes have different functions (e.g., Bank A, Bank B)
- What about distributedness to make system more **available?**

Centralization: Single Points of Failure

- Consider what happens when nodes fail:
  - NFS server?
  - Bank A?
  - CPU that owns a page in Ivy?
- In all these systems, there is single node with "authoritative" copy of some data
- Single point of failure: kill one node, clients may grind to halt
- How can we do better?

Replication

- **Replicate** data on several servers
- If server(s) fail, hopefully others still running; data still available, clients can still make progress
- **Consistency?**
  - Informally speaking, all replicas should hold identical copies of data
  - So as users' requests modify data, must somehow *keep* all data identical on all replicas

2PC vs. Replication

- 2PC works well if different nodes play different roles (e.g., Bank A, Bank B)
- 2PC isn't perfect
  - Must wait for all sites and TC to be up
  - Must know if each site voted yes or no
  - TC must be up to decide
  - Doesn't tolerate faults well; must wait for repair
- Can clients make progress when some nodes unreachable?
  - Yes! When data replicated.

State Machine Replication

- Any server essentially a state machine
  - Disk, RAM, CPU registers are state
  - Instructions transition among states
  - User requests cause instructions to be executed, so cause transitions among states
- Replicate state machine on multiple hosts
  - Every replica must see same operations in same order
  - If deterministic, replicas end in same state

Ensuring All Replicas See Operations in Same Order

- Nominate one “special” server: primary
- Call all other servers backups
- Clients send all operations to current primary
- Primary’s role:
  - Chooses order for clients’ operations
  - Sends clients’ operations to backups
  - Replies to clients

Ensuring All Replicas See Operations in Same Order

Didn't we say the whole point was availability, and fault-tolerance?

What if primary fails?

primary

- Primary's role:
  - Chooses order for clients' operations
  - Sends clients' operations to backups
  - Replies to clients

Primary Failure

- Last operation received by primary may not be complete
- Need to pick new primary
- Can't allow two simultaneous primaries! **(Why?)**
- Define: lowest-numbered live server is primary
  - After failure, everyone pings everyone
  - Does everyone now know who new primary is?
- Maybe not:
  - Pings may be lost: two primaries
  - Pings may be delayed: two primaries
  - Network partition: two primaries

Idea: Majority Consensus

- Require a majority of nodes to agree on primary
- At most one network partition can contain majority
- If pings lost, and thus two potential primaries, **majorities must overlap**
  - Node(s) in overlap can see both potential primaries, raise alarm about non-agreement!

Technique: View Change Algorithm

- Entire system goes through sequence of views
- View: {view #, set of participant nodes}
- View change algorithm must ensure agreement on unique successor for each view
- Participant set within view allows all nodes to agree on primary
  - Same rule: lowest-numbered ID in set is primary

Technique: View Change Algorithm

If two nodes agree on view, they will agree on primary

- View: {view #, set of participant nodes}
- View change algorithm must ensure agreement on unique successor for each view
- Participant set within view allows all nodes to agree on primary
  - Same rule: lowest-numbered ID in set is primary

View Change Requires Fault-Tolerant Agreement

- Envision view as opaque value
- Want all nodes to agree on same value (i.e., same view)
- At most one value may be chosen
- Want to agree despite lost messages and crashed nodes
- Can't guarantee to agree!
  - Can guarantee **not to agree** on different values!
  - i.e., guarantee safety, but not liveness

Paxos: Fault-Tolerant Agreement Protocol

- Protocol eventually succeeds provided
  - Majority of participants reachable
  - Participants know how to generate value to agree on
    - i.e., Paxos doesn't determine the value nodes try to agree on—value is an opaque input to Paxos
- Only widely used algorithm for fault-tolerant agreement in state machine replication

Review: State Machine Replication, Primary-Backup, Paxos

- **How did we get here?**
- Want to replicate a system for availability
- View system as state machine; replicate the state machine
- Ensure all replicas see same ops in same order
- Primary orders requests, forwards to replicas
- All nodes must agree on primary
- All nodes must agree on view
  - Participant with lowest address in view is primary
- Paxos guaranteed to complete only when all nodes agree on input (in this case, input is view)

Overview of Paxos

- One (or more) nodes decide to be leader
- Leader chooses proposed value to agree on
  - (In our case, value is view: {view #, participant set})
- Leader contacts Paxos participants, tries to assemble majority
  - Participants can be fixed set of nodes (configured)
  - Or can be all nodes in old view (including unreachable nodes)
- If a majority respond, successful agreement

Agreement is Hard!

- What if two nodes decide to be leader?
- What if network partition leads to two leaders?
- What if leader crashes after persuading only some nodes?
- What if leader got majority, then failed, without announcing result?
  - Or announced result to only a few nodes?
  - **New leader might choose different value, despite previous agreement**

Paxos: Structure

- Three phases in algorithm
- May need to restart if nodes fail or timeouts waiting for replies
- State in each node running Paxos, per-value (view):
  - n_a: greatest n accepted by node (init: -1)
  - v_a: value received together with n_a (init: nil)
  - n_h: greatest n seen in Q1 message (init: -1)
  - done: leader says agreement reached; can use new value (i.e., start new view) (init: 0)

Paxos: Phase 1

- A node (maybe more than one) decides to be leader, then it
  - picks proposal number, n
    - must be unique, good if higher than any known proposal number
    - use last known proposal number + 1, append node's own ID
  - sends Q1(n) message to all nodes (including self)
- if node receives Q1(n) and $n > n_h$
  - $n_h = n$
  - send reply R1(n_a, v_a) message

Paxos: Phase 2

if leader receives R1 messages from majority of nodes (including self)

- if any R1(n, v) contained a value (v)
  - v = value sent with highest n
- else leader gets to choose a value (v)
  - v = {old view# + 1, set of pingable nodes}
- send Q2(n, v) message to all responders

if node receives Q2(n, v) and $n >= n_h$

- $n_h = n_a = n$
- $v_a = v$
- send reply R2() message

Paxos: Phase 3

if leader receives R2() messages from majority of protocol participants

send Q3() message to all participants

if node receives Q3()

done = true

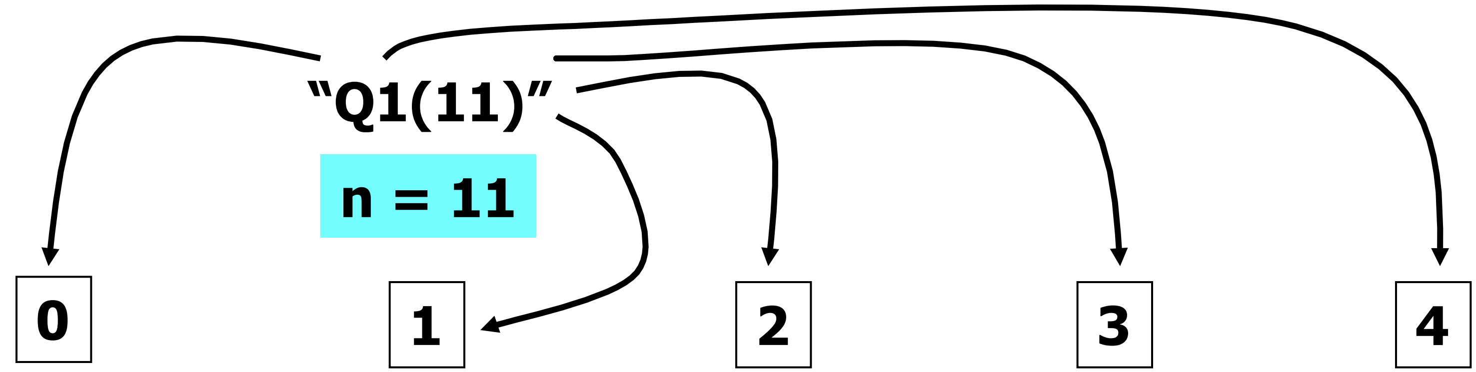
agreement reached; agreed-on value is v_a

(primary is lowest-numbered node in participant list within v_a)

Paxos: Timeouts

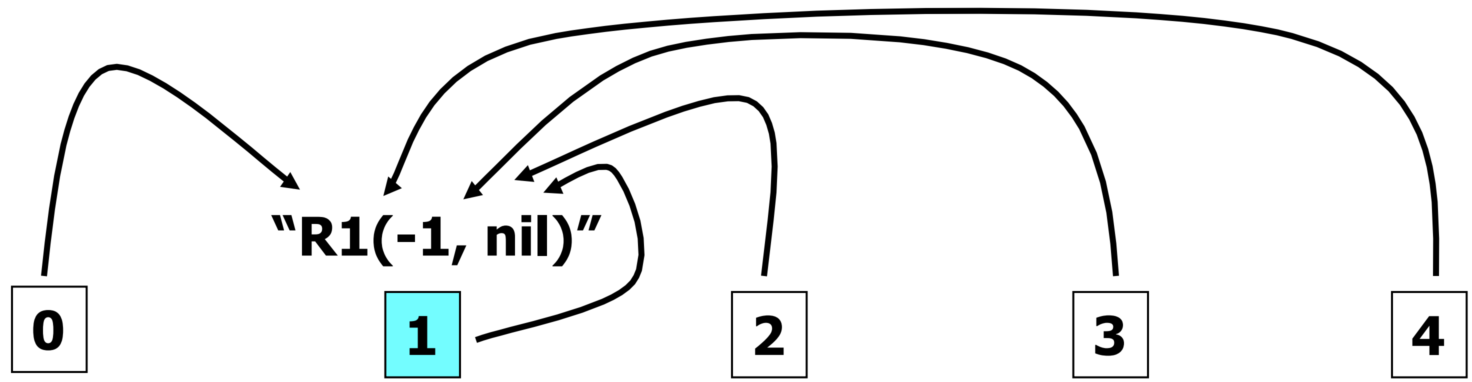
- All nodes wait a maximum period (timeout) for messages they expect
- Upon timeout, a node declares itself a leader and initiates a new Phase 1 of algorithm

Paxos with One Leader, No Failures: Phase 1



n_a	-1	-1	-1	-1	-1
v_a	nil	nil	nil	nil	nil
n_h	-1	-1	-1	-1	-1
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 1

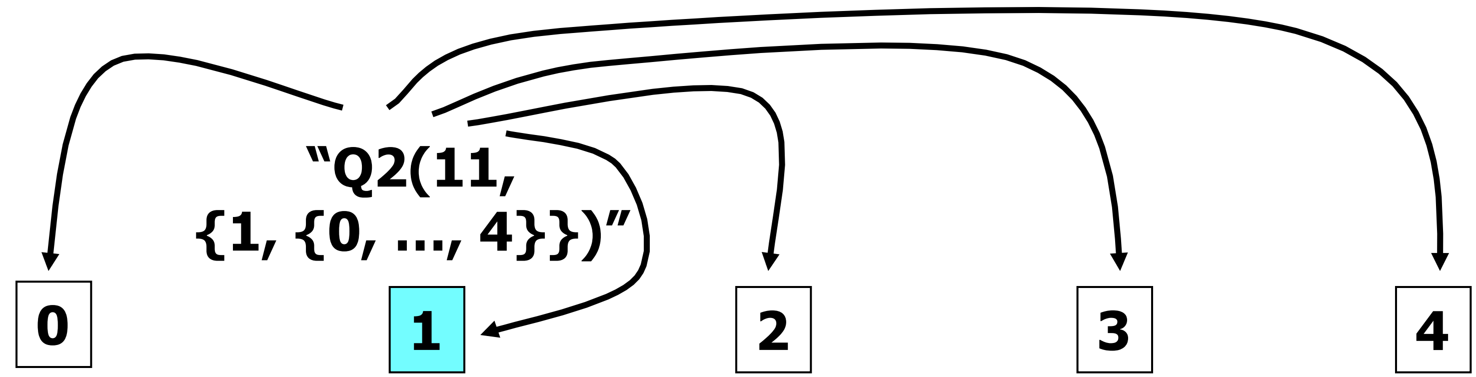


n_a	-1	-1	-1	-1	-1
v_a	nil	nil	nil	nil	nil
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 2

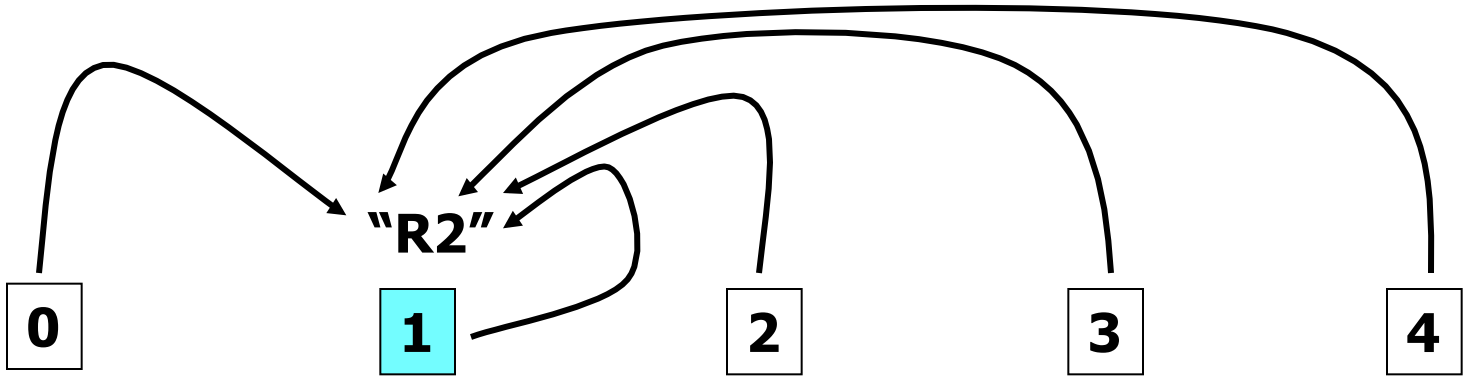
	0	R1 from majority! all v's nil 1	2	3	4
n_a	-1	-1	-1	-1	-1
v_a	nil	nil	nil	nil	nil
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 2



n_a	-1	-1	-1	-1	-1
v_a	nil	{1, {0, …, 4}}	nil	nil	nil
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 2



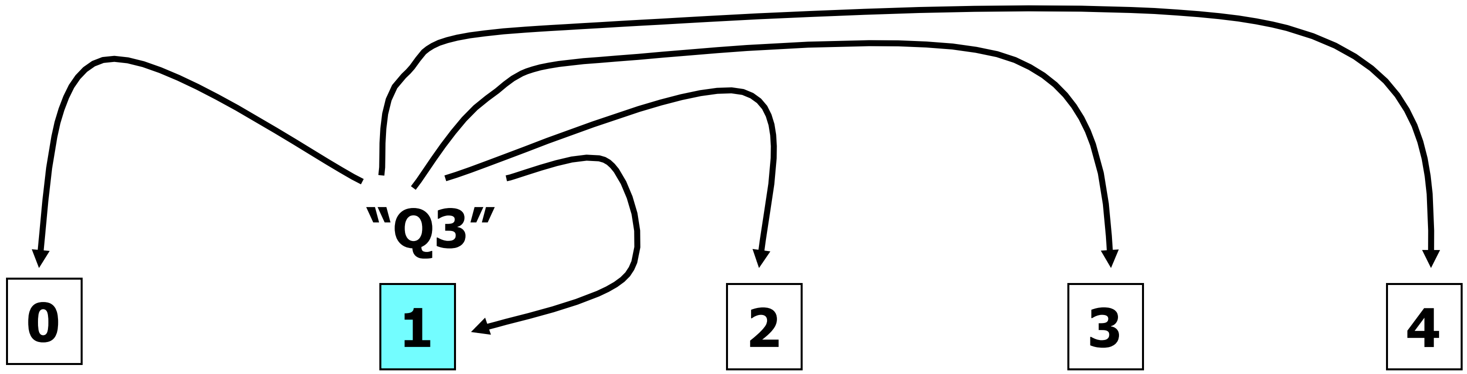
n_a	11	11	11	11	11
v_a	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 3

R2 from majority!

	0	1	2	3	4
n_a	11	11	11	11	11
v_a	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 3



	0	1	2	3	4
n_a	11	11	11	11	11
v_a	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}
n_h	11	11	11	11	11
done	F	F	F	F	F

Paxos with One Leader, No Failures: Phase 3

	0	1	2	3	4
n_a	11	11	11	11	11
v_a	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}
n_h	11	11	11	11	11
done	T	T	T	T	T

Paxos with One Leader, No Failures: Phase 3

All nodes agree on view {1,{0, …, 4}}
New primary: lowest ID, so node 0

	0	1	2	3	4
n_a	11	11	11	11	11
v_a	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}	{1, {0, …, 4}}
n_h	11	11	11	11	11
done	T	T	T	T	T

Paxos: Number of Leaders

- Clearly, when no failures, no message losses, and one leader, **Paxos reaches agreement**
- How can one ensure that with high probability, only one leader?
  - Every node must be willing to become leader in case of failures
  - Every node should delay random period after realizing pingable nodes have changed, or delay own ID x some constant

Paxos: Ensuring Agreement

- When would non-agreement occur?
  - When nodes with different v_a receive Q3
- Safety goal:
  - If Q3 could have been sent, future Q3s guaranteed to reach nodes with same v_a

Risk: More Than One Leader

- Can occur after timeout during Paxos algorithm, partition, lost packets
- Two leaders must use different n in their Q1()s, by construction of n
- Suppose two leaders proposed n = 10 and n = 11

More Than One Leader (2)

- Case 1: proposer of 10 didn’t receive R2()s from majority of participants
  - Proposer never will receive R2()s from majority, as no node will send R2() for R1(10, …) after seeing R1(11,…)
  - Or proposer of 10 may be in network partition with minority of nodes

More than One Leader (3)

- Case 2: proposer of 10 (10) did receive R2()s from majority of participants
  - Thus, 10's originator may have sent Q3()!
  - But 10's majority must have seen 10's Q2() before 11's Q1()
    - Otherwise, would have ignored 10's Q2, and no majority could have resulted
  - Thus, 11 must receive R1 from at least one node that saw 10's Q2
  - Thus, 11 must be aware of 10's value
  - Thus, 11 would have used 10's value, rather than creating one!

More than One Leader (3)

Result: agreement on 10’s proposed value!

from majority of participants

- Thus, 10’s originator may have sent Q3()!
- But 10’s majority must have seen 10’s Q2() before 11’s Q1()
  - Otherwise, would have ignored 10’s Q2, and no majority could have resulted
- Thus, 11 must receive R1 from at least one node that saw 10’s Q2
- Thus, 11 must be aware of 10’s value
- Thus, 11 would have used 10’s value, rather than creating one!

Risk: Leader Fails Before Sending Q2()s

- Some node will time out and become a leader
- Old leader didn't send any Q3()s, so no risk of non-agreement caused by old leader
- Good, but not required, that new leader chooses higher n for proposal
  - Otherwise, timeout, some other leader will try
  - Eventually, will find leader who knew old n and will use higher n

Risks: Leader Failures

- Suppose leader fails after sending minority of Q2()s
  - Same as two leaders!
- Suppose leader fails after sending majority of Q2()s
  - i.e., potentially after reaching agreement!
  - Also same as two leaders!

Risk: Node Fails After Receiving Q2(), and After Sending R2()

- If node doesn't restart, possible timeout in Phase 3, new leader
- If node does restart, it must remember v_a and n_a on disk!
  - Leader might have failed after sending a few Q3()s
  - New leader must choose same value
  - This failed node may be only node in intersection of two majorities!

Paxos: Summary

- Original goal: replicated state machines!
  - Want to continue, even if some nodes not reachable
- After each failure, perform view change using Paxos agreement
- i.e., agree on exactly which nodes in new view
- Thus, everyone can agree on new primary
- No discussion here of how to render **data** consistent across replicas!