Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
MIT
Presented to cs294-4 by Owen Cooper

The problem

- Provide a reliable answer to a computation even in the presence of Byzantine faults.
- A client would like to
  - Transmit a request
  - Wait for k replies
  - Conclude that the answer is a true answer

The Model

- Networks are unreliable
  - Can delay, reorder, drop,retransmit
- Some fraction of nodes are unreliable
  - May behave in any way, and need not follow the protocol.
- Nodes can verify the authenticity of messages

Failures

- The system requires 3f+1 nodes to withstand f failures
  - All f nodes may be faulty, and not respond
  - But there is no guarantee that the remaining n-f are good, and good nodes must outnumber bad nodes.
  - This holds if n-2f > f or n > 3f

Nodes

- Maintain a state
  - Log
  - View number
  - state
- Can perform a set of operations
  - Need not be simple read/write
  - Must be deterministic
- Well behaved nodes must:
  - start at the same state
  - Execute requests in the same order

Views

- Operations occur within views
- For a given view, a particular node in is designated the *primary* node, and the others are *backup* nodes
- Primary = v mod n
  - N is number of nodes
  - V is the view number

Protocol

A three phase protocol

- Pre-prepare: primary proposes an order
- Prepare: Backup copies agree on #
- Commit: agree to commit

Agreement

- Quorum based
- 2f+1 nodes must have same value
  - System has 3f+1 nodes
  - Any 2f+1 subset has >= 1 good node in common
    - Good nodes don't lie
    - Same decision at each node w/ quorum

Messages

- The following messages are used by the protocol, and are signed by the sender
  - Request <o,t,c> (called m)
    - Sent from the client to the primary
    - Contains: client #, timestamp, and operation
  - Reply <v,t,c,I,r>
  - Pre-prepare <v,d,n>, m
    - Multicast from primary to backups
    - Contains view #, sequence #, digest
    - Message may be sent separately

Messages 2

- Prepare <v,n,d,I >
  - Sent amongst backups
- Commit <v,n,d,I >
  - Replica I is prepared to commit seq # n, view v
- Messages are accepted in each phase
  - If the current node is in view v
  - The sequence number,n, is within a certain range
  - The node has not received contradictory messages
  - The digest matches the computed digest

Pre-prepare

- The client sends a message to the primary
- The primary assigns a sequence number to the message, and multicasts it.
- Backups:
  - Receive the pre-prepare message
  - Validate it and drop the message if invalid
  - Record the message, the pre-prepare message, and a newly generated prepare message in the log
  - Multicast the prepare message to the other backups

Prepare 2

- A prepare message indicates a backups willingness to accept a given sequence number.
- Once a quorum of messages prepare messages is received, a commit message is sent

Commit

- Nodes must ensure that enough nodes have all been prepared before applying the changes so:
  - A node waits for a quorum of commit messages before applying a change.
  - Changes are applied in order of sequence number
    - Cannot be applied until all lower numbered messages have been applied

Truncating the log

- Checkpoints at regular intervals
- Requests are in log, or already stable
- Each node maintains multiple copies of state:
  - A copy of the last proven checkpoint
  - 0 or more unproven checkpoints
  - The current working state
- A node sends a checkpoint message when it generates a new checkpoint
- checkpoint is proven when a quorum agrees
  - Then this checkpoint becomes stable
  - Log truncated, old checkpoints discarded

View change

- The view change mechanism
  - Protects against faulty primaries
- Backups propose a view change when a timer expires
  - The timer runs whenever a backup has accepted some message & is waiting to execute it.
  - Once a view change is proposed, the backup will no longer do work (except checkpoint) in the current view.

View change 2

- A view change message contains
  - # of the highest message in the stable checkpoint
    - And the check point messages
  - A pre-prepare message for non-checkpointed messages
    - And proof it was prepared
- The new primary declares a new view when it receives a quorum of messages

New view

* uncheck pointed messages
  - New primary computes
    - Maximum checkpointed sequence number
    - Maximum sequence number not checkpointed
  - Constructs new pre-prepare messages
    - Either is a new pre-prepare for a message in the new view
    - Or a no-op pre-prepare so there are no gaps

New view 2

- New primary sends a new view message
  - Contains all view change messages
  - All computed pre-prepare messages
- Recipients verify:
  - The pre-prepare messages
  - The have the latest checkpoint
    - If not, they can get a copy
  - Sends a prepare message for each pre-prepare
  - Enters the new view

Controlling View Changes

- Moving through views too quickly
  - Nodes will wait longer if
    - No useful work was done in the previous view
      - I.e. only re-execution of previous requests\
      - Or enough nodes accepted the change, but no new view was declared
  - If a node gets f+1 view change requests with a higher view number
    - It will send its own view change with the minimum view number
    - This is safe, because at least one non-faulty replica sent a message

nondeterminism

- The model requires that requests be deterministic
  - But this is not always the case
    - E.g. update a timestamp using the current clock
- Two solutions
  - Let the primary propose a value
    - Create a <value, message> pair and proceed as before
  - Allow the backups to select values
    - Wait for 2f+1
    - Start three-phase protocol

optimizations

- Don't send f+1 messages back to the client
  - Instead send f digests, and 1 result
  - If they don't match, retry with old protocol
- Tentative commit
  - After prepare, backup may tentatively execute request
  - Client waits for a querom of tentative replies, otherwise retries and waits for f+1 replies
  - Read-only
    - Clients multicast directly to replicas
    - Replicas execute the request, wait until no tentative request are pending, return the result
    - Client waits for a quorum of results

Implementation

- The protocol is implemented in a replication library
  - No mechanism to change views
- Uses upcalls to allow servers to:
  - Invoke requests (client)
  - Execute requests
  - Create and delete checkpoints
  - Retrieve checkpoints
  - Compute digests (of checkpoints)

Implementation 2

- Communication
  - Udp for point to point
  - Udp multicast for group communication

Micro benchmark

- Compares a service that executes a no-op
  - Single server vs Replicated using protocol

arg./res. (KB)	replicated		without replication
	read-write	read-only	
0/0	3.35 (309%)	1.62 (98%)	0.82
4/0	14.19 (207%)	6.98 (51%)	4.62
0/4	8.01 (72%)	5.94 (27%)	4.66

Table 1: Micro-benchmark results (in milliseconds); the percentage overhead is relative to the unreplicated case.

BFS

- Implementation of NFS using the replication library.
  - Looks like normal NFS to clients
  - Replication library runs requsts via a relay
  - Server maintains filesystem state in memory mapped files

BFS 2

- Server maintains at most 2 checkpoints
  - Using copy on write
- Digests computed incrementally
  - For efficienty

Benchmark

- Andrew benchmark
- 5 phases
  - Create subdirectories
  - Copy source tree
  - Look at file status
  - Look at file contents
  - Compile
- Implementations compared
  - NFS
  - BFS strict
  - BFS (lookup, read are read only)

Results

	BFS		
phase	strict	r/o lookup	NFS-std
1	0.55 (-69%)	0.47 (-73%)	1.75
2	9.24 (-2%)	7.91 (-16%)	9.46
3	7.24 (35%)	6.45 (20%)	5.36
4	8.77 (32%)	7.87 (19%)	6.60
5	38.68 (-2%)	38.38 (-2%)	39.35
total	64.48 (3%)	61.07 (-2%)	62.52

Table 3: Andrew benchmark: BFS vs NFS-std. The times are in seconds.