6.824 2015 Lecture 8: Harp

Replication in the Harp File System
Liskov, Ghemawat, Gruber, Johnson, Shrira, Williams
SOSP 1991

Why are we reading this paper?
  Harp was the first complete replication system that dealt w/ partition
  It's a complete case study of a replicated service (file server)
  It uses Raft-like replication techniques

How could a 1991 paper still be worth reading?
  Harp introduced techniques that are still widely used
  There are few papers describing complete replicated systems

The paper is a mix of fundamentals and incidentals
  We care a lot about replication
  We may not care much about NFS specifically
    But we care a lot about the challenges faced when integrating
    a real application with a replication protocol.
  And we care about where optimization is possible.

I'm going to focus on parts of Harp that aren't already present in Raft.
  But note that Harp pre-dates Raft by 20+ years.
  Raft is, to a great extent, a tutorial on ideas pioneered by Harp.
    Though they differ in many details.

What does Harp paper explain that Raft paper does not?
  Adapting a complex service to state machine abstraction
    e.g. possibility of applying an operation twice
  Lots of optimizations
    pipelining of requests to backup
    witness
    primary-only execution of read-only operations using leases
  Efficient re-integration of re-started server with large state
  Power failure, including simultaneous failure of all servers
  Efficient persistence on disk

Harp authors had not implemented recovery yet
  Earlier paper (1988) describes View-stamped Replication:
    http://www.pmg.csail.mit.edu/papers/vr.pdf
  Later (2006) paper has clearer description, though a bit different:
    http://pmg.csail.mit.edu/papers/vr-revisited.pdf

Basic setup is familiar
  Clients, primary, backup(s), witness(es).
  Client -> Primary
  Primary -> Backups
  Backups -> Primary
    Primary waits for all backups / promoted witnesses in current view
    Commit point
  Primary -> execute and reply to Client
  Primary -> tell Backups to Commit

Why are 2b+1 servers necessary to tolerate b failures?
  Requiring majority for each operation helps ensure no other partition.
    And majority intersection helps ensure state persists to future views.
  Not waiting for b allows tolerance of b failures.

What are Harp's witnesses?
  The witnesses are one significant difference from Raft.
  The b witnesses do not ordinarily hear about operations or keep state.
  Why is that OK?
    b+1 of 2b+1 do have state
    So any b failures leaves at least one live copy of state.
  Why are the b witnesses needed at all?
    If b replicas with state do fail, witnesses give the required b+1 majority.
    To ensure that only one partition operates -- no split brain.
  So, for a 3-server system, the witness is there to break ties about
    which partition is allowed to operate when primary and backup are
    in different partitions. The partition with the witness wins.

Does primary need to send operations to witnesses?
  The primary must collect ACKs from a majority of the 2b+1 for every r/w operation.
    To ensure that it is still the primary -- still in the majority partition.
    To ensure that operation is on enough servers to intersect with any
      future majority that forms a new view.
  If all backups are up, primary+backups are enough for that majority.
  If m backups are down:
    Primary must talk to m "promoted" witnesses to get its majority for each op.
    Those witnesses must record the op, to ensure overlap with any
      future majority.
    Thus each "promoted" witness keeps a log.

So in a 2b+1 system, a view always has b+1 servers that the primary
  must contact for each op, and that store each op.

Note: somewhat different from Raft
  Raft keeps sending each op to all servers, proceeds when majority answer
    So leader must keep full log until failed server re-joins
  Harp eliminates failed server from view, doesn't send to it
    Only witness has to keep a big log; has special plan (ram, disk, tape).
  The bigger issue is that it can take a lot of work to bring
    a re-joining replica up to date; careful design is required.

What's the story about the UPS?
  This is one of the most interesting aspects of Harp's design
  Each server's power cord is plugged into a UPS
  UPS has enough battery to run server for a few minutes
  UPS tells server (via serial port) when main A/C power fails
  Server writes dirty FS blocks and Harp log to disk, then shuts down

What does the UPS buy for Harp?
  Efficient protection against A/C power failure of ALL servers
  For failures of up to b servers, replication is enough
  If *all* servers failed and lost state, that's more than b failures,
    so Harp has no guarantee (and indeed no state!)
  With UPS:
    Each server can reply *without* writing disk!
    But still guarantees to retain latest state despite simultaneous power fail
  But note:
    UPS does not protect against other causes of simultaneous failure
    e.g. bugs, earthquake
    Harp treats servers that re-start after UPS-protected crash differently
      than those that re-start with crash that lost in-memory state
    Because the latter may have forgotten *committed* operations

Larger point, faced by every fault-tolerant system
  Replicas *must* keep persistent state to deal w/ failure of all servers
    Committed operations

    Latest view number, proposal number, &c
  Must persist this state before replying
  Writing every commit to disk is very slow!
    10 ms per disk write, so only 100 ops/second
  So there are a few common patterns:
    1. Low throughput
    2. Batching, high delay
    3. Lossy or inconsistent recovery from simultaneous failure
    4. Batteries, flash, SSD w/ capacitor, &c

Let's talk about Harp's log management and operation execution
  Primary and backup must apply client operations to their state
  State here is a file system -- directories, file, owners, permissions, &c
  Harp must mimic an ordinary NFS server to the client
    i.e. not forget about ops for which it has sent a reply

What is in a typical log record?
  Client's NFS operation (write, mkdir, chmod, &c)
  Shadow state: modified i-nodes and directory content *after* execution
  Client RPC request ID, for duplicate detection
  Reply to send to client, for duplicate detection

Why does Harp have so many log pointers?
  FP most recent client request
  CP commit point (real in primary, latest heard in backup)
  AP highest update sent to disk
  LB disk has finished writing up to here
  GLB all nodes have completed disk up to here

Why the FP-CP gap?
  So primary doesn't need to wait for ACKs from each backup
    before sending next operation to backups
  Primary pipelines ops CP..FP to the backups.
  Higher throughput if concurrent client requests.

Why the AP-LB gap?
  Allows Harp to issue many ops as disk writes before waiting for disk
  The disk is more efficient if it has lots of writes (e.g. arm scheduling)

What is the LB?
  This replica has everything <= LB on disk.
  So it won't need those log records again.

Why the LB-GLB gap?
  GLB is min(all servers' LBs).
  GLB is earliest record that *some* server might need if it loses memory.

When does Harp execute a client operation?
  There are two answers!
  1. When operation arrives, primary figures out exactly what should happen.
     Produces resulting on-disk bytes for modified i-nodes, directories, &c.
     This is the shadow state.
     This happens before the CP, so the primary must consult recent
     operations in the log to find the latest file system state.
  2. After the operation commits, primary and backup can apply it to
     their file systems.
     They copy the log entry's shadow state to the file system;
     they do not really execute the operation.
     And now the primary can reply to the client's RPC.

Why does Harp split execution in this way?

```
  If a server crashes and reboots, it is brought up to date by replaying
  log entries it might have missed. Harp can't know exactly what the
  last pre-crash operation was, so Harp may repeat some. It's not
  correct to fully execute some operations twice, e.g. file append.
  So Harp log entries contain the *resulting* state, which is what's applied.

The point: multiple replay means replication isn't transparent to the service.
  Service must be modified to generate and accept the state
    modifications that result from client operations.
  In general, when applying replication to existing services,
    the service must be modified to cope with multiple replay.

Can Harp primary execute read-only operations w/o replicating to backups?
  e.g. reading a file.
  Would be faster -- after all, there's no new data to replicate.
  What's the danger?
  Harp's idea: leases
  Backups promise not to form a new view for some time.
  Primary can execute read-only ops locally for that time minus slop.
  Depends on reasonably synchronized clocks:
    Primary and backup must have bounded disagreement
    on how fast time passes.

What state should primary use when executing a read-only operation?
  Does it have to wait for all previously arrived operations to commit?
    No! That would be almost as slow as committing the read-only op.
  Should it look at state as of operation at FP, i.e. latest r/w operation?
    No! That operation has not committed; not allowed to reveal its effects.
  Thus Harp executes read-only ops with state as of CP.
  What if client sends a WRITE and (before WRITE finishes) a READ of same data?
    READ may see data *before* the WRITE!
    Why is that OK?

How does failure recovery work?
   I.e. how does Harp recover replicated state during view change?

Setup for the following scenarios
  5 servers: 1 is usually primary, 2+3 are backups, 4+5 witnesses

Scenario:
  S1+S2+S3; then S1 crashes
  S2 is primary in new view (and S4 is promoted)
  Will S2 have every committed operation?
  Will S2 have every operation S1 received?
  Will S2's log tail be the same as S3's log tail?
  How far back can S2 and S3 log tail differ?
  How to cause S2 and S3's log to be the same?
    Must commit ops that appeared in both S2+S3 logs
    What about ops that appear in only one log?
      In this scenario, can discard since could not have committed
      But in general committed op might be visible in just one log
  From what point does promoted witness have to start keeping a log?

What if S1 crashed just before replying to a client?
  Will the client ever get a reply?

After S1 recovers, with intact disk, but lost memory.
  It will be primary, but Harp can't immediately use its state or log.
    Unlike Raft, where leader only elected if it has the best log.
    Harp must replay log from promoted witness (S4)
  Could S1 have executed an op just before crashing
```

```
    that the replicas didn't execute after taking over?
    No, execution up to CP only, and CP is safe on S2+S3.

New scenario: S2 and S3 are partitioned (but still alive)
  Can S1+S4+S5 continue to process operations?
    Yes, promoted witnesses S4+S5
  S4 moves to S2/S3 partition
  Can S1+S5 continue?
    No, primary S1 doesn't get enough backup ACKs
  Can S2+S3+S4 continue?
    Yes, new view copies log entries S4->S2, S4->S3, now S2 is primary
  Note:
    New primary was missing many committed operations
    In general some *committed* operations may be on only one server

New scenario: S2 and S3 are partitioned (but still alive)
  S4 crashes, loses memory contents, reboots in S2/S3 partition
  Can they continue?
    Only if there wasn't another view that formed and committed more ops
  How to detect?
    Depends on what S4's on-disk view # says.
    OK if S4's disk view # is same as S2+S3's.
      No new views formed.
      S2+S3 must have heard about all committed ops in old view.

Everybody suffers a power failure.
  S4 disk and memory are lost, but it does re-start after repair.
  S1 and S5 never recover.
  S2 and S3 save everything on disk, re-start just fine.
  Can S2+S3+S4 continue?
  (harder than it looks)
    No, cannot be sure what state S4 had before failure.
    Might have formed a new view with S1+S5, and committed some ops.

When can Harp form a new view?
  1. No other view possible.
  2. Know view # of most recent view.
  3. Know all ops from most recent view.
  #1 is true if you have n+1 nodes in new view.
  #2 is true if you have n+1 nodes that did not lose view # since last view.
    View # stored on disk, so they just have to know disk is OK.
    One of them *must* have been in the previous view.
    So just take the highest view number.
  And #3?
    Need a disk image, and a log, that together reflect all operations
      through the end of the previous view.
    Perhaps from different servers, e.g. log from promoted witness,
      disk from backup that failed multiple views ago.

Does Harp have performance benefits?
  In Fig 5-1, why is Harp *faster* than non-replicated server?
  How much win would we expect by substituting RPC for disk operations?

Why graph x=load y=response-time?
  Why does this graph make sense?
  Why not just graph total time to perform X operations?
  One reason is that systems sometimes get more/less efficient w/ high load.
  And we care a lot how they perform w/ overload.

Why does response time go up with load?
  Why first gradual...
```

Queuing and random bursts?
And some ops more expensive than others, cause temp delays.
Then almost straight up?
Probably has hard limits, like disk I/Os per second.
Queue length diverges once offered load > capacity