AFS Local Protocol Enhancements

Andrew Deason

June 2022

OpenAFS Workshop 2022

Motivation

- OpenAFS work is generally all public (gerrit, github)
- Some fixes require protocol changes
- Standardizing protocol changes is very slow
  - Decades of compatibility
- ∴ Some changes become site-local
  - Often non-public
  - Not advertised much
  - Small, self-contained

DPF

Disposable Protocol Framework "DPF"

- Previous talk in 2019[1]
- Problems:
  - Wire speed too slow
  - rxkad encryption too slow
- Existing solutions:
  - TCP
  - TLS
  - AES
  - Compression
- DPF: Don't alter Rx, just change the RPCs
  - 'Disposable' protocols

[1] https://workshop.openafs.org/afsbpw19/2019/schedule/faster-wan-volume-operations-with-dpf/

Disposable Protocol Framework "DPF"

- Plugin framework for data streams
  - Self-registration, string/domain-based
  - `lz4:eof:fcrypt:rx`
  - `lz4:eof:fcrypt:tcp`
  - `lz4:eof:tls:tcp`
  - Actual mech name:

    `net.sinenomine.crypt.tls12psk.des.PSK-AES128-CBC-SHA.openssl.client`

- Mechs in production:
  - TCP (libevent)
  - TLS PSK (openssl)
  - lz4

- OpenAFS integrations in production:
  - volserver
  - i.e., volume moves, releases, dumps, restores

Client→Fileserver UID/PID

Client→Fileserver UID/PID

- Problem: Who is accessing `vol.obsolete` ?
- From fileserver auditlog:
  - Authenticated user
  - Client IP
  - Time, FID, ...
  - afs-client-accessd[2]
- For example: user: `--UnAuth--`, host: `192.0.2.43`
- Who is that?
  - On the client itself: `strace`, `inotify`, detailed audits
  - What a hassle!

[2] https://github.com/openafs-contrib/afs-tools/tree/master/admin/afs-client-accessd

Client→Fileserver UID/PID

- Solution: add process info to client requests
  - Replace RPCs: `FetchStatus`, `FetchData64`, etc.
  - Different parallel site-local Rx service
- Fileserver auditlog: process local unix uid, pid
- For example:
  - UID: 33 (`www-data`)
  - PID: 1098 (`nginx`)
  - Process accounting, logs
- Not for general use (off by default)
- Fixed `afs_uint32` fields
  - Maybe more in the future?

Rapid PAGs

Rapid PAGs

- Scenario: clients with many short-lived PAGs
  - Example: Web CGI with per-request `pagsh`/`kinit`/`aklog`
- Every time you `pagsh -c 'kinit && aklog && touch'`:
  1. Client creates a new security context
  2. Client creates a new connection to fileserver
  3. Fileserver resolves groups via ptserver
  4. Fileserver asks client for ID (via a separate RPC: `WhoAreYou`, `TellMeAboutYourself`)
- Problem: That last one is a big performance hit
  - Clients don't always respond quickly
  - Fileserver throttles requests
  - `CallPreamble: Couldn't get client while handling request from host ...`
  - `afs: Waiting for busy volume ...`

Rapid PAGs

- Solution: Have client send ID in advance
  - Security class? (rxgk, rxclear)
  - Smaller workaround: issue a separate RPC (just once per conn)
- New RPC: `PreloadClientInfo`
  - Site-local Rx service
  - Same info as `TellMeAboutYourself`
- Covers vast majority of cases, but not 100%
- Client delays effectively gone

Bulk RPCs

Bulk RPCs

- Fileserver `RXAFS`:
  - `FetchStatus` for 1 file: fine
  - `FetchStatus` for 20 files: slow (20x RTT)
  - `BulkStatus`, `InlineBulkStatus`
- ubik: `Write` $\rightarrow$ `WriteV`
- What if we could say “run these 5 RPCs at once”?

Bulk RPCs

- "rxbulk" subsystem
  - No changes to Rx
  - Changes to `rxgen`
  - New per-service RPC for the "bulk handler" (`BulkCall`)

- Strategy:
  1. Client writes RPC opcodes and `IN` args
     1.1 Write $inargs_1$
     1.2 Write $inargs_2$
  2. Client receives all `OUT` args
     2.1 Read $outargs_1$
     2.2 Read $outargs_2$

- No '`split`' RPCs

Bulk RPCs

Code Example

```
/* Sequential */
code = DISK_Write(conn, &tid, ...);
code = DISK_Write(conn, &tid, ...);
code = DISK_Commit(conn, &tid, ...);

/* Bulk */
code = rxbulk_init(&bulk, ...);
code = rxbulk_DISK_Write(bulk, &tid, ...);
code = rxbulk_DISK_Write(bulk, &tid, ...);
code = rxbulk_DISK_Commit(bulk, &tid, ...);
code = rxbulk_runall(bulk, rxconn, ...);
```

Bulk RPCs

- Downsides:
  - Arbitrary "bulk handler" RPC
  - No common args, context
  - Client-side RPC stats are tricky, for example:
  - Run 3 RPCs at once:
    - 2 `DISK_Writes` take 1ms
    - 1 `DISK_Commit` takes 1sec
    - Bulk call takes 1.002 sec
  - Who gets what?

- Used in experimental KV branch[3]

[3] https://github.com/adeason/openafs/tree/adeason/vldb4-kv

Postamble

Contact

adeason@dson.org

adeason@sinenomine.net

Slides

http://dson.org/talks