Leveraging Fast Forward Collectives To Improve Lustre* RAS

*Other names and brands may be claimed as the property of others.

RAS requirements

Robust handling of client, server, router and network failure

- Functionally transparent
- Minimal performance impact
  - Accurate, prompt failure diagnosis
  - Fast server fail-over & client fail-out

Scalability

- 100,000s clients
- 1,000s servers

Simple to administer

- Simple rules governing health and policy

Proposed RAS improvements

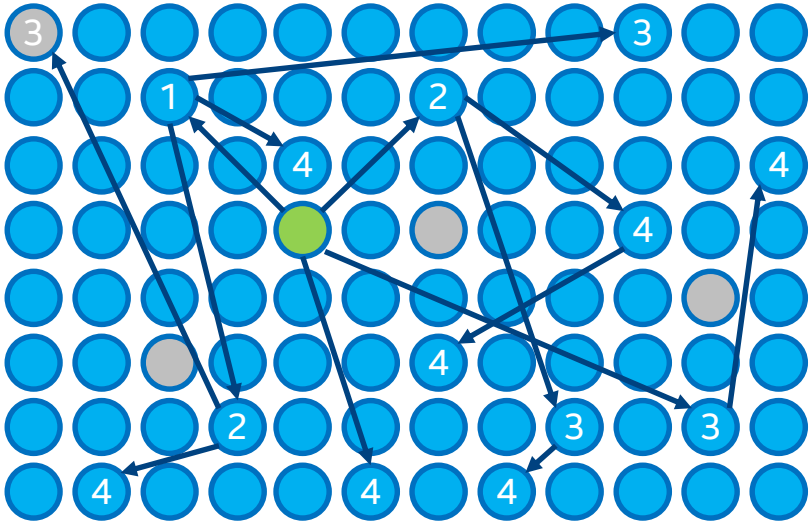
Primary fault diagnosis based on resilient collective health protocol

- Independent of storage service latency
- Scalable
  - Conservative
  - Prompt notification
- Simplified “catch-all” secondary diagnosis

Separate network & peer fault handling

- Simple retry on network failure
- Full recovery on peer failure

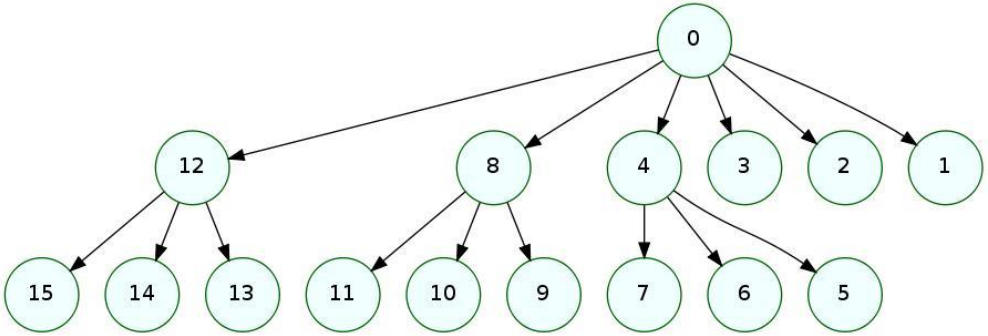
Fast Forward Collectives



Gossip

- Peer health monitoring
- Fault tolerant O(log n) state distribution
  - Every round pick 1 random peer to update
  - Vector of “last known alive” lamport clocks
  - Peer alive == recent clock tick
- Query & notification APIs
  - node_status_t gsp_query(lnet_nid_t node);
  - int gsp_register(const lnet_nid_t* nodes, size_t n, callback_func);

Fast Forward Collectives



Collective RPC

- Arbitrary membership
  - K-ary spanning trees over nodes & multiple members per NID (e.g. OSTs)
  - Membership propagates root->leaves with request
  - Single-shot & persistent groups
- Fast fail on member failure
- Idempotent – repeat until successful
- General purpose
  - Callbacks at all stages of processing
  - Simple to aggregate bulk incrementally or at root

Definitions & Assumptions

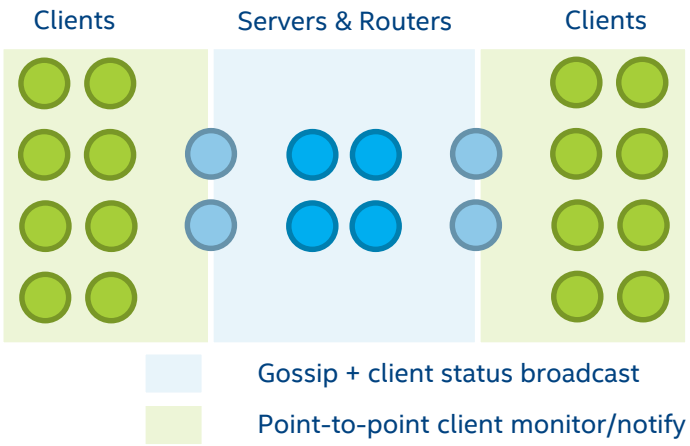
Definition

- RPC step: RPC request / bulk / RPC reply

Assumptions

- Reasonably bounded maximum network latency (seconds)
  - Strict bounds on RPC request length and # in-flight per client
    - Known maximum number of clients
    - Requests buffered eagerly at servers
  - Bounded # bulks & RPC replies in-flight across all servers
- Network communications with a live peer succeed 99.99% of the time

Health Monitoring



Gossip [targets][routers] health

- Direct notification of peer MDT, OST and router health
- 1,000 servers, 10 targets/server, 1,000 routers
  - 4Hz gossip frequency => 15s gossip latency

Distributed client monitoring & notification

- Load balance client monitoring over directly connected routers/servers
  - Monitors propagate client presence to servers and server status to clients
  - Clients may establish server connections after presence propagated
- Periodically aggregate & broadcast client health info
  - MDT0 initiates collectives
  - Collectives broadcast client status
    - Deltas normally / full client map when peers restart

Network fault handling

Make all RPC steps a round-trip

- Require ACK for RPC request, bulk PUT, RPC reply (bulk GET has REPLY anyway)

Make all RPC steps idempotent

- Discard duplicate RPC requests
- Register permanent match-all ME on bulk & RPC reply portals
  - Matched last to guarantee ACK/REPLY for duplicate

Retry active RPC steps

- Promptly on notification of router failure
- After timeout (max congested network latency)
  - Catch-all for point-to-point network failure

Peer fault handling

Assume peer healthy until notified otherwise

- Network fault handling deals with router & network failures
  - Robust lock callbacks
- Large Fixed Timeouts catch complete network failure & zombies
  - Major code simplification possible
- Mitigate zombies via internal health checks

Global client eviction

- Notify peers on decision to evict using same mechanism as client monitors
- Require client to reconnect to monitor before establishing new server connections
- Simplify SOM etc.

Summary

Health monitoring

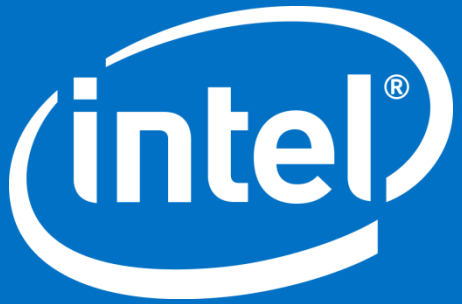
- Leverage Fast Forward Collectives
- Primary diagnosis of peer failure

Round-trip RPC steps

- Positive handshake on successful communications

Clear separation of network v. peer failure handling

- Robust communications in the face of network/router failure
- Code simplification


intel®