NFSv4.1 Sessions

Design and Linux Server Implementation Experiences

Jonathan Bauman
Center for Information Technology Integration
University of Michigan, Ann Arbor
http://citi.umich.edu
Sessions Overview

* Correctness
  * Exactly Once Semantics
  * Explicit negotiation of bounds
    * Clients make best use of available resources
  * 1 client, many sessions
    * /usr/bin (read only, no cache, many small requests)
    * /home (read/write, cache, fewer, larger requests)
* Client-initiated back channel
  * Eliminates firewall woes
  * Can share connection, no need to keep alive
Example of 4.0 Complexity

SETCLIENTID implementation discussion from RFC 3530

The server has previously recorded a confirmed \{ u, x, c, l, s \} record such that \( v \neq u, l \) may or may not equal \( k \), and recorded an unconfirmed \{ w, x, d, m, t \} record such that \( c \neq d, t \neq s, m \) may or may not equal \( k \), \( m \) may or may not equal \( l \), and \( k \) may or may not equal \( l \). Whether \( w = v \) or \( w \neq v \) makes no difference.

The server simply removes the unconfirmed \{ w, x, d, m, t \} record and replaces it with an unconfirmed \{ v, x, e, k, r \} record, such that \( e \neq d, e \neq c, r \neq t, r \neq s \).

The server returns \{ e, r \}.

The server awaits confirmation of \{ e, k \} via SETCLIENTID_CONFIRM \{ e, r \}. 
Sessions Overview (continued)

* **Simplicity**
  * CREATECLIENTID, CREATESESSION
    * Eliminate callback information
  * Duplicate Request Cache
    * Explicit part of protocol
    * New metadata eases implementation; RPC independent
    * See implementation discussion

* **Support for RDMA**
  * Reduce CPU overhead
  * Increase throughput
  * See NFS/RDMA talks for more
Draft Issues

* False Starts
  * Channels & Client/Session Relationship
  * Chaining

* Open Issues
  * Lifetime of client state
  * Management of RDMA-specific parameters

* Future Directions
  * “Smarter” clients & servers
  * Back channel implementation
Channels

* Originally, sessionid ≈ clientid;
  1 session, many channels
* Direct correspondence to transport instance
  * Back & operations channels are similar
  * Same BINDCHANNEL operation
* Protocol Layering Violation
  * ULP should be insulated from transport
  * Killer use case: Linux RPC auto-reconnects
  * Lesson: layering violations & LLP assumptions
Channels (continued)

* Now clientid:sessionid is 1:N
  * Per-channel control replaced by per-session
  * Sessions can be accessed by any connection
    * Facilitates trunking, failover
  * No layering violations on forward channel
* Back channel still bound to transport
  * Only way to achieve client-initiated channel
  * Layering violation, not required feature
  * Not yet implemented, possibly more to learn
**Chaining Example**

**NFS v4.0**

Allows COMPOUND procedures to contain an arbitrary number of operations.

**NFS v4.1 Sessions**

Since the maximum size of a COMPOUND is negotiated, arbitrarily large compounds are not allowed. Instead, COMPOUNDS are “chained” together to preserve state.

<table>
<thead>
<tr>
<th>COMPOUND 1</th>
<th>COMPOUND m</th>
<th>COMPOUND n</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAIN: BEGIN</td>
<td>CHAIN: CONTINUE</td>
<td>CHAIN: END</td>
</tr>
<tr>
<td>OPERATION 1</td>
<td>OPERATION i + 1</td>
<td>OPERATION j + 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OPERATION i</td>
<td>OPERATION j</td>
<td>OPERATION k</td>
</tr>
</tbody>
</table>

*
Chaining

- Max request size limits COMPOUND
  - 4.0 places no limit on size or # of operations
  - File handles live in COMPOUND scope

- Originally sessions proposed chaining facility
  - Preserve COMPOUND scope across requests
  - Chain flags in SEQUENCE

- Chaining eliminated
  - Ordering issues across connections problematic
  - Annoying to implement and of dubious value
  - Large COMPOUNDS on 4.0 get errors anyway
  - Sessions can still be tailored for large COMPOUNDS
Implementation Challenges

- Constantly changing specification
  - Problem for me, but not for you
  - Time implementing dead-end concepts
- Fast pace of Linux kernel development
  - Difficulty merging changes from 4.0 development
- Lack of packet analysis tools
- SEQUENCE operation
  - Unlike other v4 operations
  - Requires somewhat special handling
- Duplicate Request Cache
Duplicate Request Cache

- No real DRC in 4.0; Compare to v3.0 (on Linux)
  - Global scope
    - All client replies saved in same pool
    - Unfair to less busy clients
  - Small
    - Unlikely to retain replies long enough
    - No strong semantics govern cache eviction
- General DRC Problems
  - Nonstandard and undocumented
  - Difficult to identify replay with IP & XID
4.1 Sessions Cache Principles

- Actual part of the protocol
  - Clients can depend on behavior
  - Increases reliability and interoperability
- Replies cached at session scope
  - Maximum number of concurrent requests & maximum sizes negotiated
- Cache access and entry retirement
  - Replays unambiguously identified
  - New identifiers obviate caching of request data
  - Entries retained until explicit client overwrite
DRC Initial Design

* Statically allocated buffers based on limits negotiated at session creation

* How to save reply?
  * Tried to provide own buffers to RPC, no can do
  * Start simple, copy reply before sending

* Killer problem: can’t predict response size
  * If reply is too large, it can’t be saved in cache
  * Must not do non-idempotent non-cacheable ops
  * Operations with unbounded reply size: GETATTR, LOCK, OPEN...
DRC Redesign

* No statically allocated reply buffers
* Add reference to XDR reply pages
  * Tiny cache footprint
  * No copies, modest increase in memory usage
  * Layering? This is just one implementation; Linux RPC is inexorably linked to NFS anyway
  * 1 pernicious bug: RPC status pointer
* Large non-idempotent replies still a problem
  * Truly hard to solve, given current operations
  * In practice, not a problem at all (rsize, wsize)
DRC Structures

### Session State
```
struct nfs4_session {
    /* other fields omitted */
    u32 se_maxreqsize;
    u32 se_maxrespsize;
    u32 se_maxreqs;
    struct nfs4_cache_entry *se_drc; }
```

### SEQUENCE Arguments
```
struct nfsd4_sequence {
    sessionid_t se_sessionid;
    u32 se_sequenceid;
    u32 se_slotid;
};
```

### Table: Session State and SEQUENCE Arguments

<table>
<thead>
<tr>
<th>Slot ID</th>
<th>Sequence ID</th>
<th>Status</th>
<th>XDR Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>complete</td>
<td>0xBEEFBE10</td>
</tr>
<tr>
<td>1</td>
<td>286</td>
<td>in-progress</td>
<td>0xDECABF0010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>maxreqs - 1</td>
<td>0</td>
<td>available</td>
<td>0x0000000000</td>
</tr>
</tbody>
</table>
DRC Fringe Benefit

* 4.0 Bug: Operations that generate upcalls
  * Execution is deferred & revisited (pseudo-drop)
  * Partial reply state not saved
  * Non-idempotent operations may be repeated

* Sessions Solution
  * When execution is deferred retain state in DRC
  * Only additions are file handles & operation #
  * Revisit triggers DRC hit, execution resumes
DRC Future

* Refinement, stress testing
  * Compare performance to v3
  * Quantify benefits over stateful operation caching in 4.0
* Backport to v4.0
  * No session scope, will use client scope
  * No unique identifiers, must use IP, port & XID
  * More work, but significant benefit over v3
Implementation Delights

* Draft changes made for *better* code
  * DRC & RPC uncoupled
  * SETCLIENTID & SETCLIENTID_CONFIRM
* Relatively little code
  * CREATECLIENTID
  * CREATESESSION
  * DESTROYSESSION
  * SEQUENCE (Duplicate Request Cache)
Conclusions

✽ Basic sessions additions are positive
  ✽ Reasonable to implement
  ✽ Definite improvements: correctness, simplicity

✽ Layering violations
  ✽ Avoid in protocol
  ✽ Can be leveraged in implementation

✽ Further additions require more investigation
  ✽ Back channel
  ✽ RDMA
Questions & Other Issues

* **Open Issues**
  * Lifetime of client state
  * Management of RDMA-specific parameters

* **Future Directions**
  * “Smarter” clients & servers
  * Back channel implementation

* **RDMA/Sessions Draft**
  * Under NFSv4 Drafts at IETF site