Efficient Transactions for Parallel Data Movement

Jay Lofstead (SNL), Jai Dayal (GT), Ivo Jimenez (UCSC), Carlos Maltzahn (UCSC)

Sandia National Labs./Georgia Tech/UC Santa Cruz

gflofst@sandia.gov

Parallel Data Storage Workshop

November 18, 2013
A Little History

- Georgia Tech 2005ish
  - How do we distinguish different parallel outputs?
- LWFS 2006
  - Transaction support for file systems
- LDRD 2010
  - General parallel transactions for data movement and system reconfig
- Lustre/Intel FastForward 2012+
  - Epochs
- SNL/GT & SNL/UCSC 2013+
  - Exploring application scenarios and alternatives
Why Transactions?

- All-or-nothing operations
- Grouping operations into an atomic set
- Well understood semantic
- Challenge: M clients to N servers
First pass was a “Full” Protocol

Client and server “sides” different

Aggregate on each side to a single coordinator

Coordinator-to-coordinator communication for configuration and metadata

Invasive requirements on servers

Overall transaction and a collection of sub-transactions
Version 1.0 Performance

- Adding a second server is bad!
- Total overhead would reach several seconds at scale
D^2T Version 1.0 @ Cluster 2012

- **Positives**
  - Demonstrated one possibility for MxN transactions
  - Identified scaling bottlenecks

- **Negatives**
  - Multi-polling performance problems
  - Single point bottlenecks
    - Number and/or aggregate size of messages too big for a single node
Version 2.0 Changes

- Second aggregation level added
  - Solves message size/count problem
- Server requirements almost non-existent, but with a catch
  - How to do vote/commit without a little server support?
- Multi-protocol polling eliminated
- Vastly better performance!
Version 2.0 Changes

- Multiple roles for some processes
- 0 is coordinator, sub-coordinator, and subordinate
- 3 is sub-coordinator and subordinate
- 1, 2, 4, 5 are all just subordinates
- S1, S2, S3 are servers
Version 2.0 Global Knowledge

- Addressing failures requires global knowledge
  - Singleton sub-transactions
  - Global sub-transactions
  - Which processes are in which roles

- Must use a resilient protocol for communication or it all comes down
Version 2.0 Performance

- **Notes:**
  - Always used at least 2 sub-coordinators to slow it down
  - Added a sub-coordinator when subordinate count exceeded 256
  - 64K processes = 256 sub-coordinators with 256 subordinates each
  - Overhead only for complete set of transaction calls (no op. costs)
Detailed Performance Numbers

- 64K processes case
  - `txn_create_sub_transaction_all` maximum time 0.0310 seconds (mean 0.01)
  - All other transaction ops < 0.005 seconds mean (0.012 maximum)
  - Protocol Init/finalize 0.38/0.0002 seconds.
    - Similar to MPI_Init/MPI_Finalize

- Total time, worst case for each operation across all tests, for a transaction + sub-transactions start to finish < 0.45 seconds for 64K
Additional Features

- Fault detection
  - Overhead = timeout value + typical operation time
  - Targeting HPDC 2014 deadline

- Minimal metadata and data storage services as examples
  - No performance tuning, error checking, or scalability considerations
Next Steps

- UCSC/SNL transaction spectrum project

- GT/SNL use for “containers” project

- SNL use for data staging/in transit processing/code coupling

- Working with Intel/Lustre FastForward team to help inform their effort
Questions

Jay Lofstead

gflofst@sandia.gov