Efficient, Failure Resilient Transactions for Parallel and Distributed Computing

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

Scalable System Software
Sandia National Laboratories
Albuquerque, NM, USA
gflofst@sandia.gov

DISCS Workshop at Supercomputing 2014

November 16, 2014
2014-0689C
Why Transactions?

- Yes, database-like distributed transactions
  - But doubly distributed (clients and servers)
- Motivation: how to bracket MxN operations?
- Special parallel considerations (superset of distributed)
  - Data sizes bigger than can fit in one node
  - O(1M) clients, O(1K) servers
  - Not just data, but operations too
Doubly Distributed Txns (D^2T)

- Naïve protocol problems
  - M times N messages for global knowledge
  - Aggregation runs into memory limits (message sizes and/or counts)
  - Requires active support by servers

- Optimized protocol
  - Tree aggregation on client side only
  - Assumption client can know if server was successful
D²T Performance

- Baseline total average overheads for full transaction with multiple sub-transactions
- Cielo (SNL/LANL Cray) performance
Fault Tolerance

- Basic approach is timeouts
  - Short/long (5/10 seconds)
  - Different for gather and broadcast at each level

- Consider each level
  - Coordinator
  - Sub-coordinator
  - Subordinates

- Handle cascading timeouts
  - Handling a failure does not cause a false failure elsewhere

- Maintain global knowledge
  - Any process to any role
Subordinate Failure Recovery

- Simplest case
- Detect at sub-coordinator
- Short timeout
- Spread knowledge, but no reconfiguration necessary
Sub-Coordinator Failure Recovery

- Hardest case
- Detect at subordinates
- Self promote within sub-group
- Notify around
- Avoid cascading failures in other sub-coordinator trees
Coordinator Failure Recovery

- A special case of sub-coordinator failure
- Detect at subordinate level
- Promote new process
- Globally Notify
  - Short-circuits potential cascading false faults
Overall D²T Picture

- D²T overhead in normal case small, in failure case, can be barely larger
- Incorporate third-party services if success/failure can be determined
- Fairly straightforward API
- No centralized coordinating server required

How difficult to incorporate third-party tools?
Example Operating Model

- Consider data staging example
- Ideally has metadata and data storage services to make multiple staging areas
- Use transactions across multiple services atomically
  - e.g., Operation sequence: Analysis marks in progress, reads from Staging A, processes, writes to Staging B, deletes from Staging A
Example Generic Staging Services

- Datastore Service
- Metadata Service
- Performance not a consideration
Datastore Service

- Simple object store

- Adding transactions support
  - Support “data hiding” until txn committed
  - Support rollback (delete)
  - Maintain visibility for transaction participants

- Wrapper around generic service
  - API wrapper to add txn id field and filter for visibility
  - Everything else pass through
Metadata Service

- Distributed array database
  - Keep track of chunks that comprise the whole 3-D (n-dimensional) space
  - Maintain metadata about type and name information for whole and parts

- Adding transaction support
  - Support hiding multiple items as a group behind txn id
  - Support rollback cleanup and commit visibility

- Wrapper around service
  - API wrapper to add txn id field and filter for visibility
  - Everything else pass through
Using D²T With 3rd Party Services

- API wedge easy to add, minimal performance impact
- Additional data requirements generally small

- Datastore overhead unsurprisingly small

- Metadata overhead large percentage of base data size

- Bottom line:
  - Additional data volume not large and adding a wedge around the service not difficult. No need for services to support transactions directly to participate.
Questions?

Email:
gflofst@sandia.gov

Papers and code (shortly):
http://www.lofstead.org/txn