Tango: Distributed Data Structures over a Shared Log

Alex Roberts

(Slides adapted from SOSP 2013 presentation)
Handling Metadata

• Distributed Data
  • Amazon S3
  • Azure Blob Store
• Centralized Metadata
  • Mapreduce Scheduler
  • Resource Allocation Tables
  • etc
CENTRALIZING downsides

• Single Point of Failure
• Solve by “hardening”
  • Make highly available
  • Standard Practice
• ...This can be difficult though
  • Re-write code
CENTRALIZED PROPERTIES

• Centralized services looks different than highly available services
  • Usually utilize RPC
  • In-memory data structures
  • Centralizing simplifies transaction access
    Simple locking
HIGHLY AVAILABLE PROPERTIES

- Requires different abstractions
- Can utilize external service Zookeeper / Chubby
  - Can require code rewrite
  - Only provides single data-structure
- SMR with Paxos
  - Requires writing program as state machine
Goal of Tango

• Provide “Centralized” metadata store
• In-memory data structures
• ...but be highly available and not require a code rewrite
Tango Overview

- Provide Tango Objects
- In-memory data structure
- Backed by shared log
Tango Properties

- Persistance
  - Objects and data structures can be rebuilt from shared log
  - Shared log holds “real” state
  - Views hold soft state
TANGO PROPERTIES

• High Availability
  • Hold multiple views of the same object on several different machines
• Objects synchronized with shared log on use
• Linearizability as a result
Tango Properties

- Elasticity
  - Initialize by reading shared log
  - Don’t need to interact with other clients
Tango Properties

• Transaction Semantics
• Provided by CORFU with Commit Records
• Gives Atomicity and Isolation
**Tango Properties**

- Messages only between Shared Log and Clients
- Clients never communicate to each other
Tango Objects

- On the surface look like regular objects
- Easy to instantiate and use
- Tango provides linearizability for single objects
curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)
curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)
EXAMPLE

Client

curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)

Under the hood

Updates since last read
`curowner = presmap.get("slides")`
```
If (curowner == me)
    slides.add(slidedeck)
```
curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)
curowner = presmap.get("slides")
if (curowner == me):
    slides.add(slidedeck)
**Example - Transactions**

Client

BeginTX()

curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)

EndTX()

Under the hood
Example - Transactions

Client

BeginTX()

curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)

EndTX()

Under the hood
EXAMPLE - TRANSACTIONS

Client

BeginTX()

curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)

EndTX()
Example - Transactions

Client

BeginTX()

curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)

EndTX()

Under the hood
BeginTX()
curowner = presmap.get("slides")
If (curowner == me)
    slides.add(slidedeck)
EndTX()
EXAMPLE - TRANSACTIONS

Client

BeginTX()

curowner = presmap.get("slides")
If (curowner == me)
   slides.add(slidedeck)

EndTX()

Under the hood

Commit or Abort?
Did red client modify any of the reads?
```java
BeginTX()

cuowner = presmap.get("slides")
If (cuowner == me)
    slides.add(slidedeck)

EndTX()
```
**Tango Objects**

- Tango provides several “out of the box” data structures
- API provides way to create your own
- Data structures / Objects Tango API
  - update_helper - appends log
  - query_helper - reads updates from log
  - apply
Tango Object Use Cases

- Replicating State
  - Clients A and B have exact same state
- Indexing State
  - Different representation of the same data
- Partitioning State
  - Many clients, objects don’t overlap
- Sharing State
  - Many clients, objects overlap
Replicating State

- Clients A and B have same state
- Ex. Job scheduling service across multiple application servers
INDEXING STATE

• Different representation of the same data
• Ex. same data in a tree-based map and hash-based map
Clients A and B use different data structures.

Partitioning State
• Clients A and B use overlapping data structures
• This can cause issues!
Multiple Tango Objects

- Multiple data structures and objects appear in a system
- A combination of them may be used to complete one transaction (Shared State use case)
- How do Tango and Corfu handle this?
  - Stream abstraction
  - Decision Records
MULTIPLE TANGO OBJECTS - ISSUES
MULTIPLE TANGO OBJECTS - ISSUES
Multiple Tango Objects - Issues
MULTIPLE TANGO OBJECTS - ISSUES
MULTIPLE TANGO OBJECTS - ISSUES

Must read all these updates
MULTIPLE TANGO OBJECTS - ISSUES

readnext(streamid)

append(value, streamid1, ...)
STREAM ABSTRACTION

Stream A

0 → 1 → 3 → 4 → 7

Stream B

0 → 2 → 5 → 6 → 7

Shared Log Backpointers:

0 1 2 3 4 5 6 7 ...
Multiple Tango Objects - Issues
MULTIPLE TANGO OBJECTS - ISSUES
MULTIPLE TANGO OBJECTS - ISSUES

Commit Or Abort?
Has blue been modified?
Abort
MULTIPLE TANGO OBJECTS - ISSUES

Commit Or Abort?
Has blue been modified?
Abort

Commit Or Abort?
Has blue been modified?
Unsure
MULTIPLE TANGO OBJECTS - ISSUES
EVALUATION

- System consisted of 36 8-core machines across two racks
  - Gigabit NICs
  - 20 Gbps switches
- 18 Node CORFU deployment
  - 32 core sequencer on separate rack
EVALUATION - SINGLE OBJECT

The graph shows the request latency (ms) against the number of requests per second (Ks) for different write ratios from 1.0 to 0.0. Each line represents a different write ratio, with 1.0 being the highest and 0.0 being the lowest. The x-axis represents the number of requests per second ranging from 0 to 175, and the y-axis represents the request latency ranging from 0 to 6 milliseconds.
EVALUATION - SINGLE OBJECT

Read Latency

MILLISECONDS

Ks/sec

TARGET WRITES (Ks/sec)

TWO VIEWS
EVALUATION - SINGLE OBJECT

N Readers, 10K Writes/Sec

Multiple views
EVALUATION - TRANSACTIONS

- Goodput: committed transactions
EVALUATION - TRANSACTIONS

![Graph showing the evaluation of transactions with 18-server and 6-server logs. The x-axis represents the number of clients, ranging from 2 to 18, and the y-axis represents the number of transactions per second (Ks of Tx/sec), ranging from 0 to 250. The graph compares the performance of 18-server and 6-server logs under varying client loads.](image-url)
Evaluation - Transactions

![Graph showing transactions per second vs. percentage of cross-partition transactions for Tango and 2PL.]

- **Y-axis**: Ks of Txes/sec
- **X-axis**: % of Cross-Partition Txes

The graph compares the performance of Tango and 2PL in handling transactions. The x-axis represents the percentage of cross-partition transactions, and the y-axis shows the throughput in Ks of transactions per second. The bars for each percentage show the performance of both systems, with Tango generally performing better than 2PL at lower percentages but showing a decrease in performance as the percentage of cross-partition transactions increases.
EVALUATION - TRANSACTIONS
Other Data Structures

• Implemented ZooKeeper in ~1000 lines of code
  • Much less than the 13K in the original
• Implemented BookKeeper in ~300 lines of code
CONCLUSION

• Metadata services have been neglected, Tango fixes that by utilizing CORFU
• Tango objects: distributed objects backed by shared log
• Provides
  • Availability
  • Consistency
  • Durability
  • History
  • Elasticity