JUNO DISTRIBUTED CRYPTOLEDGER

CIB New Product Development

Will Martino, Stuart Popejoy, Brian Schroeder, Libby Kent

March 2016

J.P.Morgan

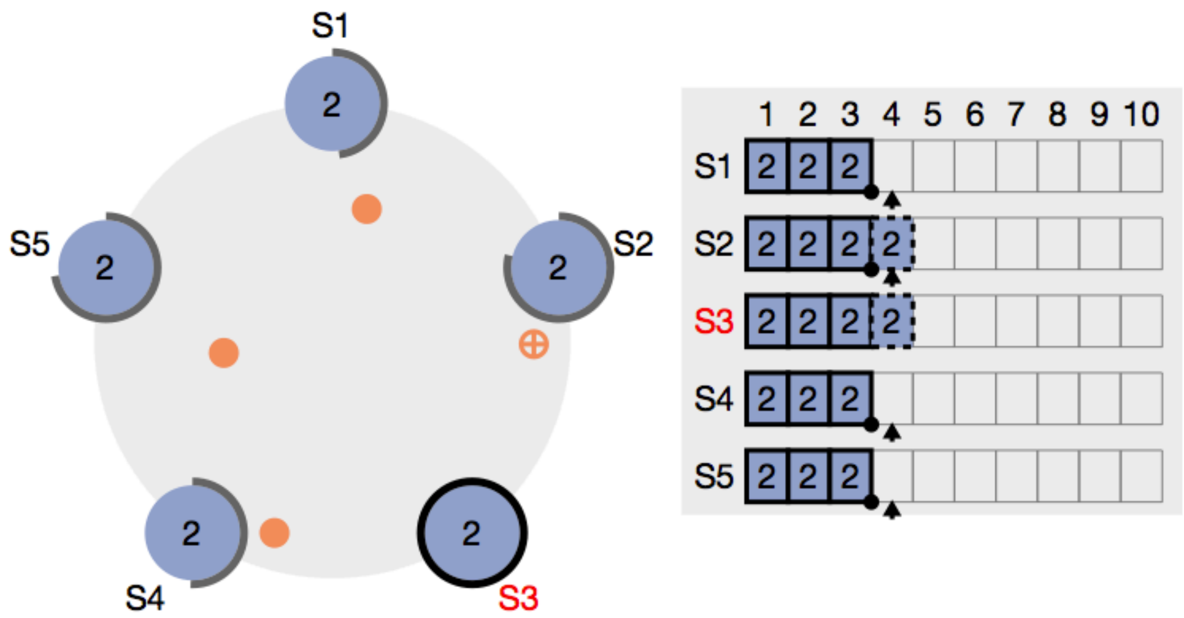
Agenda

- Consensus: Raft
- "BFT-Hardened" Tangaroa variant
- High performance
- Novel system properties
- Smart contracts: Hopper
- Persistence
- Privacy
- Community Contributions

Consensus: Raft

Why Raft?

- Simpler than Paxos, PBFT
- Consensus around inputs, not outputs
- Formally proven (Verdi Raft)
- Single-leader attractive for performance, system visibility, non-forking
- Effectively or explicitly write-behind – slow nodes do not slow entire system



"BFT-Hardened": Tangaroa Variant

Summary

- Tangaroa (Copeland/Zhang):
  - Complete Raft implementation with "BFT-hardening"
  - Lazy votes to prevent election "flutter"
  - Additional crypto signatures on all messages
- "BFT-Hardened" vs BFT:
  - Non-public network
  - "Full BFT" is application-specific
  - Tangaroa implements a Byzantine Fault Tolerance for *consensus* only
  - Application-specific Fault Tolerance can be implemented separately.
- Signed messages, known participants:
  - Multiple sets for keys for different traffic (Client to Cluster, Intra-Cluster, Admin to Cluster)
- Juno resolves significant issues with the original Tangaroa codebase and protocol.

High Performance

Juno offers considerable performance benefits over other solutions in the marketplace

- Un-optimized Juno instance running four nodes on a MacBook Pro:
  - Throughput of Consensus: 500/Sec
  - Latency of Consensus: ~2ms
  - Throughput of Hopper: ~1500 transfers/Sec
- NB: Expect 50msg/sec & 20ms latency for the open source version, mostly due to slow logging and synchronous program application. These issues are fixed in our internal (but not yet ready for release) version of Juno.
- Can comfortably forecast a 2-3x times performance increase with basic optimizations:
  - GC tuning
  - Batching Log Entries
  - Messaging Layer configuration
- In production:
  - Latency is a factor of the **longest network latency to a quorum node**
  - Node count has a small impact on throughput and latency
- Future work: "Write-behind" application execution, where long-running programs do not hamper consensus computation
- Amenable to gossip protocol

Novel system properties

Deterministic State Machine (DSM)

- Raft offers a general "state-machine stepper", need not be a deterministic state-machine
- Juno *supports* non-deterministic state-machines
- Juno *prefers DSMs* as they underwrite system integrity:
  - Juno *guarantees* deterministic ordering of executions
  - In DSMs inputs determine outputs
- If output of DSM is a diff this allows for cheap verification of emergent application state.
- Supports EVM (Masala)
- Supports integration with external state-machines

Single transaction blocks

- Hardened Raft establishes linearity: no forking
- Each transaction hashed against prior

Future Work: Application-Level Integrity

- Using a deterministic state machine and "diff-able" state, we may offer verifiable application state
  - Incremental hashing of state diffs
- Individual diffs verify entire state up to that point
- Currently weighing a few options:
  - Send hash in checkpoint application message and verify offline
  - Hash against Ledger hash for easy verification vs ledger (after replay, etc.)
  - Include state hash in Append Entries Response

Smart contracts: Hopper

Hopper Language

- Simple code for transactional execution using one of two surface languages:
- Informative error messages (“Account not found”, “Insufficient Balance”)
- Inherently transactional – no control flow, no observations of current state
- Fully deterministic, with “diff-able”, serializable outputs.

Future Work:

- Ownership models expressed within language (Linear Types)
- Module System
- Execution cost model

Surface Languages

Transfer Syntax

```
transfer (00273601 -> jpm -> 10456782, 100 % 1)
transfer (22347589 -> jpm -> 33457688, 100 % 1)
```

“Transmatic” LISP-like language

```
(let ((jpm "jpm")
      (xferThruJpm (lambda (from to amount)
                     (#transfer from jpm amount)
                     (#transfer jpm to amount)))
      (usd100 (% 100 1)))
  (xferThruJpm "00273601" "10456782" usd100)
  (xferThruJpm "22347589" "33457688" usd100))
```

Persistence

Persistence and DB options

- Ledger query:
  - Ranged Query: match particular Ledger transactions before/after/between Log Indexes
  - Clean Query: query undergoes consensus (committed) then responded
  - Dirty Query: query is responded to instantly, may be inflight Log Entry that impacts read
- Data in key-value format, can work with any key-value store
- Current Integrations for Log Storage:
  - SQLite

Privacy

Today

- Persist non-transactional encrypted data alongside transactional code.
- Implies external encryption shared between transacting parties outside system

Future work

Local Node Encryption/Decryption

- Allows local, trusted nodes to manage keys.

Local Node Encrypted Smart Contract Execution

- Run Hopper in a non-transactional mode -- “Show me the computed payments for this loan”, etc.

Community Contributions

Open Source Repositories

- Juno: https://github.com/buckie/juno
- Masala (EVM): https://github.com/slpopejoy/masala
- Hopper: https://github.com/hopper-lang/hopper