Nezha: A High-Performance Consensus Protocol Using Accurately Synchronized Clocks

Jinkun Geng, Anirudh Sivaraman, Balaji Prabhakar and Mendel Rosenblum
Background and Motivation

• Consensus protocols are widely used in practice
  • Coordination: Chubby, Zookeeper, etcd
  • Storage: BigTable, TiDB, CockroachDB, MongoDB, RedisRaft
  • Messaging: RabbitMQ, Apache Kafka
  • Blockchain-related: Ethereum, Hyperledger
  • DNS Server: CoreDNS
  • Container orchestration tool: Docker Swarm, Kubernetes
  • ...
Background and Motivation

• Popular and widely-deployed consensus protocols
  • Like Raft, Multi-Paxos, Fast Paxos, Epaxos are deployable anywhere
  • But they perform poorly

• Recently-developed protocols use network support to improve performance; e.g.,
  • Speculative Paxos (needs multicast + control over paths)
  • NOPaxos, NetChain (uses programmable switches)
  • Mu (runs on top of RDMA)
    → But network support is not available in virtualized environments like the public cloud

• Our goal:
  • Bridge the gap, especially in the public cloud where we can’t get network support
  • We use synchronized clocks to achieve this goal
Raft and NoPaxos

• The leader multicasts requests to followers and conducts quorum check

Raft

• The network provides **hardware multicast** and **mostly-ordered delivery**; the sequencer (programmable switch) tags sequential number to each request to enable **drop notification**
• The client conducts the quorum check

NOPaxos
We build **deadline-ordered multicast** (DOM) primitive, which uses synchronized clocks at both the clients and the replicas.

DOM tags each request with a deadline for **ordering**, and multicasts it to all replicas.

DOM releases the requests according to their deadlines, i.e. replicas append requests to their logs in the **same order**.

In case of a timeout (e.g., due to a packet drop or large delay), the client retries the request with a new deadline.
Nezha-Fast Path

1. DOM tags the deadline to the request.
2. DOM multicasts the request to all replicas, and it is accepted by the Early Buffer.
3. DOM releases the request according to its deadline and appends to log.
4. Leader executes the request.
5. Replicas compute a hash and reply to the client for quorum check.

$$hash_n = \bigoplus_{1 \leq i \leq n} h(request_i)$$
Nezha-Slow Path

1. DOM tags the deadline to the request
2. DOM multicasts the request to all replicas, and it is accepted by the Late Buffer
3. Leader modifies the request deadline to make it eligible to enter the Early Buffer
4. DOM releases the request from the leader’s Early Buffer
5. Leader executes the request
6. Leader replies to the client
7. Leader broadcasts its log indices to the followers
8. Followers modify their logs to keep consistent with the leader
9. Followers fetch missing logs from their Late Buffers
10. Followers reply to clients for quorum check
Evaluation

• 4 baselines: Multi-Paxos, Fast Paxos, NOPaxos, and Raft

• 2 types of tests
  • Closed-loop: each client only maintains 1 outstanding request
  • Open-loop: each client submits requests following a Poisson distribution

• 2 applications: Redis and CloudEx

• All experiments are conducted in Google Cloud
Comparison for diskless protocols (final copy in memory)

Closed-loop (Speedup: 1.8-19.9x)

Open-loop (Speedup: 2.3-15.5x)
Nezha vs. Raft (data persisted to disk)

- We convert Nezha to a disk-based protocol for fair comparison
- We tune the batch size (disk write) for each protocol to achieve their best performance
- Raft-1: Diego Ongaro’s implementation; Raft-2: Our implementation based on Multi-Paxos

![Graphs showing latency vs. throughput for Nezha vs. Raft in closed-loop and open-loop scenarios.](image-url)
Scalability with Replicas

Bottleneck at client side

- Client-side multicast and quorum check make the clients a throughput bottleneck, and add complexity to client-side logic
- We introduce **stateless** proxies between clients and replicas to mitigate clients’ burden
- The proxies also make Nezha a “drop-in replacement” for Raft

Throughput ($\times 1K$ reqs/sec) vs. Number of Replicas

- Multi-Paxos
- NOPaxos-Optim
- Nezha (10 Clients)
- Nezha (15 Clients)

Open-loop Test
Proxy

- We employ 5 proxies to serve 10 clients and 9 replicas
- The proxies reduce CPU costs for clients
- The proxies also provide latency benefit under high throughput load
- 20 closed-loop clients submit requests under 10ms SLO
- YCSB-A Workload (HMSET/HGETALL on 1000 keys)
- Nezha is only within 6% that of the unreplicated system
• 48 open-loop traders + 16 gateways + 1 matching engine
• Only Nezha saturates the matching engine processing capacity (~43K orders/sec) among the four protocols
• Nezha prolongs end-to-end latency by 19.7% but achieves very close order processing latency
Conclusion

• Nezha is a high-performance consensus protocol and can be easily deployed in public cloud

• Nezha relies on synchronized clocks for high performance but not for correctness

• Nezha outperforms the 4 baselines (Multi-Paxos, Fast Paxos, NOPaxos, Raft) by 1.8–19.9x in throughput, and by 1.2–2.2x in latency

• With a proxy, Nezha can serve as a drop-in replacement of Raft/Multi-Paxos
Thanks!

Jinkun Geng, Anirudh Sivaraman, Balaji Prabhakar and Mendel Rosenblum