Self-Programming Networks Status Update: CloudEx, On-Ramp, Nezha and COLA

Balaji Prabhakar and Mendel Rosenblum
Self-Programming Networks Research Group
Stanford University
Our Research: Timeliness

Our research related to Timeliness: Accurate clock sync and consequences
- Previously: Huygens clock sync, SIMON network measurement, Clockchain event ordering system, C-Raft consensus protocol, etc

Recent research
- CloudEx: Using time perimeters to build fair financial exchanges in the cloud
- On-Ramp: Using accurate one-way delay measurements to build “drop-free networks”
- COLA: Building better autoscalers for K8s using Reinforcement Learning
- Nezha: A high-performance consensus protocol using accurate clocks
Motivated by discussions with our partners at Nasdaq
• We developed CloudEx—a financial exchange in the cloud using “time perimeters”
→ See poster presentation by Jinkun Geng, et al on Thu

Meanwhile, our partners at Google and Nasdaq have taken huge steps to make Cloud Exchanges a reality!

"...Google Cloud unit would eventually power markets that handle trillions of dollars in trades each day."

“...This landmark partnership with AWS seeks to power a truly cloud-based market infrastructure that is more resilient, scalable, and accessible for all market participants,” said Adena Friedman, CEO, Nasdaq.
CloudEx: Recent Progress

Previous version published in HotOS’21\cite{1}

Order Processing Throughput: 61K orders/second
End-to-end Median Latency: ~1ms

Current version of CloudEx

Order Processing Throughput: 718K orders/second
End-to-end Median Latency: 173μs (with DPDK)
(Note: this is still with SW timestamps; with NIC timestamps, could be even faster)

Optimizations:
- Improved networking: UDP instead of TCP; kernel bypass with DPDK
- Fixed-length serialization format
- Lockless order processing

On-Ramp Overview

1. On-Ramp (OR) Sender and Receiver are clock-synced
2. Sender measures one-way delay (OWD) using packet and ack timestamps
3. If OWD > a threshold; then, Sender pauses flow briefly

More details in our NSDI ‘21 paper[1]

**Meta Testbed: On-Ramp vs DCTCP**

- **Joint work with:** Ahmad Ghalayini (*Stanford*), Shiyu Liu and Yilong Geng (*Clockwork*), Terry Lam (*Meta*), Abdul Kabbani (*Microsoft*), Mohamad Alizadeh (*MIT*)

- Using a synthetic traffic generator
  - Time-sensitive **incast flows**
  - Bandwidth-intensive **background flows**

- Algorithms compared
  - cubic, dctcp, OR+cubic (with both SW and HW timestamps)

![Graphs showing performance comparisons between Incast and Background flows]
**Meta Deployment in Production: On-Ramp with DCTCP**

- On-Ramp can also work underneath DCTCP and improve DCTCP’s performance

- **OR+DCTCP** in a production service deployed in a single datacenter at Meta
  - **Throughput** does not change
  - **Congestion discards** drop *3.6x* from 900 packets/s to 250 packets/s
  - **ECN-marked packets** drop *1.8x* from 330k/s to 185k/s
  - **Ethernet PAUSE frames** (Aggregator-to-ToR) drop *5.3x* from 80 frames/s to 15 frames/s
  - **90p and 99p indexer-to-aggregator OWD** drop *2.5-3x*

The traffic pattern of the production job
**COLA**: Learned Autoscaling for Cloud Microservices

**Research of**: Vig Sachidananda (*Stanford*), Anirudh Sivaraman (*NYU*)

**Microservices underlie some of the largest applications**
- Google, Facebook, Netflix, Amazon...
- Applications often use autoscalers to scale resources as workload changes
- Autoscaling commonly relies on setting resource utilization targets (CPU)

**We introduce COLA, an autoscaler which:**
- Learns autoscaling policies tailored to applications and their workloads
- Optimizes for end-to-end latency rather than machine utilization
- Reduces dollar cost of meeting a latency target for an application by 26.4% when compared with various utilization/ML based approaches
Given an application, workload, and latency target COLA iteratively:

1. Selects most congested microservice to optimize by VM CPU utilization
2. Uses multi armed bandit algorithm to optimize the number of replicas for this selected microservice

Across 5 open-source applications and 39 total workloads:

• On average, COLA reduces the dollar cost to meet a latency target for all applications (by 19% - 53.1% over next cheapest autoscaler to do so)
• COLA pays for its own training cost within 1-7 days depending on the application
Nezha: Motivation

**Joint work with:** Jinkun Geng (*Stanford*) , Anirudh Sivaraman (*NYU*)

**Consensus protocols are widely applied in practice**
- Coordination: Chubby, Zookeeper, etcd
- Storage: BigTable, TiDB, CockroachDB, MongoDB, RedisRaft
- Messaging: RabbitMQ, Apache Kafka
- Blockchain-related: Ethereum, Hyperledger
- ...

**Existing protocols**
- Easily deployable but suffer from poor performance
e.g., Multi-Paxos, Raft, Viewstamped Replication, Fast Paxos, EPaxos, etc.
- High-performance but use network support
e.g., Speculative Paxos, NOPaxos, NetChain, Mu, etc.

**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
Nezha: Synchronized Clocks for Ordering

Main objectives of consensus protocols: ensure leader and followers nodes have
- The *same requests* → Set Equality
- The requests are processed in the *same order* → Uniform Ordering
- Set equality requires reliable transport; uniform ordering requires *event ordering across clients*
- Uniform ordering is hard to achieve and severely affects performance

How uniform ordering is achieved
- Raft/Multi-Paxos/VR rely on the order of requests in the leader
  → high latency (2 RTTs), and the leader can become a throughput bottleneck
- Speculative Paxos/NOPaxos use a highly-engineered network
  → unable to deploy in public cloud

Nezha uses synchronized clocks for uniform ordering
- More precisely, Nezha achieves uniform ordering in fast path; fixes set inequality in slow path
  “Depends on clock synchronization for performance but not for correctness” (Liskov, 1993)
- Outperforms 4 baselines (Multi-Paxos, Fast Paxos, NOPaxos, Raft) by 1.8–19.9x in throughput, and by 1.2–2.2x in latency