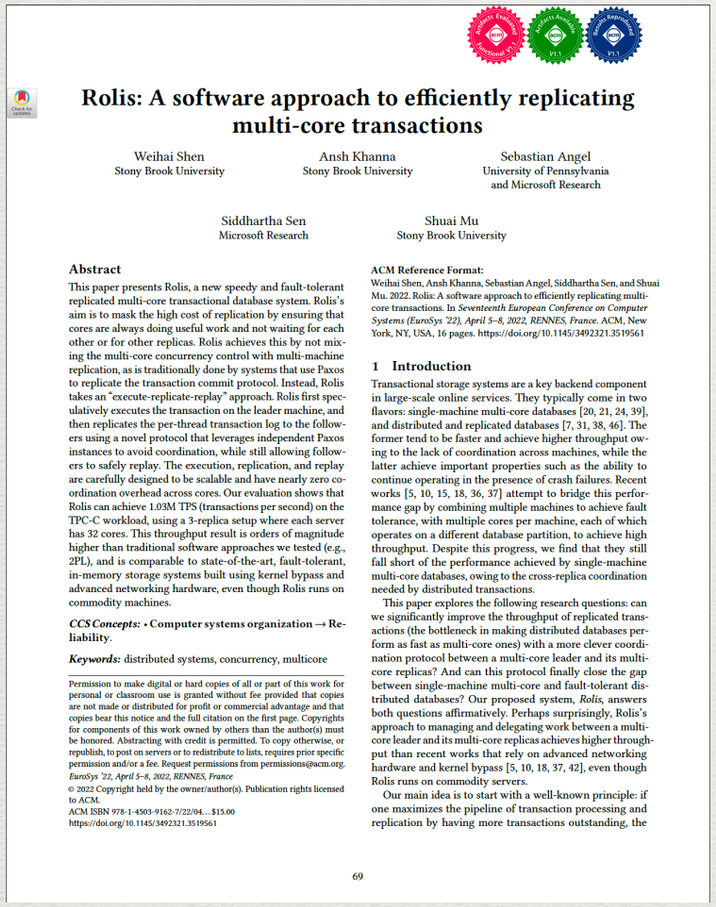
COMP 513 - PROJECT PRESENTATION

Sagar Nandeshwar
Student ID: 260920948

Topic

Rolis is a fault-tolerant multi-core distributed database system.

- Transactions Data
- Distributed Systems
- Backup storage



Rolis: A software approach to efficiently replicating multi-core transactions

Weihai Shen
Stony Brook University

Ansh Khanna
Stony Brook University

Sebastian Angel
University of Pennsylvania
and Microsoft Research

Siddhartha Sen
Microsoft Research

Shuai Mu
Stony Brook University

Abstract

This paper presents Rolis, a new speedy and fault-tolerant replicated multi-core transactional database system. Rolis's aim is to mask the high cost of replication by ensuring that cores are always doing useful work and not waiting for each other or for other replicas. Rolis achieves this by not mixing the multi-core concurrency control with multi-machine replication, as is traditionally done by systems that use Paxos to replicate the transaction commit protocol. Instead, Rolis takes an "execute-replicate-replay" approach. Rolis first speculatively executes the transaction on the leader machine, and then replicates the per-thread transaction log to the followers using a novel protocol that leverages independent Paxos instances to avoid coordination, while still allowing followers to safely replay. The execution, replication, and replay are carefully designed to be scalable and have nearly zero coordination overhead across cores. Our evaluation shows that Rolis can achieve 1.03M TPS (transactions per second) on the TPC-C workload, using a 3-replica setup where each server has 32 cores. This throughput result is orders of magnitude higher than traditional software approaches we tested (e.g., 2PL), and is comparable to state-of-the-art, fault-tolerant, in-memory storage systems built using kernel bypass and advanced networking hardware, even though Rolis runs on commodity machines.

CCS Concepts: • **Computer systems organization → Reliability**.

Keywords: distributed systems, concurrency, multicore

EuroSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9162-7/22/04…$15.00
https://doi.org/10.1145/3492321.3519561

ACM Reference Format:
Weihai Shen, Ansh Khanna, Sebastian Angel, Siddhartha Sen, and Shuai Mu. 2022. Rolis: A software approach to efficiently replicating multi-core transactions. In *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3492321.3519561

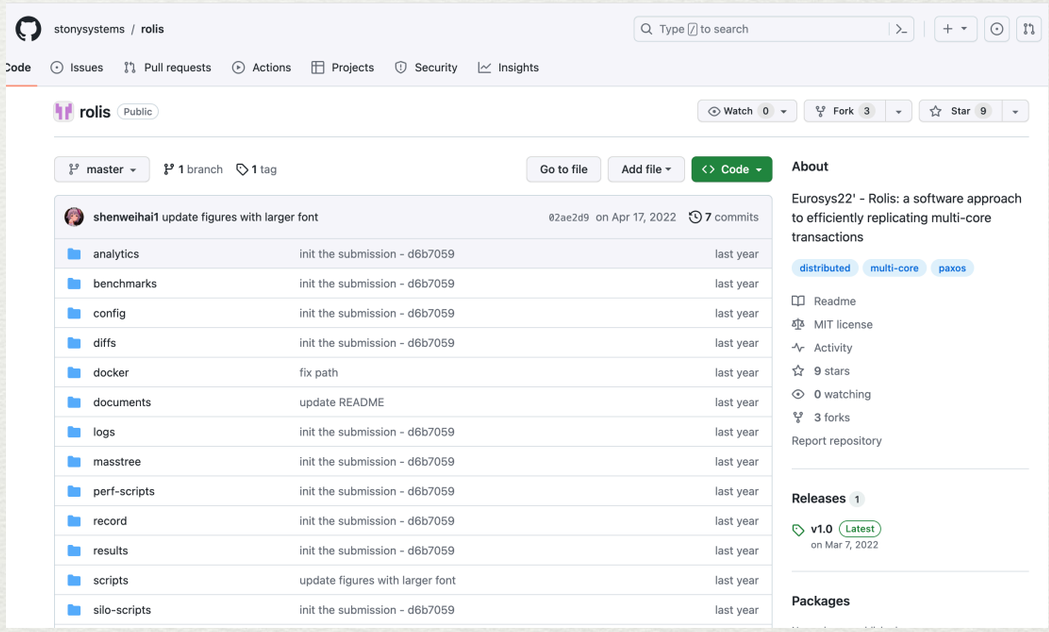
1 Introduction

Transactional storage systems are a key backend component in large-scale online services. They typically come in two flavors: single-machine multi-core databases [20, 21, 24, 39], and distributed and replicated databases [7, 31, 38, 46]. The former tend to be faster and achieve higher throughput owing to the lack of coordination across machines, while the latter achieve important properties such as the ability to continue operating in the presence of crash failures. Recent works [5, 10, 15, 18, 36, 37] attempt to bridge this performance gap by combining multiple machines to achieve fault tolerance, with multiple cores per machine, each of which operates on a different database partition, to achieve high throughput. Despite this progress, we find that they still fall short of the performance achieved by single-machine multi-core databases, owing to the cross-replica coordination needed by distributed transactions.

This paper explores the following research questions: can we significantly improve the throughput of replicated transactions (the bottleneck in making distributed databases perform as fast as multi-core ones) with a more clever coordination protocol between a multi-core leader and its multi-core replicas? And can this protocol finally close the gap between single-machine multi-core and fault-tolerant distributed databases? Our proposed system, *Rolis*, answers both questions affirmatively. Perhaps surprisingly, Rolis's approach to managing and delegating work between a multi-core leader and its multi-core replicas achieves higher throughput than recent works that rely on advanced networking hardware and kernel bypass [5, 10, 18, 37, 42], even though Rolis runs on commodity servers.

Our main idea is to start with a well-known principle: if one maximizes the pipeline of transaction processing and replication by having more transactions outstanding, the

69



Rolis Architecture

It consist of several servers:

(One) Leader Replica:

- Responsible for interact with clients and accept new transaction requests
- A transaction is first executed on the leader replica
- It guaranteed isolation from other transactions
- The execution will generate a log entry for the transaction (globally unique timestamp)

(Multiple) Follower Replica

Store the log entry of the transaction

Rolis Architecture

Three main components:

Server's Database:

- This implemented through Silo

Replication layer:

- Implemented through MultiPaxos (variant of the Paxos algorithm)

Watermark tracking:

- To avoid explicit dependency tracking
- Implemented through compare-and-swaps

Baseline Systems

Silo

- Multi-core in-memory high-speed transactional database
- Transactions is executed on threads which is dedicated to a worker thread in the database.

CL2

- Pessimistic concurrent control protocol
- Each transaction performs 4 read-write accesses or 4 read accesses by incrementing 4 random chosen keys.

Calvin

- Central sequencer to determine the order of batched transactions which are sent to all replicas to execute deterministically

Set Up (Hardware)

Paper

- Three Microsoft Azure VM

- Intel Xeon Platinum 8272CL CPU 2.60GHz Processor

- 32 (hyperthreaded) CPU cores with single socket

- 128G RAM

- 16000 Mbps network

Mine

- Three AWS EC2 Instance

- c5a.8xlarge Ubuntu 20.04

- 32vCPU (100%)

- 64 Gib Memory (50%)

- 8000 Mbps (50%)

Total Cost of Ownership: 490.8

Don't worry, I had Amazon Credits

(**But always open to sponsorship**)

Set Up (Experiment)

Rolis package have ‘**one-click.sh**’ file which runs all the experiments for you

Set Up (Experiment)

~~Rolis package have '**one-click.sh**' file which runs all the experiments for you~~

NO! This was a bait…

Set Up (Experiment)

For each of the three server, I needed to

- Set up Docker Container
- Set up Rolis Packages
- Establish Connection between each server (such ssh IP works
- Change EC2 Configuration (allow network access with (.pem), password or username)
- Change DockerFile configuration (host names, permissions)
- Network security and inteference group
- (in-bound SSH from 0.0.0.0/0 -> Opened traffic from Anyware)
- (However I shutdown all the three servers, so it too late, to steal my project)
- Some Codes Modification (ect. scp comands)

Experiments

And then,

(with lot of tears)

$ bash one-click.sh

- This file will produces results in the form of Text Logs Format.
- Takes about 4 hours to run all the experiments

Experiments 1

Test the performance of Rolis

TPC-C Dataset:

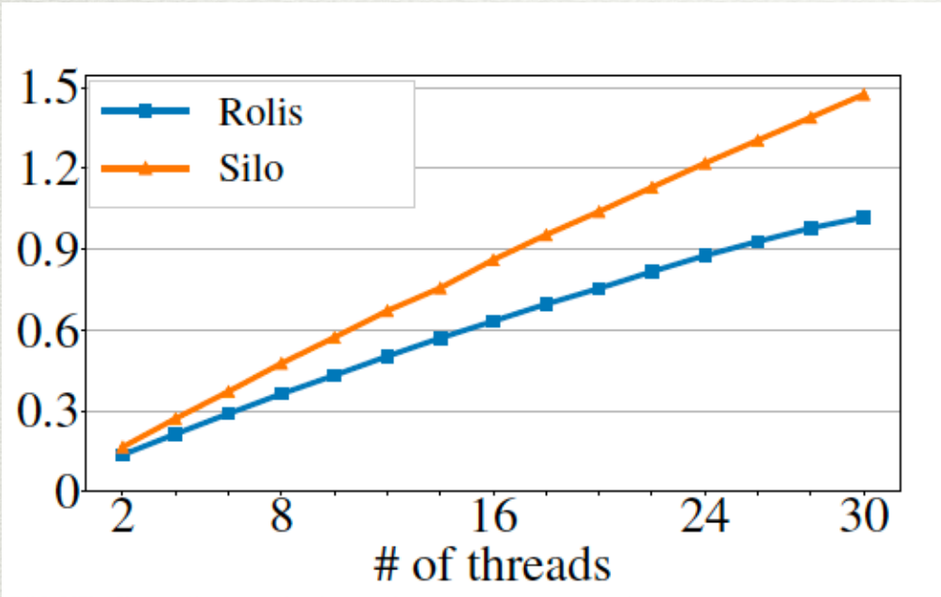
- Consist of complex transactions, such as NewOrder (NEW), Payment (PAY), OrderStatus (ORDER), StockLevel (STOCK), and Delivery (DLVR).
- To evaluate performance when handling complicated transactions.

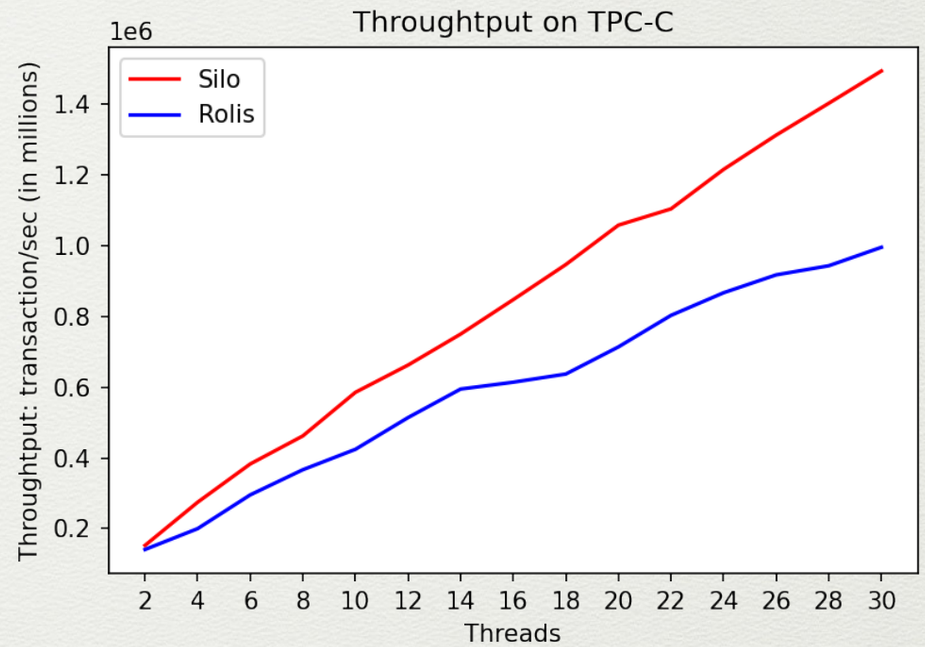
YCSB++ Dataset:

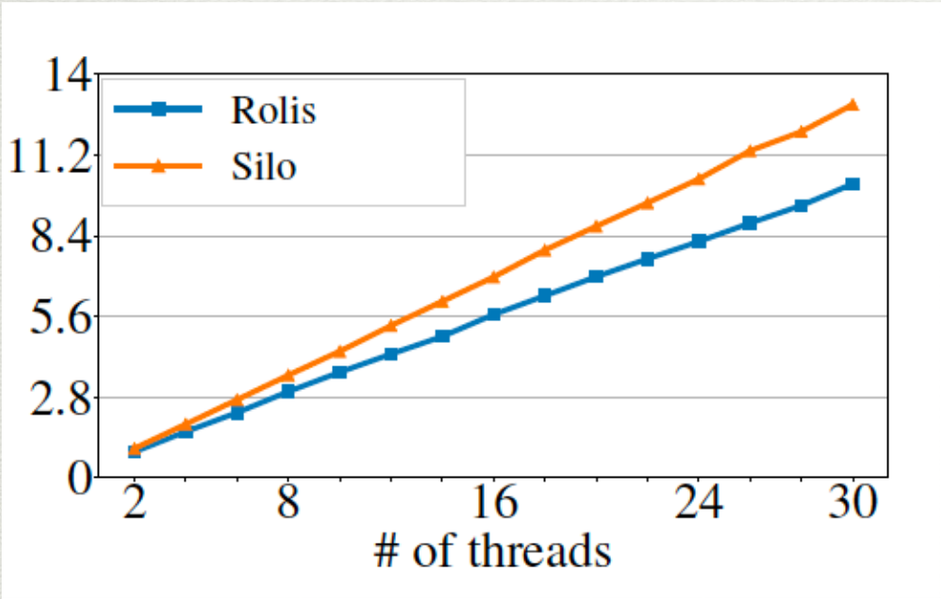
- Consist of simple Read-Only (READ) and Read-Modify-Write (RMW); 50% Reads and 50% RMW, 1 million keys and pairs
- To test the limit of Rolis on heavy workload

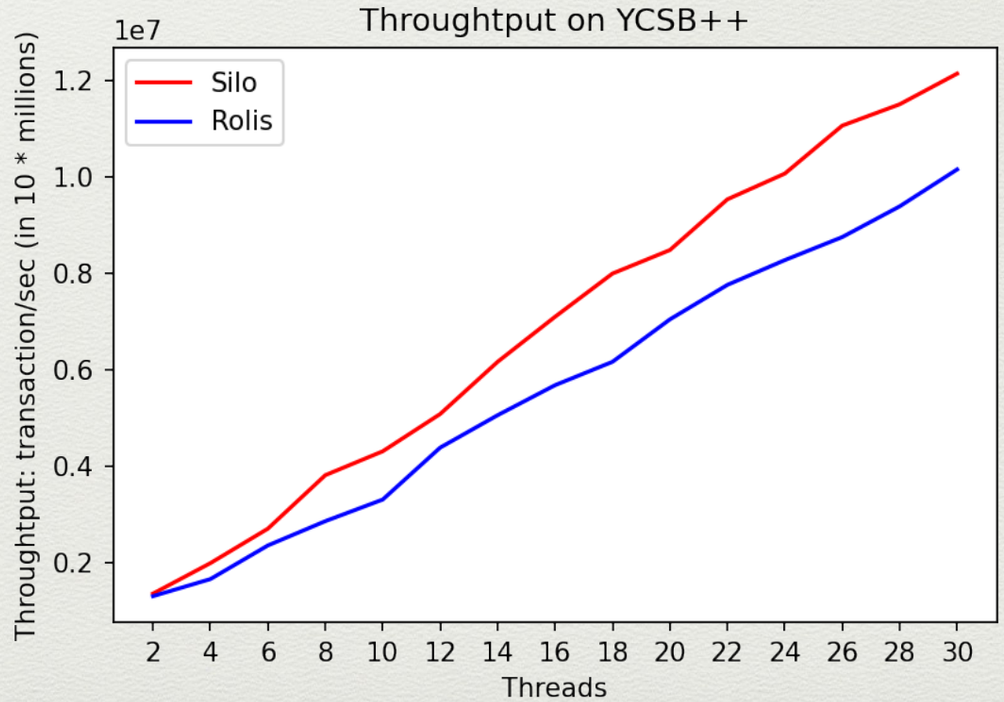
We compared Rolis performance against Silo

Experiments 1









Experiments 2

Compare the Rolis with traditional software implementations

- Calvin
- 2PL

Rolis provide “one-click.sh” file for both Calvin and 2PL to evaluate the performance

Experiments 2

Compare the Rolis with traditional software implementations

- Calvin
- 2PL

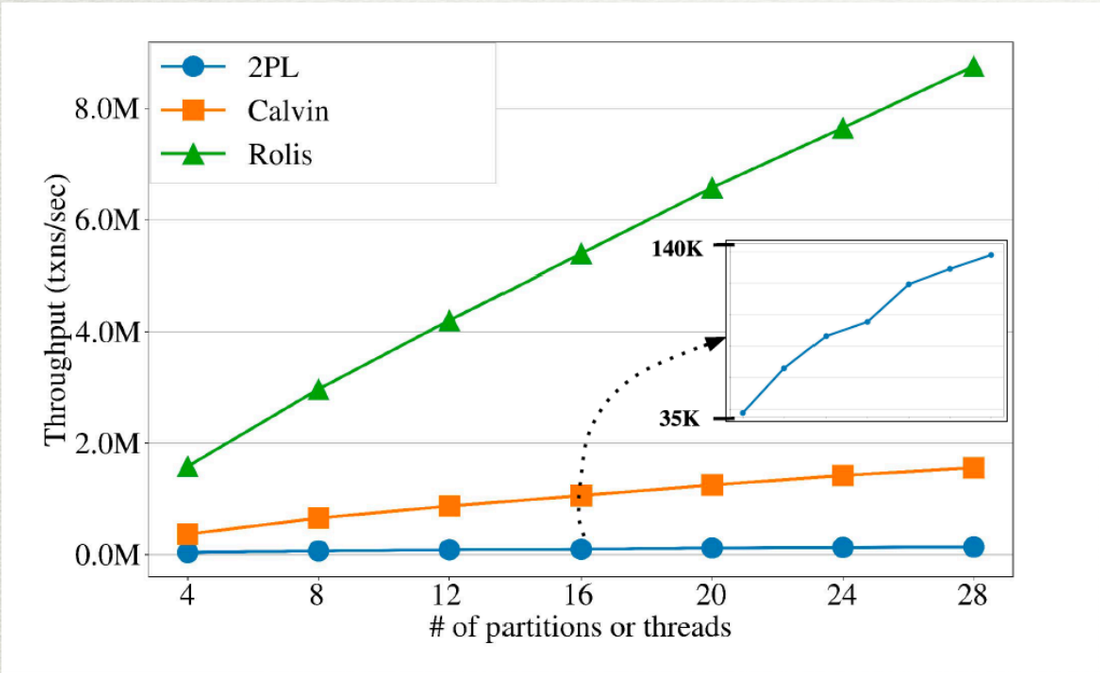
~~Rolis again prove “one-click.sh” file for both Calvin and 2PL to run the comparision~~

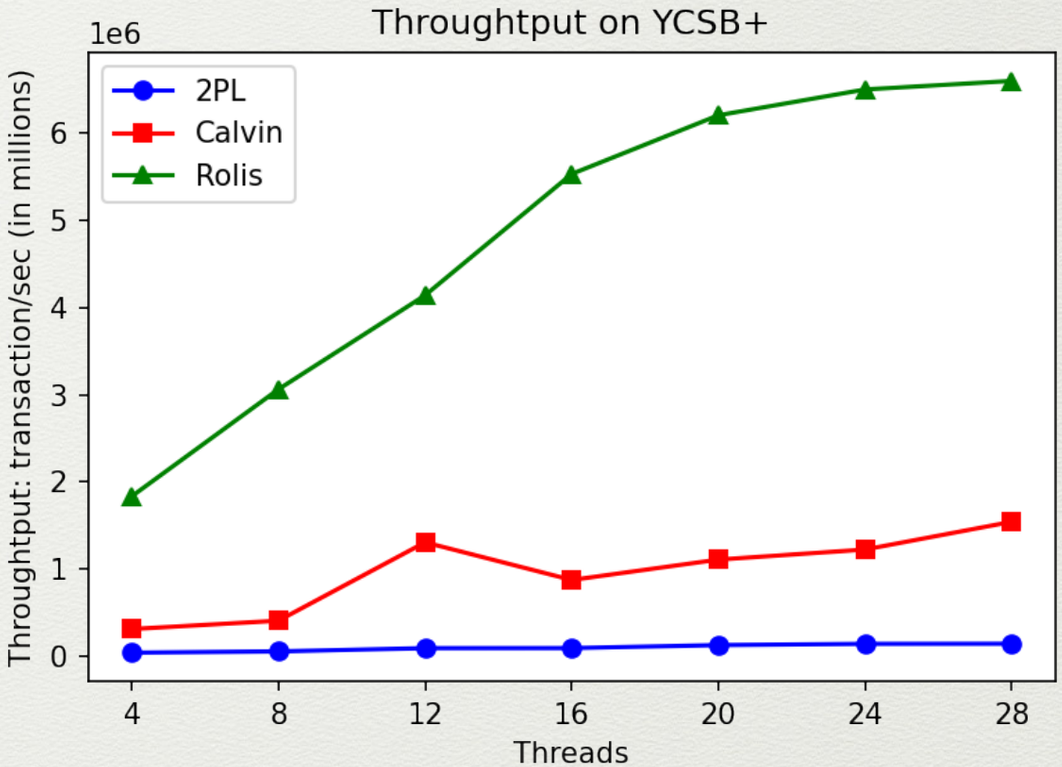
NOPE! bait again…

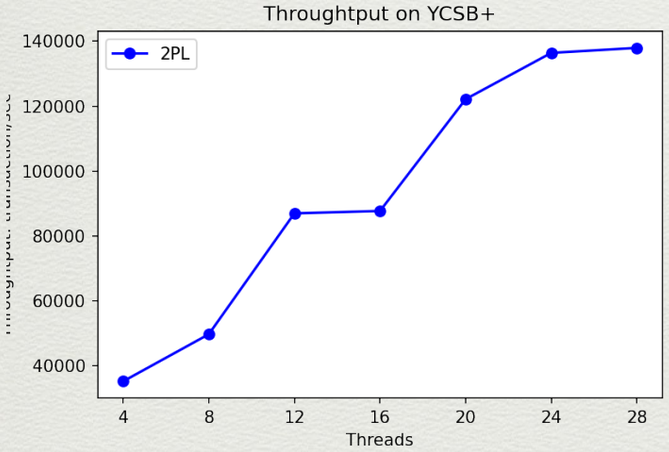
(Fool me Once shame on you, Fool me twice shame on me)

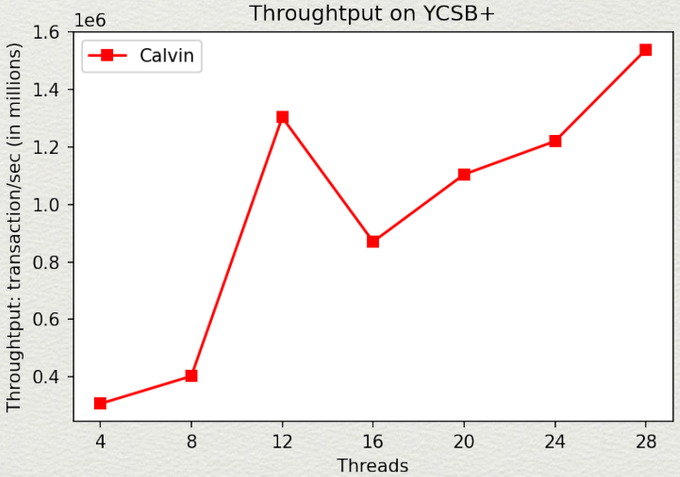
For 2PL, we need to set up 32 4-core EC22 instance(Lucky, it was on free tier)

Experiments 2





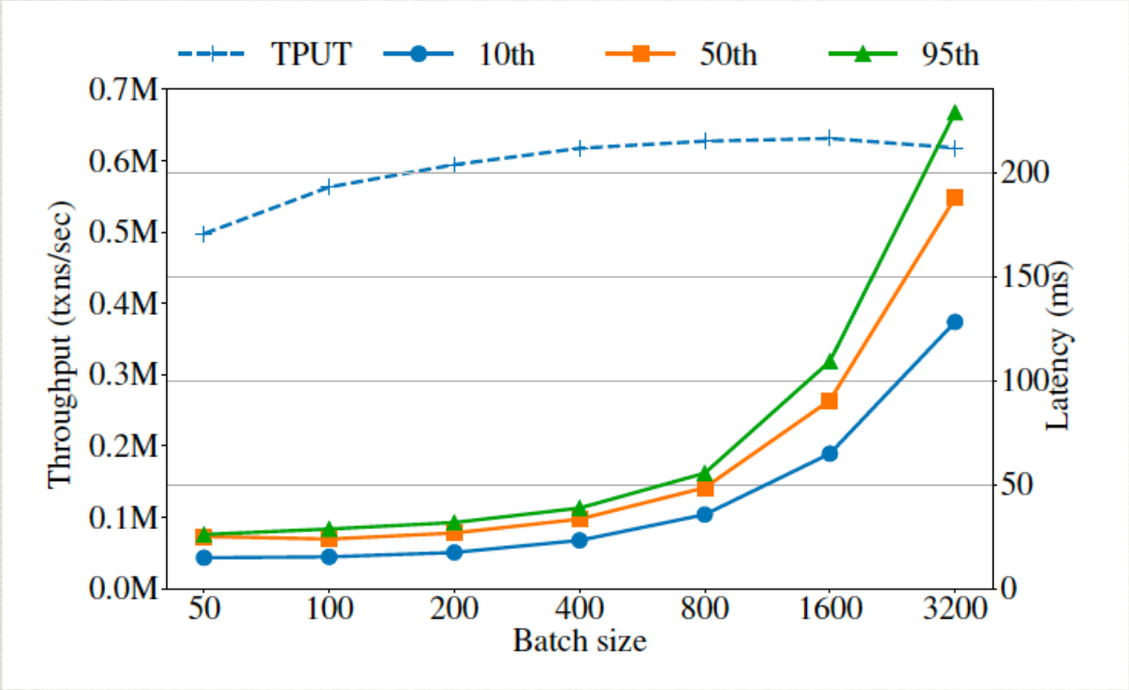


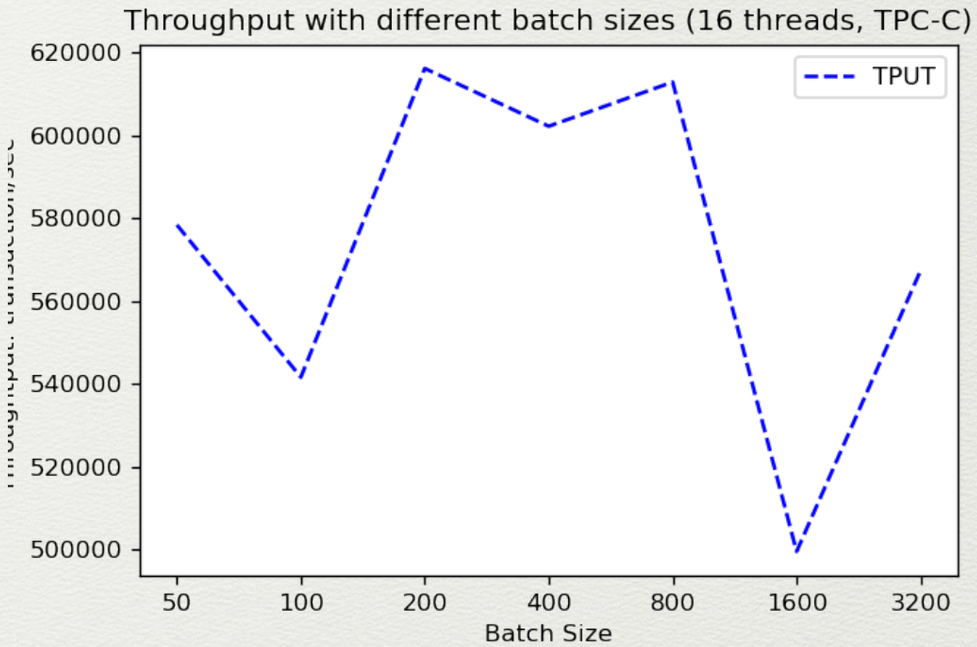


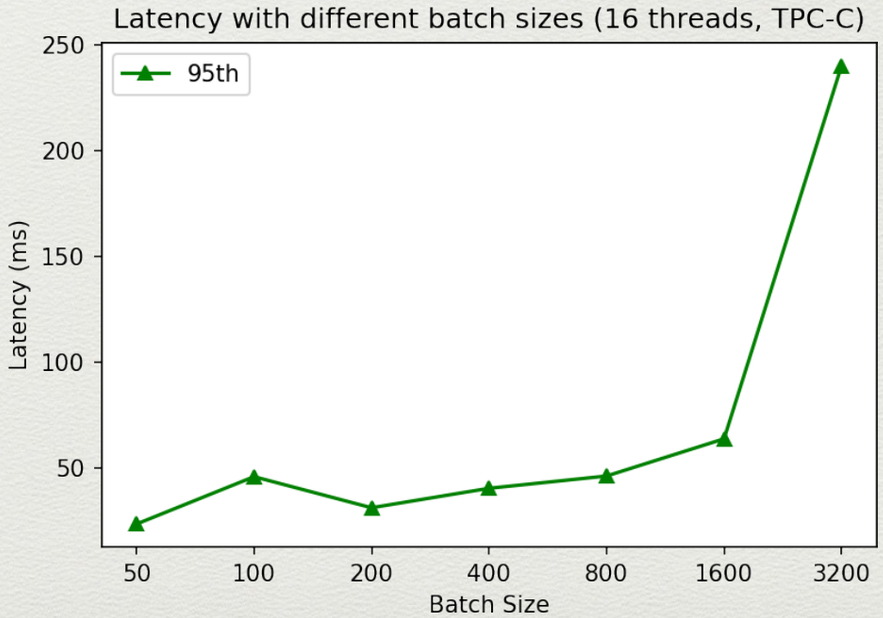
Experiments 3

Evaluate the effect of batch size on throughput and latency

Experiments 3







Thank you