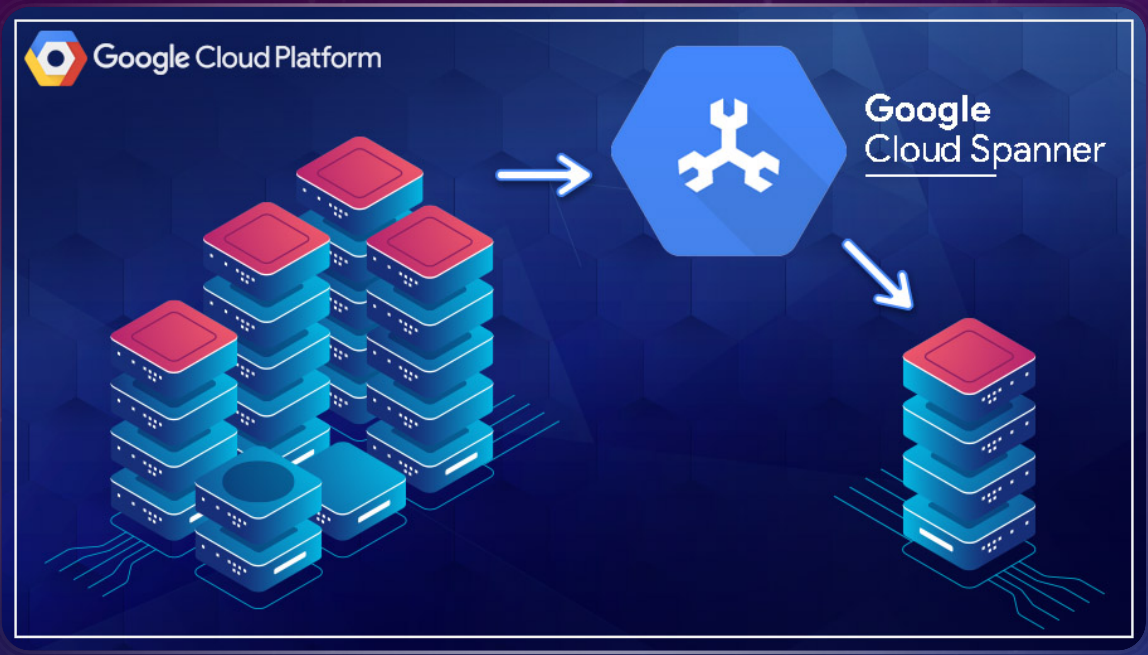
SPANNER

GOOGLES GLOBALLY DISTRIBUTED DATABASE



WHAT IS IT?

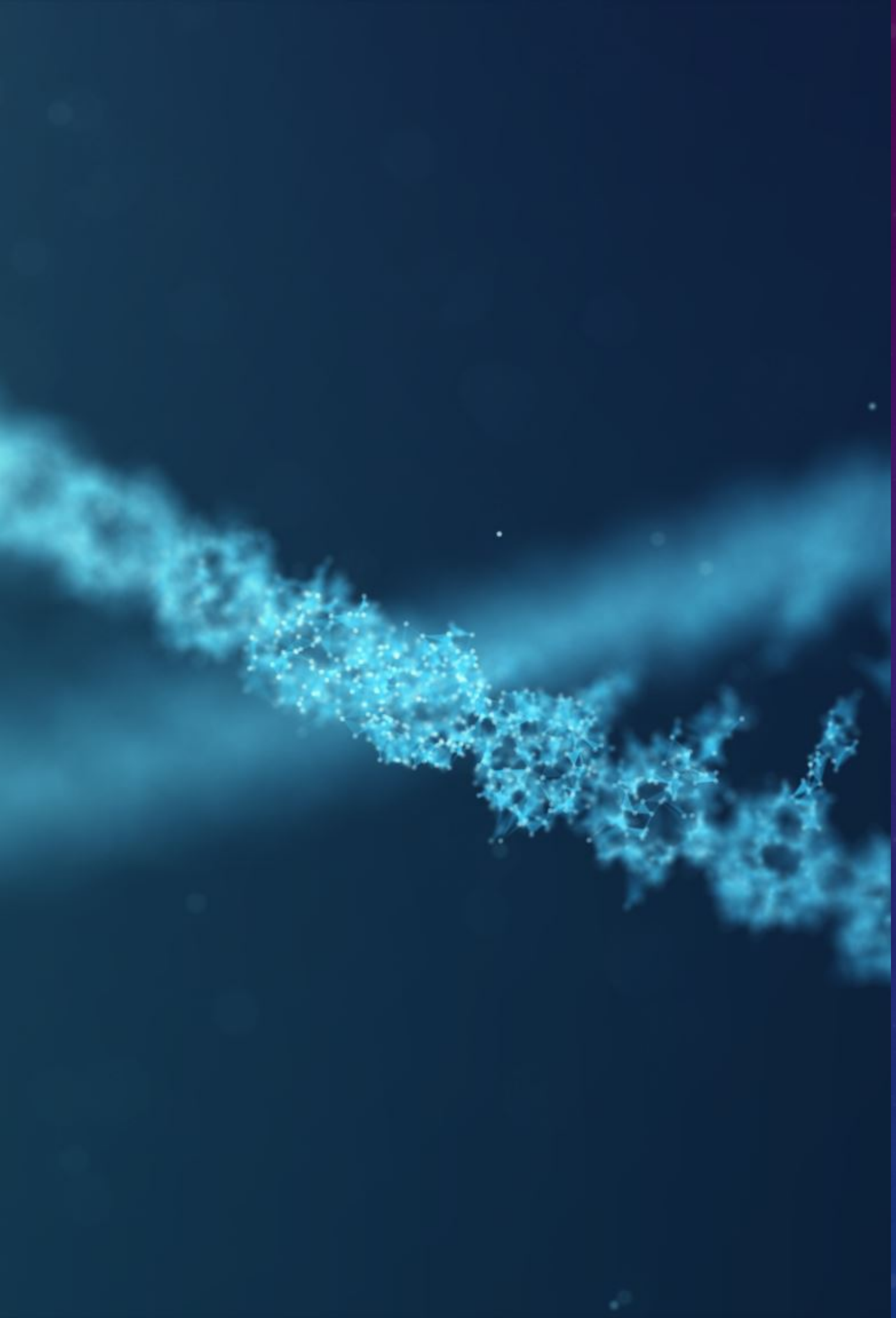
- Scalable, globally distributed database
- Shards data across many sets of Paxos State Machines
- Automatic failover for clients between replicas
- Auto re-sharding of data as well as data migration
- Designed for scale, scale, scale

WHY USE IT

- Applications want high availability, even in the face of natural disasters
- Spanner can replicate across continents (main focus)
- Cross between Bigtable and Megastore

HOW DOES IT WORK

- Temporal multi-version database
- Data stored in schematized tables
- Data is versioned and stamped with a commit time
- Support for general purpose transactions and SQL-based queries



SOME INTERESTING FEATURES

- Replication can be dynamically controlled by the application
- Externally consistent reads and write as well as globally consistent reads at a timestamp
- Globally meaningful commit timestamps which reflect serialization order
- TrueTime API
  - Directly exposes clock uncertainty
  - Uncertainty less than 10m/s
  - Uses a combination of GPS and Atomic clocks

ORGANIZATION

- Spanner deployment is called a universe
    - Test/Playground
    - Dev/Prod
    - Prod only
- Zones can be added to or removed from a running system as new datacenters are brought into service and old ones are turned off

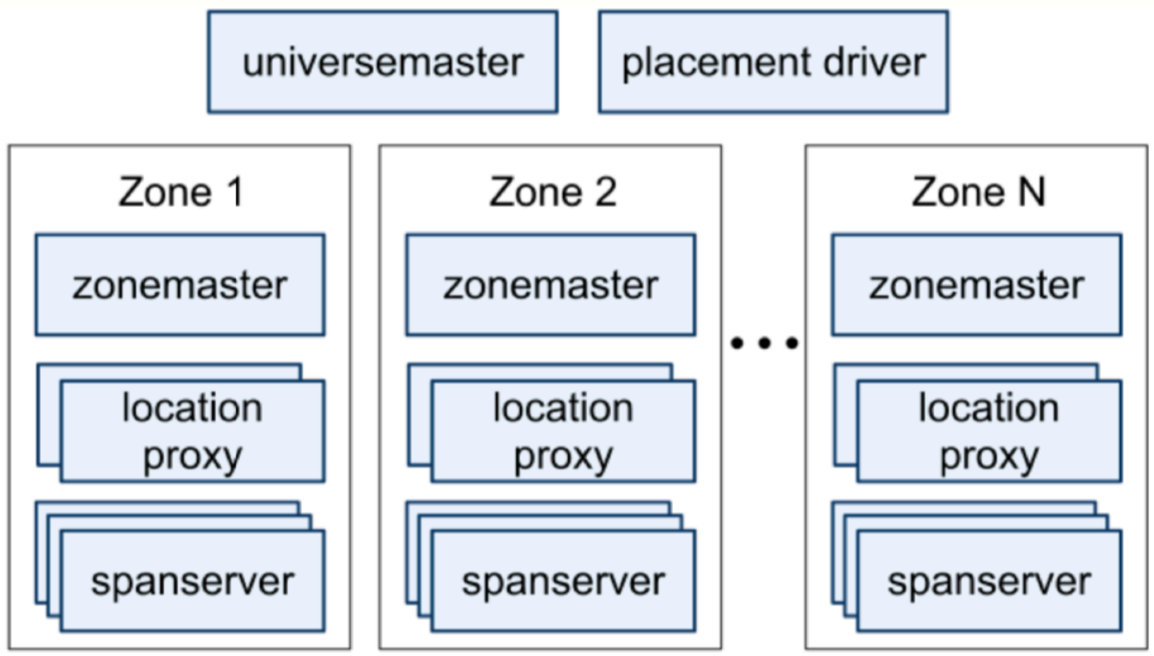
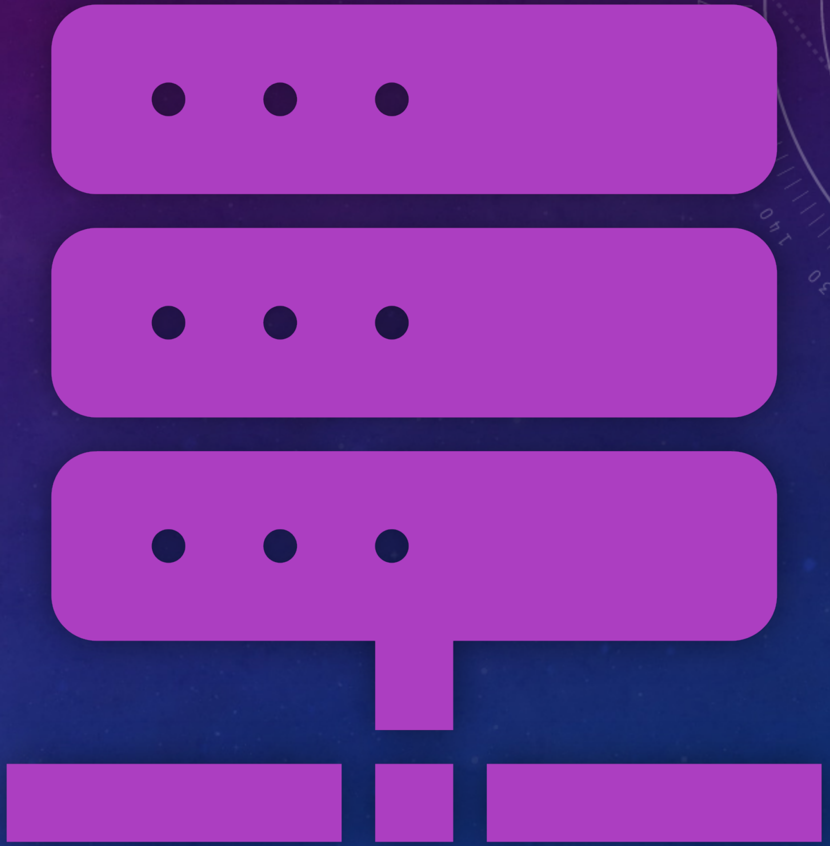


Figure 1: Spanner server organization.

SPANSERVER STACK

- Each server responsible for between 100 and 1000 instances of a data structure: **Tablet**
- (key:string, timestamp:int64) → string
- All data has timestamps
- Single Paxos state machine on top of each tablet for replication
  - Stores meta data in tablet
  - Writes must initiate protocol at leader
  - Reads access tablet directly



LEADER REPLICAS

- Lock table implemented at every replica that is a leader
  - Designed for long-lived transactions that can take minutes
  - Ops that need synchro can use table, others can bypass
- Transaction Manager also implemented
  - Used to implement a participant leader
  - Transactions involving more than one Paxos group use this to coordinate with 2PC

DIRECTORIES

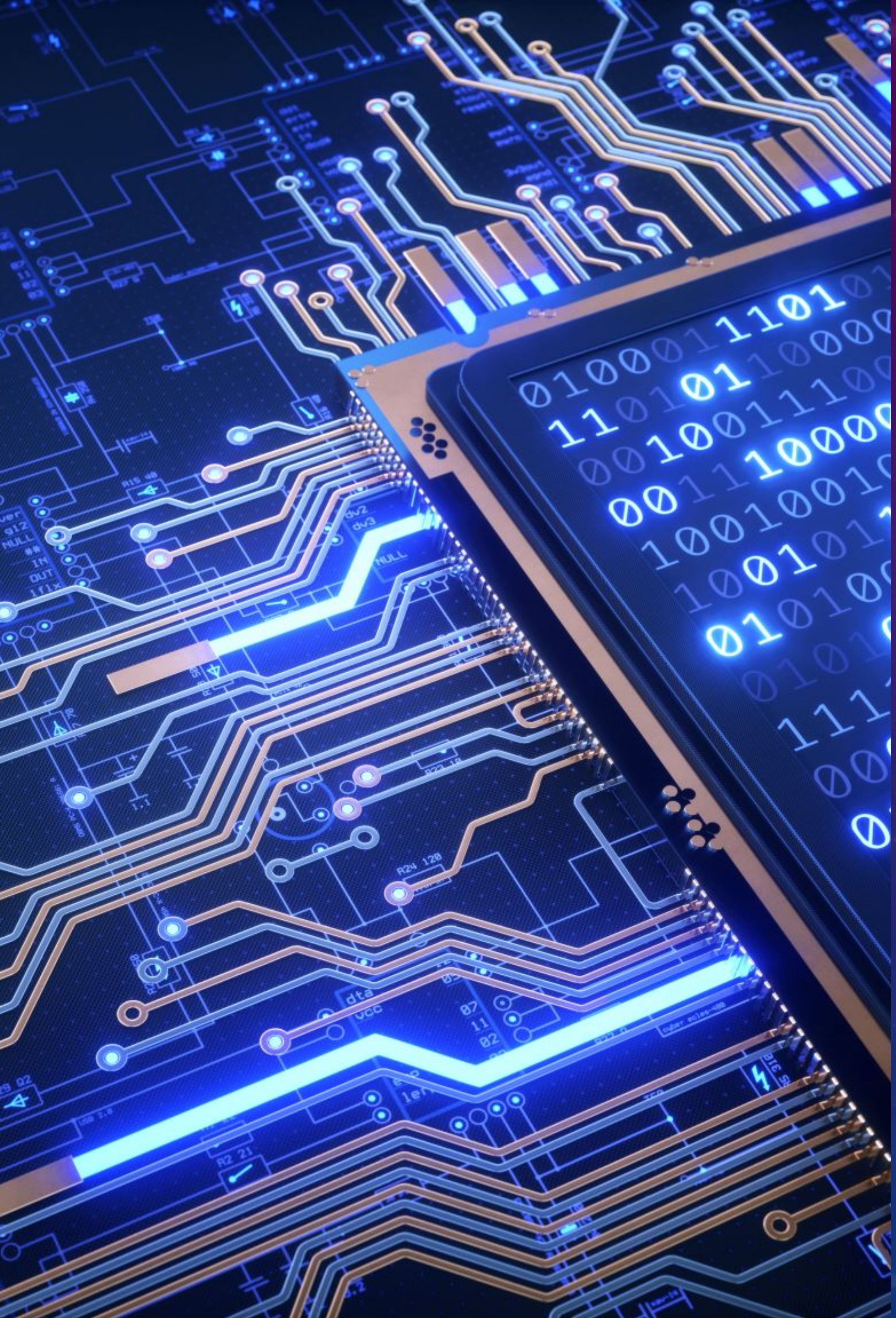
Set of keys that share a common prefix

Allow applications to control locality of data

When data is moved between Paxos groups, it is moved directory by directory

Can be moved while client operations are ongoing

Moves data in the background



APPLICATION DIRECTORY INTERACTION

- Smallest unit where location can be specified by application
- Admins control the # and types of replicas, Applications control how data is replicated
- Spanner can shard a directory if it grows too large

DATA MODEL

- Applications create one or more databases in the universe
- Each DB can have an unlimited number of schematized tables (relational db tables)
- Every table has an ordered set of primary key columns
- Applications can control data locality through choice of keys

TRUE TIME

Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

Table 1: TrueTime API. The argument t is of type $TTstamp$.

TRUE TIME

- GPS and atomic clocks since each have different failure modes
- Implemented with time master machines
    - Some have GPS antennas and the rest have atomic clocks
- Masters cross check with references and self evict if there are significant differences
- GPS masters have clock drift typically close to zero

SUPPORTED OPERATIONS

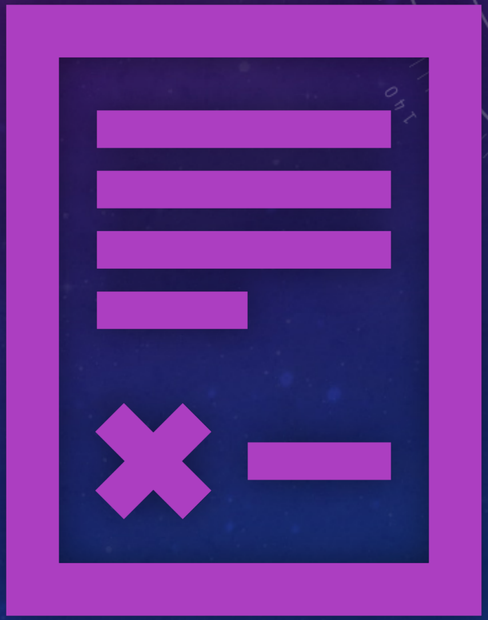
- RW transactions
  - Standalone writes
- RO transactions
  - Non-snapshot standalone reads
- Snapshot reads

LEADER LEASES

- Leader gets timed leases on the quorum
- Lease interval for leader starts when it gets a quorum and ends when it no longer does
- Leader can request lease vote extension if near expiration
- 10 second leases

RW TRANSACTION TIMESTAMPS

- Transactions assigned as the same for the Paxos system write (monotonically increasing)
- If start of T_2 occurs after commit of T_1 then T_2 timestamp must be greater than T_1
- Two rules
    - Start – Leader assigns timestamp s_i no less than TT.now.latest()
    - Commit Wait – Clients cannot see any data commited by T until after TT.after(s_i) is true



RO TRANSACTION TIMESTAMPS

- Assign timestamp s_{read} and then execute read from snapshot at s_{read}
- Timestamp = TT.now().latest
- Assign oldest timestamp that preserves external consistency to prevent blocking

BENCHMARK SETUP

Clients and Spansevers on separate machines

Each zone = 1 spanserver, placed in a set of datacenters with < 1 ms network distance

50 Paxos groups, 2500 directories

Standalone read and writes of 4KB

1 warmup round

LATENCY BENCHMARKING

- Clients issued few enough operations to avoid queues building at the server
- With 1 replica commit wait ~= 5ms and Paxos latency ~=9ms
- As replica numbers increase, latency stays roughly constant
- Latency to achieve a quorum decreases since more replicas

THROUGHPUT BENCHMARKS

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.

- Client issues enough operations to saturate server
- Throughput increases linearly with number of replicas

AVAILABILITY

- 5 zones, each have 25 spanservers
- Sharded into 1250 Paxos groups
- 100 test clients con- stantly issued non-snapshot reads at an aggregrate rate of 50K reads/second
- 5 seconds in, all servers in 1 zone are killed
- All leaders are located in Zone 1: Z_1

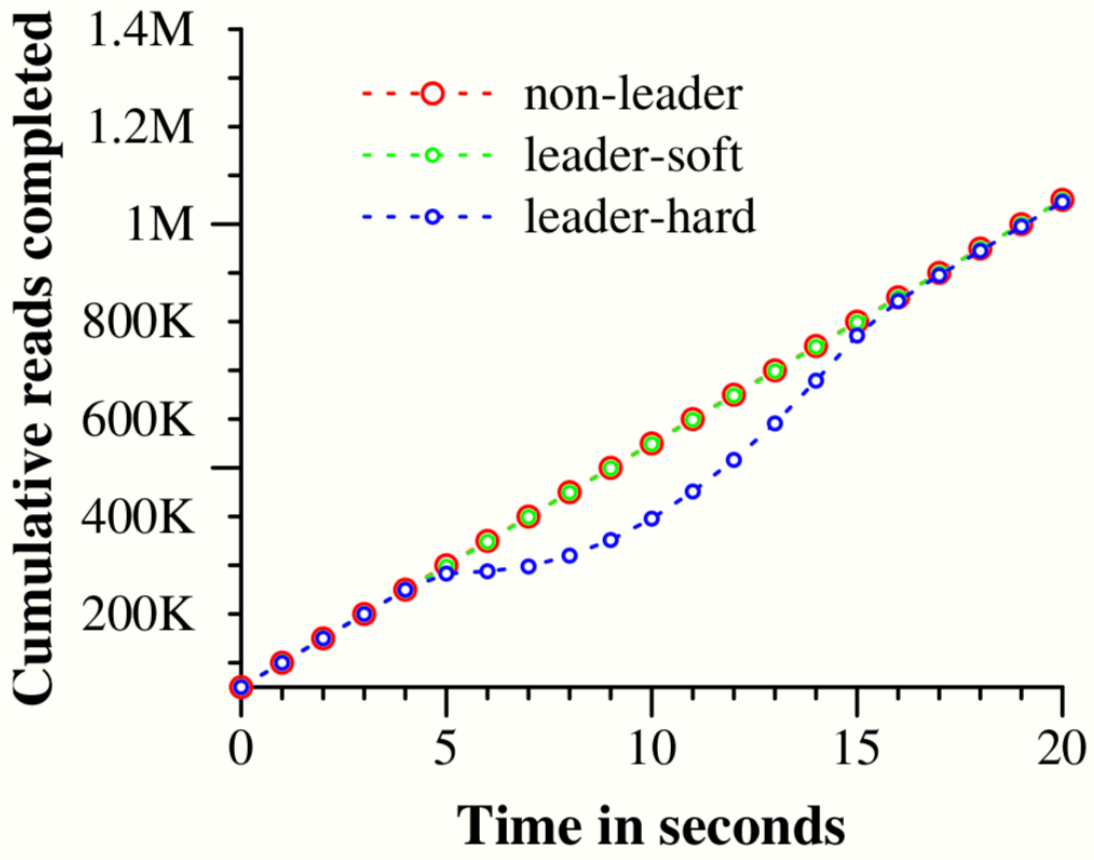
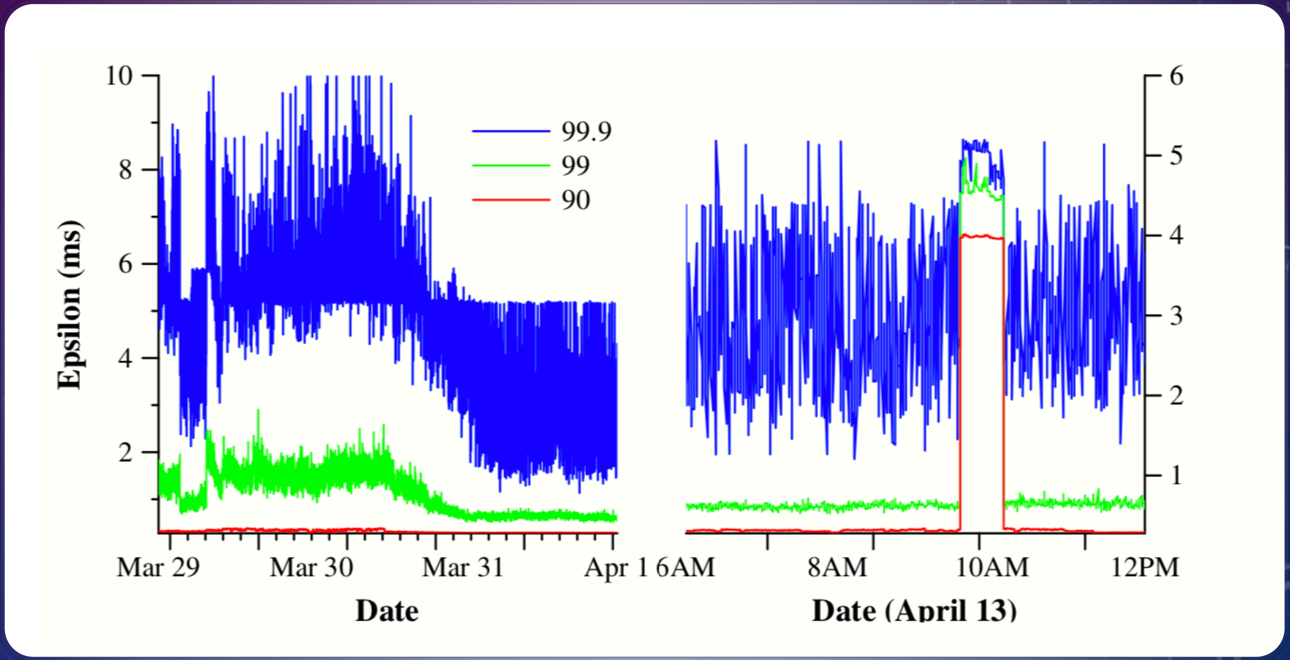


Figure 5: Effect of killing servers on throughput.

TRUETIME TESTS

- Is TrueTime reliable/trustworthy?
- Mostly yes but outside factors can influence reliability
  - Networking Improvements
  - Shutdown of time masters at datacenters for maintenance



DISCUSSION TIME