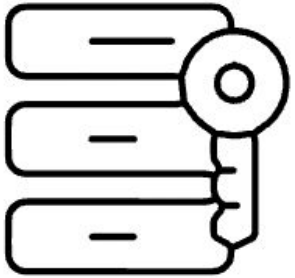
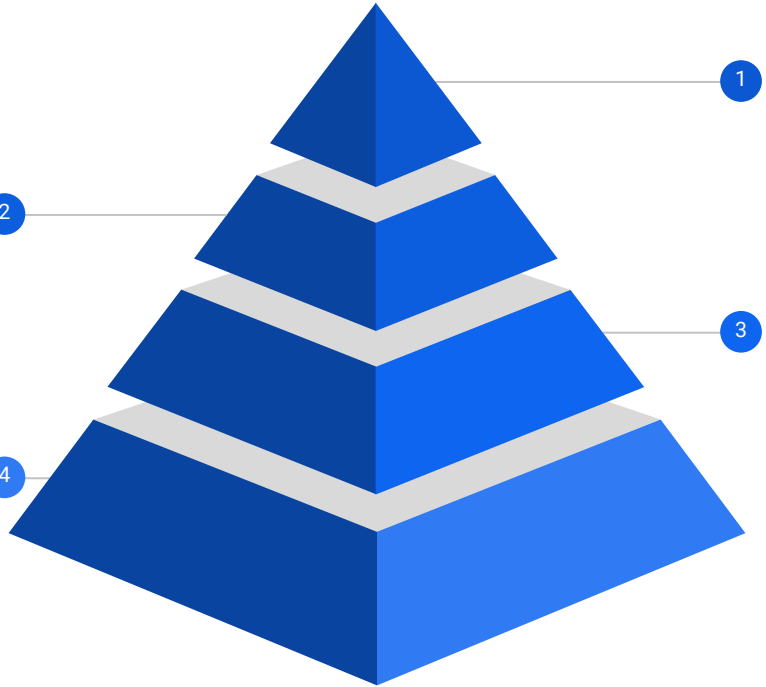


Key-value databases with *etcd* and *Hazelcast*

INFO-H415 Advance Databases
Fall 2022

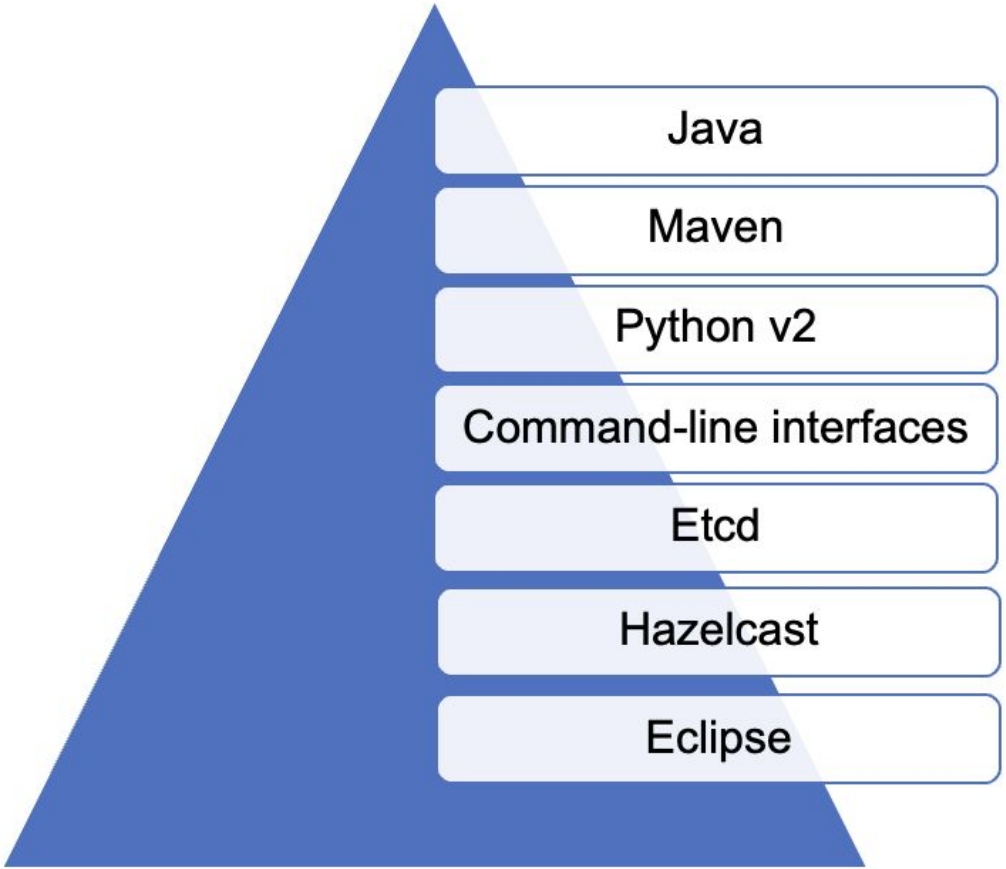
Aliakberova, Liliia,
Gepalova, Arina,
Lorencio Abril, Jose Antonio
Mayorga Llano, Mariana

Professor: Zimányi, Esteban


1
Technology Introduction
2
etcd and Hazelcast
3
YCSB Benchmark
4
YCSB Benchmark Results

Technology Introduction

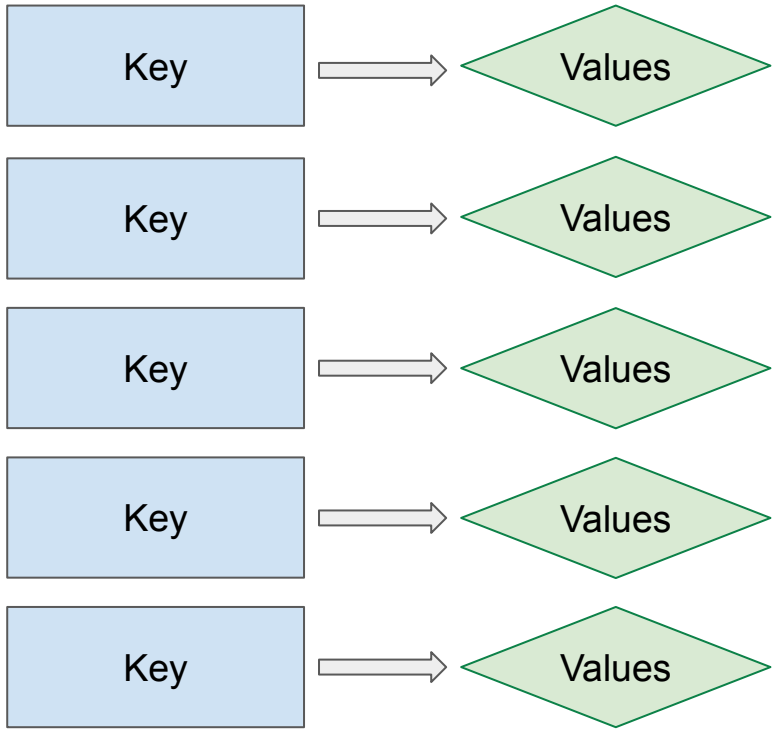
Used Tools


Java
Maven
Python v2
Command-line interfaces
Etcd
Hazelcast
Eclipse

Key-value stores

Features

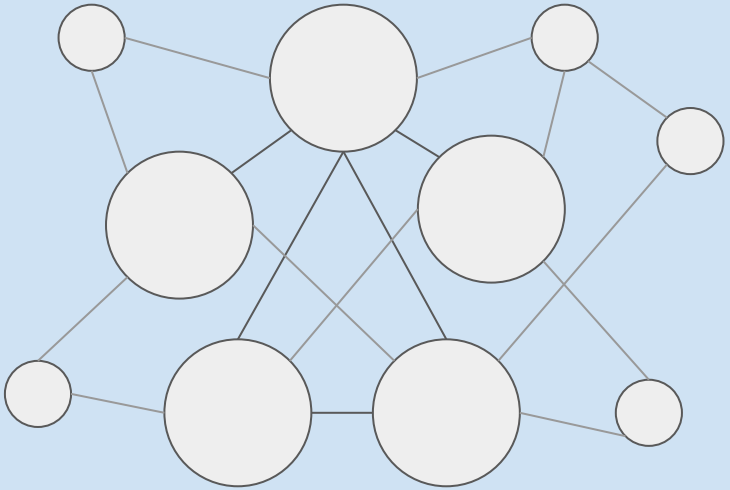
- NoSQL database
- Schemaless data model
- Flexibility
- High performance
- Etcd and Hazelcast: distributed systems



etcd and *Hazelcast*

etcd

Distributed system



Independent components connected

Open-source, **key-value** database

Distributed systems challenge:

- **Consistent** single source of information
- **Coordination** amongst clusters

Designed to

- Store infrequently updated data
- Provide reliable watch queries

Have you heard about **RAFT** algorithm?

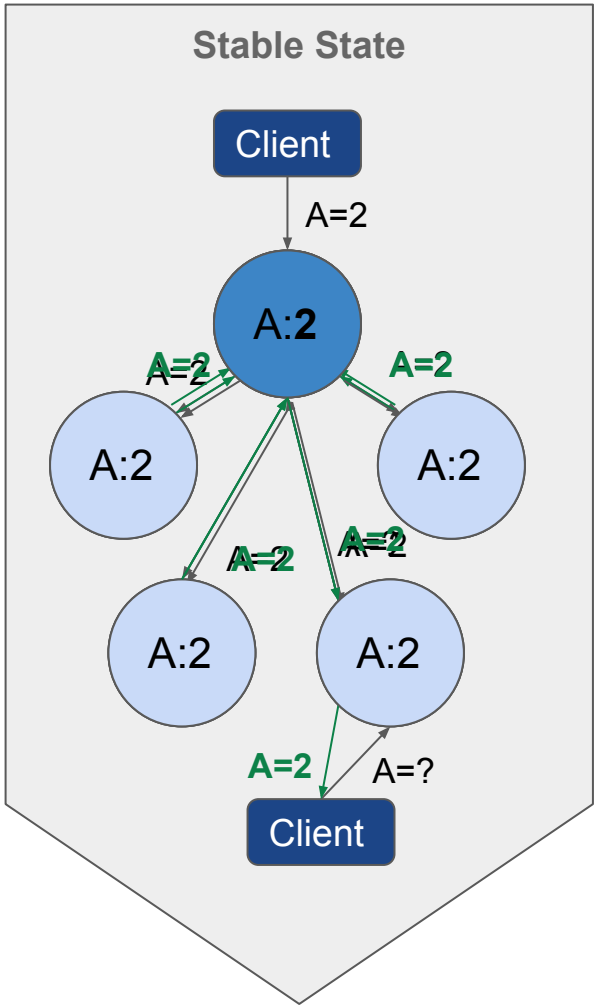
RAFT Algorithm

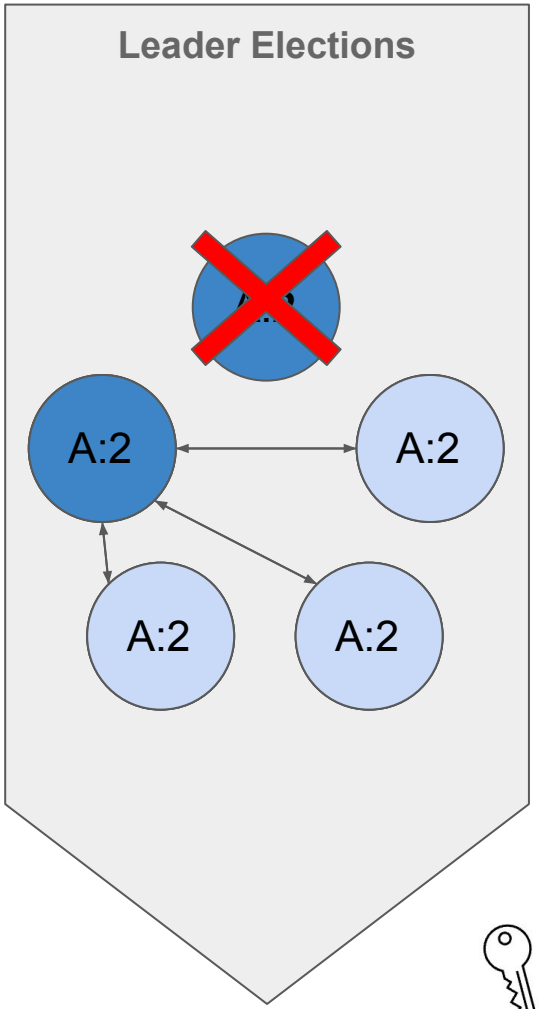
Roles:

- Leader
- Follower
- Candidate
- Learner

Advantages:

- Replicated
- Consistent
- Highly available
- Fault Tolerant
- Performance





etcd

Hazelcast IMDG:

- in-memory object store

Data partitioning:

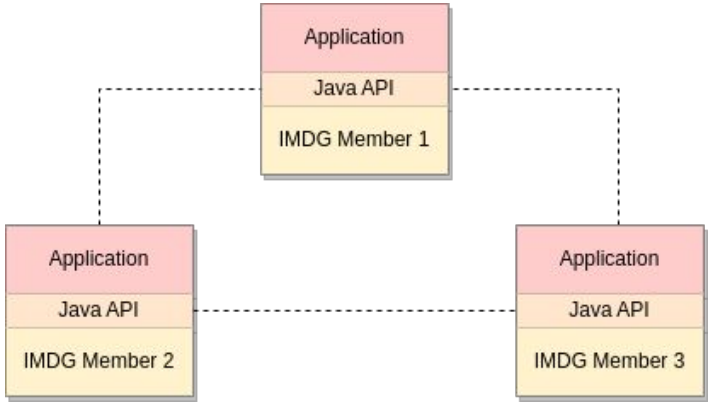
- segments

Features and advantages:

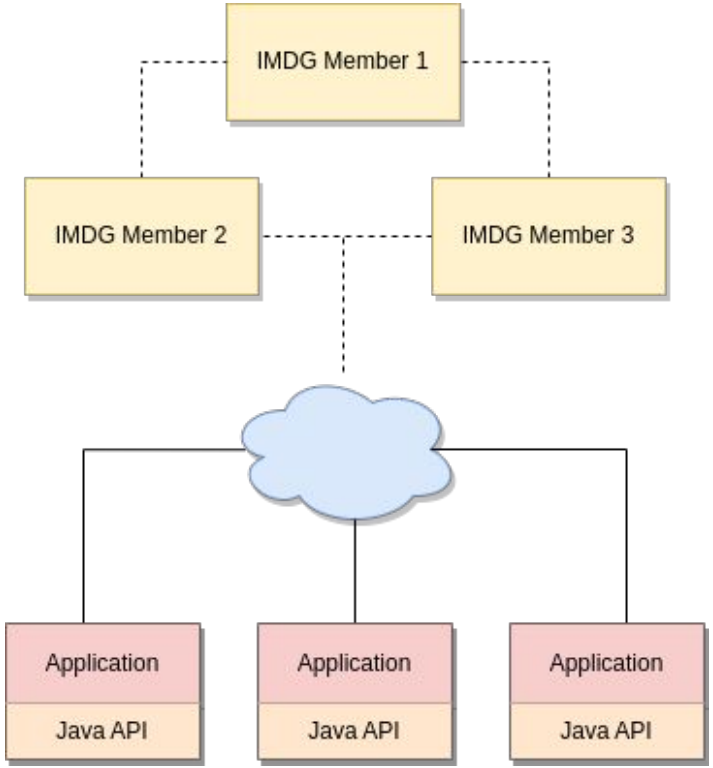
- open source
- used with a simple installation of Java
- uses in-memory storage
- peer-to-peer
- scalability can be performed dynamically
- resilient to failure
- hash-functions for keys

Reference: the image copied from https://hazelcast.com/brand-assets/

Hazelcast Topology



Embedded Topology



Client/Server Topology

HAZELCAST

Benchmark

etcd

YCSB Benchmark

Developed and published:

- ➔ By **Yahoo! Research** (2010).
- ➔ As an **open source benchmark** that could be **extended** to different technologies.

Objective:

- ➔ **Standard benchmark**
- ➔ **Evaluation** of different systems
- ➔ Focusing on **read/write** operations.
- ➔ **Extensibility**

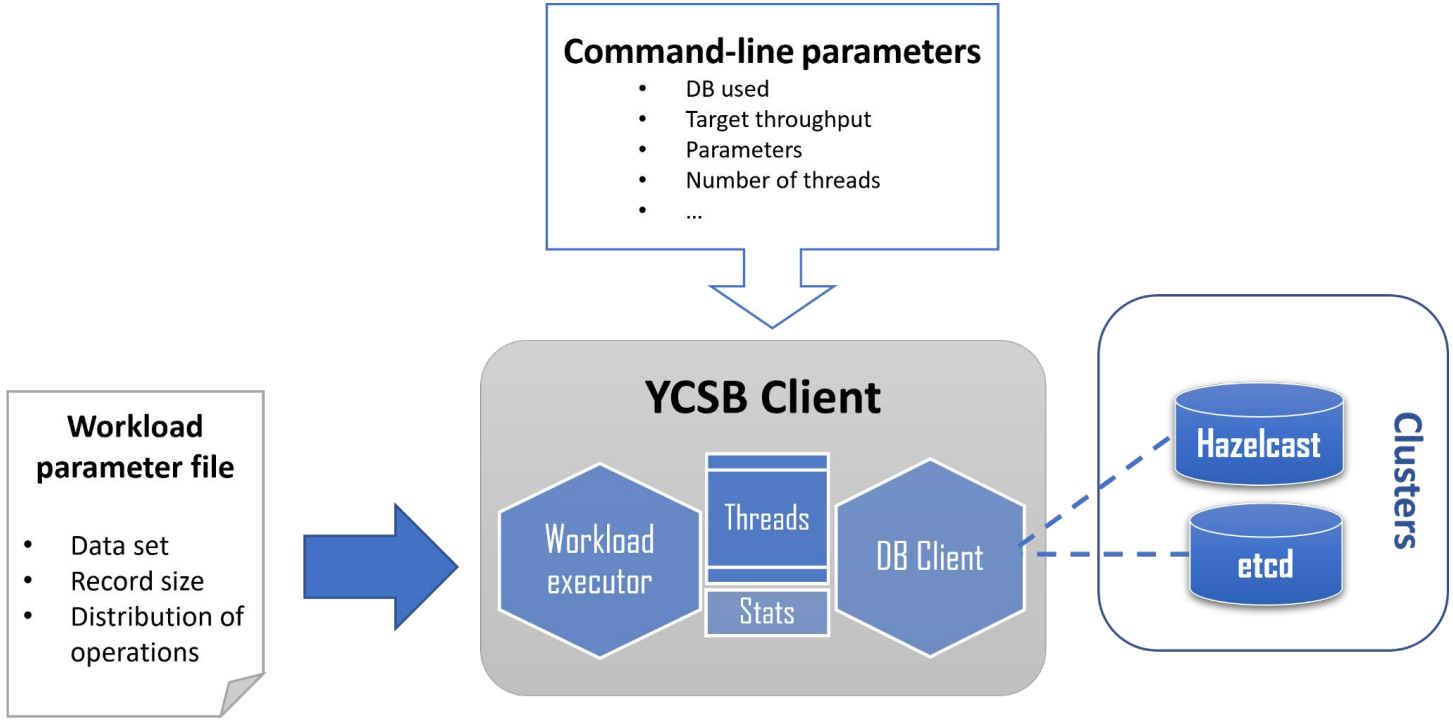
Benchmarking Cloud Serving Systems with YCSB

Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears

Yahoo! Research
Santa Clara, CA, USA
{cooperb,silberst,etam,ramakris,sears}@yahoo-inc.com

YCSB Benchmark

Architecture



YCSB Benchmark

Workloads

OPERATIONS

- **Insert** a new record.
- **Update** a record replacing its value.
- **Read** either a randomly chosen field or all of them.
- **Scan** records in order, starting at a randomly chosen record key. The number of records to scan is randomly chosen.
- **Delete** a single record

Workload A	Update heavy workload	50 % read 50% write (not necessarily the same records)
Workload B	Read mostly workload	95 % read 5 % write (not necessarily the same records)
Workload C	Read only	100% Read
Workload D	Read latest workload	5% insert 95 % read
Workload E	Short ranges	95 % scan 5 % insert
Workload F	Read-modify-write	Forces read of the records to be modified and updated

How about a **custom workload** ?

Implementation of YCSB in *etcd* and *Hazelcast*

Downloading YCSB	Installation of tools	Modifications in pom files	Extending ycsb.DB class	Adaptation of the YCSB core
From the official repository.	Install databases to benchmark. Java 8. Maven. Python 2.	Add pom files to define the new DB Class extensions. Required for Maven to be able to detect the newly implemented databases and compilation.	Implement the five main functions of the DB Class interface: - Read - Insert - Delete - Update - Scan	We have to add some information about the new databases in the main python script of ycsb, which orchestrates the benchmarking and needs to know how to find the DB.

Use Case

Simple Process Orchestrator:

- Receives processes to execute from a **producer.**
- Have to distribute the processes between **servers**.
- Keeps track of all processes and servers.
- For this, it makes use of the **database**.

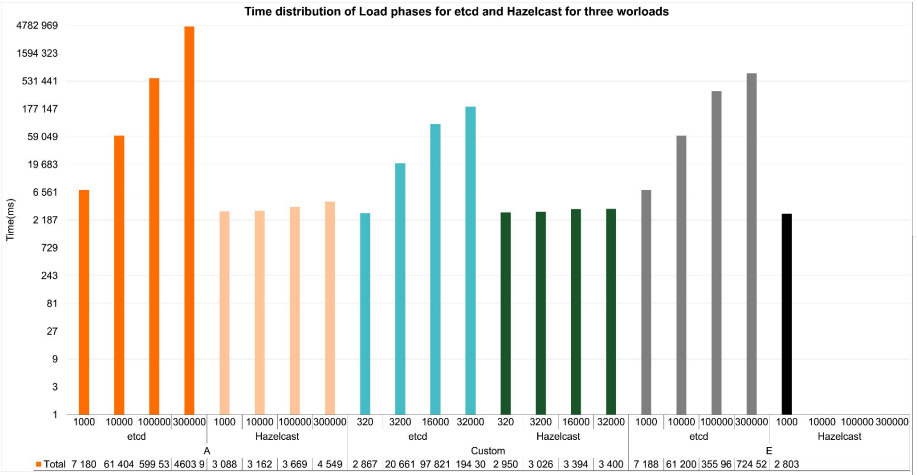
All in all:

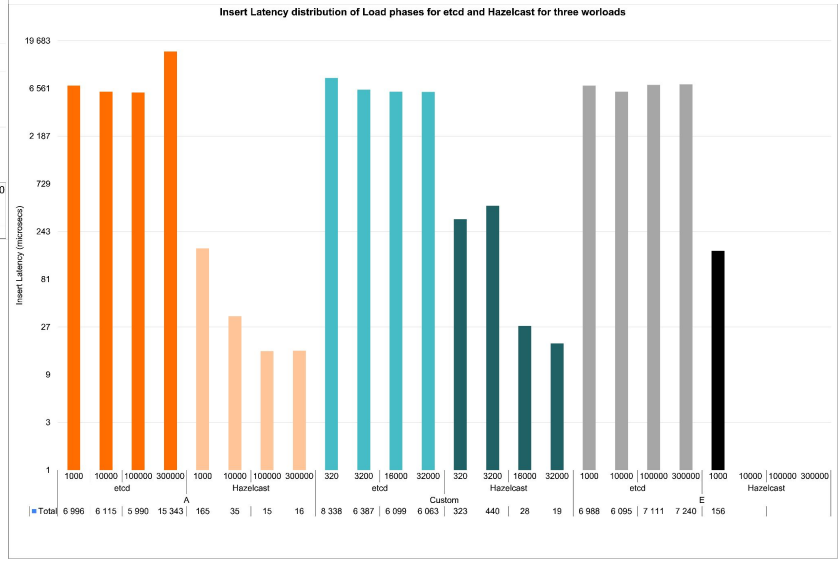
- **42% read**
- **42% update**
- **16% insert**

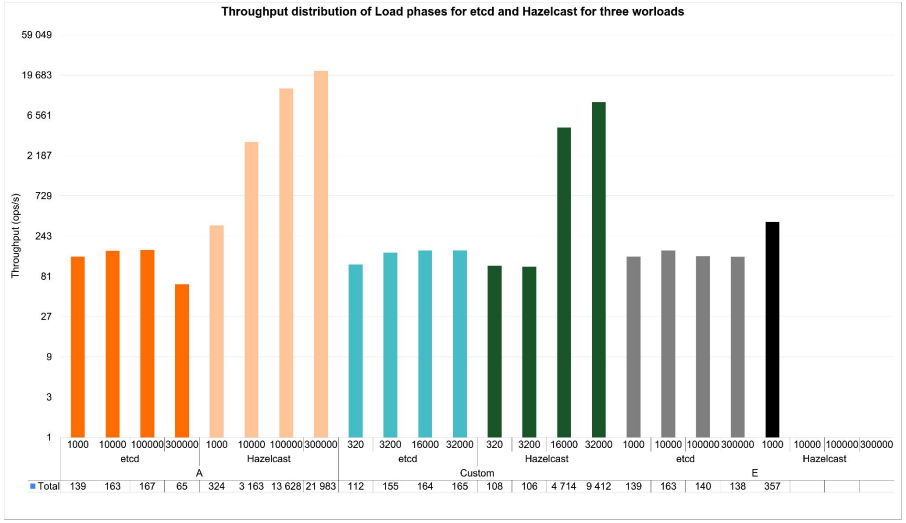
```
orchestrator(KVstore db, Servers S, ProcessGen pGen, int coresPerServer):
    init(db, S, coresPerServer)
    while(pGen.generateProcess proc):
        search db to find a server with freecores>0 -> s
        core = db.read(s, freecores) # obtain core in which execute the process
        db.update(s, freecores = core-1) # reduce free cores
        search db to find first free processID -> pid
        db.insert(proc.id, localpid=pid) # associate real id to local pid
        db.update(pid, server=s, core=core, realpid=proc.id) # associate server s and core
     to process pid

        if(process_ended(Server s, Process pid, int core)):
            result = db.read(s, core)
            real_pid = db.read(pid, realpid)
            db.insert(real_pid, result=result) # Keep track of past processes outcomes
            db.update(pid, 'free') # Free localpid in db
            db.update(s, freecores = freecores + 1) # Free one core in S
            db.update(s, core, null) # Set result to null
            return (pid, result) to pGen

init(KVstore db, Servers S, int coresPerServer):
    for(s in S):
        db.put(s, freecores = coresPerServer)
        for(core in range(1, coresPerServer):
            db.put(s, core, null) # In this record we can introduce the result

    for(processID in range(1,coresPerServer*S):
        db.put(processID, 'free')
```

YCSB Benchmark Results

Benchmark Results



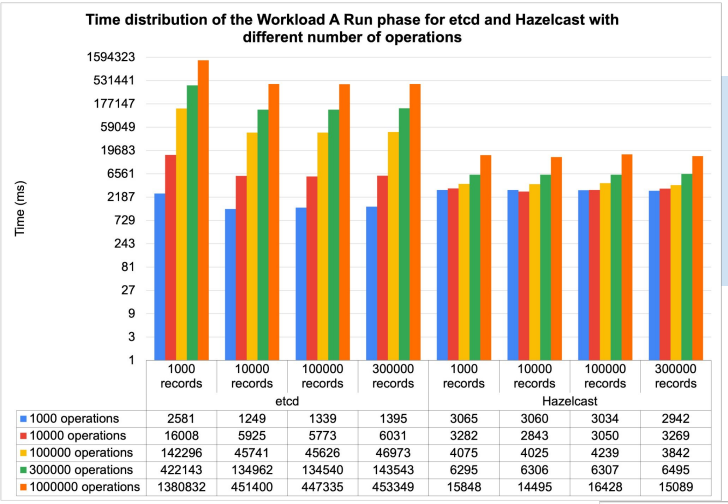




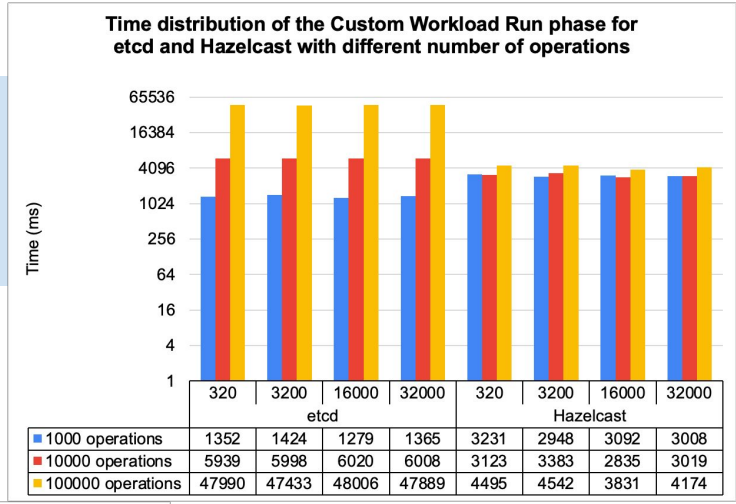
Load phase for 3 workloads

Benchmark Results

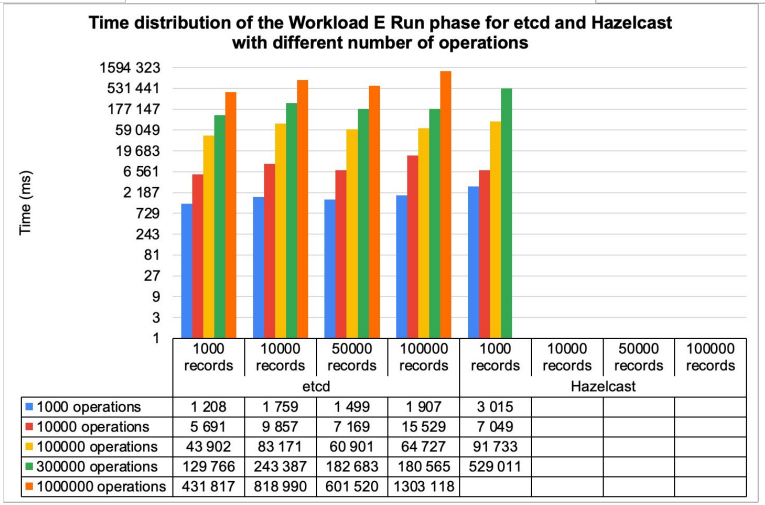
Run time for 3 workloads



	etcd				Hazelcast			
1000 operations	2581	1249	1339	1395	3065	3060	3034	2942
10000 operations	16008	5925	5773	6031	3282	2843	3050	3269
100000 operations	142296	45741	45626	46973	4075	4025	4239	3842
300000 operations	422143	134962	134540	143543	6295	6306	6307	6495
1000000 operations	1380832	451400	447335	453349	15848	14495	16428	15089



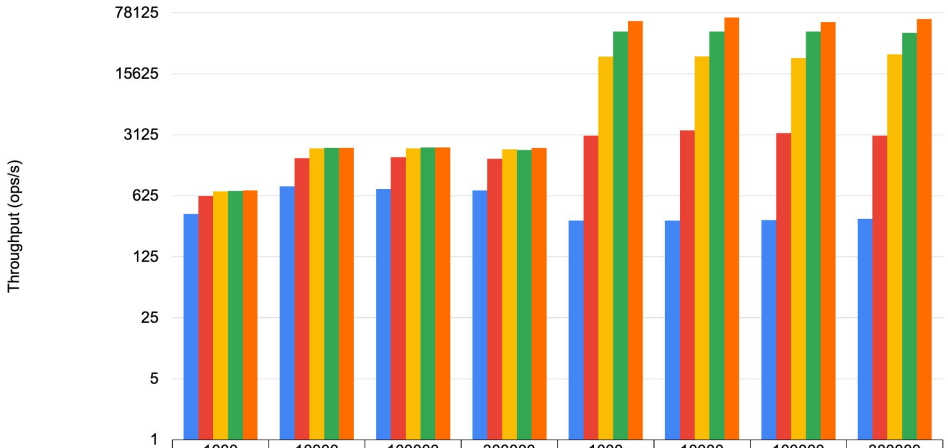
	etcd				Hazelcast			
1000 operations	1352	1424	1279	1365	3231	2948	3092	3008
10000 operations	5939	5998	6020	6008	3123	3383	2835	3019
100000 operations	47990	47433	48006	47889	4495	4542	3831	4174



	etcd				Hazelcast			
1000 operations	1 208	1 759	1 499	1 907	3 015			
10000 operations	5 691	9 857	7 169	15 529	7 049			
100000 operations	43 902	83 171	60 901	64 727	91 733			
300000 operations	129 766	243 387	182 683	180 565	529 011			
1000000 operations	431 817	818 990	601 520	1303 118				

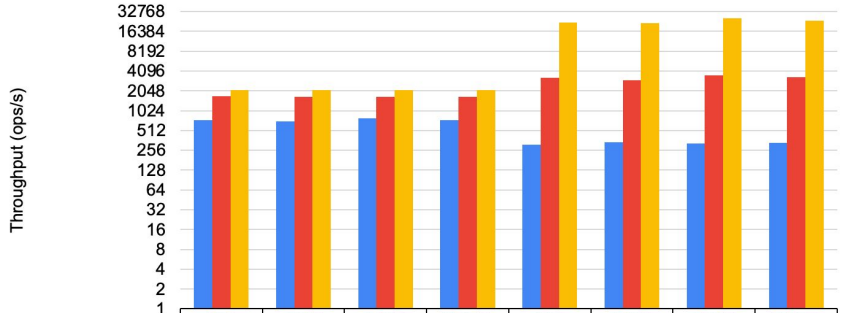
Benchmark Results

Throughput distribution of the Workload A Run phase for etcd and Hazelcast with different number of operations



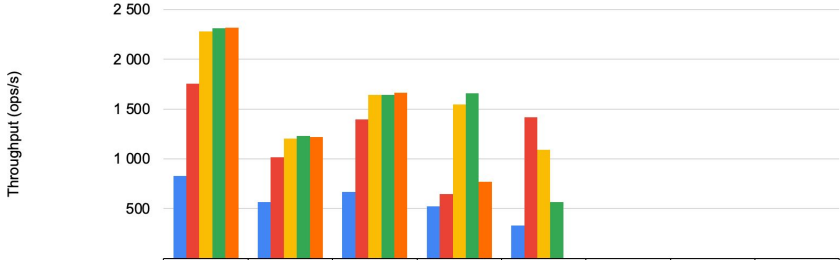
	etcd				Hazelcast			
	1000 records	10000 records	100000 records	300000 records	1000 records	10000 records	100000 records	300000 records
1000 operations	387,45	800,64	746,83	716,85	326,26	326,8	329,6	339,9
10000 operations	624,69	1687,76	1732,2	1658,1	3046,92	3517,41	3278,69	3059,04
100000 operations	702,76	2186,22	2191,73	2128,88	24539,88	24844,72	23590,47	26028,11
300000 operations	710,66	2222,85	2229,82	2089,97	47656,87	47573,74	47566,2	46189,38
1000000 operations	724,2	2215,33	2235,46	2205,81	63099,44	68989,31	60871,68	66273,44

Throughput distribution of the Custom Workload Run phase for etcd and Hazelcast with different number of operations



	etcd				Hazelcast			
	320	3200	16000	32000	320	3200	16000	32000
1000 operations	739,64	702,25	781,86	732,6	309,5	339,21	323,42	332,45
10000 operations	1683,79	1667,22	1661,13	1664,45	3202,05	2955,96	3527,34	3312,36
100000 operations	2083,77	2108,24	2083,07	2088,16	22246,94	22016,73	26102,85	23957,83

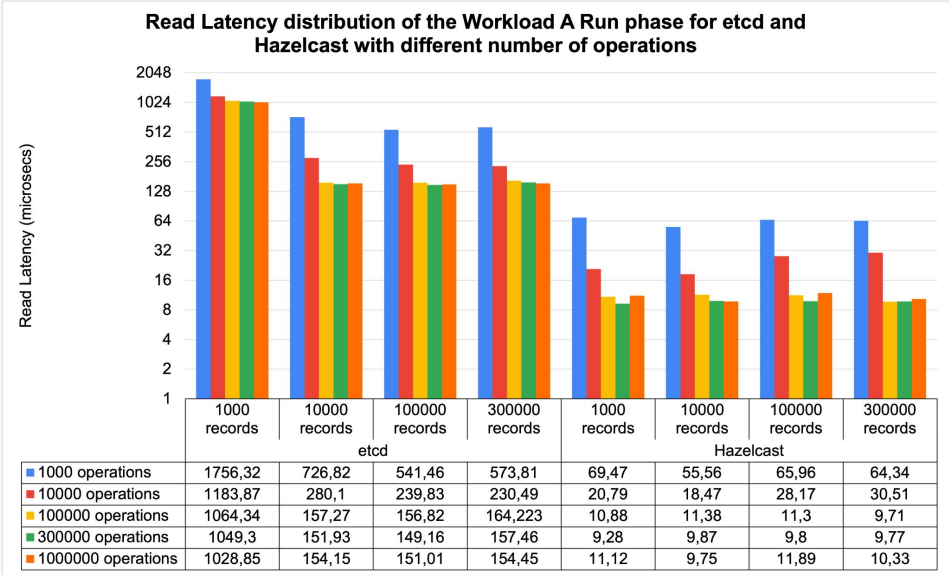
Throughput distribution of the Workload E Run phase for etcd and Hazelcast with different number of operations



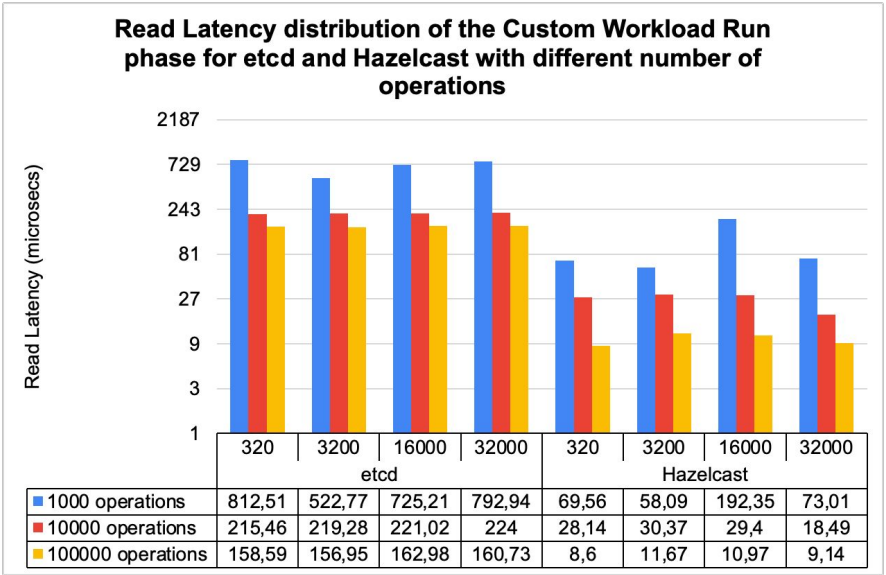
	etcd				Hazelcast			
	1000 records	10000 records	50000 records	100000 records	1000 records	10000 records	50000 records	100000 records
1000 operations	828	569	667	524	332			
10000 operations	1 757	1 015	1 395	644	1 419			
100000 operations	2 278	1 202	1 642	1 545	1 090			
300000 operations	2 312	1 233	1 642	1 661	567			
1000000 operations	2 316	1 221	1 662	767				

Throughput for 3 workloads

Benchmark Results



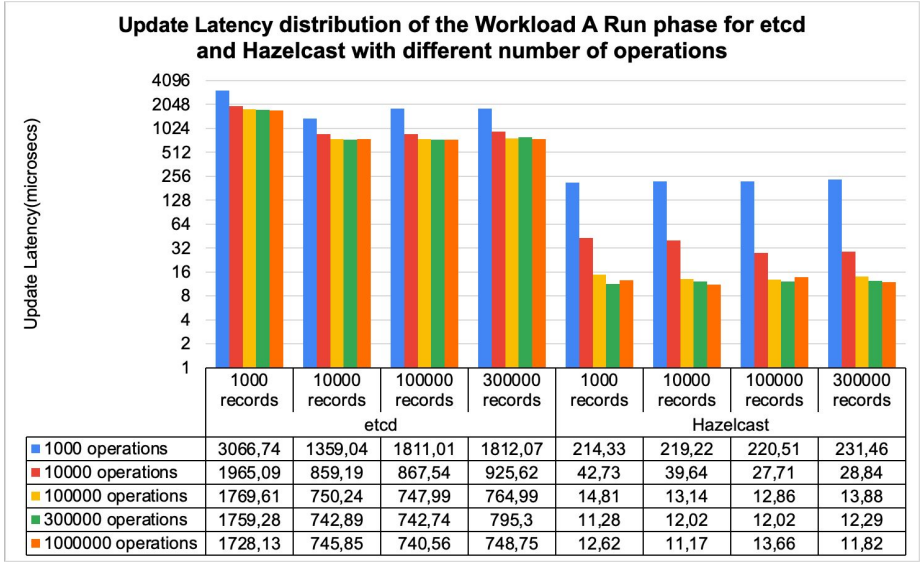
	etcd				Hazelcast			
	1000 records	10000 records	100000 records	300000 records	1000 records	10000 records	100000 records	300000 records
1000 operations	1756,32	726,82	541,46	573,81	69,47	55,56	65,96	64,34
10000 operations	1183,87	280,1	239,83	230,49	20,79	18,47	28,17	30,51
100000 operations	1064,34	157,27	156,82	164,223	10,88	11,38	11,3	9,71
300000 operations	1049,3	151,93	149,16	157,46	9,28	9,87	9,8	9,77
1000000 operations	1028,85	154,15	151,01	154,45	11,12	9,75	11,89	10,33



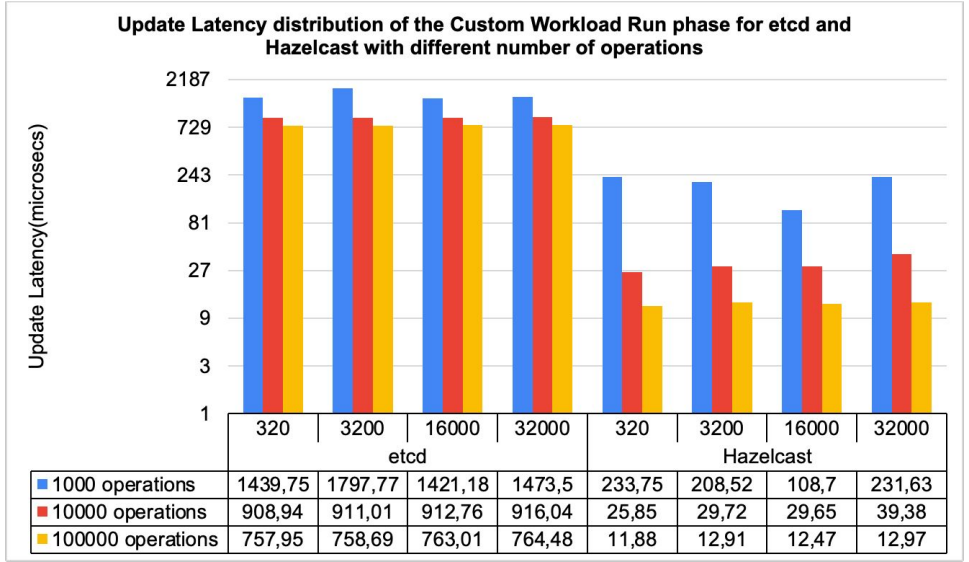
	etcd				Hazelcast			
	320	3200	16000	32000	320	3200	16000	32000
1000 operations	812,51	522,77	725,21	792,94	69,56	58,09	192,35	73,01
10000 operations	215,46	219,28	221,02	224	28,14	30,37	29,4	18,49
100000 operations	158,59	156,95	162,98	160,73	8,6	11,67	10,97	9,14

Read latency for 3 workloads

Benchmark Results



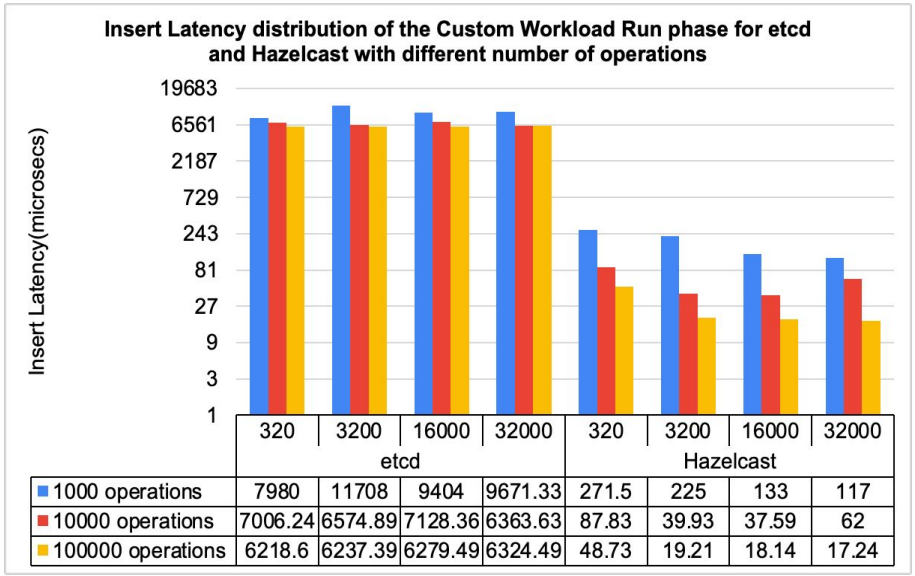
	1000 records	10000 records	100000 records	300000 records	1000 records	10000 records	100000 records	300000 records
	etcd				Hazelcast			
1000 operations	3066,74	1359,04	1811,01	1812,07	214,33	219,22	220,51	231,46
10000 operations	1965,09	859,19	867,54	925,62	42,73	39,64	27,71	28,84
100000 operations	1769,61	750,24	747,99	764,99	14,81	13,14	12,86	13,88
300000 operations	1759,28	742,89	742,74	795,3	11,28	12,02	12,02	12,29
1000000 operations	1728,13	745,85	740,56	748,75	12,62	11,17	13,66	11,82



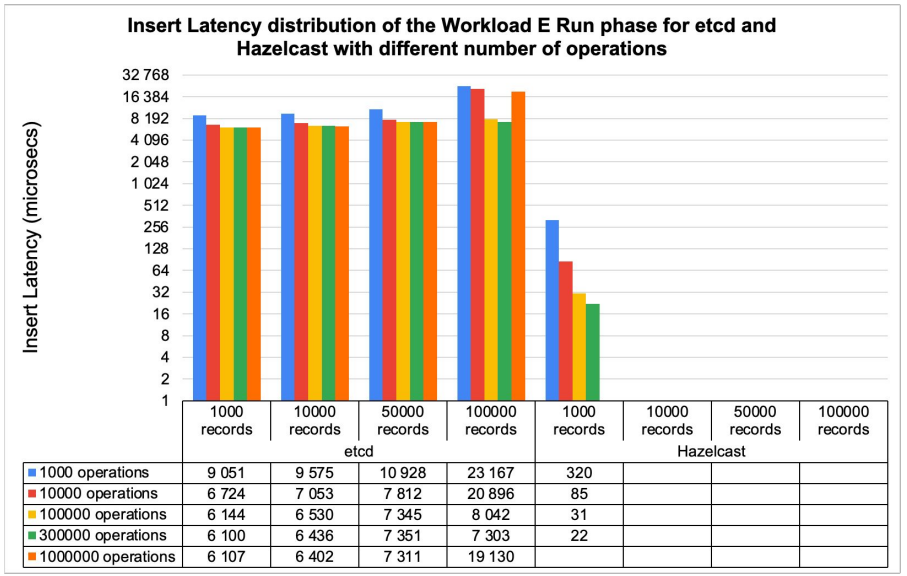
	320	3200	16000	32000	320	3200	16000	32000
	etcd				Hazelcast			
1000 operations	1439,75	1797,77	1421,18	1473,5	233,75	208,52	108,7	231,63
10000 operations	908,94	911,01	912,76	916,04	25,85	29,72	29,65	39,38
100000 operations	757,95	758,69	763,01	764,48	11,88	12,91	12,47	12,97

Update latency 3 workloads

Benchmark Results



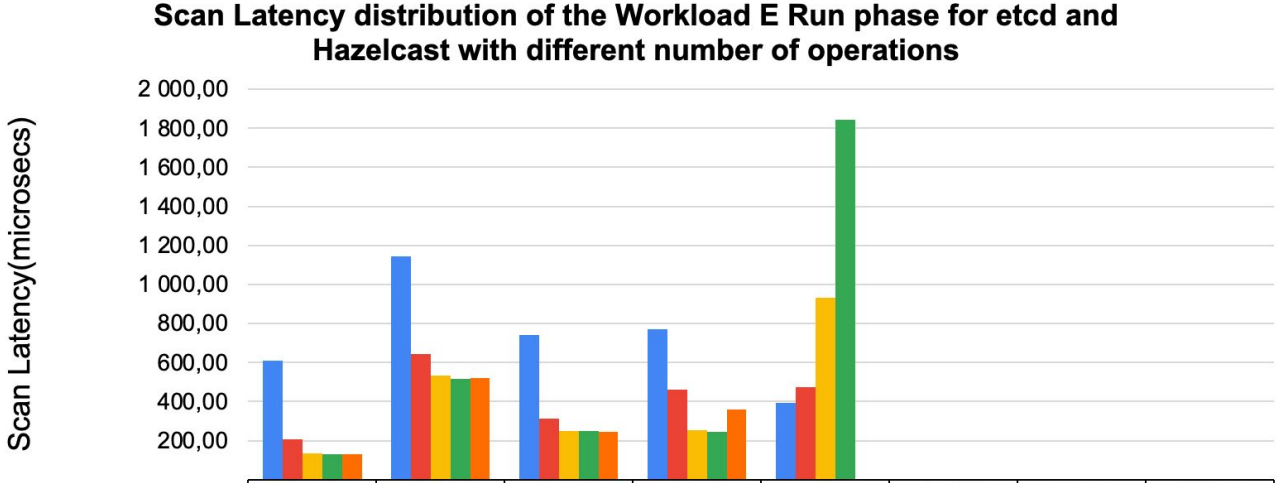
	etcd				Hazelcast			
	320	3200	16000	32000	320	3200	16000	32000
1000 operations	7980	11708	9404	9671.33	271.5	225	133	117
10000 operations	7006.24	6574.89	7128.36	6363.63	87.83	39.93	37.59	62
100000 operations	6218.6	6237.39	6279.49	6324.49	48.73	19.21	18.14	17.24



	etcd				Hazelcast			
	1000 records	10000 records	50000 records	100000 records	1000 records	10000 records	50000 records	100000 records
1000 operations	9 051	9 575	10 928	23 167	320			
10000 operations	6 724	7 053	7 812	20 896	85			
100000 operations	6 144	6 530	7 345	8 042	31			
300000 operations	6 100	6 436	7 351	7 303	22			
1000000 operations	6 107	6 402	7 311	19 130				

Insert latency 3 workloads

Benchmark Results



	etcd				Hazelcast			
	1000 records	10000 records	50000 records	100000 records	1000 records	10000 records	50000 records	100000 records
1000 operations	607,86	1 144,60	742,20	769,19	393,56			
10000 operations	206,34	642,50	314,01	462,99	475,52			
100000 operations	136	532	250	254	932			
300000 operations	131	516	249	244	1 843			
1000000 operations	130	520	244	360				

Scan latency for 3 workloads

Conclusion: **Comparison with relational model**

	Key-Value Stores	Relational Model
Data model	Schema-less	Relational schema
Flexibility	Totally flexible	Not flexible
Applicability	Most use cases, but a unique key identifier is needed for all stored data	Most use cases, but performance varies according to the natural structure of the data
Data retrieval	Faster	Slower
Code	Usually cleaner	Long queries are frequently encountered
Cloud capabilities	Naturally high, CP or AP	The relational model can be extended to cope with distributed operations

Conclusion: **Comparison of the two tools**

	etcd	Hazelcast IMDG
Data model	Key-Value store	More general object store
Data persistency	Yes	Optional
CAP	CP, and fairly good A	AP, configurable to CP
Physical storage	On disk B+Tree	In-Memory structures: IMap for key-value stores
Index	Lexically ordered index on keys	Implicit hash index on keys
Insertion	Inefficient	Efficient
Read	Efficient	Very efficient
Update	Inefficient	Efficient

HAZELCAST

etcd

Conclusion: **Comparison of the two tools**

	etcd	Hazelcast IMDG
Delete	Inefficient	Efficient
Scan	Efficient	Impracticable
Targeted applications	Distributed non-changing values: - Distributed configurations - Discovery services	Distributed highly changing values with high availability: - Passing messages through the cluster - Storage for session data - Distributed data streaming applications
In our use case	Not a good option: fast changing data	Fairly good option: the IMap provides all necessary features needed for the use case Also, some other structures of Hazelcast could be useful (IList or IQueue)

HAZELCAST

etcd

Conclusion: **YCSB**

- Good tool for benchmarking cloud applications: particularly KV stores
- Outdated:
  - Many modern tools are not developed
  - Most of the tools are implemented in a very old version (e.g. PostgreSQL v9)
- Delete operation cannot be used for workloads, even though it is useful in some use cases
- The program that runs the workload is developed in Python 2, and does not work with Python 3

Thus, we think that YCSB should be revised and updated, to cope with new trends, technologies and tools!

Thanks for your attention

HAZELCAST

etcd

YCSB