Autoscaling Tiered Cloud Storage in Anna

Ranjitha Kosgi

What is Anna

- A Distributed KV Store where values are lattices
- Not Strongly Consistent
- Highly Performant
- Sharded(consistent hashing) and Replicated
- Coordination free consistency
- Shared Nothing Architecture
- Multi Master Replication

Taxonomy of existing KVS systems

System	Scale	Memory Model	Per-Key Consistency	Multi-key Consistency
Masstree	M	SM	Linearizable	None
Bw-tree	M	SM	Linearizable	None
PALM	M	SM	Linearizable	None
MICA	M	SM	Linearizable	None
Redis	S	N/A	Linearizable	Serializable
COPS, Bolt-on	D	MP	Causal	Causal
Bayou	D	MP	Eventual, Monotonic Reads/Writes, Read Your Writes	Eventual
Dynamo	D	MP	Linearizable, Eventual	None
Cassandra	D	MP	Linearizable, Eventual	None
PNUTS	D	MP	Linearizable Writes, Monotonic Reads	None
CouchDB	D	MP	Eventual	None
Voldemort	D	MP	Linearizable, Eventual	None
HBase	D	MP	Linearizable	None
Riak	D	MP	Eventual	None
DocumentDB	D	MP	Eventual, Session, Bounded Staleness, Linearizability	None
Memcached	M & D	SM & MP	Linearizable	None
MongoDB	M & D	SM & MP	Linearizable	None
H-Store	M & D	MP	Linearizable	Serializable
ScyllaDB	M & D	MP	Linearizable, Eventual	None
Anna	**M & D**	**MP**	**Eventual, Causal, Item Cut, Writes Follow Reads Monotonic Reads/Writes, Read Your Writes, PRAM**	**Read Committed, Read Uncommitted**

Auto Scaling

The automatic adjustment of resources to handle changes in workload demand

Why Autoscaling is Challenging?

- Scaling Stateless applications is easy - Just bring up/down new instance
- Scaling Stateful applications is challenging - Involves redistribution of data along with handling current requests.
- Adjusting resources should not impact the latency requirements.

Motivation & Existing Challenges

- Large-Scale variations in the workload
  - Many applications generate a skewed access distribution, where some data is **hot**, while other is **cold**

- Cost-Performance Barriers
  - Data should move adaptively across storage tiers, to match with workload skew and shifting hotspots

- Static Deployment Barriers
  - No existence of truly Auto scaling service
  - Elasticache needs manual allocation and deallocation of services.
  - S3 autoscales to data volume but ignores workload

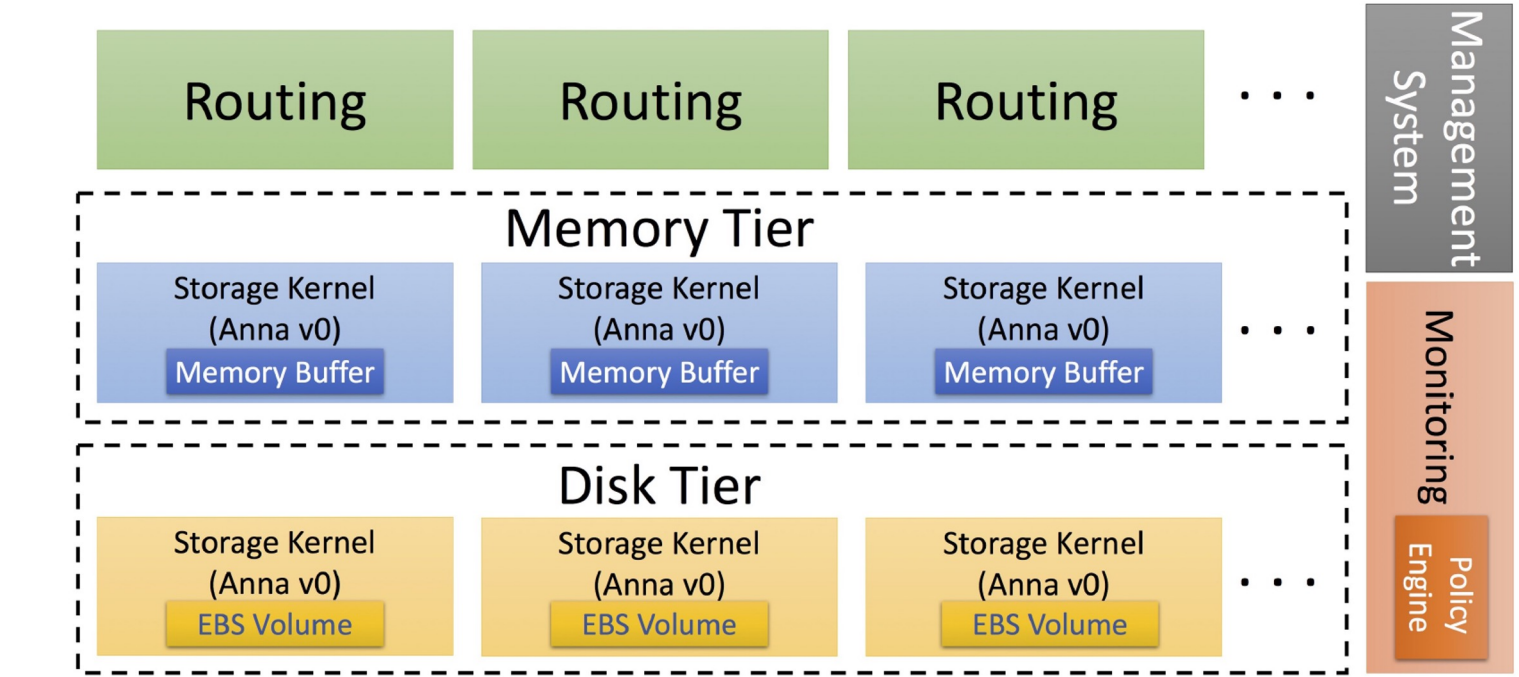
Objective

- Design system to dynamically adjust configuration and match resources to the workloads.
- Emphasis on **Efficiency** - performance to cost ratio

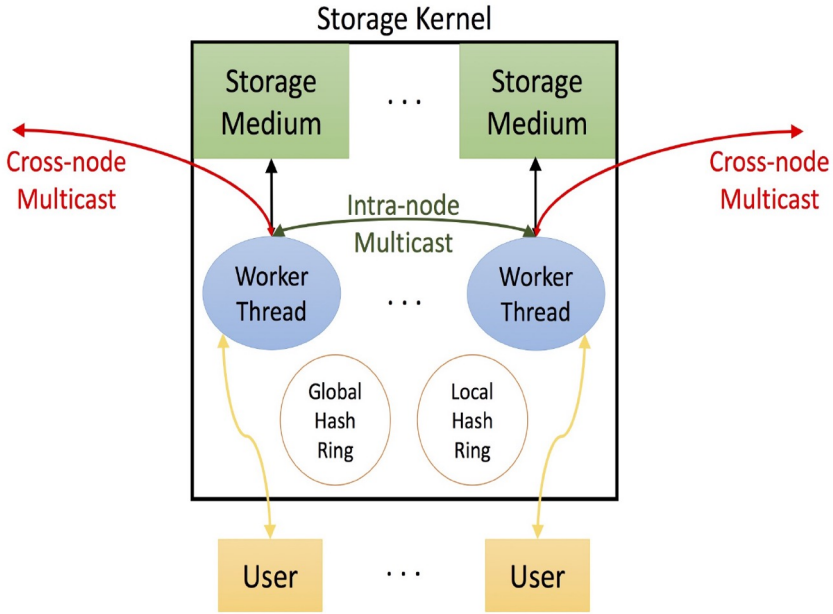
Goals

- Horizontal elasticity to adaptively scale deployments - **Performance**
- Vertical data movement in storage hierarchy to reduce cost by demoting cold keys to cheaper storage - **Cost**
- Multi-master selective replication of hot keys across nodes and cores to efficiently scale request handling for non-uniform access patterns - **Performace**

Architecture to Support autoscaling



Architecture - Storage Kernel



Multiple storage tiers and uniformity across tiers. Only difference is the procedure for translating data for persistence

Modified Consistent hashing algorithm to partition and replicate keys

Coordination free execution model - each thread has its own private memory

Gossip protocol to exchange updates with other replicas

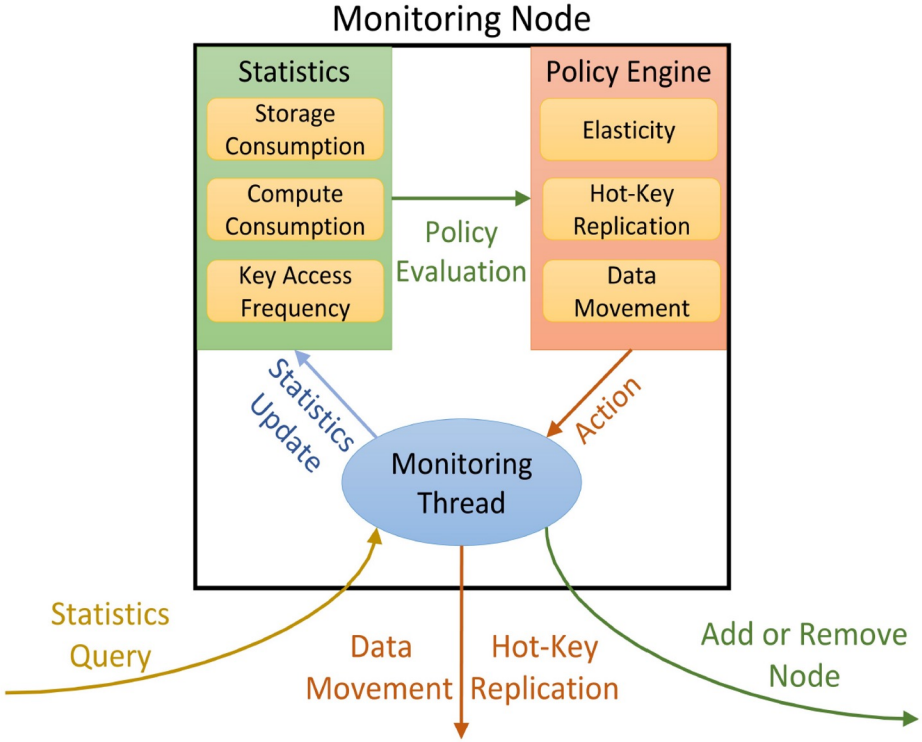
Shared-nothing, asynchronous messaging scheme eliminates thread synchronization and asynchronously resolves conflicting updates to replicas.

Architecture - Metadata Management

- Global hash ring **G** - each storage tier maintains **G** to determine the nodes in the tier responsible for a key
- Local hash ring **L** - each storage tier maintains **L** to determine the threads in a node responsible for a key
- Replication Vector - each key has the replication vector **[< R1, ……Rn >, < T1,.....Tn >]**. Ri represents the number of nodes in tier i storing key K, and Ti represents the number of threads per node in tier i storing key. Currently *i* is either M(memory) or E(EBS)
- Monitoring statistics - Access frequency of each key and storage consumption of each node.

Storage system is used to store all kinds of metadata & autoscaling is achieved by just making updated to the metadata

Architecture - Monitoring & Policy Engine



- Monitoring thread periodically retrieves the stored statistics from the storage engine and triggers the policy engine
- Policy engine analyzes these stats and issues actions to meet SLOs
  - Latency
  - Cost budget
  - Fault tolerance (allowed number of replica failures)
- Cluster manager performs the actions specified by Policy engine

Cross-Tier Data movement

- Policy engine uses its monitoring statistics to calculate how frequently each key was accessed in the past T seconds.
- Access frequency exceeding the threshold - at least one replica promoted to memory tier
- Access frequency below threshold - replicas in memory tier are demoted to EBS tier

Hot-Key Replication

- When access frequency of a key stored in the memory tier increases, hot-key replication increases the number of memory-tier replicas of that key.
- Only increase in memory tier replication factor as only that will ensure better performance
- The policy engine computes the target replication factor, R_ideal, using the ratio between the observed latency for the key and the latency objective.
- Replication across nodes rather than replicating across cores in a node as network is typical bottleneck in distributed systems.

Elasticity

- To handle insufficient storage and compute capacity
- Insufficient storage - new storage nodes based on the data size
- Insufficient compute - new memory tier nodes.
- Node removal - Scale down memory tier based on compute consumption.

Fault Tolerance

- All components except storage kernel are stateless or maintain soft state which can be reconstructed
- Anna guarantees k-fault tolerance by ensuring k+1 replicas to be alive at all times.
- On a storage node failure, other nodes detect via timeout and remove node from the hash ring. Repartitioning of keys done by Anna on the update of hash ring.

Evaluation - Setup and workload

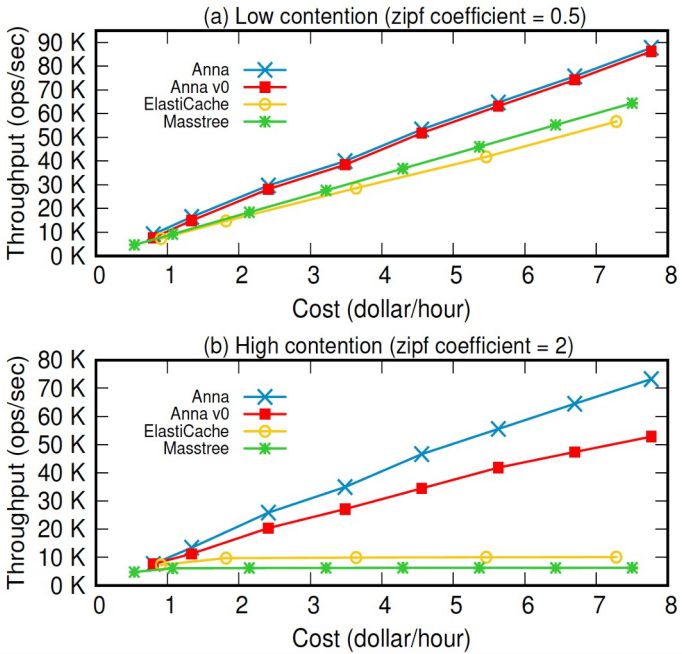
- 1 million key-value pairs, with keys being 8 bytes and values being 256KB
- Replication factor - 3
- YCSB-style read-modify-write of a single key chosen from a Zip an distribution
  - Zipfian coefficient is used to vary the contention levels - a higher coefficient value mean a more skewed workload
- Client machines - 40 client machines with 8 threads each

Evaluation - Replica Placement

- Comparison of Intra-node vs cross-node replication

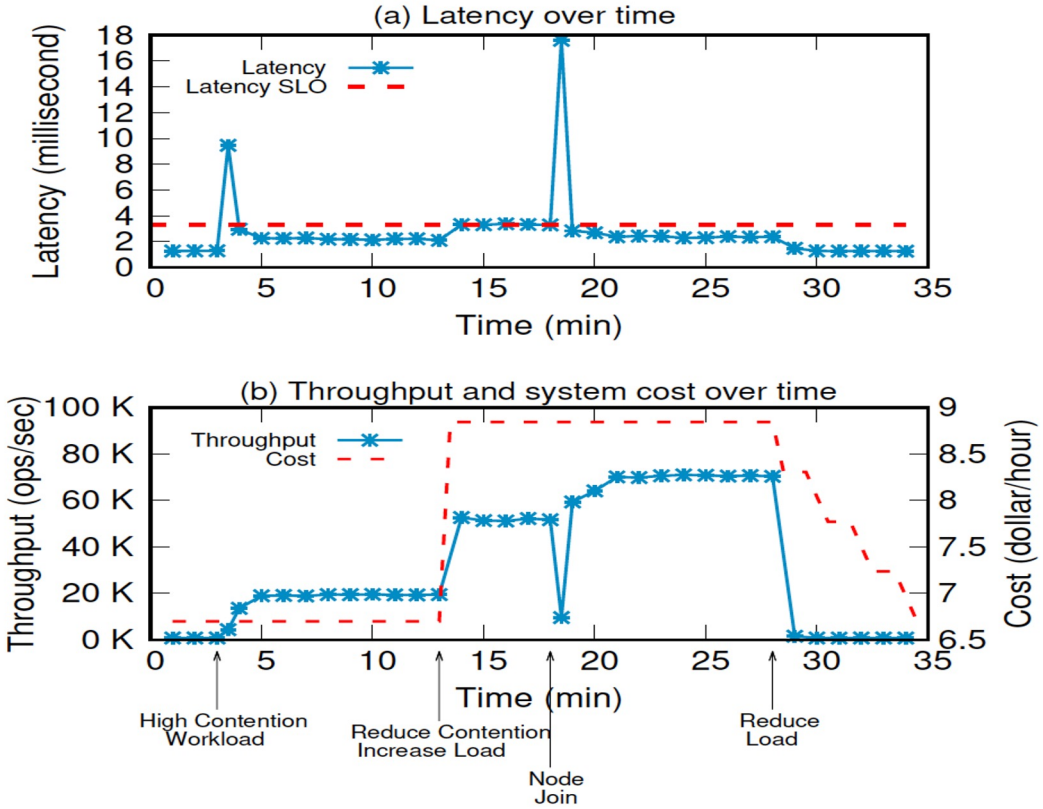
- Single replica of key in single node - 2000 ops/sec
- 4 replicas across 4 nodes - 8000 ops/sec
- 4 replicas in single node - 4000 ops/sec

Evaluation - Selective Replication

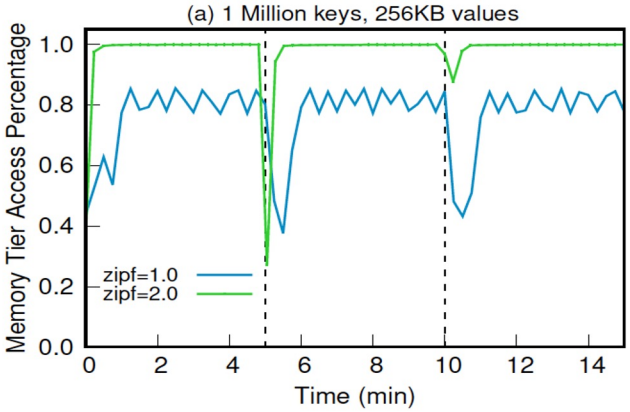


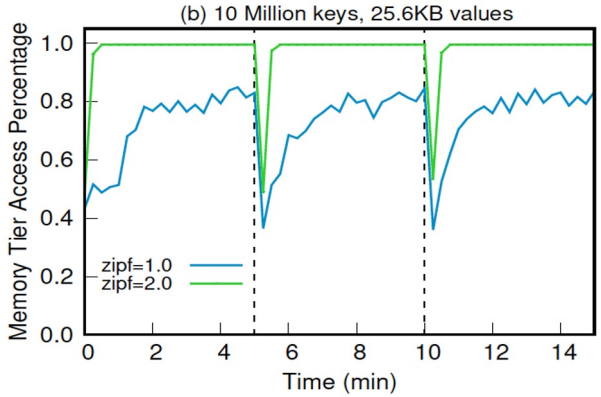
- Comparison of Anna's memory-tier against Anna v0, AWS ElastiCache (using managed Memcached), and Masstree, at various cost points.
- Tuned Anna v0's single replication factor to the optimal value for each Zipfian setting
- Anna consistently outperforms both Masstree and ElastiCache under low contention
- Under high contention Anna's throughput increases linearly with cost

Evaluation - Dynamic Workload Skew & Volume

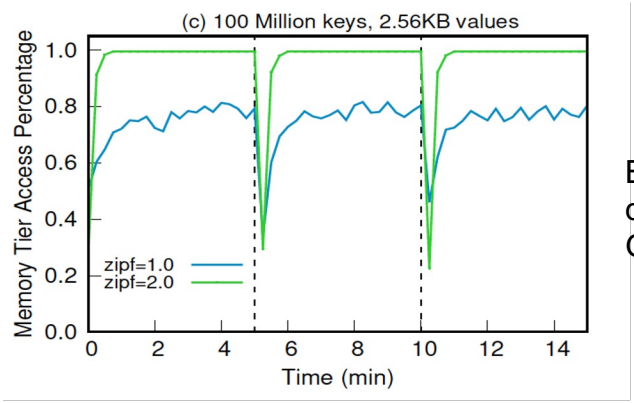


Evaluation - Varying Hotspot



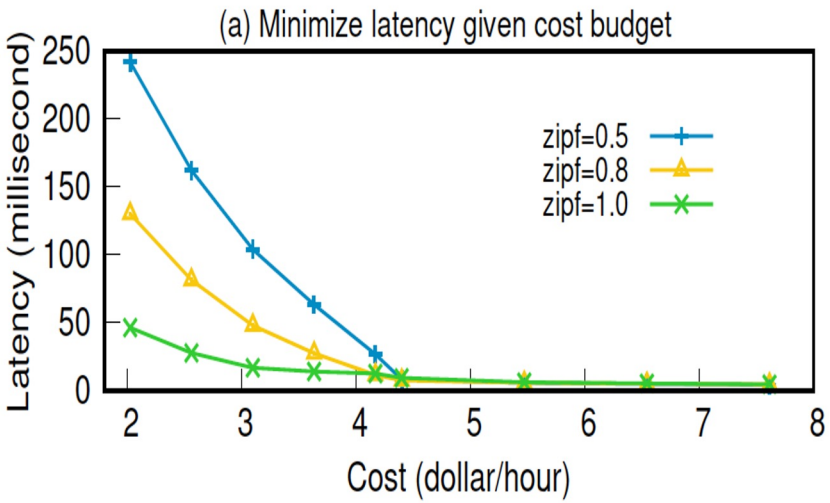


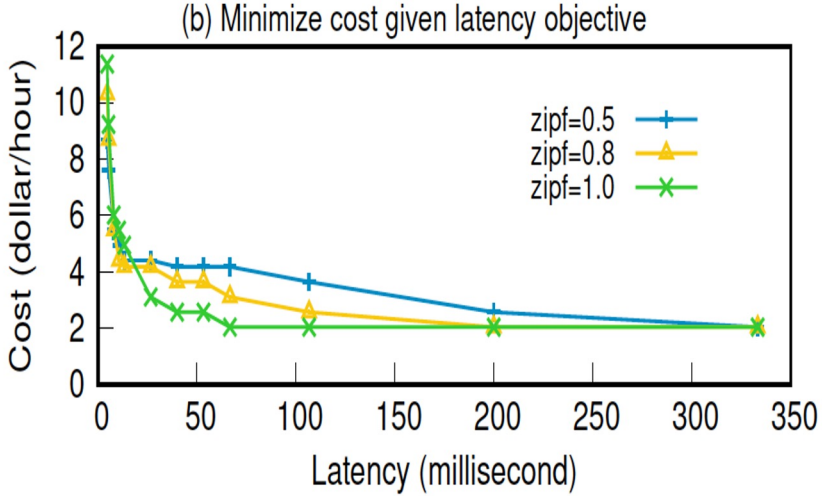
- Evaluate Anna's ability to detect and react to changes in workload hotspots.
- At minute 0, the workload centered around one hotspot. At minute 5, it is switched to a different, largely non-overlapping hotspot, and at minute 10, it is again switched to a third, unique hotspot.



Blue - moderately skewed data
Green - Highly skewed data

Evaluation - Cost-Performance Tradeoffs





- Given cost budget, minimize latency
- Given latency objective, minimize cost

Conclusion & Thoughts

- A highly performant, scalable key value store, with consistency put at stake
- Use of coordination free mechanisms to achieve high performance
- Detailed evaluation of all the goals stated.

Questions?

Thank You