PufferFish

Dynamic Storage-Performance Tradeoff in Data Stores

Anurag Khandelwal, Rachit Agarwal, Ion Stoica

Modern data stores

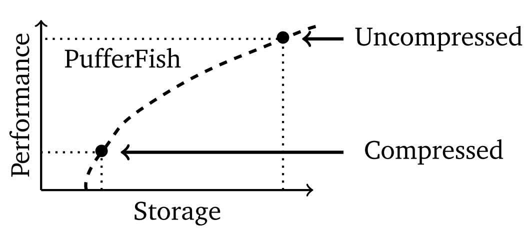
- Data sharded, replicated, cached
- Two fundamental primitives:
  - *Random Access, e.g.*, `get()` in NoSQL stores
  - *Search* across records, columns, etc.
- Total Data size ≫ amount of fastest storage (*e.g.*, RAM)
- **Goal:** Maximize performance by maximizing
  - Number of shards *cached*
  - Number of queries *executed off of faster storage*

Problem

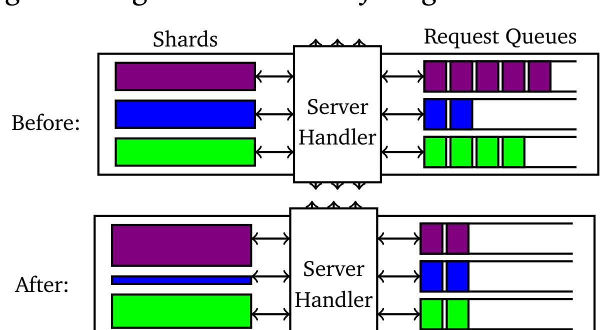
- Existing systems expose a hard tradeoff for each shard:
  - **Uncompressed:** More cache, high performance
  - **Compressed:** Less cache, low performance
- Navigation between these two operating points
  - Requires compression/decompression
  - Cannot be done at fine-grained timescales
- Degraded performance when workload and/or underlying infrastructure changes

PufferFish

- *Smooth tradeoff* between storage & performance



- *Increase/decrease* storage "fractionally", just enough to meet performance goals
- *Navigate* along the curve over *fine-grained* time scales



PufferFish Techniques

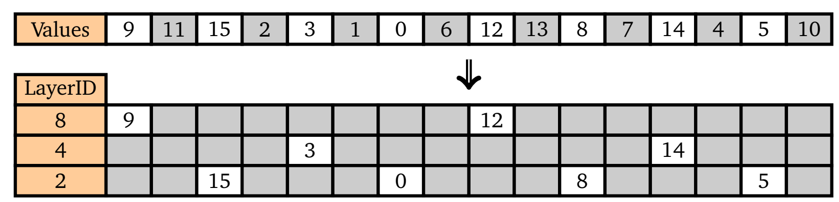
Builds on top of Succinct, that stores

- Two *sampled* arrays
  - Sampling rate (α) proxy for storage/performance
  - Storage: $2n\lceil \log n \rceil / \alpha$;
  - Latency for computing an unsampled value: α
- Another small array for computing unsampled values

Storage-Performance Tradeoff

PufferFish introduces Layered Sampled Array (LSA)

- Stores sampled array across multiple *layers*



Dynamic Navigation

Navigates tradeoff curve over fine-grained timescales

- Layers added/deleted independent of existing layers
- New layers populated opportunistically during query execution (Succinct computes unsampled values)
- Layer deletion easy

Request scheduling and Shard management

Each shard, and each shard replica, may operate at different point on the storage-performance tradeoff curve

Challenges:

- Sharing cache on the same server
- Sharing cache across servers
- Scheduling requests across replicas

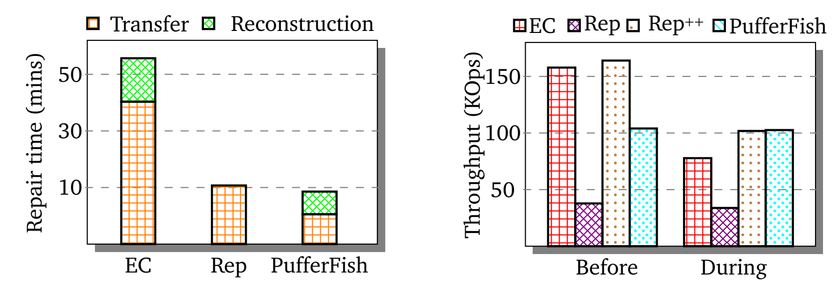
Unified solution using techniques from scheduling literature — Join-the-shortest-queue mechanism:

- One "request queue" per shard
- Layer addition/deletion based on request queue lengths
- Scheduling across replicas using request queue lengths

PufferFish Applications

Storage & Bandwidth Efficient Data Recovery

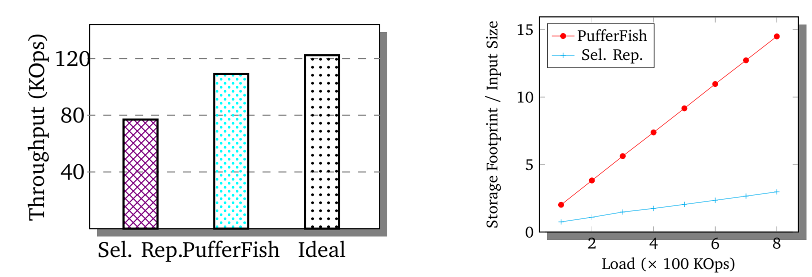
	Repair Bandwidth	Storage Overhead
Replication	1×	3×
Erasure Codes	10×	1.2×
PufferFish	1×	1.8×



PufferFish achieves 1.5× lower throughput under no failures, comparable throughput during failures with 3.5× faster repair.

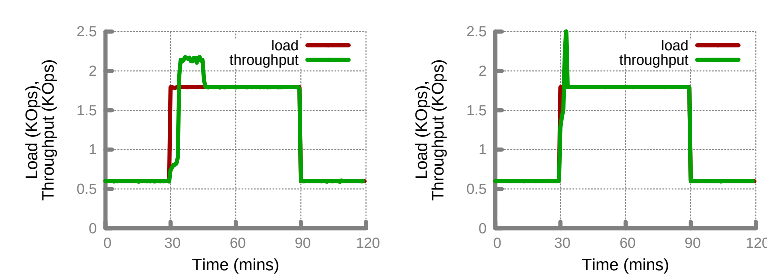
Spatial Skew: varying load across data items

- Typical approach: Selective Replication
- #Replicas ∝ Load
  - Coarse grained (2× storage → 2× throughput)



Temporal Skew

PufferFish can adapt to time-varying workloads



(a) Uniform (b) Skewed

- Adapts to spiked (∼ 3×) variations
- In fine-grained (< 5 mins) time-scales

{anuragk, ragarwal, istoica}@berkeley.edu