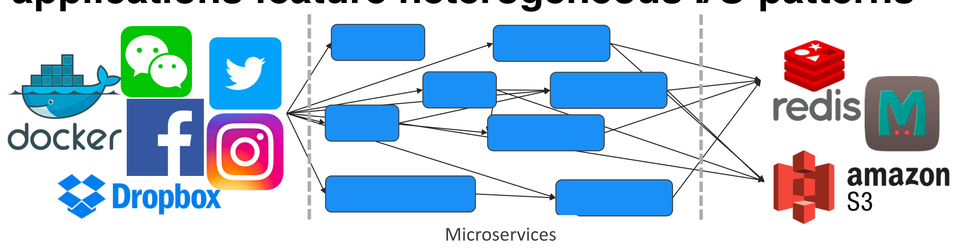
InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache

Ao Wang* **and Jingyuan Zhang***, Xiaolong Ma†, Ali Anwar‡, Lukas Rupprecht‡,
Dimitrios Skourtis‡, Vasily Tarasov‡, Feng Yan†, Yue Cheng*
*George Mason University †University of Nevada, Reno ‡IBM Research-Almaden

Introduction

Web applications feature heterogeneous I/O patterns

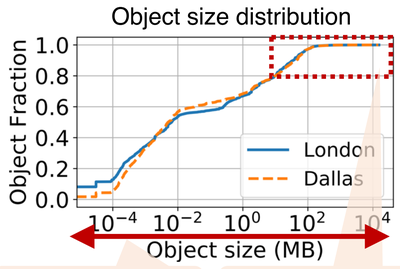


To quantify the I/O behavior in those storage-intensive applications, we conduct a case study:

IBM docker registry traces

- The traces are collected from five data centers in the IBM Cloud container registry service across 75 days during 2017
- We select two busiest data centers: Dallas & London

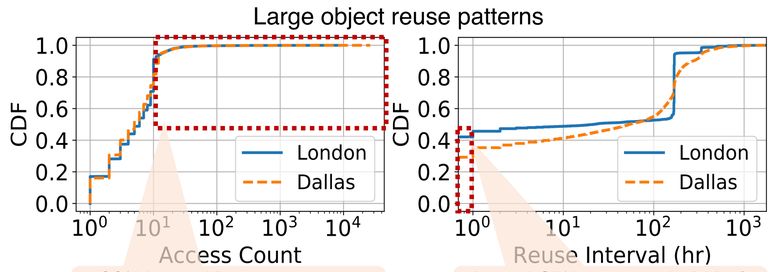
Object size distribution



Object sizes span over 9 orders of magnitude

20% of objects larger than 10MB

Large object reuse patterns

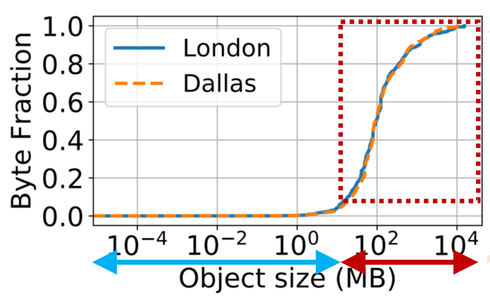


>30% large object access more then 10 times

Around 45% got reused within 1 hour

Caching large objects is beneficial

To cache both small and large objects, we analyze the storage footprint of objects:



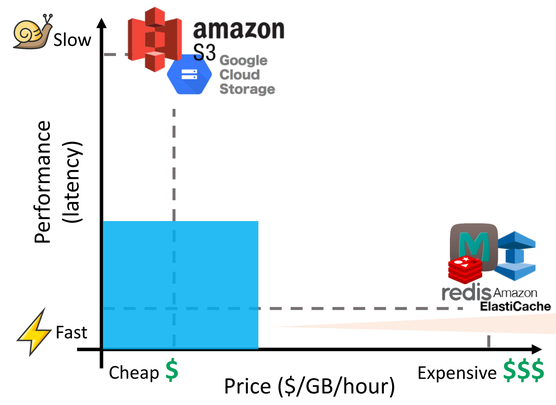
Large objects occupy 95% storage footprint

Large objects compete memory and bandwidth resources

Extreme tension between small and large objects

Motivation

Performance-cost trade-off in existing solutions



Object stores are cheap but too slow

AWS S3: $0.023 per GB per month

In-memory caches are fast but too expensive

AWS ElastiCache: $0.016 per GB per hour

How can we achieve the best of both worlds?

***InfiniCache:** A **first-of-its-kind** in-memory caching solution atop **Serverless Computing** platform*

Insights:

Serverless functions' <CPU, Mem> resources are pay-per-use ➡ Cost Effectiveness

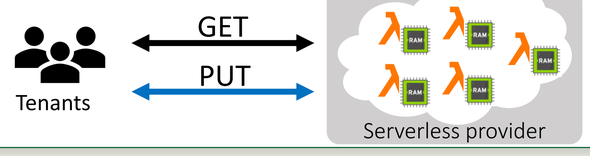
Serverless providers offer "free" function caching for tenants ➡ High Performance

A primer on Serverless Computing

* Serverless computing enables cloud tenants to launch short-lived tasks (i.e., Lambda functions) with **high elasticity** and **fine-grained resource billing**

* Function: basic unit of deployment. Application consists of multiple serverless functions

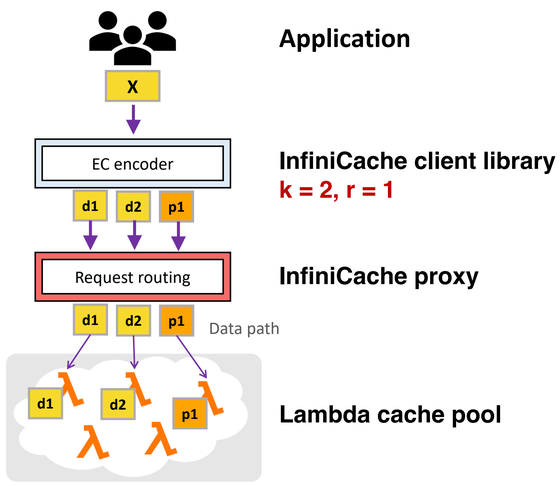
* Popular use cases: Backend APIs, data processing…



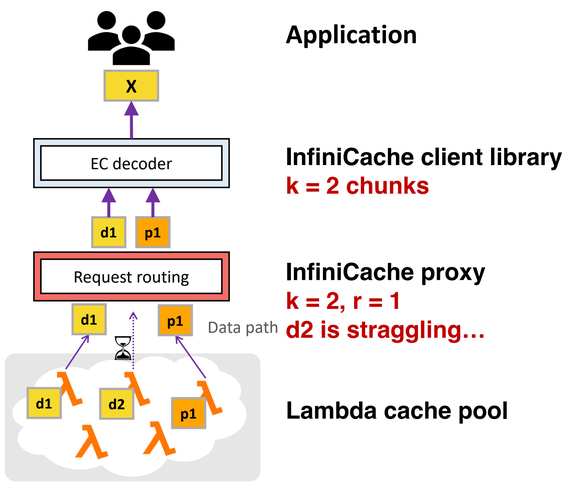
Design

InfiniCache achieves **high performance** by utilizing the aggregated network bandwidth of multiple serverless functions in parallel

PUT request example



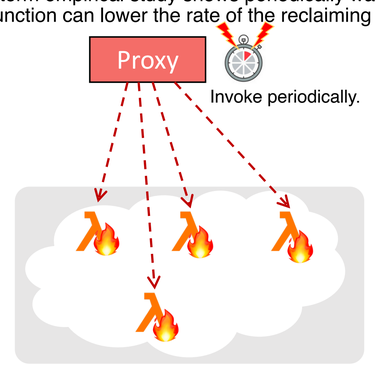
Get request example



InfiniCache achieves **high data availability** by leveraging 1. Erasure coding

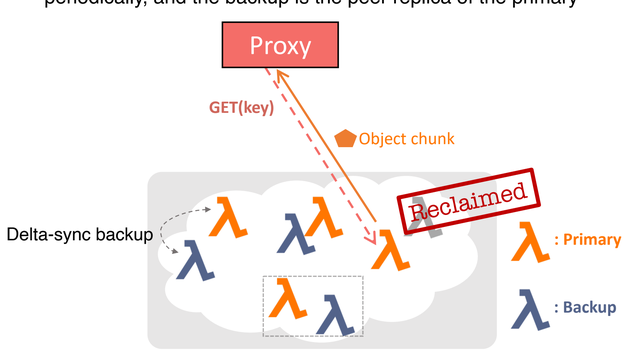
2. Periodic warm-up

AWS may reclaim Lambda functions after idling for a period. Our long-term empirical study shows periodically warming up lambda function can lower the rate of the reclaiming event



3. Periodic backup & Seamless failover

The failover is done seamlessly by exploiting the auto-scaling (elasticity) feature of AWS Lambda. Note each cache node (primary) get backed up periodically, and the backup is the peer replica of the primary

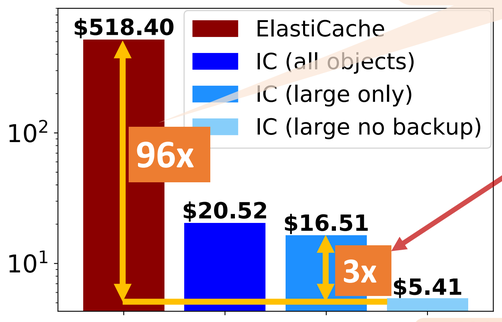


Primary and backup are peer replicas in AWS Lambda function deployment

Evaluation

InfiniCache improves cost-effectiveness 31 - 96 times compared to ElastiCache

Cost Effectiveness



Hit ratio and $$ cost tradeoff

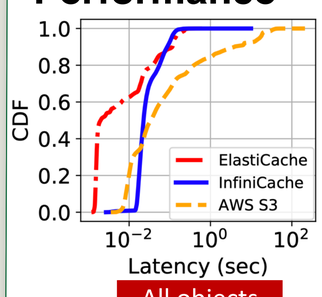
Workload	ElastiCache	InfiniCache	InfiniCache w/o backup
All objects	67.9%	64.7%	---
Large object only	65.9%	63.6%	56.1%

InfiniCache achieves same or higher performance than ElastiCache for large objects

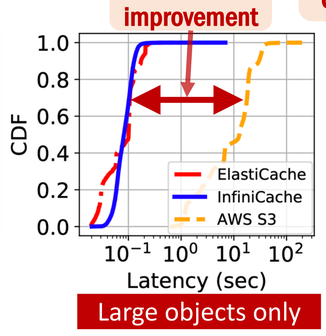
Lambda invocation overhead (~13ms) dominates for fetching small objects

Performance

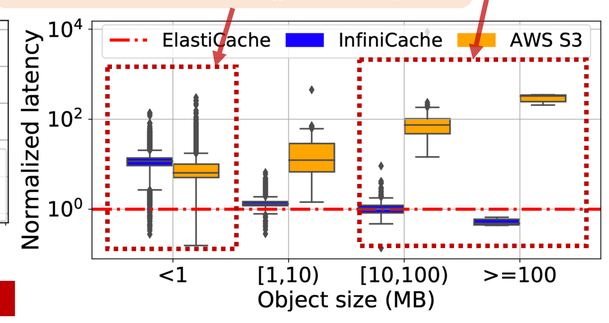
> 100 times improvement



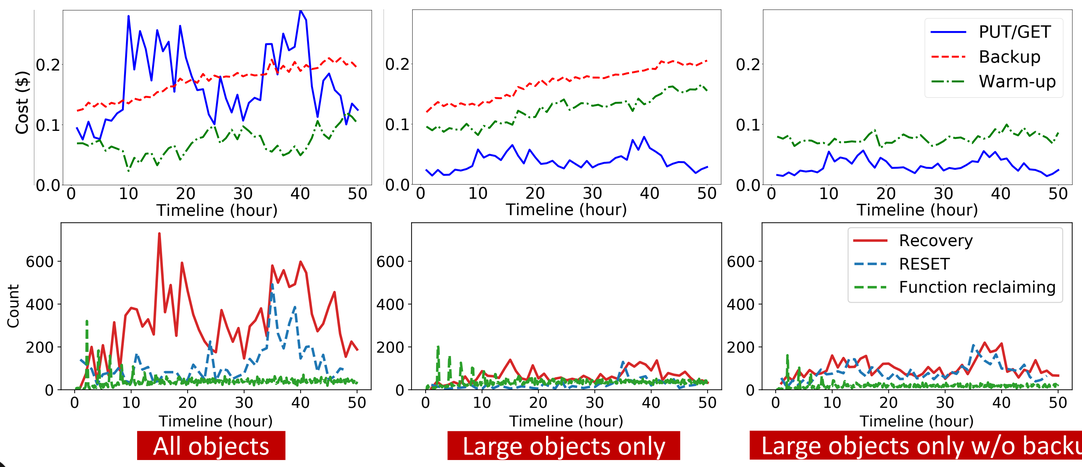
All objects



Large objects only



Timeline of cost breakdown & fault tolerance activities



All objects | Large objects only | Large objects only w/o backup

Fork InfiniCache on GitHub! https://github.com/mason-leap-lab/infinicache

FAST'20 18th USENIX Conference on File and Storage Technologies

George Mason University | University of Nevada, Reno | IBM