Pocket:
Elastic Ephemeral Storage for Serverless Analytics

Ana Klimovic and Yawen Wang, Stanford University; Patrick Stuedi, Animesh Trivedi, and Jonas Pfefferle, IBM Research; Christos Kozyrakis, Stanford University

Group 17
Sara Machado, 86923
Rafael Figueiredo, 90770
Ricardo Grade, 90774

pocket

Introduction

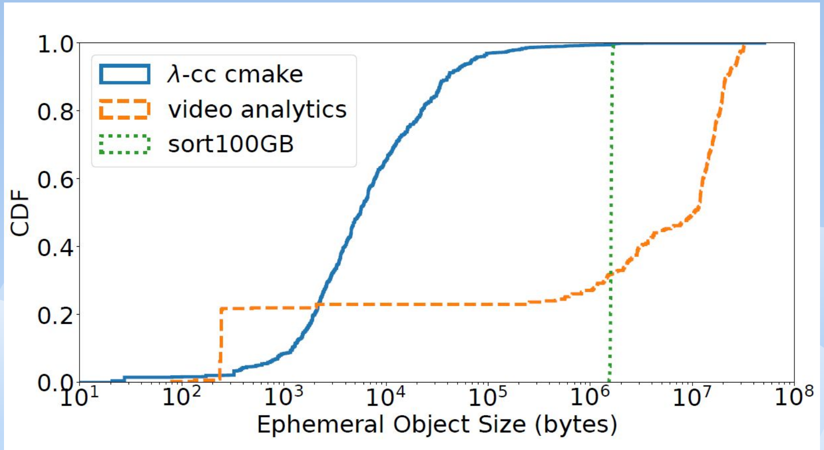
- ➔ Serverless computing has become more popular for data intensive applications, like interactive analytics.
- ➔ The availability of multiple storage medias increases the complexity of choosing a cluster configuration that balances performance and cost.

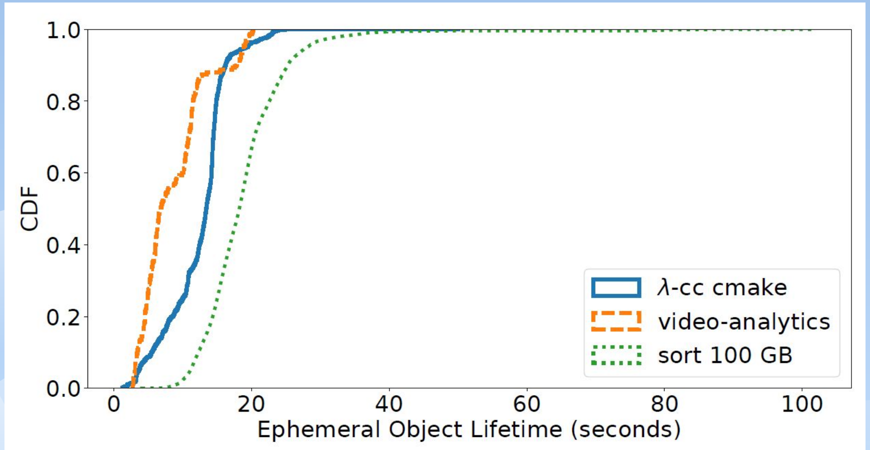
Why is this important?

- ➔ Pocket is the first platform targeting data sharing in serverless analytics.
- ➔ Pocket brings a fast and well-defined approach that automatically balances the performance and cost of the system.

Ephemeral Storage Requirements

- High performance for a wide range of object sizes
- Automatic and fine-grain scaling
- Storage technology awareness
- Fault-(in)tolerance





Existing Systems

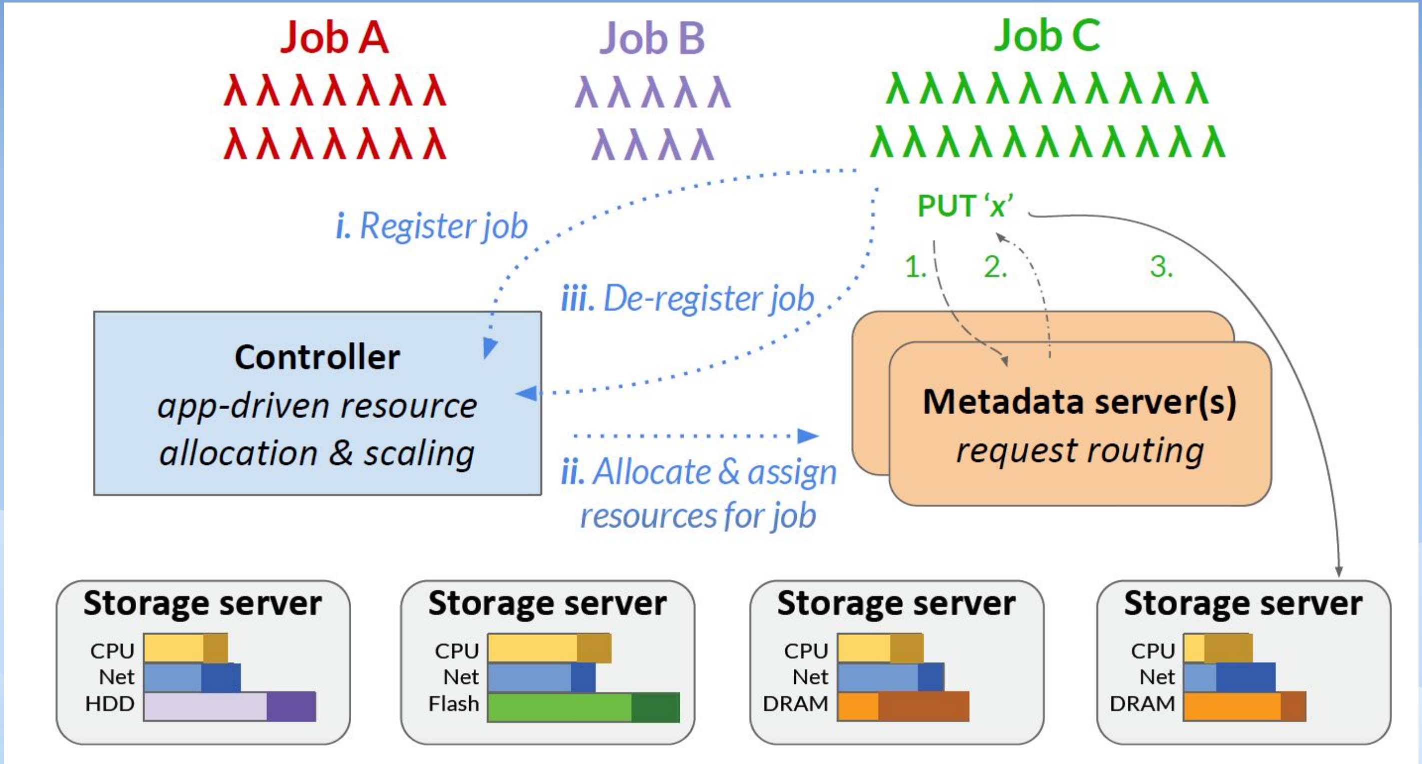
	Elastic scaling	Latency	Throughput	Max object size	Cost
S3	Auto, coarse-grain	High	Medium	5 TB	$
DynamoDB	Auto, fine-grain, pay per hour	Medium	Low	400 KB	$$
Elasticache Redis	Manual	Low	High	512 MB	$$$
Aerospike	Manual	Low	High	1 MB	$$
Apache Crail	Manual	Low	High	any size	$$
Desired for λs	*Auto, fine-grain, pay per second*	*Low*	*High*	*any size*	*$*

Pocket Design

➔ Follows 3 key design principles

1. Separation of responsibilities.
   a. The control plane
   b. The metadata plane
   c. The data plane
2. Sub-second response time.
3. Multi-tier storage.

Pocket Architecture



Application Interface

- Pocket object storage API:
  - Control functions:
    - *register_job:* Register job with controller and provide optional hints, returns a job ID and metadata server IP address.
    - *deregister_job:* Notify controller job has finished, delete job's non-PERSIST data.
  - Metadata functions:
    - *connect:* Open connection to metadata server.
    - *close:* Close connection to metadata server
  - Storage functions:
    - *put:* Write data, set PERSIST flag if want data to remain after job finishes.
    - *get:* Read data, set DELETE flag if data is only read once.

Life of a Pocket Application

➔ Steps that a Pocket app goes through:

1. Register the job in the controller.
2. Launch lambdas.
3. Establish connection with metadata serves.
4. Make *put* / *get* operations in the storage servers indicated by the metadata servers.
5. Deregister the job when the execution of the last lambda is complete.

Rightsizing Application Allocation

- Pocket estimates jobs latency, throughput, and capacity requirements, finding a cost-effective allocation of resources.
  - Determining job I/O requirements:
    - Pocket takes advantage of hints like:
      - Latency sensitivity.
      - Maximum number of lambdas.
      - Throughput.
      - Capacity.
  - Assigning resources:
    - Pocket allocates resources by generating a weight map for each job.

Rightsizing the Storage Cluster

- Pocket continuously monitors the cluster nodes and decides when and how to scale the storage and metadata servers.
  - Scaling mechanisms:
    - Horizontal: Launching new nodes.
    - Vertical: Making the most of the resources of the active nodes.
  - Cluster sizing policy:
    - Scale up: ***Any*** exceeds the upper threshold.
    - Scale down: ***All*** are below its lower threshold.
  - Balancing load with data steering:
    - The controller assigns higher weights in the incoming jobs to the nodes that are underutilized.

Evaluation - Methodology

Pocket Server	EC2 Server	DRAM (GB)	Storage (TB)	Network (Gb/s)	$/hr
Controller	m5.x1	16	0	~8	0.192
Metadata	m5.x1	16	0	~8	0.192
DRAM	r4.2x1	61	0	~8	0.532
NVMe-based Flash	i3.2x1	61	1.9	~8	0.624
SATA/SAS-based Flash or SSD	i2.2x1	61	1.6	2	1.705
HDD	h1.2x1	32	2	~8	0.468

Type and cost of EC2 VMs used for Pocket

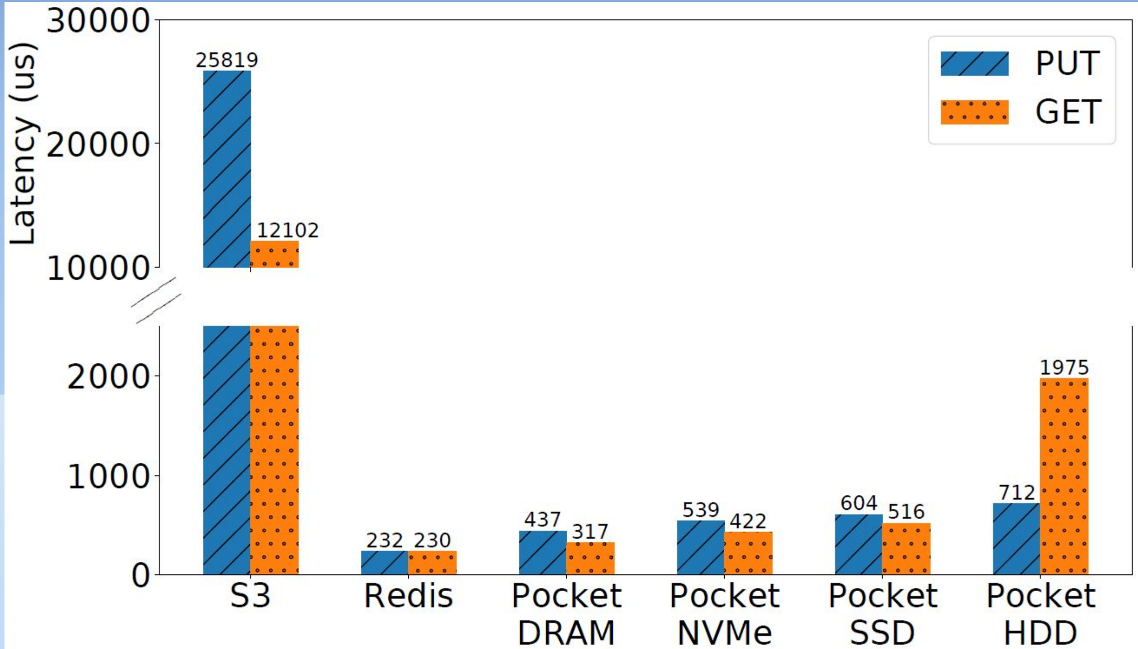
Evaluation - Methodology

➔ Three different serverless analytics applications:

- Video Analytics:
  - 25 Minute video with 40K 1080p frames;
  - 1º Stage: 160 lambdas;
  - 2º Stage: 305 lambdas;
- MapReduce Sort:
  - 100GB Sort, which generates 100GB ephemeral data;
  - Job runned with 250, 500 and 1000 lambdas;
- Distributed Software Compilation (λ-cc):
  - Maximum parallelism of 650 tasks;
  - Generates 850MB of ephemeral data.

Evaluation - Microbenchmarks

Storage Request Latency



Unloaded latency for 1KB access from lambda.

Evaluation - Microbenchmarks

Metadata Throughput

1 Core	4 Cores
90K Operations/second	175K Operations/seconds

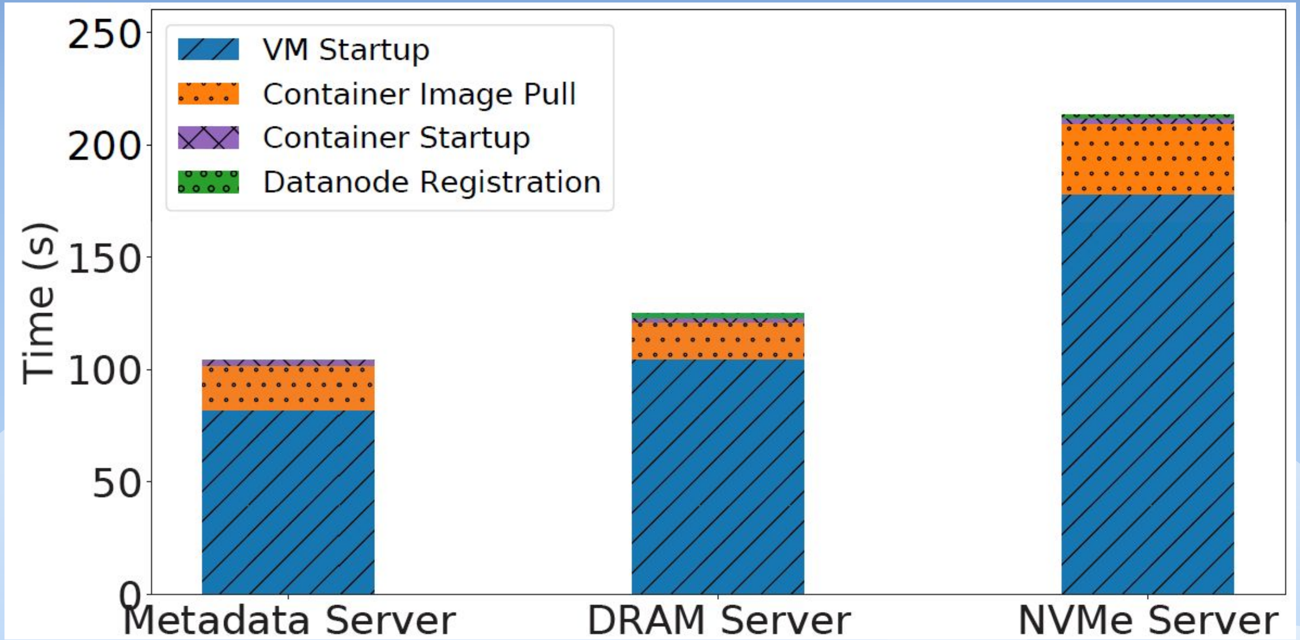
Nº of metadata operations that a metadata server can handle per second

75 Operations/second

Peak of metadata request rate observed for the serverless analytics applications per lambda

Evaluation - Microbenchmarks

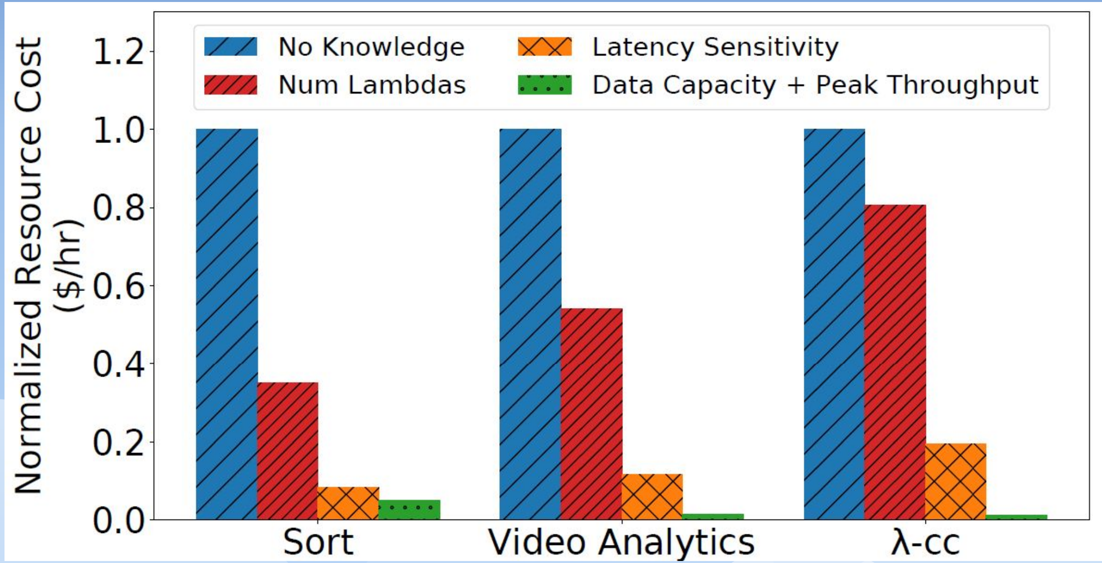
Adding or Removing Servers



Node Startup Breakdown.

Evaluation - Rightsizing Resource Allocations

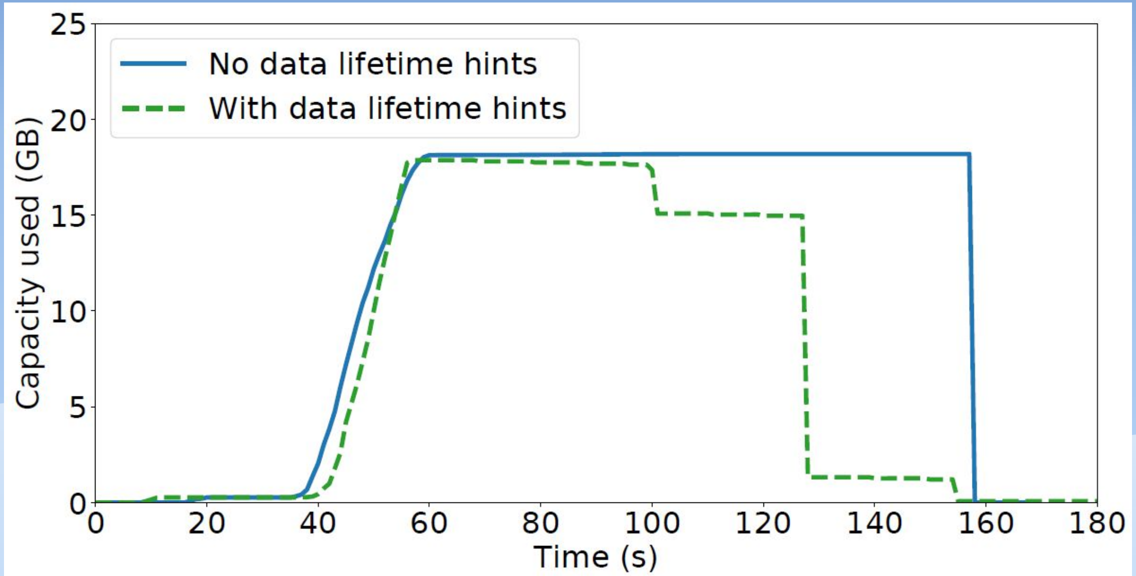
Rightsizing with application hints



Pocket leveraging cumulative hints about job characteristics to allocate resources cost-efficiently.

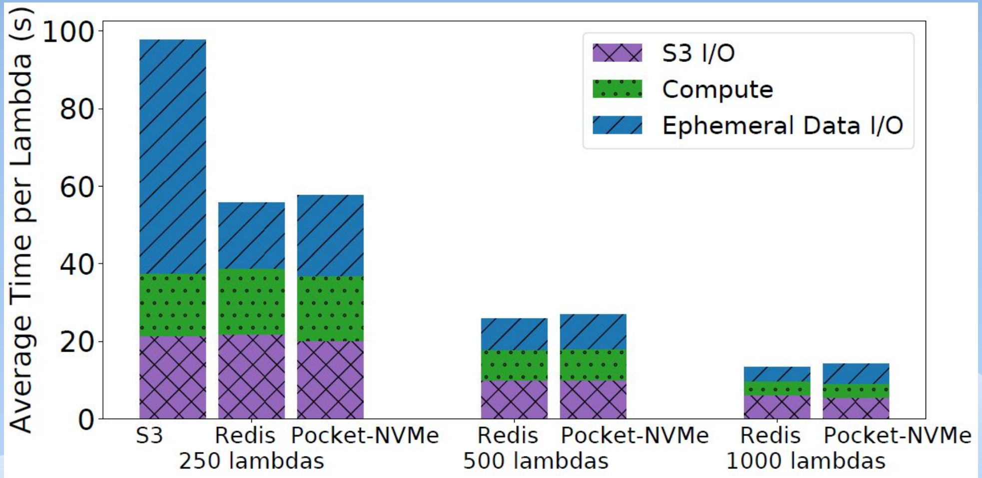
Evaluation - Rightsizing Resource Allocations

Reclaiming capacity using hints



Example of using the DELETE hint for get operations in a video analytics job.

Evaluation - Comparison with S3 and Redis



Execution time breakdown of 100GB sort

Job	S3	Redis	Pocket
100 GB sort	0.05126	5.320	2.1648
Video analytics	0.00034	1.596	0.6483
λ-cc	0.00005	1.596	0.6480

Hourly ephemeral storage cost (in USD)

Conclusion

- It was analysed challenges associated with efficient data;
- Presented Pocket, which provides high elastic, cost-effective and fine grained storage solution for analytics workloads;
- Evaluated Pocket, which showed to have:
  - High performance;
  - Automatic fine-grain scaling;
  - Cost-effective.