

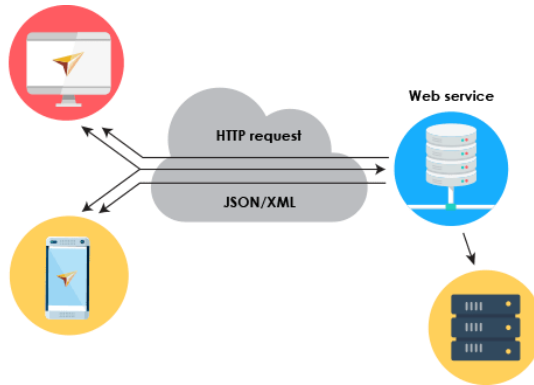
Pocket: Elastic Ephemeral Storage for Serverless Analytics

Ana Klimovic*, Yawen Wang*, Patrick Stuedi⁺,
Animesh Trivedi⁺, Jonas Pfefferle⁺, Christos Kozyrakis*

*Stanford University, ⁺IBM Research

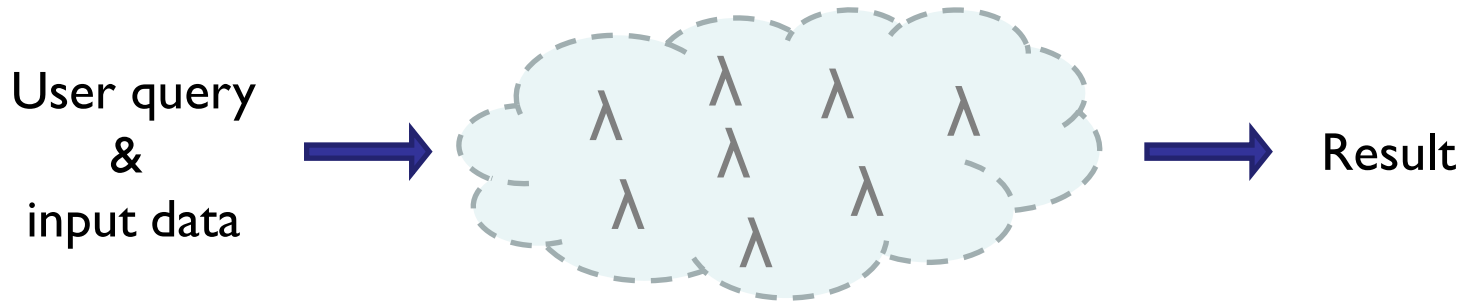
Serverless Computing

- ❑ Serverless computing enables users to launch short-lived tasks with *high elasticity* and *fine-grain resource billing*



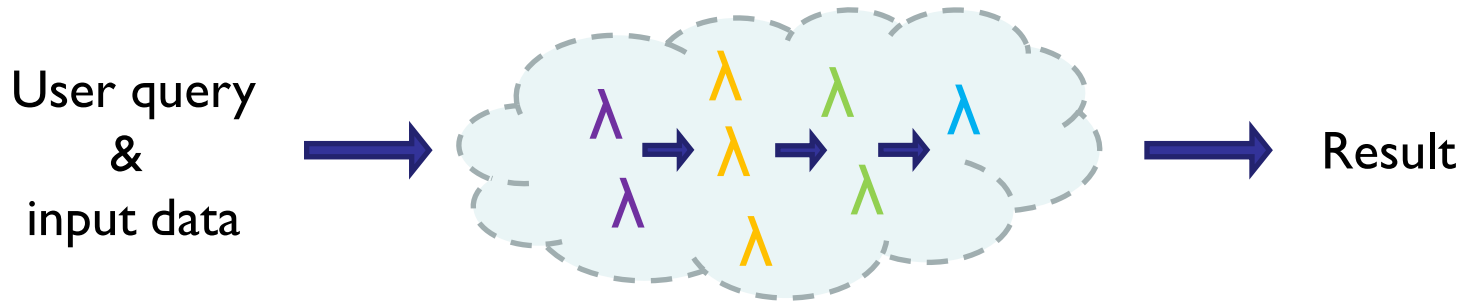
Serverless Computing

- ❑ Serverless computing enables users to launch short-lived tasks with *high elasticity* and *fine-grain resource billing*
- ❑ This also makes serverless appealing for *interactive analytics*



Serverless Computing

- ❑ Serverless computing enables users to launch short-lived tasks with *high elasticity* and *fine-grain resource billing*
- ❑ This also makes serverless appealing for *interactive analytics*



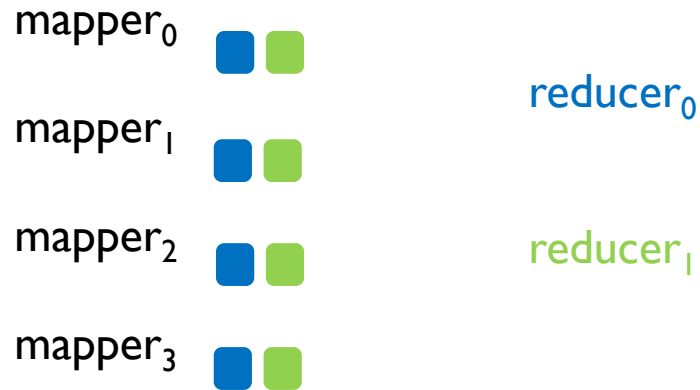
Serverless Computing

- ❑ Serverless computing enables users to launch short-lived tasks with *high elasticity* and *fine-grain resource billing*
- ❑ This also makes serverless appealing for *interactive analytics*
- ❑ **The challenge:** serverless tasks (*lambdas*) need an efficient way to communicate intermediate data between execution stages

ephemeral data

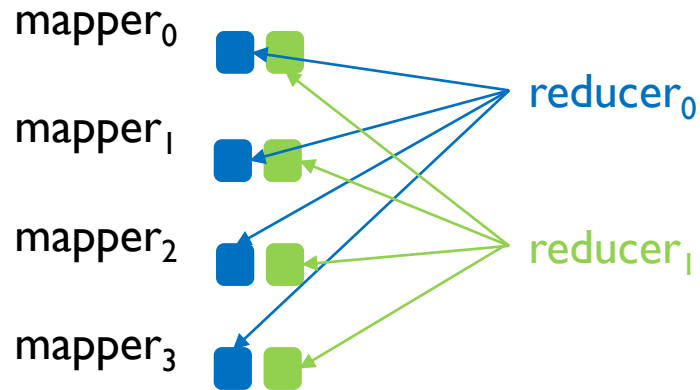
In traditional analytics...

- Ephemeral data is exchanged directly between tasks



In traditional analytics...

- Ephemeral data is exchanged directly between tasks



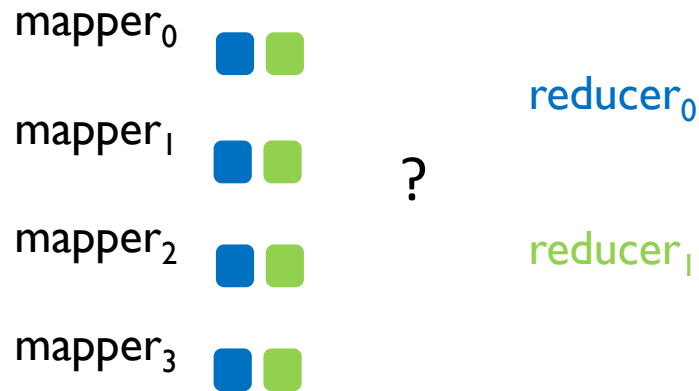
In serverless analytics...

- ❑ Direct communication between lambdas is difficult:
 - ❑ Lambdas are short-lived and stateless
 - ❑ Users have no control over lambda scheduling



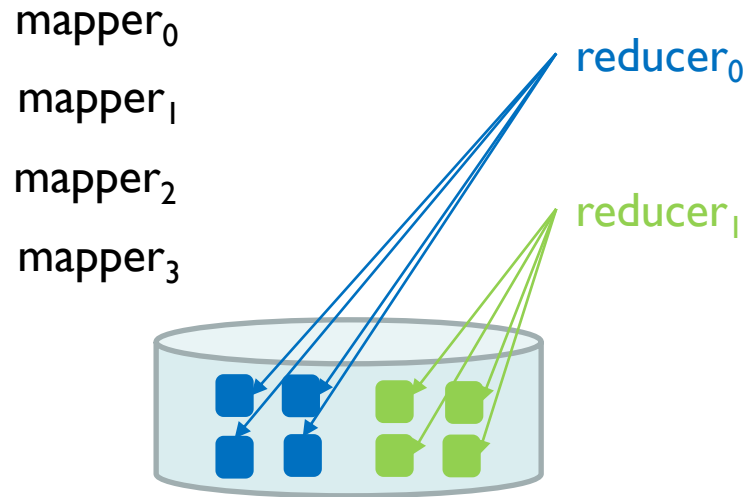
In serverless analytics...

- ❑ Direct communication between lambdas is difficult:
 - ❑ Lambdas are short-lived and stateless
 - ❑ Users have no control over lambda scheduling



In serverless analytics...

- The natural approach for sharing ephemeral data is through a *common data store*



In serverless analytics...

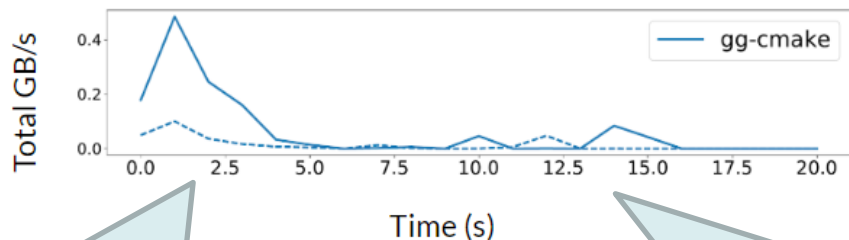
- ❑ The natural approach for sharing ephemeral data is through a *common data store*
- ❑ However, existing storage systems do not meet the elasticity, performance and cost demands of serverless analytics jobs

Requirements for Ephemeral Storage

Application Type

Distributed
Compilation

Ephemeral I/O Throughput:
Write (dotted), Read (solid)



Ephemeral Data Capacity

0.85 GB

High throughput and IOPS
due to high parallelism:
lambdas each compile
independent files

Final stage lambdas are serialized
as they depend on prior lambdas
→ low parallelism, low I/O rate

Requirements for Ephemeral Storage

Application Type

Ephemeral I/O Throughput:
Write (dotted), Read (solid)

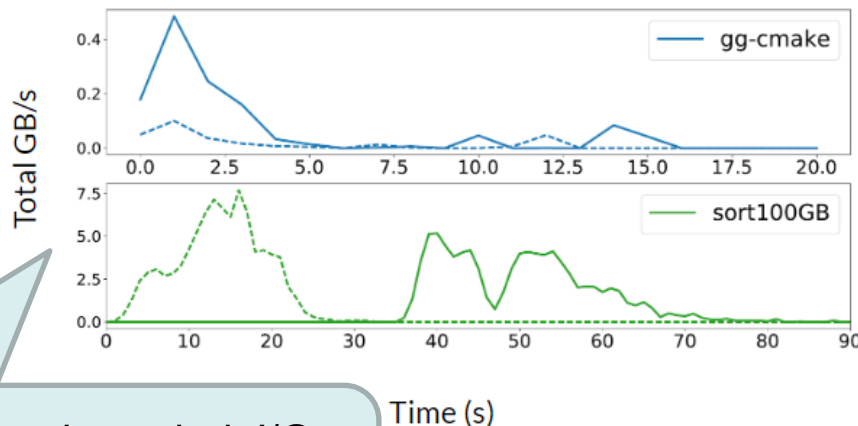
Ephemeral Data Capacity

Distributed
Compilation

0.85 GB

MapReduce

100 GB



High throughput due to high I/O intensity and parallelism (up to 7.5 GB/s with 500 lambdas)

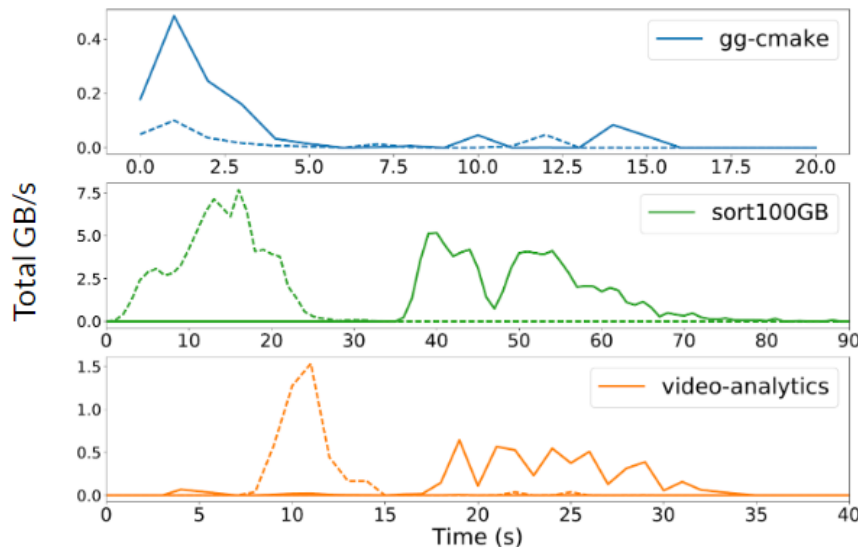
Requirements for Ephemeral Storage

Application Type

Ephemeral I/O Throughput: Write (dotted), Read (solid)

Ephemeral Data Capacity

Distributed
Compilation



0.85 GB

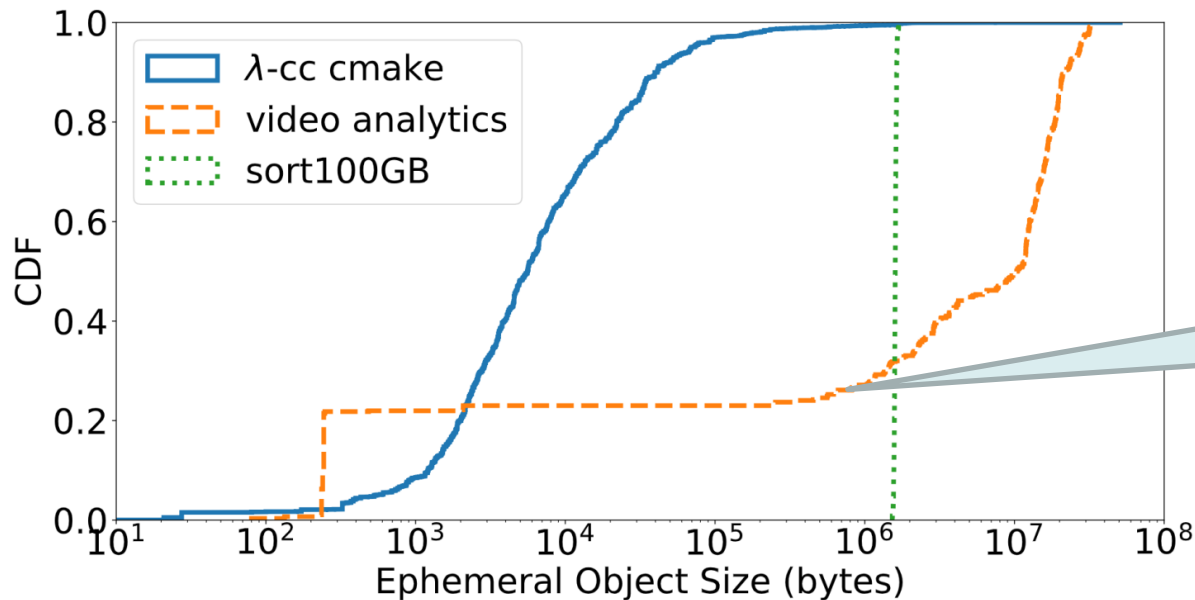
100 GB

Video Analytics

6 GB

Requirements for Ephemeral Storage

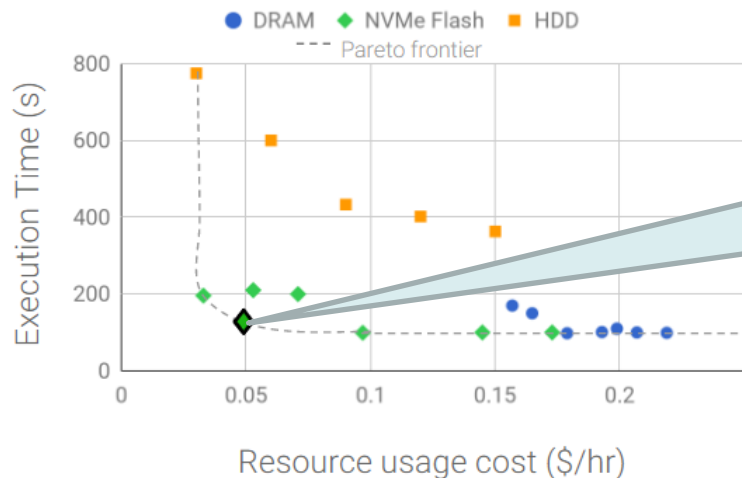
- Need high throughput (for large objects) **and** low latency (for small objects).



Object sizes vary from 100s of bytes to 100s of MBs

Requirements for Ephemeral Storage

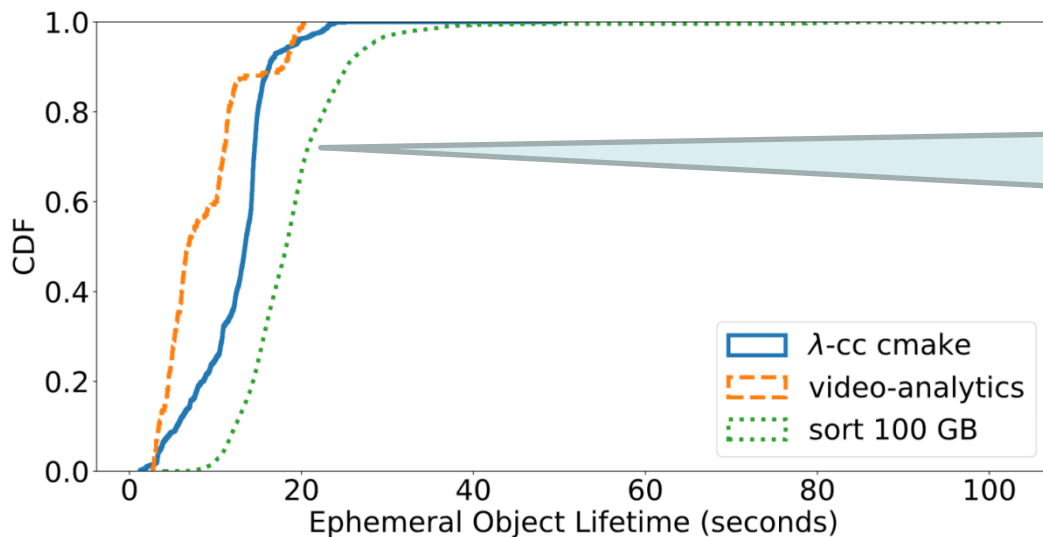
- ❑ Need automatic resource scaling and storage technology awareness
- ❑ Example of performance-cost tradeoff for a serverless video analytics jobs with different ephemeral data store configurations



Finding the Pareto optimal resource allocation is non-trivial...and gets harder with multiple jobs.

Requirements for Ephemeral Storage

- ❑ Do **not** need high fault tolerance, contrary to traditional storage systems
- ❑ Fault tolerance is typically baked into application frameworks



Ephemeral data has short lifetime; it is only valuable during job execution

Requirements for Ephemeral Storage

Summary:

1. High performance for a wide range of object sizes
2. Automatic resource scaling with storage technology awareness
3. Fault-(in)tolerance

Pocket



- ❑ An elastic, distributed data store for ephemeral data sharing in serverless analytics
- ❑ Key properties:
 - ❑ High throughput, low latency for a wide range of object sizes [Performance]
 - ❑ Automatic resource scaling and rightsizing [Cost, scalability]
 - ❑ Intelligent data placement across multiple storage tiers [Cost]
- ❑ Pocket achieves similar performance to Redis, an in-memory key value store, while saving ~60% in cost for various serverless analytics jobs

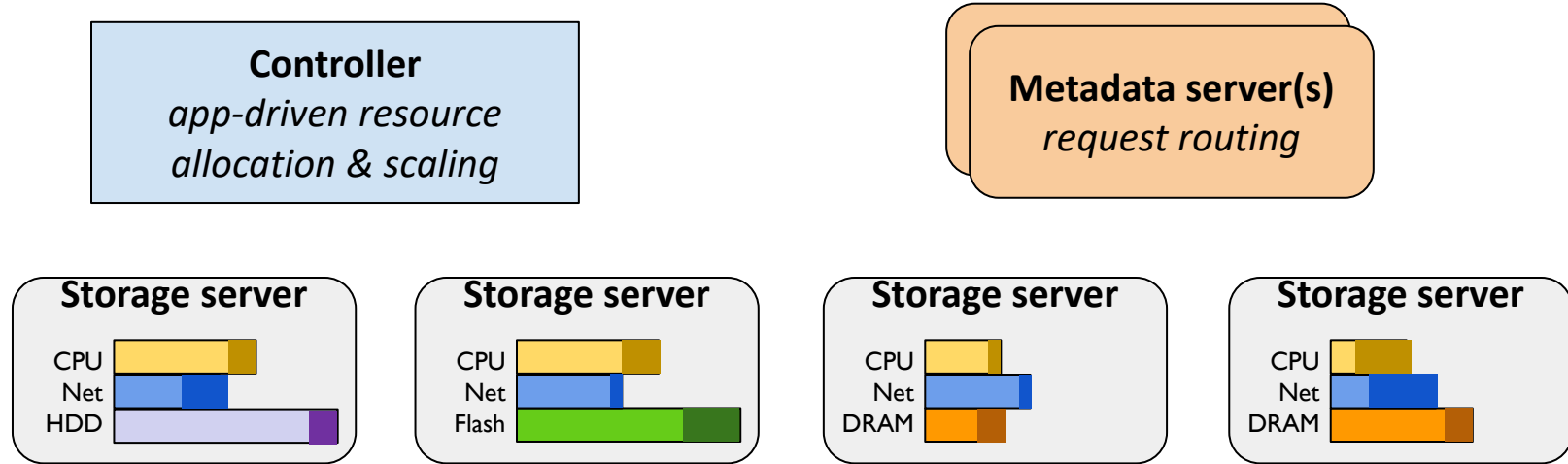
Pocket



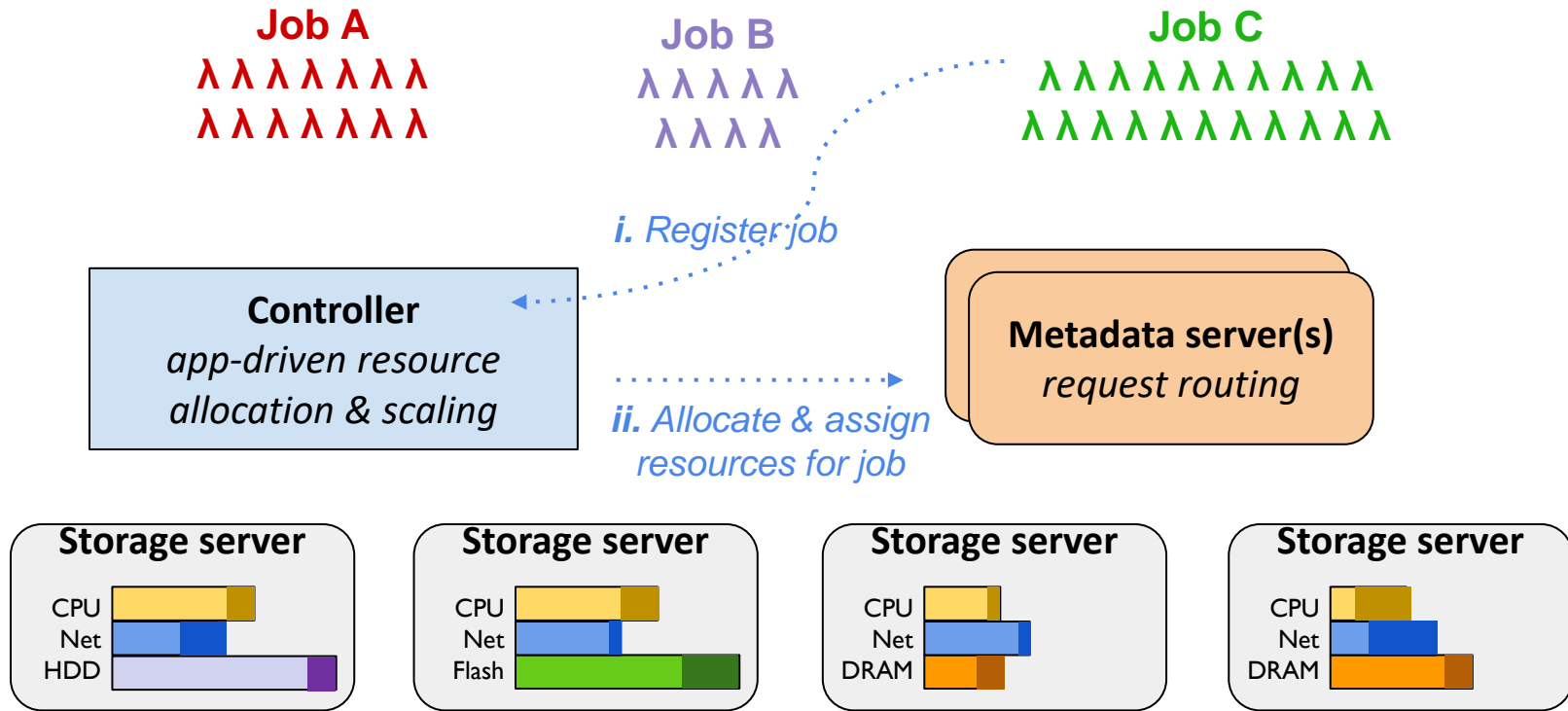
- Design principles:
 - **Separation of responsibilities:** control, metadata, and data plane can each be scaled independently
 - **Sub-second response time:** storage servers optimized for fast, simple I/O operations
 - **Multiple storage tiers:** use DRAM, Flash, and/or HDD to meet application I/O requirements at low cost



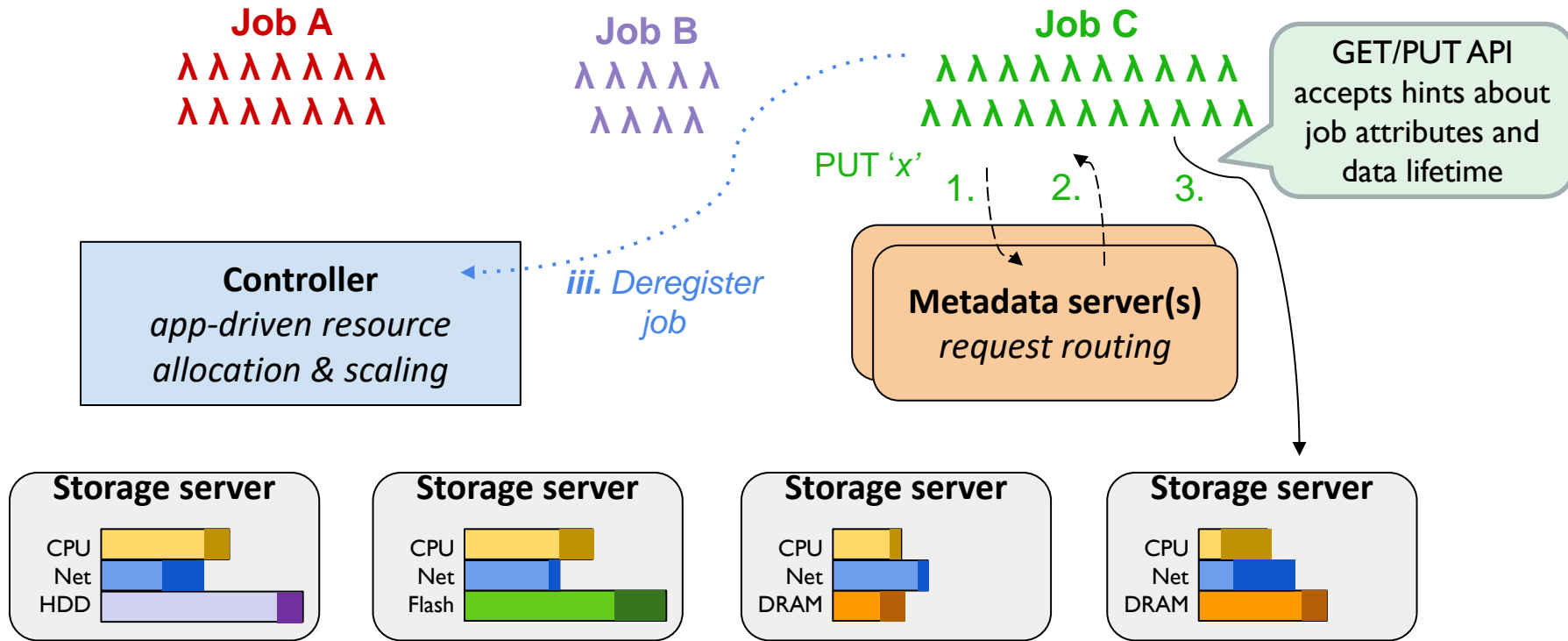
Pocket: System architecture



Pocket: System architecture



Pocket: System architecture



Resource allocation

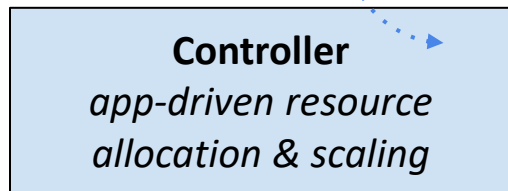
Job A
Λ Λ Λ Λ Λ Λ Λ
Λ Λ Λ Λ Λ Λ Λ

Optional hints about job attributes:

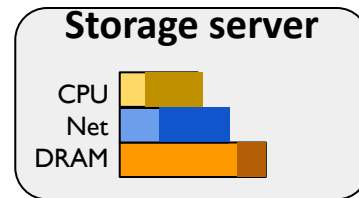
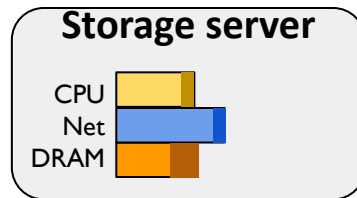
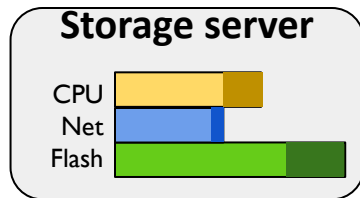
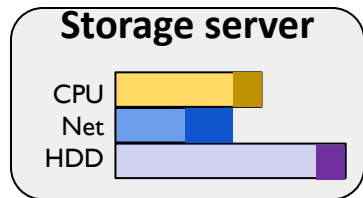
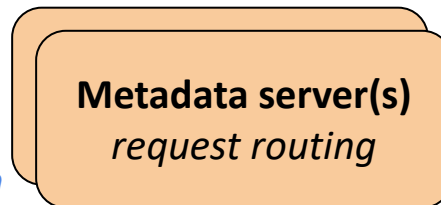
- Latency sensitivity
- Maximum # of concurrent lambdas
- Total ephemeral data capacity
- Peak aggregate bandwidth required

1. Throughput allocation
2. Capacity allocation
3. Choice of storage tier(s)

i. Register job



ii. Allocate & assign
resources for job



Resource assignment

1. Throughput allocation
2. Capacity allocation
3. Choice of storage tier(s)

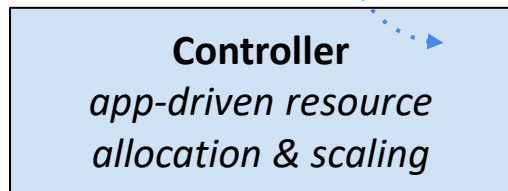


Job Weight Map

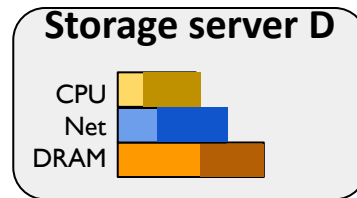
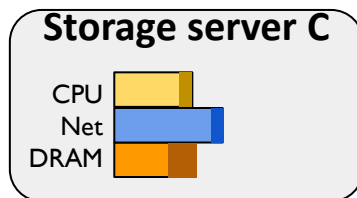
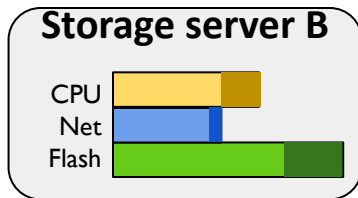
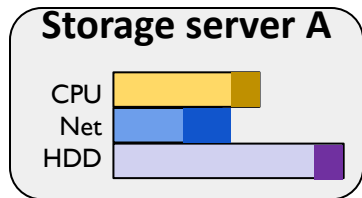
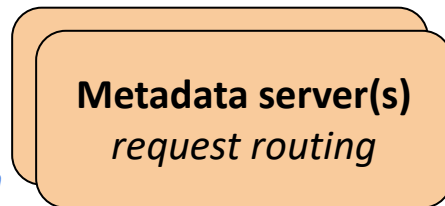
Job A:	
Server C	→ 0.4
Server D	→ 0.6
Job B:	
Server A	→ 0.2
Server B	→ 0.3
Server C	→ 0.5

Job A
Λ Λ Λ Λ Λ Λ Λ
Λ Λ Λ Λ Λ Λ Λ

i. Register job



ii. Allocate & assign resources for job

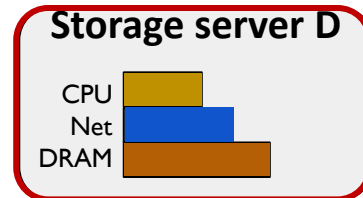
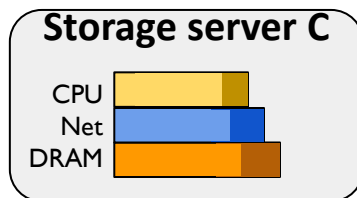
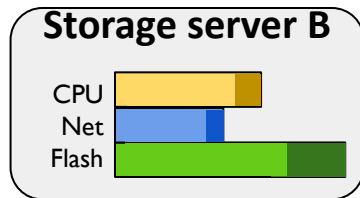
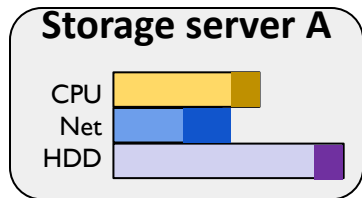


Elastic Rightsizing

- ❑ The controller continuously monitors cluster resource utilization
 - ❑ Nodes send CPU, network bandwidth, and storage capacity usage every second
- ❑ The controller scales resources dynamically as jobs register and deregister
 - ❑ **Policy:** keep CPU, network bandwidth and storage tier capacity utilization within a target range (e.g., 60-80%)
 - ❑ **Mechanism:** use weight map to balance load by *steering data for incoming jobs* onto active storage nodes and away from nodes that will be taken down

Elastic Rightsizing

- ❑ The controller continuously monitors cluster resource utilization
 - ❑ Nodes send CPU, network bandwidth, and storage capacity usage every second
- ❑ The controller scales resources dynamically as jobs register and deregister
 - ❑ **Policy:** keep CPU, network bandwidth and storage tier capacity utilization within a target range (e.g., 60-80%)
 - ❑ **Mechanism:** use weight map to balance load by *steering data for incoming jobs* onto active storage nodes and away from nodes that will be taken down

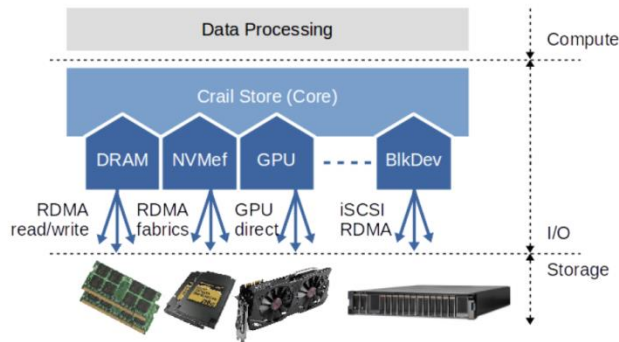


Implementation

- ❑ Pocket's storage and metadata server implementation is based on the **Apache Crail** distributed storage system
- ❑ We use **ReFlex** for the Flash storage tier
- ❑ Pocket runs the storage and metadata servers in containers, orchestrated using **Kubernetes**

Apache crail

- ❑ High-performance distributed data store designed for ephemeral data sharing in distributed data processing frameworks (e.g., Spark)
- ❑ Originally designed to leverage high-performance RDMA networks
- ❑ Pluggable storage tiers and network processing layers



ReFlex

- ❑ Software for fast access to NVMe Flash over commodity networks

1. Low latency, high throughput with low compute overhead:

- ❑ Direct access to NIC and NVMe queues from userspace
- ❑ Polling-based, run to completion execution model
- ❑ Minimal data copying; forward data directly between NIC and Flash
- ❑ Adaptive batching

2. Predictable performance on shared Flash with QoS-aware I/O scheduler

- ❑ Enforce throughput and tail latency SLOs for tenants sharing Flash
- ❑ Provide isolation to mitigate read/write request interference

www.github.com/stanford-mast/reflex

Pocket deployment

- We deploy Pocket on Amazon Web Services (AWS) EC2

Pocket Controller / Metadata server	m5.xlarge
DRAM server	r4.2xlarge
NVMe Flash server	i3.2xlarge

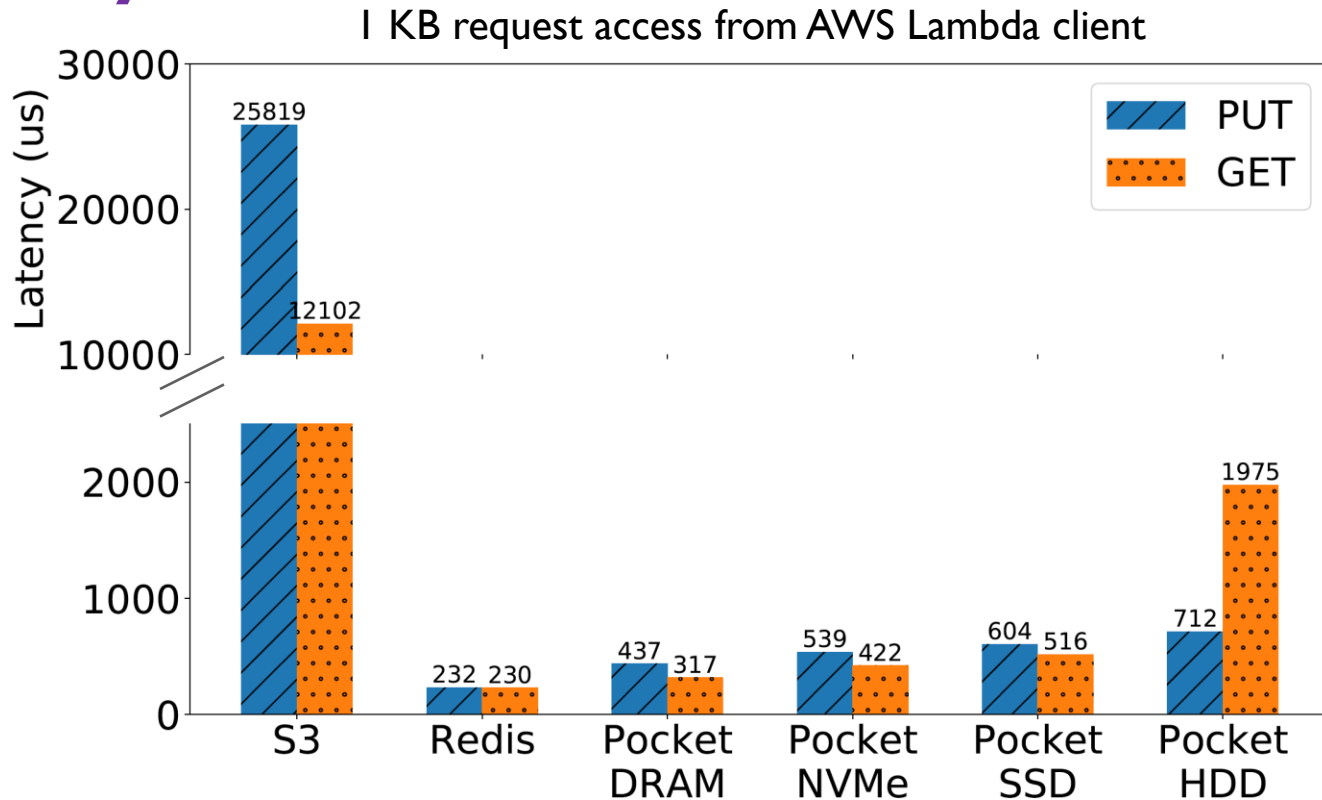


Amazon EC2



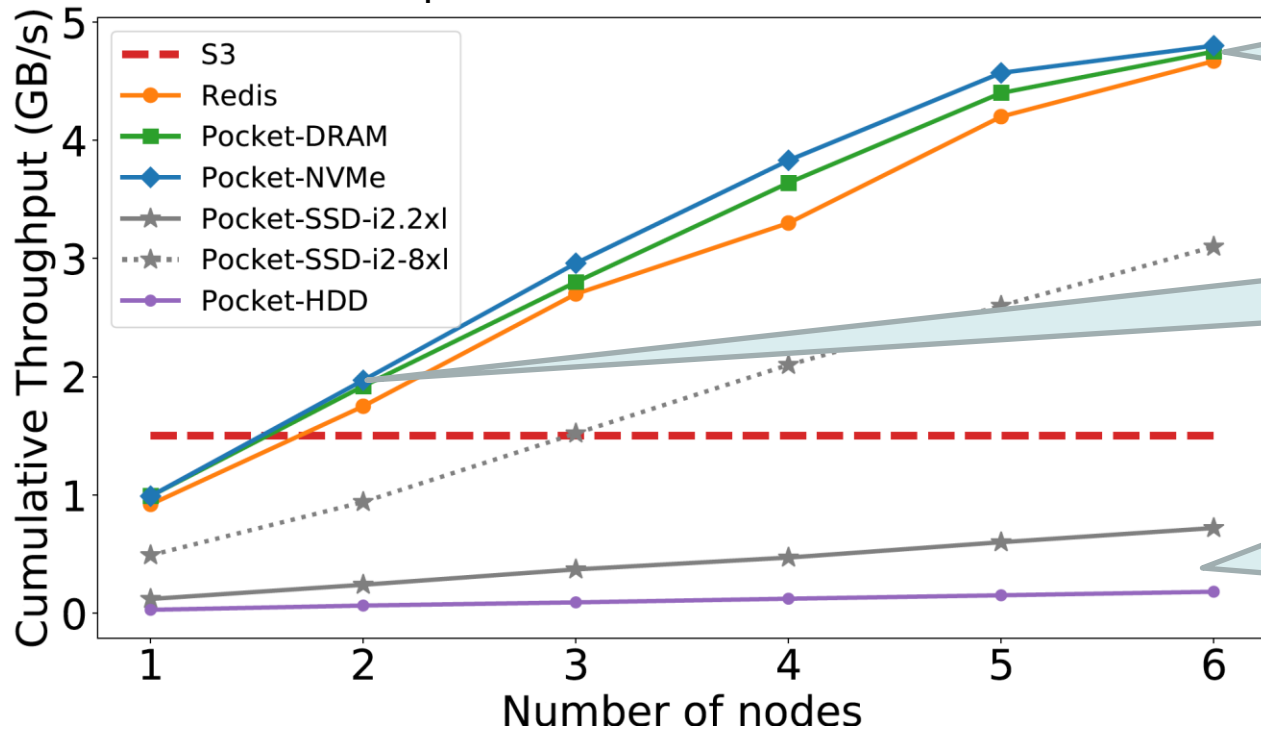
- We use AWS Lambda as our serverless platform

Latency



Throughput scaling

1 MB requests from 100 concurrent lambdas

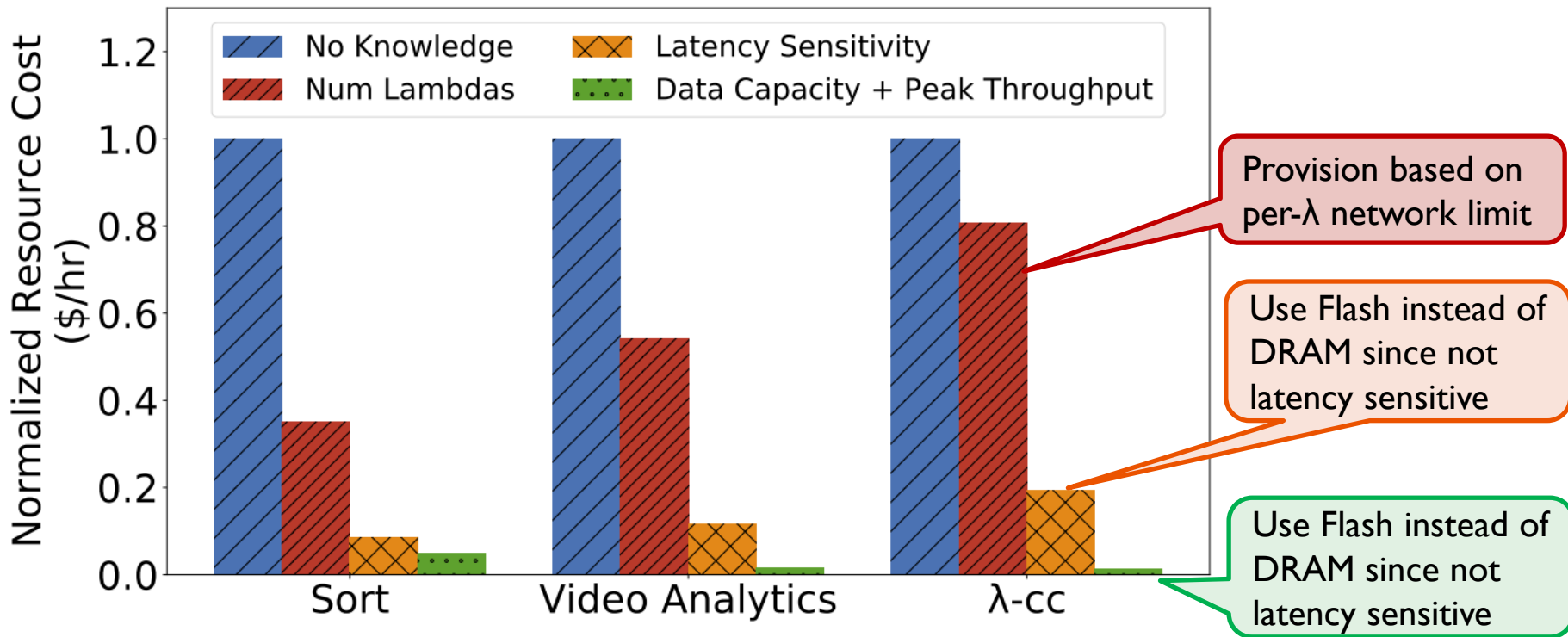


Reach AWS Lambda per- λ network limit

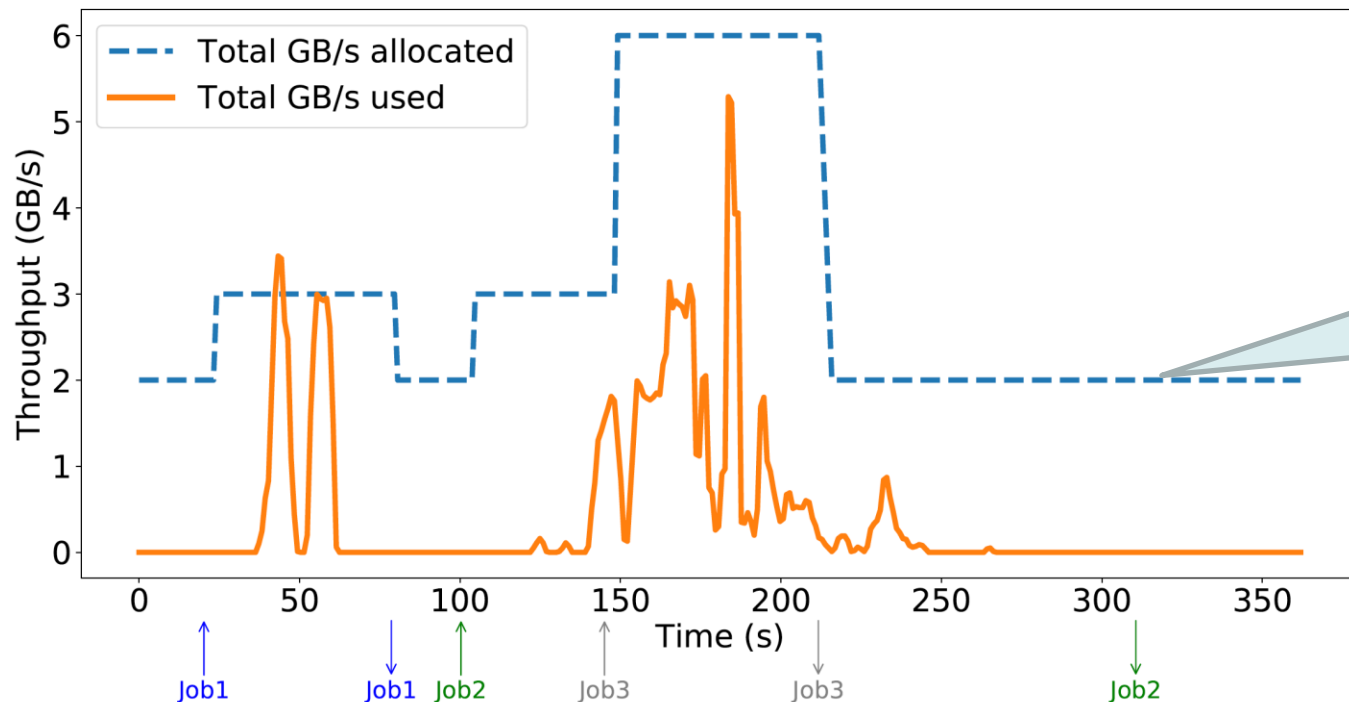
With 2 nodes, Pocket-NVMe and Pocket-DRAM offer higher throughput than S3

SATA/SAS-based SSD and HDD tiers offer significantly lower throughput

Rightsizing with hints

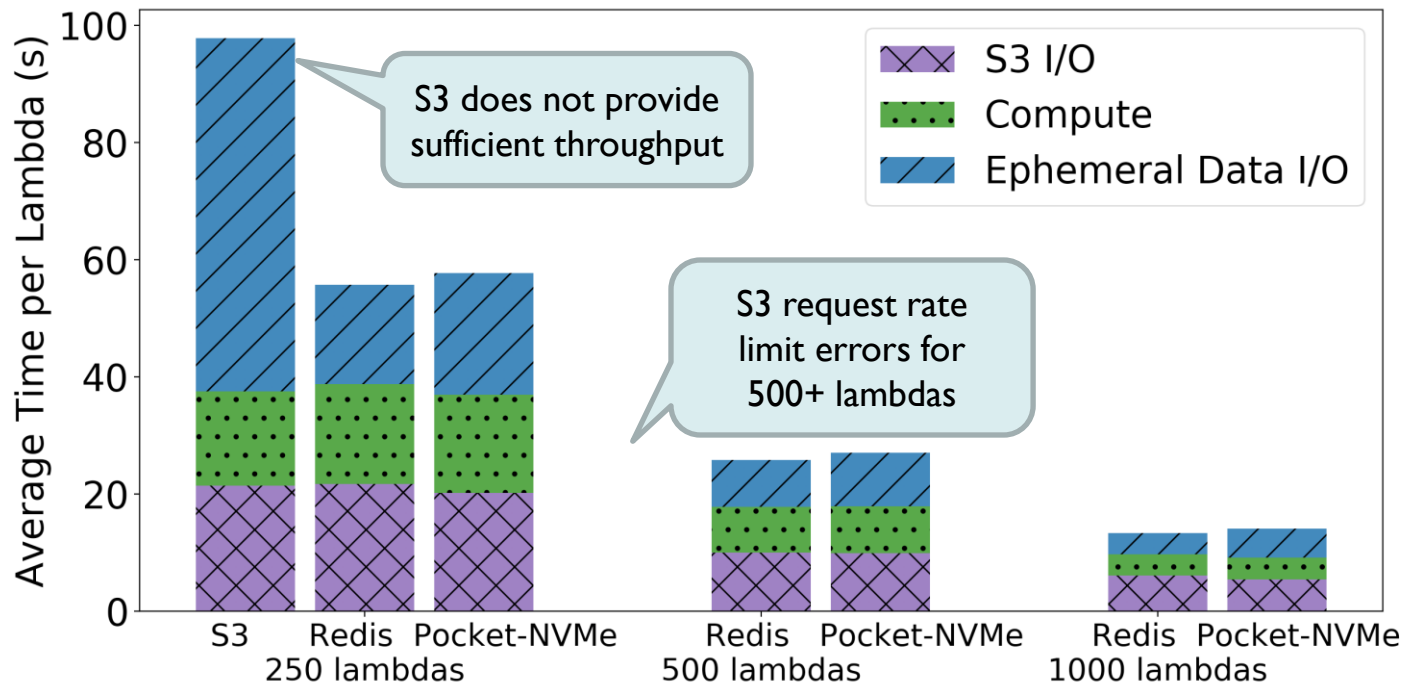


Rightsizing with multiple jobs

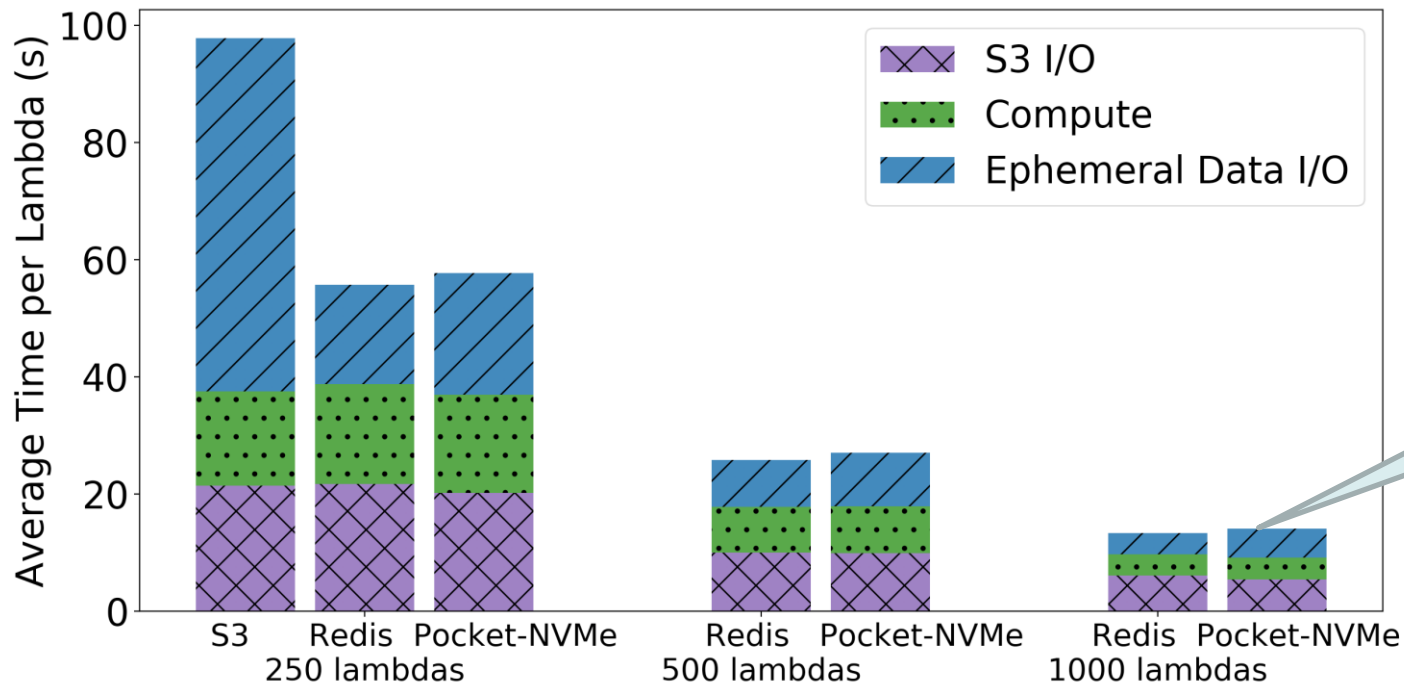


The controller elastically scales resources to meet the requirements of multiple jobs

Execution time for 100 GB sort job



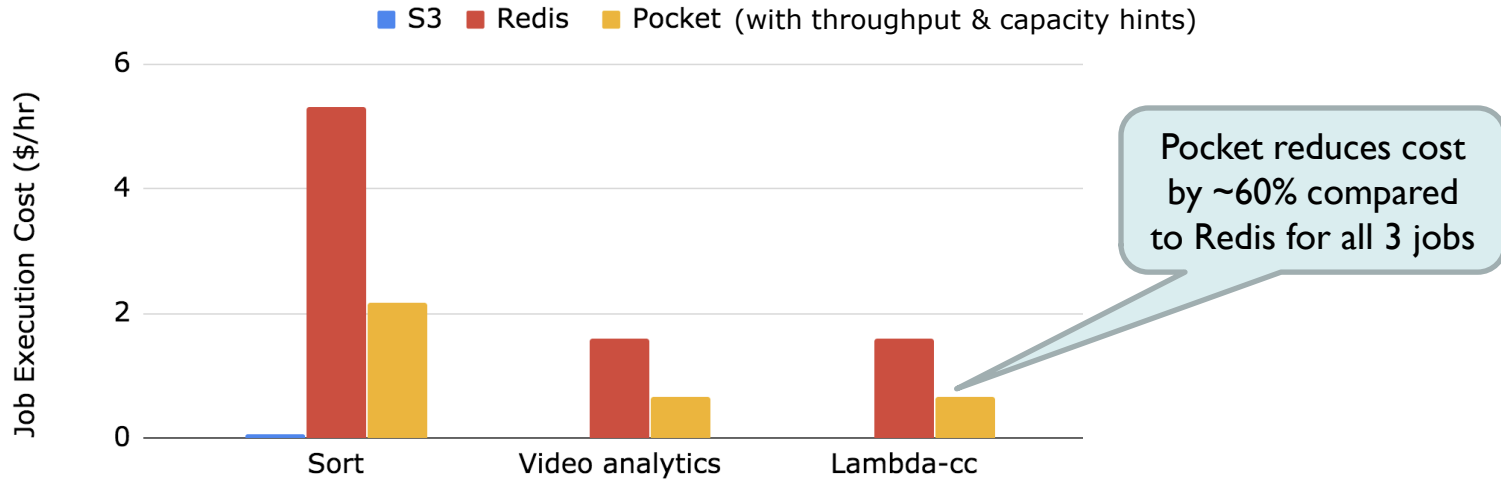
Execution time for 100 GB sort job



Pocket-NVMe achieves similar performance to Redis

Cost analysis

- ❑ Pocket leverages job attribute hints for cost-effective resource allocation and amortizes VM costs across multiple jobs, offering a pay-what-you-use model
- ❑ S3 is much cheaper but the cost comparison is not fair as S3 pricing is based on cloud *provider* resource costs vs. cloud *customer* resource costs



Future work

- ❑ Autonomously learn application characteristics across jobs
- ❑ Use slack resources in the datacenter to run ephemeral storage nodes
- ❑ Explore other use cases for distributed ephemeral storage, beyond serverless computing

Conclusion

- ❑ Pocket is a distributed ephemeral storage system providing:
 - ❑ Low latency, high throughput
 - ❑ Automatic resource scaling
 - ❑ Intelligent data placement across nodes
- ❑ We designed Pocket for ephemeral data sharing in serverless analytics. However, Pocket can be used more generally for applications requiring an elastic, distributed / tmp.

www.github.com/stanford-mast/pocket