Fast, Elastic Storage for the Cloud

Ana Klimovic

Stanford University

Platform Lab Seminar on Feb. 5, 2019
Cloud computing offers high…

Elasticity
Cost-efficiency
Scalability
Performance

…only if each application receives the storage and compute resources it needs.

Allocating resources to achieve these goals is hard.
Why is resource allocation difficult?

Consider the resource utilization of a large-scale service at Facebook, normalized over 6 months: [EuroSys’16]

Resource requirements vary across applications.

Compute and storage requirements vary dynamically over time.
Why is resource allocation difficult?

Consider the resource utilization of a large-scale service at Facebook, normalized over 6 months: [EuroSys’16]

Compute and storage requirements are often uncorrelated.

But servers in a cloud facility have fixed ratios of compute and storage. → Lack of flexibility leads to imbalanced resource usage

Storage is underutilized.
At datacenter scale...

~10,000s of servers
~10s of Megawatts

...underutilizing resources is extremely wasteful.

Huge opportunity to improve resource efficiency!

**Challenge:** maintain high performance
How should we build resource-efficient and high performance storage systems for diverse, elastic applications in the cloud?
Key insights for cloud storage

Decouple storage from compute

Leverage hints and/or machine learning to dynamically allocate storage resources for elastic applications
Decouple storage from compute

Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.
Decouple storage from compute

Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.
Decouple storage from compute

Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.

**Goal**: enable any CPU to use any storage device in a cloud facility that has spare capacity & bandwidth.
Share storage to improve utilization

Share storage devices among applications to increase utilization and improve cost-efficiency.

**Goal**: enable any CPU to use any storage device in a cloud facility that has spare capacity & bandwidth.
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need fast, predictable access to remote data.
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.

*Local access to storage*
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need fast, predictable access to remote data.
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.

Networking **hardware** is fast → up to 100 Gb/s

Remote access to storage

Traditional **software** for processing network storage requests is slow
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need fast, predictable access to remote data.

Remote access to storage

Request interference can cause unpredictable performance
Requirement #2: Automatic, elastic allocation of storage resources

- We need policies to decide which storage resources to allocate to each application to satisfy latency, capacity, throughput requirements

- Need to automatically adjust allocations as application load varies
Requirement #2: Automatic, elastic allocation of storage resources

- We need policies to decide which storage resources to allocate to each application to satisfy latency, capacity, throughput requirements.

- Need to automatically adjust allocations as application load varies.

Leverage hints & machine learning to dynamically allocate storage resources for elastic applications.
Two major requirements for cloud storage

1. Fast access to remote data

2. Automatic, elastic allocation of storage resources
Contributions

1. Fast access to remote data

2. Automatic, elastic allocation of storage resources

[OSDI’14] IX: Dataplane OS for Fast Networking

[EuroSys’16] Flash Storage Disaggregation

[HotStorage’17] Rack-Scale Disaggregated Storage

[ASPLOS’17] ReFlex: Remote Flash $\approx$ Local Flash

[ATC’18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI’18] Pocket: Elastic Ephemeral Storage for Serverless Analytics

[ATC’18b] Selecta: ML-based Heterogeneous Cloud Storage Configuration
Focus of this talk

1. Fast access to remote data

2. Automatic, elastic allocation of storage resources

[OSDI’14] IX: Dataplane OS for Fast Networking

[EuroSys’16] Flash Storage Disaggregation

[HotStorage’17] Rack-Scale Disaggregated Storage

[ASPLOS’17] ReFlex: Remote Flash \(\approx\) Local Flash

[ATC’18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI’18] Pocket: Elastic Ephemeral Storage for Serverless Analytics

[ATC’18b] Selecta: ML-based Heterogeneous Cloud Storage Configuration
Fast access to remote flash
Flash storage

NVMe Flash:
- 70 µs read latency → 100x faster than disk
- 1,000,000 IOPS → 1000x higher than disk

What if we use existing remote storage approaches to access remote flash?
Issue #1: Existing software is too slow

Existing remote storage solutions have significant overheads.

Software is the bottleneck.

5.5x throughput drop

2x latency increase
Issue #2: Unpredictable performance on shared flash

Write requests from one tenant can **interfere** with read requests from another tenant, leading to unpredictable performance on shared flash.

![Graph showing p95 read latency vs Total IOPS](image)

Write requests increase the tail latency of read requests.
ReFlex: Remote Flash ≈ Local Flash

A software system for fast, predictable access to remote flash storage
ReFlex: Remote Flash ≈ Local Flash

A software system for fast, predictable access to remote flash storage

**Issue #1:** Existing software is too slow

→ Custom network-storage OS, designed for low latency and high throughput

✓ Run to completion
✓ Adaptive batching
✓ Reduce data copies
✓ Direct access to hardware
How does ReFlex achieve high performance?

Linux vs. ReFlex

**Software**
- Network queues
- Flash storage queues

**Hardware**
- Network queues
- Flash storage queues
How does ReFlex achieve high performance?

Linux vs. ReFlex

**Linux**
- **Software**
  - Interrupt driven → context-switch
- **Hardware**
  - Network queues
  - Flash storage queues

**ReFlex**
- **Software**
- **Hardware**
  - Network queues
  - Flash storage queues
  - **Run to completion**

**Improves data cache locality & Removes scheduling unpredictability**
How does ReFlex achieve high performance?

Linux vs. ReFlex

Software
- Interrupt driven → context-switch
- Adaptive batching

Hardware
- Network queues
- Flash storage queues

Improves instruction cache locality
How does ReFlex achieve high performance?

Linux vs. ReFlex

- **Linux**
  - Software: Multiple data copies
  - Hardware:
    - Network queues
    - Flash storage queues

- **ReFlex**
  - Software: Forward request payload directly
  - Hardware:
    - Network queues
    - Flash storage queues

Avoids copying data in software
How does ReFlex achieve high performance?

**Linux** vs. **ReFlex**

- **Linux**
  - Many layers of generalized abstractions
  - Network queues
  - Flash storage queues

- **ReFlex**
  - Direct access to hardware queues
  - Flexibility to implement a custom network-storage OS
ReFlex performance: throughput per core

4KB requests

p95 Read Latency (μs)

0 200 400 600 800 1000

I/O operations per second

0 250K 500K 750K 1000K 1250K

Linux: 75K IOPS/core

ReFlex: 850K IOPS/core

Local

ReFlex remote

Linux remote
ReFlex performance: throughput per core

The diagram shows the comparison of ReFlex performance across different local and remote configurations. The y-axis represents the p95 Read Latency (us) and the x-axis represents the I/O operations per second.

- **Local** performance compared to **Linux** shows a 11x improvement.

The graph displays four lines:
- **Green** dashed line: Local
- **Blue** solid line: ReFlex remote
- **Orange** solid line: Linux remote

For the Local configuration, the p95 Read Latency remains relatively constant as the number of I/O operations increases. In contrast, the Linux remote configuration shows a significant increase in latency as the number of I/O operations rises, indicating a 11x improvement with ReFlex.
ReFlex performance: latency

- Local Flash: 78 µs
- ReFlex: 99 µs
- Linux: 200 µs
ReFlex: Remote Flash ≈ Local Flash

A software system for fast, predictable access to remote flash storage

**Issue #1:** Existing software is too slow
→ Custom network-storage OS, designed for low-latency and high throughput

**Issue #2:** Unpredictable performance on shared flash storage
→ Novel I/O scheduler provides tail latency and throughput guarantees
How does ReFlex provide predictable performance?

ReFlex schedules requests to satisfy the performance objectives specified by each tenant.

**Goal**: Provide tail latency and throughput guarantees to tenants sharing flash.

**Challenge**: Account for the performance properties of different request types (e.g., read vs. write, 4 KB vs. 1 MB request)

**Solution**: profile device to derive a request cost model that captures how each type of request impacts tail latency and throughput
How does ReFlex provide predictable performance?

Step 1: Build a request cost model
→ account for different request types

For this device,
write cost = 10 x read cost

Weighted IOPS = (1 x Read IOPS) + (10 x Write IOPS)
How does ReFlex provide predictable performance?

Step 1: Build a request cost model  
→ account for different request types

Step 2: Schedule requests

Example scenario

**Tenant A:**
- 1ms tail latency
- 200K IOPS

**Tenant B:**
- Best-effort (use slack)

*For this device, write cost = 10 x read cost*

\[ \text{Weighted IOPS} = \text{Read IOPS} + (10 \times \text{Write IOPS}) \]
How does ReFlex provide predictable performance?

Step 1: Build a request cost model
   → account for different request types

Step 2: Schedule requests

Example scenario

**Tenant A:**
- 1ms tail latency
- 200K IOPS

**Tenant B:**
- Best-effort (use slack)

For this device, write cost = 10 x read cost

Weighted IOPS = Read IOPS + (10 x Write IOPS)

1ms tail latency SLO

200K IOPS

Device max IOPS: 510K

Slack: 310K
ReFlex performance guarantees on shared Flash

Tenants A & B: latency-critical; Tenant C + D: best effort
ReFlex performance guarantees on shared Flash

Tenants A & B: latency-critical; Tenant C + D: best effort
ReFlex performance guarantees on shared Flash

Tail latency

I/O sched disabled

I/O sched enabled

Tenant A

Tenant B

Tenant C

Tenant D

Throughput

Tenant A objective

Tenant B objective

Tenants A & B: latency-critical; Tenant C + D: best effort
ReFlex impact

- ReFlex provides high throughput, low latency with the flexibility to use commodity networks
- Broadcom is porting ReFlex to a SoC platform
- ReFlex integrated into Apache Crail storage system → runs on

https://github.com/stanford-mast/reflex
Contributions

1. Fast access to remote data

2. Automatic, elastic allocation of storage resources

[OSDI’14] IX: dataplane OS for fast networking
[EuroSys’16] Flash storage disaggregation
[HotStorage’17] Rack-Scale Disaggregated Storage
[ASPLOS’17] ReFlex: Remote Flash ≈ Local Flash

[ATC’18a] Understanding Ephemeral Storage for Serverless Analytics
[OSDI’18] Pocket: Elastic Ephemeral Storage for Serverless Analytics
[ATC’18b] Selecta: ML-based Heterogeneous Cloud Storage Configuration
Automatic, elastic allocation of storage resources
Serverless computing

Serverless computing is a new cloud service → users can launch thousands of tiny tasks with high elasticity and fine-grain billing.

Users focus on writing code for their applications → no resource management.

Cloud providers automatically allocate and scale resources.
Serverless analytics

Serverless computing is increasingly used for interactive analytics to exploit massive task parallelism to get a result in near real-time.
Data sharing in serverless analytics

- Analytics jobs involve multiple stages of execution

**Challenge:** exchange intermediate data efficiently between tasks

*ephemeral data*
Data sharing in serverless analytics

- Direct communication between serverless tasks is difficult:
  - Tasks are short-lived and stateless

![Diagram showing mapper and reducer tasks](attachment:image.png)
Data sharing in serverless analytics

- The natural approach for sharing ephemeral data is through a shared remote data store

mapper_0
mapper_1
mapper_2
mapper_3

reducer_0
reducer_1
Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency
Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency
   - Example of performance-cost tradeoff for a serverless video analytics job with different ephemeral data store configurations

Finding Pareto optimal resource allocations is non-trivial...and gets harder with multiple jobs.
Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency
3. Fault-tolerance

Today’s cloud storage systems are not optimized for the requirements of serverless analytics jobs.

* e.g., - S3 has low cost, but is optimized for large objects
  - Redis offers high performance, but uses DRAM → expensive
Pocket

- A fast and elastic storage service for ephemeral data sharing in serverless analytics

- Pocket achieves high performance and cost efficiency by:
  - Leveraging multiple storage technologies
  - Rightsizing resource allocations for applications
  - Autoscaling storage resources in the cluster based on usage

- Similar performance to Redis, an in-memory data store, while saving ~60% in cost for various serverless analytics jobs
Pocket design

Job A
λ λ λ λ λ λ λ
λ λ λ λ λ λ λ
λ λ λ λ λ λ λ

Job B
λ λ λ λ λ
λ λ λ λ

Job C
λ λ λ λ λ λ λ λ λ λ λ λ λ
λ λ λ λ λ λ λ λ λ λ λ λ λ

Controller
app-driven resource allocation & scaling

Metadata server(s)
request routing

Storage server

Storage server

Storage server

Storage server
1. Leverage multiple storage technologies

- **Controller**
  - *app-driven resource allocation & scaling*

- **Job A**
  - Disk
  - Flash
  - DRAM

- **Job B**
  - Disk
  - Flash

- **Job C**
  - Disk
  - Flash

- **Metadata server(s)**
  - *request routing*

- **ReFlex**
2. Allocate resources based on job requirements

Optional hints about job:
- Latency sensitivity
- Maximum # of concurrent tasks
- Total ephemeral data capacity
- Peak aggregate bandwidth required

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

Controller
app-driven resource allocation & scaling

Storage server
Disk

Storage server
Flash

Storage server
Flash

Storage server
DRAM

Job A
λ λ λ λ λ λ λ
λ λ λ λ λ λ λ
λ λ λ λ λ λ λ

i. Register job

Metadata server(s)
request routing
3. Autoscale resources based on utilization

Controller
app-driven resource allocation & scaling

Job A
λ λ λ λ λ λ λ λ λ λ
λ λ λ λ λ λ λ λ λ λ

Job B
λ λ λ λ λ λ λ

Job C
λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ
λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ λ

Metadata server(s)

Storage server
CPU
Net
HDD

Storage server
CPU
Net
Flash

Storage server
CPU
Net
Flash

Storage server
CPU
Net
DRAM

Storage server
CPU
Net
DRAM
Evaluating Pocket

- Pocket is intended to be managed by cloud providers
- Evaluate Pocket running on Amazon EC2 virtual machines
- Run serverless applications on AWS Lambda:
  - Video analytics (object recognition)
  - MapReduce sort
  - Distributed software compilation

https://github.com/stanford-mast/pocket
Application performance with Pocket

- Compare Pocket to S3 and Redis, which are commonly used today

- **S3 does not provide sufficient throughput**

**MapReduce sort job hints**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ephemeral capacity</td>
<td>100 GB</td>
</tr>
<tr>
<td>Latency sensitive</td>
<td>False</td>
</tr>
<tr>
<td>Aggregate peak throughput</td>
<td>100 Gb/s</td>
</tr>
</tbody>
</table>
Application performance with Pocket

- Compare Pocket to S3 and Redis, which are commonly used today

Pocket has similar performance to Redis at lower cost (uses Flash vs. DRAM)

MapReduce sort job hints

<table>
<thead>
<tr>
<th>Ephemeral capacity</th>
<th>100 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency sensitive</td>
<td>False</td>
</tr>
<tr>
<td>Aggregate peak throughput</td>
<td>100 Gb/s</td>
</tr>
</tbody>
</table>
Application storage cost with Pocket

- Pocket leverages job attribute hints for cost-effective resource allocation and amortizes VM costs across multiple jobs, offering a pay-what-you-use model.

- Pocket reduces cost by ~60% compared to Redis for all 3 jobs.
Autoscaling the Pocket cluster

<table>
<thead>
<tr>
<th>Job hints</th>
<th>Job1: Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency sensitive</td>
<td>False</td>
</tr>
<tr>
<td>Ephemeral data capacity</td>
<td>10 GB</td>
</tr>
<tr>
<td>Aggregate throughput</td>
<td>3 GB/s</td>
</tr>
</tbody>
</table>
Autoscaling the Pocket cluster

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

<table>
<thead>
<tr>
<th>Job hints</th>
<th>Job1: Sort</th>
<th>Job2: Video analytics</th>
<th>Job3: Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency sensitive</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Ephemeral data capacity</td>
<td>10 GB</td>
<td>6 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Aggregate throughput</td>
<td>3 GB/s</td>
<td>2.5 GB/s</td>
<td>3 GB/s</td>
</tr>
</tbody>
</table>
Summary

To build resource-efficient, high performance storage systems, we need:

1. **Fast access to remote data**
   - [OSDI’14] **IX**: Dataplane OS for Fast Networking
   - [EuroSys’16] Flash Storage Disaggregation
   - [HotStorage’17] Rack-Scale Disaggregated Storage
   - [ASPLOS’17] **ReFlex**: Remote Flash ≈ Local Flash

2. **Automatic, elastic allocation of storage resources**
   - [ATC’18a] Understanding Ephemeral Storage for Serverless Analytics
   - [OSDI’18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics
   - [ATC’18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration
What’s next?
Machine learning for cloud systems

- Use ML to enhance or replace heuristics and user hints
  - E.g., Automate cluster resource allocation
ML-based resource allocation

- Selecta [ATC’18] predicts near-optimal configuration for a job using sparse training data across jobs.

- Profile on 20% of configs
- Profile on 2 reference configs

- Perf/Cost Objective (e.g., minimize cost)

- SELECTA selects near-optimal configuration for a job using sparse training data across jobs.
ML-based resource allocation

- Selecta [ATC’18] predicts near-optimal configuration for a job using sparse training data across jobs
  - 94% probability of recommending near-optimal performing configuration
  - 80% probability of recommending near-optimal cost configuration
Machine learning for cloud systems

- Use ML to enhance or replace heuristics and user hints
  - Automate cluster resource allocation
  - Predict resource utilization for opportunistic computing
  - Debug performance and security by detecting anomalies
Systems for machine learning

ML-based applications

Hardware

GPU
Systems for machine learning

- Near-storage computing for massive datasets
- Distribute computing between the cloud and edge
- Privacy in the cloud with secure enclaves
Conclusion

As we continue exponentially generating and analyzing data in the cloud, we need:

1. Fast access to remote data

   - [OSDI’14] IX: Dataplane OS for Fast Networking
   - [EuroSys’16] Flash Storage Disaggregation
   - [HotStorage’17] Rack-Scale Disaggregated Storage
   - [ASPLOS’17] ReFlex: Remote Flash $\approx$ Local Flash

2. Automatic, elastic allocation of storage resources

   - [ATC’18a] Understanding Ephemeral Storage for Serverless Analytics
   - [OSDI’18] Pocket: Elastic Ephemeral Storage for Serverless Analytics
   - [ATC’18b] Selecta: ML-based Heterogeneous Cloud Storage Configuration
Acknowledgements

Christos Kozyrakis
Patrick Stuedi
Heiner Litz
Yawen Wang
Adam Belay
Animesh Trivedi
Jonas Pfefferle
Binu John
Eno Thereska
George Prekas
Edouard Bugnion