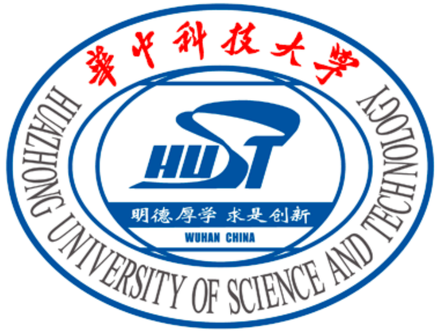
Mass: Workload-Aware Storage Policy for OpenStack Swift

Yu Chen, Wei Tong, Dan Feng, Zike Wang

Huazhong University of Science and Technology



Outline

- Background and Motivation
  - Motivation study
  - Goals & Challenges
- Mass
- Evaluation
- Conclusion

Cloud object storage

- Features
  - Flat address space
  - HTTP-based RESTful web APIs (CRUD)
  - Storage virtualization
- Advantages
  - High availability
  - Flexibility
  - Simple data management

Amazon S3

Ceph

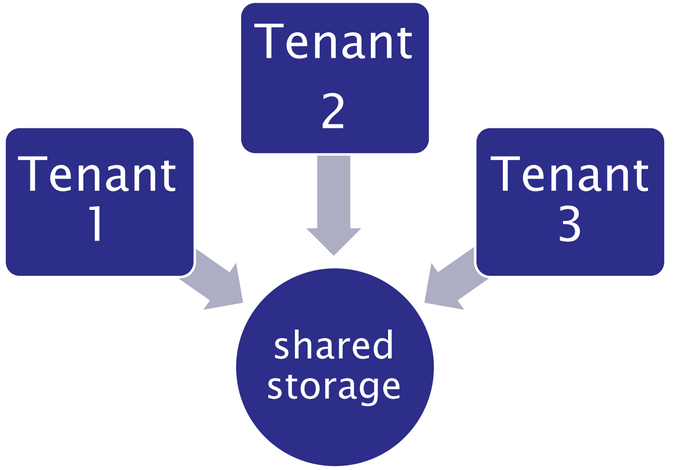
OpenStack Swift

Gap between workloads and storage

- Multi-tenant workloads
  - Different access characteristics
  - Different requirements (latency & throughput)
- Shared storage
  - Monolithic configuration
  - Same service level
- Results in…

➡Limited workload performance

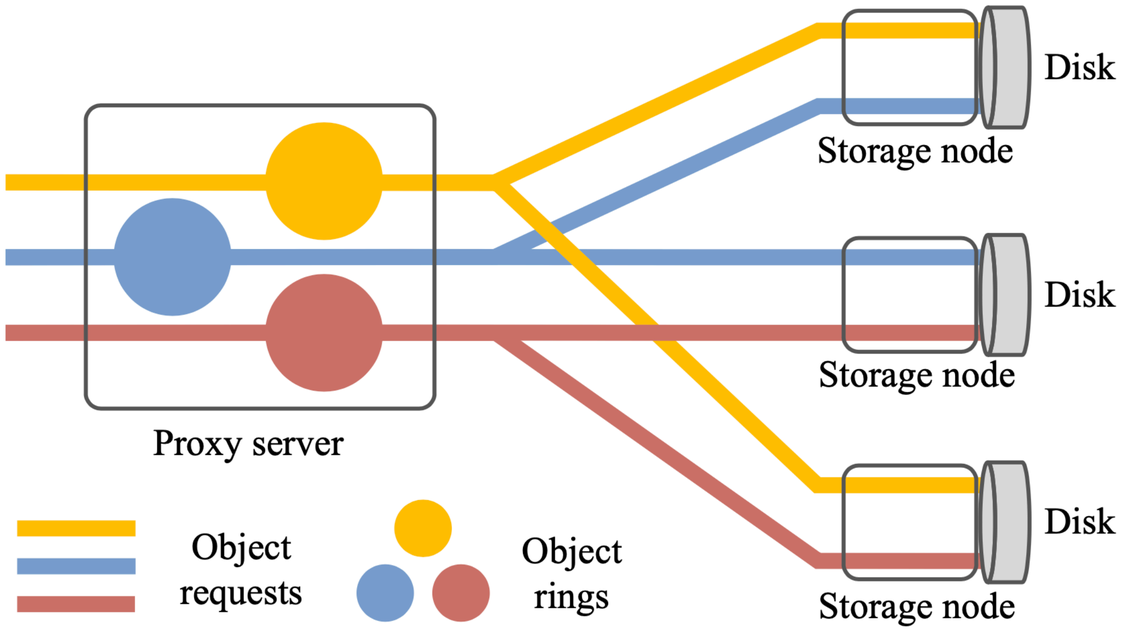
➡Low system efficiency



Application	I/O workload profile	
	Read/write percentage	Size in bytes
Database online transaction processing	70%/30%	8KB
Web file server	95%/5%	8KB, 64KB
Decision support systems	100%/0%	1024KB
Online game hosting	5%/95%	64KB

Storage policy mechanism of Swift

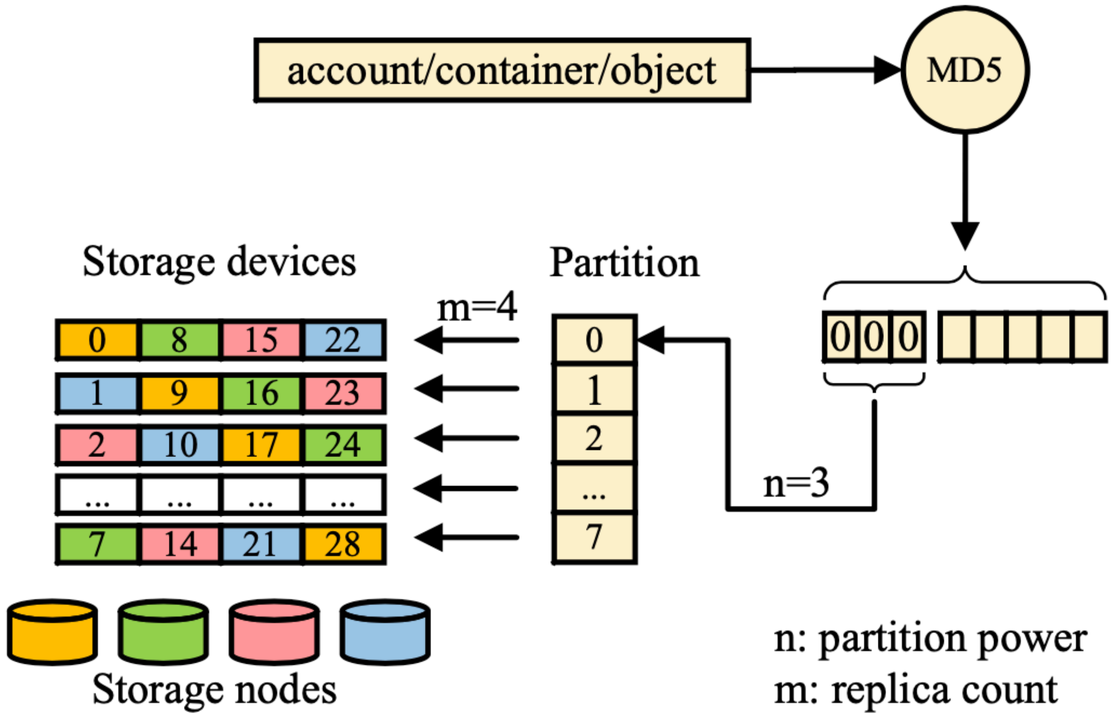
- Two-tier architecture
  - Access tier ⟶ forwarding requests
  - Storage tier ⟶ managing storage devices
- Proxy server
  - Object ring
- Storage node
  - Partition



Request forwarding

Storage policy mechanism of Swift

- Object rings
  - Key role of request forwarding
  - Consistent hashing
  - Two-level mapping
- Storage policy mechanism
  - Creation of the particular object ring
  - Configurable n,m values



Two-level mapping of object ring

Motivation study - advantages

- Comparing with the monolithic setup

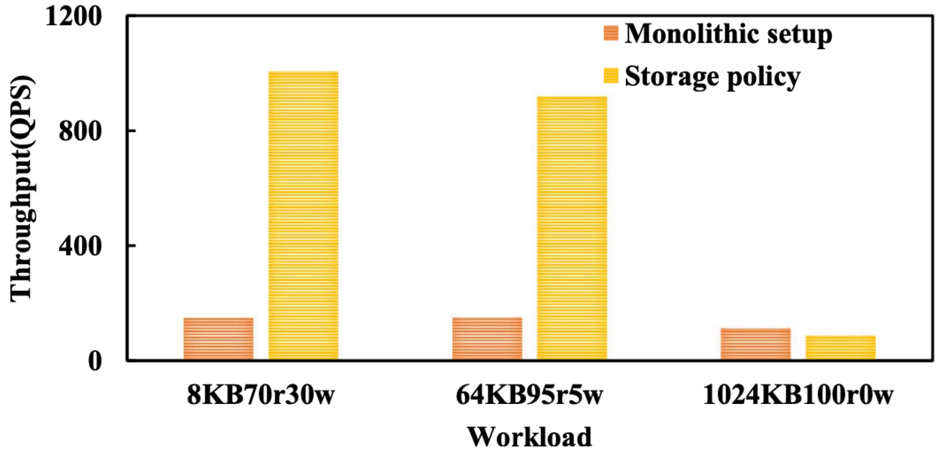
  ➡ **NOT** similar performance level

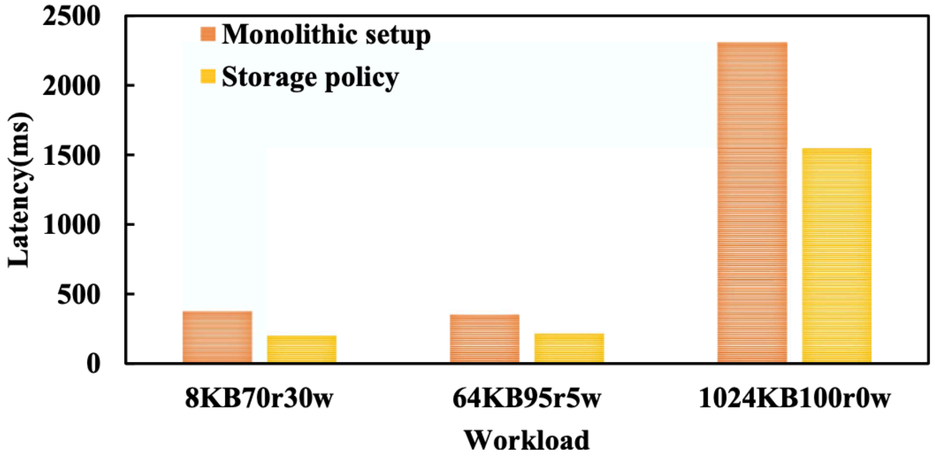
  ➡ Throughput: up to 8.5x increase

  ➡ Latency: up to 33% decrease

  } **better** workload performance

- Analysis
  - Isolated forwarding paths
  - Mitigating resource competition



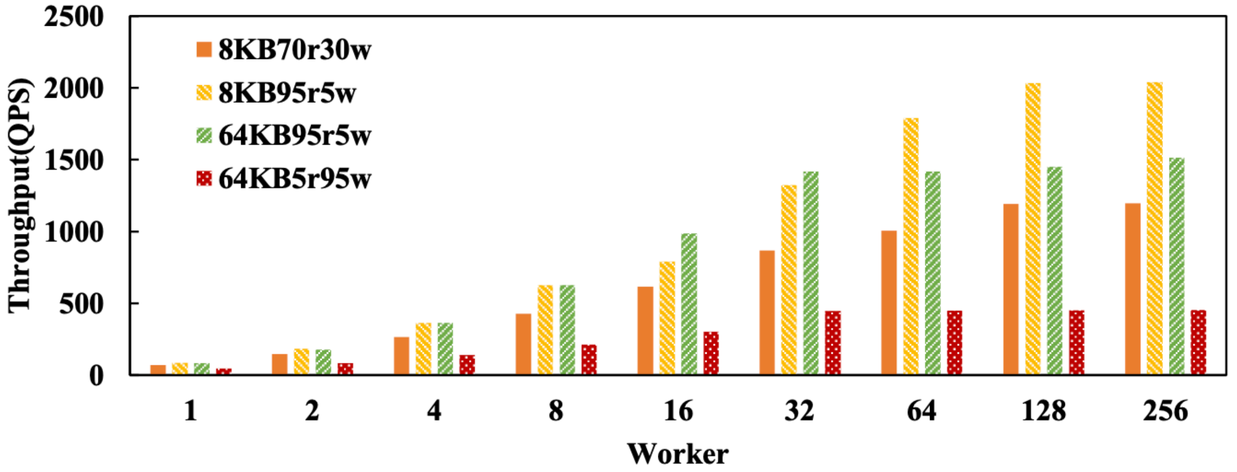


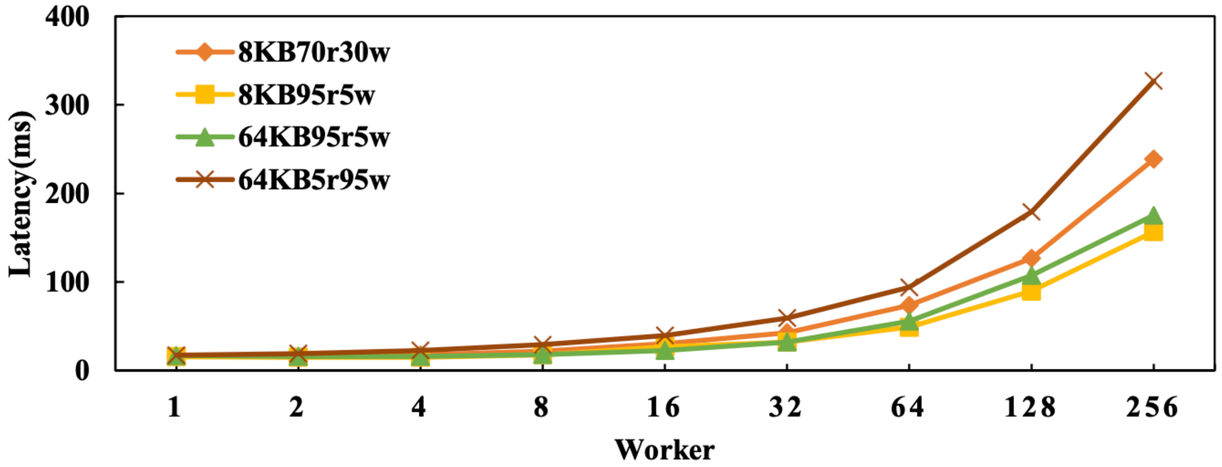
Motivation study - limitations

- Stress tests
  - Varying request concurrency
  - Same storage policies
- Performance results

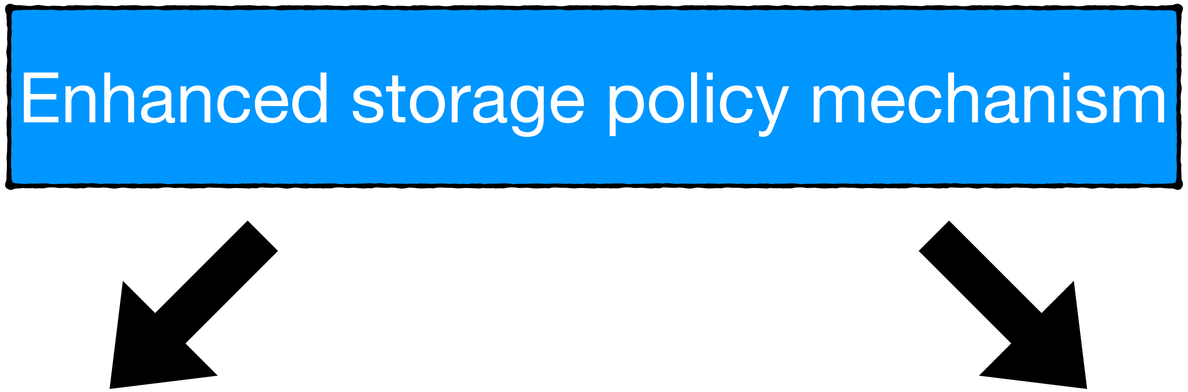
  ➡Throughput reaching saturation

  ➡Latency increasing sharply
- Indicates that…
  - Performance of intensive workloads has room for improvement → **Why?**





Goals & Challenges



Goals

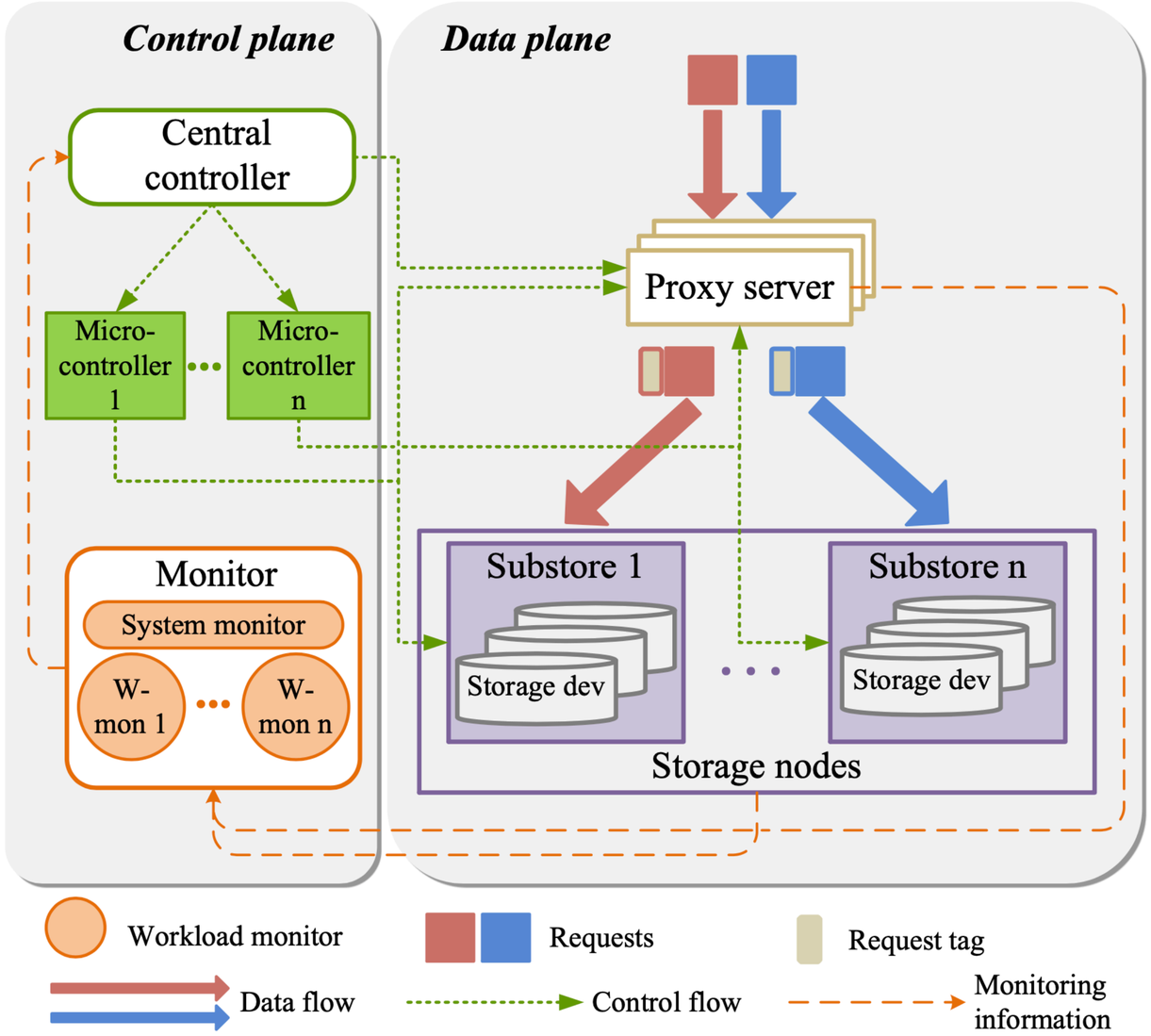
- Covering full-path of request
- Workload-specific
- Performance optimization
- Dynamic mechanism

Challenges

- Controlling request processing path
- Workload classification
- Request identification at storage layer
- Policy adjustment at runtime

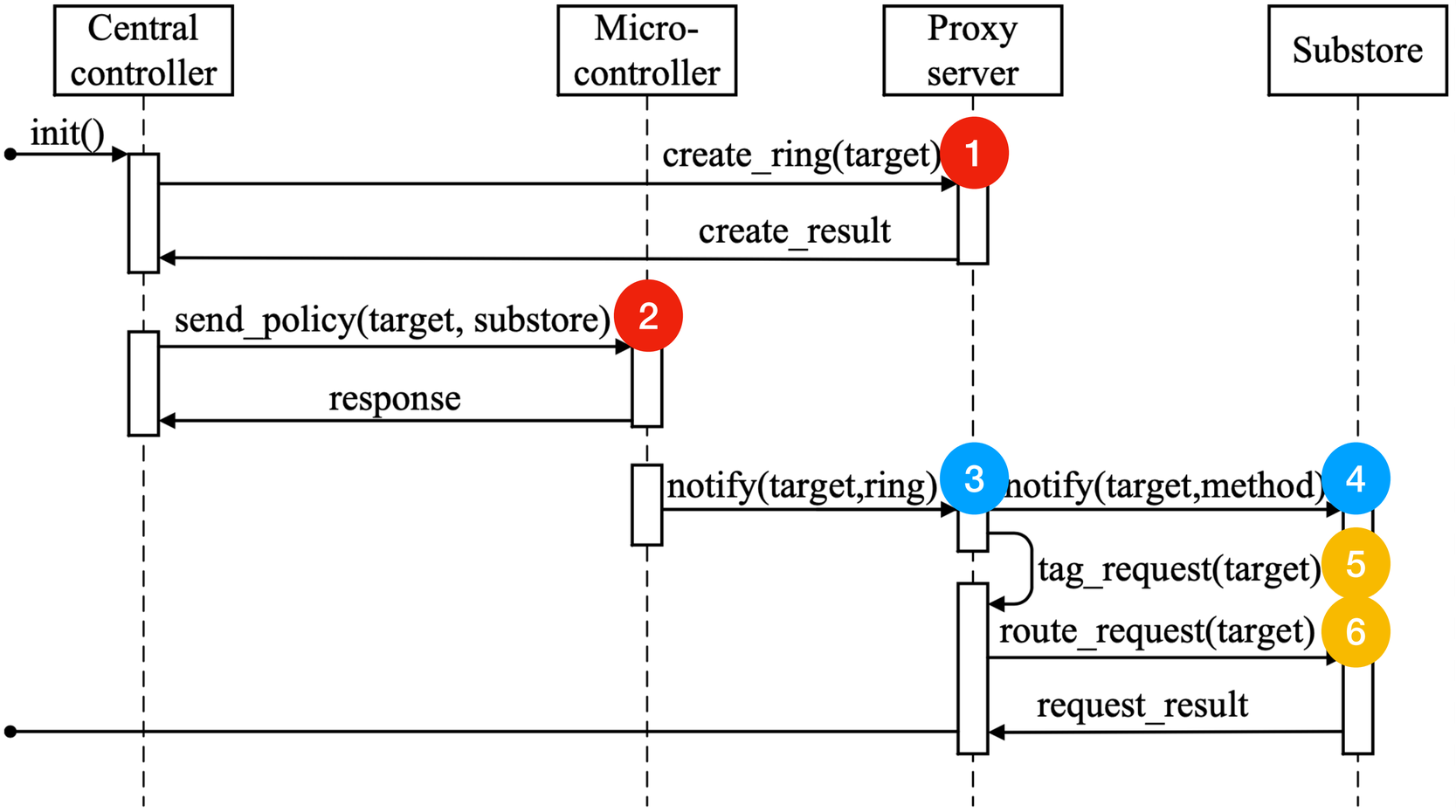
Mass

- Control & Data planes
  - Controller
  - Monitor
  - Substore
- Workload classification
  - Access characteristics
  - Read-dominated, write-dominated, read-write mixed
- Request identification
  - Cross-layer tagging



Overall architecture

Life cycle of a policy



Component interaction

i. **Policy preparation**
   - Monitoring
   - Workload classification

ii. **Policy formulation**
   - Triple: {tenant, ring, method}

iii. **Policy deployment**
   - Optimized request processing

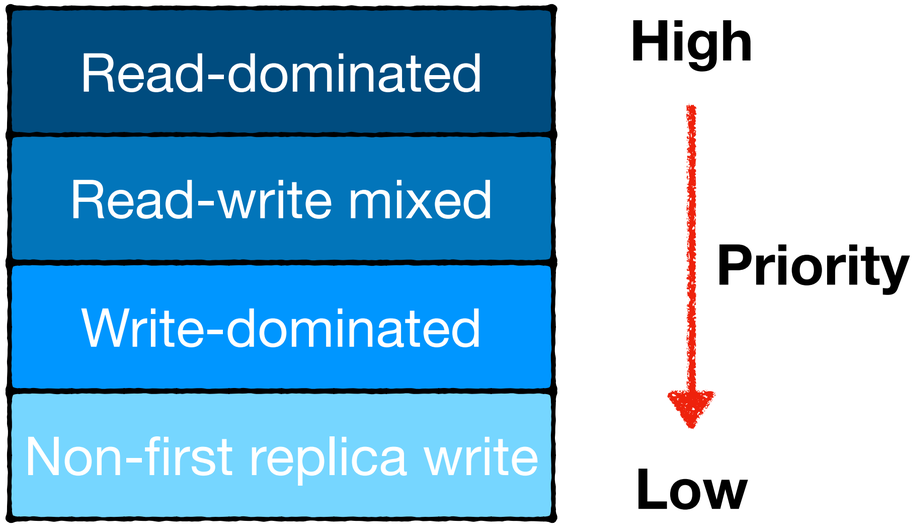
iv. **Policy execution**

Two-level processing optimizations

- Substore-level policy
  - Workload-specific
  - Performance optimization
  - Programmable

Workload type	Performance requirement	Policy
Read-dominated	Latency	Cache
Write-dominated	Throughput	Batch
Read-write mixed	Latency & Throughput	Merge

- Storage node level policy
  - Priority-based queuing
  - System efficiency



Dynamic policy mechanism

- Workload changes
  - External
  - Internal
- Validation
- Policy adjustment

- Improper resource allocation
- Policy overhead

- Insertion
- Deletion

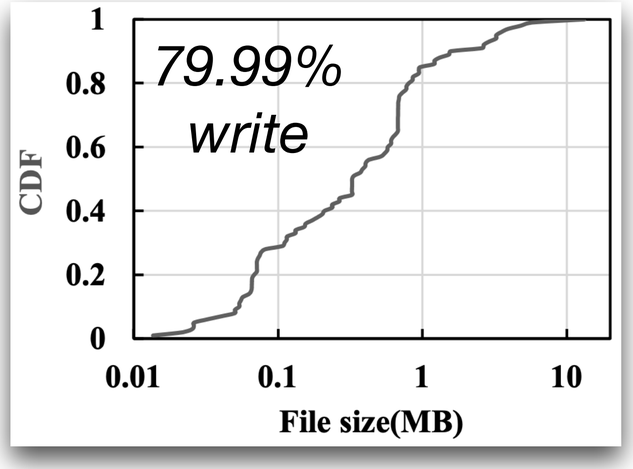
Evaluation setup

- Cluster
  - 2 proxy servers
  - 5 storage nodes
  - 3 workload generators
- Workload
  - Synthetic workloads
  - Real-world traces

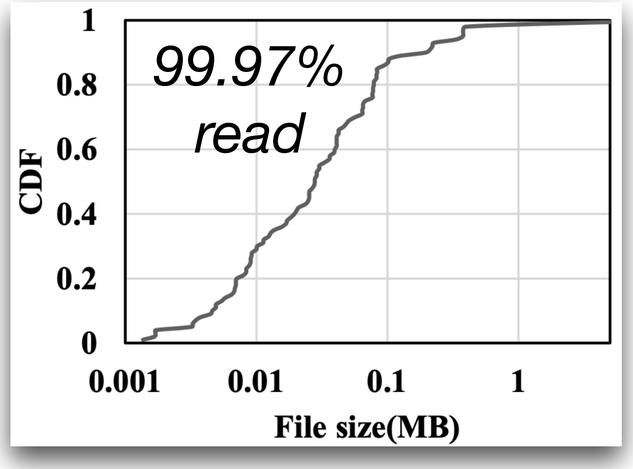
- Storage setup
  - Default: Swift's original policies
  - Crystal: Manual workload-specific policies
  - MASS: Dynamic workload-specific policies & priority-based queuing

Workload	Read/write ratio	Object size	Policy
A	100%/0%	512KB	Cache
B	50%/50%	64KB	Merge
C	0%/100%	8KB	Batch

Synthetic workloads

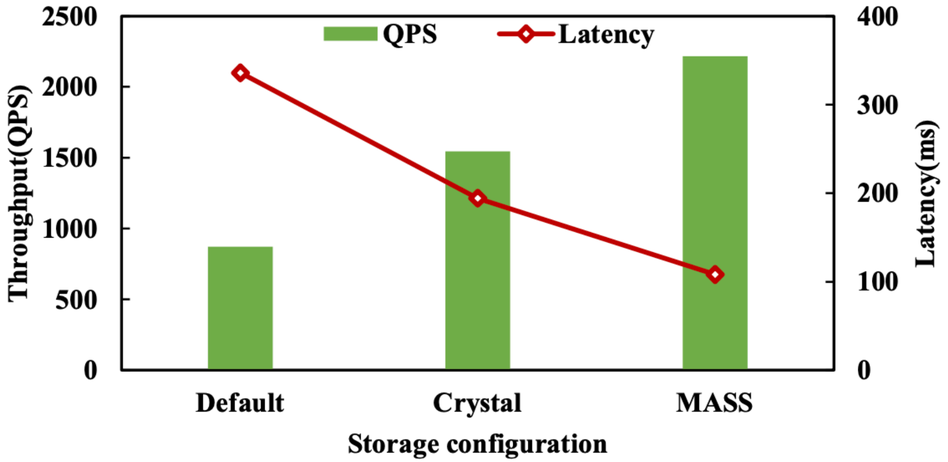


Idiada trace



Arctur trace

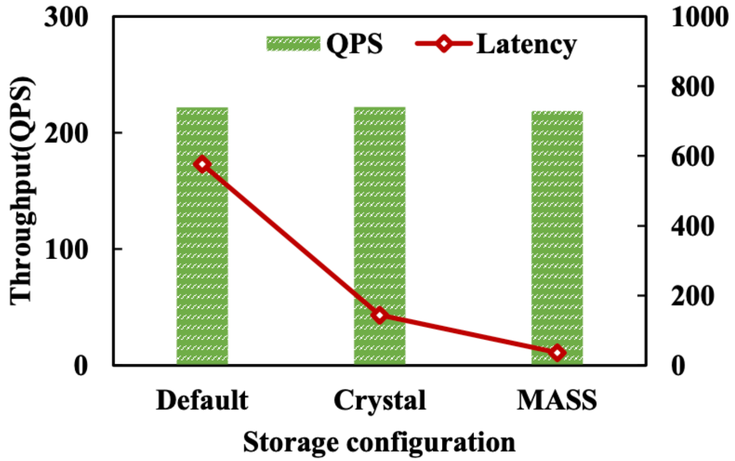
Effectiveness of policy



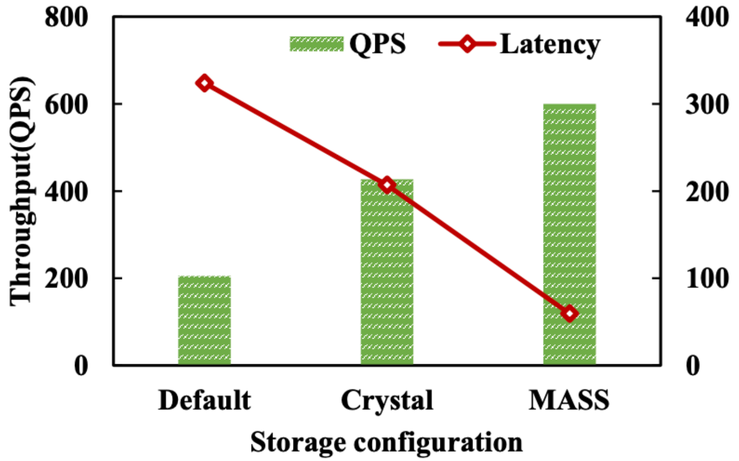
- Overall system performance

  ➡ 154.3% higher throughput and 67.8% lower latency

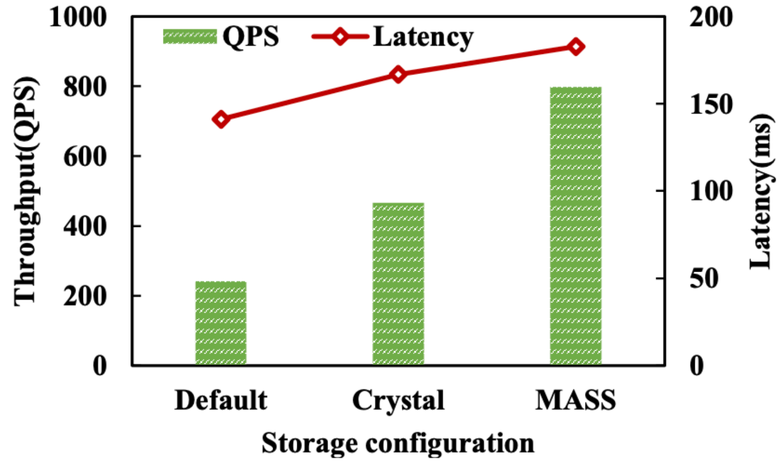
Workload A
93.7% lower latency



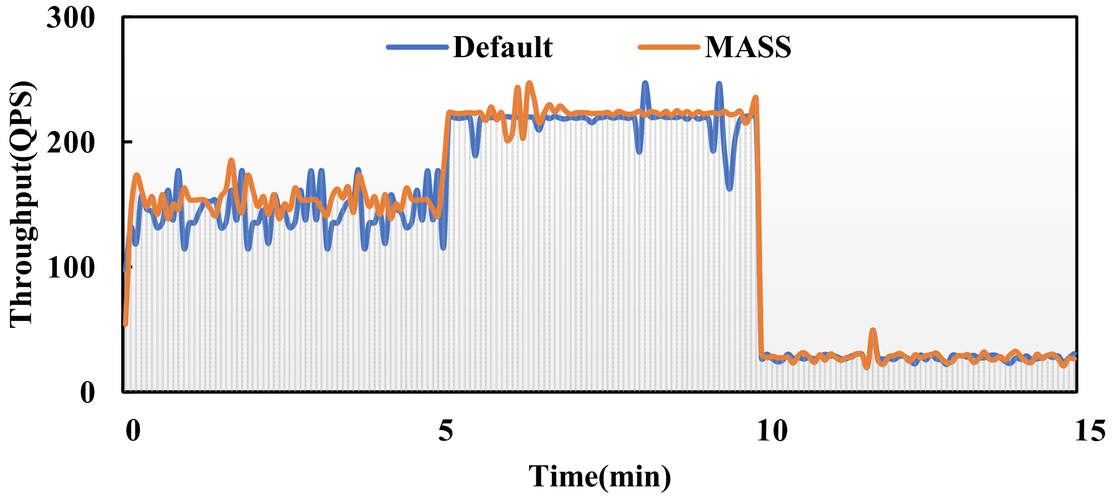
Workload B
81.6% lower latency
191.2% higher throughput



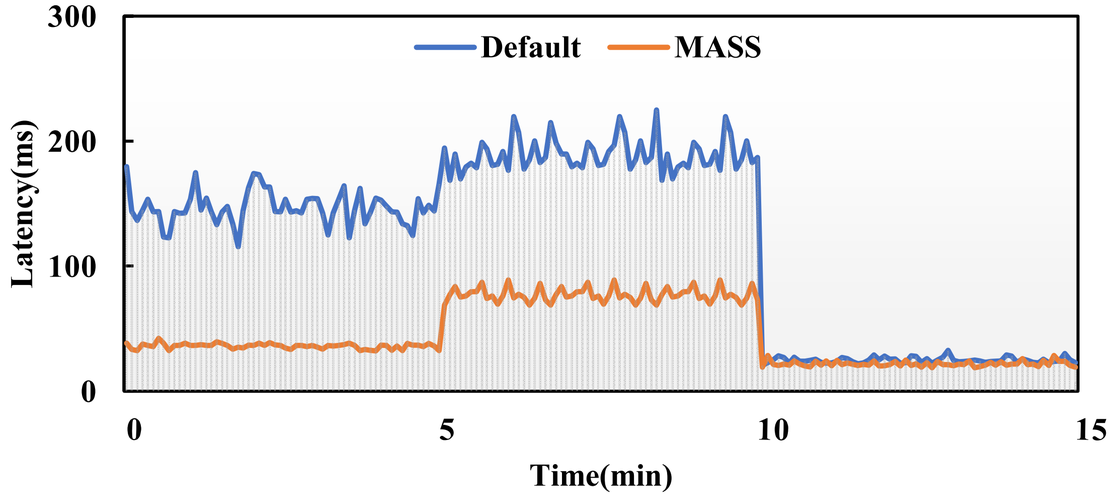
Workload C
231.5% higher throughput



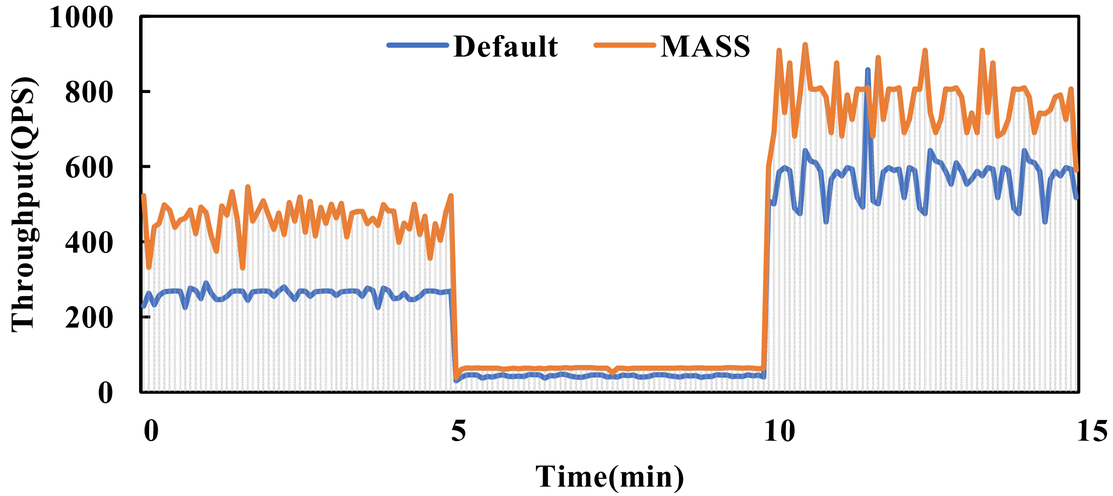
External workload change



Workload A



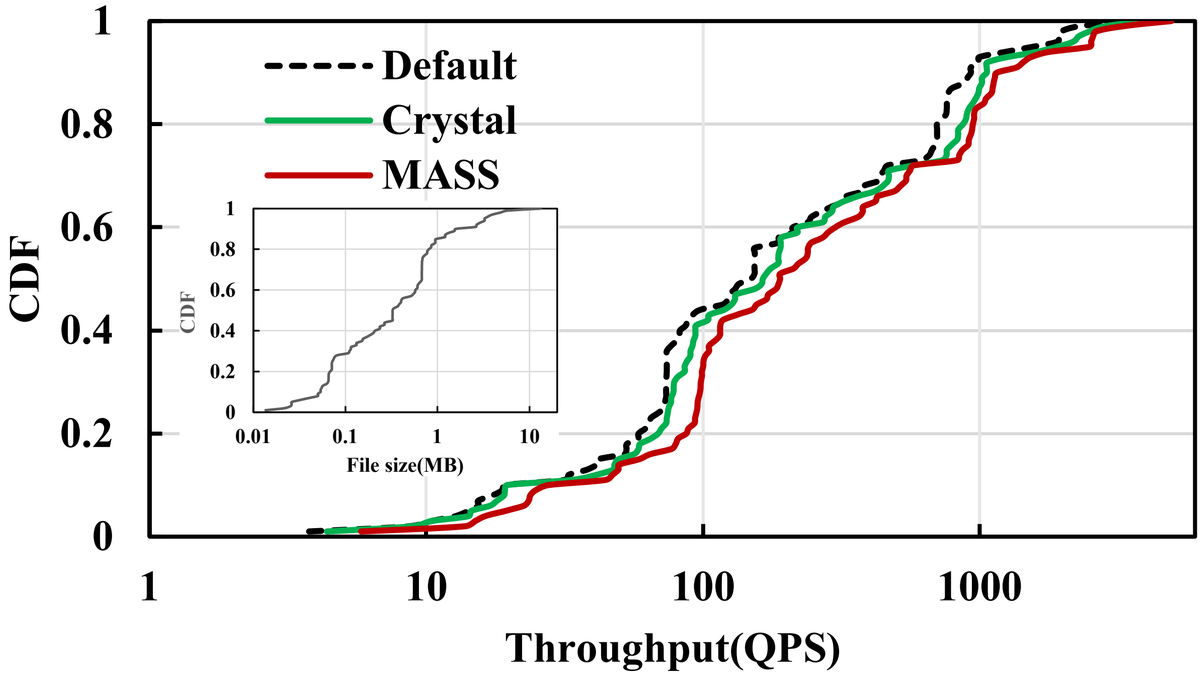
Workload A

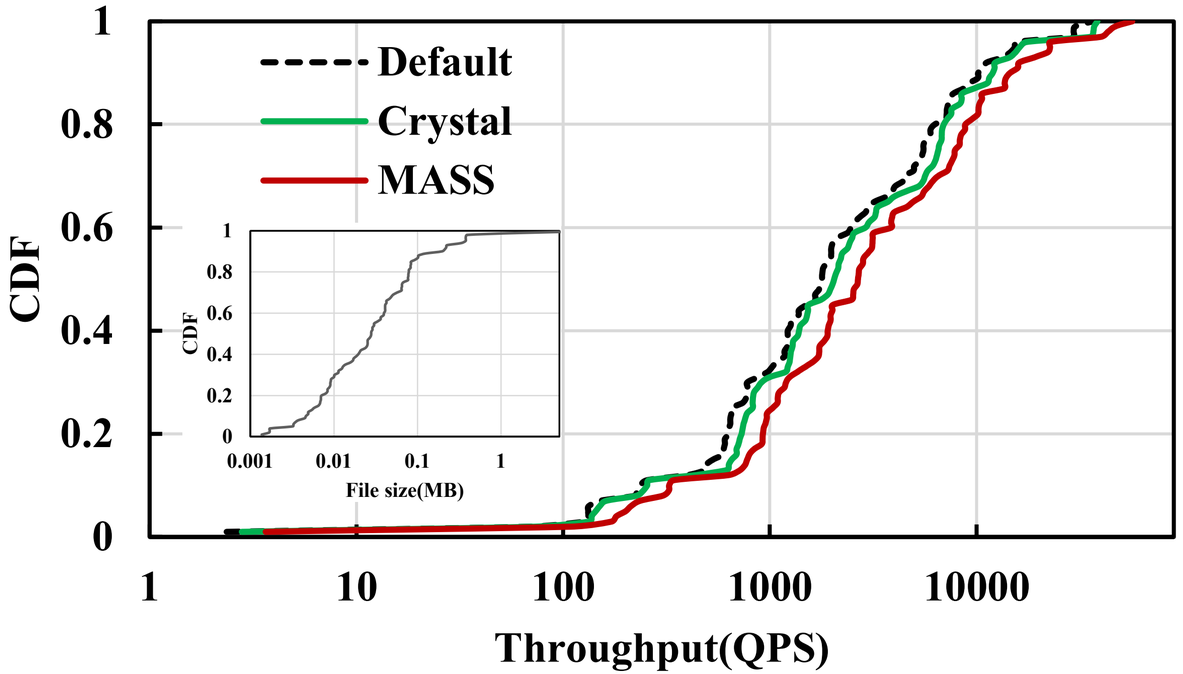


Workload C

- Three-stage test
  - Baseline & A-dominated & C-dominated
  - Workload A: 61.9% lower latency
  - Workload C: 55.2% higher throughput

Internal workload change





- Comparing with
  - Default: average 61.3% promotion
  - Crystal: average 37.6% promotion

- Comparing with
  - Default: average 59.4% promotion
  - Crystal: average 39.3% promotion

Conclusion

- Original storage policy mechanism
  - Poor performance of intensive workloads
  - Unable to react to workload changes
- We propose Mass to enhanced flexible polices
  - Covering full storage path
  - Workload-aware optimizations based on access characteristics
  - Dynamic policy adjustment
- Better workload performance and system efficiency

Thanks! Q&A

Email: chloe_chen@hust.edu.cn