Client-aware Cloud Storage

Feng Chen
Computer Science & Engineering
Louisiana State University

Michael Mesnier
Circuits & Systems Research
Intel Labs

Scott Hahn
Circuits & Systems Research
Intel Labs

Cloud storage service is flourishing

Enterprise Cloud Storage

Consumer Cloud Storage

- Personal cloud storage subscriptions reach **500 million** in 2012[1]
- Public/private cloud storage market is predicted to be **$22.6 billion** by 2015[2]

[1] http://www.networkworld.com/news/2012/090712-personal-clouds-262231.html
[2] http://www.idc.com/getdoc.jsp?containerId=prUS23097611

QoS in cloud storage – a critical challenge

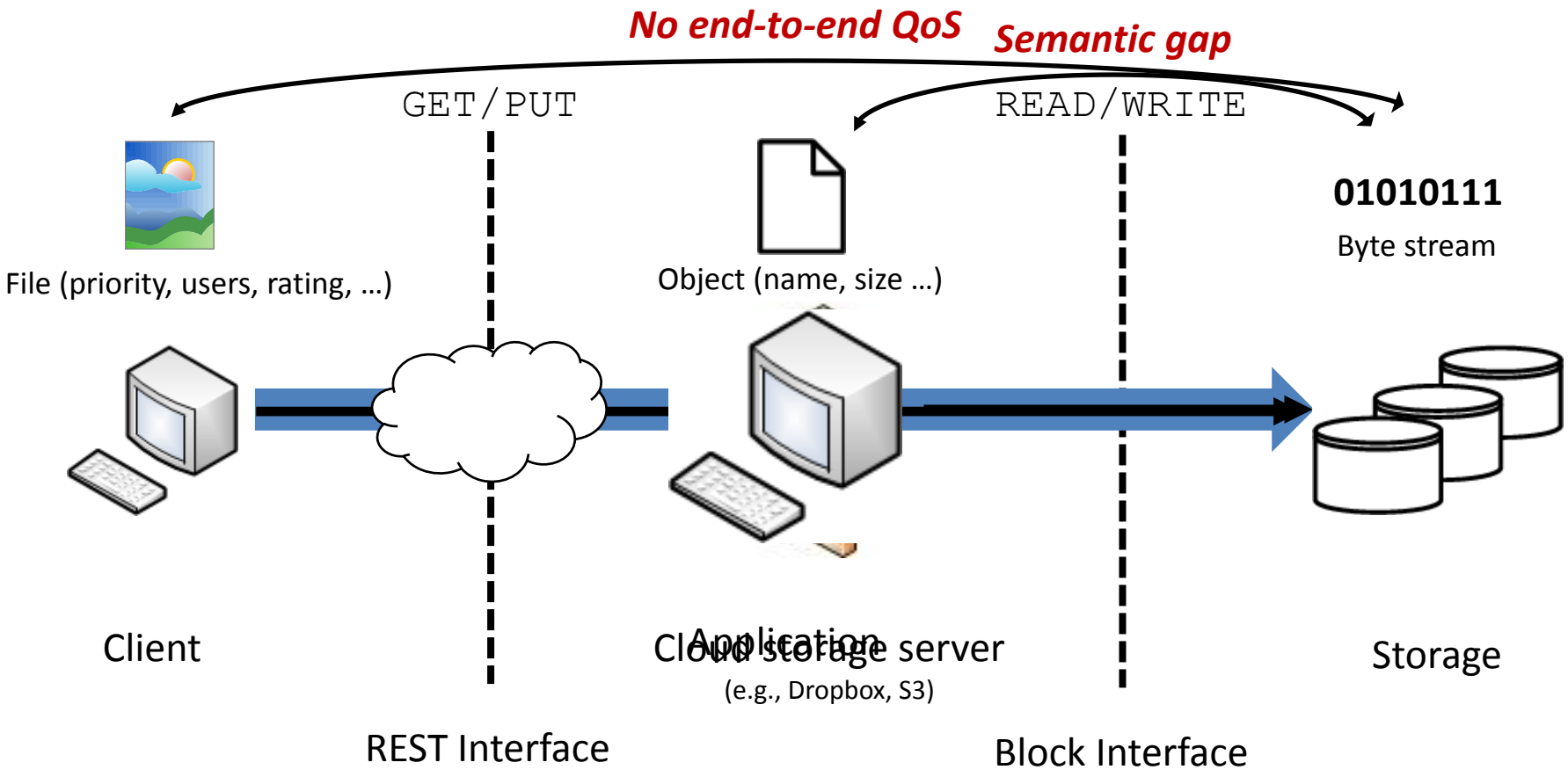
- QoS in today's cloud storage services
  - QoS is a real demand (e.g., Amazon Glacier, SSD-backed EC2 instance[1])
  - Limitations: one-size-fits-all, coarse grained, and difficult to use

- Differentiated Storage Services [SOSP'11, VLDB'12]
  - Flexible, fine grained, and configurable to apps (e.g., database)
  - Data classification could enable **end-to-end QoS** in cloud storage

$0.01/GB/Month

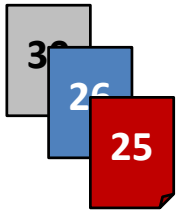
[1] http://techblog.netflix.com/2012/07/benchmarking-high-performance-io-with.html

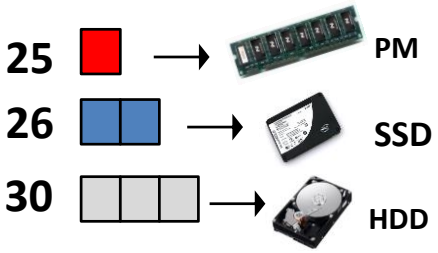
Semantic gap further widens in cloud storage

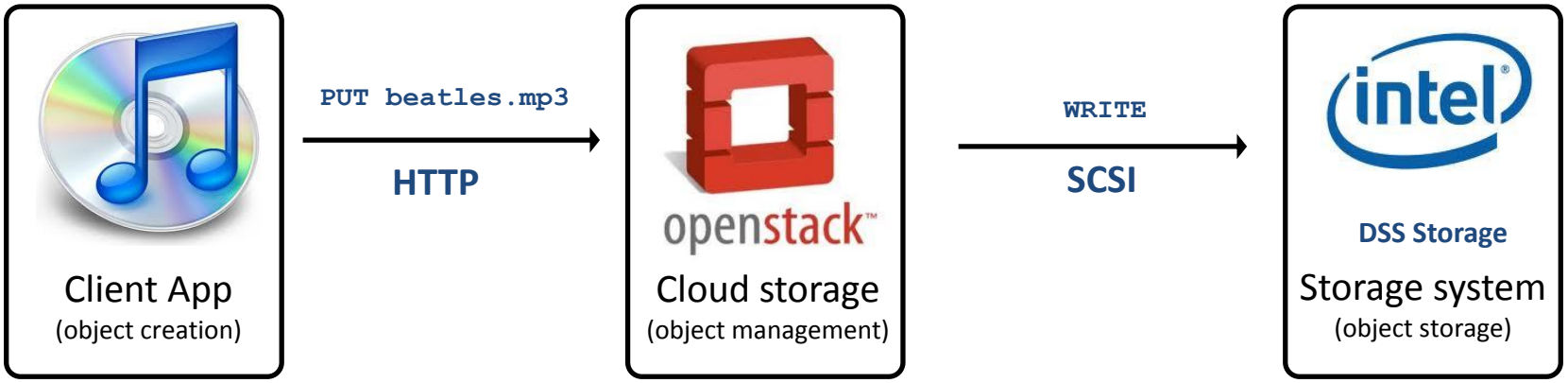


Our goal – let semantic information flow together with data from end to end

An example of many possible use cases …







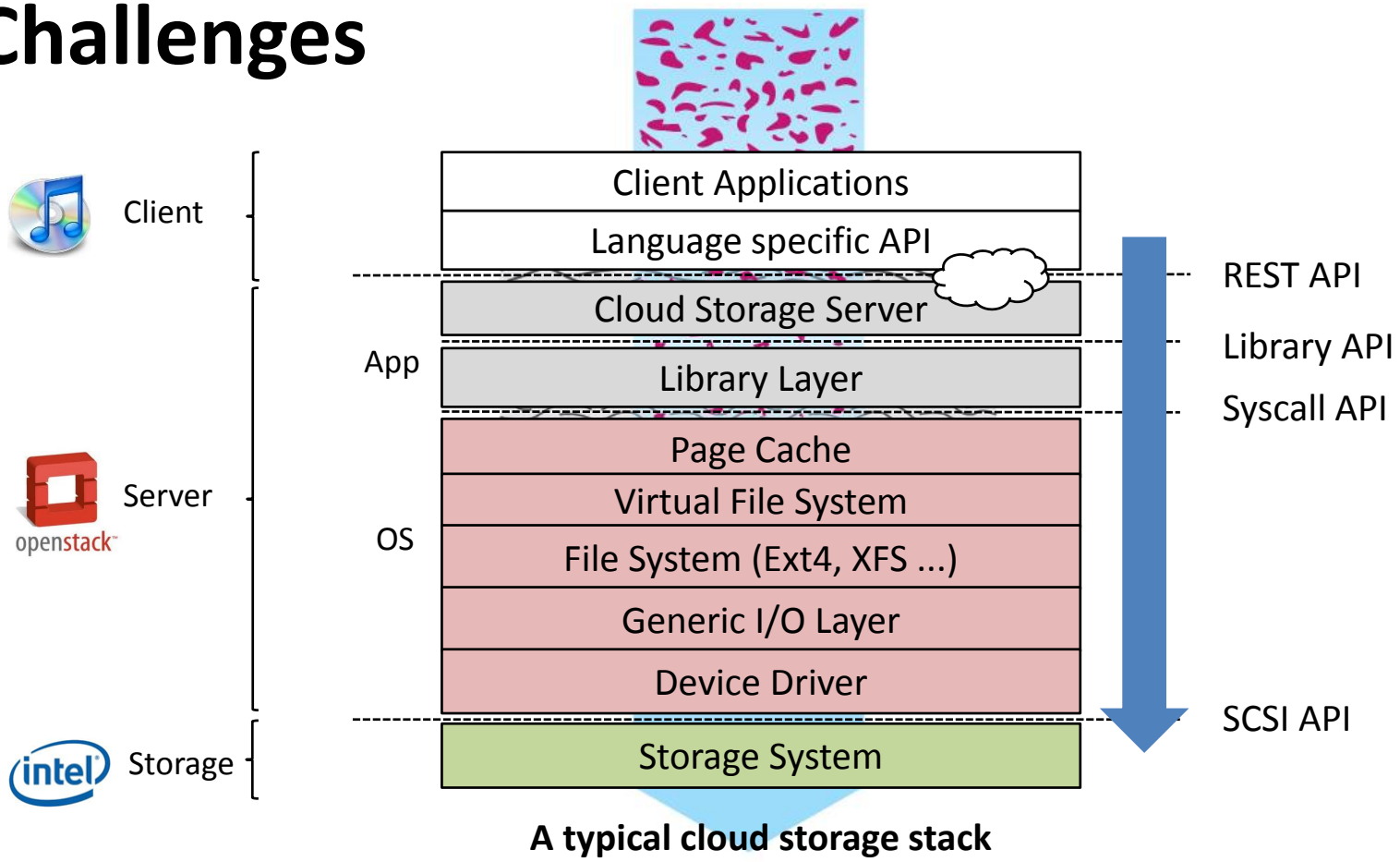
Classify data

Generate classified I/Os

Enforce QoS policies

Client-awareness *can enable many QoS opportunities (performance, reliability, security ..)*

Challenges

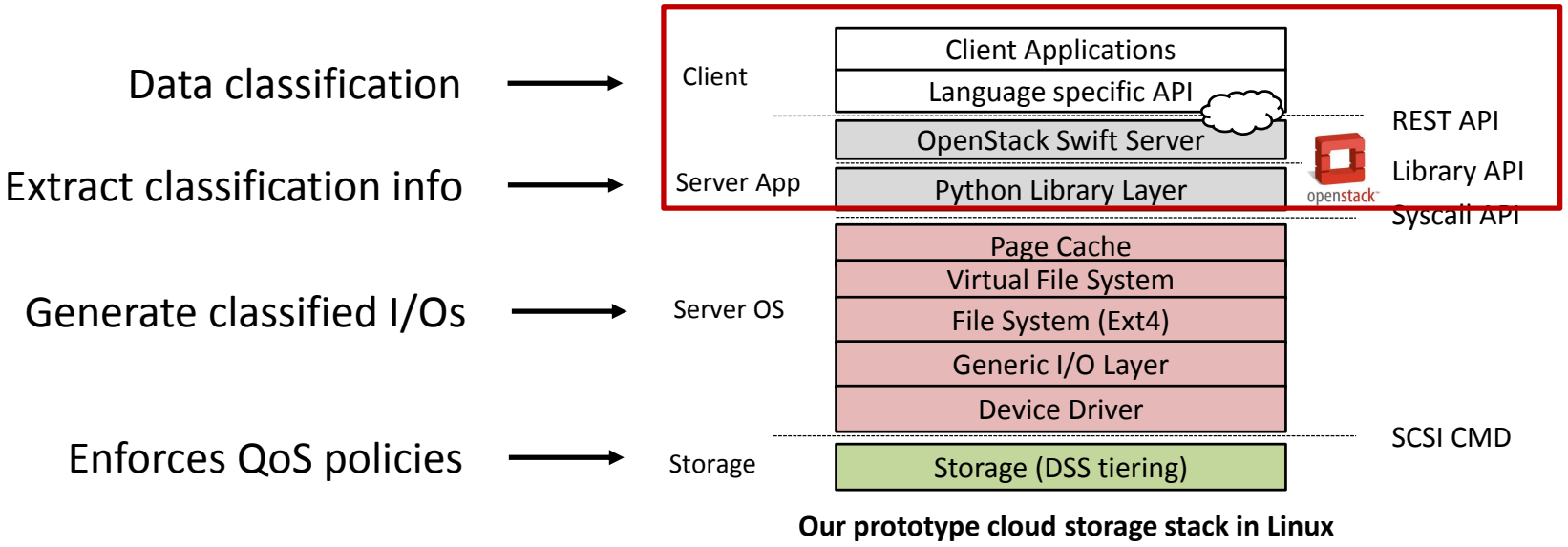


A typical cloud storage stack

- Semantic info flow must cross multiple layers/interfaces with data
- No support in existing cloud storage for end-to-end semantic info flow

Client-aware cloud storage

- **Client** – classify data and generate classified HTTP requests
- **Server** – Modified **OpenStack Swift 1.4.6** to handle classified requests
- **Application API lib** – interface to the DSS-enabled scatter/gather I/O syscalls
- **Linux OS kernel** – modified Linux 3.2.1 kernel with patched Ext4 to handle DSS I/Os
- **Storage system** – Differentiated Storage Services (RAID-9) module in Linux kernel



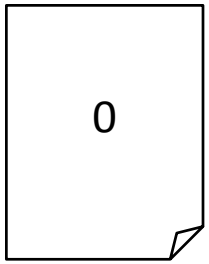
Our prototype cloud storage stack in Linux

Outline

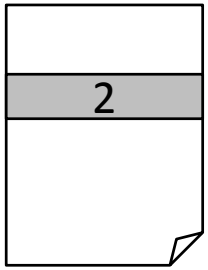
- Introduction
- Design & Prototype
- Experimental Evaluation
- Conclusion

Collecting semantic hints on client

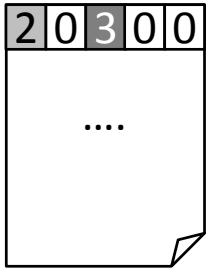
- Principle – separating *data classification* and *policy enforcement*
  - Similar to DiffServ in network scenario for classifying network streams
  - "*What data is*" (Client) vs. "*How data should be treated*" (Storage)
- **Class** – a numerical value labeling a group of data (i.e., handle, index …)
- Data classification in client-aware cloud storage
  - Object-based classification – classify an entire object (e.g., music)
  - Range-based classification – classify a range of data in an object (e.g., I-frames in video)
  - Block-based classification – classify each block in an object (e.g., VM disk)



Object-based Classification



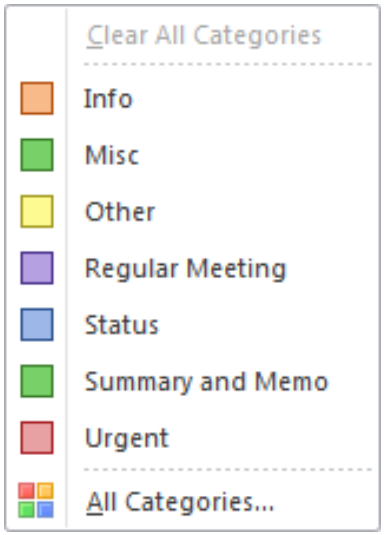
Range-based Classification



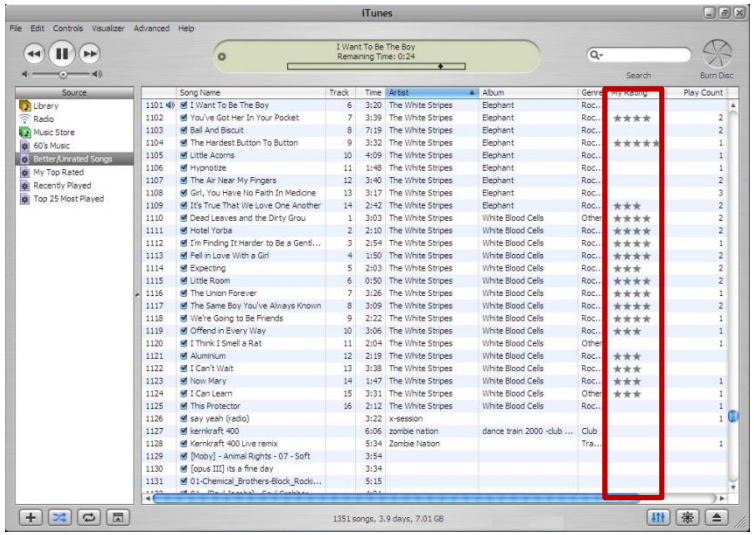
Block-based Classification

How one might classify data

- **Manual** – classify objects in CLI, or label files in a right-click menu, etc.
- **Automatic** – apps extract valuable semantic info (e.g., ranking for songs)
- **General-purpose** – classify data based on file type, size, user group, etc.



Color labeling in MS Outlook



Rating info in Apple iTunes

Client needs to be modified to send classification information to cloud storage

Transmitting classifiers to server

- An REST HTTP request (under the hood of Dropbox, S3, …)

- How to transmit classifiers to server

	In-band mode	Out-of-band mode
Embed classifiers in header	• Object-based • Range-based	• Object-based • Range-based
Embed classifiers in object	• Object-based • Range-based • Block-based	• Object-based • Range-based • Block-based

Embed classifiers in headers

- Object-based classification
  - PUT with a new HTTP header "X-DSS-Object-Class"

```
$ curl -X PUT -H "X-Auth-Token: abc" -T "foo" \
       -H "X-DSS-Object-Class: 25" \
       http://localhost:8080/v1/Auth_test/c1/foo
```

- Range-based classification
  - PUT with a new HTTP header "X-DSS-Range-Class"
  - Range format: <offset>-<len>-<class>
  - Multiple ranges can be specified with "," in between

Class is just a handle (to lookup an associated QoS policy)

```
$ curl -X PUT -H "X-Auth-Token: abc" -T "foo" \
       -H "X-DSS-Range-Class: 0-64-25,1024-32-26" \
       http://localhost:8080/v1/Auth_test/c1/foo
```

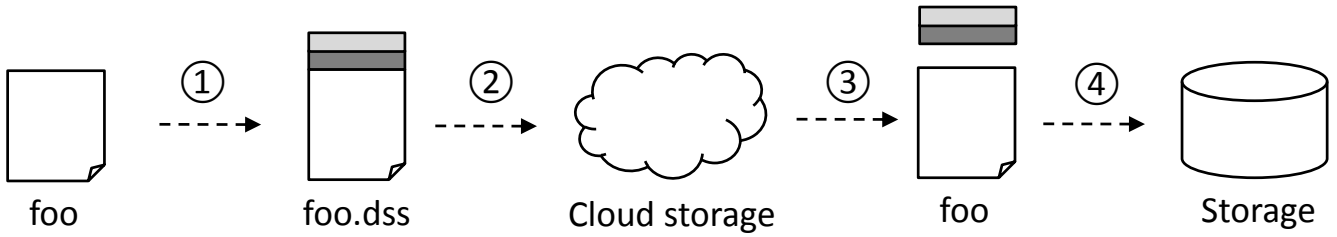
Limitation – the HTTP header size is limited (e.g., 8190 bytes for Apache)

Embed (unlimited) classifiers in objects

- Block-based classification
  - PUT with a new HTTP header "`X-DSS-Object-File: True`"

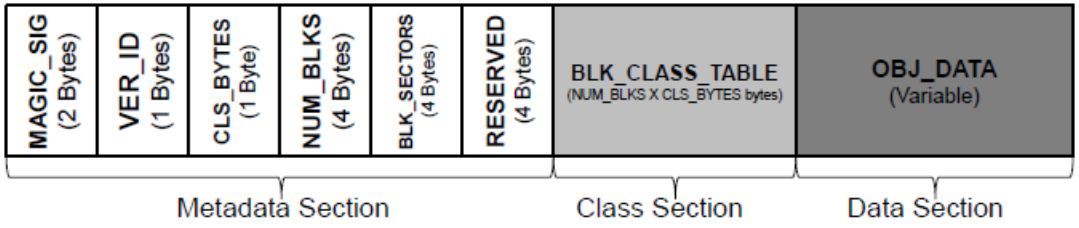
```
$ curl -X PUT -H "X-Auth-Token: abc" -T "foo.dss" \
       -H "X-DSS-Object-File: True" \
       http://localhost:8080/v1/Auth_test/c1/foo
```

  - Instrument the object with a self-describing format
    - *Metadata* – interpreting the format of class section
    - *Class* – specifying the classes of blocks in the object
    - *Data* – intact content data of the object

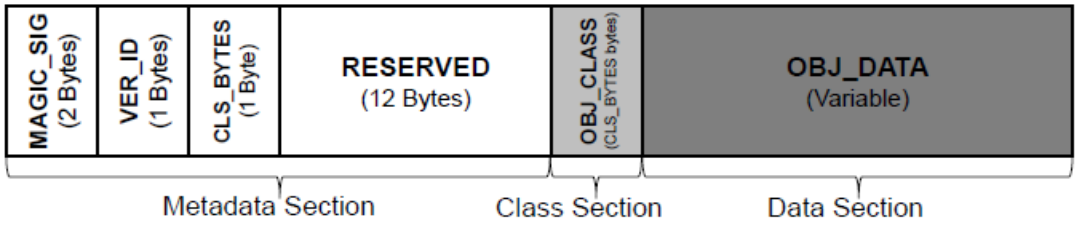


An example self-describing object format

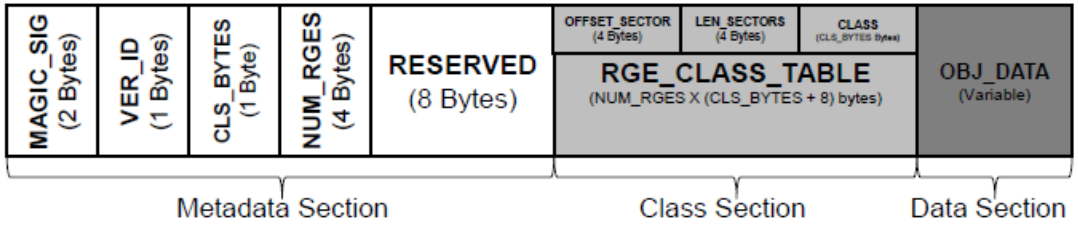
Block-based classification



Object-based classification

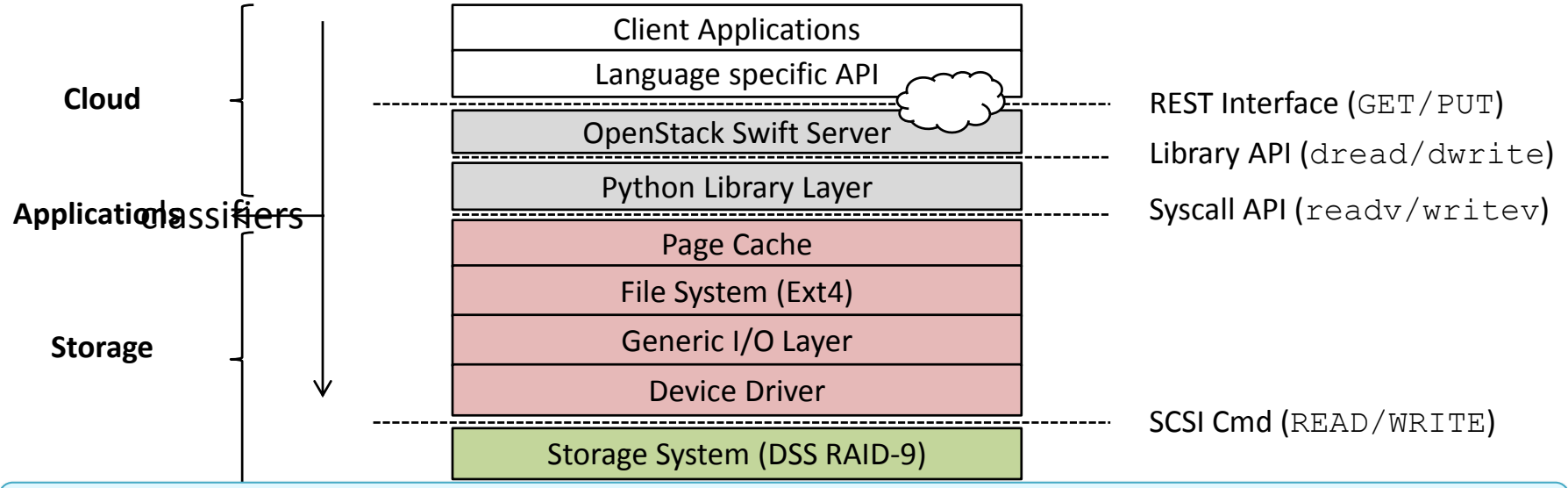


Range-based classification



Handling classified requests at server

- Cloud storage server extracts classifiers from classified HTTP requests
- Generate classified I/Os using language-specific (e.g., Python) APIs
- Submit I/Os to OS kernel via scatter/gather syscalls (`readv`/`writev`)
- OS kernel receives and passes classified I/Os from app to FS, BIO, to DD
- Classifiers are copied to the 5-bit group number of `READ`/`WRITE` SCSI CDB



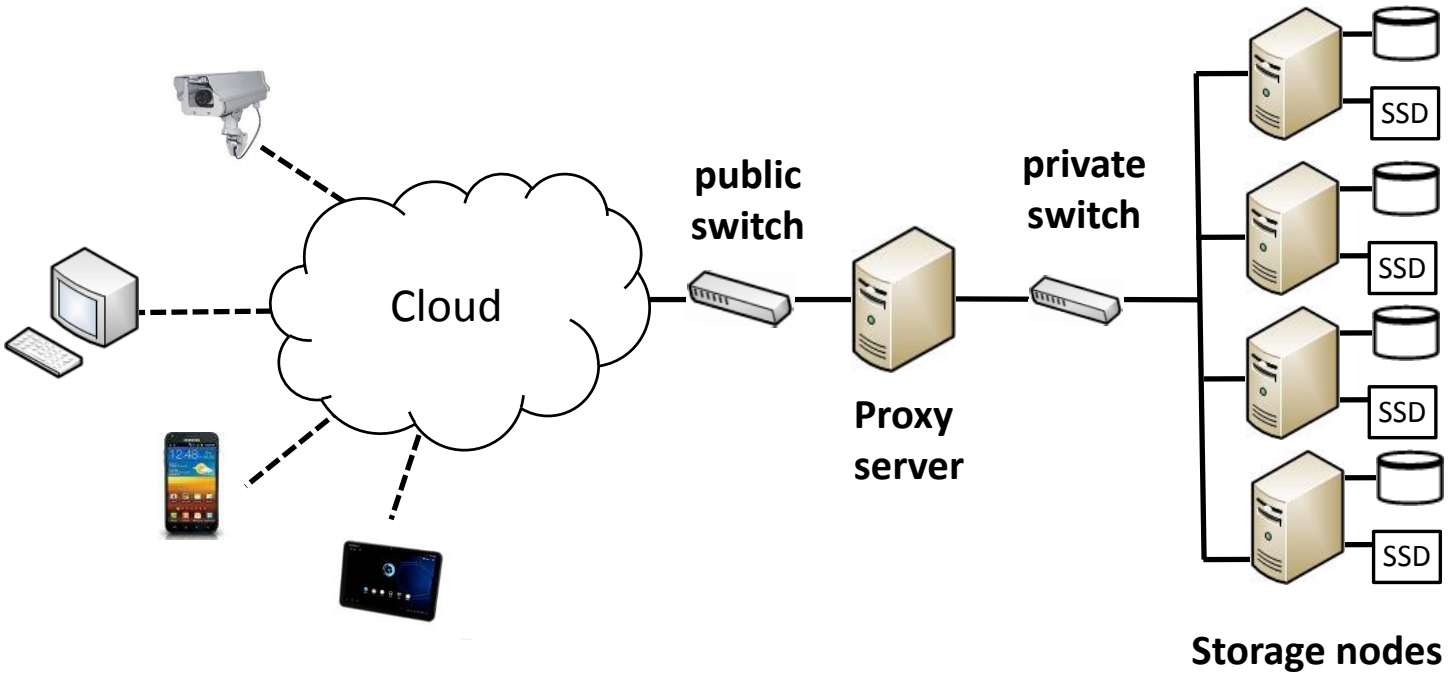
Cloud storage server needs to be modified to handle classified requests

Outline

- Introduction
- Design & Prototype
- Experimental Evaluation
- Conclusion

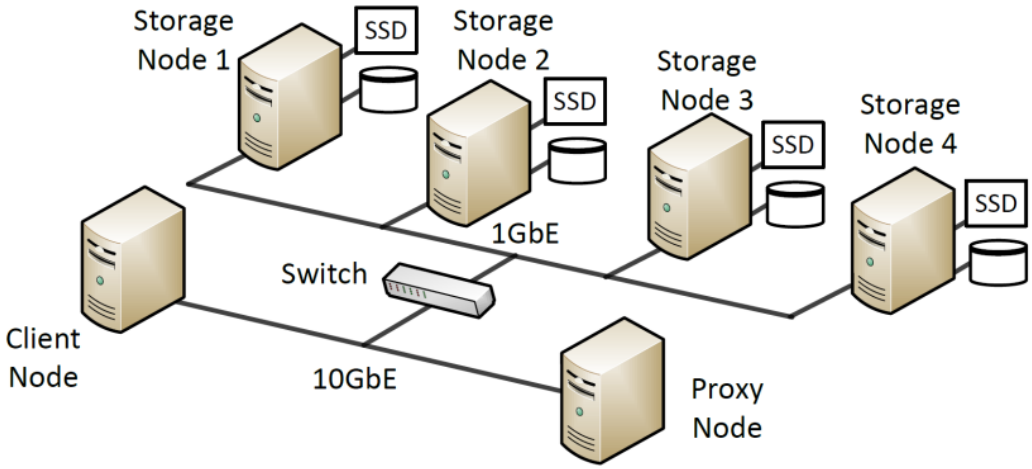
Evaluation Model

- Simulate a typical cloud storage service (e.g., Real WS)
- Compare various storage solutions for cloud storage services
  - Bandwidth, latencies, cost efficiency, etc.



System setup

- Cluster configuration
  - 1 Proxy & 1Client – 2x 8-core Xeon Sandy Bridge 2.9GHz with 128GB memory
  - 4 Storage nodes – 2x 6-core Xeon Westmere 3.3 GHz with 24GB memory
  - Storage devices – 1x Intel 710 SSD, 1x Seagate Constellation 1TB SATA HDD
  - Network setup – 10GbE links for proxy/clients, 1GbE links for storage nodes



A typical OpenStack Swift cluster setup

System setup

- Software configuration
  - OS: Fedora Core 14
  - Patched Linux kernel 3.2.1 and Ext4 for DSS
  - Modified OpenStack Swift 1.4.6 for handling classes
    - Services – proxy, object, container, account, updater, replicator, auditor
    - Proxy service with 32 workers and storage services with 8 workers
    - Each storage device is set as an individual zone, 3 replicas for reliability
- Storage node configuration

HDD-only

LRU caching

DSS caching

SSD-only

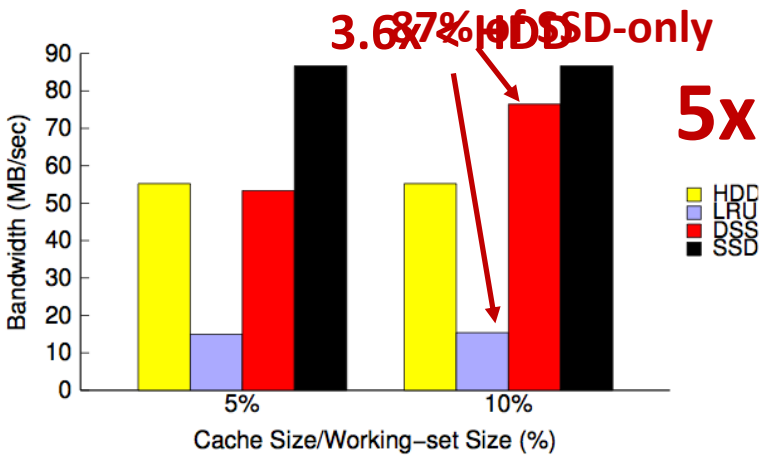
Case 1: Persistent caching for cloud storage

- **Object classification**
  - Distributions are mostly based on real files (10,711 pictures, 319,073 videos)
  - Each object type is given an individual class (USER1-USER5)
  - For all object types, large files (>=10MB) are classified to USER7
- **DSS Caching policy**
  - Caching high-priority data first in the order of USER1-USER7
  - LRU caching is used for data within the same class

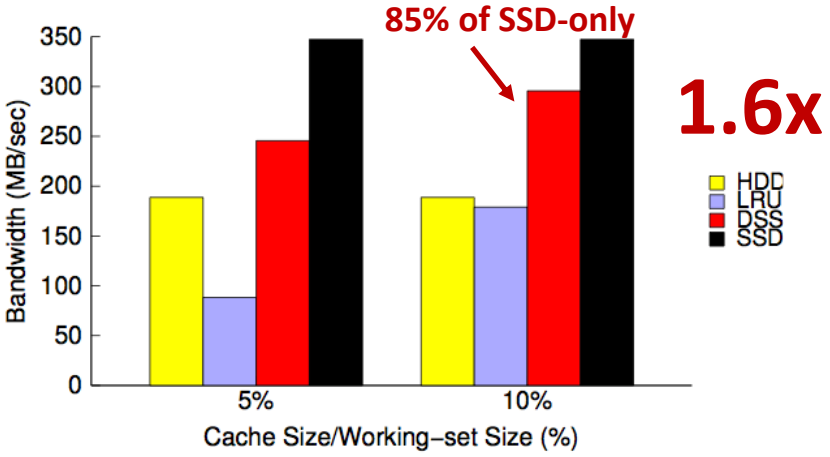
Object Size	Files (60%)	Picture (35%)	Music (4%)	Video (0.9%)	VMDK (0.1%)
≤64KB	USER1 (79%)	USER2 (17.1%)	USER3 (0%)	USER4 (0.3%)	USER5 (0%)
≤512KB	USER1 (14%)	USER2 (43.6%)	USER3 (0%)	USER4 (2.8%)	USER5 (0%)
≤1MB	USER1 (3%)	USER2 (14.5%)	USER3 (0.8%)	USER4 (4.9%)	USER5 (0%)
≤5MB	USER1 (2%)	USER2 (20%)	USER3 (52.3%)	USER4 (32.4%)	USER5 (0%)
≤10MB	USER1 (1%)	USER2 (3.5%)	USER3 (39.6%)	USER4 (31.7%)	USER5 (0%)
>10MB	USER7 (0%)	USER7 (1.3%)	USER7 (7.3%)	USER7 (27.9%)	USER7 (100%)

Distribution and Classification

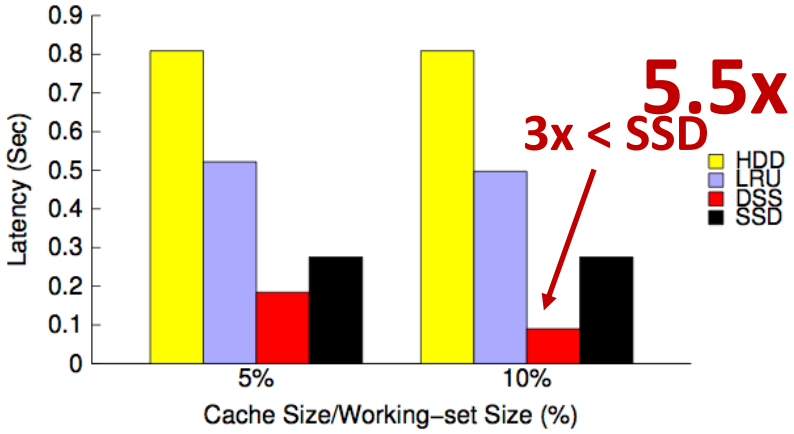
Experimental result highlights



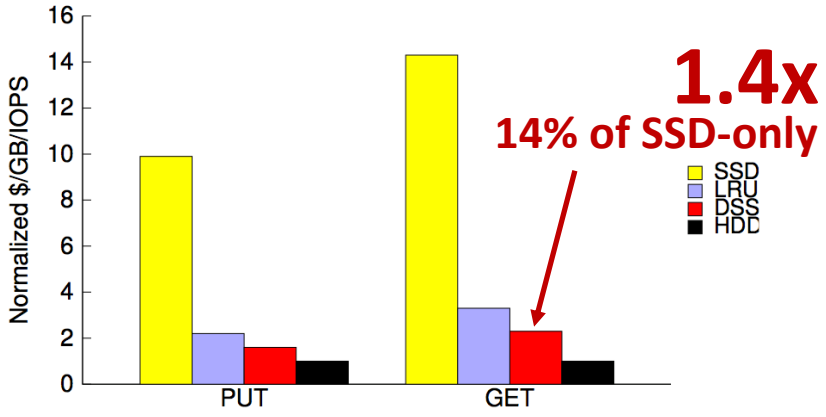
Uploading Bandwidth (MB/sec)



Downloading Bandwidth (MB/sec)



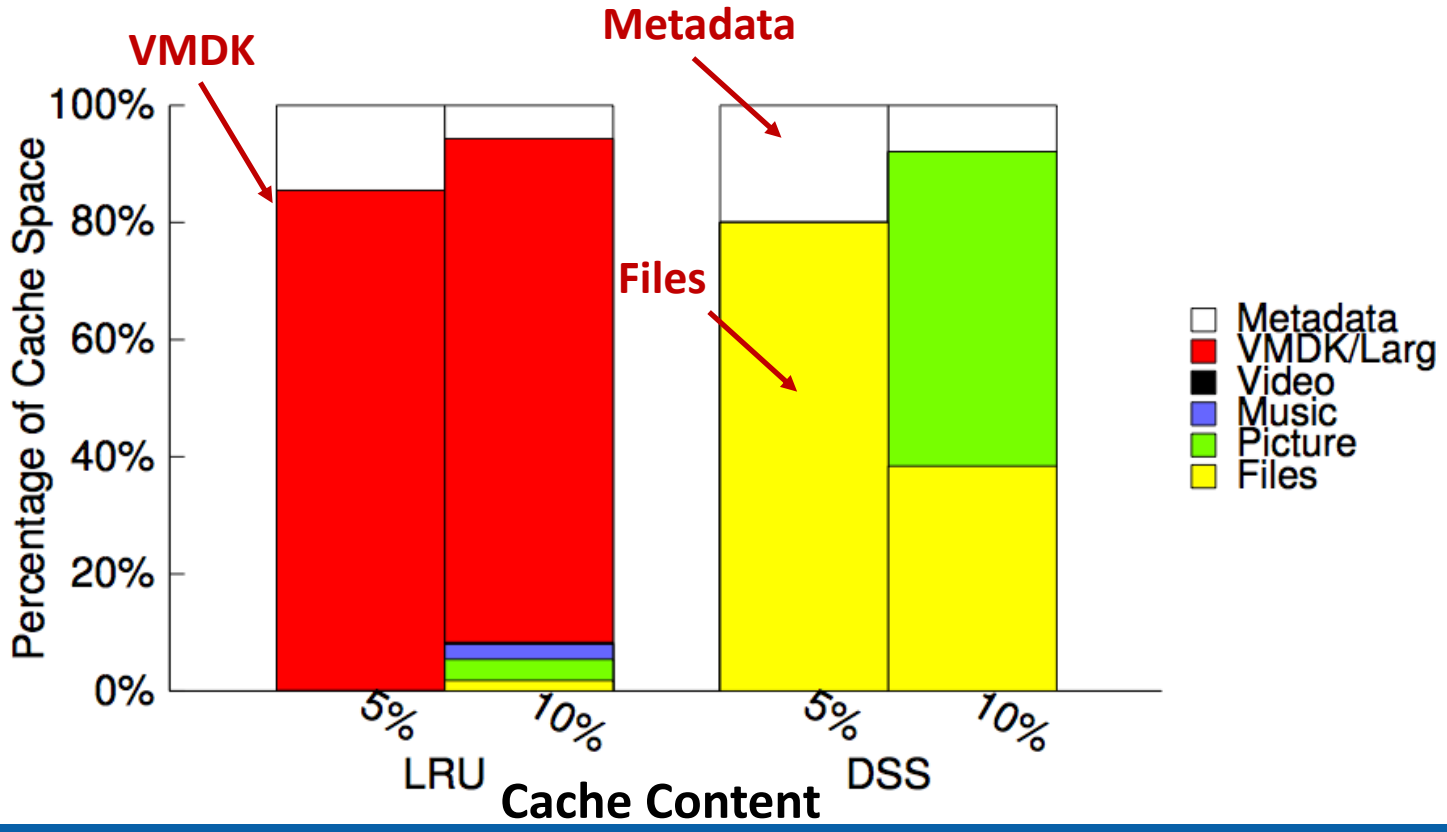
Downloading Latency (Seconds)



Cost Effiicency ($/GB/IOPS)

Explaining cache effects

- CACS selectively caches the most important data
  - LRU disregards the user-specified data “importance”



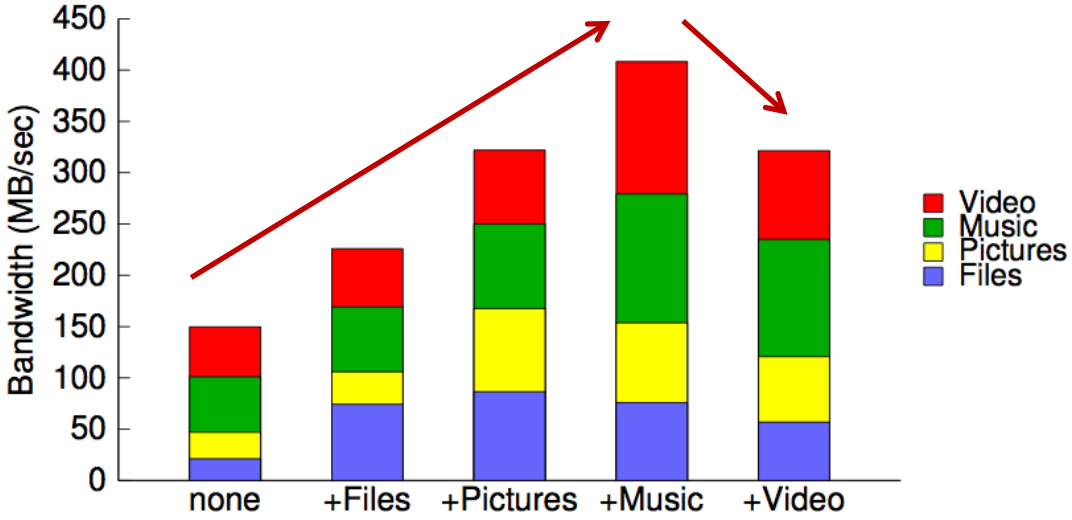
Case 2: Fine-grained traffic control

- **Traffic direction policy**
  - Each object type is given an individual class
  - Specified data objects will be directed to SSDs as needed
  - The un-specified data objects will be directed to HDDs (UNCLASSIFIED)
  - 25 parallel requests for each object type for uploading data
  - We do four runs, and each time we add one object type to the SSD

Object Size	Files (60%)	Picture (35%)	Music (4%)	Video (1%)
≤64KB	79%	17.1%	0%	0.3%
≤512KB	14%	43.6%	0%	2.8%
≤1MB	3%	14.5%	0.8%	4.9%
≤5MB	2%	20%	52.3%	32.4%
≤10MB	1%	3.5%	39.6%	31.7%
>10MB	0%	1.3%	7.3%	27.9%

Distribution and Classification

Bandwidth and throughput

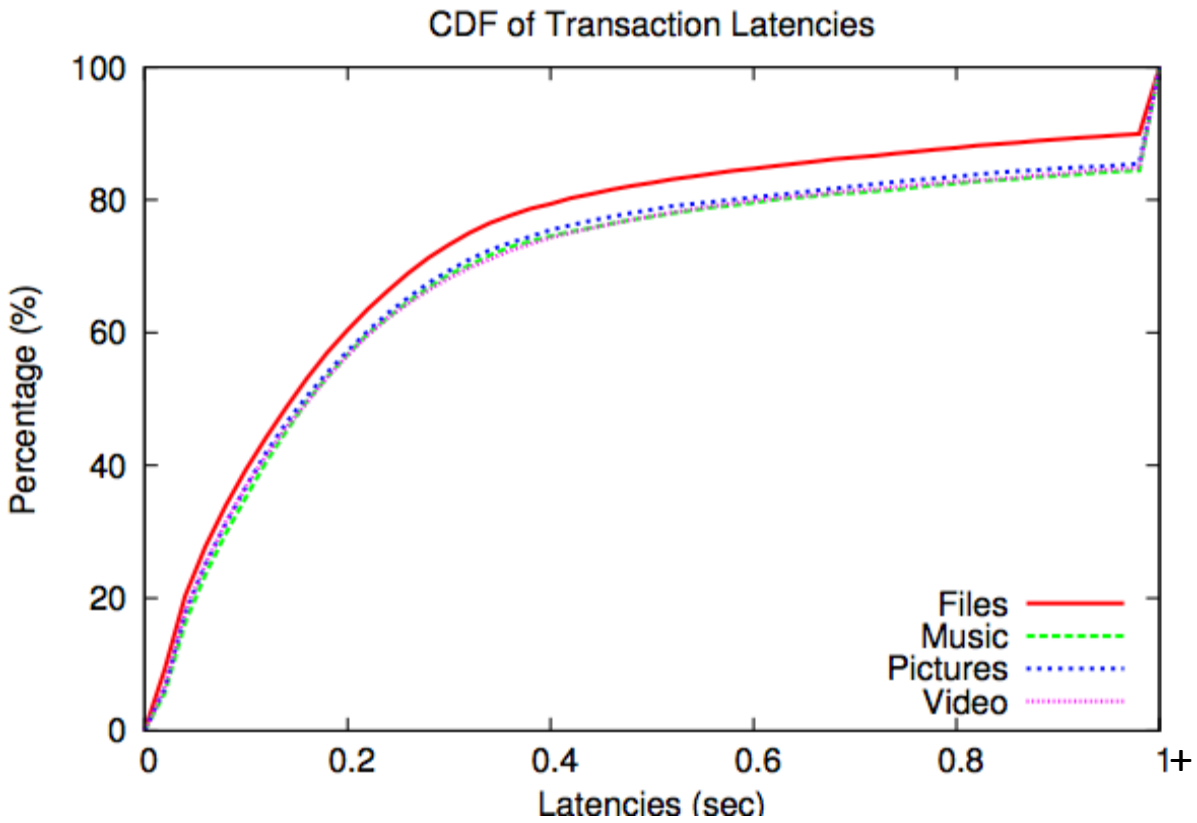


Directing traffic to SSD moves pressure for workloads on HDD

We also need to be careful – parallelizing device uses is important

	Speedup			
	Files	**Pic**	**Music**	**Video**
+Files	3.5x	1.2x	1.1x	1.1x
+Pictures	4x	3.1x	1.5x	1.5x
+Music	3.6x	3x	2.3x	2.6x
+Video	2.7x	2.4x	2.1x	1.8x

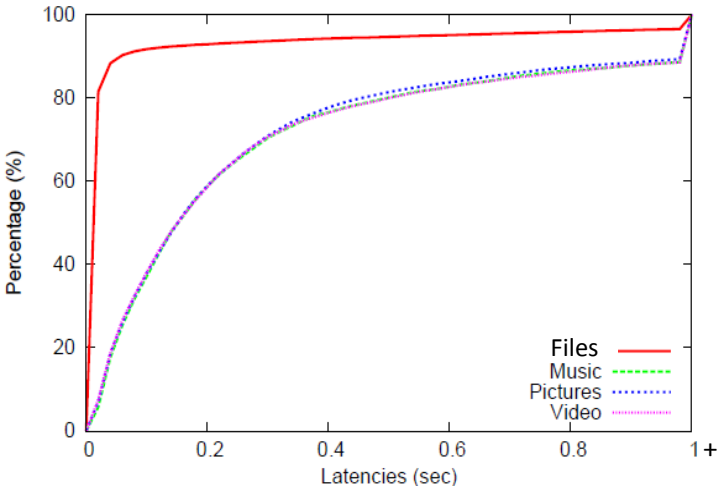
Distributions of latencies



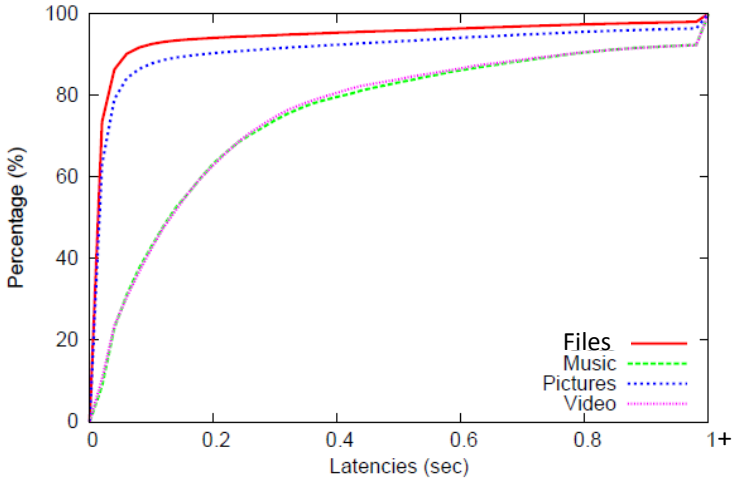
Before I/O redirection

Distributions of latencies

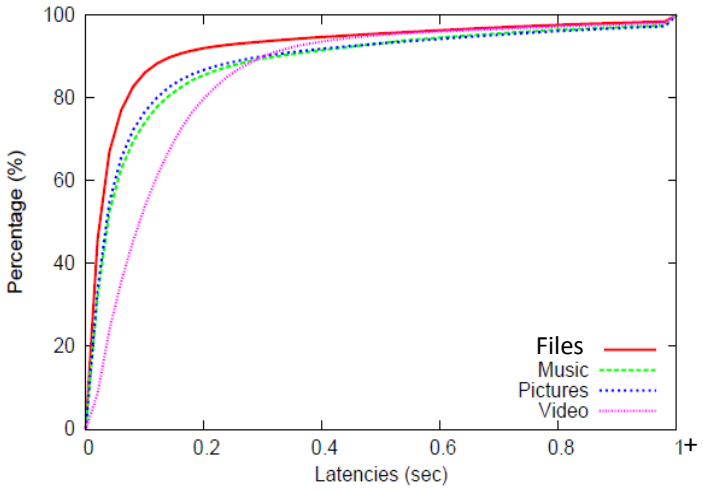
Files



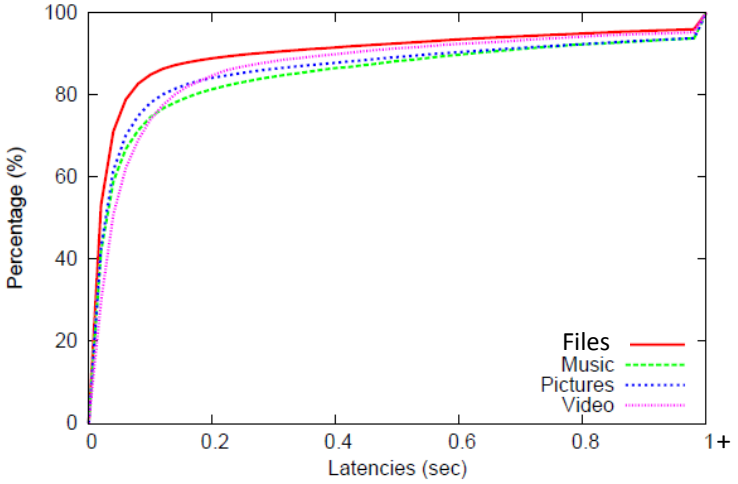
Files + Pictures



Files + Pictures + Music



Files + Pictures + Music + Video



Conclusion

- Cloud storage brings a critical QoS challenge
  - The widening semantic gap makes end-to-end QoS difficult to realize
- A client-aware cloud storage framework
  - Client classifies data and transmits data and semantic hints to server
  - Cloud storage server handles classified requests and generates I/Os
  - Storage system enforces QoS policy based on data classification
- Client-awareness can provide significant benefits
  - 5x bandwidth, 5.5x latency, 14% of the SSD cost
  - Other optimizations are also possible (e.g., reliability, security …)

Thank you!

fchen@csc.lsu.edu
michael.mesnier@intel.com
scott.hahn@intel.com