Oasis: An Active Storage Framework for Object Storage Platform

Yulai Xie[1], Dan Feng[1], Darrell D. E. Long[2] , Yan Li[2]

[1]School of Computer, Huazhong University of Science and Technology
Wuhan National Laboratory for Optoelectronics
[2]University of California, Santa Cruz

Outline

- Background and Motivation
- Oasis Design and Implementation
- Evaluation
- Conclusions and Future Work

Background and Motivation

◆Existing problem:

● Data center and clouds are networked architecture that is constructed via interconnecting a large number of servers, it’s critical to avoid network bottleneck on the interconnect incurred by big data transfer so as to provide real-time service to users.

◆ Solutions:

● Active storage: offloading computation from host to storage device to reduce bandwidth requirement

Background and Motivation

◆Faced Challenges

- Transparent and multi-granularity processing
- Security
- Resource contention

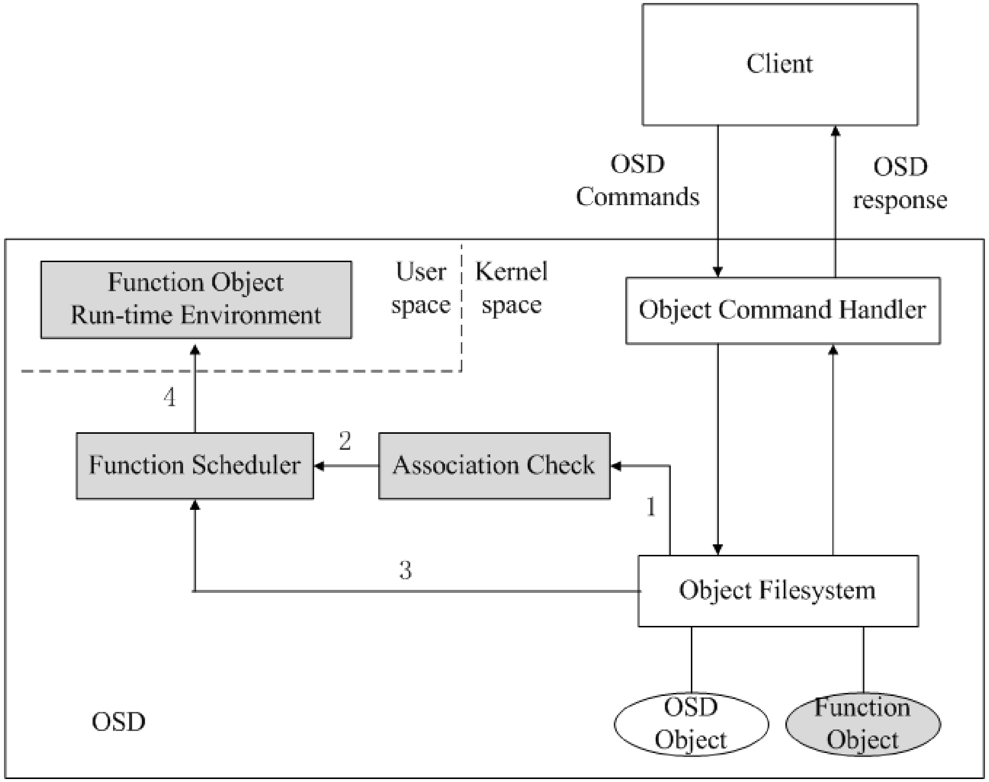
◆An example: offload erasure coding to storage device

- How to automatically execute encoding and apply it on a vast amount of data?
- How to ensure the encoding authorized by the legal user?
- How to dynamically apply encoding on the client and storage device according to the workload status?

Our proposal and contributions

- Propose Oasis, **An active storage framework for object storage platform that leverages the OSD's processing power to run data intensive applications**
- Frees users from needing to remember the details of offloaded computation and use signature scheme and access control to ensure the security of execution .
- Monitor system resources and partition application computations between host and OSD dynamically
- Extensive experiments evaluate the performance, scalability and implementation overhead of Oasis on three typical real-world applications: database selection, blowfish decryption and edge detection.

Oasis Architecture Overview

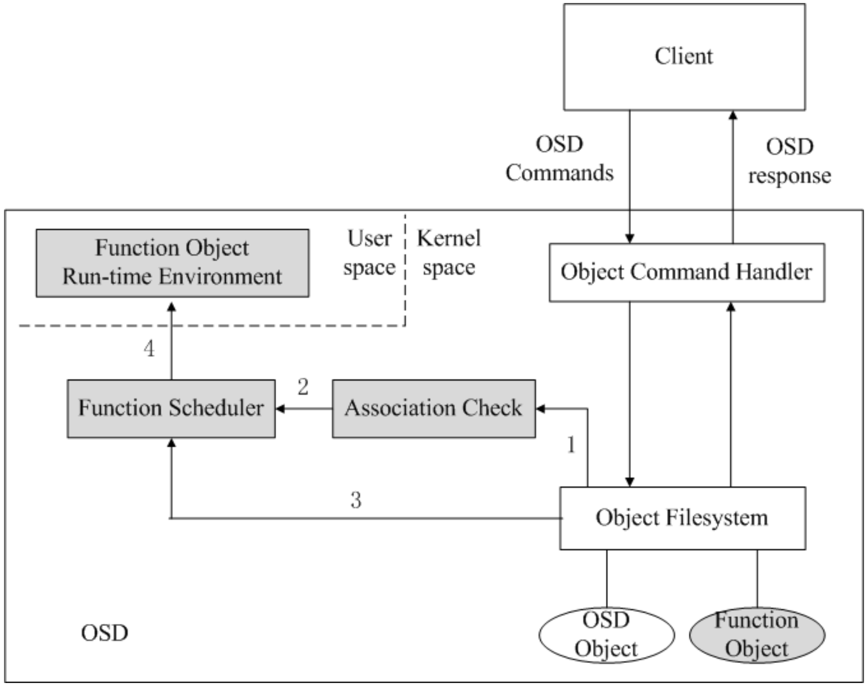


- Object Command Handler
  - Get and analyze OSD command
- Object Filesystem
  - Manage various objects
- Association Check
  - check whether any function objects are associated with an OSD object
- Function Scheduler
  - schedule function object to execute
- Function Object
  - Used to hold the offloaded application function (e.g., compression、encryption、encoding, etc)
  - A piece of code that can be executed in OSD to perform operations on certain user objects

How we use this system?
--------Example 1: Decryption

- If a user want to read an encrypted file from an OSD, what should he do?

- ◆ Create a function object that represent Decryption application in the OSD
- ◆ Associate this function object with an OSD object
- ◆ Send a READ command to the OSD object
- ◆ Then the associated function object that represents decryption will be scheduled to execute.

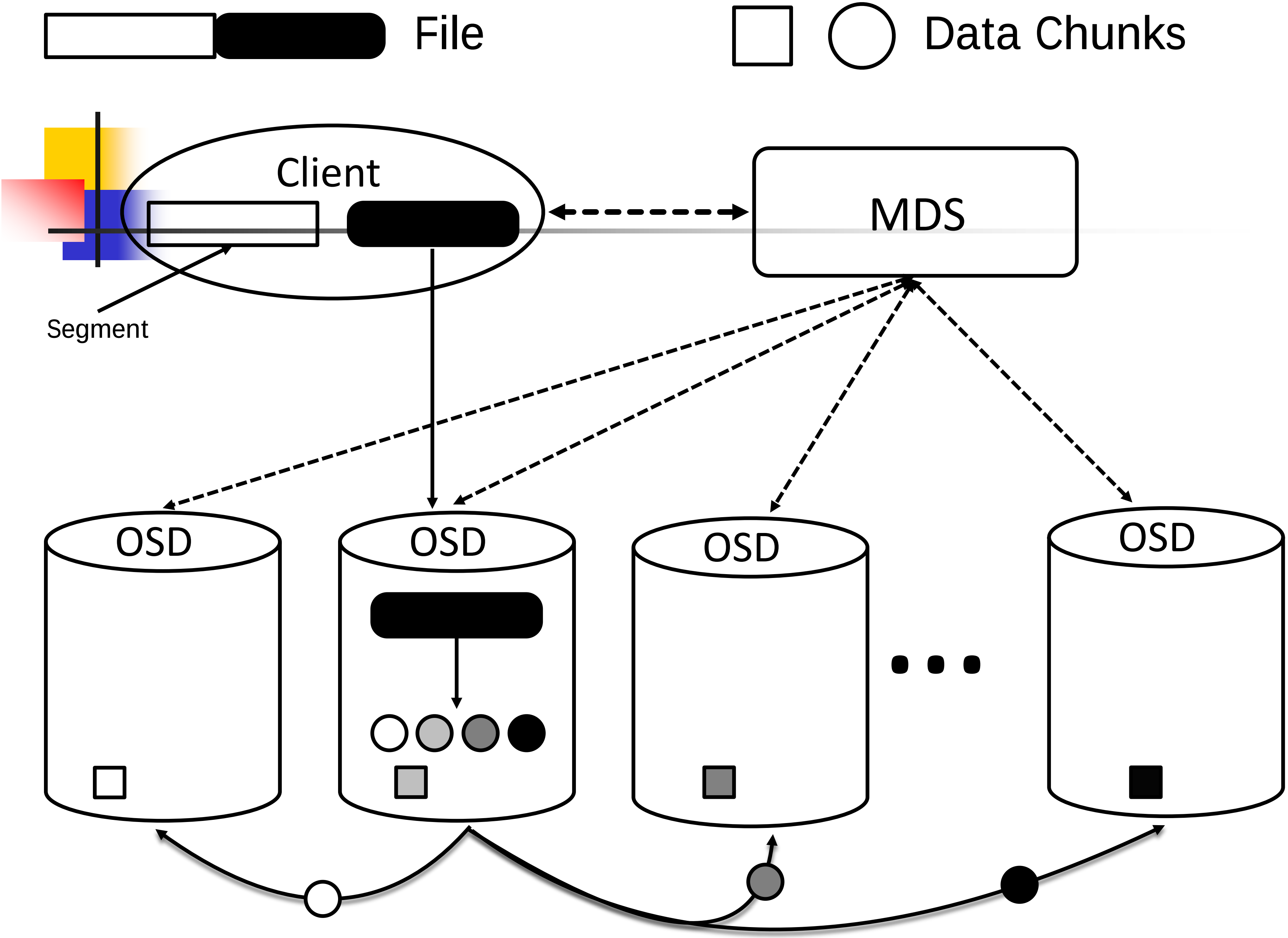


How we use this system?
--------Example 2: Encoding

● If a user want to write a file into OSDs and then encode it using erasure coding technology, what should he do?

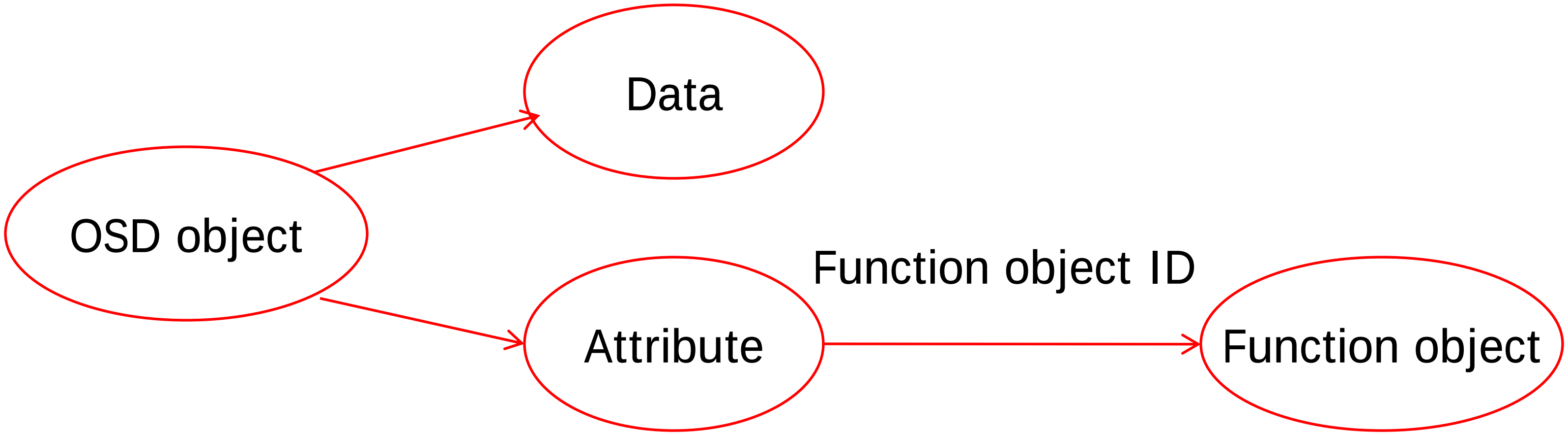
- ◆ Create a function object that represent erasure coding algorithm in the OSD
- ◆ Associate this function object with the OSD object (or file) to write
- ◆ Write the OSD object using a CREATE AND WRITE command
- ◆ Then the associated function object that represent erasure coding algorithm will be scheduled to execute.

  (e.g., the OSD object will be first splitted into multiple data chunks and encoded into parity chunks, then these chunks can be distributed into different OSDs)

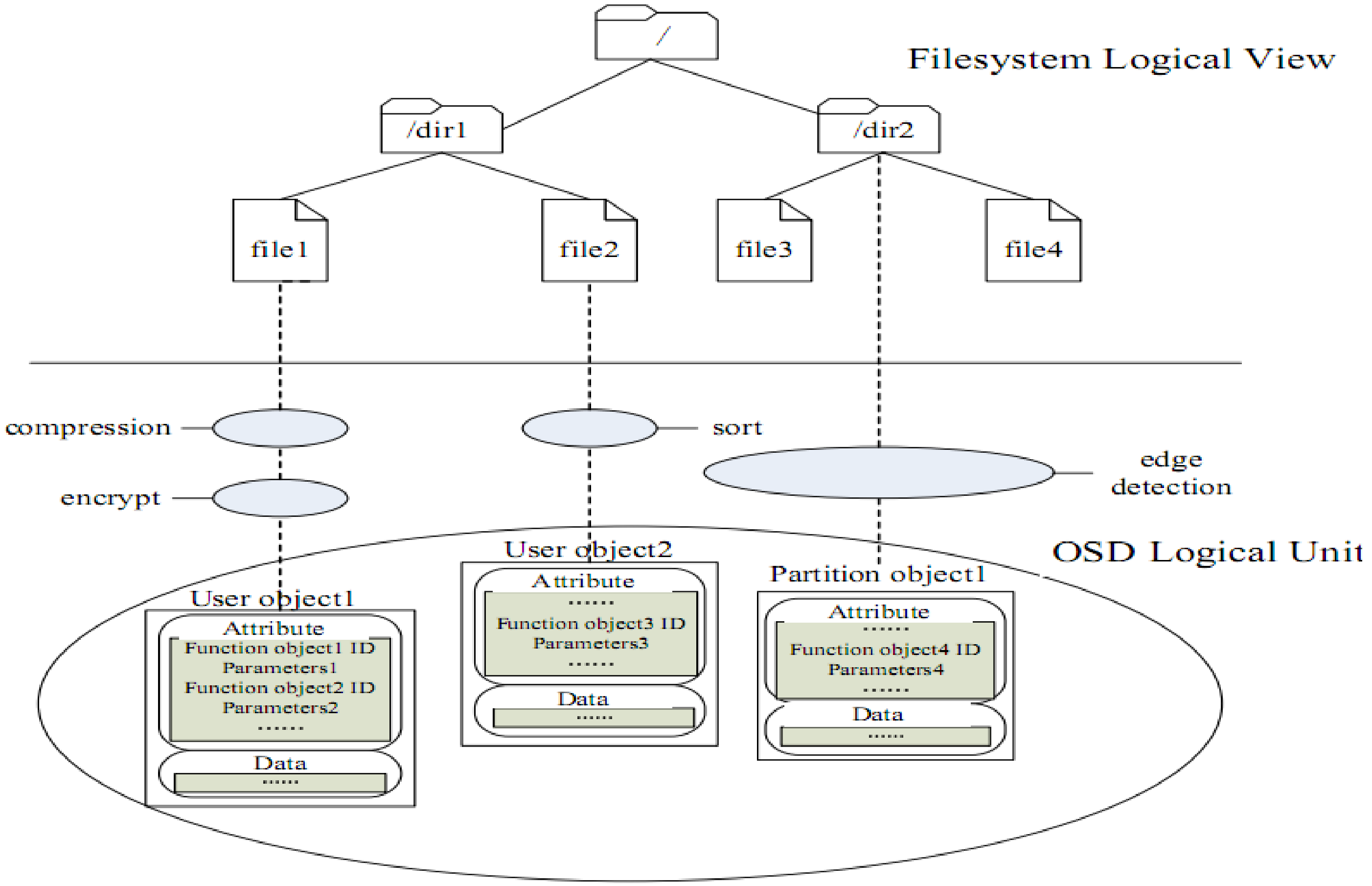

File
Data Chunks
Client
MDS
Segment
OSD
OSD
OSD
OSD

Critical characteristics for practical use:
-- Transparent and Multi-granularity processing

- Associate a function object with an OSD object
  - The function object will be invoked to execute during the read or write process.
  - We can flexibly apply different application function to different kinds of files.
  - Support different processing granularity.



An association example



Critical characteristics for practical use: -- Flexible and efficient management

- We use a separate partition to store function object

- We use OSD commands that manage OSD objects to manage function objects by specifying the partition ID that holds the function objects.

  - Download a function object to the storage device.
    ------ CREATE AND WRITE command
  - Remove a function object from the storage device.
    ------ REMOVE command
  - Conveniently view which function objects are there in the storage device.
    ------ LIST command
  - A user can know which function objects are associated with an OSD object.
    ------ GET ATTRIBUTES command

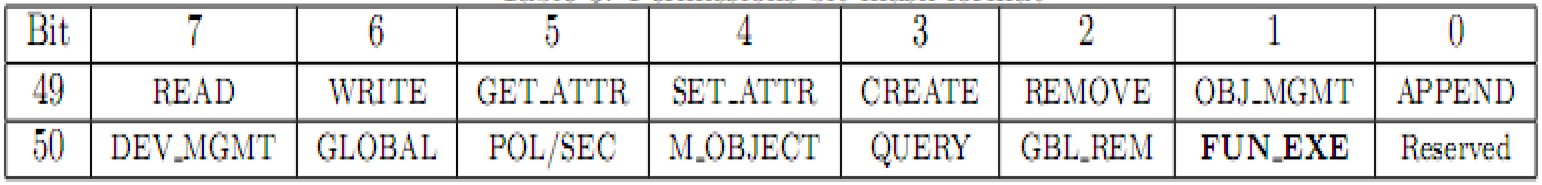
Critical characteristics for practical use: -- Security consideration

- Function object should be developed by vendors
  - The vendor has professional knowledge and tools to write and validate code.

- Administrator controls what function objects can run on the OSDs, and can allow a signed function object by installing the vendor or user's certificates.

- We add a permission bit called FUN_EXE into the capability to prevent unauthorized access
  - Two users may both have authority to set the attributes of an OSD object, but only the user that downloaded function object into OSD can invoke the function object to execute .

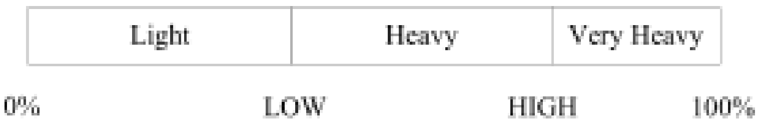
Table 3: Permissions bit mask format



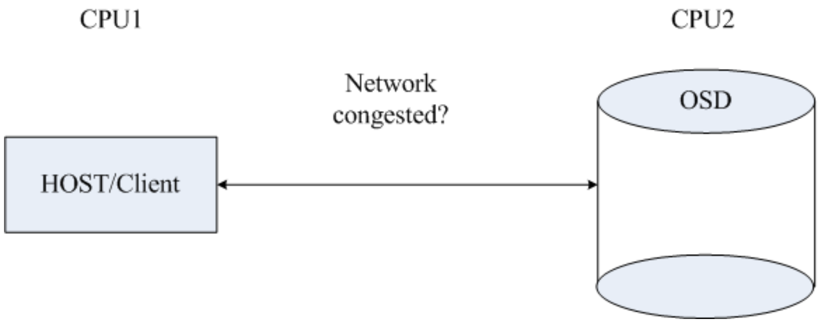
Bit	7	6	5	4	3	2	1	0
49	READ	WRITE	GET_ATTR	SET_ATTR	CREATE	REMOVE	OBJ_MGMT	APPEND
50	DEV_MGMT	GLOBAL	POL/SEC	M_OBJECT	QUERY	GBL_REM	**FUN_EXE**	Reserved

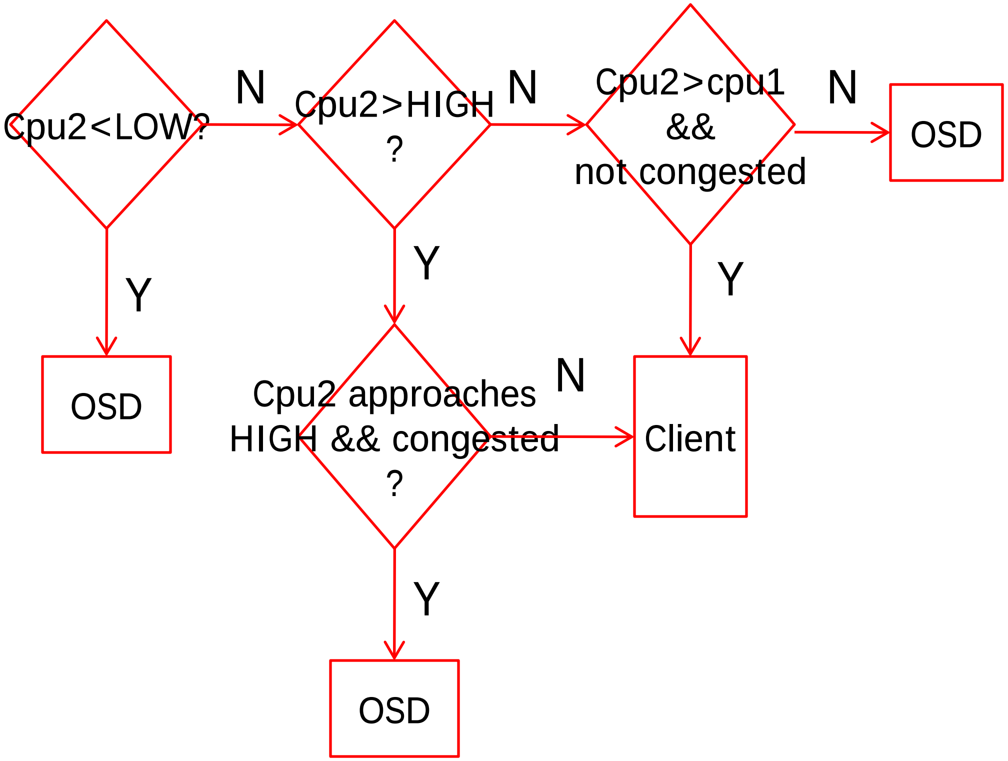
Critical characteristics for practical use: -- Adaptive Computation Partition

- Partition the application computation workload according to the CPU and network status



CPU utilization partition





Cpu2<LOW?
Y
OSD
N
Cpu2>HIGH ?
Y
Cpu2 approaches HIGH && congested ?
Y
OSD
N
Client
N
Cpu2>cpu1 && not congested
Y
Client
N
OSD

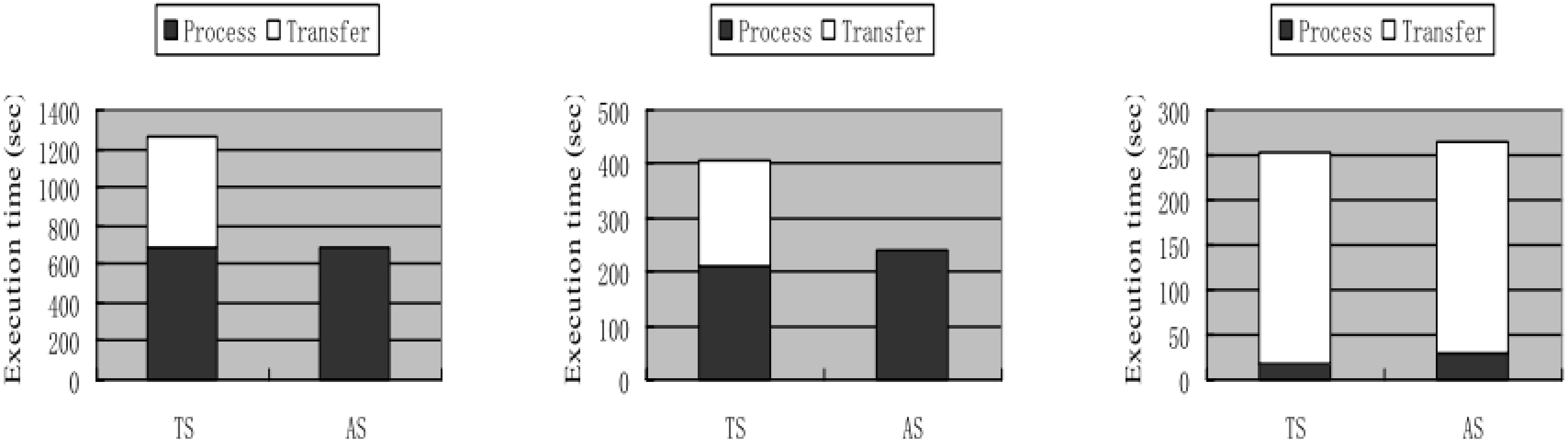
Evaluation

- Experimental setup
  - A host and 1, 2 or 4 OSDs, are connected via 1Gbps Ethernet. All machines run Redhat 2.4.20 to emulate the restricted execution environment of OSD.
  - Oasis is developed based on Intel OSD reference implementation (REFv20).

- Workload

Application	size of dataset	% of data filtering
Database Selection	1.77GB (33 million line records)	87.4%
Edge Detection	584MB(10000 images)	96.7%
Blowfish Decryption	800MB(100 million line records)	0

- Performance improvement

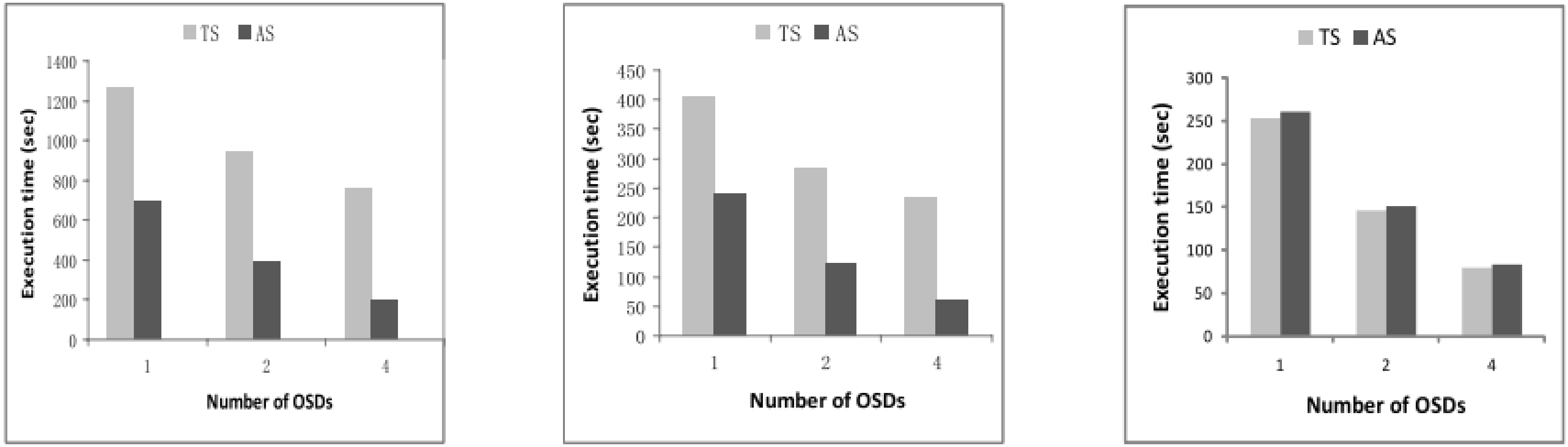


(a) Database Selection (b) Edge Detection (c) Blowfish Decryption

Fig. 4. Execution time breakdown for different applications with one OSD

- TS: Traditional Storage (run application in client)
- AS: Active Storage (run application in OSD)

● Scalability



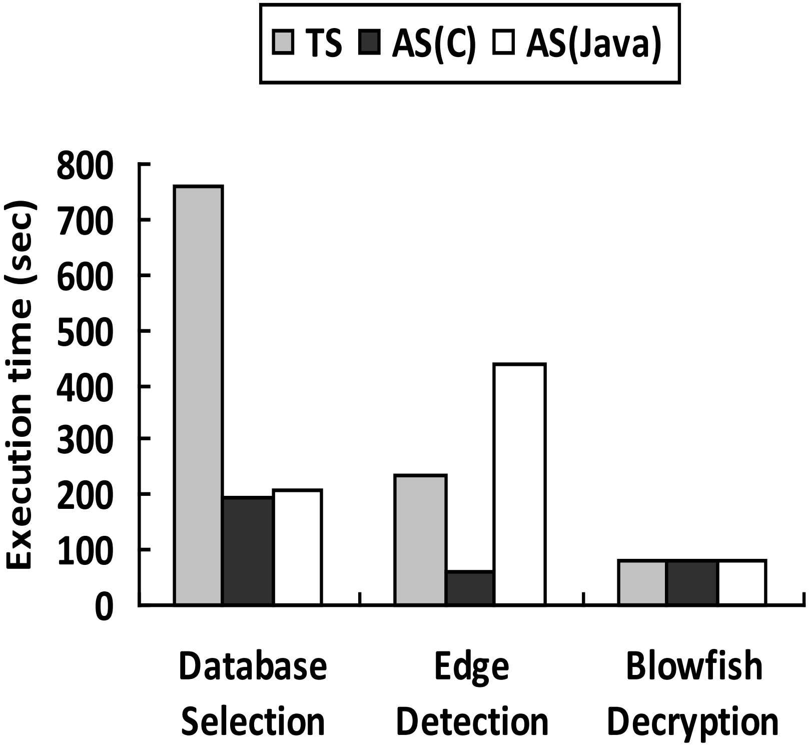
(a) Database Selection (b) Edge Detection (c) Blowfish Decryption

Fig. 5. Execution time for different applications with different number of OSDs

- The performance of AS and TS are both consistent with the increase in the OSDs.
- TS and AS are comparable in the Blowfish Decryption as no data reduction exists in this application.

Impact of language of function objects

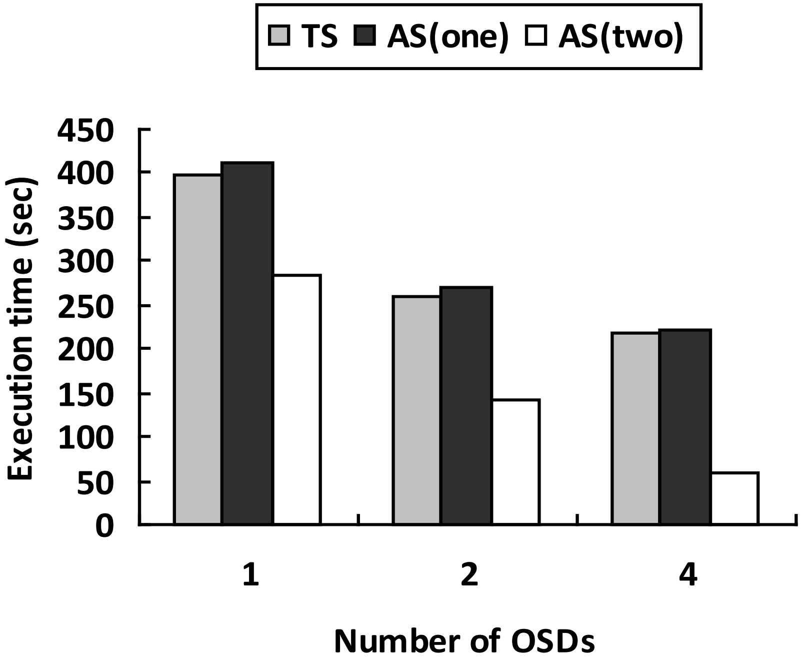
- A large number of I/O operations are required for the Edge Detection algorithm to generate the output image.
- Edge Detection algorithm implementation using the Java language is significantly slower than the implementation using the C language.
- Even such performance degradation with the Java implementation may compromise the benefits of data reduction in the Edge Detection application achieved by the active storage technology.



- **I/O intensive application would incur a performance bottleneck with Java implementation.**

Impact of multiple function objects

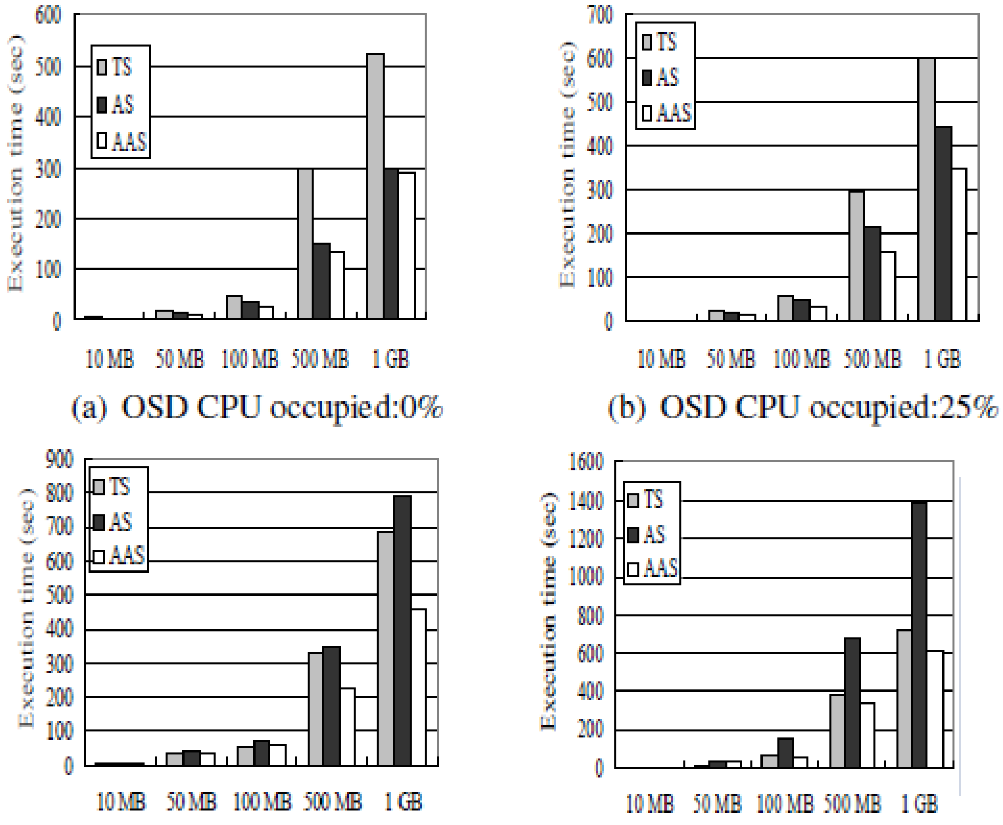
- For a hybrid application that is composed of multiple applications, only applications that can make data reduction across the I/O interconnect can really benefit system performance.



AS(one): first decryption on OSD, then selection on host

AS(two): process both decryption and selection applications on OSDs

● Performance with Adaptive Computation Partition



(a) OSD CPU occupied:0%

(b) OSD CPU occupied:25%

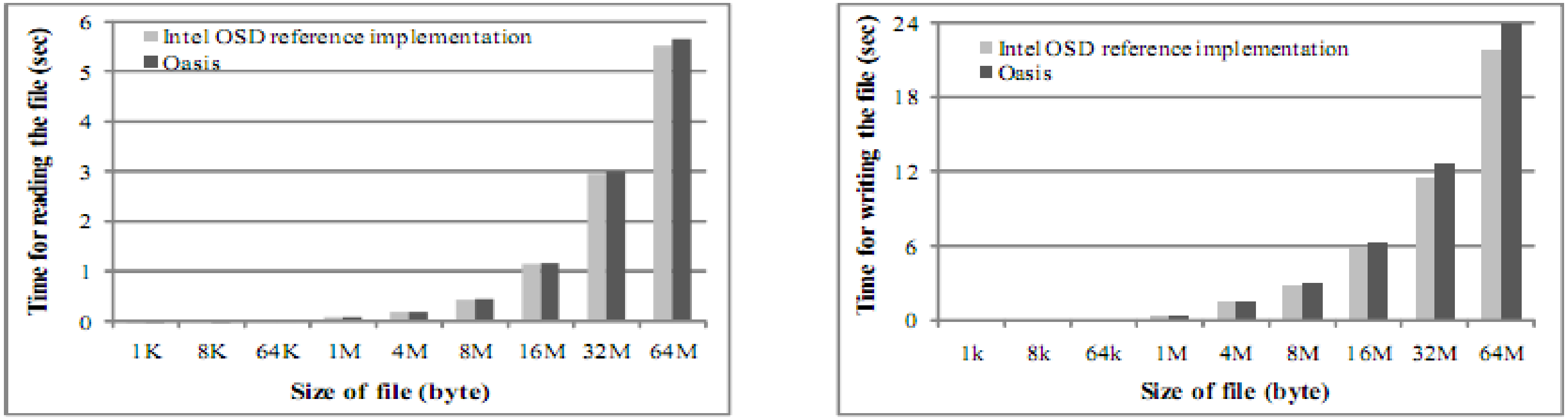
(c) OSD CPU occupied:50%

(d) OSD CPU occupied:75%

- TS: Traditional Storage
- AS: Active Storage
- AAS: Adaptive Active Storage

- AAS has better performance than TS and AS.

Implementation overhead

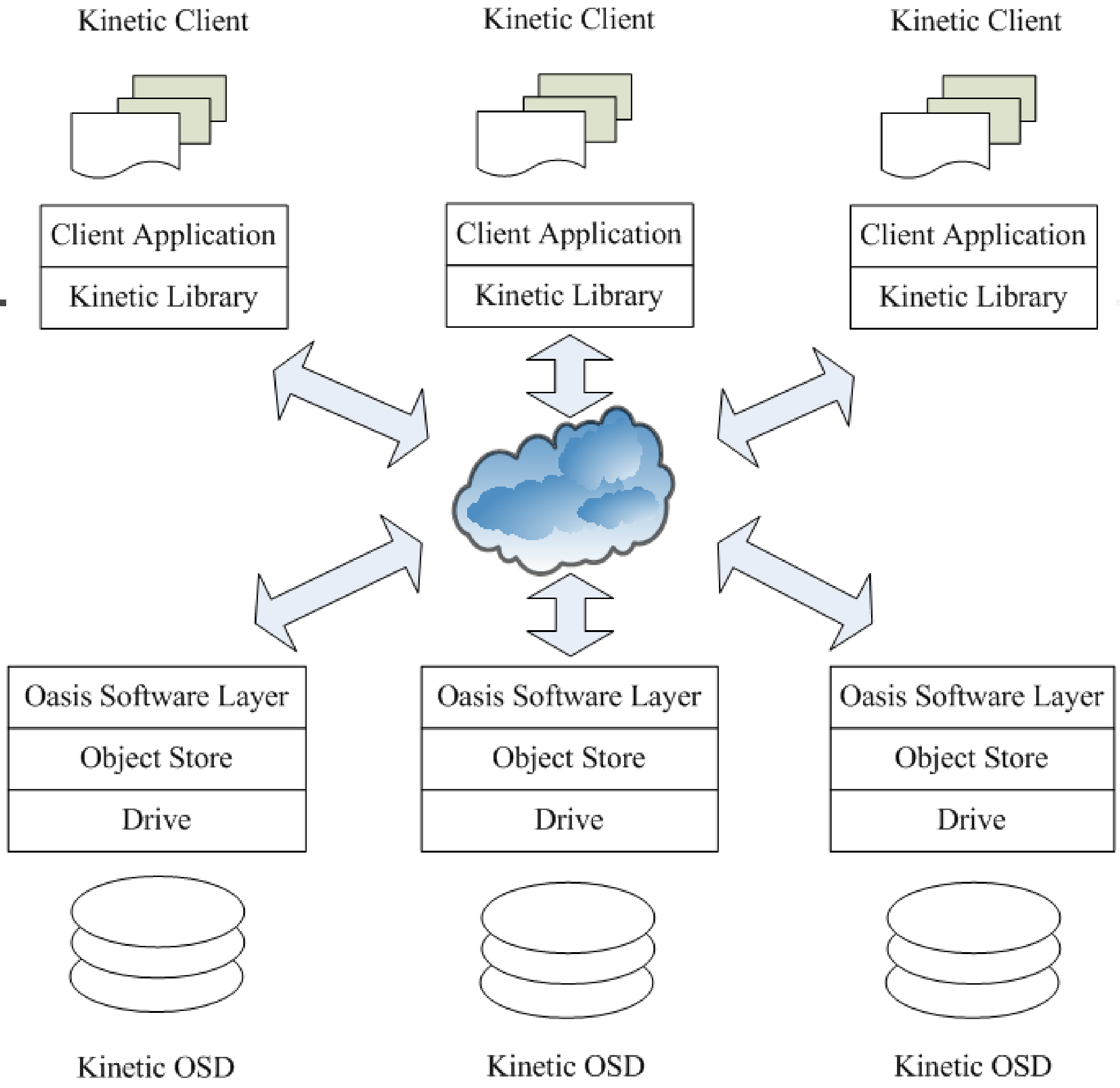


(a) read overhead

(b) write overhead

Figure 8: Implementation overhead of Oasis over the Intel OSD reference implementation

- During every Read or Write, the system has to check whether there exists any function object associated with the OSD object that is being read or written.
- The overhead is small, 1.2%-5.9% for read and 0.6%-9.9% for write.



Integrating Oasis with Seagate Kinetic Object Storage Platform

Conclusions and Future work

- Oasis: an active storage framework for object storage platform
  - Four kinds of critical characteristics in terms of user case

    1) Transparent and multiple granularity processing,
    2) flexible management,
    3) preliminary security,
    4) adaptive computation partition
  - System demonstration on three real world applications in terms of performance, scalability, language, etc.

- Future work
  - Concurrent execution of multiple function objects by employing sandbox technology
  - Evaluation on big data center and cloud storage

Thanks!