CS353 Linux Kernel

Chentao Wu吴晨涛
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

7C. Advanced File System Technology (Object-based Storage)

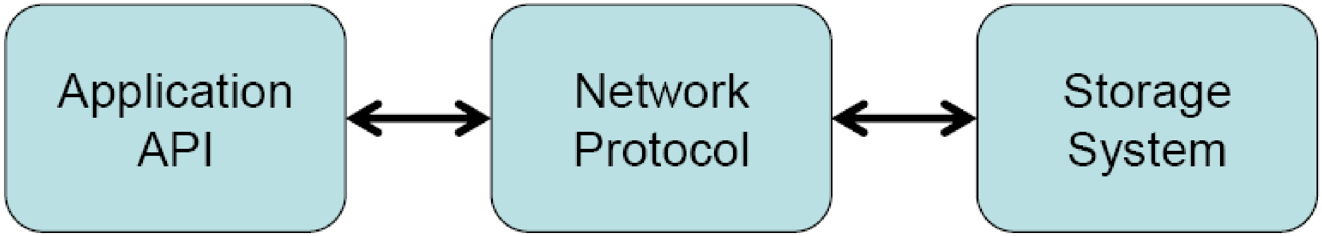
Chentao Wu
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

上海交通大學

Object Storage Technology

Agenda

- Object storage background and history
  - Abstract data containers
  - NASD, OSD, HTTP
- Objects and File Systems
  - Lustre, PanFS, Ceph, many others
- Objects and Web Storage
  - S3, Azure, Swift, many others



Two Paths to Objects

Storage Devices

- Move smarts into the device (NASD)
  - Network Attached Secure Disk
- Raise the level of abstraction
  - Containers
  - Attributes
  - Security
- SCSI Model
  - OSD command set

Web Services

- Add storage abstraction to a web-based system
  - Containers
  - Metadata
  - Security
- REST Model
  - HTTP protocol

Some History

- NASD (Network Attached Secure Disk)
  - 1990's research by Dr. Garth Gibson about moving intelligence into the storage device
  - Google cites NASD as inspiration for data node in its file system
- OSD (Object-based Storage Device)
  - Standards effort in the early 2000's created a SCSI command set for objects
  - There is a storage <u>device</u> behind this interface
- HTTP (HyperText Transfer Protocol)
  - A simple put/get protocol for the world-wide web
  - There is an arbitrary <u>service</u> behind this interface

Object Storage

- Objects are containers for data and attributes
  - Every file system has an *inode* that is data blocks plus attributes
  - They are created, deleted, read, written, and have attributes

Block Based Device

Operations

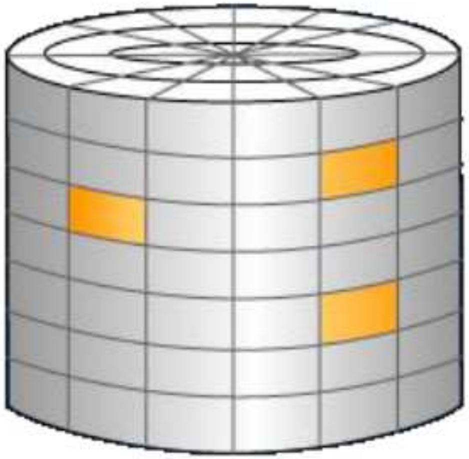
Read block
Write block

Addressing

Block range

Allocation

External



Object Based Device

Operations

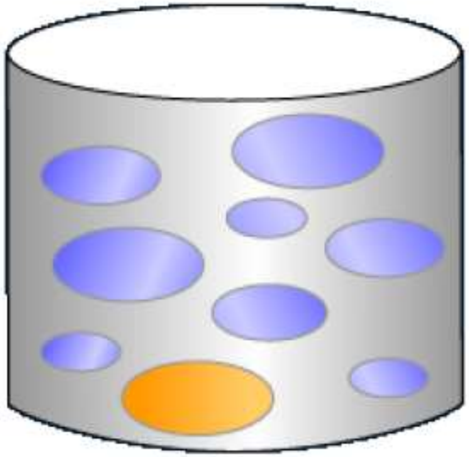
Create object
Delete object
Read object
Write object
Get Attribute
Set Attribute

Addressing

[object, byte range]

Allocation

Internal



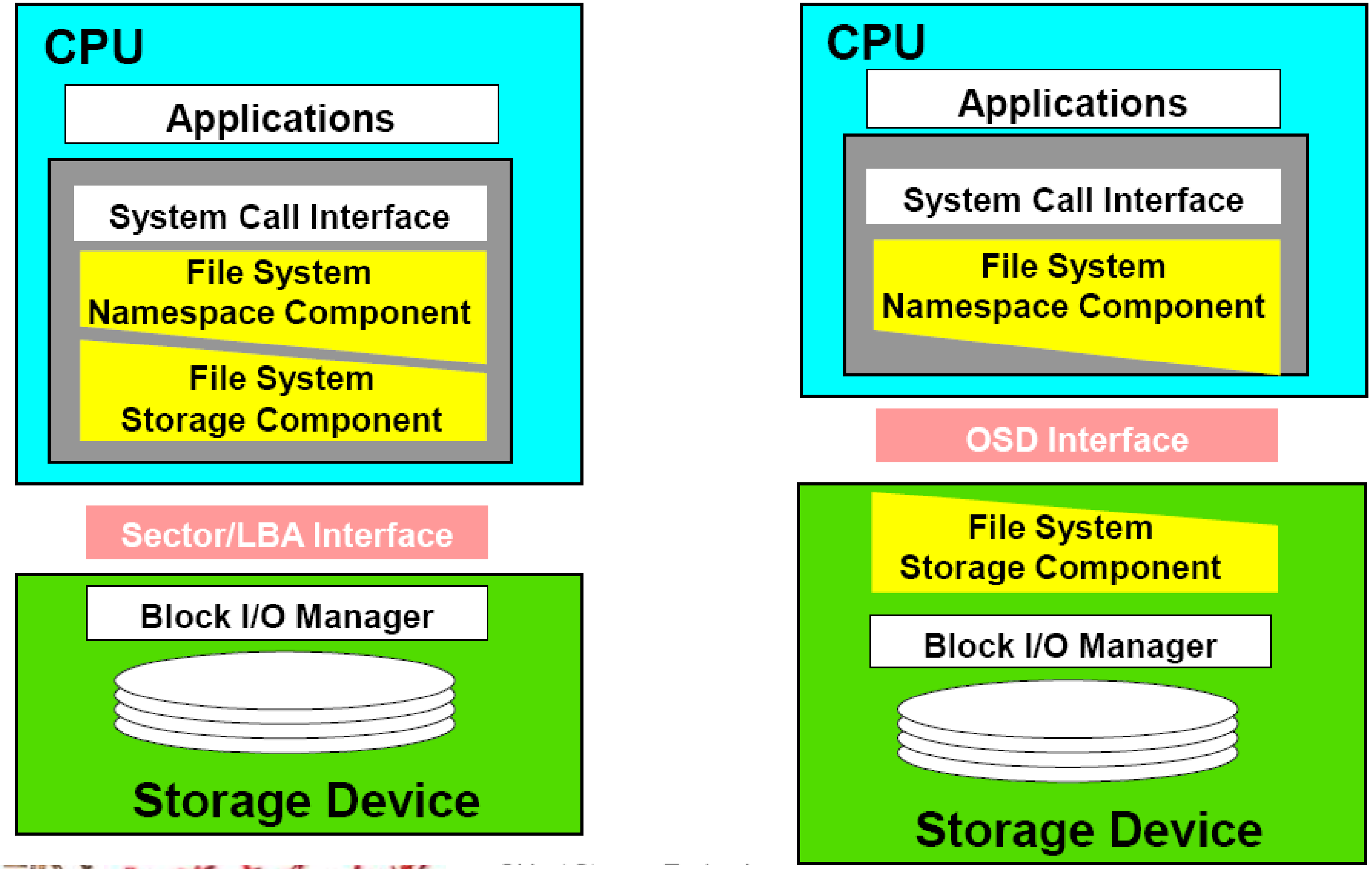
Object Storage Device (OSD)

- OSDs hold objects, which are like files in a simple file system
  - Objects are dynamically created and freed
  - Object are variable length
  - Objects have extensible attributes
  - Objects are identified by a 64 bit Object ID (OID)
  - Objects in an OSD are grouped within partitions, which are identified by a 64 bit Partition ID
    - 64 bit OID plus 64 bit PID gives a 128 bit namespace
- OSDs manage space allocation of Objects
  - A 4TB disk has 1 billion 4KB blocks – OSD hides this

OSD Standards

- There is a standard for OSDs under ANSI INCITS T10 (the SCSI specification)
  - *ANSI INCITS 458-2011*
  - OSD-1 is basic functionality
    - Read, write, create, delete objects and partitions
    - Security model, Capabilities, manage shared secrets and working keys
  - OSD-2 adds:
    - Snapshots
    - Collections of Objects
    - Extended exception handling and recovery

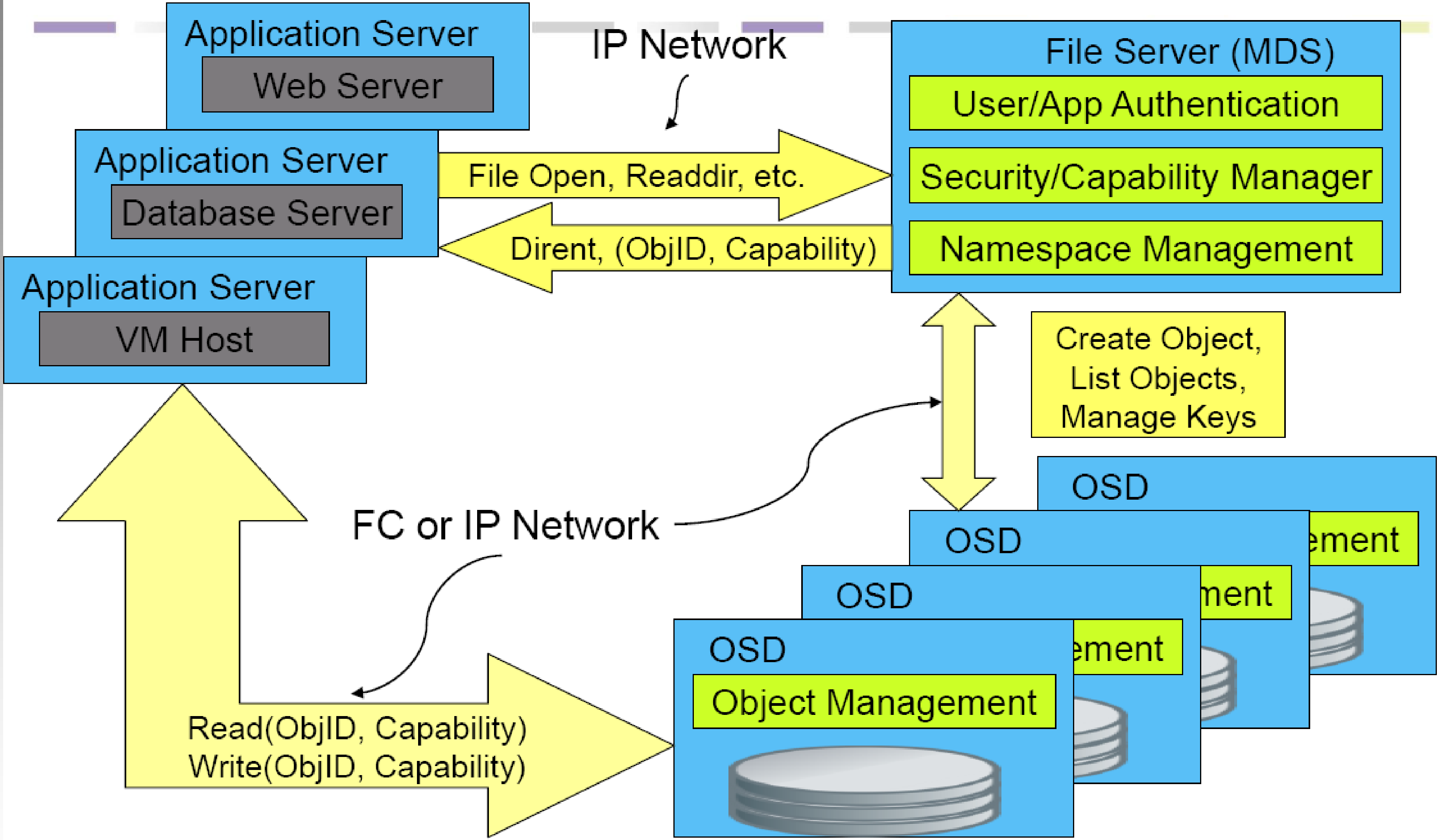
Objects and the File System Stack



Traditional File System

Object-based File System

Scaling OSD File Systems



Why Objects Help Scaling

- 90% of File System cycles are in the read/write path
  - Block allocation is expensive
  - Data transfer is expensive
  - OSD offloads both of these from the file server
  - Security model allows direct access from clients
- Higher level interfaces allow optimization
  - The more function behind an API, the less often you have to use the API to get your work done
- Higher level interfaces provide more semantics
  - User authentication and access control
  - Namespaces and indexing

OSD Capabilities

- Unlike disks, where access is granted on an all or nothing basis, OSDs grant or deny access to individual objects based on *Capabilities*
- A Capability must accompany each request to read or write an object
  - Capabilities are cryptographically signed by the Security Manager and verified (and enforced) by the OSD
  - A Capability to access an object is created by the Security Manager, and given to the client (application server) accessing the object
  - Capabilities can be revoked by changing an attribute on the object

OSD Security Model

- OSD and File Server know a secret key
  - Working keys are periodically generated from a master key
- File server authenticates clients and makes access control policy decisions
  - Access decision is captured in a capability that is signed with the secret key
  - Capability identifies object, expire time, allowed operations, etc.
- Client signs requests using the capability signature as a signing key
  - OSD verifies the signature before allowing access
  - OSD doesn’t know about users, ACLs, or whatever policy mechanism the File Server is using

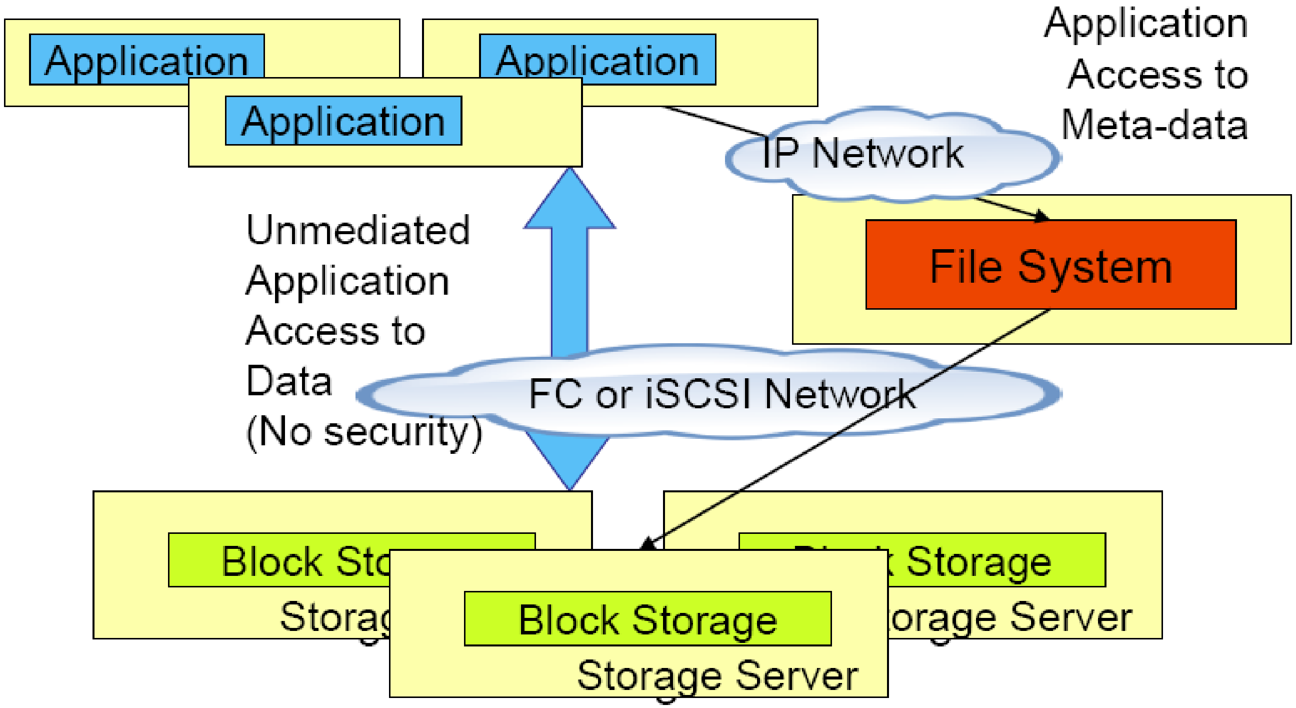
Object Storage File Systems

- Lustre
  - Custom OSS/OST model
  - Single metadata server
- PanFS
  - ANSI T10 OSD model
  - Multiple metadata servers
- Ceph
  - Custom OSD model
  - Crush metadata distribution
- pNFS
  - Out-of-band metadata service for NFSv4.1
  - T10 Objects, Files, Blocks as data services

- These systems scale
  - 1000's of disks (i.e., PB's)
  - 1000's of clients
  - 100's GB/sec
  - all in one file system

What about SAN File Systems?

- SAN file systems out-of-band metadata service
  - Security model does not support fine grain sharing
  - Block allocation must be managed by the file system
  - This can work well at small scale (many products)

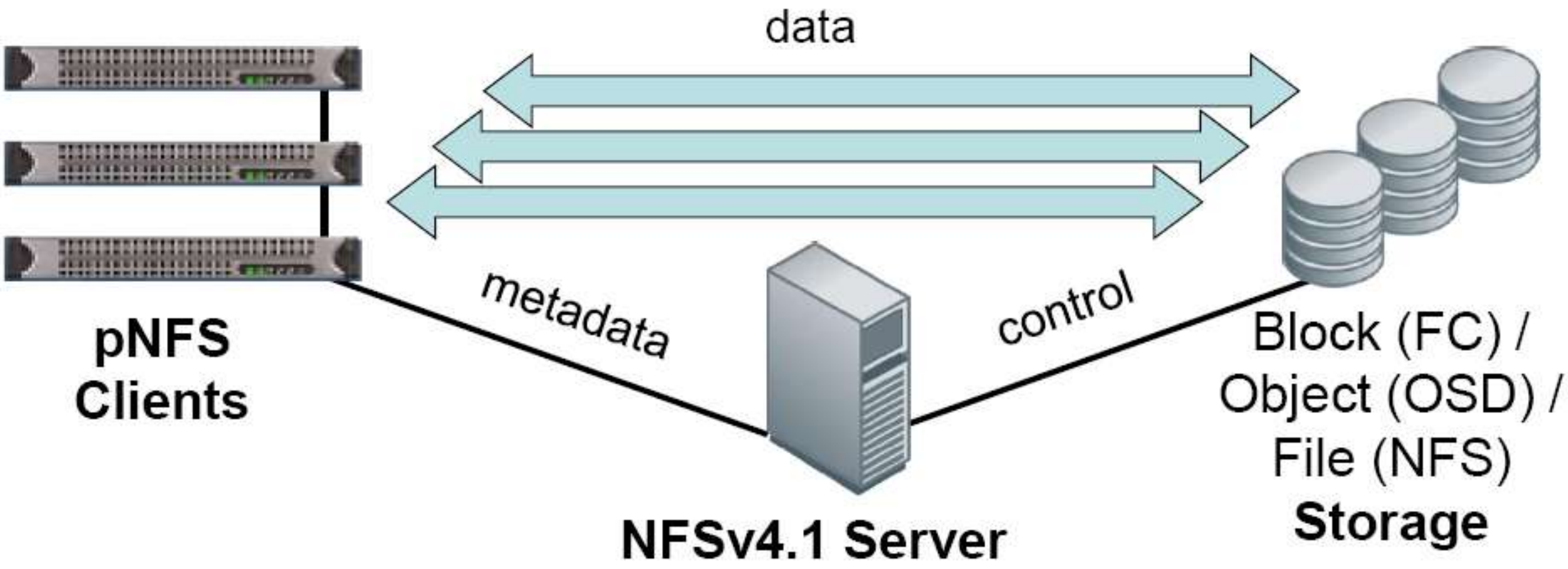


Why a Standard for Parallel I/O?

- NFS is the only network file system standard
  - IETF Standard
  - Proprietary file systems have unique advantages, but aren’t right for everyone
    - PanFS, Lustre, GPFS, IBRIX, CXFS, HDFS, etc.
- pNFS widens the playing field
  - Most major NFS vendors have announced pNFS support
  - Broader market benefits vendors
  - More competition benefits customers
- NFS standard effort very active with 4.2 and beyond

Parallel IO in the NFS Standard

- How do you get the NFS file server out of the data path?
  - Add *Layouts*, and introduce a level of indirection
  - Conceptually small addition to the standard ☺
- What is the benefit?
  - Better overall performance from a large storage system
    - Load balancing among storage devices
  - Standard client for high performance, large scale storage systems



pNFS Status

- RFCs for NFSv4.1, pNFS-objects, and pNFS-blocks published January 2010
  - RFC 5661 - Network File System (NFS) Version 4 Minor Version 1 Protocol
  - RFC 5662 - Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description
  - RFC 5663 - Parallel NFS (pNFS) Block/Volume Layout
  - RFC 5664 - Object-Based Parallel NFS (pNFS) Operations
- Linux 3.4 kernel (mid 2012) has all of the patches necessary for the pNFS client
  - RHEL 6.4 and SLES offer support via back-ports
  - Enterprise distros based on more modern kernels by end of 2013

Web Object Features

- REST-ful API (i.e., web-based)
- Security/Authentication tied to Billing
- Metadata capabilities
- Highly available
- Loosely consistent
- Data Storage
  - Blobs
  - Tables
  - Queues
- Other related APIs (compute, search, etc.)
  - Storage API is relatively simple in comparison

RESTful API

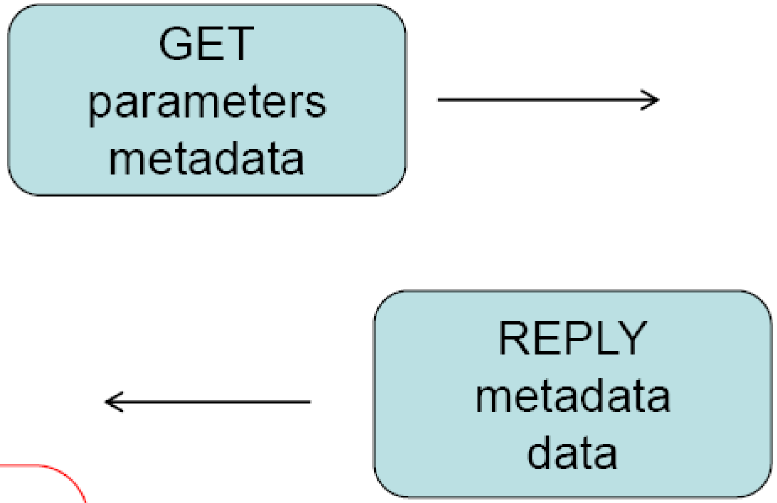
- Representational State Transfer (i.e., it’s the web)
  - Client-server
  - Stateless
    - Every request transmits session state (i.e., Cookies)
  - Cacheable (or not)
    - Explicit in the protocol
  - Layered
    - transparent proxies
  - Code on demand (optional)
    - javascript
  - Uniform
    - URI, Content-Type, MIME headers, …

Simple, flexible, text-based protocol that is easy to implement and extend.
It is just HTTP

Simple HTTP example

```
% telnet www.google.com 80
GET /index.html HTTP/1.0
(blank line)
HTTP/1.0 200 OK
Date: Wed, 13 Feb 2013 07:24:07 GMT
Content-Type: text/html; charset=ISO-8859-1

<html>
<head><title>Google</title></head>
<body><img src= /images/srpr/logo3w.png>
<form><input type=text name=q>
<input type=submit value=“Google Search” name=“search”>
<input type=submit value=“I’m Feeling Lucky” name=“lucky”>
</form></body></html>
```



GET
parameters
metadata

REPLY
metadata
data

HTTP and Objects

- Request specifies method and object:
  - Operation: GET, POST, PUT, HEAD, COPY
  - Object ID (/index.html)

This is a method call on an object

- Parameters use MIME format borrowed from email
  - Content-type: utf8;
  - Set-Cookie: tracking=1234567;

These are parameters

- And a data payload
  - Optional
  - Separated from parameters with a blank line (like email)

This is data

- Response has identical structure
  - Status line, key-value parameters, optional data payload

Security

- Shared Keys and Signatures
  - Shared secret is typical
  - Obtained from service provider
  - Two factor, or public key/private key also possible
- Signature computed over the request
  - GET/PUT line and some/all the request metadata
  - Signed with the secret using SHA256
  - Signature appears as metadata on the request

```
Authorization: SharedKey
myaccount:ctzMq410TV3wS7upTBcunJTDLEJwMAZuFPfr0mrrA08=
```

- Details handled by an SDK

Access Control

- Services allow setting up Access Control Lists on containers and objects
    - Accounts have one or more Idenities
    - Security token encodes account+identity
    - Publically readable objects are supported
- E.g., you can set up an image or content repository for your public web site, but restrict update/delete operations
    - Generalization of WebDav

OpenStack REST API for Storage

- `GET v1/account HTTP/1.1`
  - login to your account
- `HEAD v1/account HTTP/1.1`
  - List account metadata
- `PUT v1/account/container HTTP/1.1`
  - Create container
- `PUT v1/account/container/object HTTP/1.1`
  - Create object
- `GET v1/account/container/object HTTP/1.1`
  - Read object
- `HEAD v1/account/container/object HTTP/1.1`
  - Read object metadata

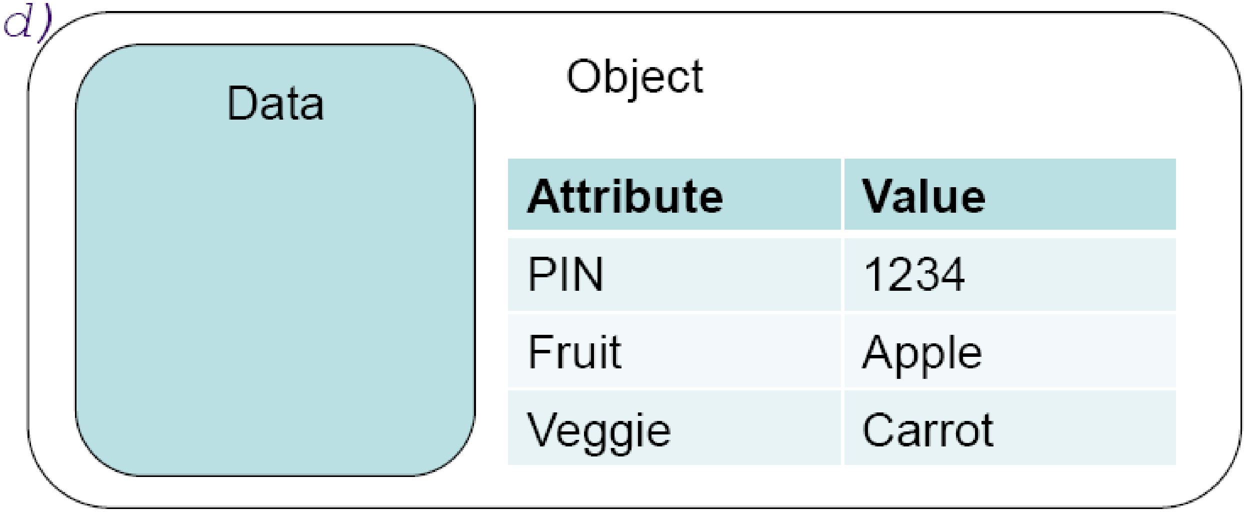
Creating an Object

```
PUT /v1/<account>/<container>/<object> HTTP/1.1
Host: storage.swiftdrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
ETag: 8a964ee2a5e88be344f36c22562a6486          MD5 checksum
Content-Length: 512000
X-Delete-At: 1339429105          Mon Jun 11 08:38:25 PDT 2012
Content-Disposition: attachment; filename=platmap.mp4
Content-Type: video/mp4
Content-Encoding: gzip
X-Object-Meta-PIN: 1234          User defined metadata
[ ...object content... ]
```

Updating metadata

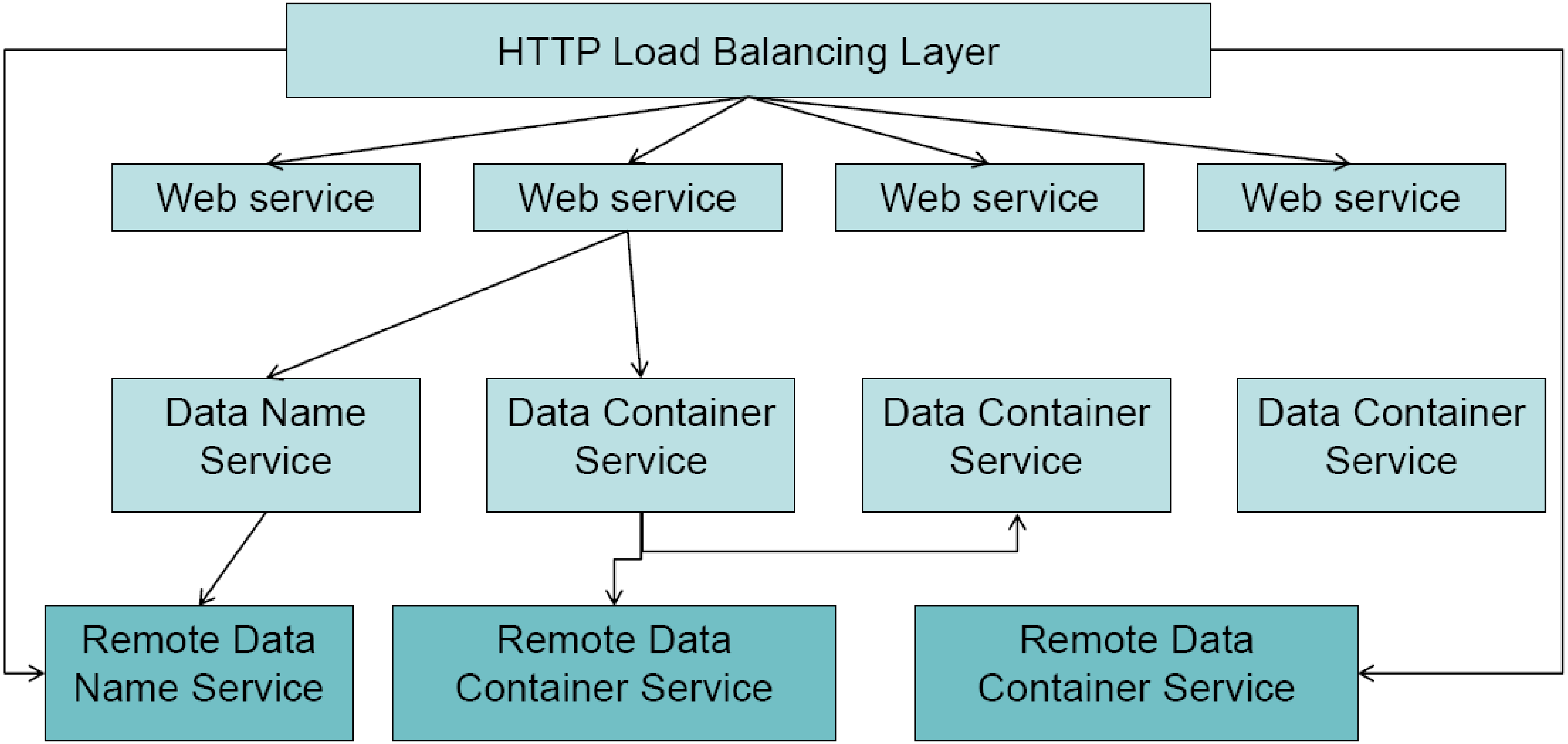
```
POST /v1/<account>/<container>/<object> HTTP/1.1
Host: storage.swiftdrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
X-Object-Meta-Fruit: Apple
X-Object-Meta-Veggie: Carrot
```

(no data payload)



Attribute	Value
PIN	1234
Fruit	Apple
Veggie	Carrot

Consistency and Availability



CAP Theorem

- Impossible to provide all of these simultaneously
  - **C**onsistency, **A**vailability, **P**artition Tolerance
- Replication across partitions increases availability
- The web access load balancer may direct requests to different partitions that may not be consistent
  - Partitions will become consistent eventually
- CAP Theorem =>
  Applications may see old versions of objects
  - Not see a newly created object
  - See a recently deleted object
  - Read the previous value of an object

Beyond Blobs

- Blob
  - Write or read the whole object (PUT / GET)
  - No in-place modification
  - Only metadata can be modified incrementally (POST)
- Great for images, static page content
  - Not so elegant for search
  - No well defined sub-structure

Tables

- Table has overall metadata and access control
  - Similar or same as blob
- NoSQL model
  - Key selects a row
    - Sometimes a “partition” results in a two-part key, and makes load balancing across table servers explicit
  - Rows can have dynamically changing columns
    - Add columns later as you decide what you need
    - Not every row may store a particular column
  - Element at row,column has a data type
- Queries search over rows within a single table
  - E.g., find rows where a column matches a filter
  - No “join” that combines different tables

Queue

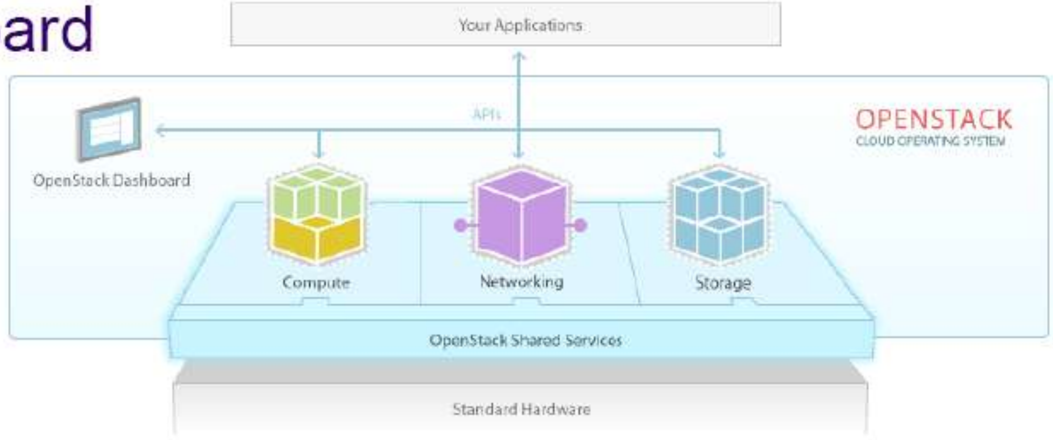
- Queue has metadata and access control
  - Similar or same as Blob
- PUT a message visible to other clients
- GET (and consume) a message
  - FIFO (First in, First Out)
- Messages limited in size (e.g., 8K or 64K)
  - Can store a reference to a larger Blob in the message
- Can Peek, Delete, and Clear messages

Web Object Platforms

- Here we give no endorsement implied or explicit about the relative merits of these cloud platforms
  - Amazon Web Services
  - Windows Azure
  - Apache OpenStack
  - SNIA CDMI (Cloud Data Management Interface)
- These are all similar with REST APIs
  - But plenty of differences among the offerings
- CDMI is a standard for storage and storage management REST API

OpenStack Cloud Operating System

- Open Source project part of the Apache project
- Compute (VM provisioning)
- Object Storage (in the cloud, or locally)
- Block Storage (volume provisioning)
- Networking (Software defined networks)
- Authentication (unified, multi-tenent)
- Management Dashboard



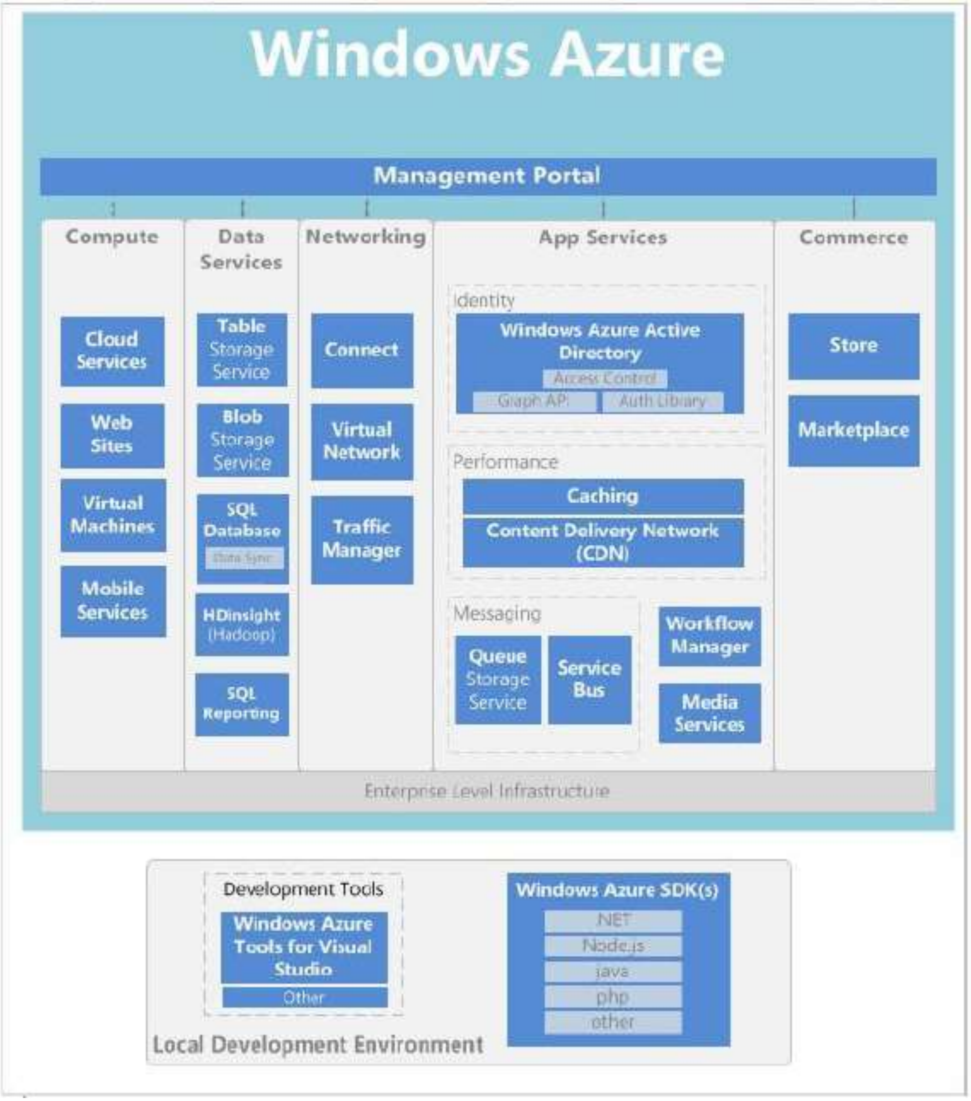
Amazon Web Services

- Compute (EC2)
- Content Delivery
- Database (Dynamo)
- Deployment
- Identity
- Queue Service
- Search
- Notifications
- Monitoring
- Network load balancing
- Billing
- Simple Storage Service (S3)
- Elastic Block Storage
- Jobs

amazon
web services

Microsoft Windows Azure

- Compute
- Data
  - blob, table, queue
- Networking
- Applications
- Commerce
- Dev Tools



CDMI

- ISO Standard in Oct 2012
- Cloud Data Management Interface
  - Client API for blob and queue storage abstractions
  - Management APIs for accounts and access control
  - Capability APIs to introspect on system capabilities
- REST APIs for these abstractions
  - Data
  - Container
  - Domain
  - Queue
  - Capability

Web Object APIs

- OpenStack
  - http://api.openstack.org/api-ref.html
- Amazon S3
  - http://aws.amazon.com/documentation/s3/
- Microsoft Windows Azure
  - http://msdn.microsoft.com/en-us/library/windowsazure/
- SNIA CDMI
  - http://snia.org/sites/default/files/CDMI%20v1.0.2.pdf

Web Object Summary

- There are several competing Cloud Operating Systems that each offer similar storage facilities
  - As well as several offerings (e.g., Cloud Foundry) that seek to provide a common API
- The storage is provided in the context of a broader set of services for virtual machine management and application development
  - This is an interesting counter point to POSIX, but it is still a world of de facto standards

Two Classes of Object Storage

Storage Devices

- Move smarts into the device (NASD)
- Raise the level of abstraction
  - Containers
  - Attributes
  - Security
- SCSI Model
  - OSD command set

Web Services

- Add storage abstraction to a web-based system
  - Containers
  - Metadata
  - Security
- REST Model
  - HTTP protocol

Project 4:
File System

上海交通大學

Source

- Inode.c/Makefile (kernel source of romfs)
- Test.img (a romfs image, you can mount it to a dir with ‘mount –o loop test.img xxx)
- Say test.img is mounted in t, ‘find t’ output
  - aa
  - bb
  - ft
  - fo
  - fo/aa

Practice 1

- Change romfs code to hide a file/dir with special name
- Test & result
  - insmod romfs hided_file_name="aa“
  - Mount –o loop test.img t
  - then ls t, ls t/fo, no "aa" and "fo/aa". found
  - ls t/aa, or ls fo/aa, no found
  - Without the code change, above two operations can find file ‘aa’

Practice 2

- change the code of romfs to correctly read info of an ‘encrypted’ romfs
- Test & result
    - insmod romfs hided_file_name=“bb“
    - Mount –o loop test.img t
    - Say bb’s original content is ‘bbbbbbb’
    - With the change, cat t/bb output ‘ccccccccc’

Practice 3

- change the code of romfs to add ‘x’ (execution) bit for a specific file
- Test & result
  - insmod romfs hided_file_name=“bb“
  - Mount –o loop test.img t
  - Without code changes ‘ls –l t’, output is ‘-rw-r--r--’
  - With the change, output is ‘-rwxr-xr-x’