INFSCI 2750: Cloud Computing

Lecture 6: Key Value Stores and Amazon AWS

Dr. Balaji Palanisamy
Associate Professor
School of Computing and Information
University of Pittsburgh
bpalan@pitt.edu

Slides Courtesy: Prof. Keke Chen, Wright State University
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007

Outline

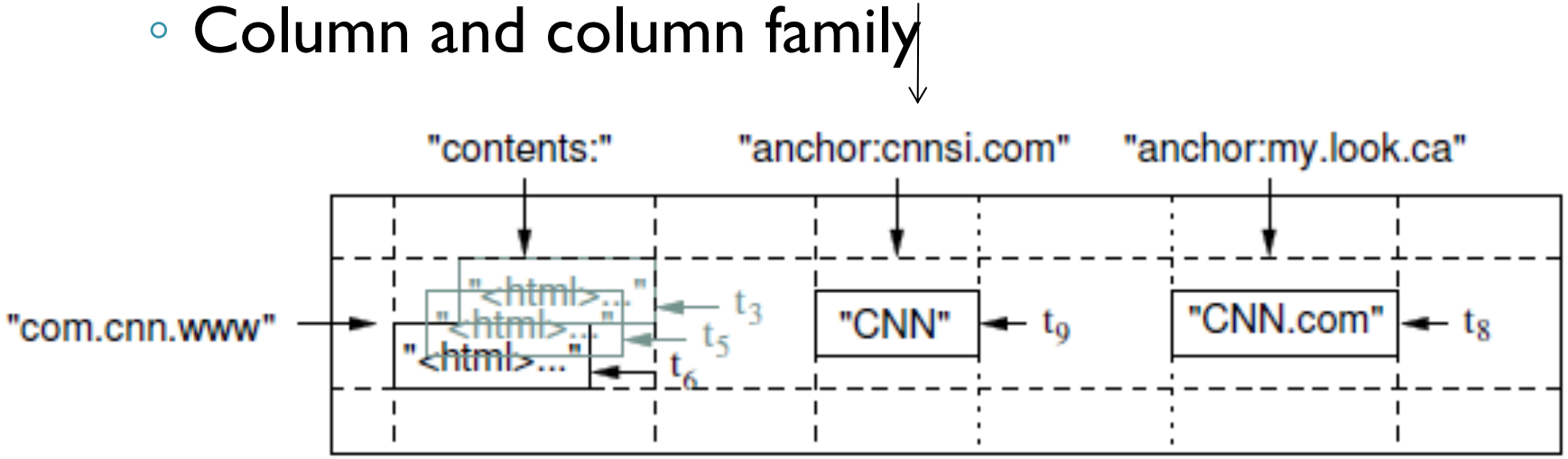
- KeyValue Stores
- Big Table
  - Frequent operations: append, squential read. Great scalability
  - GFS, HDFS
- Random Read/Write. Great scalability
  - BigTable (Hbase)
  - DynamoDB
  - Cassandra
  - PNUTS

BigTable

- Structured data
  - May generate from MapReduce processing
- Support random R/W

Data model

- Objects are arrays of bytes
- Objects are indexed by 3 dimensions
  - Row
  - Column and column family



"a big sparse table"

Data model

- Row
  - Row key: strings
  - Row range (groups of rows) – **tablet** (100-200M)
- Column
  - Grouped into "column families"
    - Column key "Family: qualifier"
  - Data physically organized by column-family – a embedded substructure of a record
  - Access controls are on column-family
- Timestamps
  - Representing versions of the same object

Operations

- API
  - Schema
    - Create, delete tables/column families
    - Access control
  - Data
    - Write, delete, lookup values
    - Iterate a subset
  - Single-row transaction
  - BigTable data can be input/output of mapreduce

Physical organization

- Built on top of GFS
- A tablet contains multiple SSTables
- SSTable: a block file with sparse index
  - Data (key, value) are stored in blocks
  - Block index
    - sorted keys,
    - Keys are partitioned to ranges
    - First Key of the range ->block(s)
  - Data Possibly organized in a nested way
- Use Chubby distributed lock service

Basic technical points

- Use index to speed up random R/W
  - Tablet servers maintain the indices
- Use locking service to solve the R/W conflicts → guarantees consistency

How to get tablet location?

- Through Chubby service

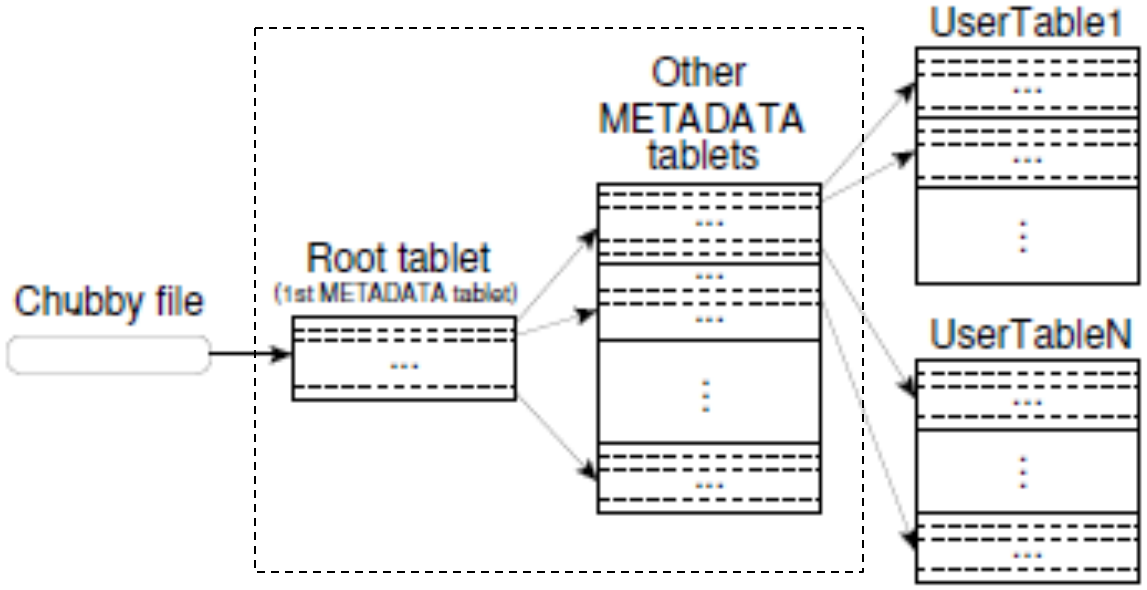


Figure 4: Tablet location hierarchy.

Chubby client
Caches locations

Metadata 128M, 1K per item ➔ 2^17 entries
Two levels ➔ 2^34 entries(tablets)

How it works

- Chubby service handles tablet locking
- Master maintains consistency
  - Start up by reading Metadata from Chubby
  - Check lock status
- insert/delete/split/merge tablets
- Tablet operations
  - Each tablet contains multiple SSTables (info stored in Metadata)
  - Use log to maintain consistency

HBASE

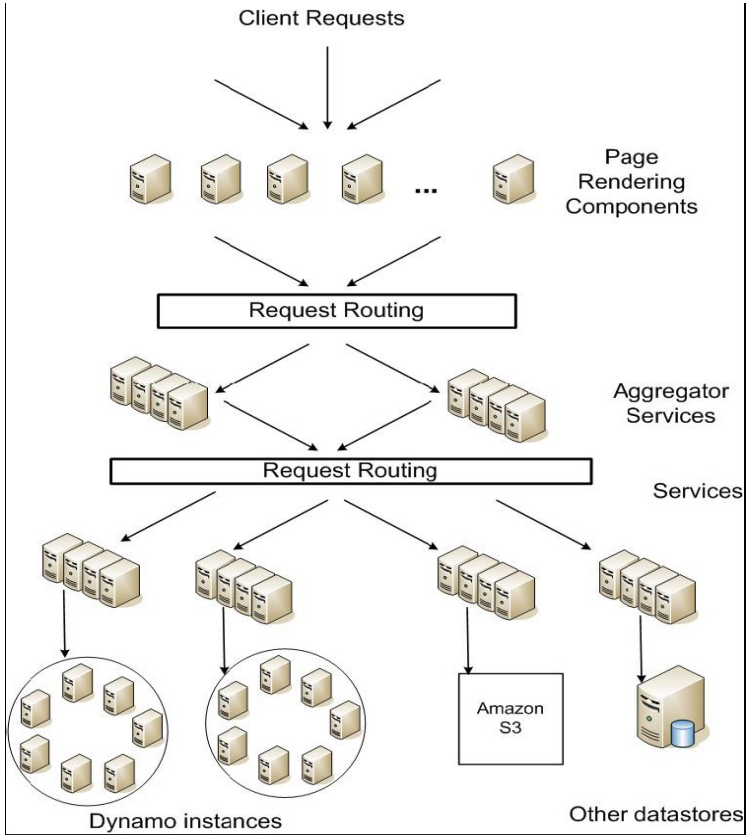
- Open source version to BigTable
- Built on top of HDFS

Dynamo (DynamoDB)

- Developed by Amazon
- Fast R/W speed, satisfying strong service level agreement
- Using a fully distributed architecture
- Store only key-value pairs

Service Level Agreements (SLA)

- Application can deliver its functionality in abounded time: Every dependency in the platform needs to deliver its functionality with even tighter bounds.
- Example: service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second.



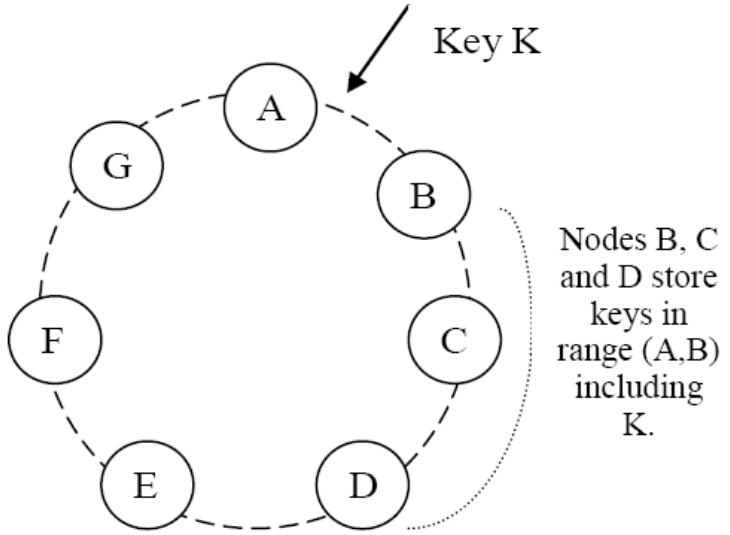
Service-oriented architecture of Amazon's platform

Design Consideration

- Sacrifice strong consistency for availability
- Conflict resolution is executed during ***read*** instead of ***write***, i.e. "always writeable".
- Other principles:
  - Incremental scalability.
  - Symmetry.
  - Decentralization.
  - Heterogeneity.

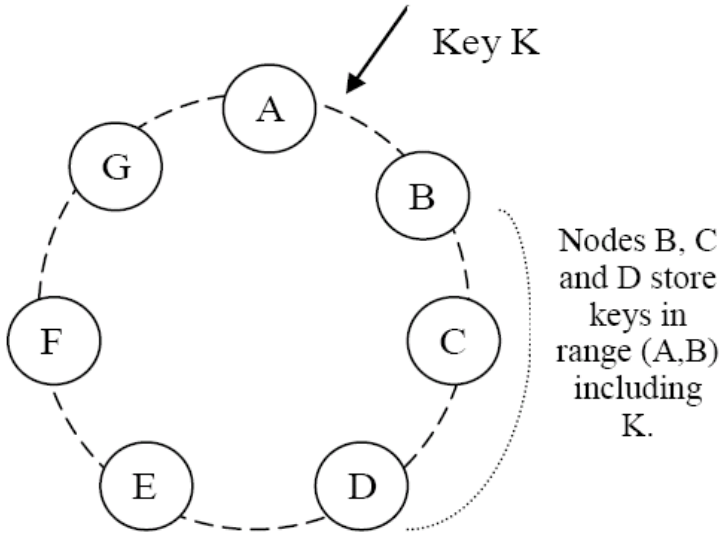
Partition Algorithm: distributed hash table

- **Consistent hashing:** the output range of a hash function is treated as a fixed circular space or “ring”.
- **”Virtual Nodes”:** Each node can be responsible for more than one virtual node.



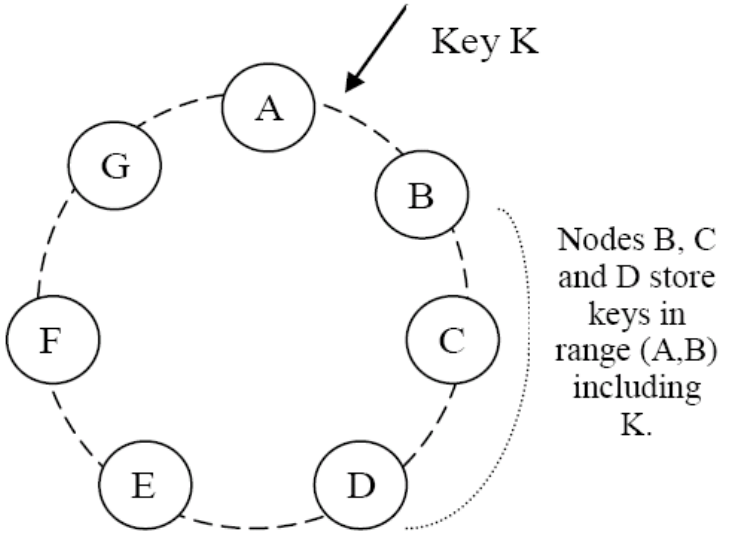
Advantages of using virtual nodes

- If a node becomes unavailable the load handled by this node is evenly dispersed across the remaining available nodes.
- When a node becomes available again, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.
- The number of virtual nodes that a node is responsible can decided based on its capacity, accounting for heterogeneity in the physical infrastructure.



Replication

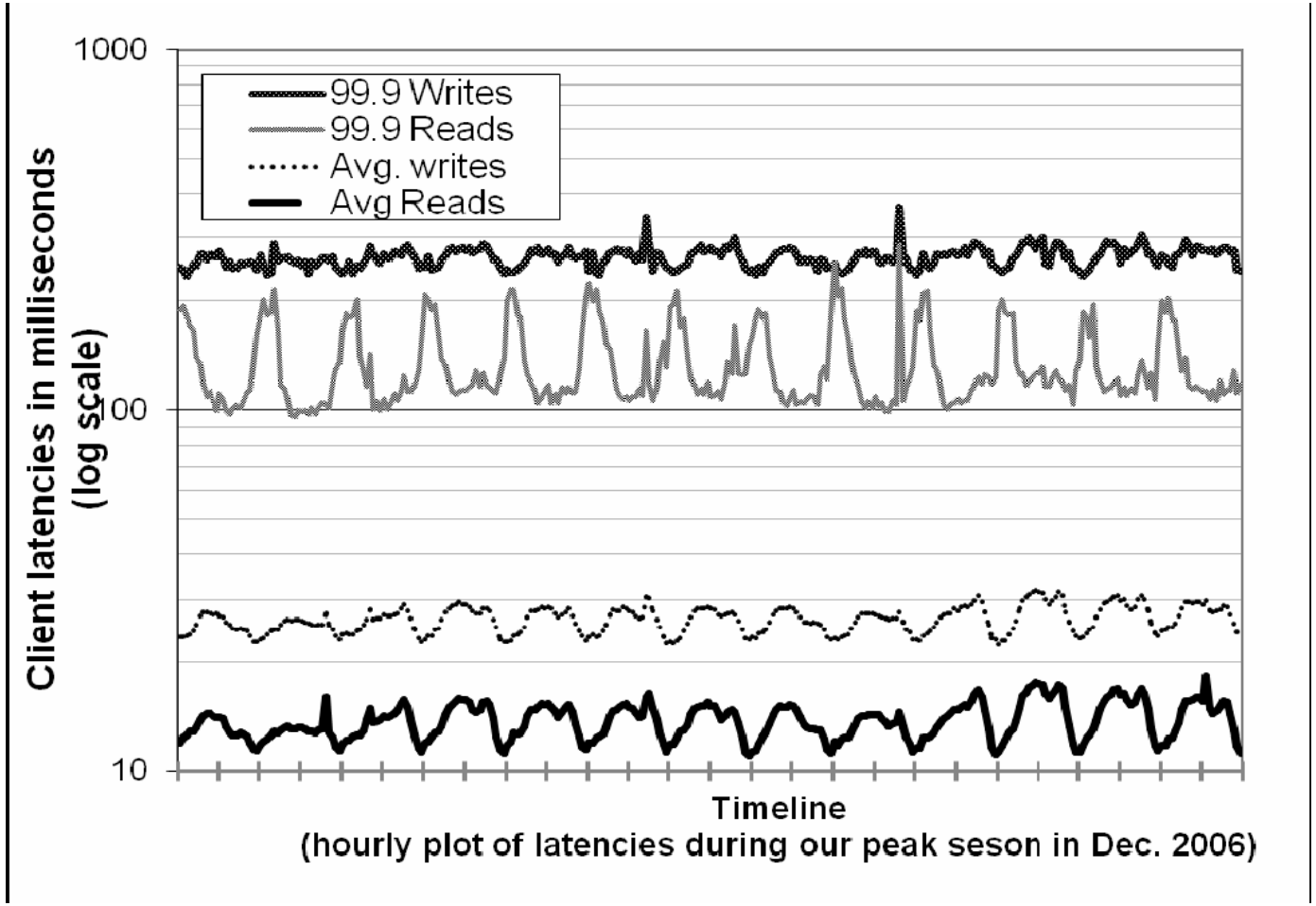
- Each data item is replicated at N hosts.
- "*preference list*": The list of nodes that is responsible for storing a particular key.



Scalability

- Easy to scale up/down
  - Membership join/leaving
  - Redundant data stores

Evaluation


1000
100
10
99.9 Writes
99.9 Reads
Avg. writes
Avg Reads
Client latencies in milliseconds
(log scale)
Timeline
(hourly plot of latencies during our peak seson in Dec. 2006)

Cassandra

- Use BigTable's data model
  - "Rows" and "column family"
- Use Dynamo's architecture

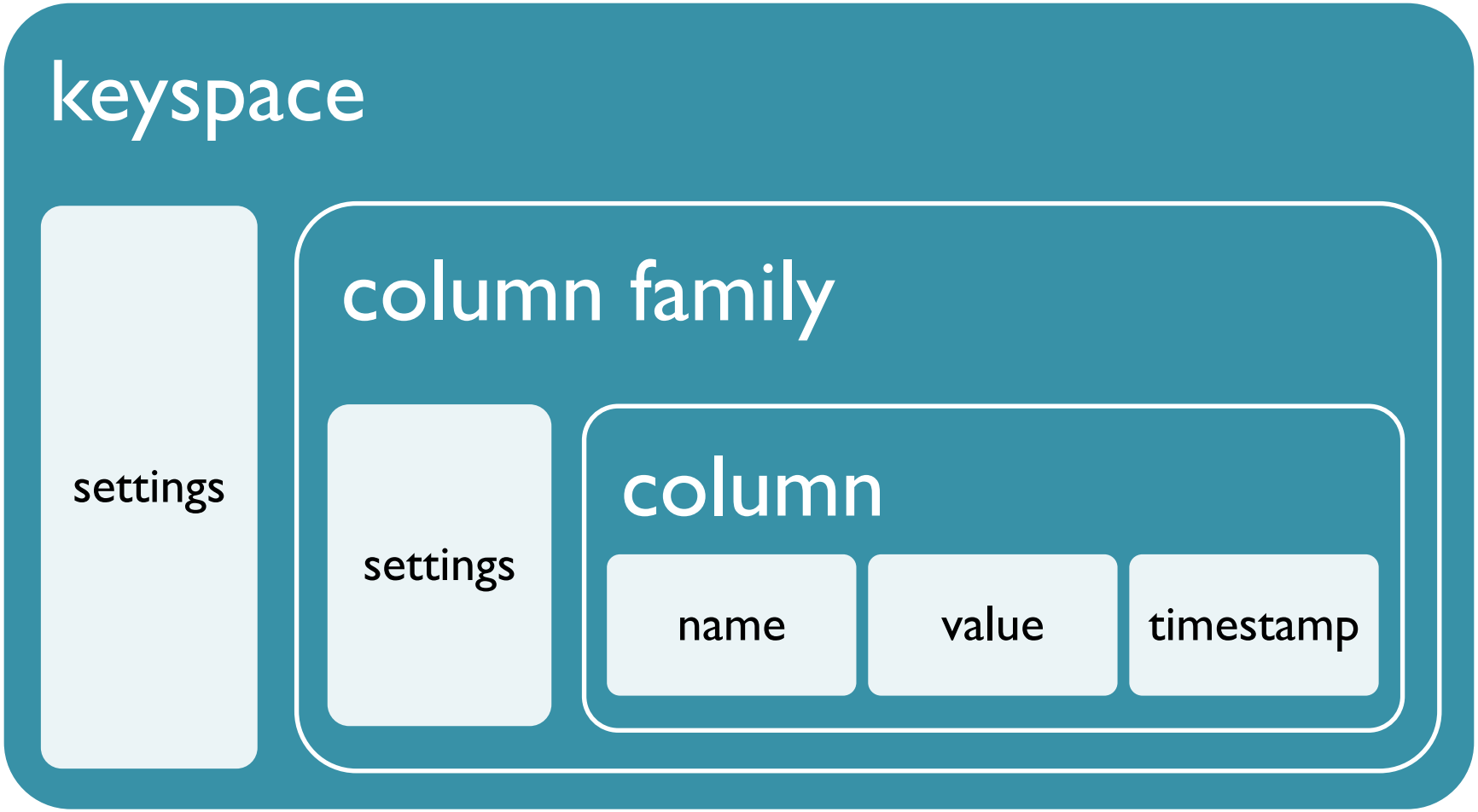
- Impressive measures
  - **MySQL > 50 GB Data**
    Writes Average : ~300 ms
    Reads Average : ~350 ms

  - **Cassandra > 50 GB Data**
    Writes Average : 0.12 ms
    Reads Average : 15 ms

Data Model

- Table is a multi dimensional map indexed by key (row key).
- Columns are grouped into Column Families.
- 2 Types of Column Families
  - Simple
  - Super (nested Column Families)
- Each Column has
  - Name
  - Value
  - Timestamp

Data Model



* Figure taken from Eben Hewitt's (author of Oreilly's Cassandra book) slides.

Architecture

- **Partitioning**
  - How data is partitioned across nodes
- **Replication**
  - How data is duplicated across nodes
- **Cluster Membership**
  - How nodes are added, deleted to the cluster

Partitioning

- **Nodes are *logically* structured in Ring Topology (DHT).**
- **Hashed value of key associated with data partition is used to assign it to a node in the ring.**
- **Hashing rounds off after certain value to support ring structure.**

- **Lightly loaded nodes moves position to alleviate highly loaded nodes.**

Replication

- **Each data item is replicated at N (replication factor) nodes.**

- **Different Replication Policies**
  - **Rack Unaware –** replicate data at N-1 successive nodes after its coordinator
  - **Rack Aware –** uses 'Zookeeper' to choose a leader which tells nodes the range they are replicas for
  - **Datacenter Aware –** similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

Gossip protocol

- Network Communication protocols inspired for real life rumour spreading.
- Periodic, Pairwise, inter-node communication.
- Low frequency communication ensures low cost.
- Random selection of peers.
- Example – Node A wish to search for pattern in data
  - Round 1 – Node A searches locally and then gossips with node B.
  - Round 2 – Node A,B gossips with C and D.
  - Round 3 – Nodes A,B,C and D gossips with 4 other nodes ……
- Round by round doubling makes protocol very robust.

Gossip protocol

- **Variety of Gossip Protocols exists**
  - **Dissemination protocol**
    - **Event Dissemination: multicasts events via gossip. high latency might cause network strain.**
    - **Background data dissemination: continuous gossip about information regarding participating nodes**
  - **Anti Entropy protocol 7**
    - **Used to repair replicated data by comparing and reconciling differences. This type of protocol is used in Cassandra to repair data in replications.**

Cluster management

- Uses Scuttleback (a Gossip protocol) to manage nodes.
- Uses gossip for node membership and to transmit system control state.
- Node Fail state is given by variable ‘phi’ which tells how likely a node might fail (suspicion level) instead of simple binary value (up/down).
- This type of system is known as Accrual Failure Detector.

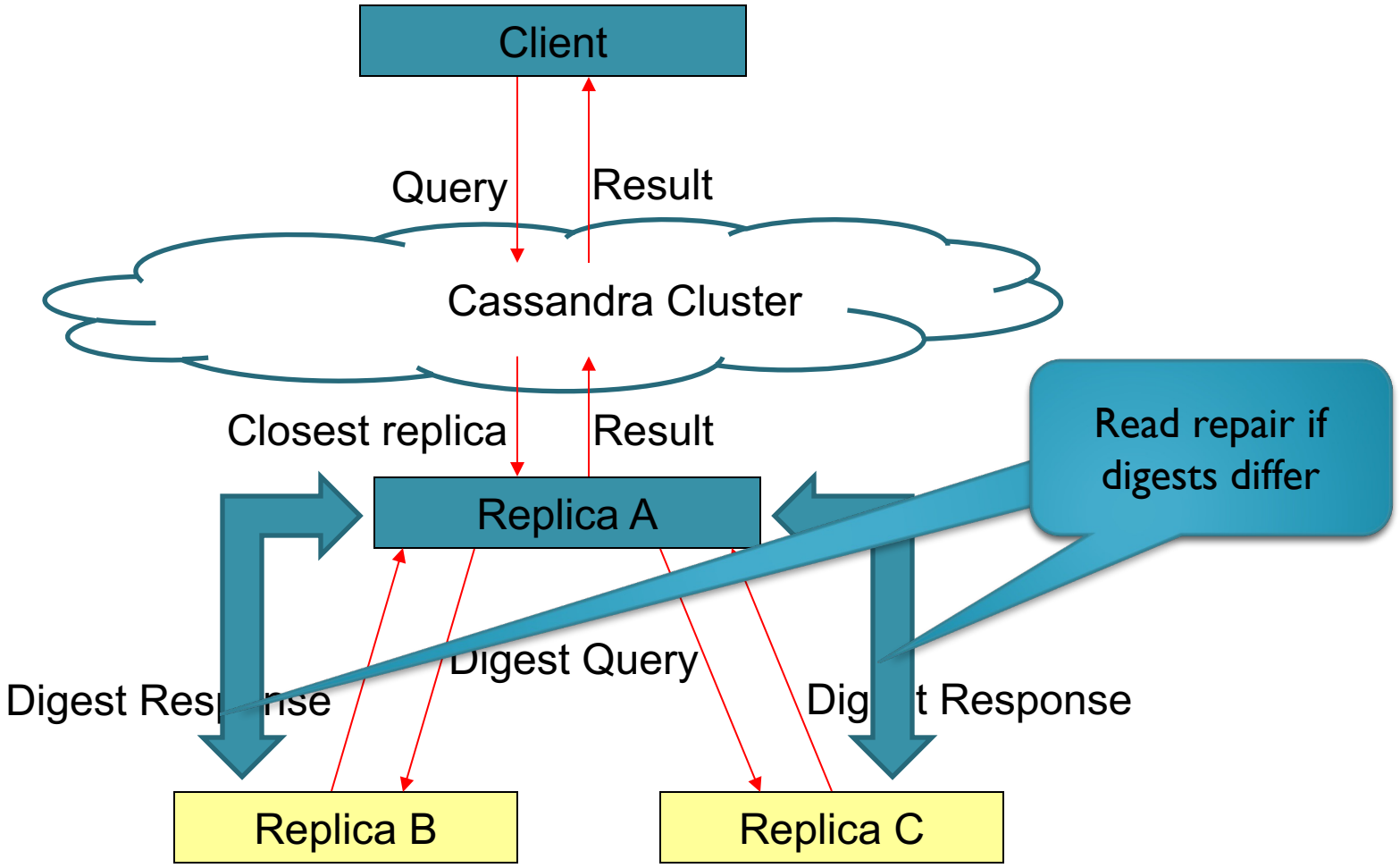
Bootstrapping and Scaling

- **Two ways to add new node**
  - **New node gets assigned a random token which gives its position in the ring. It gossips its location to rest of the ring**
  - **New node reads its config file to contact its initial contact points.**
- **New nodes are added manually by administrator via CLI or Web interface provided by Cassandra.**
- **Scaling in Cassandra is designed to be easy.**
- **Lightly loaded nodes can move in the ring to alleviate heavily loaded nodes.**

Local persistent

- Relies on local file system for data persistency.
- Write operations happens in 2 steps
  - Write to commit log in local disk of the node
  - Update in-memory data structure.
  - Why 2 steps or any preference to order or execution?
- Read operation
  - Looks up in-memory ds first before looking up files on disk.
  - Uses Bloom Filter (summarization of keys in file store in memory) to avoid looking up files that do not contain the key.

Read operation



* Figure taken from Avinash Lakshman and Prashant Malik (authors of the paper) slides.

Facebook inbox search

- Cassandra developed to address this problem.
- 50+TB of user messages data in 150 node cluster on which Cassandra is tested.
- Search user index of all messages in 2 ways.
  - Term search : search by a key word
  - Interactions search : search by a user id

Latency Stat	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Max	26.13 ms	44.41 ms

Comparison on cloud data serving systems

- Paper "Benchmarking Cloud Serving Systems with YCSB"

Update heavy workload

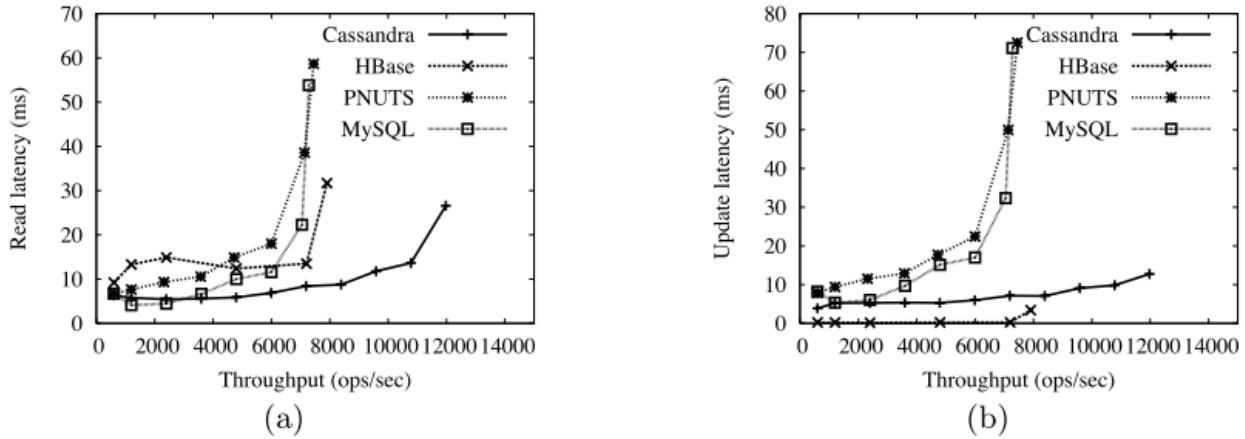


Figure 3: Workload A—update heavy: (a) read operations, (b) update operations. Throughput in this (and all figures) represents total operations per second, including reads and writes.

Cluster size

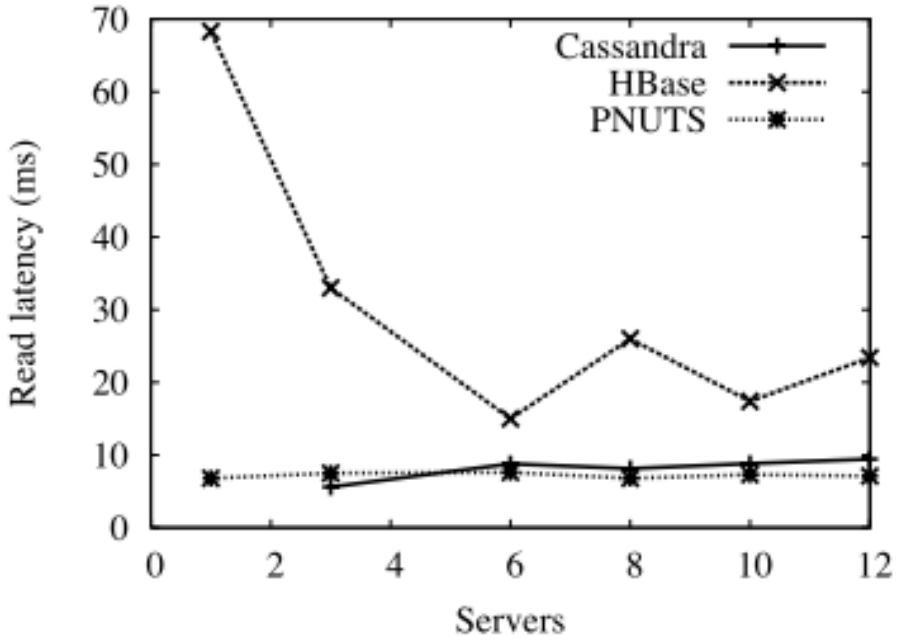


Figure 6: Read performance as cluster size increases.

Infrastructure as a service

- Elastic Compute Cloud (EC2)
- Simple Storage Services (S3)
- CloudFront
- Elastic Mapreduce

EC2

- A typical example of utility computing
- functionality:
  - launch instances with a variety of operating systems (windows/linux)
  - load them with your custom application environment (customized AMI (Amazon Machine image))
  - Full root access to a blank Linux machine
  - manage your network's access permissions
  - run your image using as many or few systems as you desire (scaling up/down)

Backyard…

- Powered by Xen – Virtual Machine
  - Different from Vmware & VPC

    - high performance
  - Hardware contributions by Intel (VT-x/Vanderpool) and AMD (AMD-V)
  - Supports "Live Migration" of a virtual machine between hosts

Amazon Machine Images

- **Public AMIs:** Use pre-configured, template AMIs to get up and running immediately. Choose from Fedora, Movable Type, Ubuntu configurations, and more
- **Private AMIs:** Create an Amazon Machine Image (AMI) containing your applications, libraries, data and associated configuration settings
- **Paid AMIs:** Set a price for your AMI and let others purchase and use it (Single payment and/or per hour)
  - AMIs with commercial DBMS

Normal way to use EC2

- For web applications
  - Run your base system in minimum # of VMs
  - Monitoring the system load (user traffic)
  - Load is distributed to VMs
  - If over some threshold → increase # of VMs
  - If lower than some thresholds → decrease # of VMs
- For data intensive analysis
  - Estimate the optimal number of nodes (tricky!)
  - Load data
  - Start processing

Type of instances

- Standard instances (micro, small, large, extra)
  - E.g., small: 1.7GB Memory, 1EC2 Compute Unit (1 2ghz core?), 160 GB instance storage
- High-CPU instances
  - More CPU with same amount of memory

AMIs with special software

- IBM DB2, Informix Dynamic Server, Lotus Web Content Management, WebSphere Portal Server
- MS SQL Server, IIS/Asp.Net
- Hadoop
- Open MPI
- Apache web server
- MySQL
- Oracale 11g
- …

Pricing

On-Demand Instance Prices

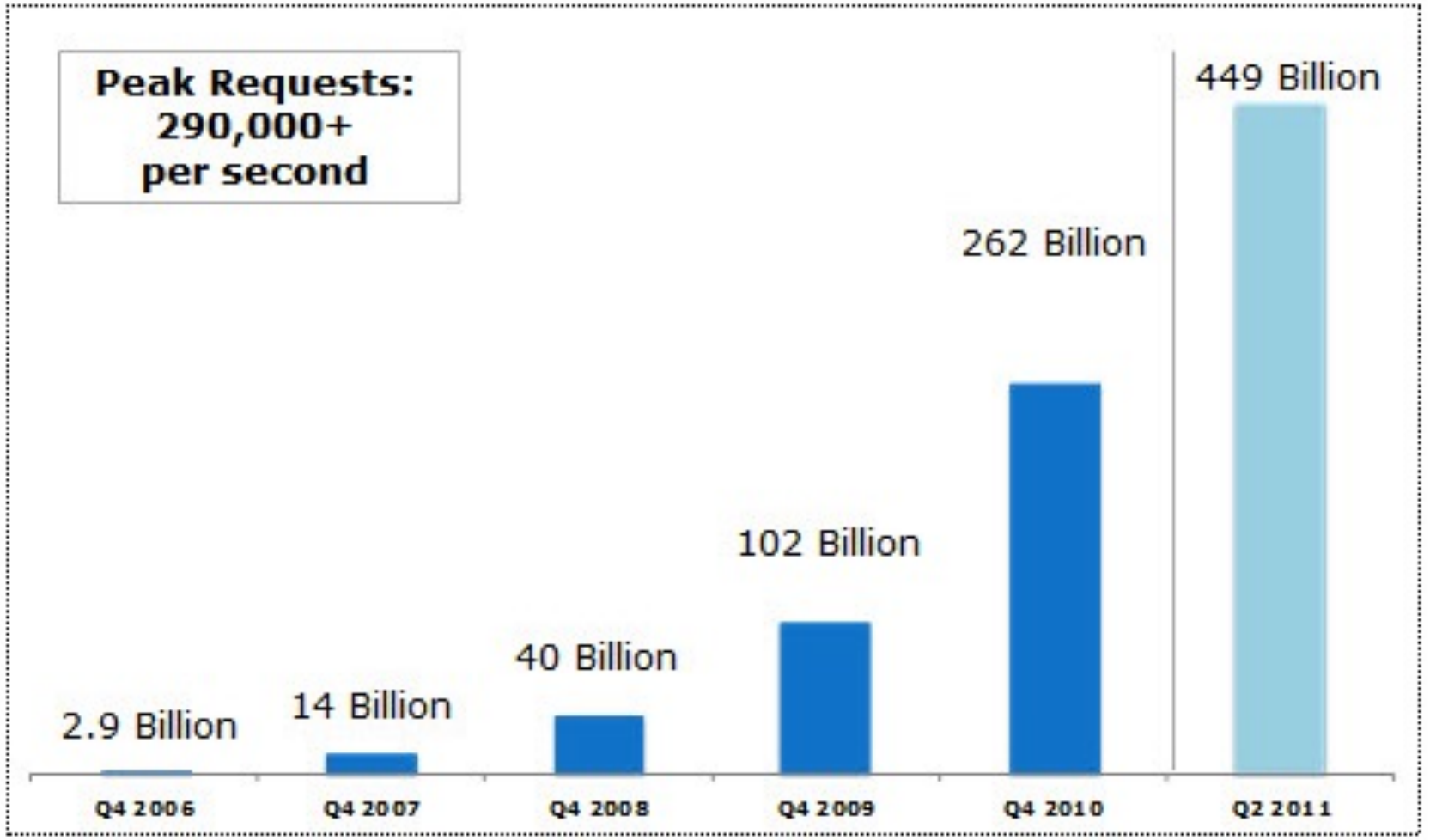
Region: US East (N. Virginia)

	Linux/UNIX Usage
Standard On-Demand Instances	
Small (Default)	$0.060 per Hour
Medium	$0.120 per Hour
Large	$0.240 per Hour
Extra Large	$0.480 per Hour
Second Generation Standard On-Demand Instances	
Extra Large	$0.500 per Hour
Double Extra Large	$1.000 per Hour
Micro On-Demand Instances	
Micro	$0.020 per Hour
High-Memory On-Demand Instances	
Extra Large	$0.410 per Hour
Double Extra Large	$0.820 per Hour
Quadruple Extra Large	$1.640 per Hour
High-CPU On-Demand Instances	
Medium	$0.145 per Hour
Extra Large	$0.580 per Hour
Cluster Compute Instances	
Quadruple Extra Large	$1.300 per Hour
Eight Extra Large	$2.400 per Hour
High-Memory Cluster On-Demand Instances	
Eight Extra Large	$3.500 per Hour
Cluster GPU Instances	

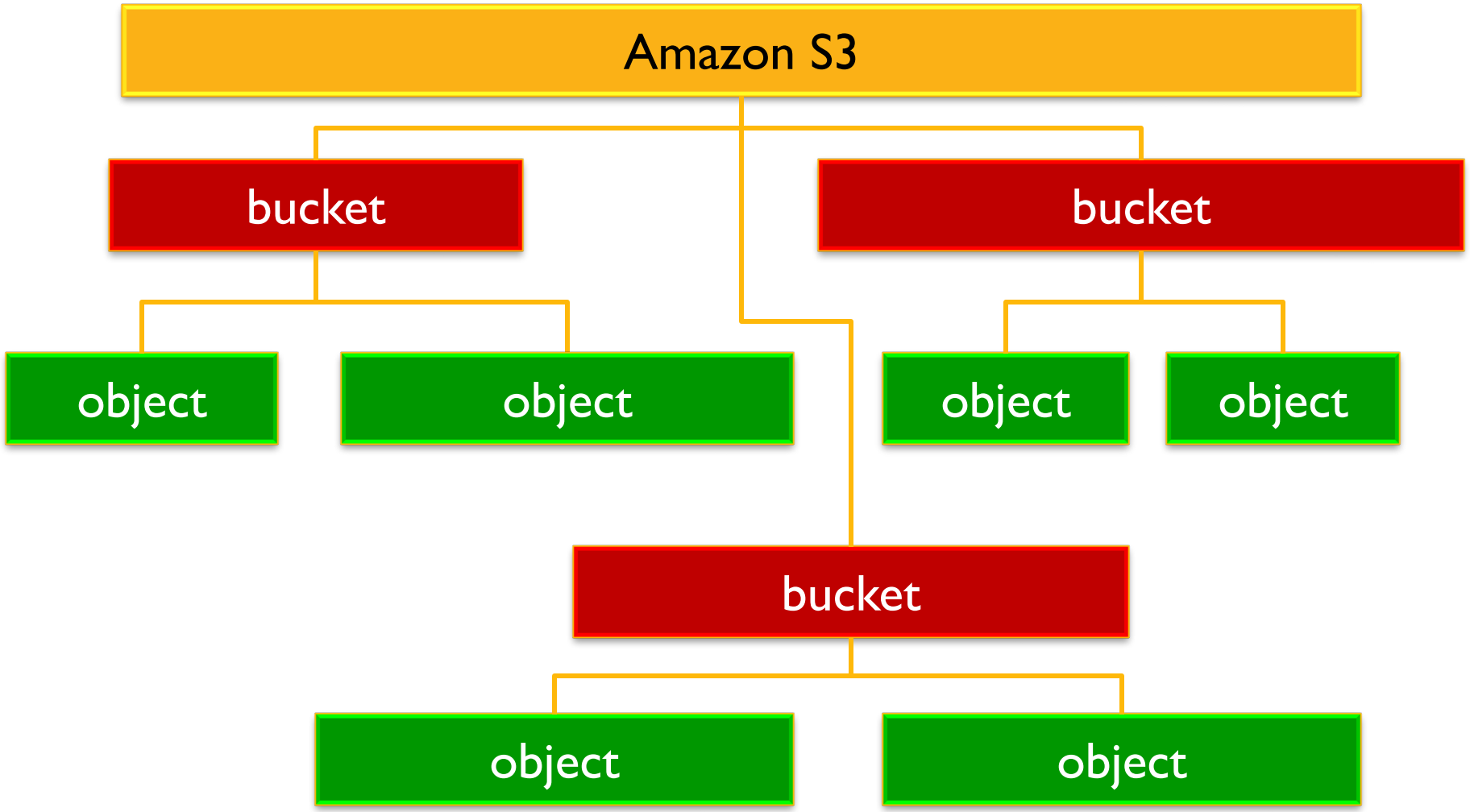
S3

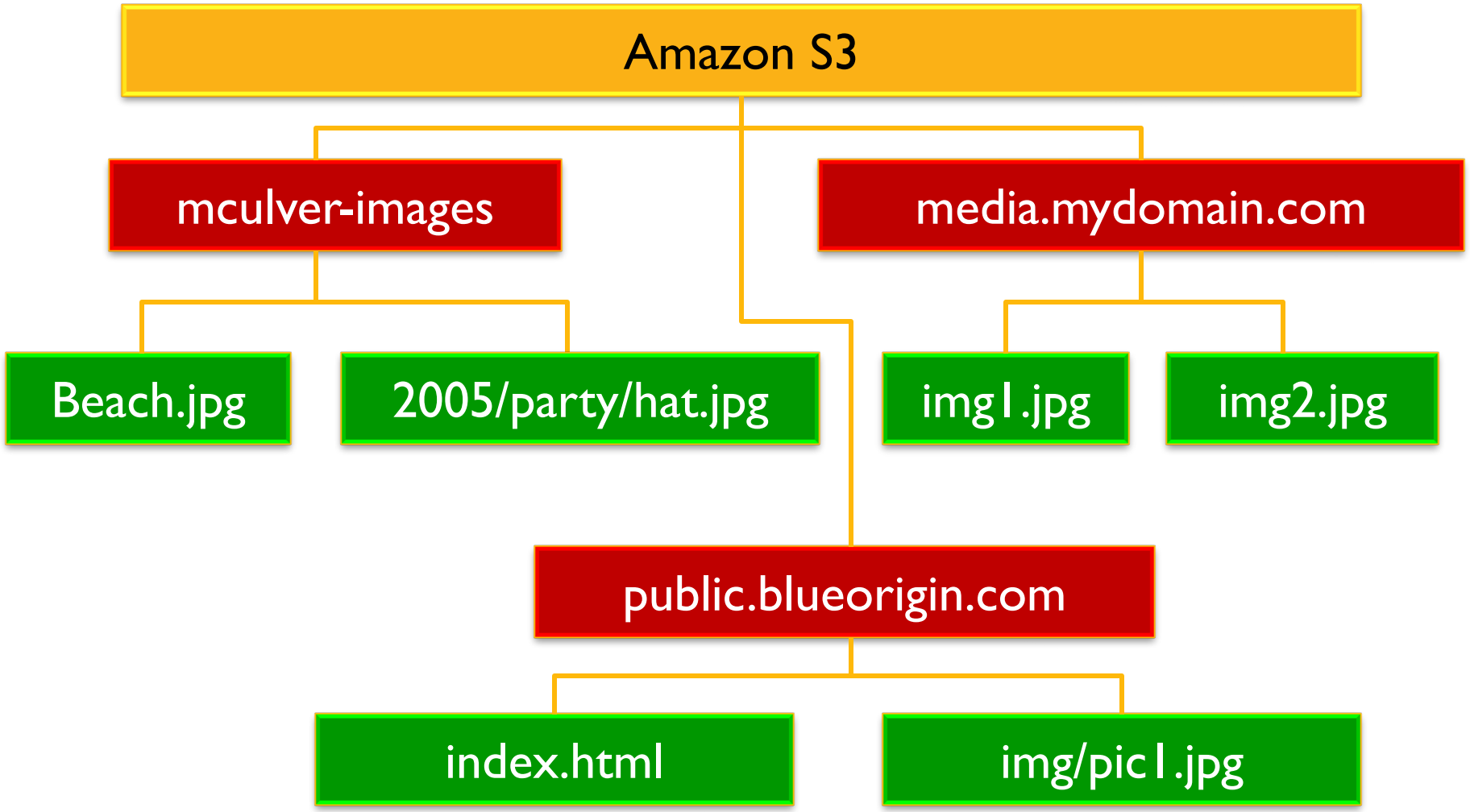
- Write,read,delete objects 1byte-5gb
- Namespace: buckets, keys, objects
- Accessible using URLs

S3 scale


Peak Requests:
290,000+
per second
2.9 Billion
14 Billion
40 Billion
102 Billion
262 Billion
449 Billion
Q4 2006
Q4 2007
Q4 2008
Q4 2009
Q4 2010
Q2 2011

S3 namespace




Amazon S3
mculver-images
media.mydomain.com
Beach.jpg
2005/party/hat.jpg
img1.jpg
img2.jpg
public.blueorigin.com
index.html
img/pic1.jpg

Accessing objects

- Bucket: keke-images, key: jpg1, object: a jpg image
  - accessible with

https://**keke-images**.s3.amazonaws.com/**jpg1**

- mapping your subdomain to S3
  - with DNS CNAME configuration
  - e.g. media.yourdomain.com → media.yourdomain.com.s3.amazonaws.com/

Elastic Mapreduce

- Based on hadoop AMI
- Data stored on S3
- “job flow”

Example

```
elastic-mapreduce --create --stream \
 --mapper
  s3://elasticmapreduce/samples/wordcount/wordSplitter.py
  \
 --input
s3://elasticmapreduce/samples/wordcount/input
 --output s3://my-bucket/output
 --reducer aggregate
```

Most popular open-source AWS equivalence

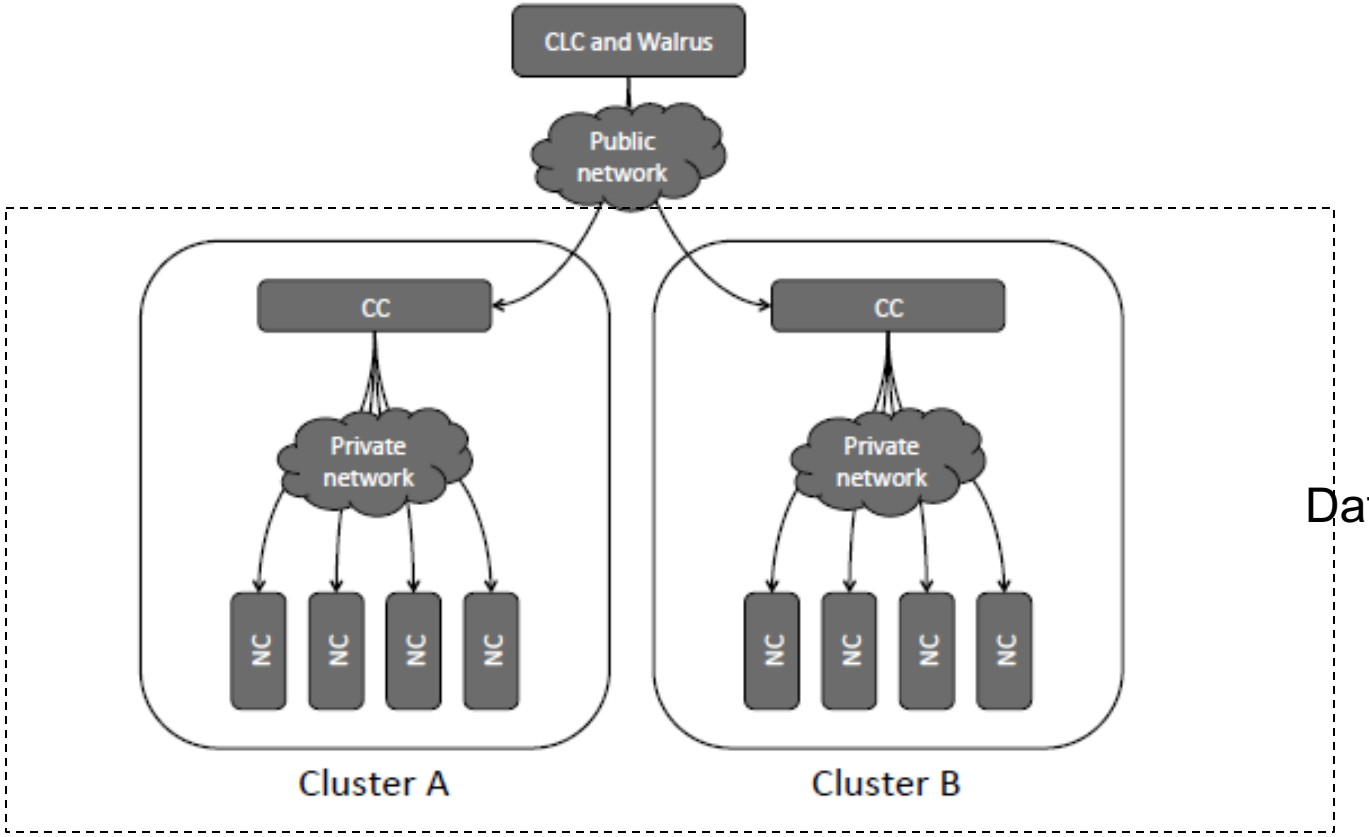
- Eucalyptus
  - Started by UCSB researchers, now a company
- OpenStack
  - Started by NASA, now an open source platform

Eucalyptus

- Compatible to AWS APIs (EC2, S3, mainly)
  - A good example for understanding how AWS works

- Paper "**The Eucalyptus Open-source Cloud-computing System"**
  - How VM instances are managed
  - How to provide virtual network (like elastic IP)
  - How to provide data storage (like S3)
  - A very brief description, but we can get something

System Design



CLC: cloud controller
CC: cluster controller

Walrus: storage controller similar to S3
NC: node controller

Components: Node Controller

- Make queries to discover physical resources
  - # of cores
  - Size of memory
  - Available disk space
  - State of VM instances
- Propagate the information to Cluster Controller
  - *DescribeResource*
  - *DescribeInstances*
- Run/terminate instances
  - CLC→CC→ NC → hypervisor (Xen)

Node controller

- Start an instance
  - Copy instance image from walrus or local cache
  - Create endpoint in the virtual network overlay
  - Instruct hypervisor to boot the instance
- Stop an instance
  - Instruct hypervisor to terminate the VM
  - Tear down the virtual network endpoint
  - Clean up the files associated with the instance

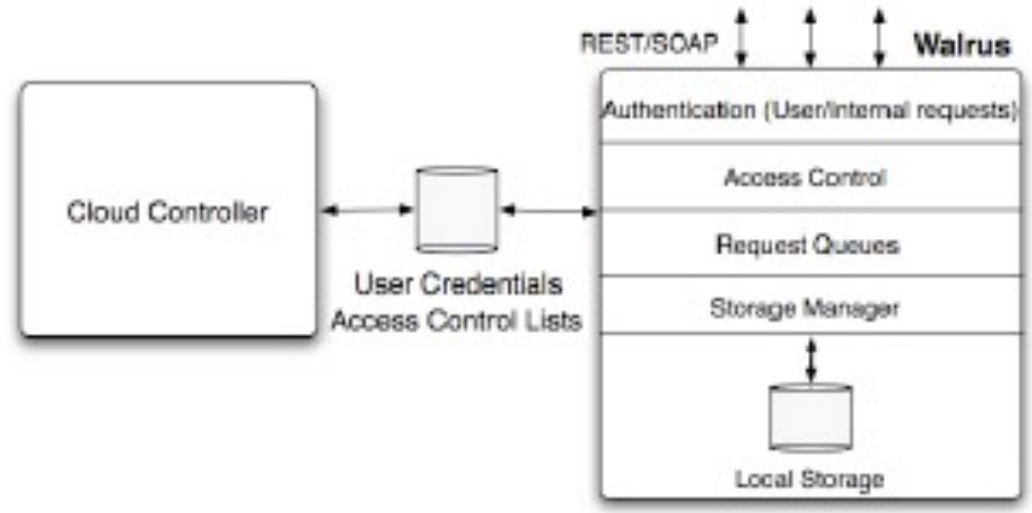
Cluster Controller

- Gather/report information of NCs
  - Through the interface provided by NCs
  - Report the summary to CLC
- Schedule incoming instance "run" requests to specific NCs
- Control the virtual network overlay

Storage Controller (Walrus)

- Provide SOAP/REST interfaces
  - Compatible with S3 – you can use S3 tools
- Use Walrus to stream data in/out of the cloud
- Store VM images (same as AMI)
  - Root file system, kernel image, ramdisk image
- No locking for object writes
  - Conflict writes – late write overwrites the earlier

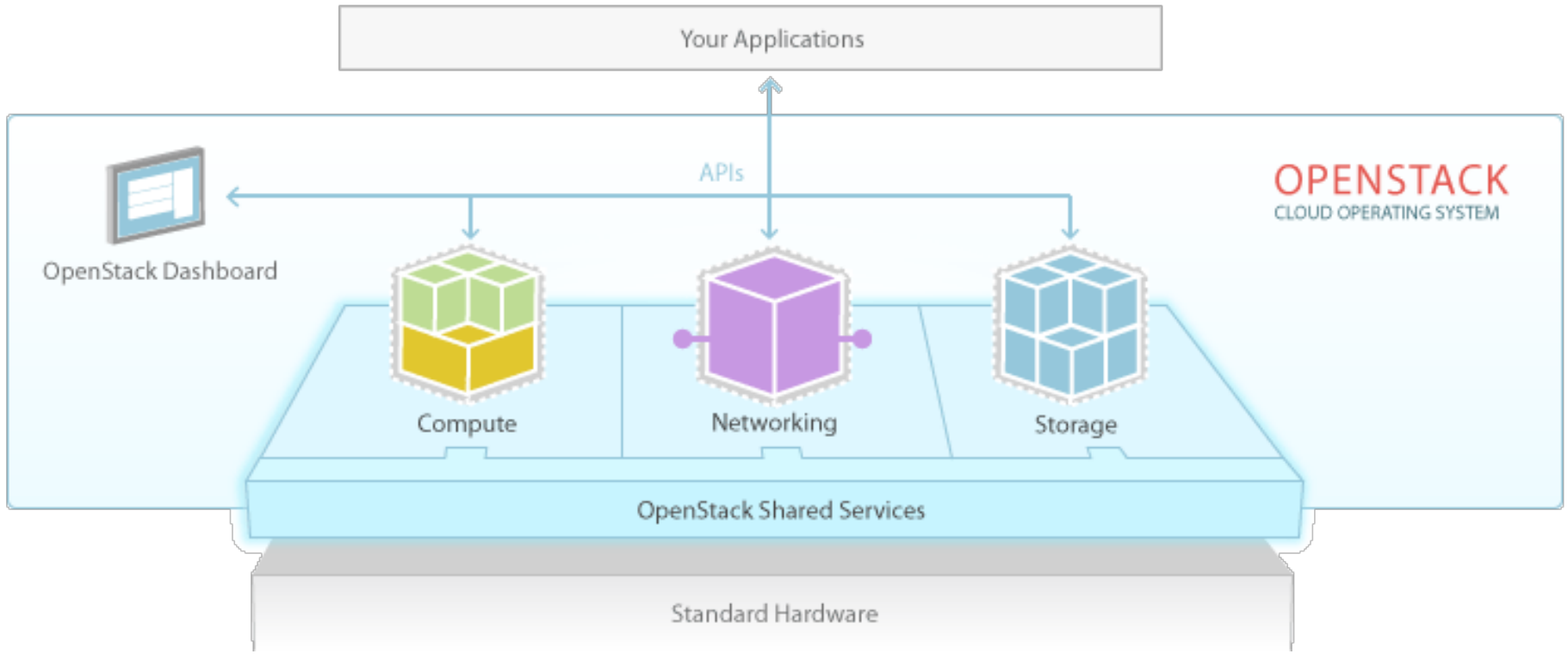
- Provides the same tool Amazon uses
  - Generate AMI
- Maintains a cache of images
- Authentication is applied when NC accesses images



OpenStack

OpenStack

- Originated at NASA, with Rackspace
- Driven by an open community process
- Multiple hypervisors: Xen, KVM, ESXi, Hyper-V
- First release: Oct 2010


Your Applications
APIs
OPENSTACK
CLOUD OPERATING SYSTEM
OpenStack Dashboard
Compute
Networking
Storage
OpenStack Shared Services
Standard Hardware

Components

- Nova – Compute (equivalent to EC2)
- Swift – object storage (S3)
- Image service (AMI)
- Networking (virtual network)
- Block storage (Elastic block storage)
- Identity
- Dashboard (AWS web console)

-- mostly implemented with python

Fastest Growing Global Open Source Community

COMPANIES

231

INDIVIDUAL MEMBERS

10,149

COUNTRIES

121

TOTAL CONTRIBUTORS

1,036

AVERAGE MONTHLY CONTRIBUTORS

238

CODE CONTRIBUTIONS

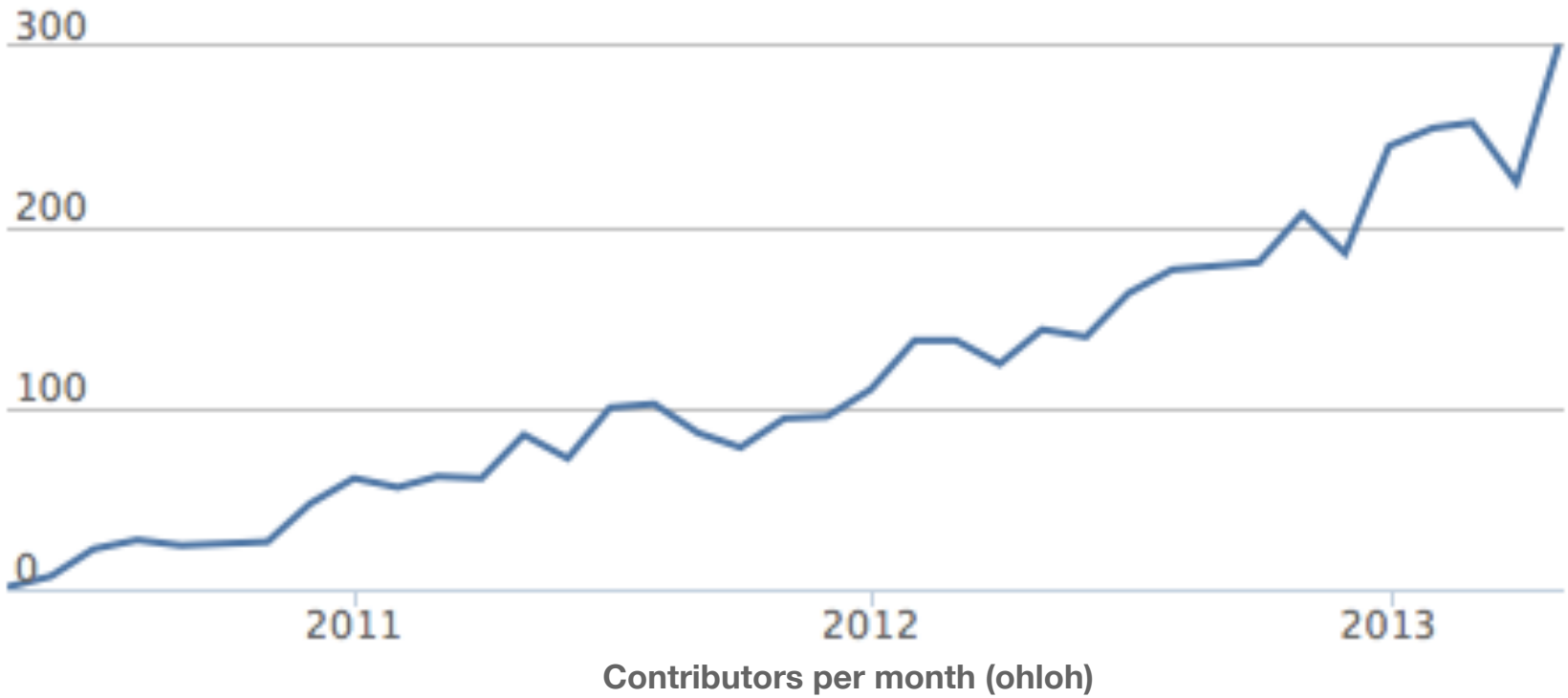
70,137

As of July 2013

Global Community

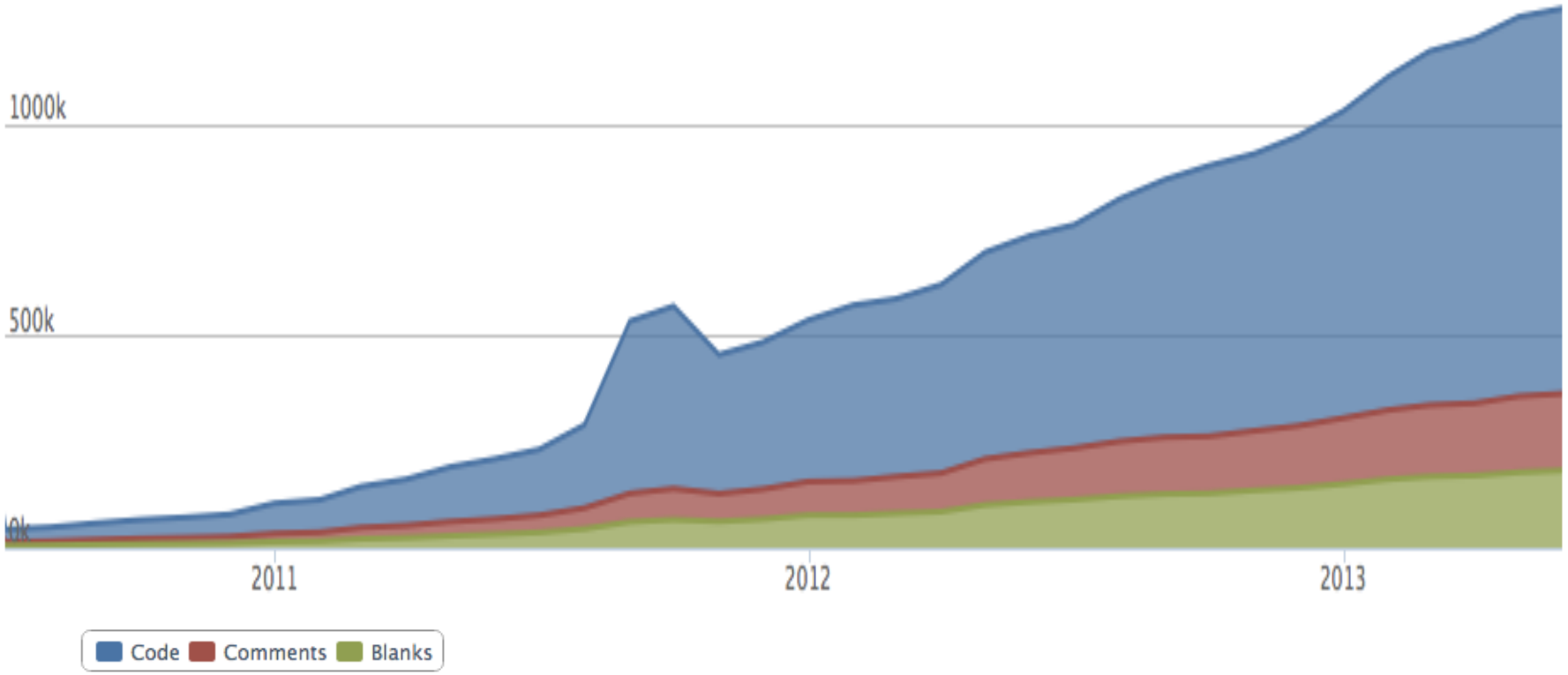
Countries with members

Developer Growth



Contributors per month (ohloh)

1 Million+ Lines of Code



Lines of code (ohloh)