

How To Build A Distributed Storage System

Will Reese

Who am I?

- Will Reese
- Rackspace since 2002
- dev manager and architect for CloudFiles
- will.reese@rackspace.com

CloudFiles

- what? distributed object/blob storage
- why? archiving data and serving media
- when? started in 2007, product as of 2009
- who? 8 developers, 4 administrators
- where? Rackspace Cloud
- how? ...

Design Goals

- 100 petabytes of storage
- 100 billion objects
- 100 gigabits per second throughput
- 100 thousand requests per second

Design Concepts

- design the system around the api
- adopt a memcache approach to scaling
- use ideas from distributed databases
- keep it as simple as possible

Basic Functionality

- accounts, containers, objects
- read and write objects
- list objects in a container
- list containers on an account
- python servers, http, eventlet

Object Server

- read and write objects and metadata
- objects are stored in files
- metadata stored in xattr
- directory per name, file per version
- filesystems vary, we use xfs

Container Server

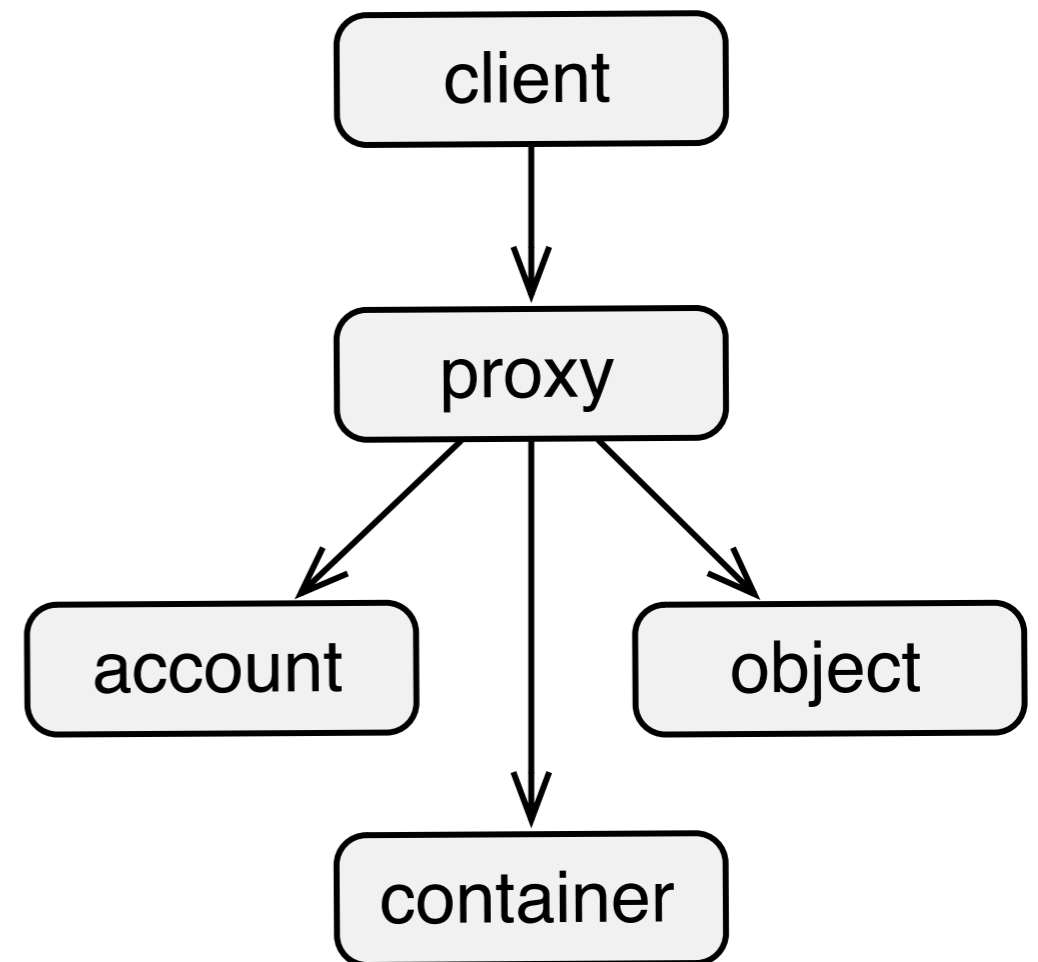
- list objects in a container
- sorted list of objects, b-tree
- sqlite db per container, row per object
- id, object name, timestamp, deleted, etc
- limitations per container
- use multiple containers (if needed)

Account Server

- list containers on an account
- sorted list of containers, b-tree
- sqlite db per account, row per container
- id, container name, timestamp, deleted, etc
- similar limitations, doesn't really matter

Proxy Server

- handles client requests
- routes requests to the appropriate servers
- works around failures
- memcache approach

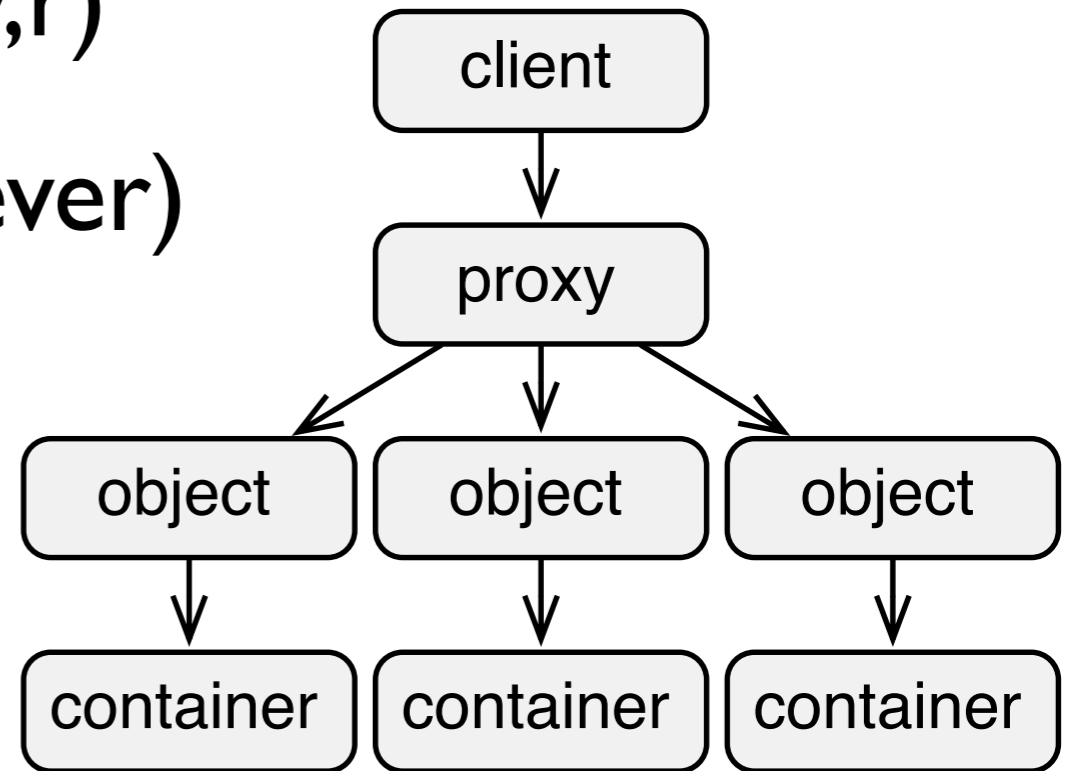


Partitioning

- spread objects, containers, and accounts across a cluster of machines
- effective use of hardware and network
- started with consistent hashing
- now we assign partitions to devices
- replicas, weights, zones

Routing

- replicas, writes, reads (n,w,r)
- write once, read many (never)
- $n=3, w=2, r=1$
- eventual consistency
- read your writes



Versioning

- last write wins, name and timestamp
- use ntp (worm, duration > offset)
- each request gets a timestamp and uuid
- directory per name, file per version
- no updates, deletes are tombstones

Replication

- background job for each server type
- responsible for local data, push not pull
- object replication syncs partitions of objects using rsync and hash lists
- account and container replication sync sqlite databases using rsync, chexor and rowids

Fault Tolerance

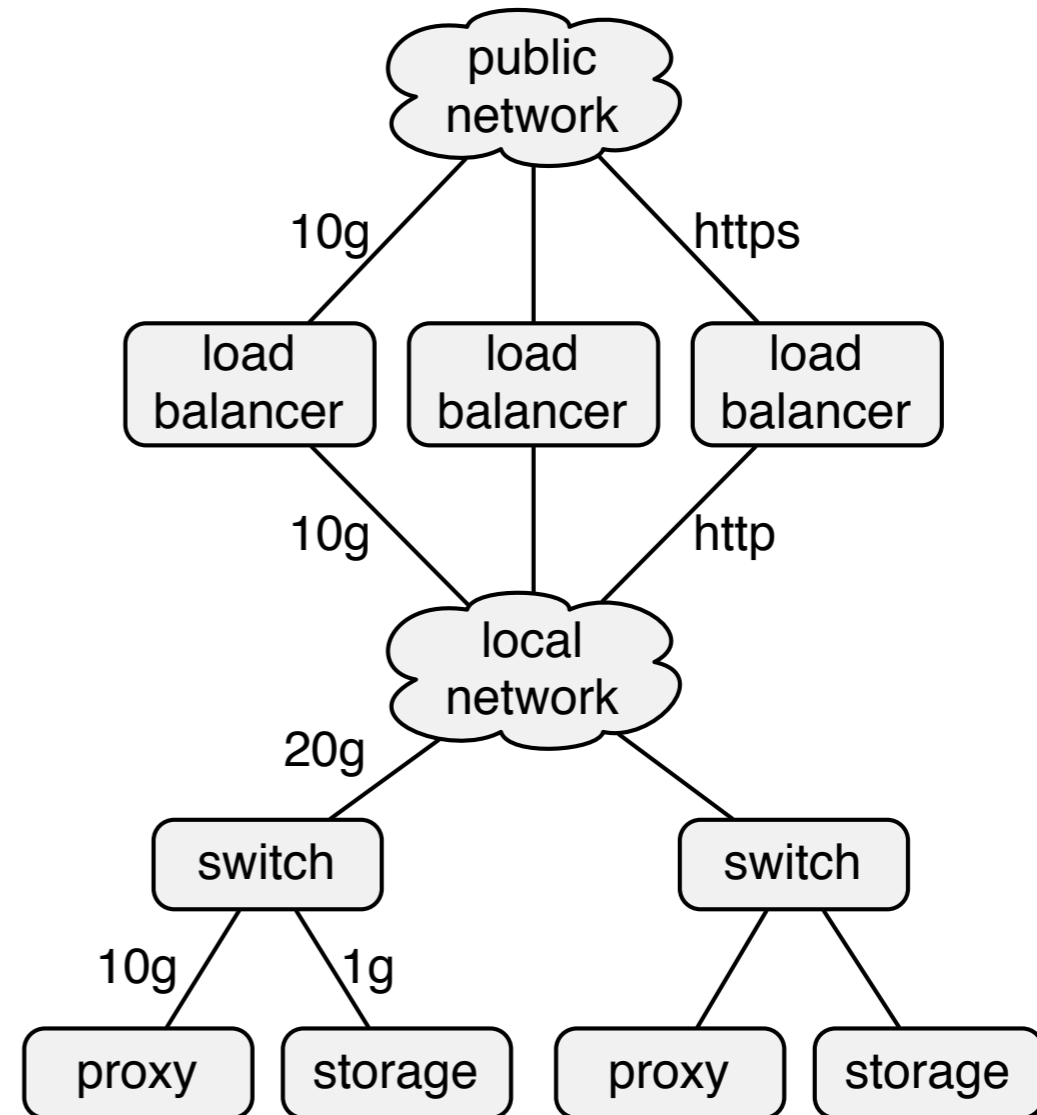
- failures are common
- temporary vs permanent
- techniques (replication, timeouts, handoff, 100-continue, error limiting, updaters, checksums, auditors)
- possible improvements (gossip, read-repair)

Hardware

- commodity, agnostic, heterogeneous
- proxy (cpu, network)
- object server (disk capacity)
- account and container servers (disk perf)
- raid not needed (or wanted)

Network

- part of the design
- 10g network
- load balancing
- horizontal scaling



OpenStack

- open source cloud computing
- starting with compute and object storage
- Rackspace and NASA
- <http://openstack.org>
- <https://launchpad.net/swift>
- launch party!



Questions?