Carnegie Mellon

Applied Distributed Systems

The Cloud 2

Overview

- **Failure in the Cloud**
- Sharing data

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

Failures in the cloud

- Cloud failures large and small
- The Long Tail
- Techniques for dealing with the long tail

Sometimes the whole cloud fails

Selected Cloud Outages 2020

- Twitter
- Microsoft Azure
- Microsoft Teams. .
- Google Cloud Platform
- YouTube.
- AWS

And sometimes just a part of it fails …

A year in the life of a Google datacenter

- Typical first year for a new cluster:
  - ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
  - ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
  - ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
  - ~5 racks go wonky (40-80 machines see 50% packetloss)
  - ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
  - ~12 router reloads (takes out DNS and external vips for a couple minutes)
  - ~3 router failures (have to immediately pull traffic for an hour)
  - ~dozens of minor 30-second blips for dns
  - ~1000 individual machine failures
  - ~thousands of hard drive failures

    slow disks, bad memory, misconfigured machines, flaky machines, dead horses, …

Amazon failure statistics

- In a data center with ~64,000 servers with 2 disks each

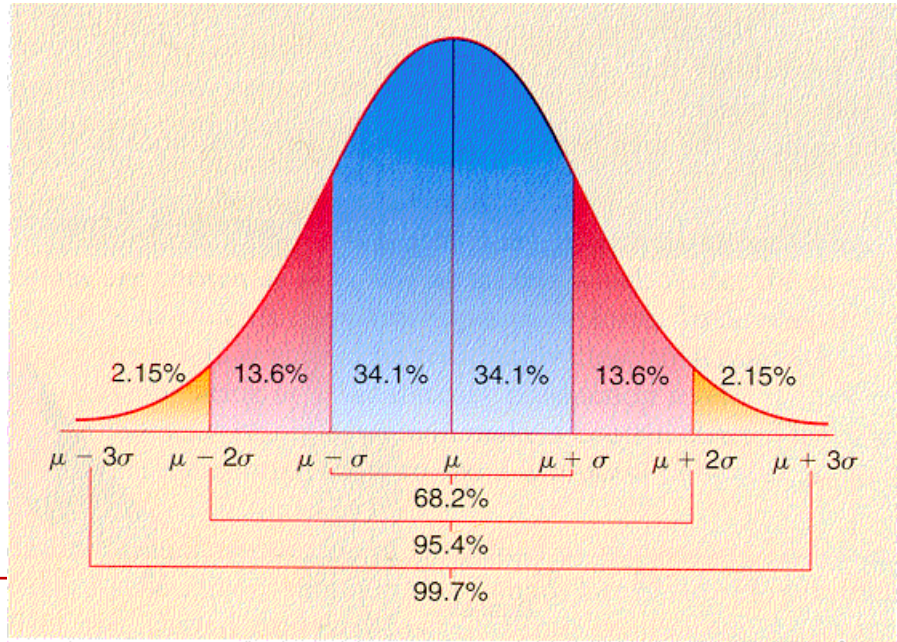
  ~5 servers and ~17 disks fail every day.

Failures in the cloud

- Cloud failures large and small
- The Long Tail
- Techniques for dealing with the long tail

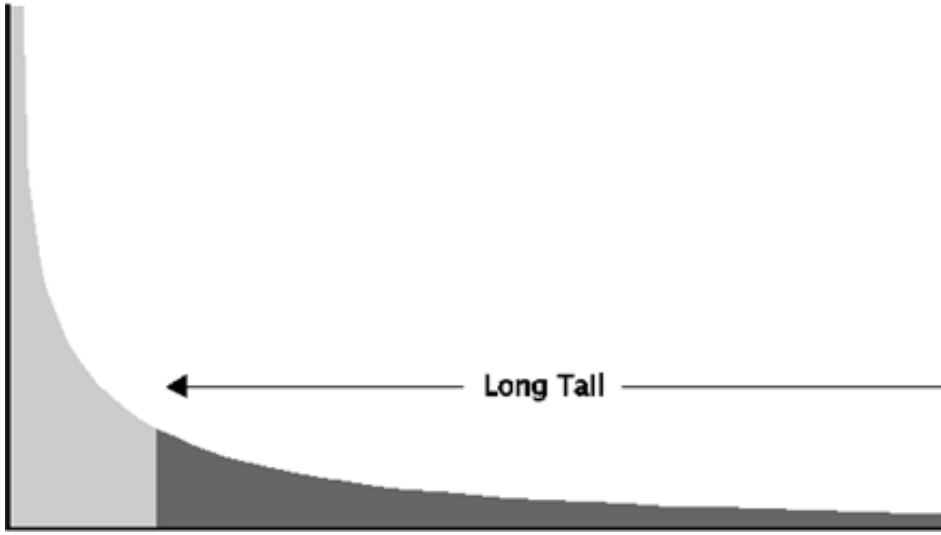
Short digression into probability

- A distribution describes the probability than any given reading will have a particular value.
- Many phenomenon in nature are “normally distributed”.
- Most values will cluster around the mean with progressively smaller numbers of values going toward the edges.
- In a normal distribution the mean is equal to the median



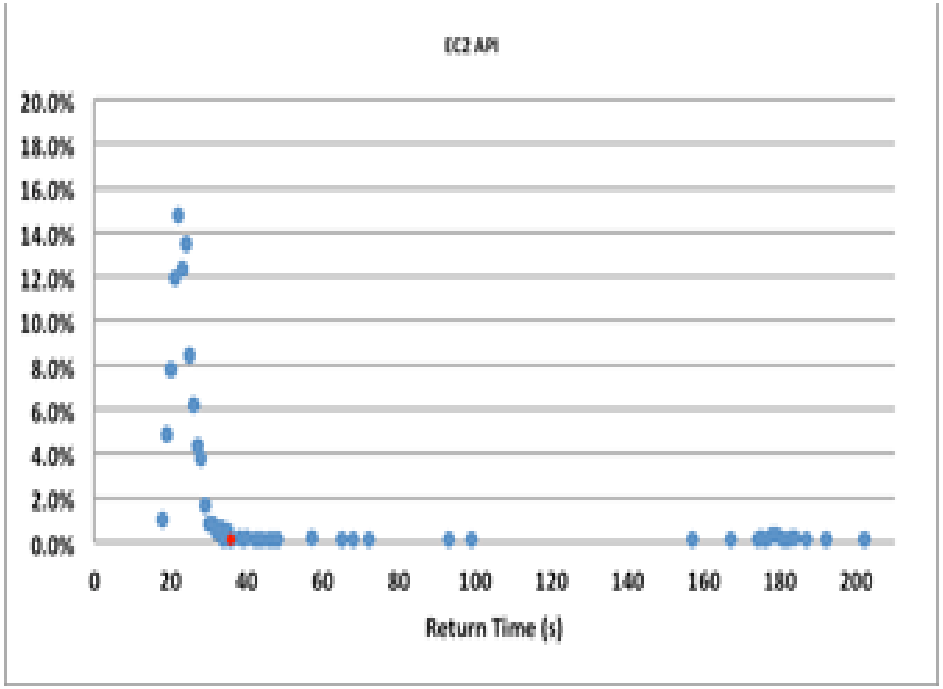
Long Tail

- In a long tail distribution, there are some values far from the median.
- These values are sufficient to influence the mean.
- The mean and the median are dramatically different in a long tail distribution.



What does this mean?

- If there is a partial failure of the cloud some activities will take a long time to complete and exhibit a long tail.
- The figure shows distribution of 1000 AWS "launch instance" calls.
- 4.5% of calls were "long tail"



Failures in the cloud

- Cloud failures large and small
- The Long Tail
- Techniques for dealing with the long tail

What can you do to prevent long tail problems?

- Recognize that failure has occurred
- Recover from failure
- Mask failure

Recognizing failure

- In a distributed system, the only communication between distinct computers is through messages.
- An unanswered request (message) to another machine may indicate the recipient has failed.
- The failure is recognized by the requestor making the request.
- Requests are asynchronous. The requestor sets a timeout.

Health Check

- Health check. Requires one machine to act as a monitor for other machines.
  - Machine sends a message to a monitor periodically saying “I am alive”
  - Ping/echo. Monitor sends a message to the machine being monitored and expects a reply within specified period.

Recovering from failure

- Retry
- Instantiate another instance of failed service
- Graceful degradation
- Set “circuit breaker” to prevent trying to use service again.

Masking failure

- "Hedged" request. Suppose you wish to launch 10 instances. Issue 11 requests. Terminate the request that has not completed when 10 are completed.
- "Alternative" request. In the above scenario, issue 10 requests. When 8 requests have completed issue 2 more. Cancel the last 2 to respond.
- Using these techniques reduces the time of the longest of the 1000 launch instance requests from 202 sec to 51 sec.

Overview

- Failure in the Cloud
- **Sharing data**

Sharing data across portions of a distributed app

- Different portions of a distributed app may wish to share data without the latency of message passing and writing to persistent storage.
- Shared data is frequently the basis of operations that must appear to be atomic.

Atomicity

- An atomic operation is one that cannot be interrupted.
- Atomic operations appear to be instanteous.
- Difficult to achieve in distributed systems because of the reliance on messages.

Message timing considerations

- Main memory reference 100 ns
- 1K bytes over 1 Gbps network 0.01 ms
- Round trip within same datacenter 0.5 ms
- Packet CA->Netherlands->CA 150 ms

- Maintaining appearance of atomicity using messages will take time

Non-atomic operations can cause problems

- Problems that may result from non atomic operations.
  - Data base updates may result in inconsistent data
  - Resources (e.g. memory) may be overwritten
  - Interrupt handlers might themselves be interrupted
  - Etc

Atomicity in distributed systems

- The appearance of atomicity can be achieved at low latency by the use of an intermediary
- Problem comes if intermediary fails.

Types of fast, in memory intermediaries

- Strictly data storage
  - Memcached
- Distributed coordination systems
  - Zookeeper
  - etcd
  - Consul

Memcached - 1

- Key-value database
  - Key up to 250 bytes
  - Value up to 1 megabyte
- Client – server
- Multiple servers, known to all clients

Memcached - 2

- Data Is divided among servers based on a hash of the key
  - Hashing key yields a server number and that is the server on which the data is stored
- Different clients with the same key can share data

Problems with Memcached

- No failure recovery – if a server fails, data is lost
- Any coordination among clients over a single data item is outside of the scope of memcached.

Distributed coordination systems

- Fault tolerant with high availability
- Manages coordination among clients.
- Operations appear atomic

Locks as a use case

- We will examine distributed coordination systems using locking in Zookeeper as a use case.
- Other use cases are:
  - Leader election
  - Groups membership
  - Synchronization
  - Configuration

Locks

- One technique to give the appearance of atomicity is to lock critical resources.
- A lock is a software construct.
- Associate with each resource a lock bit.
  - If bit is 0, no lock on resource
  - If bit is 1, resource is locked
- In practice, locks have other information in addition to lock bit.

Problem with locks

- Can lead to deadlock – two processes waiting for each other to release critical resources
  - Process one gets a lock on row 1 of a data base
  - Process two gets a lock on row 2.
  - Process one waits for process 2 to release its lock on row 2
  - Process two waits for process 1 to release its lock on row 1
  - No progress.

More problems with locks

- Getting a lock across distributed systems is not an atomic operation.
  - It is possible that while requesting a lock another process can acquire the lock. This can go on for a long time (it is called livelock if there is no possibility of ever acquiring a lock)
- Suppose the virtual machine holding the lock fails. Then the owner of the lock can never release it.

Locks in distributed systems

- Utilizing locks in a distributed system has a more fundamental problem.
- It takes time to send a message (create a lock, for example) from one computer to another.

Synchronizing data

- The general problem is that you want to manage synchronization of data across a distributed set of servers where up to half of the servers can fail.
- Paxos is a family of algorithms that use consensus to manage concurrency. Complicated and difficult to implement.
- An example of the implementation difficulty
  - Choose one server as the leader that keeps the "authoritative" state.
  - Now that server fails. Need to
    - Find a new leader
    - Make sure it is up to data with the authoritative state.

Zookeeper as an example

- Zookeeper provides a guaranteed consistent (mostly) data structure for every instance of a distributed application.
  - Definition of “mostly” is within eventual consistency lag (but this is small).
  - Zookeeper keeps data in memory. This is why it is fast.
  - Limited to 1 Mbyte
- Zookeeper deals with managing failure as well as consistency.
  - Done using consensus algorithm.

Zookeeper API

- All calls return atomic views of state – either succeed or fail. No partial state returned. Writes also are atomic. Either succeed or fail. If they fail, no side effects.

Function	Type
create	write
delete	write
Exists	read
Get children	Read
Get data	Read
Set data	write
+ others	

Lock creation

- Locks have names.
- Client N
  - Create Lock1
    - If success then client owns lock.
    - If failure, then it is placed on waitlist for lock 1 and control is returned to Client N
  - When owner of Lock 1 finishes, it deletes Lock1
  - If there is a waitlist, then those clients are informed about Lock 1 deletion and they try again to create Lock 1
- If Client N fails, Zookeeper will delete Lock 1 and waitlist clients are informed of deletion

Summary

- Failure of VMs in the cloud is something that application programmers must allow for.
- The long tail is an example of what can happen when a VM fails
- Synchronization of data across a distributed system id complicated
  - Timing delays across distributed systems
  - Possible failures of nodes of various types
- Open source systems implement complicated consensus algorithms in a fashion that is relatively easy to use.