Some Lessons Learned From Operating Amazon’s Web Services

Marvin Theimer
Talk Outline

• Brief introduction to AWS
• An analogy: Evolving a Cessna prop-plane into a 747 jumbo jet in-flight
• Issues you will encounter
• Keep it simple
• CAE trade-off: cost-efficient, available, elastic: pick any two
• Don’t ignore your business model
• Semantics of elastic resources
Brief Introduction to AWS

- Elastic Compute Cloud (EC2)
- Elastic block storage service (EBS)
- Virtual Private Clouds (VPC)
- Simple storage service (S3)
- Simple queue service (SQS)
- SimpleDB
- Cloudfront CDN
- Elastic Map-Reduce (EMR)
An Analogy for Building a Successful, Evolving, Highly-Available Service

• Start with a Cessna prop-plane
• 4-9’s availability means you get to land for 52 minutes every year
  – Includes scheduled maintenance
  – Includes refueling
  – Includes crash landings
• Success => growth and evolution => rebuilding the plane in mid-flight
  – Passenger capacity goes from 4-person cabin to 747 jumbo wide-body cabin
  – Support for “scale out” means you add jet engines and remove the propellers while flying
  – Testing and safety inspections for these changes get done in-flight – with passengers – as well

©2009 Amazon Web Services LLC
The Unexpected Happens

- A fuse blows and darkens a set of racks
- Chillers die in a datacenter and a fraction of servers are down
- The electric plug of a rack bursts into flames
- A Telco severs connectivity to a datacenter
- Tornados and lightening strike a datacenter
- A datacenter floods from the roof down
- Simultaneous infant mortality occurs of servers newly-deployed in multiple datacenters
- Power generation doesn’t start because the ambient temperature is too high
- The DNS provider creates a black hole
- Load

©2009 Amazon Web Services LLC
Networking Challenges

• The IP protocol is deeply embedded in systems – you de-facto have to use it
• IP networks can have lost packets, duplicate packets, and corrupted packets
• Even if you use TCP your applications still need to worry about lost packets, duplicate packets, and corrupted packets
• Software (and hardware) bugs can result in consistent loss or corruption of some packets
• You have to be prepared for message storms
• Client software is sometimes written without a notion of backing off on retries
Things You Should Be Able To Do Without Causing Outages

• Adding new hardware
• Deploying a new version of software
• Rolling back to a previous version of software
• Recovering from the absence, loss, or corruption of non-critical data
• Losing a mirror of a DBMS
• Recovering from having lost a mirror of a DBMS
• Losing a host in its fleet
• Losing a datacenter
• Losing network connectivity between data centers
System Resources/Objects Have Lives of Their Own

- Resources/objects in a service may live longer than the accounts used to create them
  - You have to be able to remap them between accounts

- Resources/objects may live longer than versions of the service
  - You have to be able to migrate them forward
  - ... with minimal or no disruption of their use
Downstream Dependencies Fail

• It’s a service-oriented architecture
  – The good news: your service has the ability to keep going even if other services become unavailable
  – The challenge: how to keep going and/or degrade gracefully if you depend on the functionality of downstream services

• Suppose all services are 4-9’s available
  – If a downstream service fails for 52 minutes, how will you meet your own SLA of failing no more than 52 minutes?

• Cascading outages happen
  – If multiple downstream services fail, how will you handle it?
You Must be Prepared to Deal with Data Corruption

• Data corruption happens
  – Hardware can be flakey
  – IO sub-systems can lie
  – Software can be wrong
  – Evolution happens
  – People can screw up

• End-to-end integrity checks are a must
  – Straight-forward data corruption checking
  – How do you know if your system is operating correctly?

• Can your design do fsck in < 52 minutes?
Keep it Simple

• It’s 4AM on Sunday morning and the service has gone down
  – Can you explain the corner cases of your design to the front-line on-call team over the phone?
  – Can you figure out what’s going on in under 52 minutes?

• Simple brute force is sometimes preferable to elegant complexity
  – Eventual consistency considered painful (but sometimes necessary)
  – P2P can be harder to debug than centralized approaches (but may be necessary)
Will Your Design Envelope Scale Far Enough? Do You Understand Your Components Well Enough?

- Cloud computing has global reach
  - Services may grow at an astonishing pace
  - The overall scale is HUGE
- The scale of cloud computing tends to push systems outside their standard design envelopes
  - The rule-of-thumb that you must redesign your system every time it grows by 10x implies you must be prepared to redesign early and often
  - Modern systems use ever-more-sophisticated components
  - Software (and hardware) systems unavoidably have implicit design assumptions built into them
  - When you go outside the design envelope you get to discover where those assumptions no longer hold
  - The result may be data corruption, unexpected performance behaviors, etc.
- It’s 4AM and you have 52 minutes to figure out why things aren’t working as expected…
CAE Trade-Off for Resources

• CAE: cost-efficient, available, elastic
  • If cost is no concern then you can provide highly-available, elastic resources by over-provisioning
  • If you don’t need elasticity (i.e. you know your workload and environment exactly) then you can provide high availability in the most cost-efficient way possible
  • If you don’t need it now then you can provide cost-efficient, elastic resources by making the client wait

• Most everyone wants high availability
• The challenge is how to provide seemingly infinite elasticity at competitive prices
Don’t Ignore the Business Model or Your TCO

- Do you know all the sources of cost? Can you accurately measure them?
- Do you know all the “dimensions of cost” that will be used in pricing? Can you meter them?
- Have you thought about ways the system can be abused?
- How will you resolve billing disputes?
- All these may affect the design of the service in fundamental ways
Elastic Resources: What Boundaries to Expose?

• High availability applications require the notion of independent failure zones → introduce the notion of availability zones (AZ)

• Concurrent applications want bounded, preferably low message latency and high bandwidths → introduce the notion of cluster affinity to an AZ

• The challenges of AZ clustering
  – Clumping effect since everyone will want to be near everyone else
  – Makes elastic scheduling harder

• Fine-tuned applications are the enemy of elasticity
  • Customers will try to divine your intra-AZ topology (co-location on the same rack, etc.)
  • Eventual evolution to different network infrastructures and topologies means you don’t want to expose more than you have to.
Summary and Conclusions

• The unexpected happens: in large systems even extremely rare events occur with a non-negligible frequency; what’s your story on handling them?
• Keep it simple: It’s 4AM and the clock is ticking – can you debug what’s going on in your system?
• Cloud computing is a business: you have to think about cost-efficiency as well as availability and elasticity
©2009 Amazon Web Services LLC.

This presentation is provided for informational purposes only. Amazon Web Services LLC is not responsible for any damages related to the information in this presentation, which is provided “as is” without warranty of any kind, whether express, implied, or statutory. Nothing in this presentation creates any warranties or representations from Amazon Web Services LLC, its affiliates, suppliers, or licensors. This presentation does not modify the applicable terms and conditions governing your use of Amazon Web Services technologies, including the Amazon Web Services website. This presentation represents Amazon Web Services' current product offerings as of the date of issue of this document, which are subject to change without notice.