Fault-Tolerant Computer System Design
ECE 695/CS 590

Putting it All Together

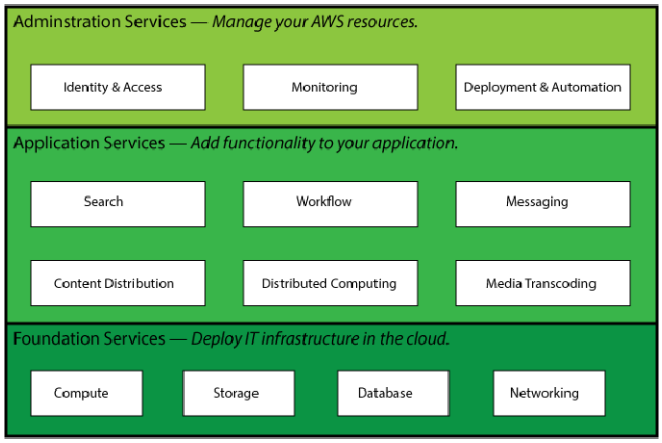
Saurabh Bagchi
ECE/CS
Purdue University

Outline

Looking at some practical systems that integrate multiple techniques that we have learned

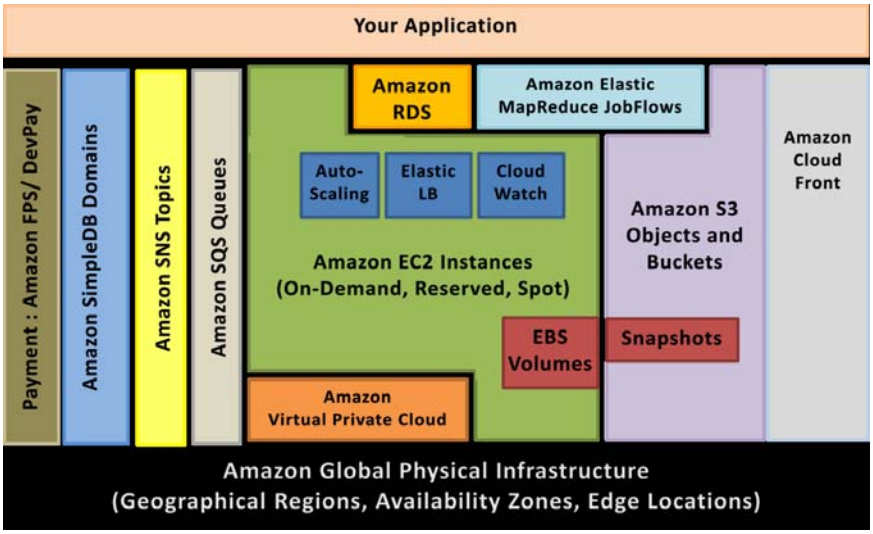
- Amazon Web Services (AWS)
- Hadoop Distributed Processing

Amazon Web Service (AWS): Case Study



- A set of services built in for reliability and security

AWS Ecosystem



AWS Reliability Examples

- Amazon S3 is highly durable and distributed data store
  - With a simple web services interface, you can store and retrieve large amounts of data as *object*s in containers
  - From anywhere on the web using standard HTTP commands
  - Copies of objects can be distributed and cached at 14 *edge locations* around the world by creating a *distribution*
  - Distribution handled using Amazon CloudFront service – a web service for content delivery (static or streaming content)
- Amazon Simple Queue Service (Amazon SQS) is a reliable, highly scalable, hosted distributed queue for storing messages as they travel between computers and application components

Amazon Web Service: Case Study

- Amazon Machine Images (AMIs): Commonly used machine instances from which the user can choose to use as an execution platform; Spare instances can be kept running
- *Amazon Elastic Block Store (Amazon EBS):* Block-level storage volumes for AMIs
  - Durability of EBS is higher than a typical hard drive due to storing data redundantly; Annual failure rate for an EBS volume is 0.1 to 0.5% compared to 4% for a regular hard drive.
  - EBS provides a *snapshot* feature – a backup of the system taken at a specific instance of time. Snapshots are stored in the Amazon S3 to ensure high durability.

Amazon Web Service: Case Study

- Autoscaling and Elastic Load Balancing: Allows EC2 capacity to go up or down as needed by load
  - Example: When # running server instances is below a threshold, launch new server instances
  - Example: Monitor resource utilization of server instances using CloudWatch service; if utilization too high, start up new server instances
  - Example: Distribute incoming traffic across EC2 instances, for load balancing or to route around failed instances
- Regions and Availability zones: Distribute the application geographically in distant data centers.
  - Each geographic location is called a *Region*.
  - Within each Region, there are *Availability Zones*.
  - Availability Zones are distinct locations that are insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region.

AWS Regions and Availability Zones

- **US East (Northern Virginia) Region**
  EC2 Availability Zones: 5
- **US East (Ohio) Region**
  EC2 Availability Zones: 3 (Launched 2016)
- **US West (Oregon) Region**
  EC2 Availability Zones: 3
- **US West (Northern California) Region**
  EC2 Availability Zones: 3
- **AWS GovCloud (US) Region**
  EC2 Availability Zones: 2

Designing for Failure in AWS

- Questions that should be asked wrt hardware failure:

What happens if a node in your system fails? How do you recognize that failure? How do I replace that node? What kind of scenarios do I have to plan for? What are my single points of failure? If a load balancer is sitting in front of an array of application servers, what if that load balancer fails? If there are master and slaves in your architecture, what if the master node fails? How does the failover occur and how is a new slave instantiated and brought into sync with the master?

- Questions that should be asked wrt software failure:

What happens to my application if the dependent services changes its interface? What if downstream service times out or returns an exception? What if the cache keys grow beyond memory limit of an instance?

Designing for Failures in AWS

- Best practices that help dealing with failures:

1. Have a coherent backup and restore strategy for your data and automate it
2. Build process threads that resume on reboot
3. Allow the state of the system to re-sync by reloading messages from queues
4. Keep pre-configured and pre-optimized virtual images to support (2) and (3) on launch/boot
5. Avoid in-memory sessions or stateful user context, move that to data stores.

AWS Specific Fault-Tolerance Techniques

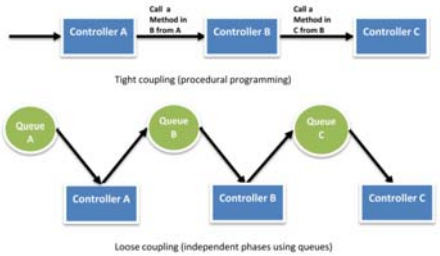
1. Failover gracefully using Elastic IPs: Elastic IP is a static IP that is dynamically re-mappable. You can quickly remap and failover to another set of servers so that your traffic is routed to the new servers. It works great when you want to upgrade from old to new versions or in case of hardware failures
2. Utilize multiple Availability Zones: Availability Zones are conceptually like logical datacenters. By deploying your architecture to multiple availability zones, you can ensure highly availability. Automatically replicate database updates across multiple Availability Zones.
3. Maintain an Amazon Machine Image so that you can restore and clone environments very easily in a different Availability Zone; Maintain multiple Database slaves across Availability Zones and setup hot replication.

AWS Specific Fault-Tolerance Techniques

4. Utilize Amazon CloudWatch (or various real-time open source monitoring tools) to get more visibility and take appropriate actions in case of hardware failure or performance degradation. Setup an Auto scaling group to maintain a fixed fleet size so that it replaces unhealthy Amazon EC2 instances by new ones.
5. Utilize Amazon EBS and set up cron jobs so that incremental snapshots are automatically uploaded to Amazon S3 and data is persisted independent of your instances.
6. Utilize Amazon RDS and set the retention period for backups, so that it can perform automated backups.

Loosely Connected Components

- Catchy principle: *The more loosely coupled the components of the system, the bigger and better it scales.*
- Key: Build components that do not have tight dependencies
  - If one component were to die (fail), sleep (not respond) or remain busy (slow to respond), the other components will continue to work as if no failure is happening
  - Each component interacts asynchronously with the others and treats them as a "black box"



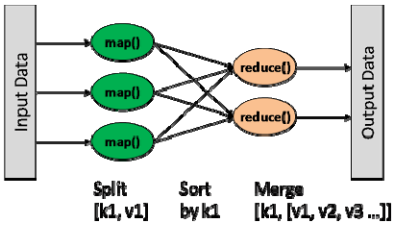
- Use Amazon SQS as buffers between components
- Make your applications as stateless as possible. Store session state outside of component (e.g., in Amazon SimpleDB)

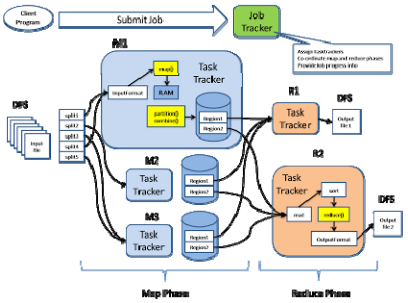
Outline

Looking at some practical systems that integrate multiple techniques that we have learned

- Amazon Web Services (AWS)
- Hadoop Distributed Processing

Hadoop: Case Study





- Runs on a collection of COTS *shared-nothing* servers

Hadoop: Case Study

- Hadoop job has two types of tasks: mappers and reducers.
  - Mappers read the job input data from a distributed file system (HDFS) and produce key-value pairs. These map outputs are stored locally on compute nodes
  - Each reducer processes a particular key range. For this, it copies map outputs from the mappers which produced values with that key (oftentimes all mappers).
  - A reducer writes job output data to HDFS.
- A Task- Tracker (TT) is a Hadoop process running on compute nodes which is responsible for starting and managing tasks locally. A TT has a number of mapper and reducer slots which determine task concurrency.
- A TT communicates regularly with a Job Tracker (JT), a centralized Hadoop component that decides when and where to start tasks.
- JT also runs a speculative execution algorithm which attempts to improve job running time by duplicating under-performing tasks.

Hadoop: Case Study

- Failure cases it worries about: non-responsiveness of a task
  - Can be due to overload of the task or network congestion
- Waits for non-responsive tasks (on the order of 10 minutes) and then re-executes the work of these tasks
- TT sends heartbeat to JT every 3 s. JT declares a TT dead if no heartbeat for 600 s.
  - Then tasks are restarted on a different node
- A reducer is considered faulty if it failed too many times to copy map outputs. This decision is made at the TT.

References

- "Architecting for the cloud: Best Practices," Jinesh Varia (AWS), Jan 2011.
- "On Designing and Deploying Internet-Scale Services," James Hamilton - Windows Live Services Platform, pp. 231-242 of the Proceedings of the 21st Large Installation System Administration Conference (LISA '07)
- Hadoop MapReduce Tutorial at:

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html