Cluster-Based Computing

Challenges in cluster-based architecture

- Many corporations manage *data centers* with a large number of server clusters
  - > 10,000s machines in one data center
    - Commodity linux boxes
  - Handle a large amount of user traffic and data
- Q: What are the challenges in managing/operating machines at this scale?
  - Hardware failures
    - Power and heat issues
    - Main source of failures: power supply, hard drive, network
  - Difficulty of ensuring consistency among nodes
- Failures are unavoidable
  - Q: Assuming 99.9% uptime (9 hour downtime per year), how many machines are down at any point with 10,000 machines?
  - A nightmare for system administrator
    - Need to automate most of maintenance tasks, including initial deployment, synchronize a replacement machine, etc.
  - Very important to have software infrastructure to
    - Manage failed nodes
    - Monitor loads on individual machines
    - Schedule and distribute tasks and data to nodes
  - Example: Kubernetes provides
    - Automatic deployment, scaling, and management of containerized applications
    - Automatic scaling and load balancing of apps based on CPU usage
    - Progressive rollout of application changes
    - Automatic restart of failed, unresponsive nodes
  - Example: MapReduce (or Hadoop) provides
    - Automatic computational task distribution
    - Failure handling
      - Monitor the health of nodes
      - Reassign a task if node does not respond
    - Speed disparity handling

- Monitor progress of a task
- Reassign/execute a backup task if a node is too slow

Parallel Computing Through Map/Reduce

- Q: How to run a computation job on thousands of machines in parallel?
- Q: Do programmers have to explicitly write code for parallelization?
  - data plumbing?
  - task scheduling?
- Q: Any way to "provide" the basic data plumbing and task scheduling? Any good programming model?
- **Example 1**: Log of billions of query. Count frequency of each query

```
Input query log:
    1,time,userid1,ip1,referrer1,query1
    2,time,userid2,ip2,referrer2,query2
    ...

Output query frequency:
    cat 200000
    dog 120000
    ...
```

  - Query log file is likely to be spread over many machines

    [diagram of nodes and the query log chunks within them]

  - Q: How can we do this?
  - Q: How can we parallelize it on thousands of machines?

* Q: Can we process each query log entry independently?

* Q: Where should we run the query extraction task?

* Remark:
  - Network bandwidth is a shared and limited resource
  - VERY IMPORTANT to minimize network bandwidth usage

- Note: this process can be considered as the following two steps:
  1. Map/transform step: (query_log) -> (query, 1)
  2. Reduce/aggregate step: group by *query* and sum up 1s

- **Example 2**: 1 billion pages. build inverted index

```
Input documents:
    1: cat ate dog
    2: dog ate zebra
    ...

Output index:
    cat 1,2,5,10,20
    dog 2,3,8,9
    ...
    zebra 7,9,10,11
```

  - Q: How can we do this?

  - Q: How can we parallelize it on thousands of machines?
    * Q: Can we process each page independently?

    * Q: Where should we place keyword extraction task?

- Note: this process can be considered as the following two steps:
  1. Map/transform step: (docid, content) -> (word1, docid), (word2, docid), ...
  2. Reduce/aggregate step: group by *word* and generate a list of docids
- General observation
  - Many data processing jobs can be done as a sequence of
    1. Map: (k, v) -> (k′, v′), (k′‴, v‴′), ...
    2. Reduce: partition/group by k and "aggregate" v's of the same k
  - Output of map function depends only on the input (k,v), not any other input
    * Each map task can be executed independently of others
  - "Aggregations" on different keys are independent of each other
    * Each reduce task can be executed independently of others
  - If any data processing follows this pattern, they can be parallelized by
    1. Split the input into independent chunks
    2. Run "map" tasks on the chunks in parallel on multiple machines
    3. Partition the output of the map task by the output key
    4. Move data of the same partition to the same node
    5. Run one reduce task per each partition
  - Depending on the application, the exact the map and reduce functions are different
  - Q: Inside "reduce" task, will v's appear in a certain order?
    * Note: reduce function should be agnostic to the order of v's
  - Q: What might be issues when we run map/reduce tasks on thousands of machines?
    * Failed node
    * Slow node

MapReduce

- Programmer provides
  - Map function (k, v) -> (k′, v′)
  - Reduce function (k, [v1, v2, . . .]) -> (k, aggr([v1, v2, . . .]))
- Given the two functions, MapReduce handles the rest
  - split/parse input file into small chunks and assign and run map task on the nodes where the chunks are located
  - partition map output, transfer partitions to reduce nodes sort each partition
  - run the reduce task once a partition data is ready
- Hadoop
  - Open source implementation of GFS and MapReduce
  - Map and reduce functions are implemented by:
    - `Mapper.map(key, value, output, reporter)`
    - `Reducer.reduce(key, value, output, reporter)`
  - Implemented in Java
- Spark
  - Open source cluster computing infrastructure
  - Supports MapReduce and SQL
    - Supports data flow more general than simple MapReduce
  - Supports multiple programming languages including Scala
- Spark example
  - Count words in a document

```
val lines = sc.textFile("input.txt")
val words = lines.flatMap(line => line.split(" "))
val word1s = words.map(word => (word, 1))
val wordCounts = word1s.reduceByKey((a,b) => a+b)
wordCounts.saveAsTextFile("output")
System.exit(0)
```

    - `map`: one output per one input
    - `flatMap`: multiple outputs per one input

- **Important Notes on Programming on Clusters**
  - **DO NOT ASSUME ANYTHING!!!**
    - * Explicitly define failure scenarios and the likelihood of each scenario
      - ▸ Failure WILL happen. plan ahead for it
      - ▸ Make sure your code is thoroughly covered for the likely scenarios
      - ▸ Choose simplicity over generality
  - Minimize state sharing among machines
    - * Decide who wins in case of conflict
  - minimize network bandwidth usage

Starting A New Web Site

- Q: You want to start a new site, called http://cs144.com. How can you do it?
  1. Buy the domain name cs144.com
     - GoDaddy.com, register.com, ... (~$10/year)
  2. Get a "web server" with a public IP and update DNS to the IP
- **Q: How can we obtain a web server?**
  - Set up a physical machine
    a. Buy a machine (~$1,000/PC)
    b. Buy an internet connection from an ISP
       - * Verizon, AT&T, Comcast, ... (~$100/month)
    c. Install OS and necessary software
    d. All maintenance hassles for physical machines and the software
  - "Rent" a machine from a cloud hosting companies
    - * Amazon Web Service, Google Cloud Platform, Windows Azure, ...
- **Q: Exactly what do we rent from a cloud company?**
  1. Infrastructure as a service (IaaS)
     - Rent a "virtual machine" and run your own virtual machine image
       - * e.g., Amazon Elastic Cloud 2, ...
     - No hardware to manage, but all software needs to be managed
  2. Platform as a service (Paas)
     - Rent a common computing platform, including OS, database, application and web servers.

    * Microsoft Azure, Google App Engine, ...
    * LAMP, MEAN, Java Servlet + JSP, Python+Django, ...
  - You maintain your own application
3. Software as a service (SaaS)
  - Rent a fully working "software" over internet
    * Google G Suite, Office 365, Salesforce.com, ...
  - No hardware or software to maintain

- **Amazon Web Services**
  - Amazon EC2 (Elastic Compute Cloud, virtual machine)
  - Amazon Elastic Block Store
  - Amazon S3 (Simple Storage Service, distributed filesystem)
  - Amazon RDS (Relational Database Service)
  - Amazon DynamoDB (NoSQL datastore)
  - Amazon Glacier (low-cost archival storage)
  - Amazon ElastiCache (in-memory object caching)
  - Amazon Elastic Load Balancing
  - Amazon CloudFront (content distribution network)