Cloud Computing: Outline

- Technologies behind cloud computing
  - Data centers
  - Virtualization
  - Programming Framework: Map-reduce
  - Distributed Key-Value Stores
- Some observations about the marketplace

Cloud Computing

- Computing as a “service” rather than a “product”
  - Everything happens in the “cloud”: both storage and computing
  - Personal devices (laptops/tablets) simply interact with the cloud
- Advantages
  - Device agonstic – can seamlessly move from one device to other
  - Efficiency/scalability: programming frameworks allow easy scalability (relatively speaking)
  - Reliability
  - Cost: “pay as you go” allows renting computing resources as needed – much cheaper than building your own systems
Cloud Computing

- Basic ideas have been around for a long time (going back to 1960’s)
  - Mainframes + thin clients (more by necessity)
  - Grid computing a few year ago
  - Peer-to-peer
  - Client-server models
  - …
- But it finally works as we wished for…
  - Why now?… A convergence of several key pieces over the last few years
  - Does it really? … Still many growing pains

First Key: Data Centers

- The key infrastructure piece that enables CC
- Everyone is building them
- Huge amount of work on deciding how to build/design them
Data Centers

- Amazon data centers: Some recent data
  - 8 MW data center can include about 46,000 servers
  - Costs about $88 million to build (just the facility)
  - Power a pretty large portion, but server costs still dominate

“Every day, Amazon Web Services adds enough new capacity to support all of Amazon.com’s global infrastructure through the company’s first 5 years, when it was a $2.76B annual revenue enterprise”

source: James Hamilton Presentation

Data Centers

- Power distribution
  - Almost 11% lost in distribution – starts mattering when the total power consumption is in millions
- Modular and pre-fab designs
  - Fast and economic deployments, built in a factory

source: James Hamilton Presentation
Data Centers

- Networking equipment
  - Very very expensive
  - Bottleneck – forces workload placement restrictions

- Cooling/temperature/energy issues
  - Appropriate placement of vents, inlets etc. a key issue
    - Thermal hotspots often appear and need to worked around
  - Overall cost of cooling is quite high
    - So is the cost of running the computing equipment
      - Both have led to issues in energy-efficient computing
  - Hard to optimize PUE (Power Usage Effectiveness) in small data centers
    - ➔ may lead to very large data centers in near future

source: James Hamilton Presentation

Second Key: Virtualization

- Virtual machines (e.g., running Windows inside a Mac) etc. has been around for a long time
  - Used to be very slow…
  - Only recently became efficient enough to make it a key for CC

- Basic idea: run virtual machines on your servers and sell time on them
  - That’s how Amazon EC2 runs

- Many advantages:
  - Security: virtual machines serves as almost impenetrable boundary
  - Multi-tenancy: can have multiple VMs on the same server
  - Efficiency: replace many underpowered machines with a few high-power machines
Virtualization

- Consumer VM products include VirtualBox, VMWare, Parallels (for Mac) etc…

- Amazon uses “Xen” running on Redhat machines (may be old information)
  - They support both Windows and Linux Virtual Machines

- Some tricky things to keep in mind:
  - Harder to reason about performance (if you care)
  - Identical VMs may deliver somewhat different performance

- Much continuing work on the virtualization technology itself

Third Key: Programming Frameworks

- Third key piece emerged from efforts to “scale out”
  - i.e., distribute work over large numbers of machines (1000’s of machines)

- Parallelism has been around for a long time
  - Both in a single machine, and as a cluster of computers

- But always been considered very hard to program, especially the distributed kind
  - Too many things to keep track of
    - How to parallelize, how to distribute the data, how to handle failures etc etc..

- Google developed MapReduce and BigTable frameworks, and ushered in a new era
  - Cassandra, MongoDB
Programming Frameworks

- **Note the difference between “scale up” and “scale out”**
  - scale up usually refers to using a larger machine – easier to do
  - scale out refers to distributing over a large number of machines

- Even with VMs, I still need to know how to distribute work across multiple VMs
  - Amazon’s largest single instance may not be enough

MapReduce Framework

- **Provides a fairly restricted, but still powerful abstraction for programming**

- **Programmers write a pipeline of functions, called map or reduce**
  - map programs
    - inputs: a list of “records” (record defined arbitrarily – could be images, genomes etc…)
    - output: for each record, produce a set of “(key, value)” pairs
  - reduce programs
    - input: a list of “(key, {values})” grouped together from the mapper
    - output: whatever

- Both can do arbitrary computations on the input data as long as the basic structure is followed
MapReduce Framework

Word Count Example

map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
MapReduce Framework: Word Count

input files

mappers

intermediate files

reducers

output files

(a, 1)
(b, 1)
(a, 1)
(c, 1)
(d, 1)
(b, 1)

(a, 1)
(a, 1)
(c, 1)
(a, 1)
(a, 1)

(a, 8)
(c, 5)

(b, 1)
(d, 1)
(b, 1)
(b, 1)
(d, 1)
(b, 1)

More Efficient Word Count

input files

mappers

intermediate files

reducers

output files

(a, 2)
(b, 2)
(c, 1)
(d, 1)

(a, 2)
(a, 3)
(c, 1)
(c, 5)

(a, 8)
(c, 5)

(b, 1)

(b, 6)
(d, 2)

Called “mapper-side” combiner
MapReduce Framework

- Has been used within Google for:
  - Large-scale machine learning problems
  - Clustering problems for Google News etc..
  - Generating summary reports
  - Large-scale graph computations

- Also replaced the original tools for large-scale indexing
  - i.e., generating the inverted indexes etc.
  - runs as a sequence of 5 to 10 Mapreduce operations

- Hadoop:
  - Open-source implementation of Mapreduce
  - Supports many other technologies as well
  - Very widely used
  - Many startups focusing on providing Hadoop services, different points in the Hadoop/DB space etc…

Bigtable/Key-Value Stores

- MapReduce/Hadoop great for batch processing of data
  - Much ongoing work on efficiency, other programming frameworks (e.g., for graph analytics, scientific applications)

- There is another usecase
  - Very very large-scale web applications that need real-time access with few ms latencies

- Bigtable (open source implementation: HBase)
  - Think of it as a very large distributed hash table
  - Support “put” and “get” operations
    - With some additional support to deal with versions

- Much work on these systems
  - Issues with “consistency” and “performance” quite challenging
Key-Value Stores

- Some Interesting (somewhat old) numbers ([http://highscalability.com](http://highscalability.com))
  - Twitter: 177M tweets sent on 3/1/2011 (nothing special about the date), 572,000 accounts added on 3/12/2011
  - Dropbox: 1M files saved every 15 mins
  - Stackoverflow: 3M page views a day (Redis for caching)
  - Wordnik: 10 million API Requests a Day on MongoDB and Scala
  - Mollom: Killing Over 373 Million Spams at 100 Requests Per Second (Cassandra)
  - Facebook's New Real-time Messaging System: HBase to Store 135+ Billion Messages a Month
  - Reddit: 270 Million Page Views a Month in May 2010 (Memcache)

- How to support such scale?
  - Databases typically not fast enough
  - Facebook aims for 3-5ms response times