Chapter 6

Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism:
Introduction

- Warehouse-scale computer (WSC)
  - Provides Internet services
    - Search, social networking, online maps, video sharing, online shopping, email, cloud computing, etc.
  - Differences with HPC “clusters”:
    - Clusters have higher performance processors and network
    - Clusters emphasize thread-level parallelism, WSCs emphasize request-level parallelism
  - Differences with datacenters:
    - Datacenters consolidate different machines and software into one location
    - Datacenters emphasize virtual machines and hardware heterogeneity in order to serve varied customers
WSC Design Factors (I)

- Cost-performance (i.e., work done/$)
  - A WSC infrastructure can cost $150 M
  - A 10% reduction in capital cost saves $15 M
    - Small savings add up.

- Energy efficiency
  - Affects power distribution and cooling
  - Work done per joule

- Dependability via redundancy:
  - At least 99.99% availability: down less than one hour/year
  - Multiple WSCs

- Network I/O: Inter and intra WSC
- Interactive and batch processing workloads (e.g., MapReduce)
WSC Design Factors (II)

- Ample computational parallelism is not important
  - Most jobs are totally independent (e.g., billions of Web pages from a Web crawl)
  - Software as a Service: millions of independent users
  - “Request-level parallelism”: little need to coordinate or synchronize.

- Operational costs count
  - Power consumption is a primary, not secondary, constraint when designing system
  - Energy, power distribution, and cooling: more than 30% of cost over 10 years.

- Scale and its opportunities and problems
  - Can afford to build customized systems since WSC require volume purchase
  - Flipside: failures.
Outages and Anomalies

<table>
<thead>
<tr>
<th>Approx. number of events in 1st year</th>
<th>Cause</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 or 2</td>
<td>Power utility failures</td>
<td>Lose power to whole WSC; doesn’t bring down WSC if UPS and generators work (generators work about 99% of time).</td>
</tr>
<tr>
<td>4</td>
<td>Cluster upgrades</td>
<td>Planned outage to upgrade infrastructure, many times for evolving networking needs such as recabling, to switch firmware upgrades, and so on. There are about 9 planned cluster outages for every unplanned outage.</td>
</tr>
<tr>
<td>1000s</td>
<td>Hard-drive failures</td>
<td>2% to 10% annual disk failure rate [Pinheiro 2007]</td>
</tr>
<tr>
<td></td>
<td>Slow disks</td>
<td>Still operate, but run 10x to 20x more slowly</td>
</tr>
<tr>
<td></td>
<td>Bad memories</td>
<td>One uncorrectable DRAM error per year [Schroeder et al. 2009]</td>
</tr>
<tr>
<td></td>
<td>Misconfigured machines</td>
<td>Configuration led to ~30% of service disruptions [Barroso and Hölzle 2009]</td>
</tr>
<tr>
<td></td>
<td>Flaky machines</td>
<td>1% of servers reboot more than once a week [Barroso and Hölzle 2009]</td>
</tr>
<tr>
<td>5000</td>
<td>Individual server crashes</td>
<td>Machine reboot, usually takes about 5 minutes</td>
</tr>
</tbody>
</table>

Figure 6.1 List of outages and anomalies with the approximate frequencies of occurrences in the first year of a new cluster of 2400 servers. We label what Google calls a cluster an array; see Figure 6.5. (Based on Barroso [2010].)
Outages and Anomalies Example

Calculate the availability of a service running the 2400 servers of the previous table. Assume:

- service cannot tolerate software of hardware failures (unrealistic)
- Reboot time = 5 minutes
- Hardware repair time = 1 hour
- Ignore slow disks
- All events have equal probability

Hours outage = (4+250+250+250)x1hr+(250+5000)x5min
Availability = (365x24-1192)/8760=86%
Prgrm’g Models and Workloads

- Batch processing framework: MapReduce
  - **Map:** applies a programmer-supplied function to each logical input record
    - Runs on thousands of computers
    - Provides new set of key-value pairs as intermediate values
  - **Reduce:** collapses values using another programmer-supplied function
### Popularity of MapReduce at Google

<table>
<thead>
<tr>
<th></th>
<th>Aug-04</th>
<th>Mar-06</th>
<th>Sep-07</th>
<th>Sep-09</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of MapReduce jobs</strong></td>
<td>29,000</td>
<td>171,000</td>
<td>2,217,000</td>
<td>3,467,000</td>
</tr>
<tr>
<td><strong>Average completion time (seconds)</strong></td>
<td>634</td>
<td>874</td>
<td>395</td>
<td>475</td>
</tr>
<tr>
<td><strong>Server years used</strong></td>
<td>217</td>
<td>2002</td>
<td>11,081</td>
<td>25,562</td>
</tr>
<tr>
<td><strong>Input data read (terabytes)</strong></td>
<td>3288</td>
<td>52,254</td>
<td>403,152</td>
<td>544,130</td>
</tr>
<tr>
<td><strong>Intermediate data (terabytes)</strong></td>
<td>758</td>
<td>6743</td>
<td>34,774</td>
<td>90,120</td>
</tr>
<tr>
<td><strong>Output data written (terabytes)</strong></td>
<td>193</td>
<td>2970</td>
<td>14,018</td>
<td>57,520</td>
</tr>
<tr>
<td><strong>Average number of servers per job</strong></td>
<td>157</td>
<td>268</td>
<td>394</td>
<td>488</td>
</tr>
</tbody>
</table>

**Figure 6.2** Annual MapReduce usage at Google over time. Over five years the number of MapReduce jobs increased by a factor of 100 and the average number of servers per job increased by a factor of 3. In the last two years the increases were factors of 1.6 and 1.2, respectively [Dean 2009]. Figure 6.16 on page 459 estimates that running the 2009 workload on Amazon’s cloud computing service EC2 would cost $133M.
Big Data: Use for MapReduce

- Size of the data becomes part of the computational solution.
- Complex structured and unstructured data.
- Heterogeneous and continuously generated data.
- Data generated from a variety of sources: sensors, cameras, scientific instruments, records of transactions, etc.
- Applications: intelligence, health-care, business analytics, disease prevention, crime prevention, etc.
Big Data

- The federal government has announced a $200 million/yr Big Data initiative for research and development into technologies to “access, store, visualize, massive and complicated data sets.”

- Paradigm for BigData computation:
  - MapReduce (introduced by Google): data and processing parallelism
  - Hadoop: open source implementation of MapReduce.
Prgrm’g Models and Workloads

- MapReduce runtime environment schedules map and reduce task to WSC nodes

- Availability:
  - Use replicas of data across different servers
  - Use relaxed consistency:
    - No need for all replicas to always agree

- Workload demands
  - Often vary considerably
Hadoop jobs consist of two types of tasks:
- Map and Reduce tasks.
- Reduce tasks can only start when all map tasks have finished.

Hadoop cluster:
- Compute nodes: where map and reduce tasks execute
- Each compute node has a number of slots for the execution of map and reduce tasks.
- Each compute node has a TaskTracker daemon that monitors the execution of map and reduce tasks.
Hadoop Distributed File System (HDFS) / MapReduce Framework

- Hadoop Head Node
  - Namenode
  - Secondary Namenode
  - Jobtracker

- Hadoop Compute Nodes
  - Datanode
  - Tasktracker
Hadoop Jobtracker/Tasktracker

Jobtracker Job Queue - Tasks

Tasktrackers

Job tasks

Tasktracker slots
Goal: count the number of occurrences of each word in a large file.

Approach:

- File is broken down into a large number of chunks.
- Each map task takes an input line from the file and breaks it into words and emits a (key, value) pair consisting of the word and a value of “1”.
- The reduce tasks add the number of “1”s for each word and output the word and the sum.
Map/Shuffle/Reduce Data Flow - Wordcount

Input File

Map Task 1
AA BB CC
DD EE AA

Map Task 2
BB EE CC
AA CC DD

Map Task 3
EE DD AA
BB BB CC

Map Task 4
CC DD EE
FF EE FF

Map Task 5
CC DD EE
FF EE FF

Map Output 1
AA/1 BB/1 CC/1
DD/1 EE/1 AA/1

Map Output 2
BB/1 EE/1
CC/1 AA/1
CC/1 DD/1

Map Output 3
EE/1 DD/1
AA/1 BB/1
BB/1 CC/1

Map Output 4
CC/1 DD/1
EE/1 FF/1
EE/1 FF/1

Map Output 5
CC/1 DD/1
EE/1 FF/1
EE/1 FF/1

Shuffle Phase

Reduce Task 1
AA/[1,1,1,1]
BB/[1,1,1,1]
CC/[1,1,1,1,1,1]

Reduce Task 2
DD/[1,1,1,1,1,1]
EE/[1,1,1,1,1,1]
FF/[1,1,1,1,1,1]

Output File
AA = 4
BB = 4
CC = 6
DD = 5
EE = 5
FF = 4
WordCount Example: Map task

WordCount Map Method

IntWritable one = new IntWritable(1);
Text word = new Text();
public void map (Object key, Text value, Context context) throws .... {
    StringTokenizer itr =
        new StringTokenizer (value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}
WordCount Example: Reduce task

WordCount Reduce Method

```
IntWritable result = new IntWritable();
public void reduce(Text key, 
    Iterable<IntWritable> values, 
    Context context) throws ...... {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}
```
MapReduce Model

- $M$: number of map tasks of a job
- $n$: total number of map slots in the worker nodes
- $c$: number of compute nodes

Assume that the $n$ slots are equally distributed among the $c$ compute nodes: $n = k \times c$, where $k$ is the number of map slots per node.
MapReduce Model

\[ n = k \times c \text{ concurrent tasks} \]
Key observation: all $k$ tasks that run concurrently in a node compete for CPU and I/O resources => queuing has to be considered.
MapReduce Model

\[ n' = \text{Mod}(M, n) = M - \left\lfloor \frac{M}{n} \right\rfloor \times n \]

Example:

A job has 54 map tasks and there are 4 compute nodes, each with 2 slots.

\[ M = 54; \quad c = 4; \quad k = 2; \quad n = 8 \]

\[ n' = \text{Mod} (54, 8) = 6 \]
Experimental Parameters

- Size of input file: 5 MB
- Number of map slots set by varying the input split size between 100K and 600K:
  - E.g., Split size of 200 KB => 26 map slots
- No reduce tasks
Avg. execution time vs. no. slots for 54 map tasks: experiments
Avg. execution time vs. no. slots for 54 map tasks: experiments

Execution time decreases as task parallelism increases then increases as node contention increases.
Concurrency Results

Concurrency slightly lower than number of slots due to time TaskTracker takes to detect end of a task and start a new one.

Concurrency = 0.9273 * No. Slots
R² = 0.999
Execution time vs. no. slots for different map tasks

![Graph showing execution time vs. number of map slots for different map tasks.]
Execution time vs. no. slots for 54 map tasks: experiment, contention and no-contention models
Computer Architecture of WSC

- WSC often use a hierarchy of networks for interconnection
- Each 19” rack holds 48 1U servers connected to a rack switch
- Rack switches are uplinked to switch higher in hierarchy
  - Uplink has $48/n$ times lower bandwidth, where $n$ = # of uplink ports
    - “Oversubscription”
  - Goal is to maximize locality of communication relative to the rack
Storage

- Storage options:
  - Use disks inside the servers, or
  - Network attached storage through Infiniband

- WSCs generally rely on local disks
- Google File System (GFS) uses local disks and maintains at least three replicas
Array Switch

- Switch that connects an array of racks
  - Array switch should have 10 X the bisection bandwidth\* of a rack switch
  - Cost of $n$-port switch grows as $n^2$
  - Often utilize content addressable memory chips and FPGAs to support high-speed packet inspection.

\* worst-case internal bandwidth
WSC Memory Hierarchy

- Servers can access DRAM and disks on other servers using a NUMA-style interface

<table>
<thead>
<tr>
<th></th>
<th>Local</th>
<th>Rack</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM latency (μs)</td>
<td>0.1</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>Disk latency (μs)</td>
<td>10,000</td>
<td>11,000</td>
<td>12,000</td>
</tr>
<tr>
<td>DRAM bandwidth (MB/sec)</td>
<td>20,000</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Disk bandwidth (MB/sec)</td>
<td>200</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>DRAM capacity (GB)</td>
<td>16</td>
<td>1,040</td>
<td>31,200</td>
</tr>
<tr>
<td>Disk capacity (GB)</td>
<td>2000</td>
<td>160,000</td>
<td>4,800,000</td>
</tr>
</tbody>
</table>
WSC Memory Hierarchy – Example 1

- What is the average latency assuming that 90% of accesses are local to the server, 9% are outside the server but local to the rack, and 1% are outside the rack but within the array?

\[(90\% \times 0.1) + (9\% \times 100) + (1\% \times 300) = 12.09 \text{ msec}\]
How long does it take to transfer 1000MB between disks within the server, between servers in the rack, and between servers in different racks of an array?

- Within server: \( \frac{1000}{200} = 5 \text{ sec} \)
- Within rack: \( \frac{1000}{100} = 10 \text{ sec} \)
- Within array: \( \frac{1000}{10} = 100 \text{ sec} \)

<table>
<thead>
<tr>
<th></th>
<th>Local</th>
<th>Rack</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM latency (microseconds)</td>
<td>0.1</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>Disk latency (microseconds)</td>
<td>10,000</td>
<td>11,000</td>
<td>12,000</td>
</tr>
<tr>
<td>DRAM bandwidth (MB/sec)</td>
<td>20,000</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Disk bandwidth (MB/sec)</td>
<td>200</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>DRAM capacity (GB)</td>
<td>16</td>
<td>1,040</td>
<td>31,200</td>
</tr>
<tr>
<td>Disk capacity (GB)</td>
<td>2000</td>
<td>160,000</td>
<td>4,800,000</td>
</tr>
</tbody>
</table>
Infrastructure and Costs of WSC

- **Location of WSC**
  - Proximity to Internet backbones, electricity cost, property tax rates, low risk from earthquakes, floods, and hurricanes

- **Power distribution**
  - 89% efficiency
Infrastructure and Costs of WSC

- Cooling
  - Air conditioning used to cool server room
  - 64 F – 71 F
    - Keep temperature higher (closer to 71 F)
  - Cooling towers can also be used
    - Minimum temperature is “wet bulb temperature”
Infrastructure and Costs of WSC

- Cooling system also uses water (evaporation and spills)
  - E.g. 70,000 to 200,000 gallons per day for an 8 MW facility

- Power cost breakdown:
  - Chillers: 30-50% of the power used by the IT equipment
  - Air conditioning: 10-20% of the IT power, mostly due to fans

- How many servers can a WSC support?
  - Each server:
    - “Nameplate power rating” gives maximum power consumption
    - To get actual, measure power under actual workloads
  - Oversubscribe cumulative server power by 40%, but monitor power closely
Measuring Efficiency of a WSC

- Power Utilization Effectiveness (PUE)
  - \[ \text{PUE} = \frac{\text{Total facility power}}{\text{IT equipment power}} \]
  - Median PUE on 2006 study was 1.69

- Performance
  - **Latency** is an important metric because it is seen by users.
  - Bing study: users will use search less as response time increases.
  - Service Level Objectives (SLOs)/Service Level Agreements (SLAs)
    - E.g., 99% of requests must be below 100 ms
Negative Impact of Delays at Bing

<table>
<thead>
<tr>
<th>Server delay (ms)</th>
<th>Increased time to next click (ms)</th>
<th>Queries/ user</th>
<th>Any clicks/ user</th>
<th>User satisfaction</th>
<th>Revenue/ user</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
<td>--</td>
<td>--</td>
<td>--0.3%</td>
<td>--0.4%</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>--</td>
<td>--</td>
<td>--1.0%</td>
<td>--0.9%</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>--0.7%</td>
<td>--1.9%</td>
<td>--1.6%</td>
<td>--2.8%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>--1.8%</td>
<td>--4.4%</td>
<td>--3.8%</td>
<td>--4.3%</td>
</tr>
</tbody>
</table>

Figure 6.12 Negative impact of delays at Bing search server on user behavior Schurman and Brutlag [2009].
Cost of a WSC

- Capital expenditures (CAPEX)
  - Cost to build a WSC
  - Example:
    - 8 MW facility
    - 46,000 servers
    - $88 M for facility
    - $67M for servers
    - $13M for networking equipment
    - PUE: 1.45

- Operational expenditures (OPEX)
  - Cost to operate a WSC
Cloud Computing

- WSCs offer economies of scale that cannot be achieved with a datacenter:
  - 5.7 times reduction in storage costs
  - 7.1 times reduction in administrative costs
  - 7.3 times reduction in networking costs
- This has given rise to cloud services such as Amazon Web Services
  - “Utility Computing”
  - Based on using open source virtual machine and operating system software
What is Cloud Computing?

- A modality of computing characterized by *on demand* availability of resources in a dynamic and scalable fashion.
  - resource = infrastructure, platforms, software, services, or storage.
- The *cloud provider* is responsible to make the resources available on demand to the *cloud users*.
  - the cloud provider must manage its resources in an efficient way so that the user needs can be met when needed at the desired QoS level.
The NIST Cloud Definition Framework
http://csrc.nist.gov/groups/SNS/cloud-computing/

- Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

- This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.
Figure 2: The Conceptual Reference Model
### Is Cloud Computing Analogous to an Electric Grid?

<table>
<thead>
<tr>
<th>Consumers use electric energy on-demand according to their needs and pay based on their consumption.</th>
<th>Cloud computing users use resources on demand according to their needs and pay based on their consumption.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power utilities have to be able to dynamically determine how to match demand and supply.</td>
<td>Cloud computing providers have to be able to dynamically determine how to match demand and supply.</td>
</tr>
<tr>
<td>The product delivered by the power grid is homogeneous (e.g., 110 V of alternating current at 60 Hz).</td>
<td>Clouds offer a variety of resources on demand.</td>
</tr>
<tr>
<td>One can plug any appliance to the power grid and it will work seamlessly as long as it conforms to a very simple specification of voltage and frequency.</td>
<td>The APIs offered by cloud providers are not standardized and may be very complicated in many cases.</td>
</tr>
</tbody>
</table>
Advantages of Cloud Computing

- Pay as you go.

- No need to provision for peak loads.

- Time to market.

- Consistent performance and availability.
Potential Drawbacks of Cloud Computing

- Privacy and security.
- External dependency for mission critical applications.
- Disaster recovery.
- Monitoring and Enforcement of SLAs.
Example: PlanetLab - www.planet-lab.org
Examples: PlanetLab

- 550 active sites with 1170 nodes scattered over 40 countries.
- Shared resources include CPU cycles, storage, and memory.
- PlanetLab Central API allow users to create automated scripts to monitor node availability. See http://www.planet-lab.org/doc/plc_api
Parallel Computation of $\pi$ Using Planetlab

Area of a circle: $\pi r^2$

Area of a quadrant when $r = 1$: $\pi/4$
Parallel Computation of $\pi$ Using Planetlab

Area of a circle: $\pi r^2$
Area of a quadrant when $r = 1$: $\pi/4$

Area of a quadrant $\approx$ number of random points in quadrant divided by total number of random points.

Node 1 . . . Node n

$m/n$ points  $m/n$ points
Parallel Computation of \( \pi \) Using Planetlab

1. (At the master node). Send \( m \) and \( n \) to all \( n \) slave nodes.

2. (At each slave node \( i, i = 1, \ldots, n \)).
   \( \text{NumPointsInQuadrant}_i \leftarrow 0. \)

3. (At each slave node \( i, i = 1, \ldots, n \)). Repeat (a) and (b) \( m/n \) times.

   (a) Randomly select a point \((x, y)\) such that \( x \) and \( y \) are random numbers uniformly distributed in \([0,1]\).

   (b) If \( \sqrt{x^2 + y^2} \leq 1 \) then
       \( \text{NumPointsInQuadrant}_i \leftarrow \text{NumPointsInQuadrant}_i + 1. \)

4. (At each slave node \( i, i = 1, \ldots, n \)).
   Send \( \text{NumberPointsInQuadrant}_i \) to the master node.

5. (At the master node).
   \( \pi \leftarrow 4 \times \sum_{i=1}^{n} \text{NumPointsInQuadrant}_i / m. \)
Parallel Computation of $\pi$ Using Planetlab

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Location</th>
<th>Participation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP1</td>
<td>Univ. Pennsylvania</td>
<td>1-10</td>
</tr>
<tr>
<td>UP2</td>
<td>Univ. Pennsylvania</td>
<td>2-10</td>
</tr>
<tr>
<td>GT1</td>
<td>Georgetown University</td>
<td>3-10</td>
</tr>
<tr>
<td>GT2</td>
<td>Georgetown University</td>
<td>4-10</td>
</tr>
<tr>
<td>GMU3</td>
<td>George Mason University</td>
<td>5-10</td>
</tr>
<tr>
<td>GMU4</td>
<td>George Mason University</td>
<td>6-10</td>
</tr>
<tr>
<td>CT1</td>
<td>Caltech</td>
<td>7-10</td>
</tr>
<tr>
<td>CT2</td>
<td>Caltech</td>
<td>8-10</td>
</tr>
<tr>
<td>CN4</td>
<td>Cornell University</td>
<td>9-10</td>
</tr>
<tr>
<td>VT</td>
<td>Virginia Tech</td>
<td>10</td>
</tr>
</tbody>
</table>

All nodes: 2 Intel Core 2 Duo E6550 Processor @ 2.44 GHz with 3.44GB memory.
Parallel Computation of $\pi$ Using Planetlab

$m = 1$ billion

Execution Time (msec) vs. No. of Nodes

$\text{ExecTime} = 124828 \times \text{(No. Nodes)}^{0.7349}$

$R^2 = 0.9055$
Parallel Computation of $\pi$ Using Planetlab

\[ S = \frac{E_1}{E_n} \]

$m = 1$ billion
Example: Amazon’s Elastic Computing Cloud (EC2)

- http://aws.amazon.com/ec2/
- Virtual site farm
- Users request the number and type of compute instances they need:
  - Standard
  - High-memory
  - High-CPU.
- Payment: by instance-hour
- One EC2 compute unit provides the equivalent of the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.
Examples: Amazon’s Elastic Computing Cloud (EC2)

- EC2’s **Auto Scaling** allows users to determine when to scale up or down their EC2 usage according to user-defined conditions.
  - Saves money

- EC2’s **CloudWatch** aggregates and reports metrics for CPU utilization, data transfer, and disk usage and activity for each EC2 instance.
Amazon’s EC2 Compute Instances

First Generation

First generation (M1) Standard instances provide customers with a balanced set of resources and a low cost platform that is well suited for a wide variety of applications.

**M1 Small Instance** - default

- 1.7 GB memory
- 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
- 160 GB instance storage
- 32-bit or 64-bit platform
- I/O Performance: Moderate
- EBS-Optimized Available: No
- API name: m1.small

**M1 Medium Instance**

- 3.75 GB memory
- 2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)
- 410 GB instance storage
- 32-bit or 64-bit platform
- I/O Performance: Moderate
- EBS-Optimized Available: No
- API name: m1.medium

**M1 Large Instance**

- 7.5 GB memory
- 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
- 850 GB instance storage
- 64-bit platform
- I/O Performance: Moderate
- EBS-Optimized Available: 500 Mbps
- API name: m1.large

**M1 Extra Large Instance**

- 15 GB memory
- 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each)
- 1,690 GB instance storage
- 64-bit platform
- I/O Performance: High
- EBS-Optimized Available: 1000 Mbps
- API name: m1.xlarge
## Amazon’s EC2 Pricing

<table>
<thead>
<tr>
<th>Instance</th>
<th>Per hour</th>
<th>Ratio to small</th>
<th>Compute units</th>
<th>Virtual cores</th>
<th>Compute units/core</th>
<th>Memory (GB)</th>
<th>Disk (GB)</th>
<th>Address size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro</td>
<td>$0.020</td>
<td>0.5–2.0</td>
<td>0.5–2.0</td>
<td>1</td>
<td>0.5–2.0</td>
<td>0.6</td>
<td>EBS</td>
<td>32/64 bit</td>
</tr>
<tr>
<td>Standard Small</td>
<td>$0.085</td>
<td>1.0</td>
<td>1.0</td>
<td>1</td>
<td>1.00</td>
<td>1.7</td>
<td>160</td>
<td>32 bit</td>
</tr>
<tr>
<td>Standard Large</td>
<td>$0.340</td>
<td>4.0</td>
<td>4.0</td>
<td>2</td>
<td>2.00</td>
<td>7.5</td>
<td>850</td>
<td>64 bit</td>
</tr>
<tr>
<td>Standard Extra Large</td>
<td>$0.680</td>
<td>8.0</td>
<td>8.0</td>
<td>4</td>
<td>2.00</td>
<td>15.0</td>
<td>1690</td>
<td>64 bit</td>
</tr>
<tr>
<td>High-Memory Extra Large</td>
<td>$0.500</td>
<td>5.9</td>
<td>6.5</td>
<td>2</td>
<td>3.25</td>
<td>17.1</td>
<td>420</td>
<td>64 bit</td>
</tr>
<tr>
<td>High-Memory Double Extra Large</td>
<td>$1.000</td>
<td>11.8</td>
<td>13.0</td>
<td>4</td>
<td>3.25</td>
<td>34.2</td>
<td>850</td>
<td>64 bit</td>
</tr>
<tr>
<td>High-Memory Quadruple Extra Large</td>
<td>$2.000</td>
<td>23.5</td>
<td>26.0</td>
<td>8</td>
<td>3.25</td>
<td>68.4</td>
<td>1690</td>
<td>64 bit</td>
</tr>
<tr>
<td>High-CPU Medium</td>
<td>$0.170</td>
<td>2.0</td>
<td>5.0</td>
<td>2</td>
<td>2.50</td>
<td>1.7</td>
<td>350</td>
<td>32 bit</td>
</tr>
<tr>
<td>High-CPU Extra Large</td>
<td>$0.680</td>
<td>8.0</td>
<td>20.0</td>
<td>8</td>
<td>2.50</td>
<td>7.0</td>
<td>1690</td>
<td>64 bit</td>
</tr>
<tr>
<td>Cluster Quadruple Extra Large</td>
<td>$1.600</td>
<td>18.8</td>
<td>33.5</td>
<td>8</td>
<td>4.20</td>
<td>23.0</td>
<td>1690</td>
<td>64 bit</td>
</tr>
</tbody>
</table>

Figure 6.15 Price and characteristics of on-demand EC2 instances in the United States in the Virginia region in dollars per hour as of March 2020.
MapReduce at Amazon’s EC2

- Short video on using Amazon’s EC2 to run MapReduce jobs.

- [http://s3.amazonaws.com/awsVideos/AmazonElasticMapReduce/AmazonElasticMapReduce.html](http://s3.amazonaws.com/awsVideos/AmazonElasticMapReduce/AmazonElasticMapReduce.html)
Examples: Google’s App Engine

- Web applications can be deployed on Google’s infrastructures.
- Applications can run in Java or Python run-time environments.
- Free startup: all applications can use up to 500 MB of storage and enough CPU and bandwidth to support an efficient app serving around 5 million page views a month for free.
  - After that, pay according to resource usage.
Examples: Google’s App Engine

- App Engine provides a powerful distributed data storage service that features a query engine and transactions.

- Applications may include:
  - dynamic web serving
  - persistent storage
  - automatic scaling and load balancing
  - user authentication
  - task queues
  - scheduled tasks
Capacity Planning for the Cloud: from the consumer’s point of view

- Problems for consumers:
  - How to select SLAs for various QoS metrics in a way that maximizes a utility function for the consumer subject to cost-constraints?
Capacity Planning for the Cloud: from the consumer’s point of view

- \( \text{SLA}_r \): SLA (in seconds) on the average response time per transaction.
- \( \text{SLA}_x \): SLA (in tps) on the transaction throughput.
- \( \text{SLA}_a \): SLA on the cloud availability.
- \( C_r(\text{SLA}_r) \): per transaction cost (in cents) when the negotiated response time SLA is \( \text{SLA}_r \).
- \( C_x(\text{SLA}_x) \): per transaction cost (in cents) when the negotiated throughput SLA is \( \text{SLA}_x \).
- \( C_a(\text{SLA}_a) \): per transaction cost (in cents) when the negotiated availability SLA is \( \text{SLA}_a \).
Capacity Planning for the Cloud: from the consumer’s point of view

- $U$: global utility. The global utility function is composed of terms that represent the utility for various metrics such as response time, throughput, and availability. The utility is a dimensionless number in the [0,1] range.

- $w_r, w_x, w_a$: weights associated to response time, throughput, and availability, respectively, used to compute the global utility. $w_r + w_x + w_a = 1$. 
Capacity Planning for the Cloud: from the consumer’s point of view

\[ C_r(\text{SLA}_r) = \alpha_r e^{-\beta_r} \text{SLA}_r \]
\[ C_x(\text{SLA}_x) = \alpha_x \text{SLA}_x \]
\[ C_a(\text{SLA}_a) = e^{\beta_a} \text{SLA}_a - e^{0.9\beta_a}, \quad \text{SLA}_a \geq 0.9. \]

\[ U = w_r \frac{2.0 e^{-\text{SLA}_r}}{1 + e^{-\text{SLA}_r}} + w_x (1 - e^{-0.1 \text{SLA}_x}) + w_a (10 \text{SLA}_a - 9). \]
Capacity Planning for the Cloud: from the consumer’s point of view

**Response time Utility Function**

**Throughput Utility Function**

**Availability Utility Function**
Capacity Planning for the Cloud: from the consumer’s point of view

\[
\begin{align*}
\text{maximize } U &= f(SLA_r, SLA_x, SLA_a) \\
\text{subject to} \\
\gamma_r^{\text{min}} &\leq SLA_r \leq \gamma_r^{\text{max}} \\
\gamma_x^{\text{min}} &\leq SLA_x \leq \gamma_x^{\text{max}} \\
\gamma_a^{\text{min}} &\leq SLA_a \leq \gamma_a^{\text{max}} \\
C_r(SLA_r) + C_x(SLA_x) + C_a(SLA_a) &\leq C_{\text{max}}
\end{align*}
\]
Capacity Planning for the Cloud: from the consumer’s point of view

\[
\begin{align*}
\text{maximize } U &= f(\text{SLA}_r, \text{SLA}_x, \text{SLA}_a) \\
\text{subject to } \\
\gamma_r^{\min} & \leq \text{SLA}_r \leq \gamma_r^{\max} \\
\gamma_x^{\min} & \leq \text{SLA}_x \leq \gamma_x^{\max} \\
\gamma_a^{\min} & \leq \text{SLA}_a \leq \gamma_a^{\max} \\
C_r(\text{SLA}_r) + C_x(\text{SLA}_x) + C_a(\text{SLA}_a) & \leq C_{\max}
\end{align*}
\]

Solvers:
- **NEOS**: [http://neos.mcs.anl.gov](http://neos.mcs.anl.gov)
- **MS Excel’s Solver** (see Tools menu)
Capacity Planning for the Cloud: from the consumer’s point of view

\[
\begin{array}{|c|c|c|c|c|}
\hline
C_{\text{max}} & \text{Utility} & SLA_r & SLA_x & SLA_a \\
\hline
0.70 & 0.641 & 1.000 & 5.625 & 0.999 \\
0.60 & 0.438 & 3.543 & 5.000 & 0.999 \\
0.55 & 0.366 & 4.000 & 5.000 & 0.978 \\
0.50 & 0.279 & 4.000 & 5.000 & 0.949 \\
\hline
\end{array}
\]

\[w_r = 0.4; \ w_x = 0.3; \ w_a = 0.3\]
Capacity Planning for the Cloud: from the provider’s point of view

- Providers have to deal with:
  - Large and complex infrastructures
  - Hard to predict and time-varying workloads

- Providers need to implement autonomic computing techniques that are capable to dynamically shift resources without human intervention to cope with negotiated SLAs.