CS144 Notes: Scalability

Capacity planning

- Q: How many requests can a machine handle?
  - really depends on your application

  - possible bottlenecks
    * disk/DB IO: disk 100-500MB/sec, 5-10ms avg seek (or ~100K IO for SSD)
    * network: 100M-1Gbps
    * CPU/memory: 3Ghz

  - Estimation of workload capacity
    * static content:
      - Q: 10KB per request, 30MB disk io, 1Gbps network, how many requests/sec?
        * Disk → memory → network diagram

        disk → memory: sequential (100MB/10KB) vs random (1sec/10ms)
        memory → network: 100MB/10KB

      - a high performance Web server can easily handle 5,000 req/sec/cpu
        * apache, lighttpd, ...
      - main bottleneck is mostly disk/network io

    * dynamic content:
      - depends on the complexity of application
      - rule of thumb: 10 request/sec/CPU
        * assuming reasonably simple application logic
        * no ssl, no video/image encoding, ...
      - cpu/context switch/io can be bottleneck

- Tools for Profiling
  - CPU/process
    * top: load avg: # processes in running or runnable state
    * ps: common options: axl
    * pstree
  - disk io
    * iostat

        * bonnie or bonnie++ or iozone
    - network io
        * netstat: common options: -i or -s
    - memory
        * free -m, ps axl, vmstat, memstat
    - DNS look up often causes lots of problem
        * disable reverse DNS lookup if possible
    - code profiling
        * java: Eclipse TPTP (test & performance tools platform)
            - Java performance tuning - O'Reilly
        * php: xdebug
            - consider opcode caching for large script - zend, apc
        * perl: Devel::DProfPP
        * a good first step to identify bottleneck

- Initial low cost PC cannot handle user traffic. How can we scale a Web site as it grows?
    - scale up: buy a larger, more expensive server
    - scale out: add more machines
    - Q: Pro/cons of scale up/out?
        * scale-up used to be the preferred approach until mid 90's but more companies use scale-out approach these days

Scaling Web applications

- **Typical Web server architecture**

Transport encryption (SSL)
HTTP server (apache)
Application server (tomcat)
Persistence/Storage layer (MySQL)

- **Q: How to scale out a Web site using cluster?**
  - **Q: Scaling out each layer?**
    * encryption layer? http layer? application logic layer?
    * persistence/database? more discussion later

  - **load balancer (TCP NAT request distributor)**
    * Hardware: Foundary Network ServerIron, Cisco LocalDirector, …
    * Software: apache mod_backhand, apache mod_jk2, …
    * DNS round robin
- **Q: How can we scale database once the limit is reached?**
  - scenario 1. Global read only data (online map, yellow pages)?
    * Q: 30 IOs/sec/machine. 3 read IOs/request. How many requests per machine?

    * Q: How can we scale if we get 20 requests/sec?

    * Remark: no DB synchronization problem
  - scenario 2. Local read and write. All user data is local. No global sharing of data (Web mail, online bank, ...)?
    * Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/session How many sessions per machine?

    * Q: How can we scale to deal with 20 sessions? Does replication help?

    * Remark: again, no DB synchronization problem
  - scenario 3. Global read/write. writes are globally visible (online auction, social network)
    * Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/session. How many sessions per machine?

        * Q: How can we scale to deal with 20 sessions? replication?

        * Q: Maximum # of sessions that can be supported using replication?

        * Q: partitioning?

        * Remark:
            - eventually write requests saturate the DB
            - scaling out DB is VERY CHALLENGING and requires careful analysis/design
            - many companies buy larger machine to scale DB for critical data

- **General remarks on scaling out**
    - CPU is rarely a bottleneck and is very easy to scale
    - after reasonable optimization, DBMS/storage is often the main bottleneck
        * two basic approaches for DB scaling: replication and partitioning
        * design your database carefully
        * identify early on how you will cache/replicate/partition your DBMS

- **Capacity planning**
    - characterize the workload:
        * req/sec, res util/req
            - measure resource utilization from your workload

    - set your min acceptable service requirement

    - remember: "premature optimization is the root of all evil" - Donald Knuth
        * do not optimize based on your "guess".
        * MEASURE THINGS from your workload first!!!
        * do not optimize unless you are sure it is important

Large-Scale Distributed file system

- Q: What if we have too much data to store in a single machine?

- Q: How can we create one big filesystem over a cluster of machines, whose data is reliable despite individual node failures?

- Basic idea:
  - split the data into small chunks
  - distribute the chunks to nodes
  - replicate each chunk at multiple nodes to deal with failure
  - a master server with the filename → chunk mapping

```
*  GFS (HDFS)

        < GFS diagram >

         client                          master


                          chunkserver  chunkserver chunkserver
```

  - provides a file system (global namespace) over cluster of machines
  - a file is split into (often) 64M chunks
    - each chunk is replicated on multiple chunkservers
  - GFS master + chuck server
  - master provides
    - file name → chuck server mapping
    - chunk server monitoring and replacement
  - chunk server provides actual data chunk
    - best chunkserver is chosen to minimize network bandwidth consumption
  - optimized for
    - sequential/random read of large file
    - append at the end
      - synchronizing random writes much more costly with failures
  - Q: Can a single master server enough to deal with the huge filesystem?
    - Ex) 1 billion files. 64 byte name per file. 4GB per file. How much to space to remember the mapping?

- SAN (Storage Area Network)
  - "scale up" approach for disks
  - provides SCSI interface over many hard disks (often thru optical LAN)
  - all nodes in the cluster see one shared big disk
  - convenient but very expensive

NoSQL datastores

- Limitation of relational databases
  - Relation: very general. But is it the best data representation?
    - e.g., java objects
  - SQL: very expressive data retrieval. But is this expressiveness necessary?
    - e.g., student records. How will it be accessed?
  - The power of SQL also leads to complexity. Do we really need it?
    - Not easy to scale out relational database
    - Traditional database focuses on functionality over scalability
    - Need for a new system that is designed from the ground up for
      - Ultimate scalability for highly distributed environment
      - Core functionality necessary for new Web apps
  - ACID: very strong data integrity guarantee. But is it necessary?
    - e.g., user's status updates. Is ACID necessary?

- Challenges in ACID and perfect synchronization
  - Byzantine generals problem
    - two generals A, B need to attack together to win
    - messenger may be lost during delivery (say with 90% chance)
    - Q: How can they make sure simultaneous attack on Sunday midnight?

    - Q: A sends a messenger to B.
      A attacks after sending a messenger.
      B attacks once he hears from a messenger.
      Any problem?

    - Q: A sends a messenger to B.
      B sends ACK messenger back to A.
      A attacks if he gets ACK from B.
      B attacks if he hears from A's messenger.
      Any better?

    - Q: A sends a messenger. B ACKS. A ACKS back to B.
      Any better?

    - Q: How to improve chance of success?

    - Remark:
      - Keeping states on multiple nodes synchronized is often VERY COSTLY.

        - synchronization overhead outweighs benefit for more than a few nodes

    - If possible, relax "state synchronization guarantee" on multiple nodes
        - otherwise, the overall performance will be too based compared to centralized processing

- CAP theorem (by Eric Brewer)
    * Consistency: after an update, all readers will see the latest update
    * Availability: continuous operation despite node failure
    * Partition Tolerance: continuous operation despite network partition

    * Only two of three characteristics can be guaranteed by any system

- Two important questions:
    - Q: How to relax consistency guarantee?
        * ACID vs. BASE
            - Basically Available: system works basically all the time
            - Soft-sate: does not have to be consistent all the time
            - Eventual consistency: will be in a consistent state eventually
        * possible consistency models
            - Read Your Own Writes (RYOW) Consistency
            - Session Consistency
            - Monotonic-Read Consistency: readers will see only newer update in the future

    - Q: What data model to use?
        * (key, value) store
        * column-oriented store
        * document datastore

- **(key, value) store**
    - e.g., Amazon DynamoDB, Project Vodemort
    - key: student id, value: (student name, address, email, major, ...)
        * Q: How to process "find student with sid 301"?

        * Q: How to process "change the major of sid 301 to CS"?

            - Read can be inefficient for a single-field retrieval
            - Update can be inefficient for single-field changes

        * Q: How to process "find student whose email is cho@cs"?

            - Data access using a non-primary key is not supported
            - Need to create and maintain a separate index

- **column-oriented store**
    - e.g., Google's BigTable, Cassandra (Facebook)
    - Data is a set of rows that are associated with a number of column families that has a number of columns.
    - Each row has a unique row-key, but may not have all column values
        * Access data by (row-key, column-name)
    - Column families are used to preserve locality of column storage
    - Compared to (key,value) store, more structure of the data is exposed to the system
        * more efficient update and retrieval
    - Q: What about retrieval based on column values (like student by email?)

        * Still data access is limited based on row-key

- **document datastore**
    - e.g., MongoDB, Amazon DynamoDB, CouchDB
    - Document consists of named fields, which are (key, value) pairs. There is no preset "schema" for documents
        * Title: CouchDB
          Categories: [Database, NoSQL, Document Database]
          Date: Jan 10, 2011
          Author: ...
    - User specifies the fields on which to build an index
        * Allows data retrieval based on "non-key" fields (by using appropriate indexes)
        * But, this datastore will be more complex than others
        * Likely to be less scalable than others

- **Consistent hashing**
    - Q: How to distribute objects with different keys to k nodes? What about "hash(object) mod k"?

    - Q: How to minimize data reorganization when a new node is introduced?
        * Consistent hashing:
            - Hash BOTH objects and *nodes* to a hash ring
            - Assign objects to the node right behind them
        * Q: What to do when a new node is introduced?

- Q: Non-uniform hash range for each node. How to minimize load imbalance?

    * Hash many virtual nodes to the ring
    * Assign virtual nodes to physical nodes

- Q: How to make data available despite node failure?

    * Replication to the next k nodes

Caching

- Q: Can we use caching to improve performance/scalability?

- Q: At what layer? Storage/DB? Application? HTTP?
  - Caching files/disk blocks (disk cache)
  - Caching database objects (object cache, such as memcached)
  - Caching dynamically-generated web pages
  - Caching content close to the users (content distribution network)

- Data object caching layer (memcached)

  `<caching layer diagram>`

  - all database access goes through caching layer
  - minimize # requests hitting DB
  - memcached data model: (key, value) pair. Key-based lookup
  - can use multiple machines for caching by partitioning the key w/ special care on possible machine failure

- Dynamic page caching layer
  - Store generated HTML page as a static file
    - E.g., WordPress cache plugins

  - Q: What if a page contains a few user-specific content?

- Content distribution network (CDN)
  - Cache pages/images/videos close to users at the edge of the network
  - Users access cached object located close to them
    - Lower delay. Lower load on the main server
  - Q: How can a browser "know" the location of the cached objects that are close to them?

Challenges in cluster-based architecture

- Many corporations manage a large number of server clusters
  - > 10,000s machines in one data center
    - commodity linux boxes
  - to handle both 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

- 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
    - e.g., MapReduce (or Hadoop) provides
      - 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

- **Important Notes for 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
    - verify data after every step
      - add CRC and checkpoints for data verification and recovery
      - replicate data as backup
  - 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?

A:

1. Buy the domain name cs144.com
    a. 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?

A:

1. Set up a physical machine
    a. Buy a machine (~$1,000/PC)
    b. Buy an internet connection from an ISP
        i. Verizon, AT&T, Comcast, … (~$100/month)
    c. Install OS and necessary software
    d. All maintenance hassles for physical machines and the software
2. "Rent" a machine from a cloud hosting companies
    a. Amazon Web Service, Google Cloud Platform, Windows Azure, …

Q: Exactly what do we rent from a cloud company?

A:

1. Infrastructure as a service (IaaS)
    a. Rent a "virtual machine" and run your own virtual machine image
        i. Amazon Elastic Cloud 2, …
    b. No hardware to manage, but all software needs to be managed

2. Platform as a service (Paas)
    a. Rent a common computing platform, including OS, database, application and web servers.
        i. Macrosoft Azure, Google App Engine, …
        ii. LAMP (Linux+Apache+MySQL+PHP), Java Servlet, Python+Django, …
    b. You maintain your own application

3. Software as a service (SaaS)
    a. Rent a fully working "software" over internet
        i. Google Apps for Work, Salesforce.com
    b. 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), …

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

```
        Query log
1,time,userid1,ip,referer,query  →   cat 100000
2,time,userid2,...                   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?

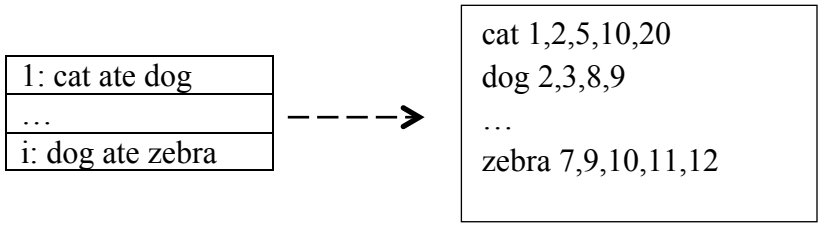
  - Q: How can we parallelize it on thousands of machines?
    * Q: Can we process each query log entry independently?

    * 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/transformation step: (query id, query_log) → (query, 1)
    2. reduce/aggregate step: group by query and sum up each query count

- Example 2: 1 billion pages. build inverted index



    - 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/transformation step: (docid, content) → (word, docid)
        2. reduce/aggregate step: group by word and output list of docids

- General observation
    - many data processing jobs can be done as a sequence of
        1. map: (k, v) → (k', v')
        2. reduce: partition/group by k' and "aggregate" v's for the same k'
    - the output (k',v') only depends on (k,v), not other input pair
        * thus each mapping task can be executed in parallel
        * "aggregation" on different keys are independent
            - thus each reduction task can be executed in parallel

- if the data processing follows this pattern, this can be parallelized as
  1. split the input into independent chunks
  2. run one "map" task per each chunk over multiple machines
  3. partition the output of the map task by the output key
  4. transfer and sort each partition by key (to put keys together) and run one reduce task per each partition

     * the exact the map and reduce functions determines what we do

     * Q: Inside "reduce" task, will v's be in the same order?

     Note: reduce function should be agnostic of the order of v's
- Q: What might be the issues when we run map/reduce tasks on thousands of machines?
  * failed node
  * slow node

- MapReduce
  - programmer provides
    * mapping function (k, v) → (k', v')
    * reduction function (k, [v1, v2, ...]) → (k, aggr([v1, v2, ...]))
    * and input/output handling functions

  - Given the two functions, MapReduce handles overall data plumbing and task scheduling
    * split/parse input file into small chunks and assign and run map task for each chunk
      - close to the chunk data location
    * partition map output, transfer partitions to reduce nodes sort each partition
      - "buffer" map output to a temporary (local) file
    * run reduce task once partition data is ready
- Hadoop
  - open source implementation of MapReduce (and GFS)
  - map and reduce functions are implemented by:
    * Mapper.map(key, value, output, reporter)
    * Reducer.reduce(key, value, output, reporter)

  - word count Hadoop example

```
package edu.cs144;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

 ////////////  MAPPER function //////////////
  public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      Text word = new Text();
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, new IntWritable(1));
      }
    }
  }

 ////////////  REDUCER function //////////////
  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    ////////// JOB description //////////
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setMapperClass(Map.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    JobClient.runJob(conf);
  }
}
```

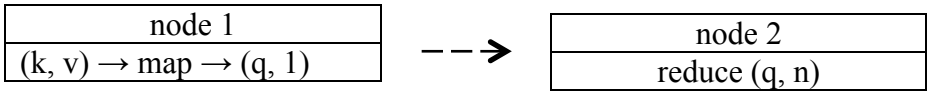
* the association of input/output files and map/reduce functions, input/output format is done through JobConf class:

```
JobConf.setMapperClass
JobConf.setReducerClass

FileInputFormat.setInputPaths(JobConf, Path)
FileOutputFormat.setOutputPath(JobConf, Path)
JobConf.setInputFormat
JobConf.setOutputFormat

JobConf.setOutputKeyClass
JobConf.setOutputValueClass
```

- Q: Any further optimization?
  * Query frequency count example



Do we need to send (q, 1) pair for every query?

  * Combiner function
    - significant saving in network bandwidth in many cases

- Note: not every computation fits the MapReduce mode