OS and Hardware for Distributed Computing

(Faculty Development Programme workshop on High Performance Computing), 26th June to 1st July 2017, IIT Bombay.

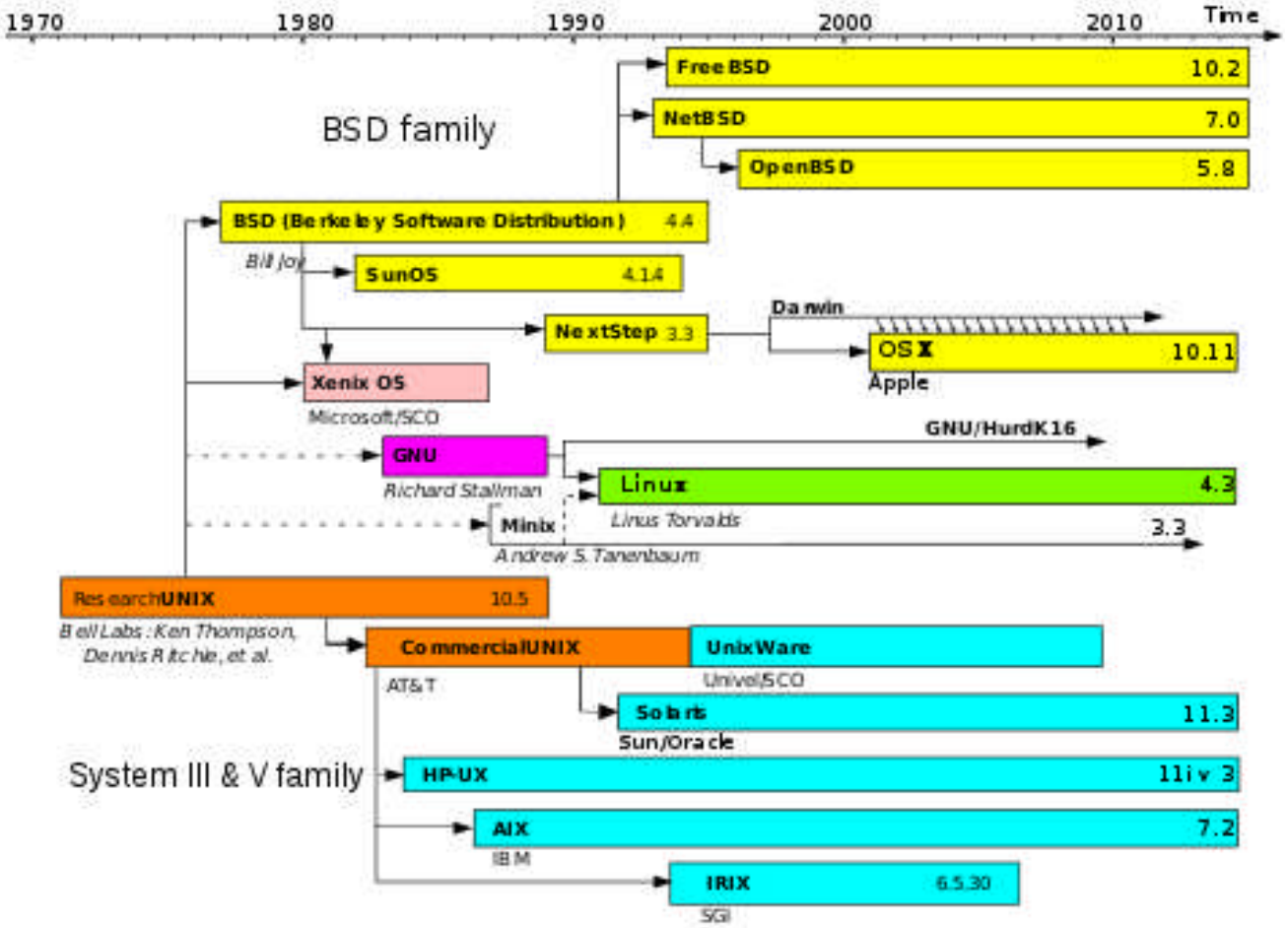
by Dr. Mohsin Ahmed

moshahmed@gmail mosh@hmi-tech.net

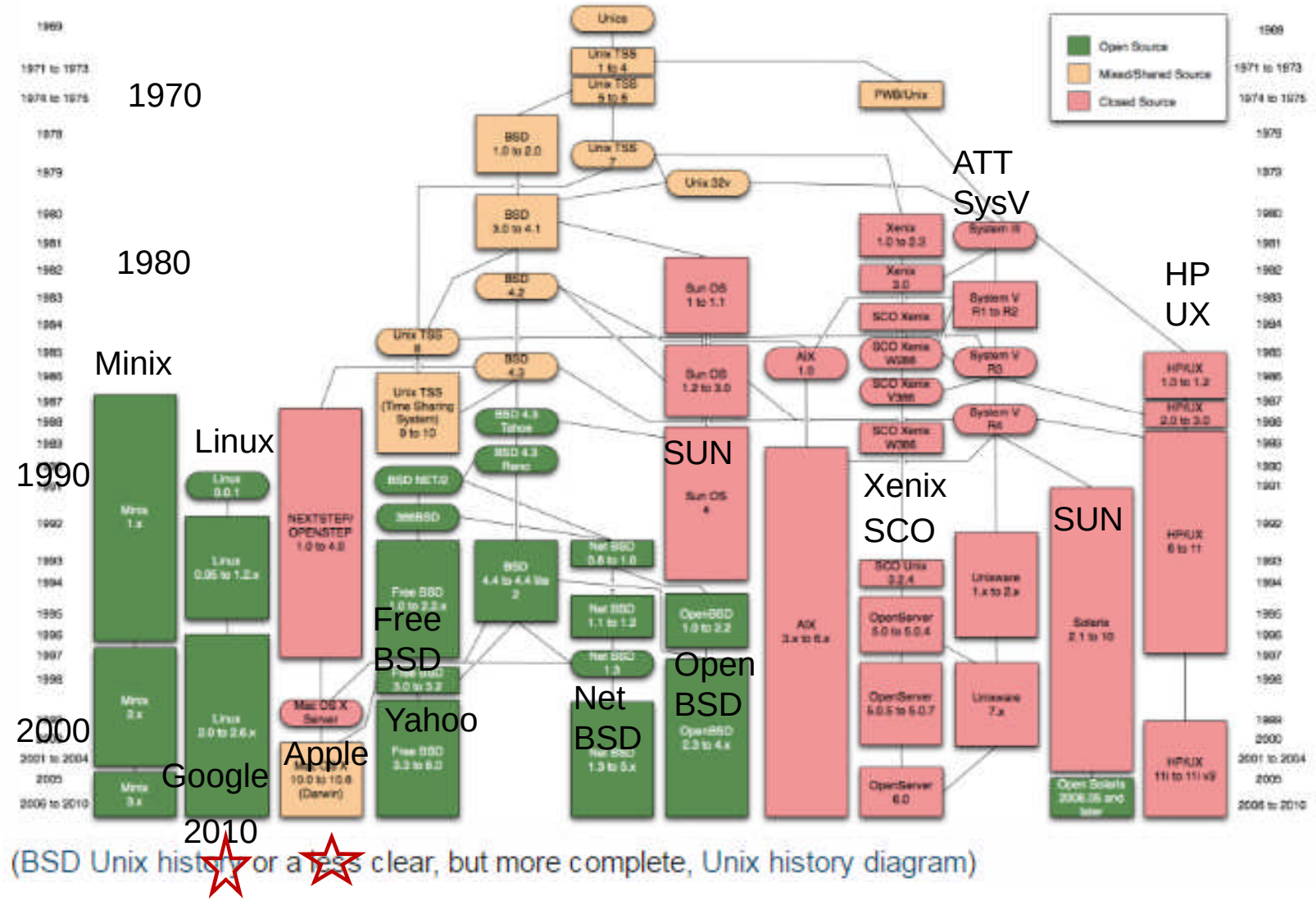
FDP on HPC at EE.IIT Bombay 1/7/2017

Unix: BSD, Solaris, HPUX, Linux

History: BSD, Solaris, HPUX, Linux



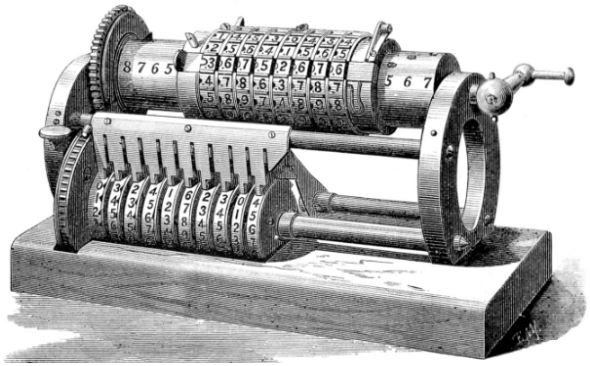
BSD



(BSD Unix history or a less clear, but more complete, Unix history diagram)

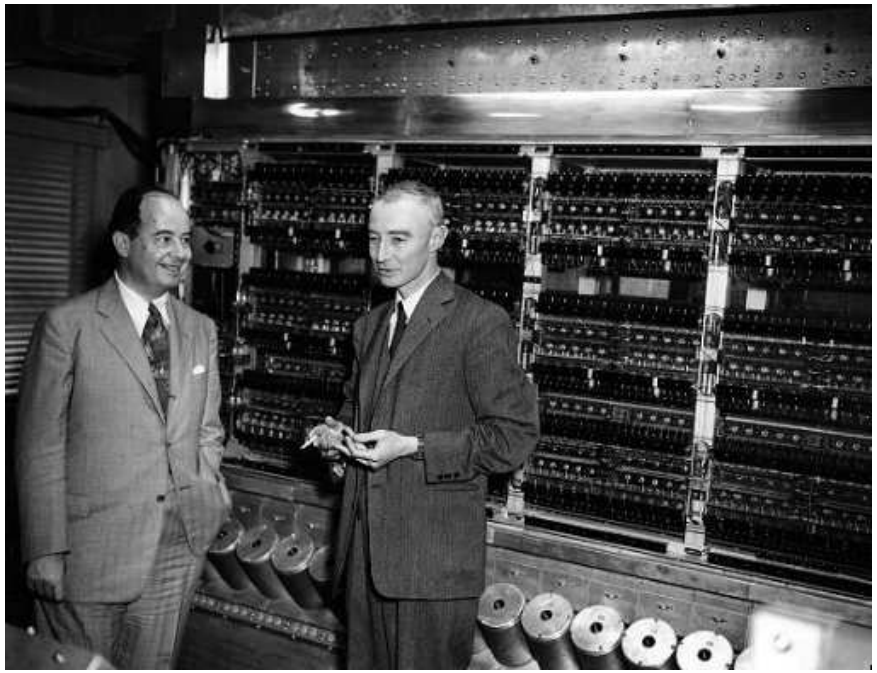
1800s

1832, Babbage's Difference Engine



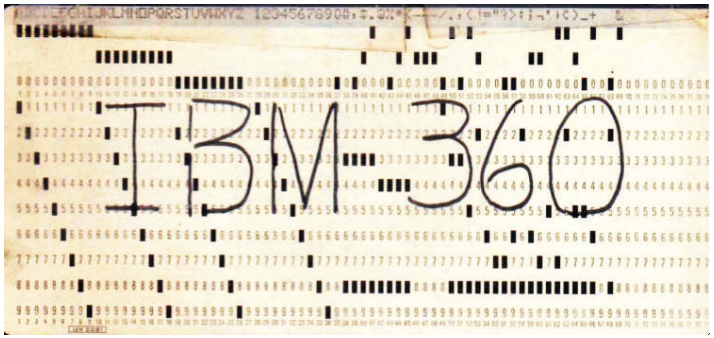
Mech Calculator 1877

1940



John von Neumann, Robert Oppenheimer
with Vacuum tube computer.

1960s - IBM 360



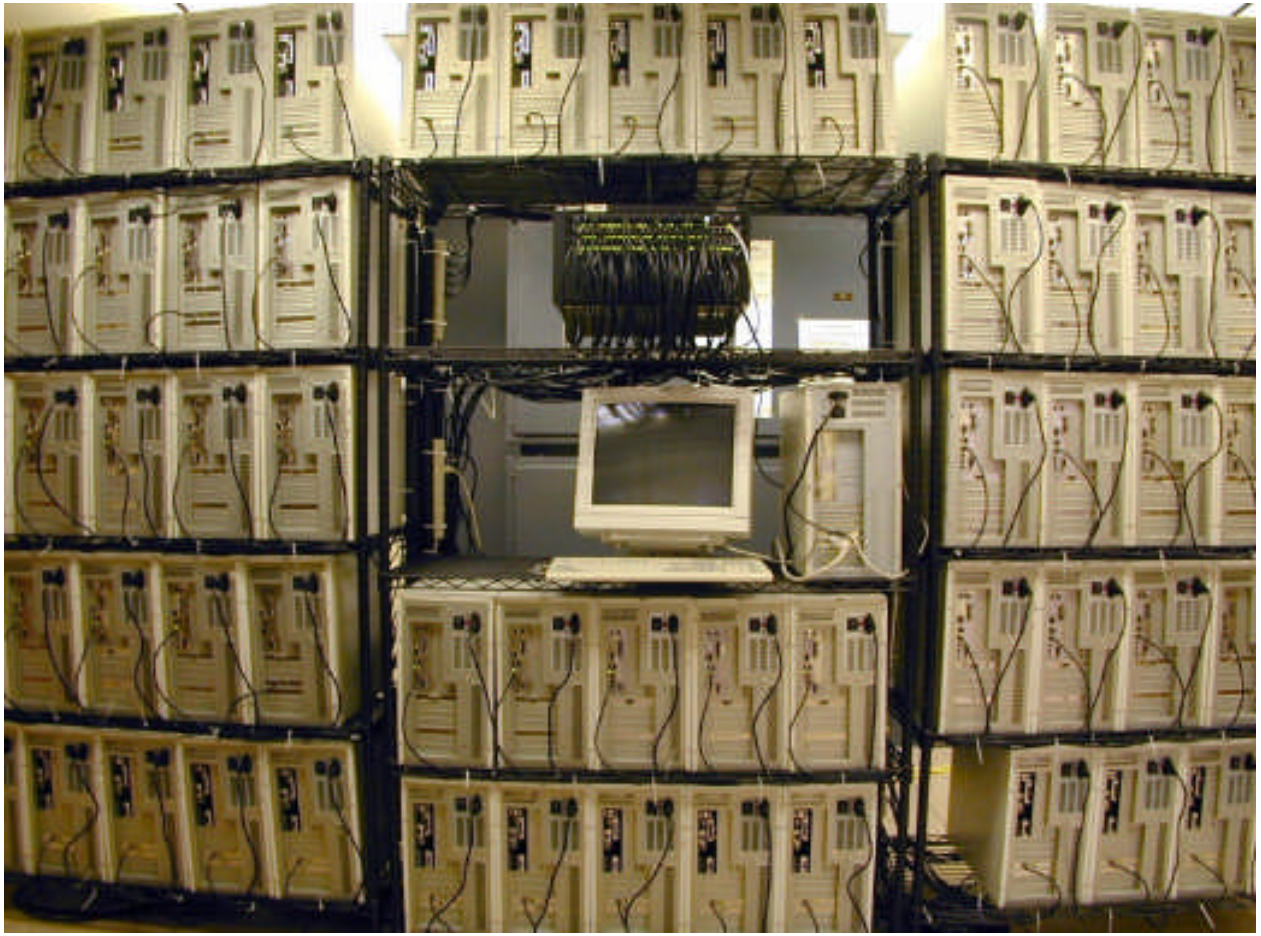
Punch card and tape operators

PDP 11, 1970s

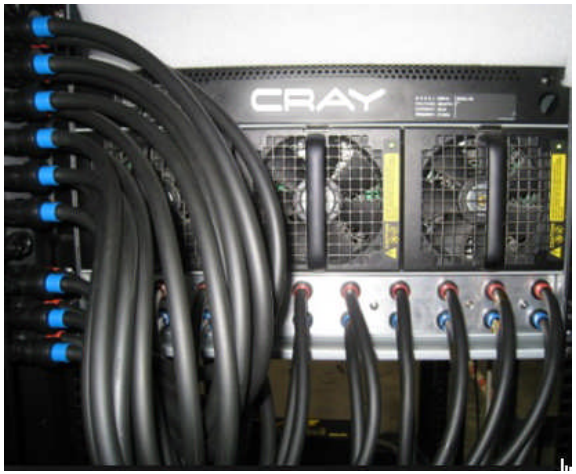
Teleprinter on pdp11

ATT Xenix

1990s Cluster

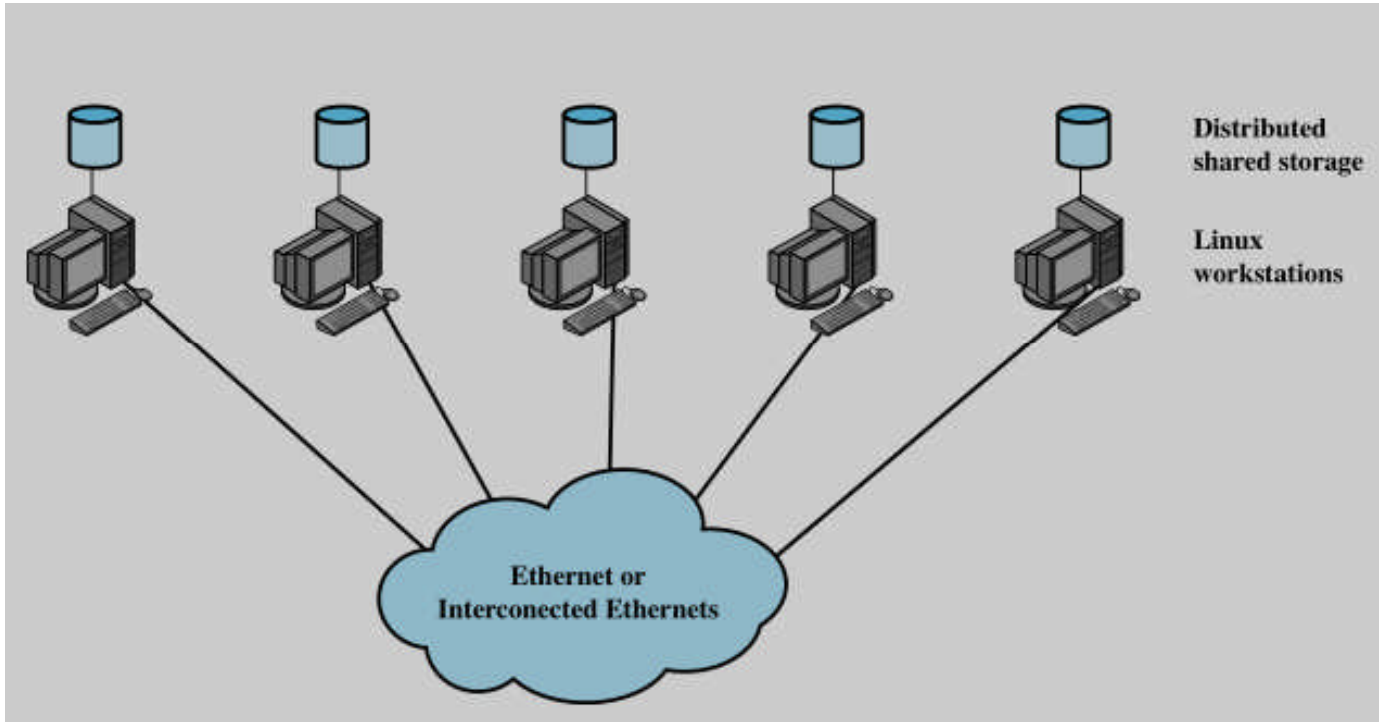


Cray supercomputer

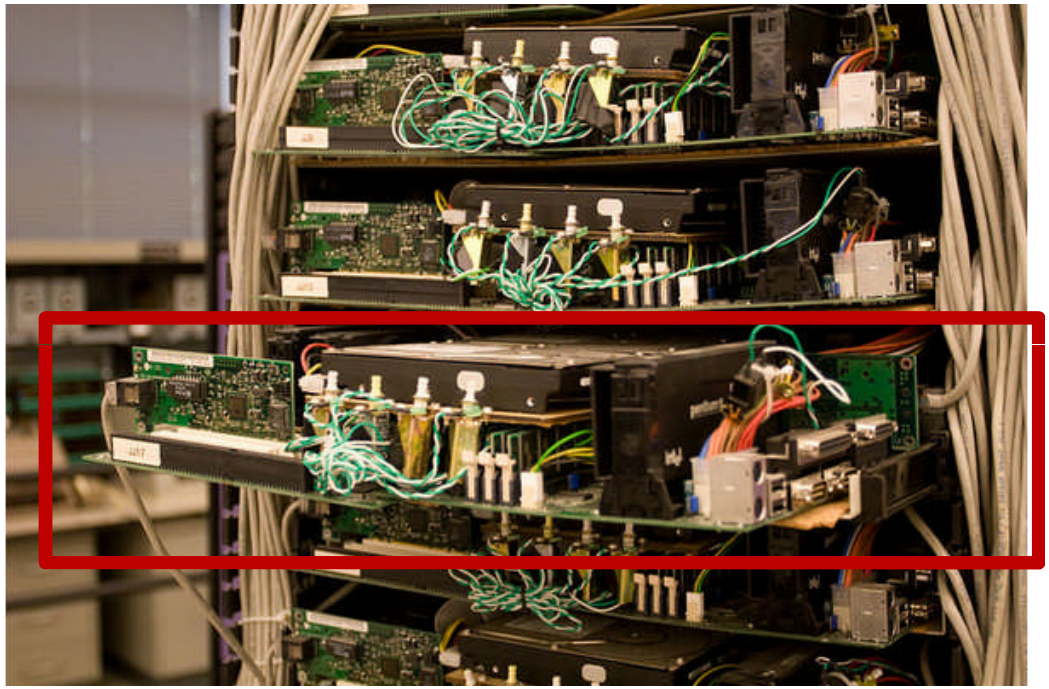


Cooling Cray CPU

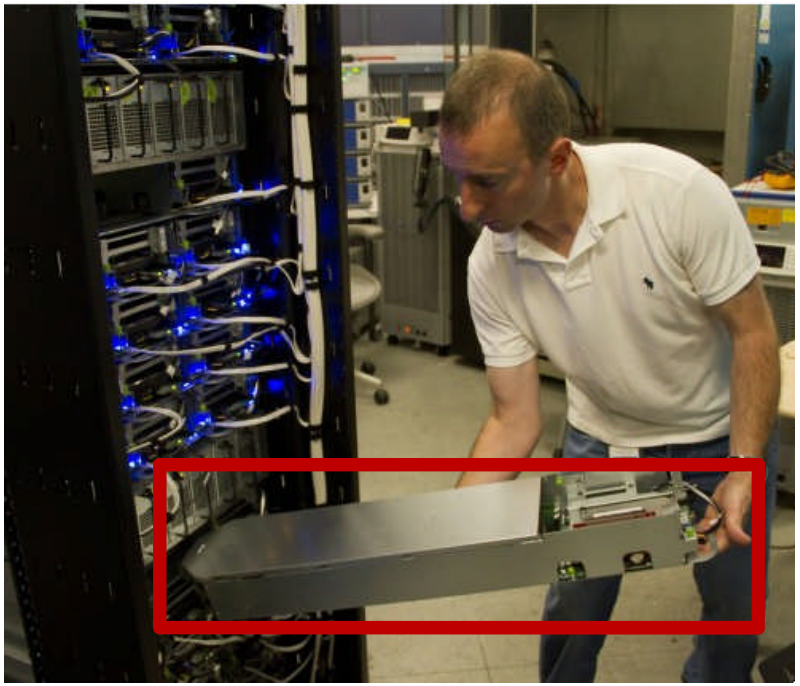
Cloud Cluster



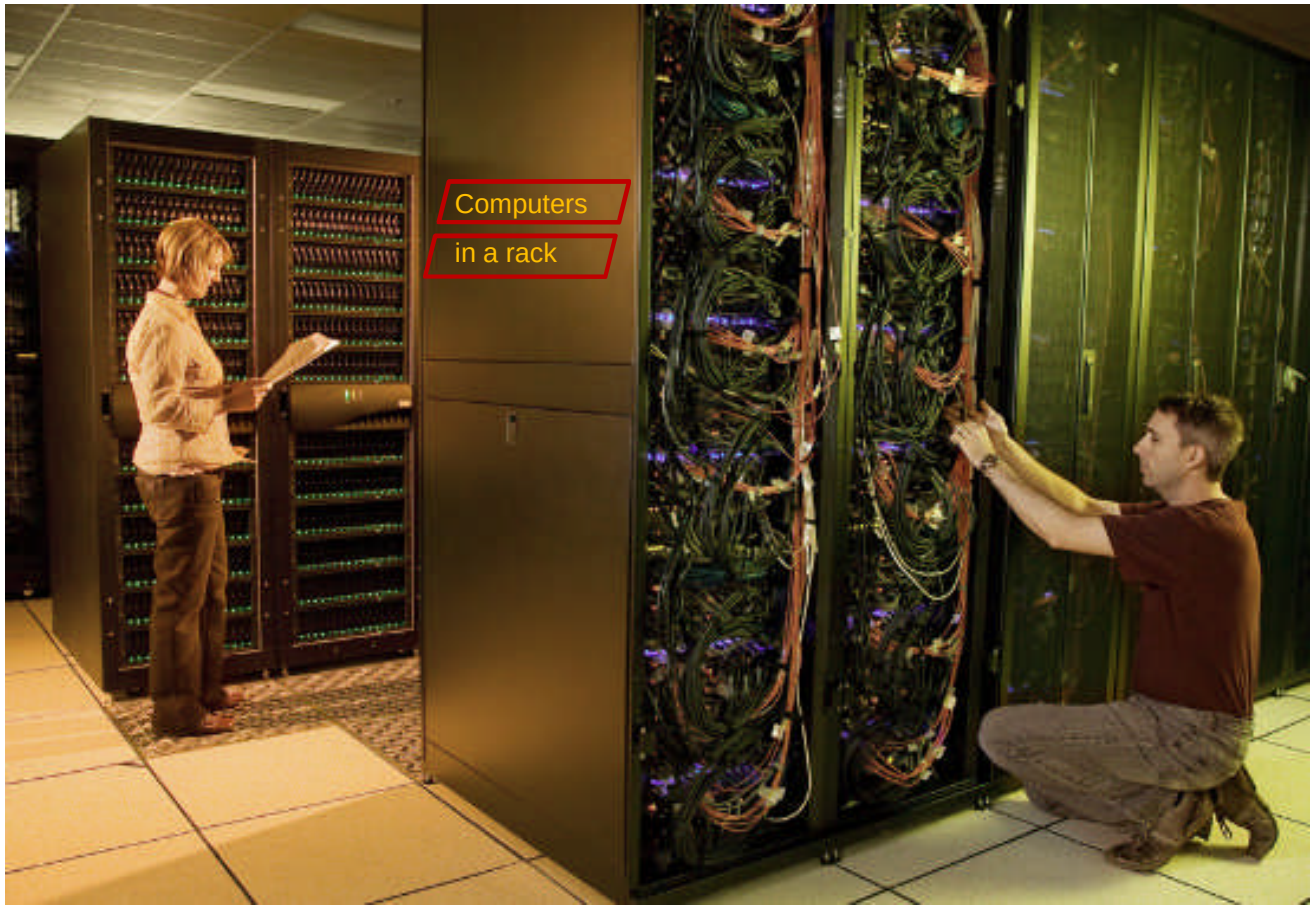
Google Rack Computer



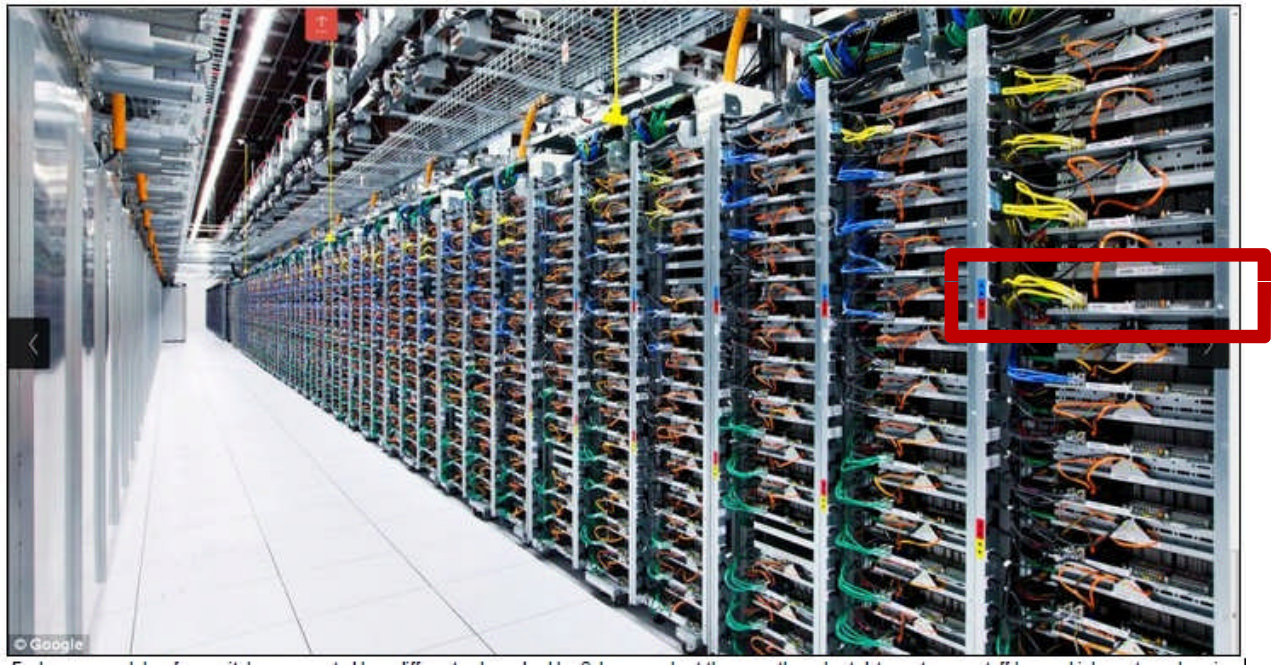
Google Rack Computer



2015 Data center



Google Data Center



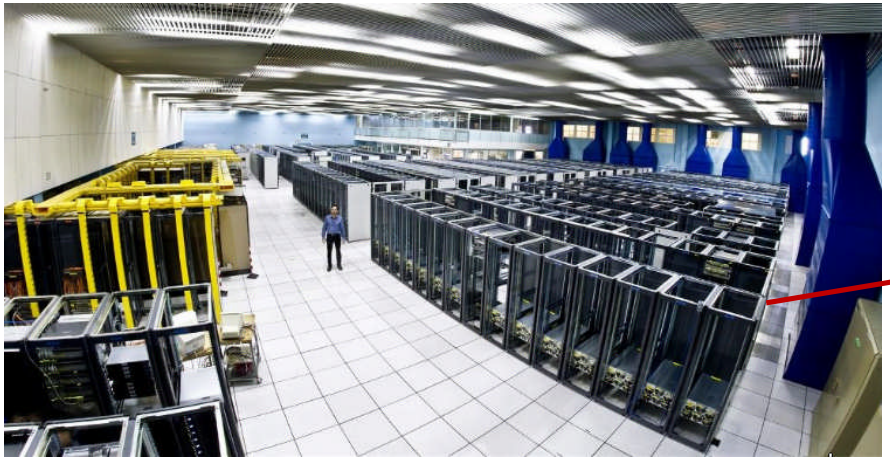
More on
http://www.dailymail.co.uk/sciencetech/article-2219188/Inside-Google-pictures-gives-look-8-vast-data-centres.html

Google Datacenter

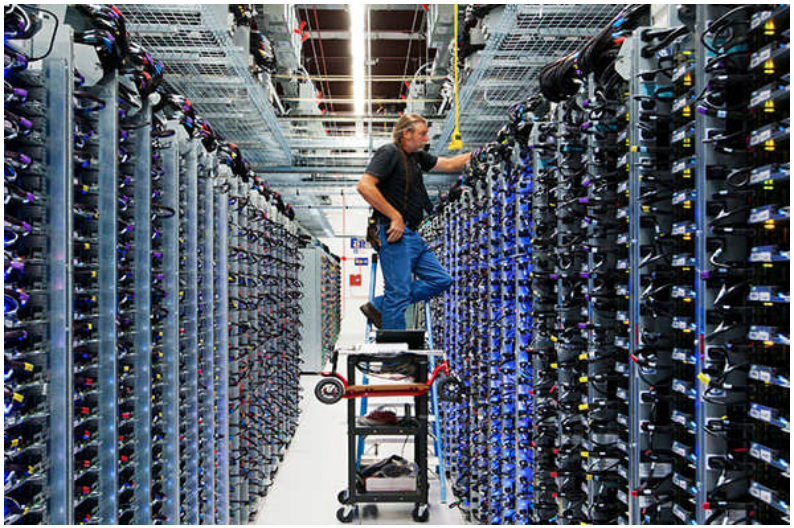
Hot air exhaust

Cooling systems

Cloud



Cages with locks to secure computers



Google Data Centers 2006

References

1. Unix/Linux/Windows OS, Stallings, 2008
2. OS Concepts, Silberschatz, 2005.
3. Advanced Windows, Richter
4. Programming Windows, Petzold
5. Advanced Prog on Unix, by Stevens
6. TCP/IP Illustrated 1,2,3 by Stevens
7. Memory http://www.tenouk.com/ModuleW.html
8. MIT OS https://pdos.csail.mit.edu/6.828/2009/overview.html
9. Wikipedia?Physical_Address_Extension

Distributed Computing Architecture

Dr.Mohsin Ahmed

FDP on HPC at EE.IITB, 1/7/2017

Scale

- 64 bit machines
  - Lots of RAM – 10s of GB
  - Lots of HD – few TB
  - No video, audio, monitors, serial ports
  - Gps NIC, USB for debugging
- Linux or BSD operating system
- Racks of 100 computers
- 100,000 machines in a data center

Hardware Issues

- Power / rack = 300W * 100 = 30KW
- Power / dc = 300W * 100,000 = 30MW
- Heat dissipation
- Power supply
- High speed network fibre
  - 100,000 * 1Mbs/machine = 100 Gps
- Physical security for data.

Hardware failures

- Reliability – 0.01% failure = 10/dc/day.
- Detect RAM,DISK,POWER,Mother board errors/failures.
- Replace components without down time.

Dealing with Failures

- So if machine m1234 fails, another free machine will be named m1234, and all programs moved from old to new machine. Users will not even realize the machine has changed!

Fault tolerant distributed DNS

- DNS maps names to machines (ip address) in DC. Machines may have names like “m1291919”
- Transparently removes bad machines from use.
- Install software on new machines
- Replace failed machines with new ones
- Inform operator of failures

Security

- Ssh with public and private keys on all machines
- Two factor authentication for users
- Proxy servers at firewalls
- Log all user access.
- Log all jobs, for security audits.
- Accounting – CPU is expensive, 10$/hour, with 20K machines would be charged 20,000$/hour charge.

Monitoring

- All machines must report back in realtime to master monitors, which compute statistics like CPU usage, users, bandwidth, power, temperature, failure rates.
- Error rate > threshold => send email, page operator, call operator, dynamically throttle traffic, instead of falling over under load. (Domino effect).

Leader_election

- **Leader election** is the process of designating a single process as the organizer of some task **distributed** among several identical computers (nodes).
- Before the task is begun, all network nodes are unaware which node will serve as the "**leader**," or coordinator, of the task.
- After a **leader election** algorithm has been run, each node in the network knows an unique node as the task **leader.**

The Chubby Lock Service for Loosely-Coupled Distributed Systems

1. **Chubby** lock service, which is intended to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system.
2. **Chubby** provides an interface much like a distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance.
3. Many instances of the service have been used for years, with several of them each handling a few tens of thousands of clients concurrently.

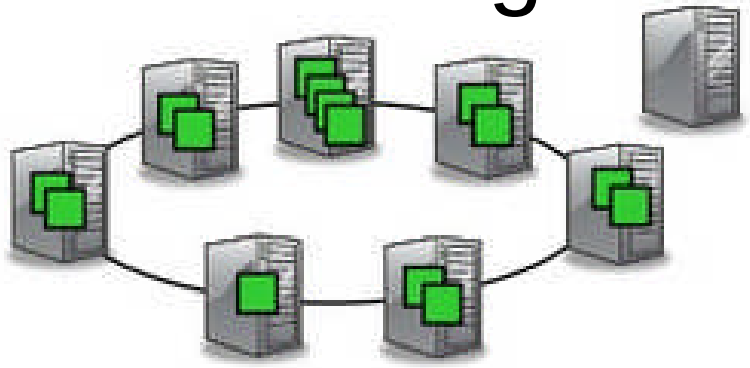
Details

- GFS
- Bigtable
- Mapreduce
- Protocol buffers

GFS

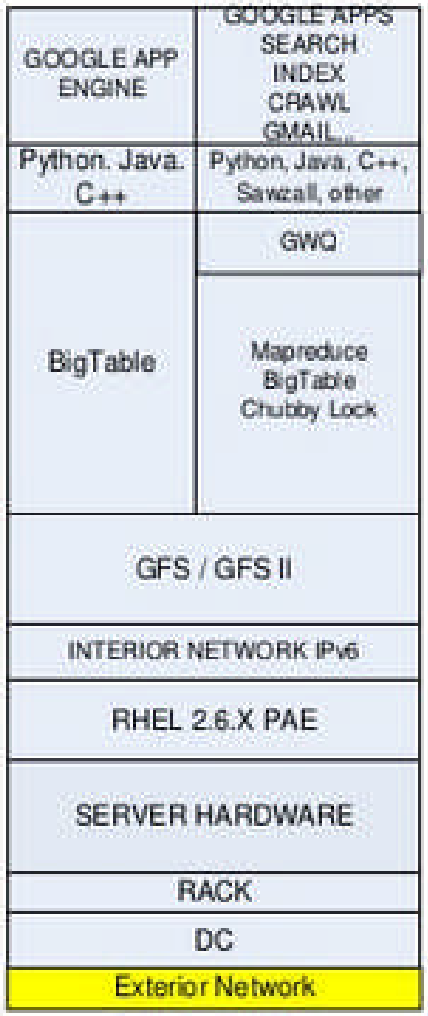
Distributed file system

GFS: Google File System Arch

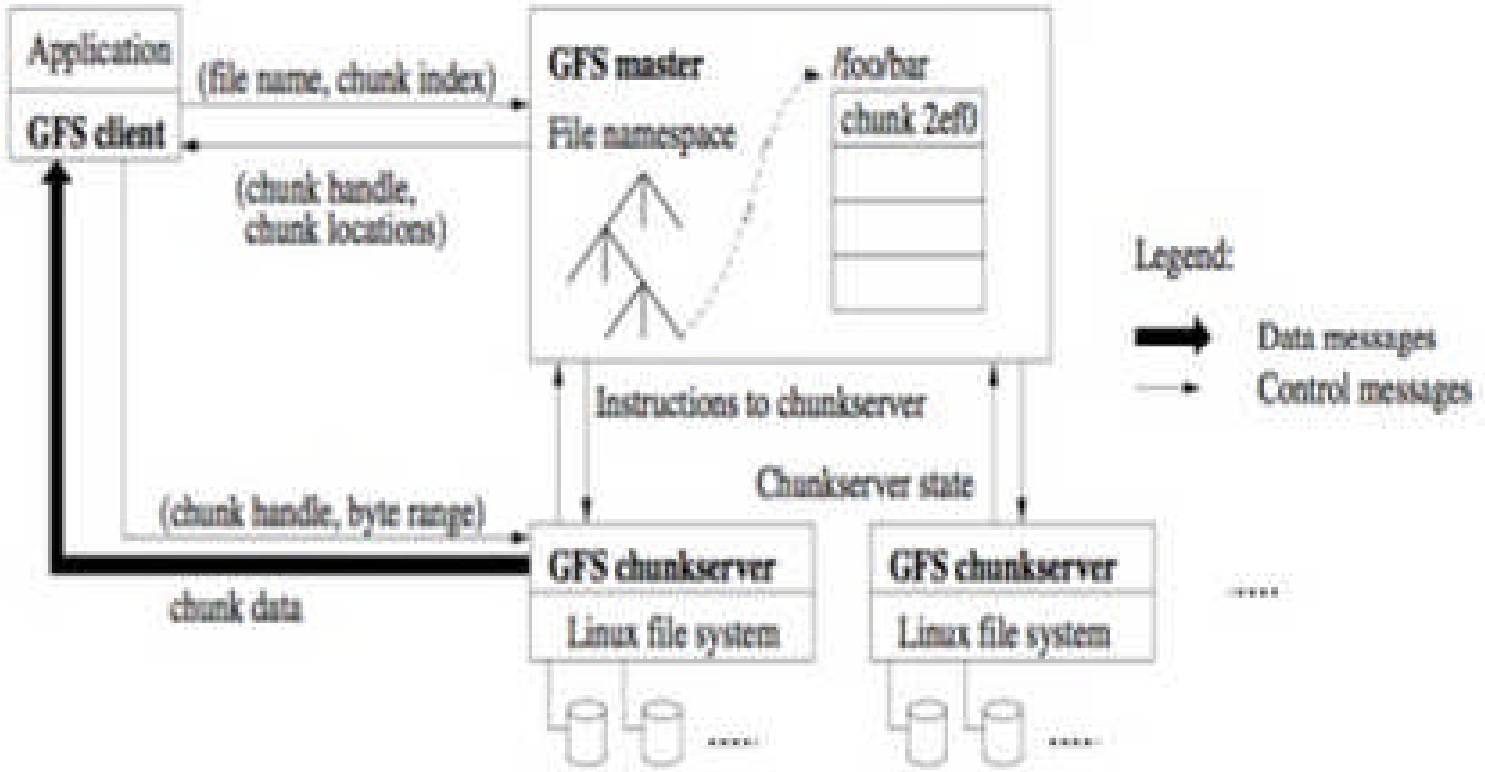


Google File System (GFS)

- Single virtual file system spread over many machines
- Optimized for sequential read and local accesses
- PRO: High throughput, high capacity
- "CON": Specialized for particular types of applications



GFS



http://www.slideshare.net/AditiTechnologies/google-architecture-breaking-it-open

GFS

- **GFS** is Google's distributed file system for efficient, reliable access to data using large clusters of cheap hardware.
- **GFS** is not in the linux kernel of an operating system, but is instead provided as a userspace library.

GFS Masters and Chunkservers

- One ***Master*** node and a large number of
- **Chunkservers** store the data files, with each individual file broken up into fixed size chunks (hence the name) of about **64 Mb**, similar to clusters or sectors in regular file systems.
- Each chunk has a unique **64-bit id**, and logical mappings of files to constituent chunks are maintained.
- Each chunk is **replicated** (>=3) throughout the network, more copies for files that have high demand or need more redundancy.

GFS Master's work

- The Master server doesn't usually store the actual chunks, but rather all the metadata associated with the chunks, such as the tables mapping the 64-bit labels to chunk locations and the files they make up, the locations of the copies of the chunks, what processes are reading or writing to a particular chunk, or taking a "snapshot" of the chunk pursuant to replicating it (usually at the instigation of the Master server, when, due to node failures, the number of copies of a chunk has fallen beneath the set number).
- All this metadata is kept current by the Master server periodically receiving updates from each chunk server ("Heart-beat messages").

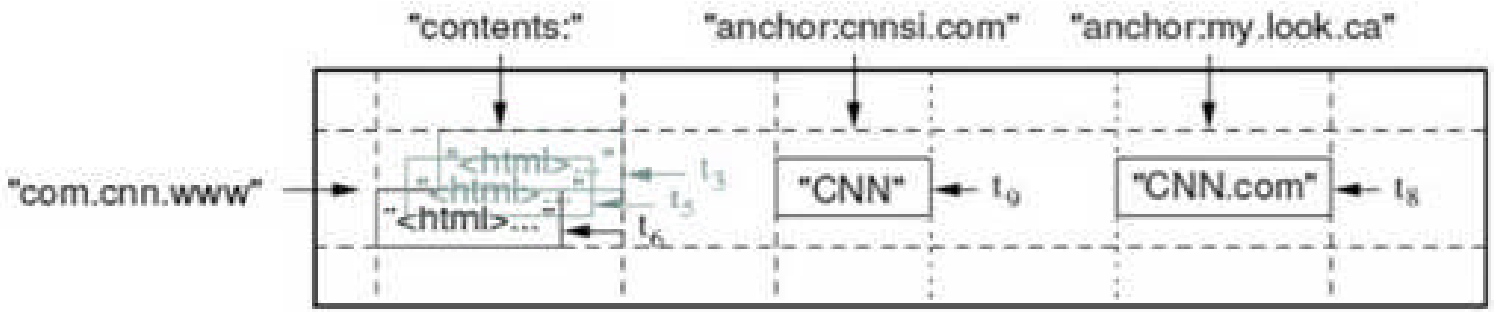
Chunk servers

- Keep adding chunk servers and more masters to increase capacity.
- See http://labs.google.com/papers/gfs.html

Big table

- **BigTable** is a compressed, high perf, and proprietary db built on GFS, Chubby Lock Service, SSTable.
- The excel sheet of the web.
- Billions of rows and columns stored on thousands of machines.

BigTable is a giant excel sheet



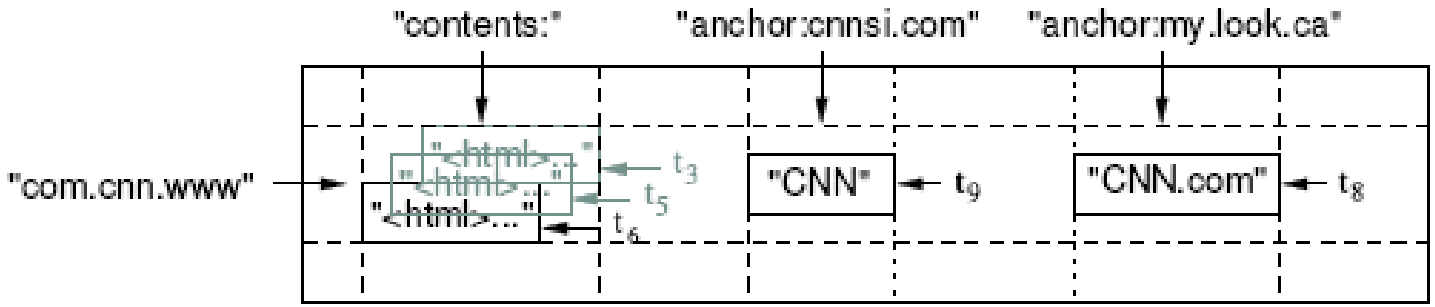
- Physically sorted on row-key – like a row-store
- Column families - like column-stores
- Variable (record-by-record) columns within a column family
- Column-values versioned; stored in reverse chronological order
- Designed to store hyperlink structure of web

BT keys and values

- **BigTable** maps two arbitrary string values (row key and column key) and timestamp (hence three dimensional mapping) into associated arbitrary byte array.
- It is not a relational database and can be better defined as a sparse, distributed multi-dimensional sorted map.
- **BigTable** is designed to scale into the petabyte range across "1000s of machines, and to make it easy to add more machines [to] the system and automatically start taking advantage of those resources without any reconfiguration"

BT Cell example

1 row, 3 columns



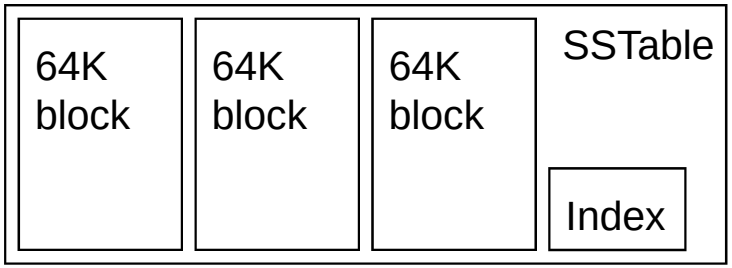
from http://labs.google.com/papers/bigtable.html

BT

- <Row, Column, Timestamp> triple for key - lookup, insert, and delete API
- Arbitrary “columns” on a row-by-row basis
  - Column family:qualifier. Family is heavyweight, qualifier lightweight
  - Column-oriented physical store- rows are sparse!
- Does not support a relational model
  - No table-wide integrity constraints
  - No multirow transactions

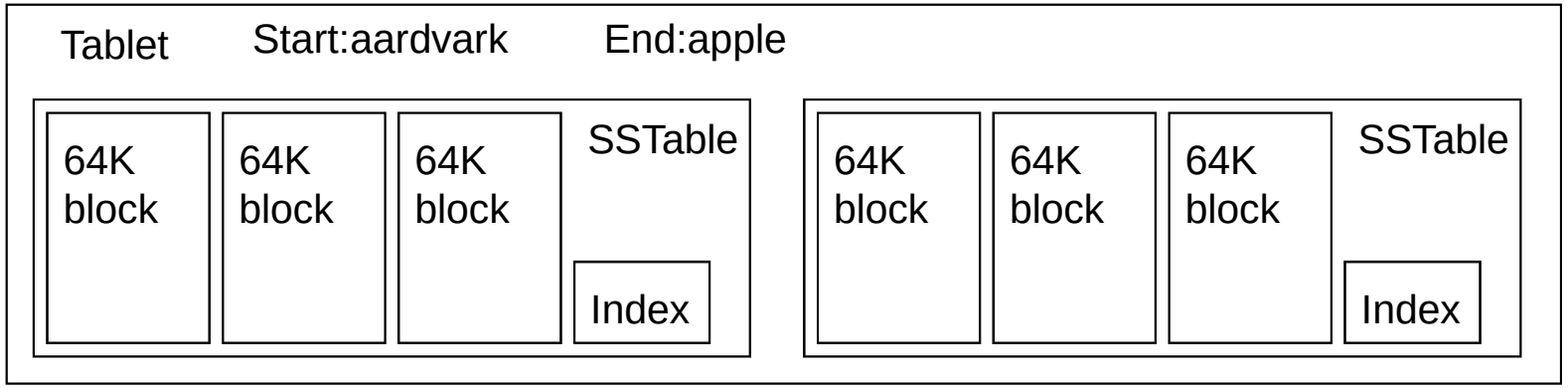
SSTable

- Hash map stored on disk
- Immutable, sorted file of key-value pairs
- Chunks of data plus an index
  - Index is of block ranges, not values



Tablet

- Contains some range of rows of the table
- Built out of multiple SSTables



Tablet Uses

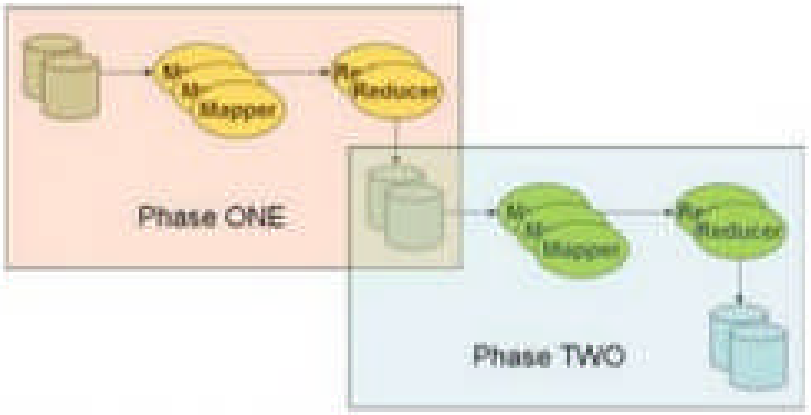
- Store the whole internet in a BT
- Store huge hash map in a BT
- Input to mapreduce
- Output from mapreduce
- Access is very fast
- Cache server

Map Reduce for parallel computations

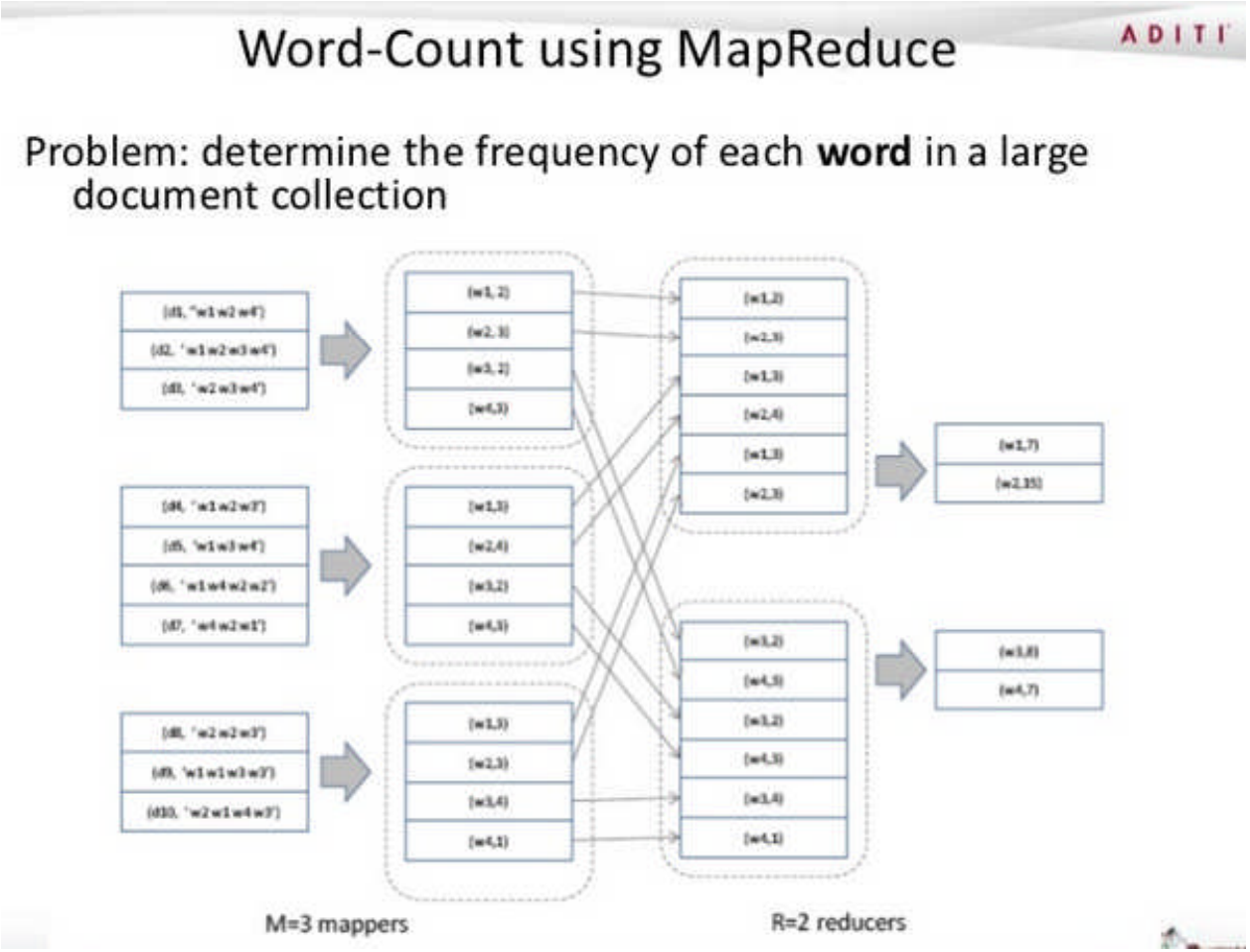
Map Reduction can be seen as a way to exploit massive parallelism by breaking a task down into constituent parts and executing on multiple processors

The Major Functions are MAP & REDUCE (with a number of intermediatary steps

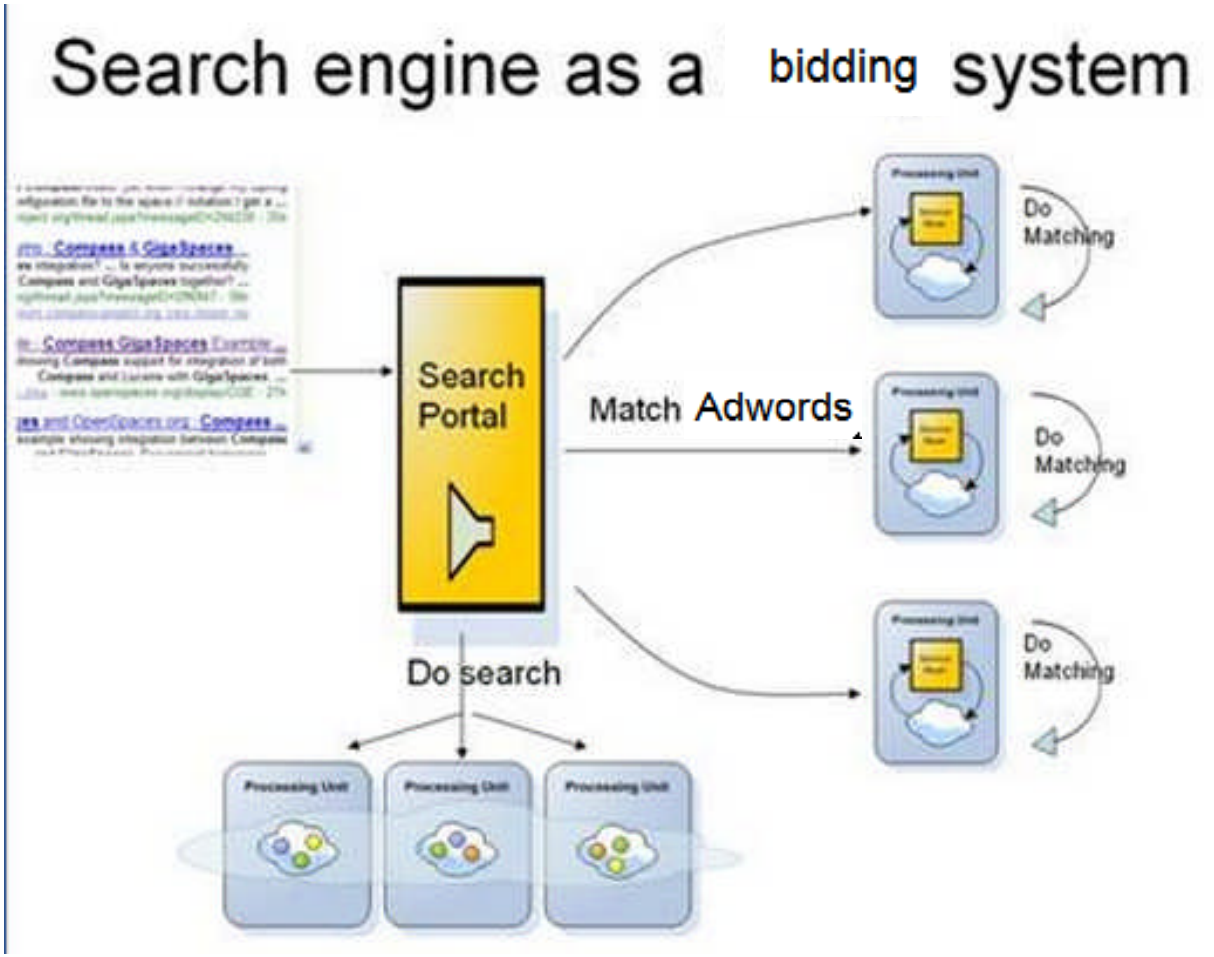
MAP **Break task down into parallel steps**
REDUCE **Combine results into final output**



Word count as MR program



Search Architecture



Google Cluster Architecture

by Barrosso, Jeff Dean, Holzle

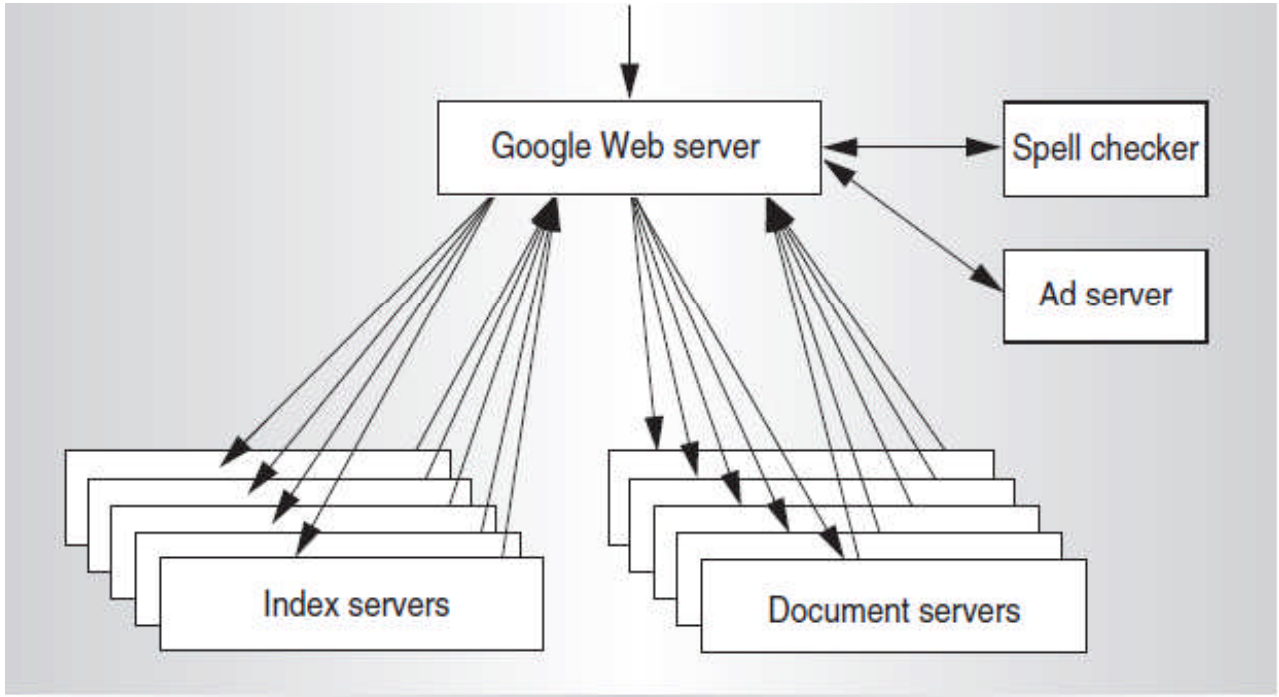


Figure 1. Google query-serving architecture.

from http://research.google.com/archive/googlecluster-ieee.pdf

Protocol buffers and gRPC

Dr.Mohsin Ahmed

FDP on HPC at EE.IITB, 1/7/2017

Protocol buffers

What are pb?

- Packed records in binary file
- Serialization Library to read/write pb in many languages: C, C++, Java, Python, Javascript, go, php, ruby, C#, etc.
- Passing data structs across langauges and over network – RPC and GRPC.
- Free with source code.

Usage

- Windows, download pb and compile with cygwin.
- Linux, download and make install.

- Create a pb definition file.
- Use protoc to generate headers and libraries for C++/Python/Java/..
- Use it in your program read/write pb data.
- see https://developers.google.com/protocol-buffers/docs/cpptutorial

Simple Example

Demo message definition in a file

```
$ cat demo.proto
syntax = "proto3";
package demo;
message demo {
   repeated int32 n=1;
}
```

Simple Example

Create a protobuf binary data using protoc (proto-compiler)

```
$ echo n : [1,2,3] |
      protoc --encode=demo.demo demo.proto \
      > demo.bin
```

Dumping proto data as text

```
$ protoc --decode=demo.demo demo.proto <
  demo.bin
       n: 1 n: 2 n: 3
```

And dump even if you don't have the proto definiton

```
$ protoc --decode_raw < demo.bin 1:
  "\001\002\003"
```

Field types

required: a well-formed message must have exactly one of this field.

optional: a well-formed message can have zero or one of this field (but not more than one).

repeated: this field can be repeated any number of times (including zero) in a well-formed message. The order of the repeated values will be preserved.

Details in https://developers.google.com/protocol-buffers/docs/proto

Example: AddressBook.proto

download from https://github.com/moshahmed/cc/tree/master/protobuf

```
syntax = "proto2";
package tutorial;
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType { MOBILE = 0; HOME = 1;   WORK = 2; }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}
message AddressBook {
  repeated Person person = 1;
}
// from https://developers.google.com/protocol-buffers/docs/cpptutorial
```

Compiling AddressBook.proto

1. For **C++**, the compiler generates a .h and .cc file from each .proto, with a class for each message type described in your file.
2. For **Java**, the compiler generates a .java file with a class for each message type, as well as a special Builder classes for creating message class instances.
3. **Python** is a little different – the Python compiler generates a module with a static descriptor of each message type in your .proto, which is then used with a *metaclass* to create the necessary Python data access class at runtime.

```
$ protoc --cpp_out=out/ addressbook.proto
$ protoc --python_out=out/ addressbook.proto
$ protoc --js_out=out/ addressbook.proto
```

AddressBook.py

```
address_book = addressbook_pb2.AddressBook()
try: # Read the existing address book.
  f = open(sys.argv[1], "rb")
  address_book.ParseFromString(f.read())
  f.close()

PromptForAddress(address_book.person.add()) # Add an address.

# Write the new address book back to disk.
f = open(sys.argv[1], "wb")
f.write(address_book.SerializeToString())
f.close()
```

What Are Protocol Buffers?

From http://code.google.com/apis/protocolbuffers/

- Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – **think XML, but smaller, faster, and simpler**.
- You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages – Java, C++, or Python.

Protocol buffers

- IPC – Interprocess communication
- RPC – remote procedure calls
- Data passing across network in protocol-buffers (open source library for C++, Python, Java from google).
- PB are compressed structs/objects
- PB 100x more efficient than XML, JSON.

Protocol buffers

- PB data can be exchanged between servers in different languages.
- Versioning of data, so old clients can keep operating with newer servers.
- Secure RPC - like SSL.
- Servers and clients have certificates
- Data is encrypted with public key.

Why Proto Buffers?

http://code.google.com/apis/protocolbuffers/docs/overview.html

- New fields could be easily introduced, and intermediate servers that didn't need to inspect the data could simply parse it and pass through the data without needing to know about all the fields.
- Formats were more self-describing, and could be dealt with from a variety of languages (C++, Java, etc.)
- Automatically-generated serialization and deserialization code avoided the need for hand parsing.

PB in RPC

- In addition to being used for short-lived RPC (Remote Procedure Call) requests, people started to use protocol buffers as a handy self-describing format for storing data persistently (for example, in Bigtable).
- Server RPC interfaces started to be declared as part of protocol files, with the protocol compiler generating stub classes that users could override with actual implementations of the server's interface.

polyline.pb example

```
message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}
message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}
message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
// From wikipedia
```

Using polyline.pb.cc

- This subsequently compiled with protoc.
- A C++ program can then use it like so:

- #include "polyline.pb.h"
- Line* createNewLine(const std::string& name) {
- Line* line = new Line;
- line->mutable_start()->set_x(10);
- line->mutable_start()->set_y(20);
- line->mutable_end()->set_x(30);
- line->mutable_end()->set_y(40);
- line->set_label(name);
- return line;
- }

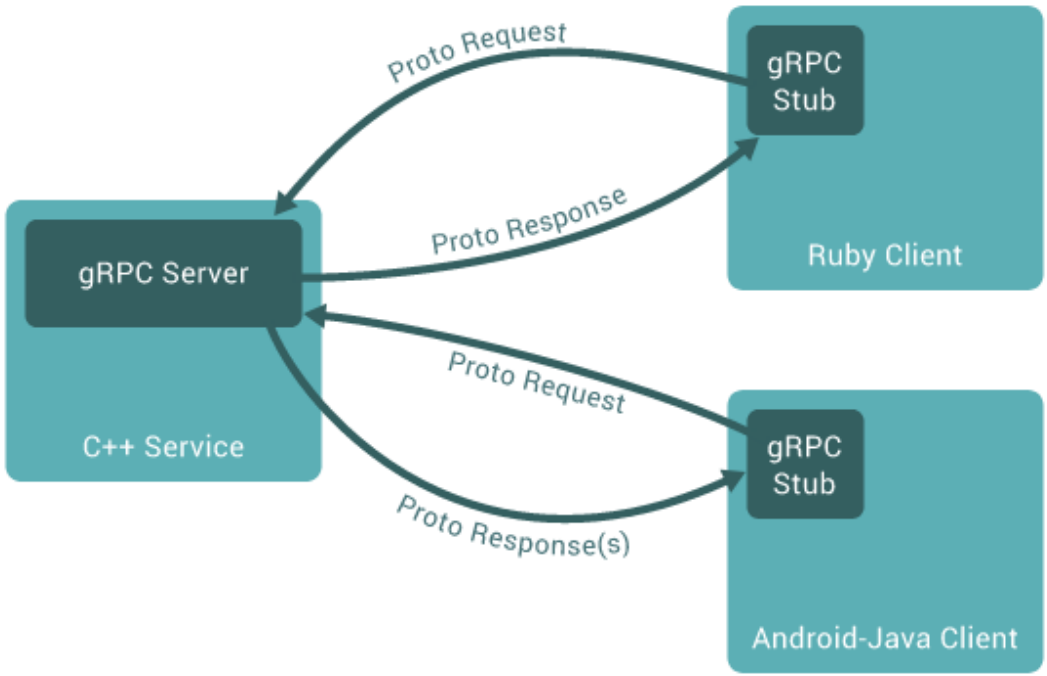
PB References

1. https://github.com/moshahmed/cc/tree/master/protobuf
2. https://en.wikipedia.org/wiki/Protocol_Buffers
3. https://github.com/google/protobuf
4. https://developers.google.com/protocol-buffers/docs/overview
5. https://developers.google.com/protocol-buffers/
6. https://github.com/nanopb/nanopb for c microcontrollers.
7. https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats

gRPC

- Google RPC.
- Distributed function calls.
- Secure/Authenticated
- Language independent using pb.
- Free software.
- Call functions from one program into another program on another machine.
- Windows: javascript version in nodejs
- Linux: C++, Java, Python, etc.

gRPC



http://www.grpc.io/docs/guides/

gRPC: Nodejs Client/Server example

```
$ npm install google-protobuf  minimist lodash grpc

$ c:/tools/nodejs/node.exe
   ./dynamic_codegen/route_guide/route_guide_server.js --
   db_path=./dynamic_codegen/route_guide/route_guide_db.json &

$ c:/tools/nodejs/node.exe
   ./dynamic_codegen/route_guide/route_guide_client.js --
   db_path=./dynamic_codegen/route_guide/route_guide_db.json
```

Server in nodejs

```
// from C:/src/grpc/examples/node/dynamic_codegen/greeter_server.js
var PROTO_PATH = __dirname + '/../../protos/helloworld.proto';
var grpc = require('grpc');
var hello_proto = grpc.load(PROTO_PATH).helloworld;

function sayHello(call, callback) {// Implements the SayHello RPC method.
  callback(null, {message: 'Hello ' + call.request.name});
}
function main() {// Starts an RPC server for the Greeter service
  var server = new grpc.Server();
  server.addProtoService(hello_proto.Greeter.service, {sayHello:
   sayHello});
  server.bind('0.0.0.0:50051', grpc.ServerCredentials.createInsecure());
  server.start();
}
main();
```

Client in nodejs

```
// from C:/src/grpc/examples/node/dynamic_codegen/greeter_client.js
var PROTO_PATH = __dirname + '/../../protos/helloworld.proto';
var grpc = require('grpc');
var hello_proto = grpc.load(PROTO_PATH).helloworld;

function main() {
  var client = new hello_proto.Greeter('localhost:50051',
      grpc.credentials.createInsecure());
  var user;
  if (process.argv.length >= 3) { user = process.argv[2];
  } else { user = 'world';  }
  client.sayHello({name: user}, function(err, response) {
    console.log('Greeting:', response.message);
  });
}
main();
```

gRPC References

1. http://www.grpc.io/

Map Reduce

Dr.Mohsin Ahmed

FDP on HPC at EE.IITB, 1/7/2017

Map Reduce

- Framework to process data in 2 phases in parallel.
- Mapreduce patented by google
- Hadoop from Apache – Yahoo
  - Hadoop is not fault tolerant, java, HDFS, Amazon S3.
- Programming – 2 base classes: Map and Reduce
  - Parallel processing of independent data 10^10 web documents
  - Consolidating 100Tb of logs records daily.

Uses

- Word count
- Log processing for accounting.
- Indexing documents (webpages).
- Computing pagerank.
- Billing ads
- Detecting fraud
- Searching
- Machine learning (spelling, voice, spam)

Sequential MR in python

```
C:\> Python
>>> print 'primes=',filter(None, map(lambda
   y:y*reduce(lambda x,y:x*y!=0,map(lambda
   x,y=y:y%x,range(2,int(pow(y,0.5)+1))),1),r
   ange(2,100)))
primes=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
   37, 41, 43, 47, 53, ..,97]
```

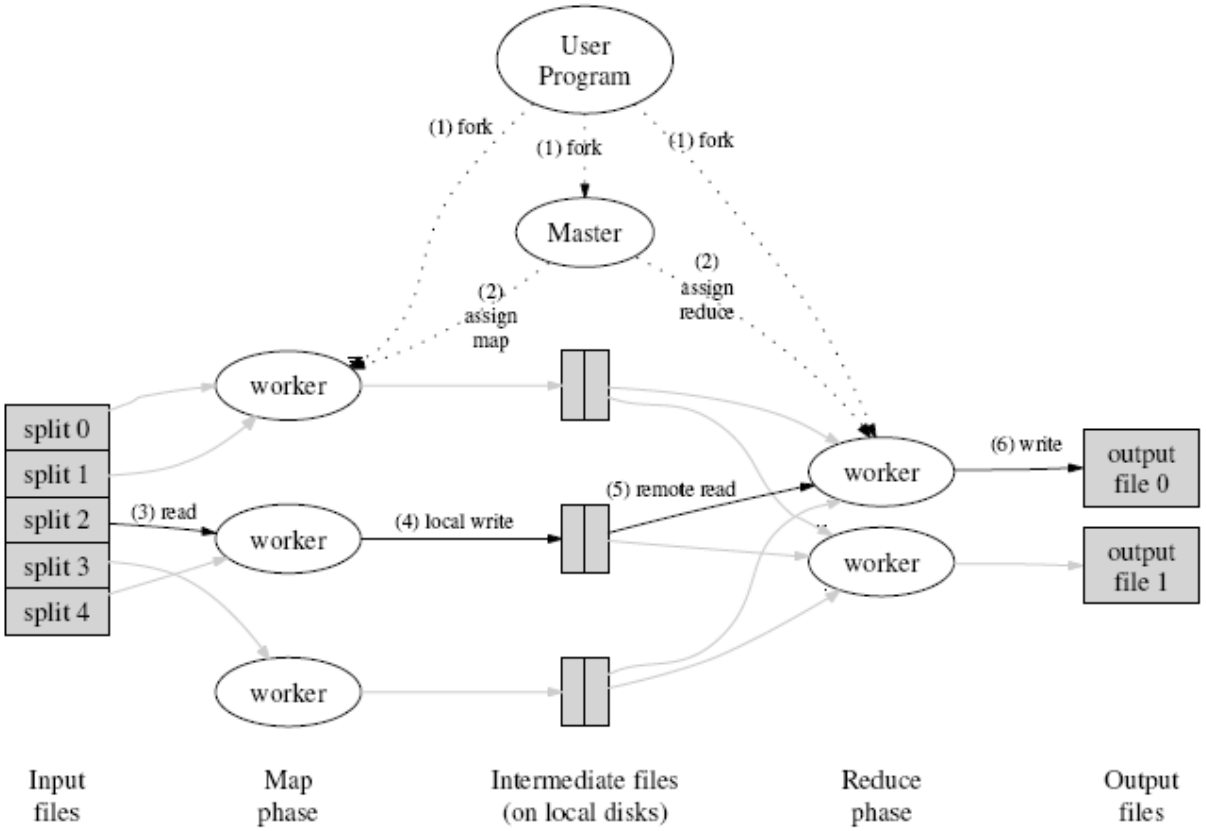
Parallel Wordcount as Map Reduce

```
1.  // Word count
2.  map(String key, String doc):
3.  foreach word in doc:
4.    sendToReduce(word, "1");

1.  reduce(String word, Iterator counts):
2.    wordcount = 0;
3.    for c in counts:
4.      wordcount += c;
5.    Output(word, wordcount);
```

MR dataflow

http://code.google.com/edu/parallel/mapreduce-tutorial.html



MR 1

- The MapReduce library in the user program first shards the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece.
- It then starts up many copies of the program on a cluster of machines.
- One of the copies of the program is special: the **master.**
- The rest are **workers** that are assigned work by the master.
- There are M **map** tasks and R **reduce** tasks to assign.
- The master picks idle workers and assigns each one a **map** task or a **reduce** task.

MR 2

- A worker who is assigned a **map** task reads the contents of the corresponding input shard.
- It parses key/value pairs out of the input data and passes each pair to the user-defined **Map** function.
- The intermediate key/value pairs produced by the **Map** function are buffered in memory.
- Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function.
- The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the **reduce** workers.

MR 3

- When a **reduce** worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the **map** workers.
- When a **reduce** worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.
- If the amount of intermediate data is too large to fit in memory, an external sort is used.

MR 4

- The **reduce** worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's **Reduce** function. The output of the **Reduce** function is appended to a final output file for this **reduce** partition.

MR 5

- When all **map** tasks and **reduce** tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.
- After successful completion, the output of the MapReduce execution is available in the R output files.

MR uses 1 - **Distributed Grep**

- The **map** function emits a line if it matches a given pattern.
- The **reduce** function is an identity function that just copies the supplied intermediate data to the output.

MR uses 2- **Count of URL Access Frequency**

- The **map** function processes logs of web page requests and outputs <URL, 1>.
- The **reduce** function adds together all values for the same URL and emits a <URL, total count> pair.

MR uses 3 - **Reverse Web-Link Graph**

- The **map** function outputs <target, source> pairs for each link to a target URL found in a page named "source".
- The **reduce** function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>.

MR uses 4 - **Term-Vector per Host**

- A term vector summarizes the most important words that occur in a document or a set of documents as a list of <word, frequency> pairs.
- The **map** function emits a <hostname, term vector> pair for each input document (where the hostname is extracted from the URL of the document).
- The **reduce** function is passed all per-document term vectors for a given host.
- It adds these term vectors together, throwing away infrequent terms, and then emits a final <hostname, term vector> pair.

MR uses 5 - **Inverted Index**

- The **map** function parses each document, and emits a sequence of <word, document ID> pairs.
- The **reduce** function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
- The set of all output pairs forms a simple inverted index.
- It is easy to augment this computation to keep track of word positions.

References

- http://en.wikipedia.org/wiki/MapReduce
- http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html