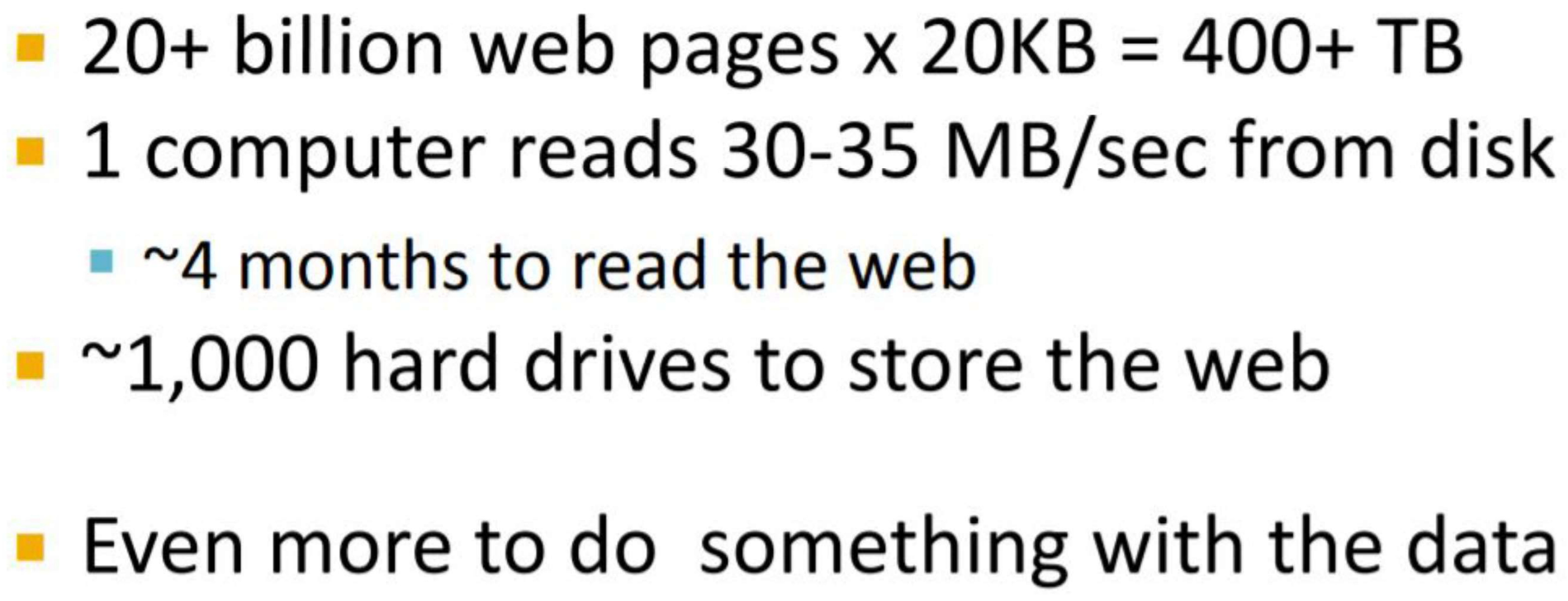
Distributed File System
and
Parameter Server

Introduction

- Modern data-mining applications, often called “big-data” analysis, require us to manage very large amounts of data, which may not be able to fit or processed by a single machine.
- For many applications, the regularity of data enables us to exploit parallel processing.
- Important applications:
    - The ranking of Web pages by importance, which involves an iterated matrix-vector multiplication where the dimension is many billions. This application, called “PageRank”.
    - Searches in “friends” networks at social-networking sites, which involve graphs with hundreds of millions of nodes and many billions of edges.
- To deal with these problems, with the help of a special type of parallel computing, viz. *Cluster computing*, a new software stack is evolved.

Motivation (Google Example)



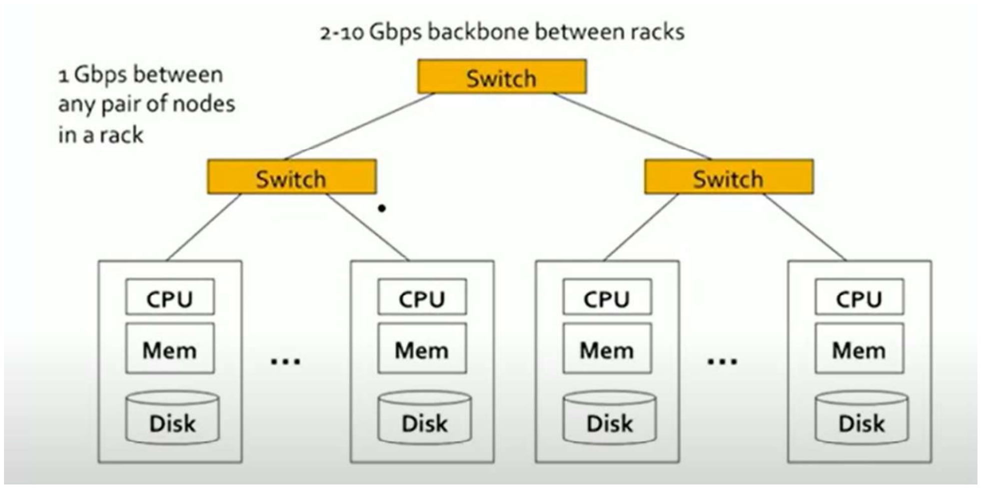
- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web

- Even more to do something with the data

Distributed File System

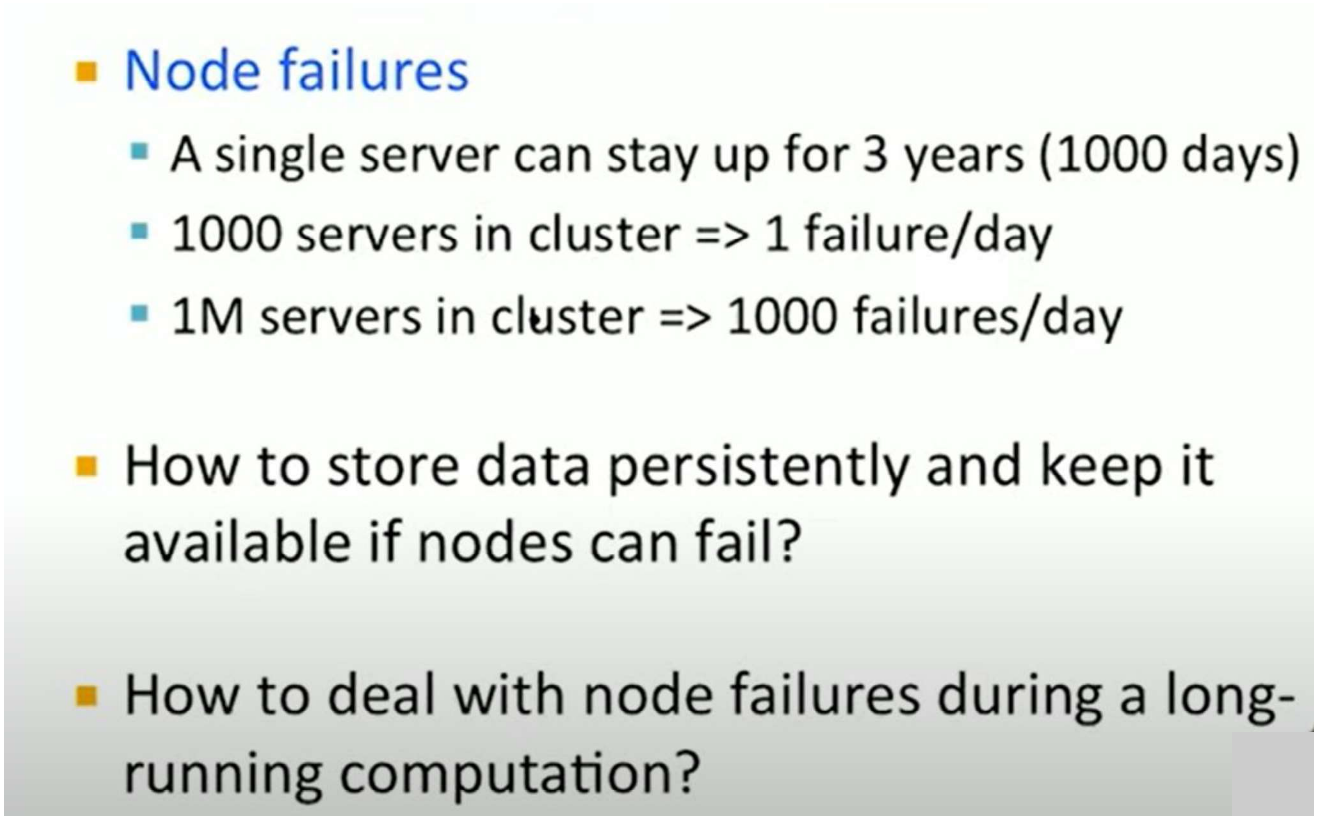
- Generally, computing is done on a single processor, with its main memory, cache, and local disk (a compute node).
- Classic parallelization - done on special-purpose parallel computers with many processors and specialized hardware - Expensive and not very scalable or reliable.
- Large-scale Web services - more and more computing done on installations with thousands of compute nodes operating more or less independently - Less expensive and more scalable and reliable.
- Less expensive as compute nodes are commodity hardware.
- Designed to get their parallelism not from a “supercomputer,” but from “computing clusters”
- “Computing clusters” are nothing but large collections of commodity hardware, including conventional processors (“compute nodes”) connected by Ethernet cables or inexpensive switches

DFS Architecture


2-10 Gbps backbone between racks
1 Gbps between any pair of nodes in a rack
Switch
Switch
Switch
CPU
Mem
Disk
...
CPU
Mem
Disk
CPU
Mem
Disk
...
CPU
Mem
Disk

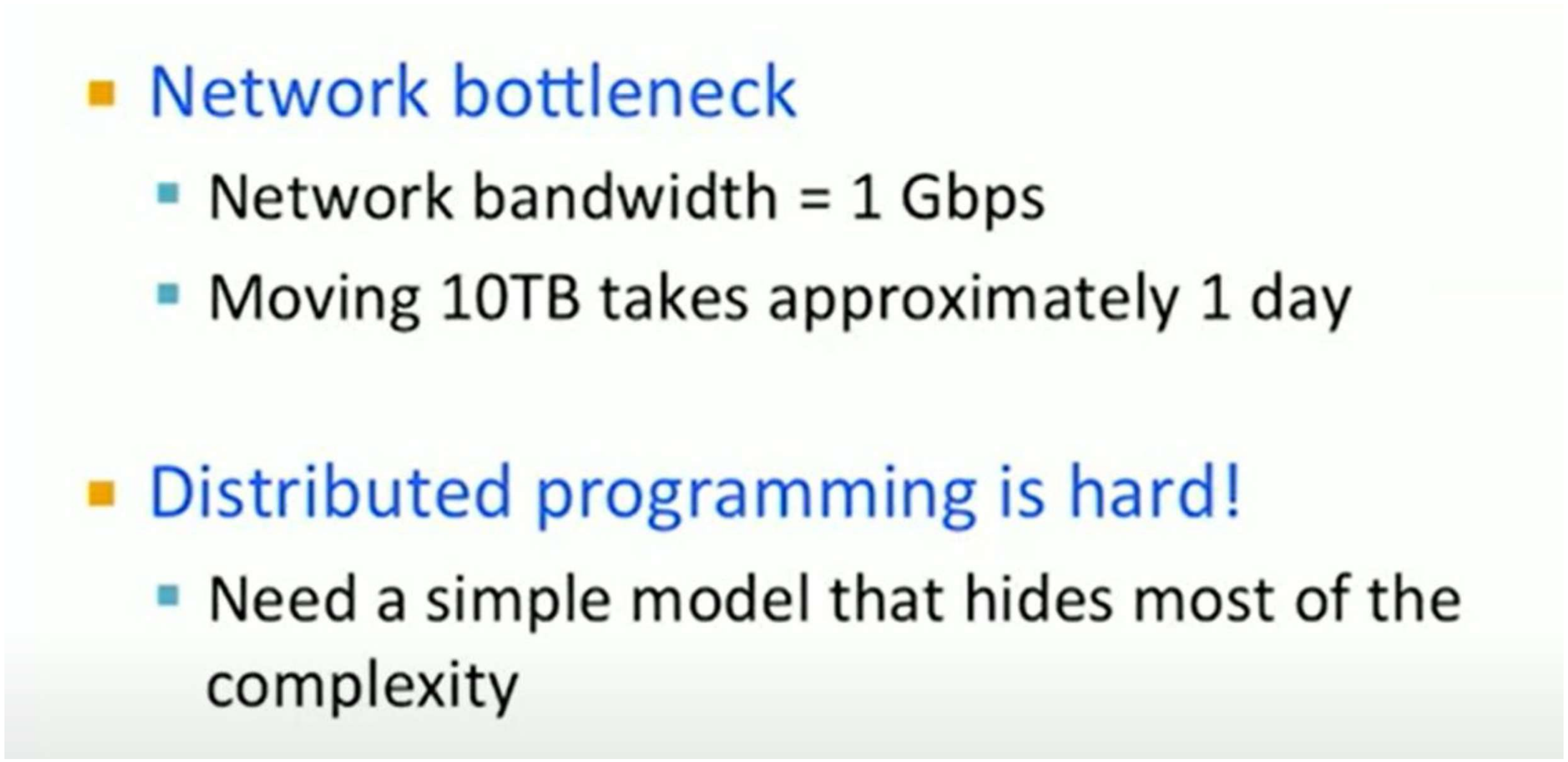
- Each rack has 16-64 commodity Linux nodes
- The nodes on a single rack are connected by a network, typically gigabit Ethernet.
- Racks are connected by another level of network or a switch.
- The bandwidth of inter-rack communication is somewhat greater than the intrarack Ethernet.

Cluster Computing Challenges



- Node failures
  - A single server can stay up for 3 years (1000 days)
  - 1000 servers in cluster => 1 failure/day
  - 1M servers in cluster => 1000 failures/day
- How to store data persistently and keep it available if nodes can fail?
- How to deal with node failures during a long-running computation?

Cluster Computing Challenges

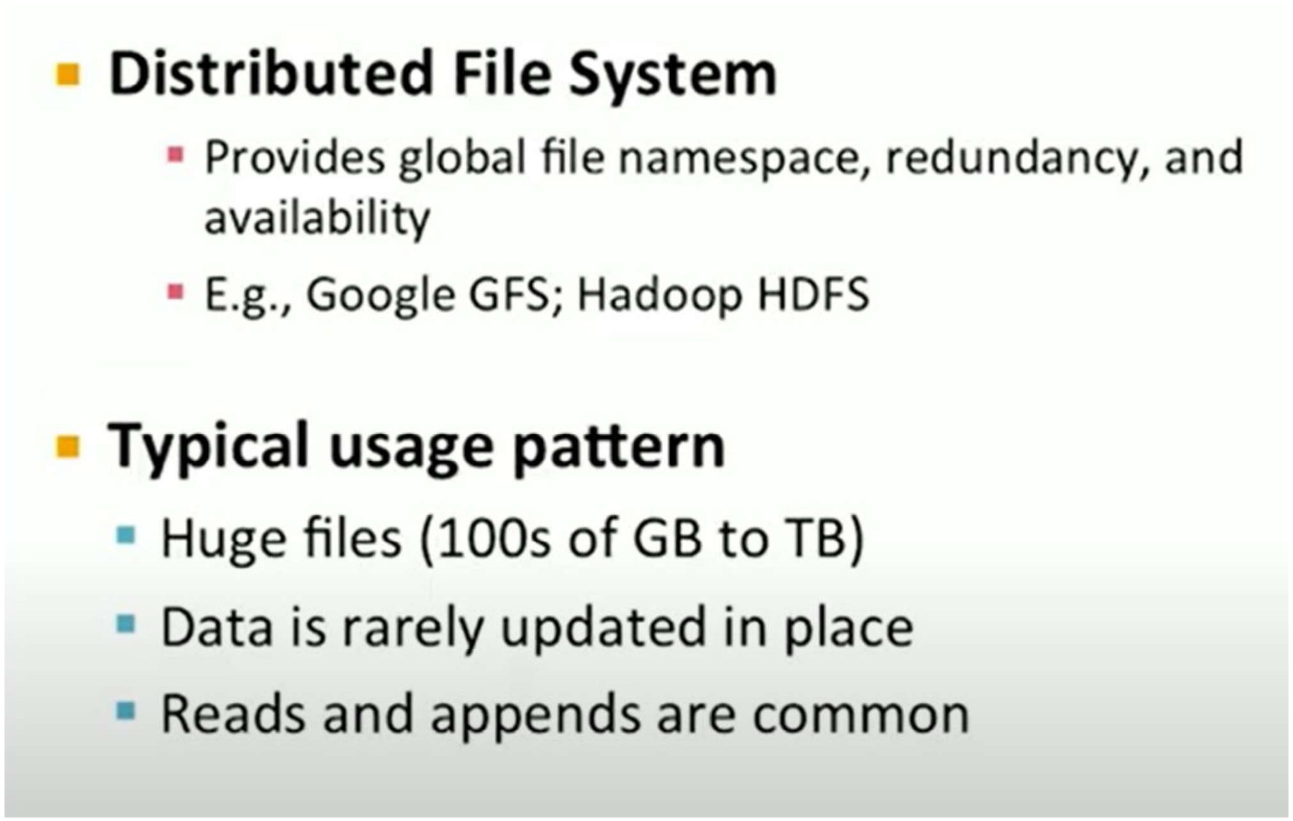


- Network bottleneck
  - Network bandwidth = 1 Gbps
  - Moving 10TB takes approximately 1 day
- Distributed programming is hard!
  - Need a simple model that hides most of the complexity

Infrastructure (File system)

- Reliability - Store files multiple times (redundancy)
- Network Bottleneck - Bring computation close to data, not move the data itself

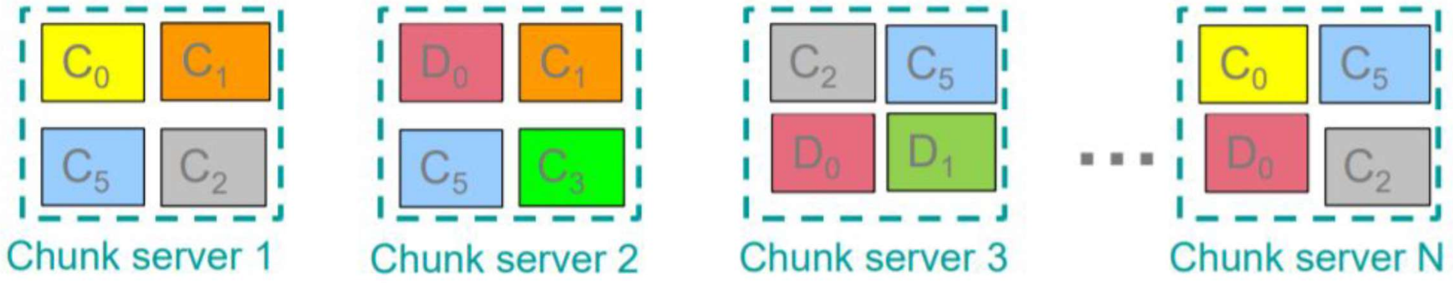
Stable Storage



- **Distributed File System**
  - Provides global file namespace, redundancy, and availability
  - E.g., Google GFS; Hadoop HDFS
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Reliable distributed file system for petabyte scale
- Data kept in 64-megabyte "chunks" spread across thousands of machines
- Each chunk replicated, usually 3 times, on different machines
  - Seamless recovery from disk or machine failure



Distributed File System

- Chunk Servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- Master node
  - a.k.a. Name Nodes in HDFS
  - Stores metadata
  - Might be replicated
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunkservers to access data

DFS Implementations

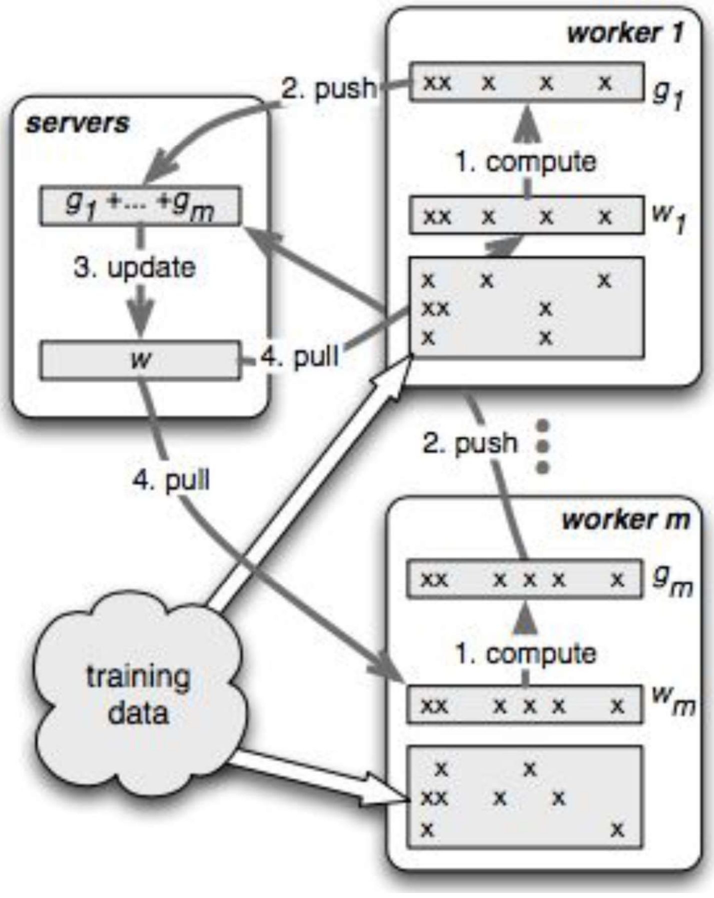
There are several distributed file systems of the type we have described that are used in practice. Among these:

1. The *Google File System* (GFS), the original of the class.

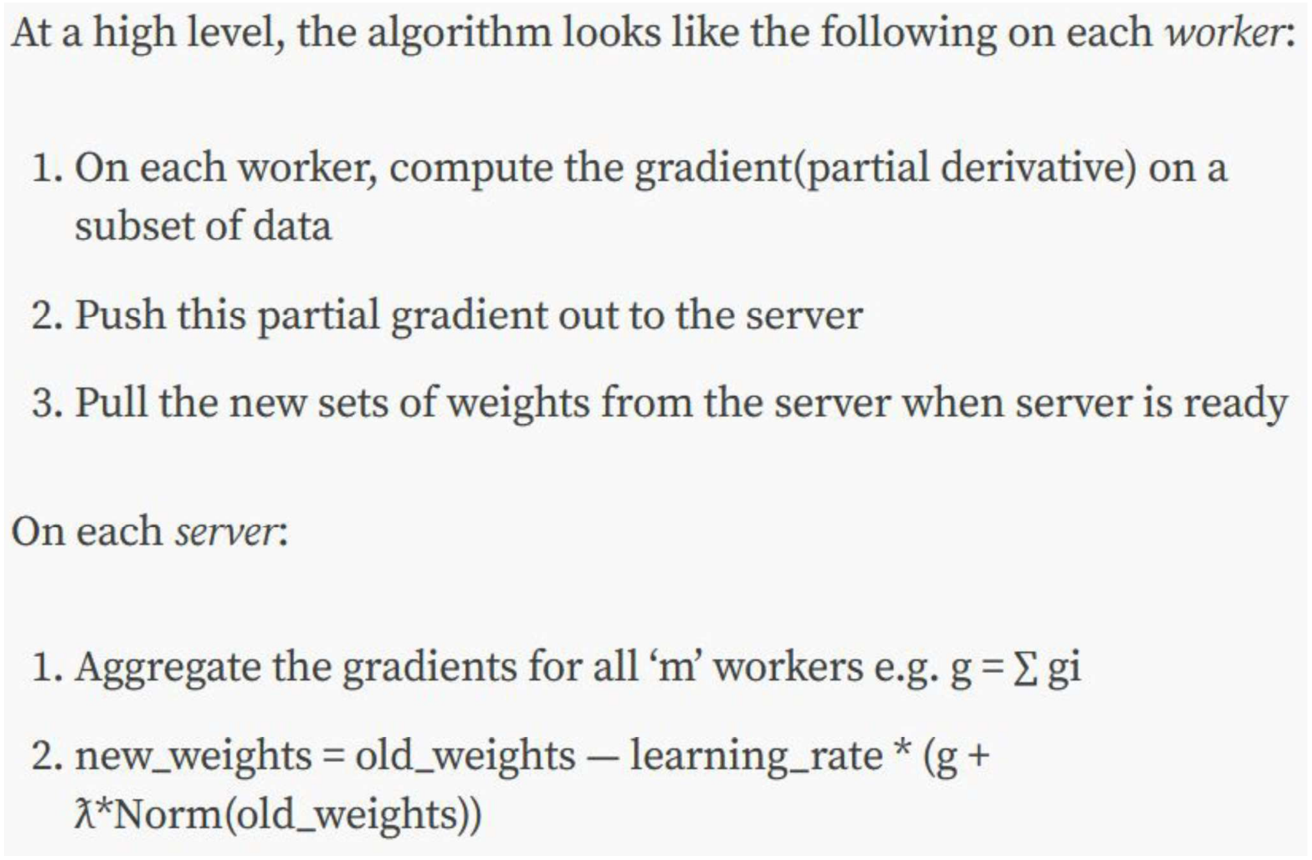
2. *Hadoop Distributed File System* (HDFS), an open-source DFS used with Hadoop, an implementation of MapReduce (see Section 2.2) and distributed by the Apache Software Foundation.

3. *Colossus* is an improved version of GFS, about which little has been published. However, a goal of Colossus is to provide real-time file service.

Parameter Server for Distributed Machine Learning



High-level View



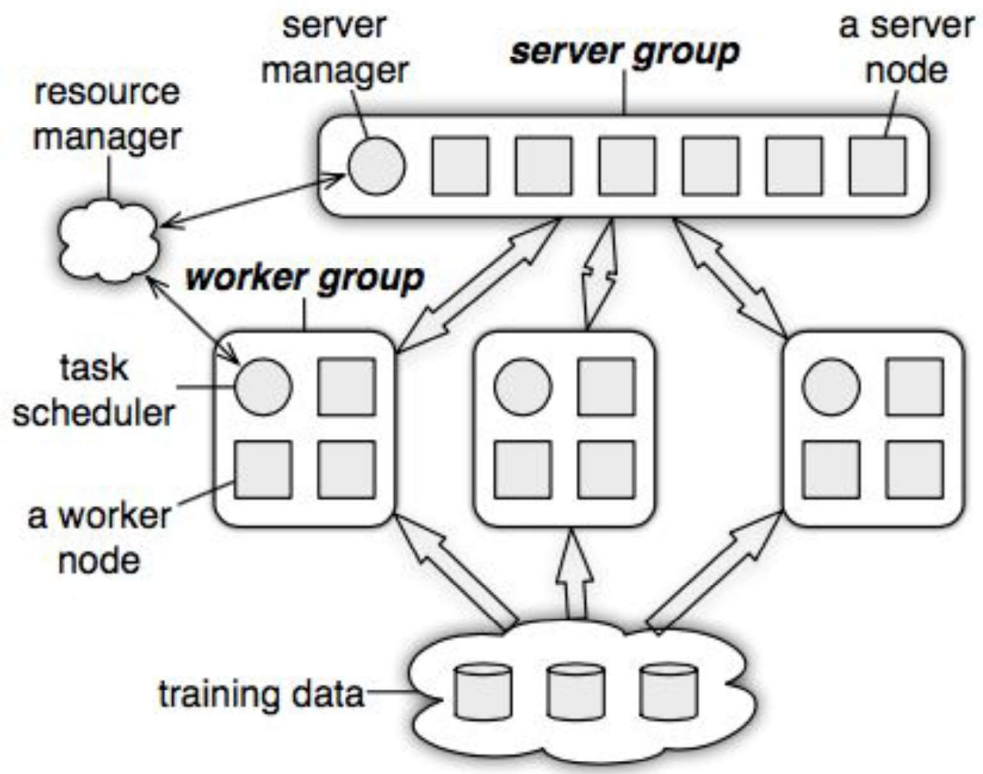
At a high level, the algorithm looks like the following on each *worker*:

1. On each worker, compute the gradient(partial derivative) on a subset of data
2. Push this partial gradient out to the server
3. Pull the new sets of weights from the server when server is ready

On each *server*:

1. Aggregate the gradients for all ‘m’ workers e.g. $g = \Sigma gi$
2. new_weights = old_weights — learning_rate * (g + λ*Norm(old_weights))

Architecture



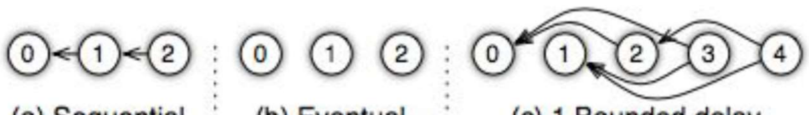
Primitives

- Key-Value API
  - Use key-value pairs for communicating the shared parameters.

- Range-based “push” and “pull”
  - Optimize for the network bandwidth usage.

- Asynchronous tasks and dependency
  - Help with the overall control flow.
  - A task marked as complete when all subtasks return.

- Flexible consistency models



(a) Sequential (b) Eventual (c) 1 Bounded delay

References

- https://medium.com/coinmonks/parameter-server-for-distributed-machine-learning-fd79d99f84c3
- http://infolab.stanford.edu/~ullman/mmds/book0n.pdf