UC San Diego

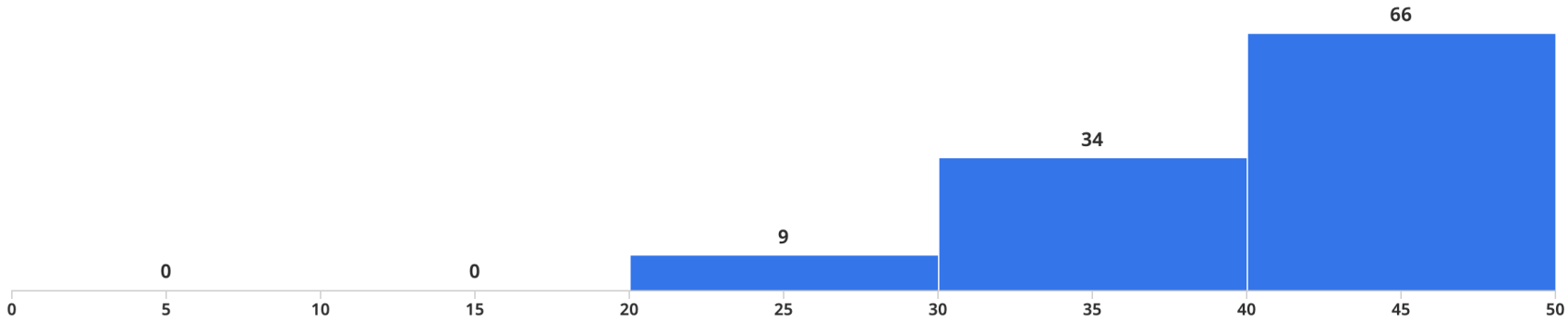
DSC 102
Systems for Scalable Analytics

Rod Albuyeh

Topic 3: Parallel and Scalable Data Processing
Part 3: Data Parallelism

Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book
Ch. 5, 6.1, 6.3, 6.4 of MLSys Book

Admin: Midterm Analysis



Minimum	Median	Maximum	Mean
22.0	42.0	50.0	40.41

Most points marked off for task graph comprehension, where performance was ~60% for the class. I'll release a walkthrough video on it.

Admin: PA2

- ”Black Box” infrastructure running on Kubernetes instead of the fully transparent AWS view.
- More of an exercise in the Spark API but limited customizability.
- Note: 2021 pandemic version of this course ran PA2 on EMR.

Outline

- Basics of Parallelism
  - Task Parallelism; Dask
  - Single-Node Multi-Core; SIMD; Accelerators
- Basics of Scalable Data Access
  - Paged Access; I/O Costs; Layouts/Access Patterns
  - Scaling Data Science Operations
- Data Parallelism: Parallelism + Scalability
  - Data-Parallel Data Science Operations
  - Optimizations and Hybrid Parallelism

Introducing Data Parallelism

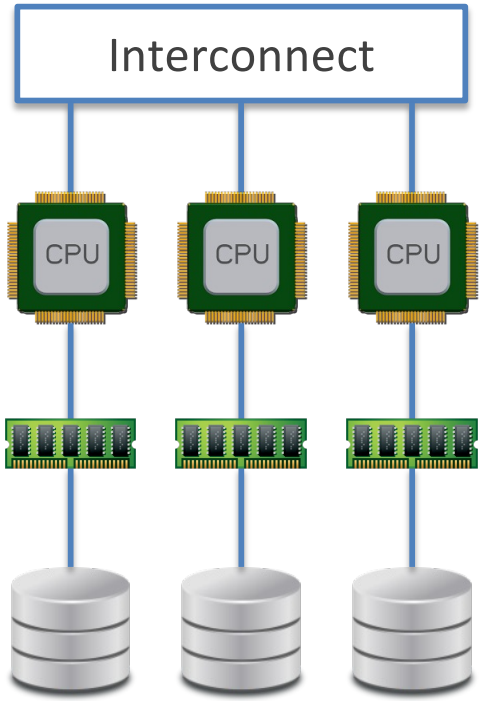
Basic Idea of Scalability: Split data file (virtually or physically) and *stage reads/writes* of its pages between disk and DRAM

Q: *What is "data parallelism"?*

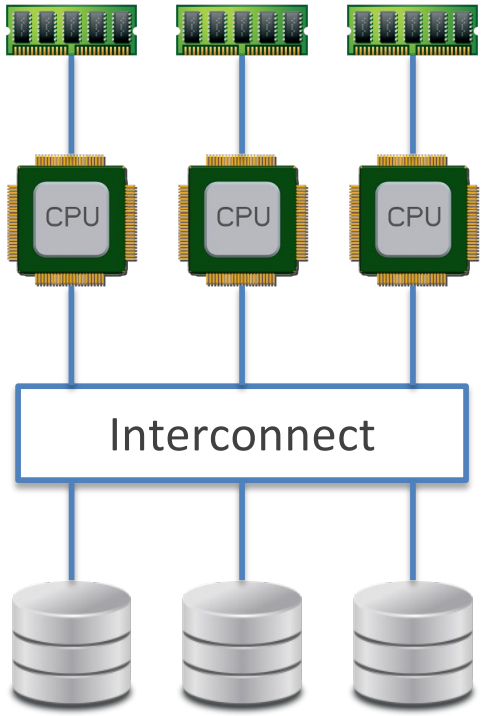
Data Parallelism: Partition large data file *physically* across nodes/workers; within worker: DRAM-based or disk-based

- The most common approach to marrying *parallelism* and *scalability* in data systems
- Generalization of SIMD and SPMD idea from parallel processors to large-scale data and multi-worker/multi-node setting
- Distributed-memory vs Distributed-disk

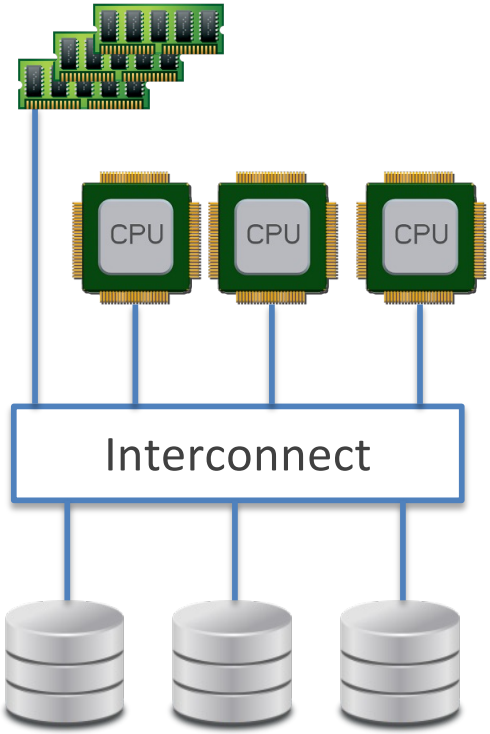
3 Paradigms of Multi-Node Parallelism



Shared-Nothing Parallelism



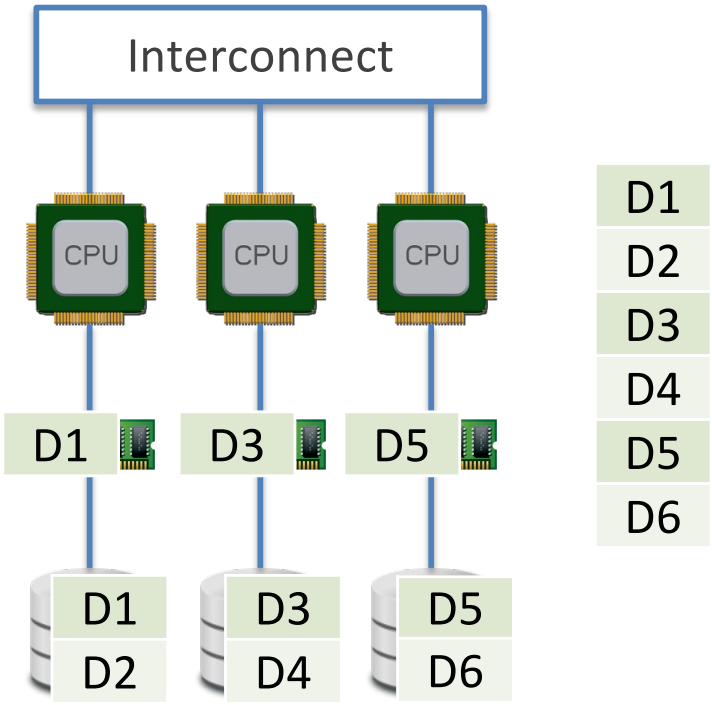
Shared-Disk Parallelism



Shared-Memory Parallelism

Data parallelism is technically *orthogonal* to these 3 paradigms but most commonly paired with shared-nothing

Shared-Nothing Data Parallelism

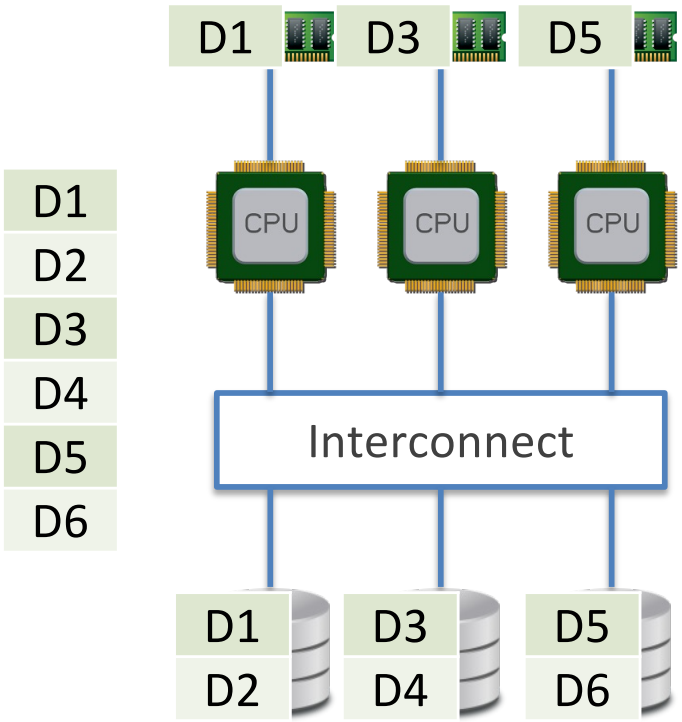


Shared-Nothing Parallel Cluster

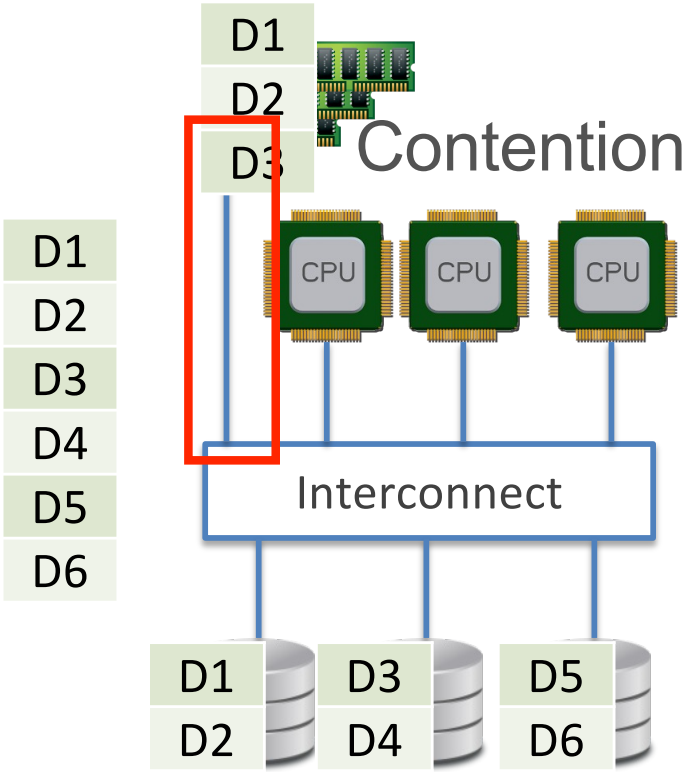
D1
D2
D3
D4
D5
D6

- Partitioning a data file across nodes is aka **sharding**
- Part of a stage in data processing workflows called **Extract-Transform-Load** (ETL)
- ETL is an umbrella term for all kinds of processing done to the data file before it is ready for users to query, analyze, etc.
  - Sharding, compression, file format conversions, etc.

Data Parallelism in Other Paradigms?



Shared-Disk
Parallel Cluster



Shared-Memory
Parallel Cluster

Data Partitioning Strategies

- Row-wise/*horizontal* partitioning is most common (sharding)
- 3 common schemes (given k nodes):
  - **Round-robin**: assign tuple i to node i MOD k
  - **Hashing-based**: needs hash partitioning attribute(s)
  - **Range-based**: needs ordinal partitioning attribute(s)
- **Tradeoffs:**
  - For Relational Algebra (RA) and SQL:
    - Hashing-based most common in practice for RA/SQL
    - Range-based often good for range predicates in RA/SQL
  - But all 3 are often OK for many ML workloads (why?)
- **Replication** of partition across nodes (e.g., 3x) is common to enable *"fault tolerance"* and better parallel *runtime performance*

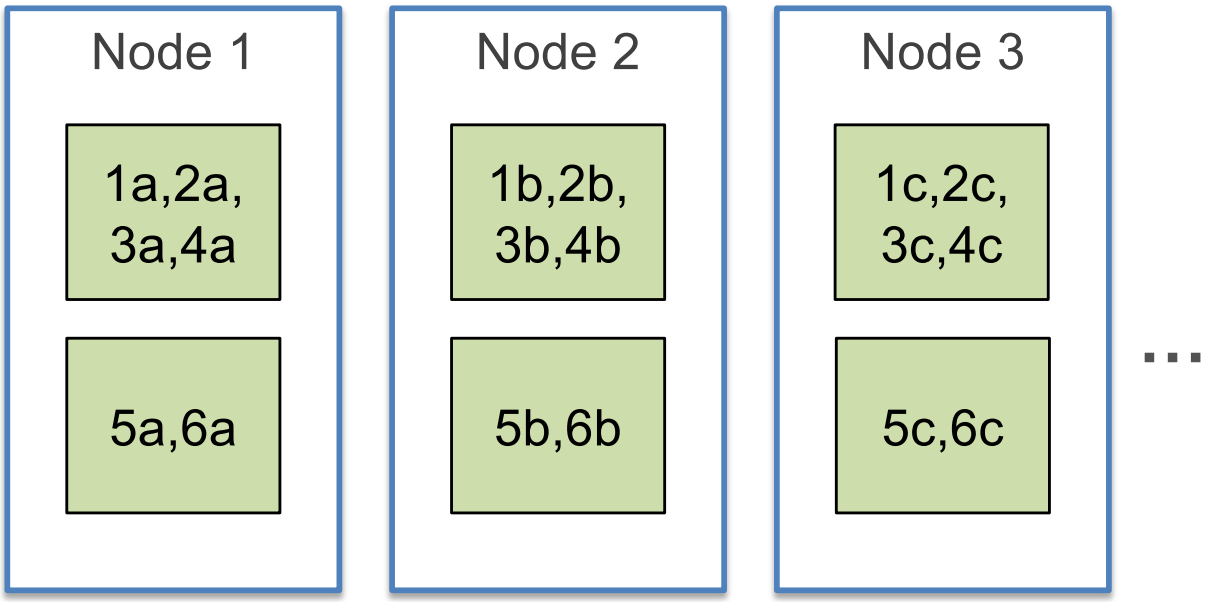
Other Forms of Data Partitioning

- Just like with disk-aware data layout on single-node, we can partition a large data file across workers in other ways too:

R

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Columnar Partitioning

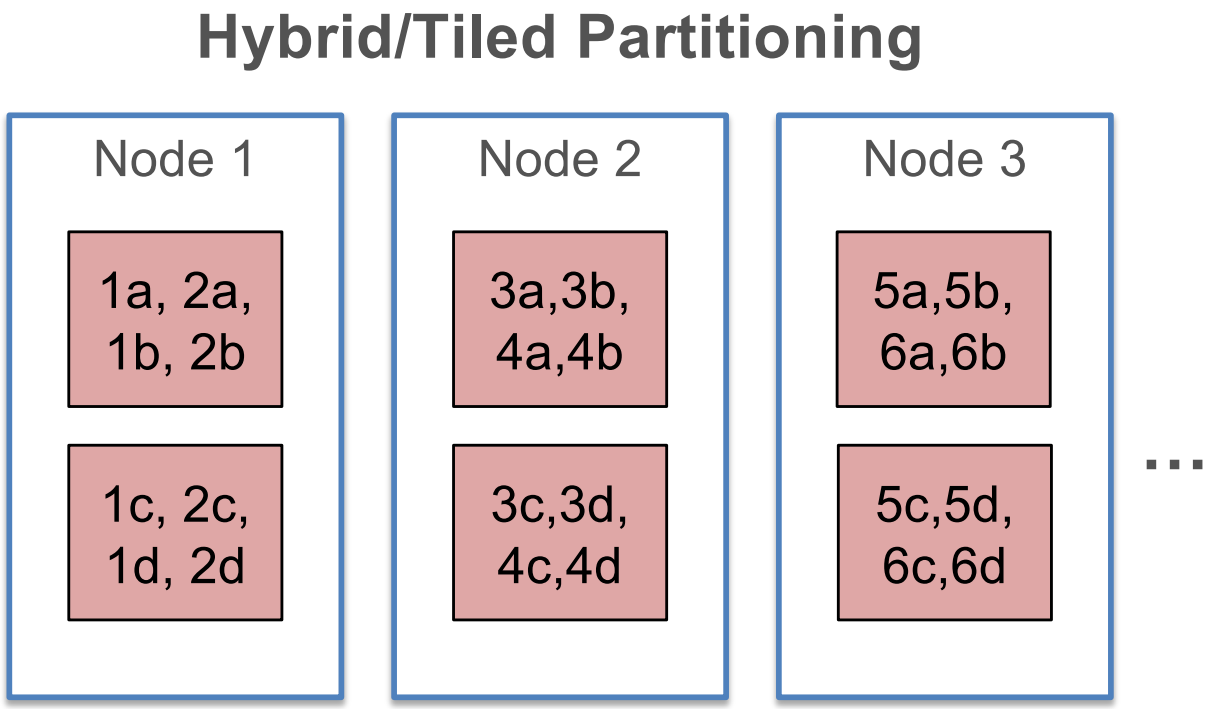


Other Forms of Data Partitioning

- Just like with disk-aware data layout on single-node, we can partition a large data file across workers in other ways too:

R

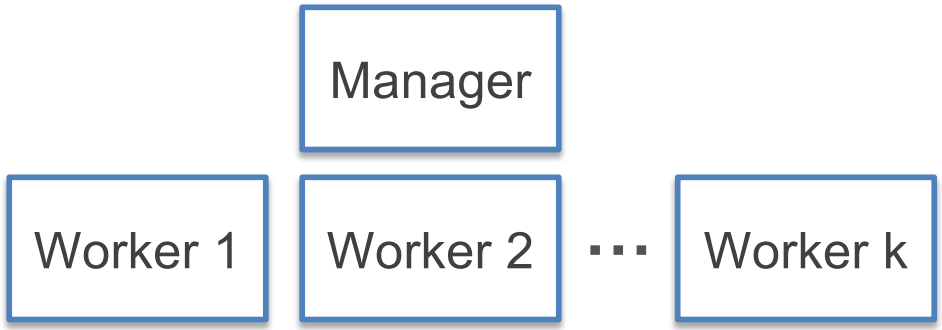
A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d



Cluster Architectures

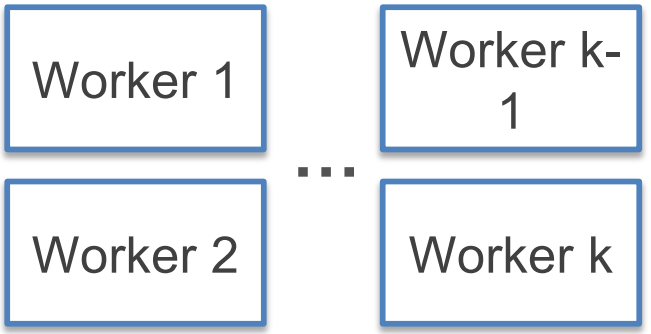
***Q:** What is the protocol for cluster nodes to talk to each other?*

Manager-Worker Architecture



- 1 (or few) special node called **Manager** (aka “Server” or archaic “Master”); 1 or more **Workers**
- Manager tells workers what to do and when to talk to other nodes
- Most common in data systems (e.g., Dask, Spark, par. RDBMS, etc.)

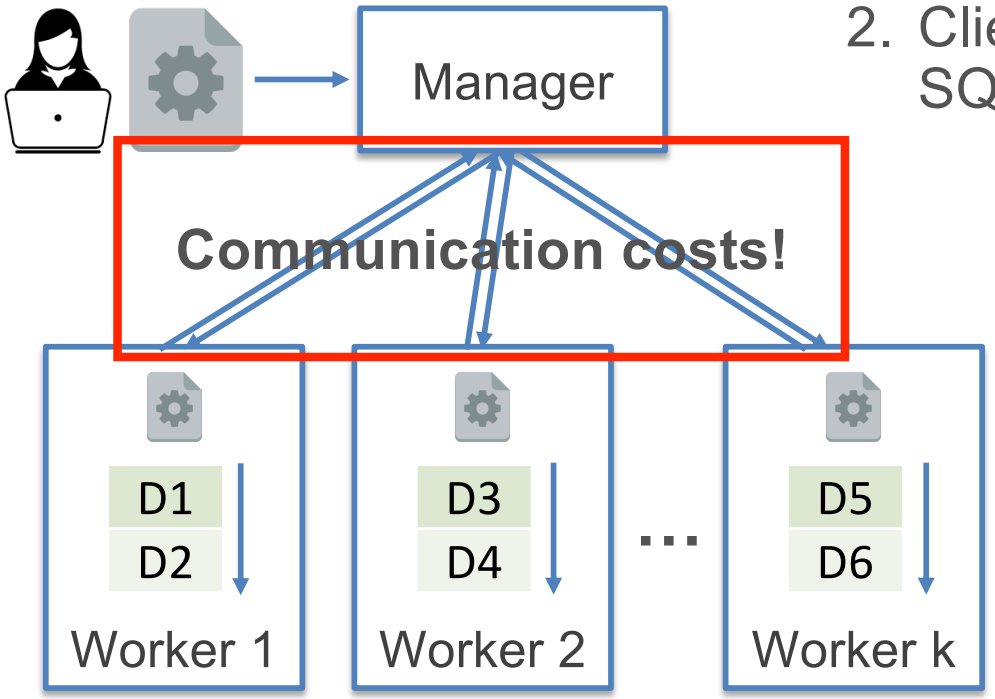
Peer-to-Peer Architecture



- No special manager
- Workers talk to each other directly
- E.g., Horovod
- Aka Decentralized (vs Centralized)

Bulk Synchronous Parallelism (BSP)

- Most common protocol of data parallelism in data systems (e.g., in parallel RDBMSs, Hadoop, Spark)
- Shared-nothing sharding + manager-worker architecture



Aka **(Barrier) Synchronization**

1. Sharded data file on workers
2. Client gives program to manager (e.g., SQL query, ML training, etc.)
3. Manager *divides* first piece of *work* among workers
4. Workers work *independently* on self's data partition (cross-talk can happen if Manager asks)
5. Worker sends partial results to Manager
6. Manager **waits** till all k done
7. Go to step 3 for next piece

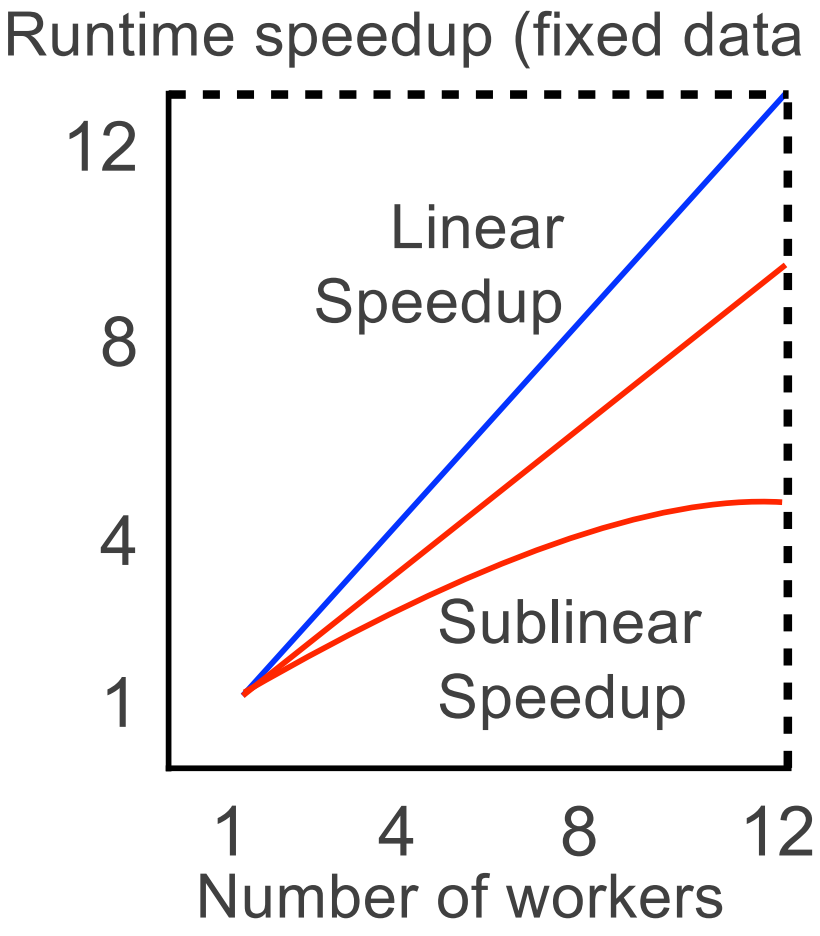
Speedup Analysis/Limits of of BSP

***Q:** What is the speedup yielded by BSP?*

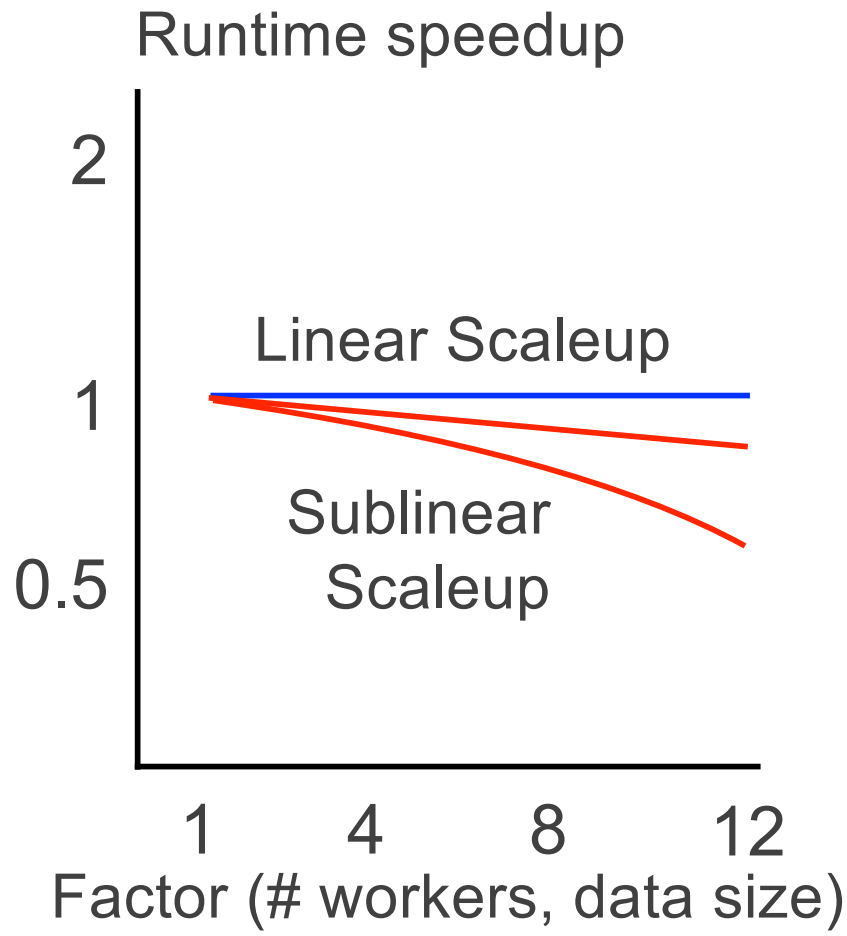
$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given k (>1) workers}}$$

- Cluster overhead factors that hurt speedup**:**
  - **Per-worker:** startup cost; tear-down cost
  - **On manager:** dividing up the work; collecting/unifying partial partial results from workers
  - **Communication costs:** talk between manager-worker and across workers (when asked by manager)
  - Barrier synchronization suffers from "**stragglers**" (workers that fall behind) due to skews in shard sizes and/or worker capacities

Quantifying Benefit of Parallelism



Speedup plot / Strong scaling



Scaleup plot / Weak scaling

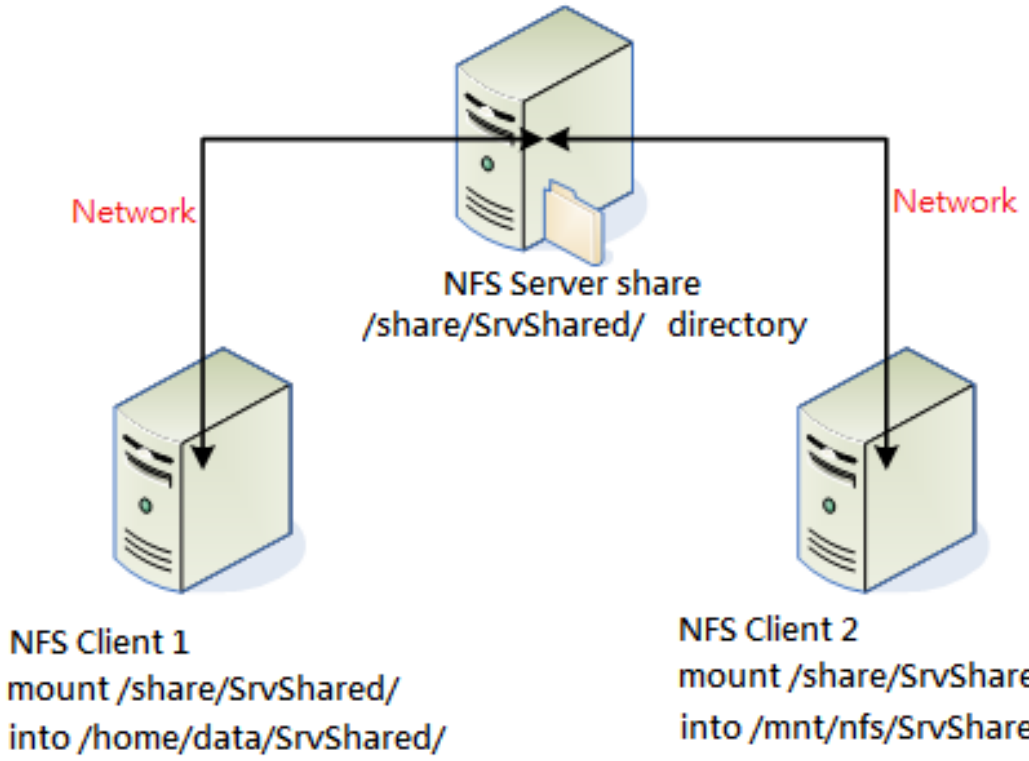
***Q:** Is superlinear speedup/scaleup ever possible?*

Distributed Filesystems

- Recall definition of file; *distributed file* generalizes it to a cluster of networked disks and OSs
- **Distributed filesystem (DFS)** is a cluster-resident filesystem to manage distributed files
  - A *layer of abstraction* on top of local filesystems
  - Nodes manage local data as if they are local files
  - *Illusion of a one global file*: DFS APIs let nodes access data sitting on other nodes
  - 2 main variants: Remote DFS vs In-Situ DFS
    - **Remote DFS:** Files reside elsewhere and read/written on demand by workers
    - **In-Situ DFS:** Files resides on cluster where workers exist

Network Filesystem (NFS)

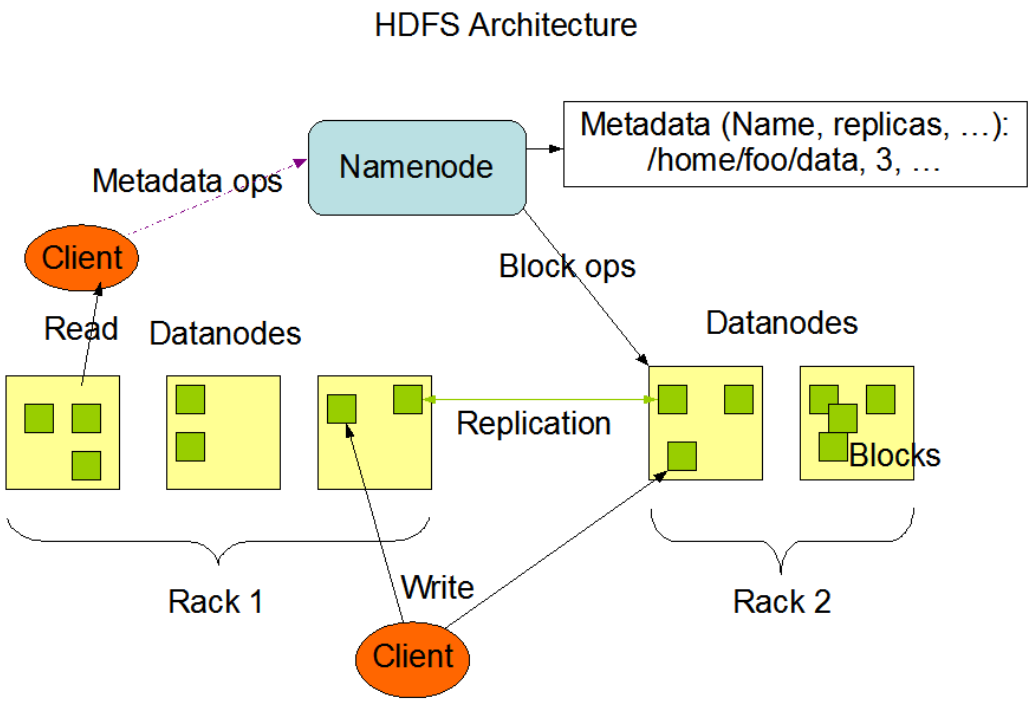
- An old remote DFS (c. 1980s) with simple client-server architecture for *replicating* files over the network



- Main pro: *simplicity* of setup and usage
- But many cons:
  - Not scalable to *very* large files
  - Full data replication
  - High contention for concurrent reads/writes
  - Single-point of failure

Hadoop Distributed File System (HDFS)

- Most popular in-situ DFS (c. late 2000s); part of Hadoop; open source spinoff of Google File system (GFS)
- *Highly scalable*; scales to 10s of 1000s of nodes, PB files



- Designed for clusters of cheap commodity nodes
- *Parallel* reads/writes of sharded data "blocks"
- Replication of blocks to improve *fault tolerance*
- Cons: Read-only + batch-append (no fine-grained updates/writes)

https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Hadoop Distributed File System (HDFS)

- NameNode's roster maps data blocks to DataNodes/IPs
- A distributed file on HDFS is just a directory (!) with individual filenames for each data block and metadata files

```
${dfs.datanode.data.dir}/
├ current
|  ├── BP-526805057-127.0.0.1-1411980876842
|  |  └── current
|  |    ├── VERSION
|  |    ├── finalized
|  |    |   ├─── blk_1073741825
|  |    |   ├─── blk_1073741825_1001.meta
|  |    |   ├─── blk_1073741826
|  |    |   └─── blk_1073741826_1002.meta
|  |    └── rbw
|  └── VERSION
└ in_use.lock
```

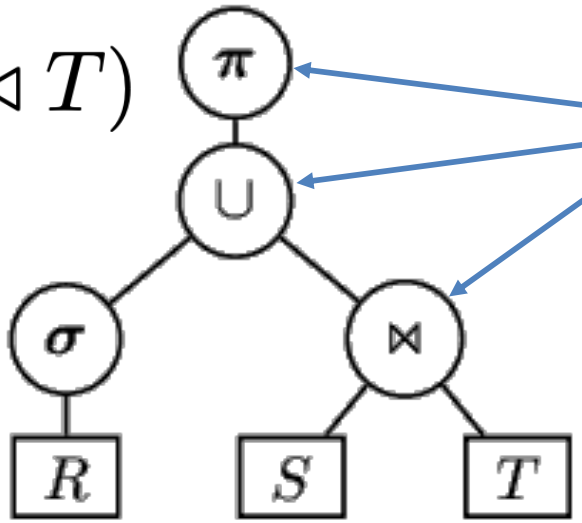
- HDFS has configurable parameters:

Parameter name	Purpose	Default value
Data block size	Splitting data into chunks	128 MB
Replication factor	Ensure data availability	3x

Data-Parallel Dataflow/Workflow

- **Data-Parallel Dataflow:** A dataflow graph with ops wherein each operation is executed in a data-parallel manner
- **Data-Parallel Workflow:** A generalization; each vertex a whole task/process that is run in a data-parallel manner

$$\pi(\sigma(R) \cup S \bowtie T)$$



All input tables are sharded

Note: In parallel environments like parallel RDBMSs and Spark: Each of these extended relational ops have scalable *data-parallel implementations*.

Q: *So how do we run data science workflows in data-parallel manner?*