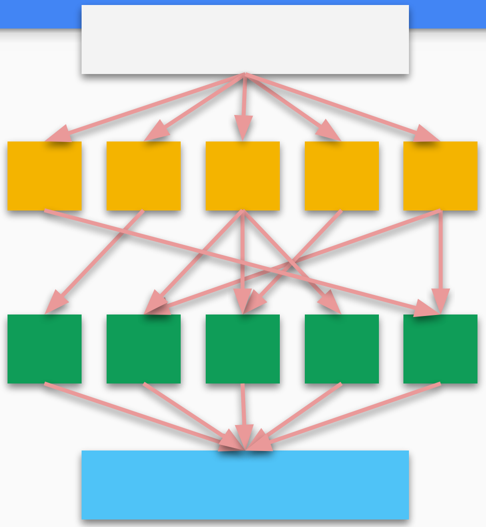
NYU Center for Data Science

Week 04.1: Distributed Storage

DS-GA 1004: Big Data

Last time: Map-Reduce

- Two functions: **mapper** and **reducer**
- **Mapper** consumes inputs, produces output:
  (*key*, *value*)
- **Reducer** consumes a single *key* and list of *values*, and produces *values*



Map-Reduce details...

- How is data **shared** over the cluster?
- How do we handle **node failure**?
- How can we **optimize for Map-Reduce** operations?

This week

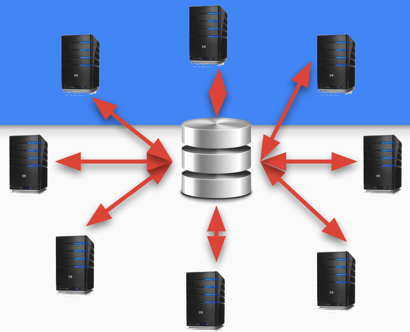
1. **Distributed storage**

2. The Hadoop distributed file system (HDFS)

   $ hadoop fs -command ...

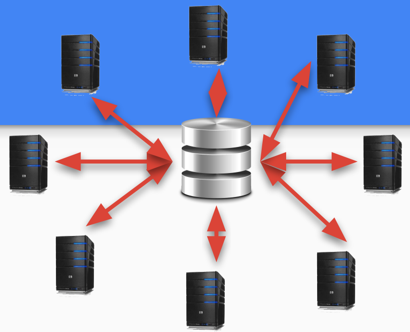
3. HDFS and Map-reduce

Start simple



- Imagine implementing Map-Reduce from scratch
  - … with all data located on a file server
- Head node sends (**mapper**, **reducer**) code to each worker **+ block of data**
- Workers send **output** back to head
  - **Mappers** → intermediate results
  - **Reducers** → final results

Start simple



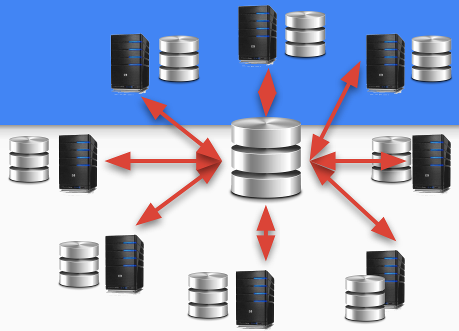
- Imagine implementing Map-Reduce from scratch
  - … with all data located on a file server
- Head node sends (**mapper**, **reducer**) code to each worker **+ block of data**
- Workers send **output** back to head
  - **Mappers** ⟶ intermediate results
  - **Reducers** ⟶ final results

This will work, but it's inefficient!

Each job moves the entire data set over the network!

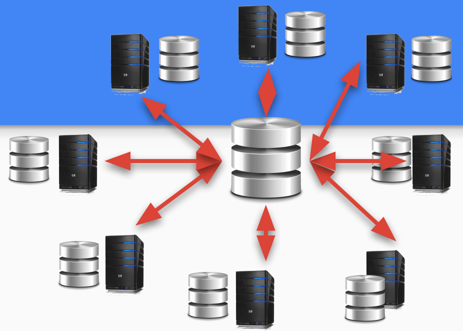
This is a **failure of locality**.

Localize all data?



- What if all data is replicated on all worker nodes?
- Head node sends (**mapper**, **reducer**) code to each worker
  + id of data block
- Workers send **output** back to head
  - **Mappers** → intermediate results
  - **Reducers** → final results

Localize all data?



- What if all data is replicated on all worker nodes?
- Head node sends (**mapper**, **reducer**) code to each worker
  + id of data block
- Workers send **output** back to head
  - **Mappers** → intermediate results
  - **Reducers** → final results

This also will work, but it’s expensive!

Each worker needs a large amount of storage.

Most workers don’t touch most of the data.

Design considerations

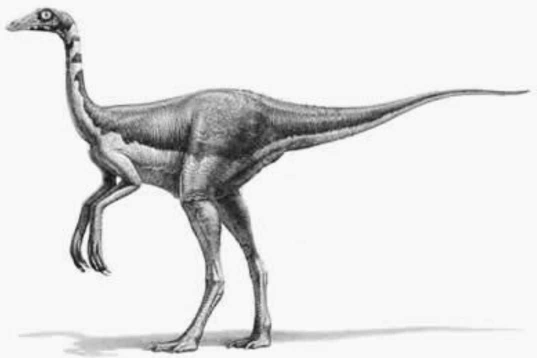
- Communication costs (bytes transferred)
- Fault tolerance
- Redundancy vs. communication
- Granularity of access
- Locality
- Common access patterns
- **Programs are small, data is big!**

Background: storage systems

Disks and disk arrays

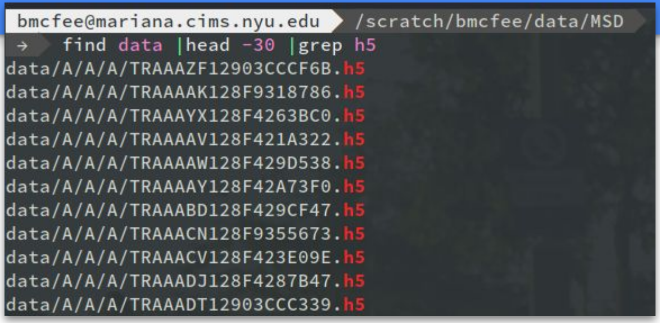
Distributed file systems

- **Distributed file systems** store data over many machines
- Different from **networked file systems** (NFS), which make centrally hosted data transparently available on many machines
- First, some background on storage systems…
  - (This will be fast and irresponsibly high-level)

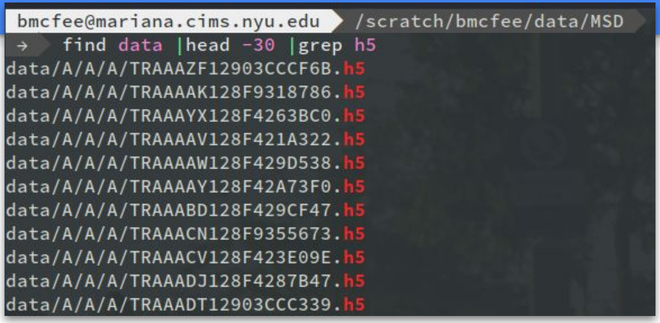


File systems and hard disks

- File systems are made of **directories** and **files**
- Disks are made of **contiguous sectors**
  - Typically 512 or 4096 bytes
  - This is the smallest addressable unit of storage
  - Each sector belongs to at most one file
- How are these reconciled?

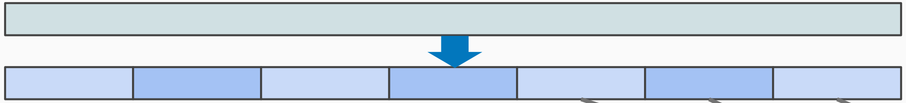


Files and blocks

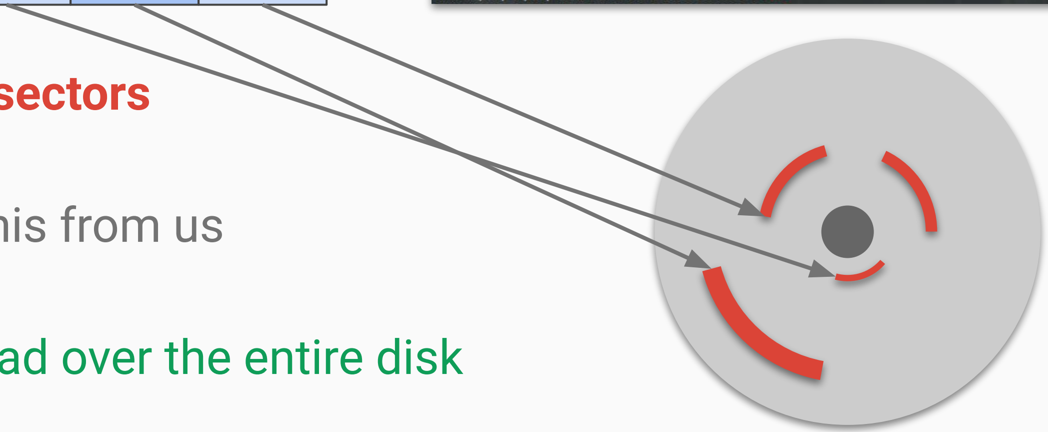


```
bmcfee@mariana.cims.nyu.edu  /scratch/bmcfee/data/MSD
→  find data |head -30 |grep h5
data/A/A/A/TRAAAZF12903CCCF6B.h5
data/A/A/A/TRAAAAK128F9318786.h5
data/A/A/A/TRAAAYX128F4263BC0.h5
data/A/A/A/TRAAAAV128F421A322.h5
data/A/A/A/TRAAAAW128F429D538.h5
data/A/A/A/TRAAAAY128F42A73F0.h5
data/A/A/A/TRAAABD128F429CF47.h5
data/A/A/A/TRAAACN128F9355673.h5
data/A/A/A/TRAAACV128F423E09E.h5
data/A/A/A/TRAAADJ128F4287B47.h5
data/A/A/A/TRAAADT12903CCC339.h5
```

- **Files** are broken into **blocks**
  - Block size ≥ sector size



- Each **block** is mapped onto **sectors**
- The file system (OS) hides this from us
- **Result**: a single file can spread over the entire disk



Large-scale storage

- What if our data is **too big** for a single disk?
- What happens if a **disk fails**?
- Throughput is limited by moving the **head**
  - … what if we have multiple simultaneous read/write requests?

Redundant array of inexpensive disks (RAID)

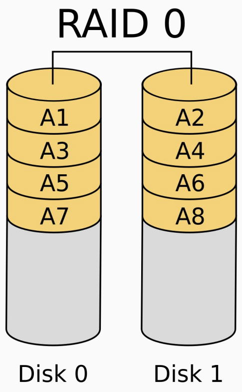
- **RAID** systems distribute storage over multiple disks in a single machine
  - But look like a single volume to the OS

- Goals of **RAID**:
  - **Capacity**
  - **Reliability**
  - **Throughput**

- Comes in multiple “levels” with different reliability-capacity trade-offs

[Patterson, Gibson, & Katz, SIGMOD 1988]

Commonly used RAID levels



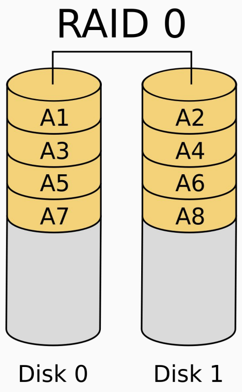
Striping only

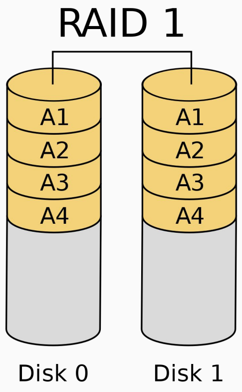
No fault-tolerance

Capacity scales linearly

Figures from https://en.wikipedia.org/wiki/Standard_RAID_levels

Commonly used RAID levels





Striping only

No fault-tolerance

Capacity scales linearly

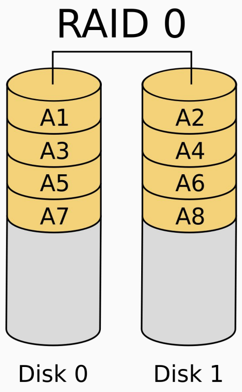
Full redundancy

(n-1) fault-tolerance

Capacity is constant

Figures from https://en.wikipedia.org/wiki/Standard_RAID_levels

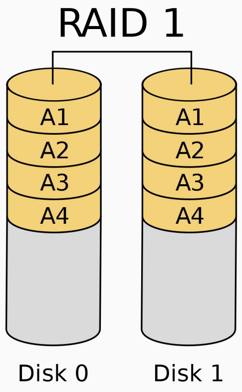
Commonly used RAID levels



Striping only

No fault-tolerance

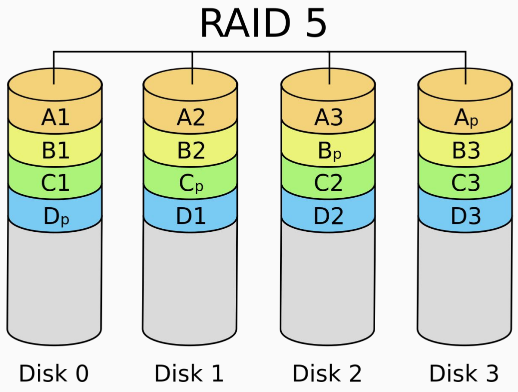
Capacity scales linearly



Full redundancy

(n-1) fault-tolerance

Capacity is constant



Striping with distributed **parity**

$A_p = A_1 \text{ XOR } A_2 \text{ XOR} \ldots \text{XOR } A_{n-1}$

Can tolerate some failure

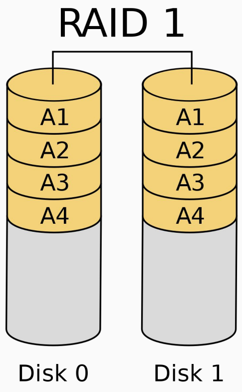
Capacity scales almost linearly

Requires n ≥ 3 disks

Figures from https://en.wikipedia.org/wiki/Standard_RAID_levels

Throughput with replication

- **Write** speed **decreases**: all data must be pushed to **all disks**
- **Read** speed can **increase**: blocks can be read in parallel from **any disk**
- *When is this trade-off worth it?*



Features of RAID

- **Striping blocks** over multiple drives increases **capacity and throughput**
  - **Not** reliability
- **Striping blocks** ⇒ a file can be larger than any single drive
- Adding **parity blocks** improves reliability
  - If a data block is corrupted or lost, it can be recovered from other blocks + parity
  - If a parity block is corrupted or lost, it can be recomputed from data blocks

Journaling

- Some file system operations are **non-atomic**
  - Creating a file: 1) allocate blocks and inodes, 2) create directory entry
- What if the system crashes mid-operation?
- **Solution**: use a **journal** to stage operations to be completed
- Crash recovery plays back the journal, does not need to check entire disk!
  - Rolls back partially completed operations
  - Clears completed operations

Quick recap so far

- **Combining multiple disks** increases capacity
- **Redundant storage** increases reliability and read-throughput
- **Parity blocks** provide error detection
- **Journaling** provides fast recovery from system failures

Why wasn’t RAID enough?

- RAID improves capacity, fault-tolerance, and (read) throughput **on a single machine**

- What about distributed computation?
    - **Communication** over the network is a **bottleneck**

- *What are the common access patterns?*
    - Can we do **better** than both **fully localized** and **fully distributed**?

Up next...

- Part 2: Hadoop distributed filesystem
- Part 3: HDFS + MapReduce