NYU | Center for Data Science

Week 04.3: HDFS and Map-Reduce

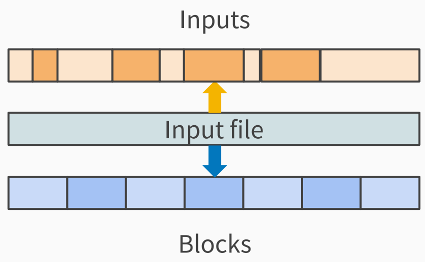
DS-GA 1004: Big Data

How does HDFS help Map-Reduce?

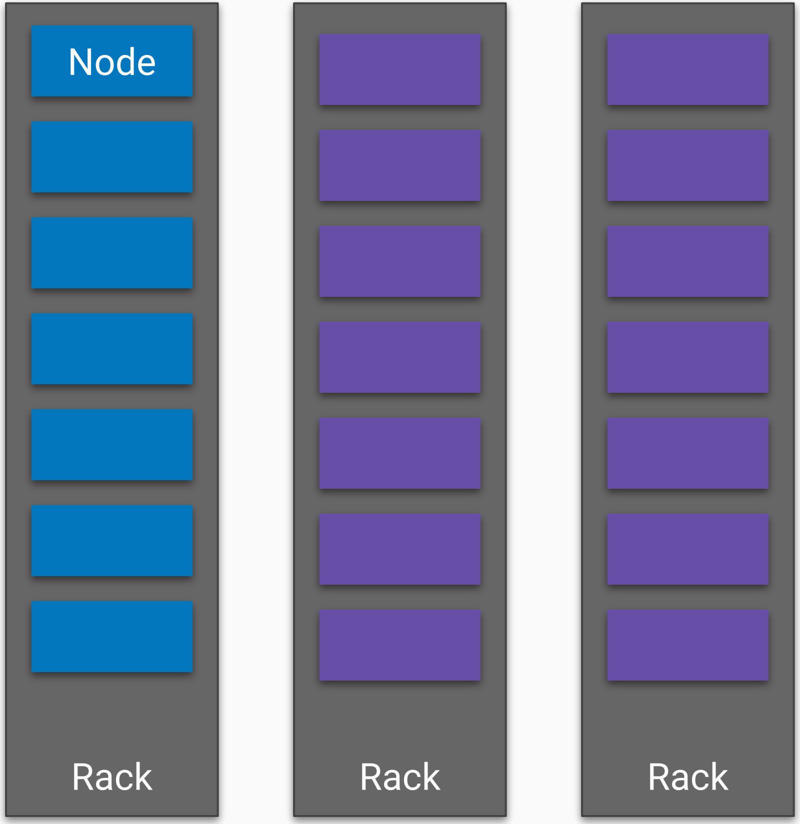
- HDFS shares **blocks** over data nodes
- Map-Reduce shares **jobs** over compute nodes
- Wouldn’t it be great if these were the same thing?
  - For big data, bringing **compute ⇒ data** is cheaper than the other way around!

Job scheduling and input splits

- A typical map-reduce job runs over one large file
  - Each file contains an array of (independent) inputs
- MapReduce divides the input into **splits**
- Each **split** maps onto one or more **blocks**
  - Try to assign work such that work for a **split** is done on a machine with its **blocks**
- HDFS exposes block layout to the application layer to make this possible

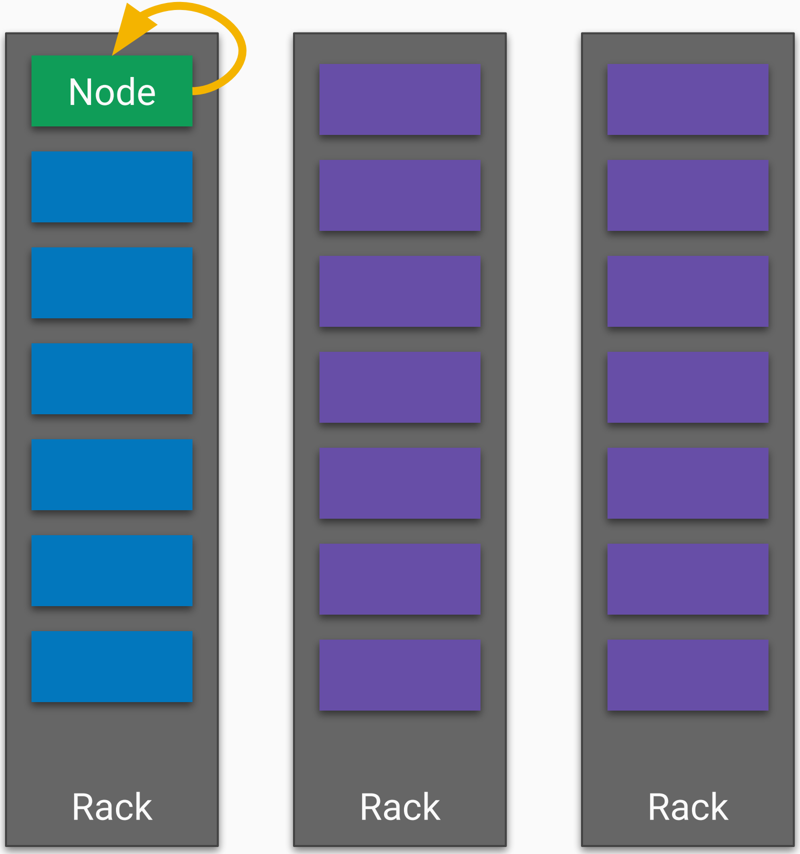


Where to execute?



https://commons.wikimedia.org/wiki/File:CERN_Server.jpg
Florian Hirzinger – www.fh-ap.com, CC BY-SA 3.0 via Wikimedia Commons

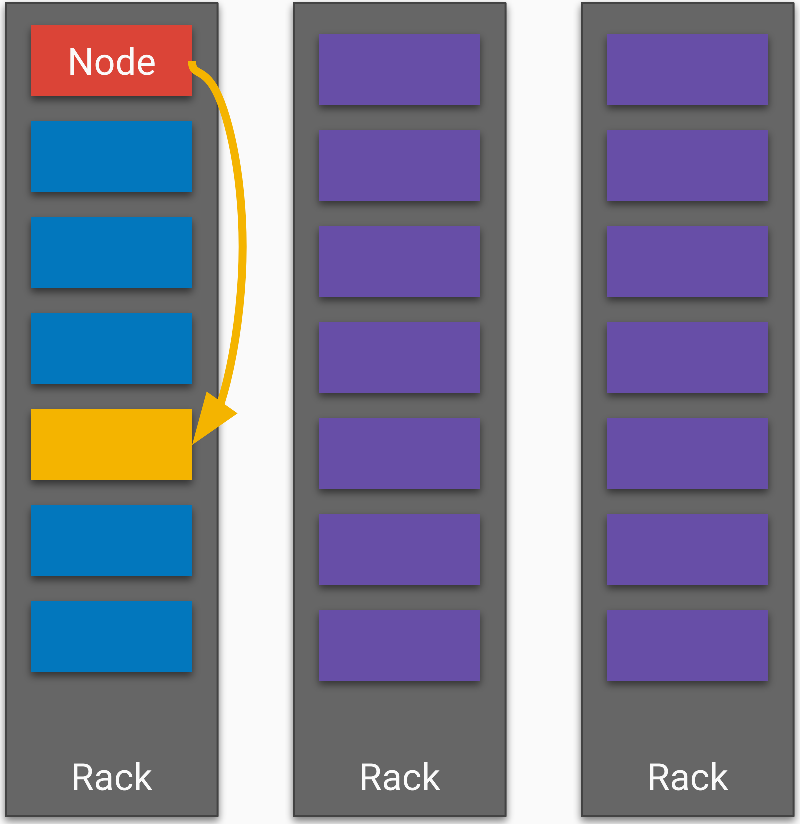
Where to execute?



Best case:

Execute on a node that stores the block(s) we need

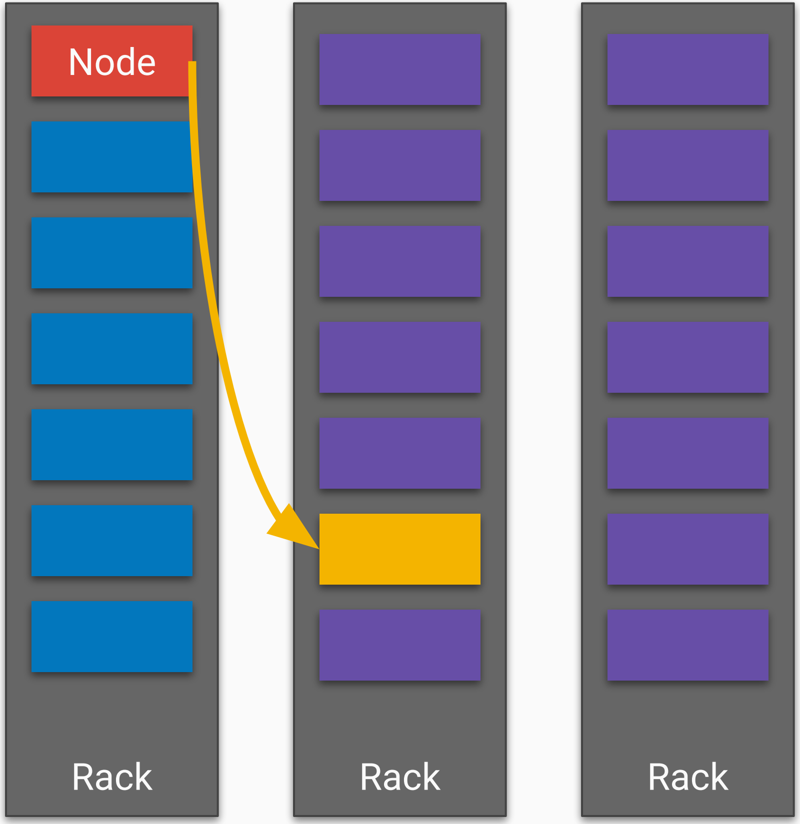
Where to execute?



Okay case:

Execute on a different node in the same rack
Within-rack communication is relatively fast

Where to execute?



Worst case:

Execute on a node in a different rack
Between-rack communication is slowest

https://commons.wikimedia.org/wiki/File:CERN_Server.jpg

Replication factors

- If we copy a block to multiple nodes, scheduling becomes easier
  - We're more likely to find a free worker that has our data
- HDFS lets you set the replication factor for each file
  - Replication isn't free: cost is multiplicative in the data size
- Typical setup: 3x replication
  - If possible, 2 nodes in one rack, +1 in a separate rack
  - This protects against both **node failure** and **rack failure**

CAP and HDFS

The CAP theorem for DFS

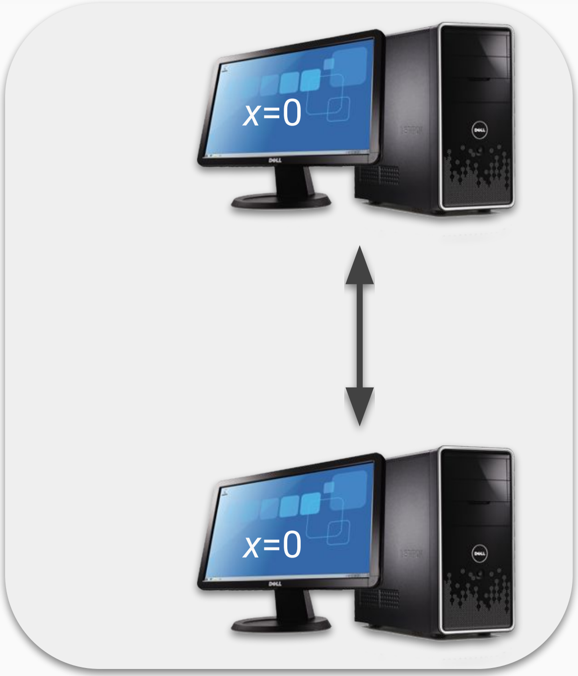
- **Consistency**:
  - Read always produces the most recent value
- **Availability**:
  - Requests cannot be ignored
- **Partition-tolerance**:
  - System maintains correctness during network failure

Pick two

(but networks always fail)

[Brewer 1998, Gilbert & Lynch 2002]

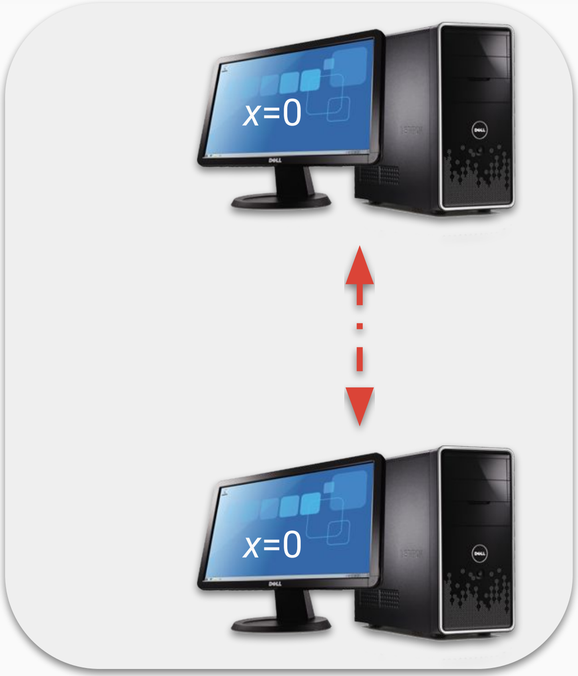
Why can't we have CAP? Assume that we could...



1. M_1 and M_2 initialized with x=0

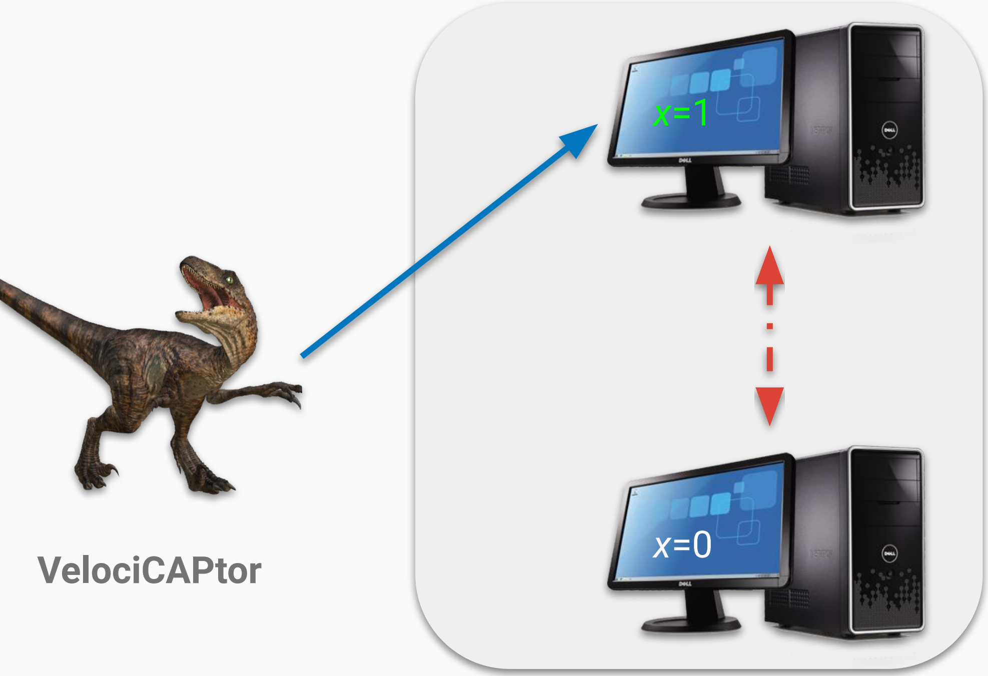
Why can’t we have CAP?
Assume that we could...

VelociCAPtor



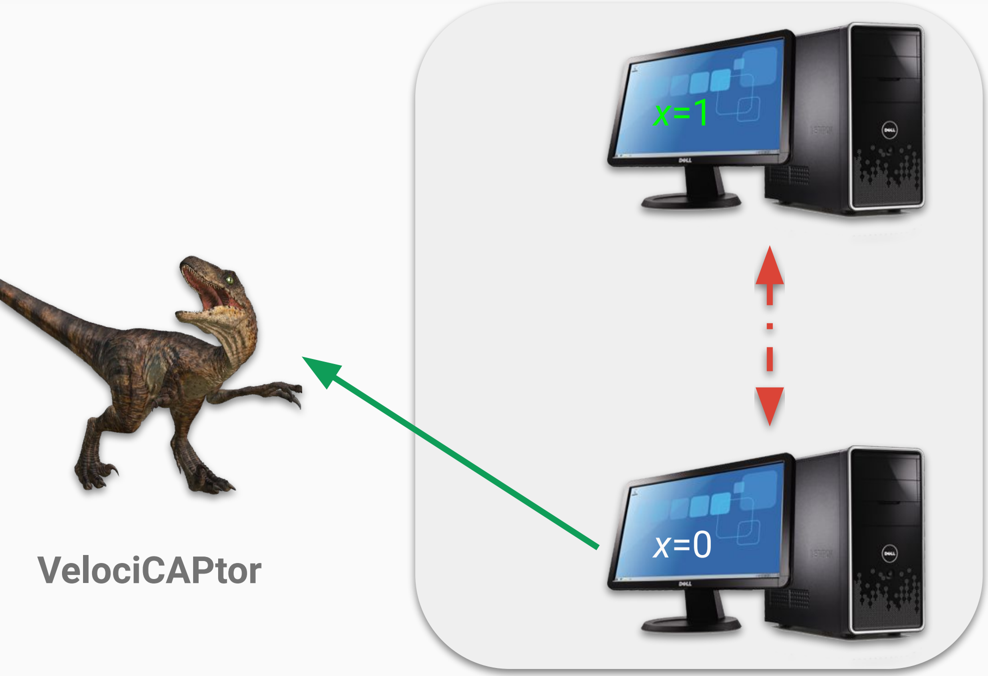
1. M_1 and M_2 initialized with x=0
2. Network fails

Why can’t we have CAP? Assume that we could…



1. M_1 and M_2 initialized with x=0
2. Network fails
3. Update x=1 on M_1

Why can't we have CAP?
Assume that we could...



VelociCAPtor

1. M_1 and M_2 initialized with x=0
2. Network fails
3. Update x=1 on M_1
4. Read x from M_2

What happens?

HDFS and CAP

- Consistency
  - Centralized name node always has a consistent view of the file system
  - Data can be added (appended), but **not modified**!
- Availability
  - If the name node goes offline, we're out of luck
- Partition-tolerance
  - Depends on network configuration and replication factor

Updates to HDFS since (Shvachko’10)

- Federation (2.x)
  - Multiple name nodes within the same HDFS
  - Each managing different portions of the file system
- High-availability mode (2.x)
  - Primary and stand-by name nodes, no single-point-of-failure
- Better HA mode (3.x)
  - >2 name nodes!

Wrap-up on HDFS

- Files divide into blocks, and are replicated across the cluster
- Checksums are replicated with each block
- Name node allocates blocks and directs clients
- Blocks are append-only ⇒ optimized for write-once, read-many patterns