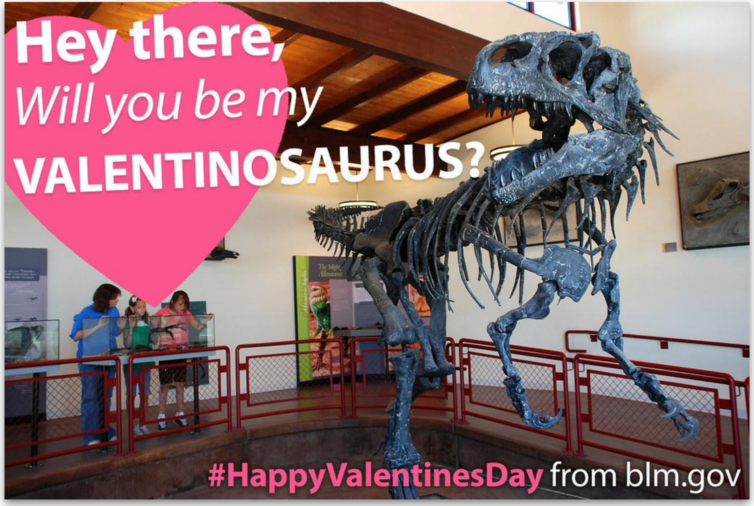
NYU Center for Data Science

Week 04.4: HDFS

DS-GA 1004: Big Data

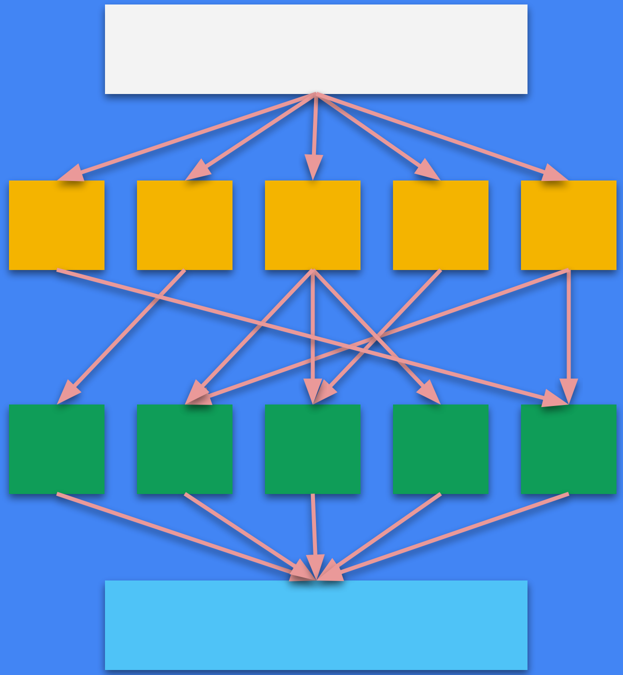
Announcements



https://www.flickr.com/photos/mypubliclands/31928717984

- Lab #1 (SQL/RDBMS)
  Due (02/18)
- Lab #2 (Map-Reduce)
  Starts (02/16)
- HPC accounts
- No class next week (02/21)

Previously...



1. Introduction to Map-Reduce
2. Criticisms of Map-Reduce

This week

1. Distributed storage

2. The Hadoop distributed file system (HDFS)

   $ hadoop fs -command ...

3. HDFS and Map-reduce

Hadoop distributed file system

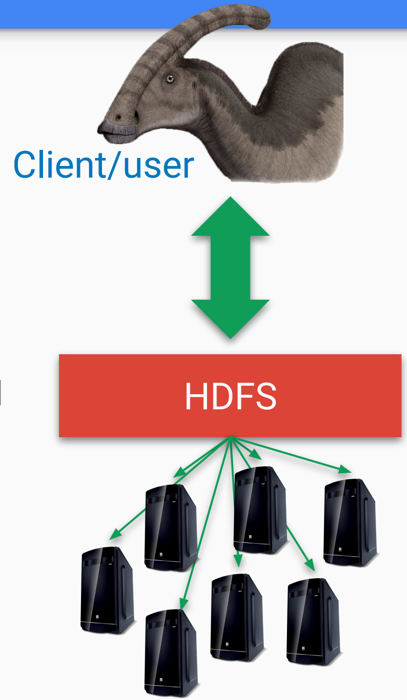
- HDFS is the storage component of Hadoop
    - **Useful beyond map-reduce!**
- Provides distributed, redundant storage
- Optimizes for **single-write**, **multiple-read** patterns

RAID vs NFS vs HDFS

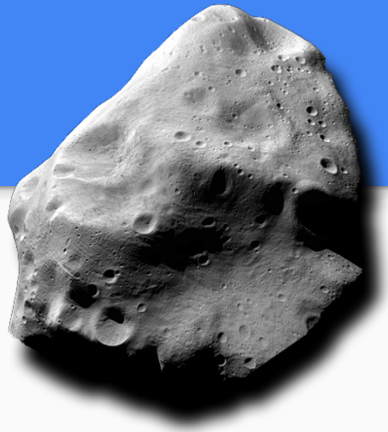
- RAID distributes storage over multiple disks in **one machine**
- NFS stores data on one machine, but provides access from **multiple machines**
- HDFS spreads each file across **multiple machines**
    - ~1 disk per machine with no internal redundancy
- If a disk fails, you need to take the machine offline anyway
    - Fault tolerance is at the level of **machines**, **not disks**
    - **Designing this way lets us tolerate other machine-level failures, not just disks!**

Using HDFS

- HDFS is a "file system", but we use it differently
- HDFS sits on top of the operating system's built-in FS
- Better to think of it as an **application** that stores files for you
  - Kind of like Google Drive or Apple iCloud
  - Data can be accessed through the "**hadoop fs**" command



Division of responsibilities



- Name nodes do not store **data**!
- Data nodes do not store **metadata** (e.g. file names)!
- Name node failure is **catastrophic**
- Data node failure can be tolerated, up to a point
    - Depends on how much replication you have

slido

Why do you think the HDFS designers parallelized storage at the level of blocks instead of files?

ⓘ Start presenting to display the poll results on this slide.

Several reasons...

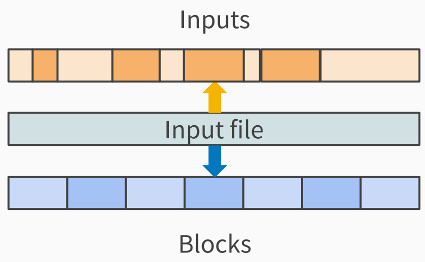
- A file can be larger than any single disk
- Map-Reduce programs typically only see small portions of a large file
- Uniform (maximum) block size makes allocation / replication easier

HDFS isn’t quite POSIX-compliant

- Updates are append-only
  - No changing old data!
  - This makes replication logic much simpler: if data is there, you can trust it
- Not all file modes are supported
  - Not all modes make sense in this limited context anyway
  - E.g., executable

Job scheduling and input splits

- A typical map-reduce job runs over one large file
  - Each file contains an array of (independent) inputs
  - E.g., lines in text files
- MapReduce divides the input into **splits**
  - Split = unit of work assigned to a mapper, contains **multiple records**
- 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



slido

What do you think will happen if a split is spread across multiple HDFS blocks?

ⓘ Start presenting to display the poll results on this slide.

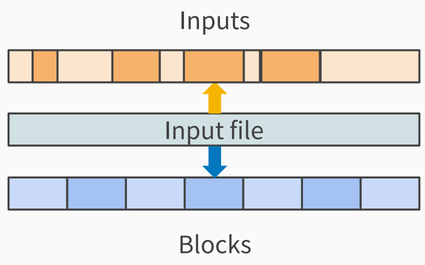
slido

[2] What do you think will happen if a split is spread across multiple HDFS blocks?

ⓘ Start presenting to display the poll results on this slide.

Splits and blocks

- **Split** = one logical division of input data for a map process
  - Each **split** typically consists of **multiple records**, e.g., rows of a CSV file
- Machine running map() must have access to the entire split, so **data may have to move!**
- By default, **split size** = **block size**, but some fragmentation is unavoidable



A typical Map-Reduce work-flow

1. Upload data from UNIX filesystem to HDFS
    a. hadoop fs -put my_file.csv

2. Run map-reduce program
    a. Each mapper sees a portion of my_file.csv
    b. Each mapper produces intermediate outputs as HDFS files
    c. Shuffle stage collects intermediate outputs to give to reducers
    d. Reducers operate on intermediates, produce final output as multiple blocks

3. Retrieve output from from HDFS
    a. hadoop fs -getmerge my_output_file.csv

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**

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]

Which CAP property does HDFS sacrifice?

ⓘ Start presenting to display the poll results on this slide.

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

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

Next week

- Spark and Spark-SQL
  - [Zaharia et al., 2010]
  - [Armbrust et al., 2015]
- Quiz #2 (2022-02-25): Map-Reduce and HDFS
- No class on 2022-02-21

slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.