NYU Center for Data Science

Week 04.2: HDFS

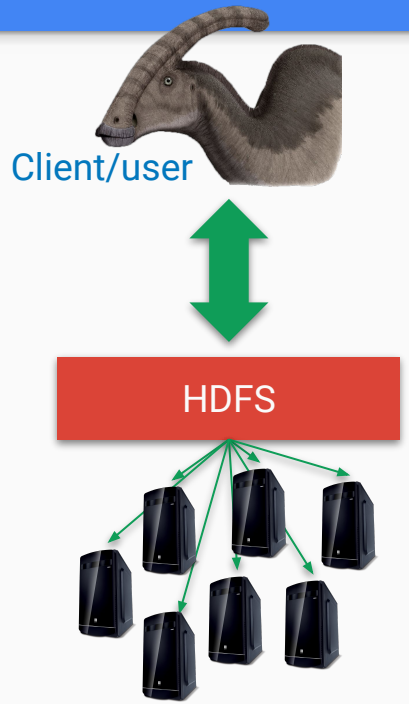
DS-GA 1004: Big Data

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
  - Typical of map-reduce applications

Using HDFS

- HDFS is a “file system”, but not like your OS file system
- 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



Two types of nodes in HDFS

Name node

Data nodes

The name node

- Clients talk to the name node to **locate data**
  - Analogous to the file system (but not storage device) in a standard OS
- Name node knows the mappings of:
  - Files → blocks
  - Blocks → data nodes
- Keeps a journal of transactions
  - Backed up remotely for durability

Data nodes

- Stores each block as two files in the local file system:
  - **Data block** (variable size up to defined max, typically 128MB)
  - Metadata:
    - **Checksum**
    - **Generation stamp**
- **Checksum**: used to detect storage errors
  - Analogous to **parity blocks** in RAID, but replicated on all nodes
  - Weaker than parity: can detect errors, but not correct them
- **Generation stamp**: used to detect updates

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

Example operation: writing to HDFS

Add plz

1. Client wants to add a block

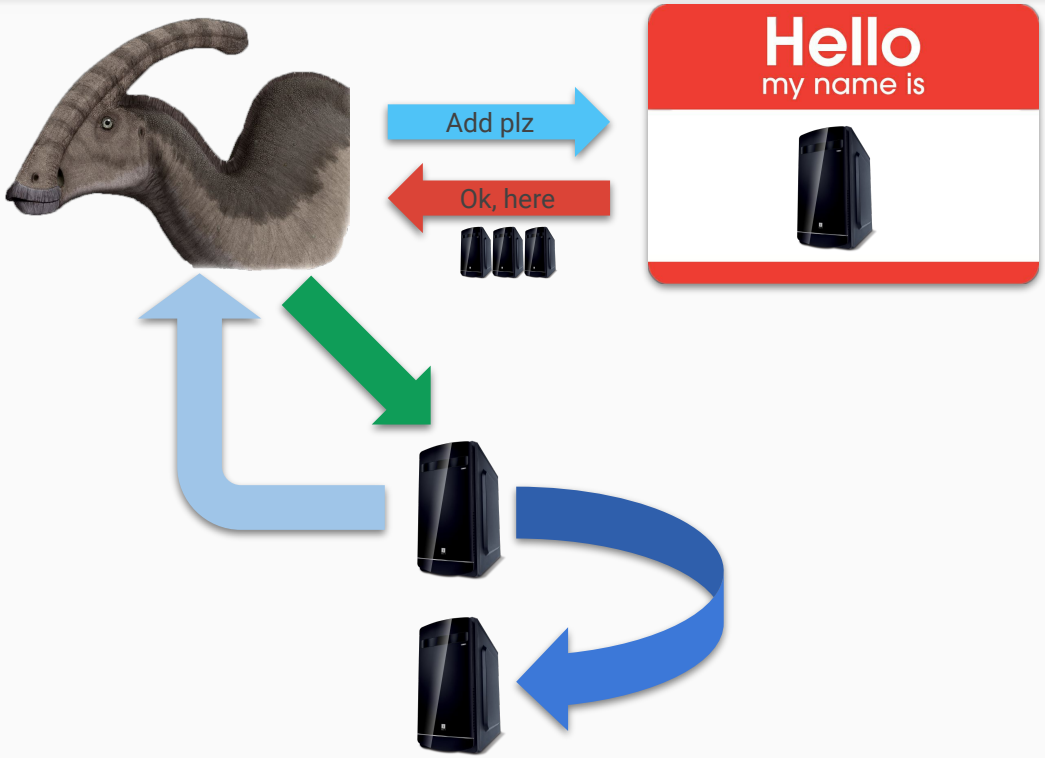
Example operation: writing to HDFS

1. Client wants to add a block
   a. Name node responds with a list of data nodes

Example operation: writing to HDFS

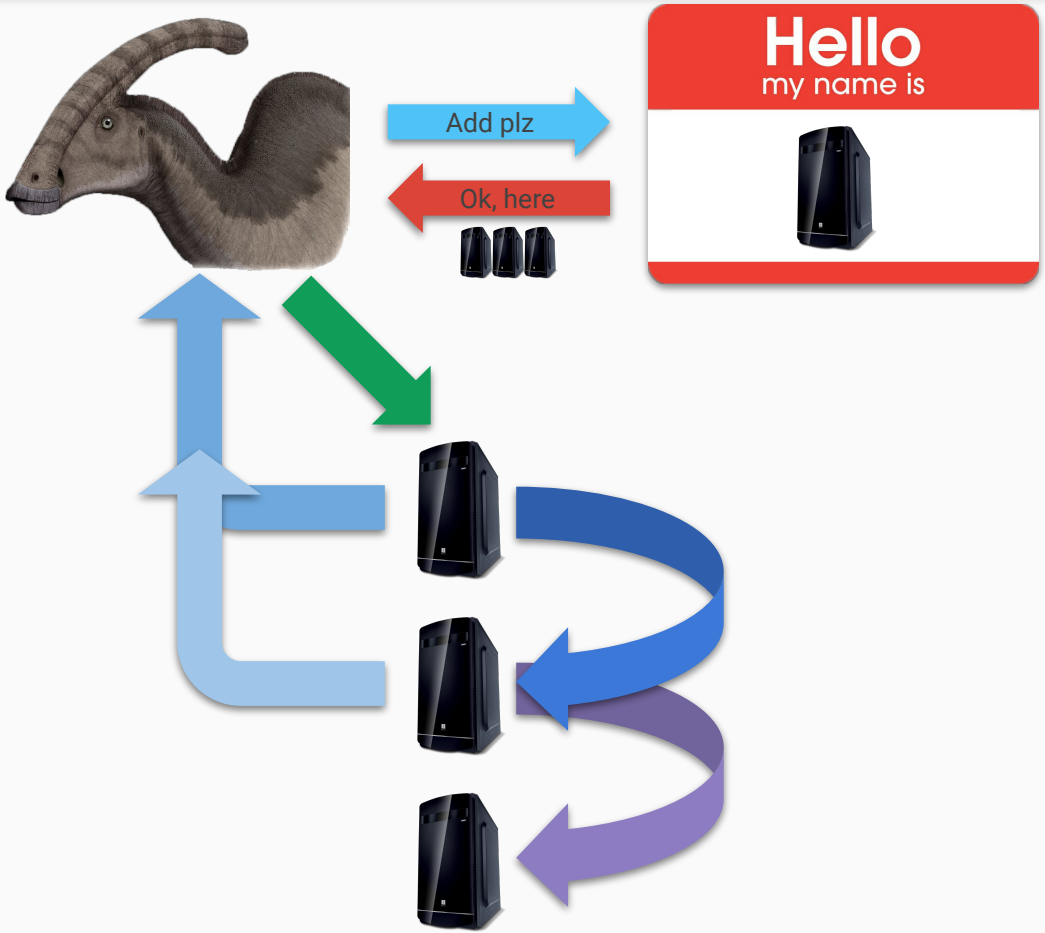
1. Client wants to add a block
   a. Name node responds with a list of data nodes
2. Client sends block to DN1

Example operation: writing to HDFS



1. Client wants to add a block
   a. Name node responds with a list of data nodes
2. Client sends block to DN1
3. DN1 stores, acknowledges, and sends block to DN2

Example operation: writing to HDFS



1. Client wants to add a block
   a. Name node responds with a list of data nodes
2. Client sends block to DN1
3. DN1 stores, acknowledges, and sends block to DN2
4. DN2 stores, acknowledges, and sends block to DN3

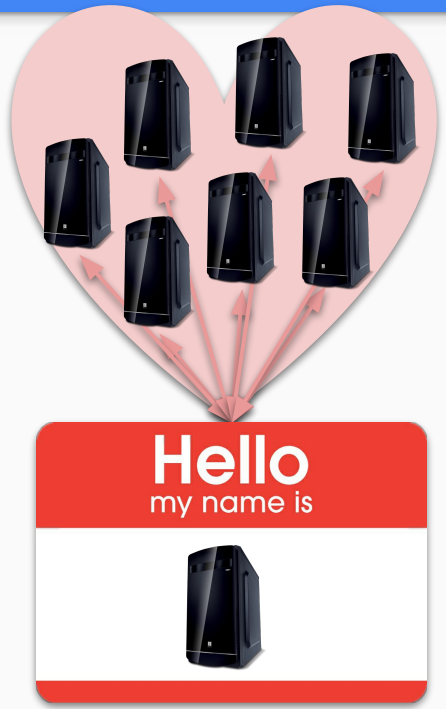
Example operation: writing to HDFS

1. Client wants to add a block
   a. Name node responds with a list of data nodes
2. Client sends block to DN1
3. DN1 stores, acknowledges, and sends block to DN2
4. DN2 stores, acknowledges, and sends block to DN3
5. DN3 stores and acknowledges

Add complete, close file, **alert name node**

Name and data node communication

- Data nodes periodically signal the name node
  - “Heartbeat”
- The name node always knows which data nodes are alive
  - At least, within the last 3 seconds
  - Name node can infer failures and insufficient replication
- Name node may respond with update messages
  - E.g.: replicate block *x* from data node *y*



Recovering from failure: checkpoints

- Checkpoints are snapshots of the current name node's state
  - Directory structure, block maps, and journal
  - Name node keeps all of this information in RAM
- These are created periodically to ensure fast recovery when NN fails
- Checkpoints cannot be updated, only replaced

HDFS isn’t quite POSIX-compliant

- Updates are append-only
  - No changing old data!
  - This makes replication logic much simpler
  - **What other benefits might immutability offer?**
- Not all file modes are supported
  - Not all modes make sense in this limited context anyway
  - E.g., executable

Why doesn’t HDFS work like my desktop?

- Desktop computing needs to support all kinds of uses
  - E.g. thousands of small configuration files
  - Files that update frequently (e.g., browser cache)
- Large data analysis jobs have different needs
  - Few, large files
  - Frequent read access (analysis)
  - Infrequent updates (append-only can be okay)

Up next

- Part 3: HDFS + MapReduce