CSx55: Distributed Systems [HDFS]

Circumventing The Perils of Doing Too Much

A namenode needs gumption
With guardrails to avoid failure
What's not an option?
Playing it by ear

With data volumes on an upward trajectory
avoid the bottleneck strain
A way out? This ain't much of a mystery
separate data from the control plane

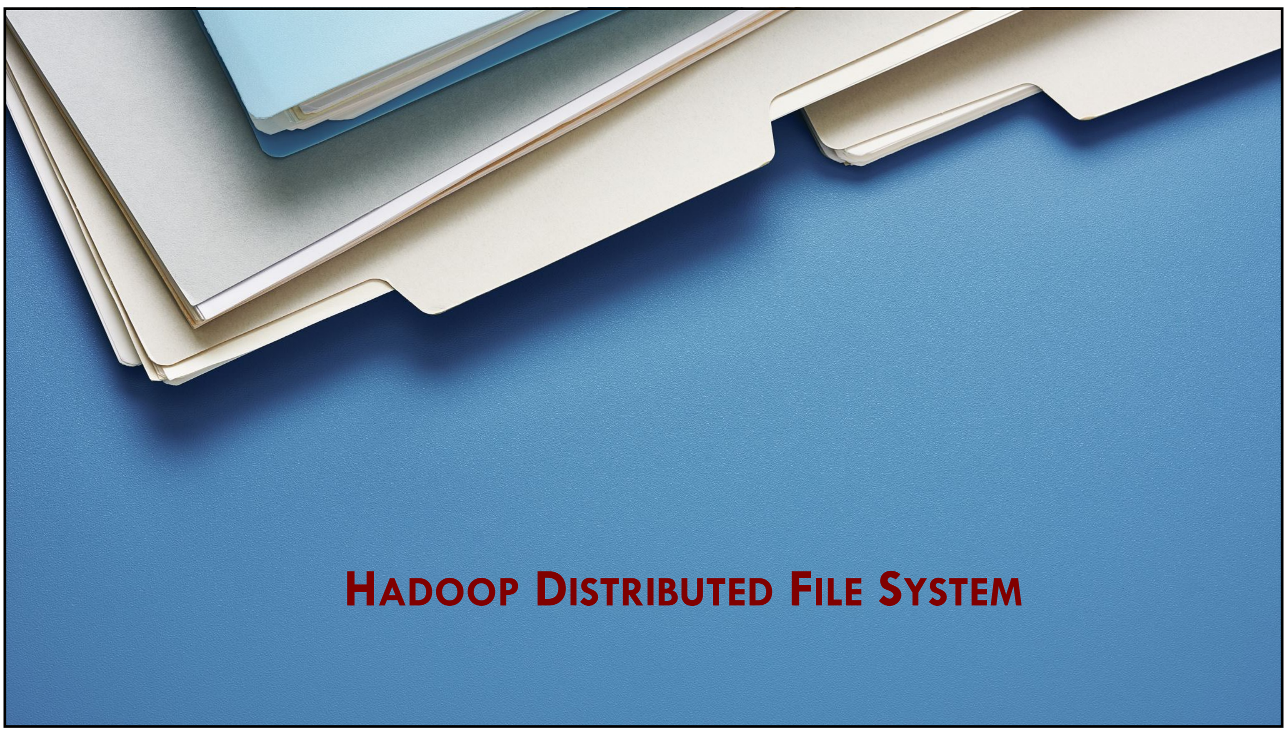
Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Can the number of reducers be chosen based on the number of available cores to maximize concurrency?
- Why are two consecutive blocks of a file not placed on the same machine?
- Can a combiner run before all mappers have finished?
- At a reducer, are keys sorted based on their values or based on their hash codes?

Topics covered in today's lecture

- HDFS



Block

- Filesystems for a single disk deal with data in blocks
  - Integral number of the HDD block size
- Block sizes
  - Filesystem blocks are a few KB
  - Disk blocks are normally 512 bytes

HDFS Blocks

- Have a much larger size: **256 MB** [default]
- Files are **broken** into block-sized *chunks*
  - Each block is stored as an independent unit
- If the last chunk is less than the HDFS block size?
  - No space is wasted because the blocks are themselves stored as files

Why is the block-size so big?

- **Time to transfer** data from disk can be made significantly larger than the time to *seek* first block
- If the seek time is 10 ms and transfer rate is 100 MB/sec?
  - To make seek time 1% of the transfer time, block size should be 100 MB
- Must be careful not to overdo block size increase
  - Since tasks operate on blocks, the number of tasks could reduce

Benefits of the block abstraction in distributed file systems

- File can be **larger than any single disk** in the cluster
- Simplifies the storage subsystem
  - File metadata (including permissions) handled by another subsystem and not stored with the block

Blocks and replication

- Each block is replicated on a small number of **physically separate** machines
- If a block becomes unavailable?
  1. Copy *read from another location* transparently
  2. That block is also *replicated from its alternative locations* to other live machines
     - Bring replication factor back to the desired level

HDFS' fsck command

- List blocks that make up each file in the filesystem

```
% hadoop fsck / -files -blocks
```

Nodes in the HDFS

- Namenode {master}
- Datanode {worker}

Namenode

- Manages filesystem **namespace**
- Maintains filesystem tree and metadata
  - For all files and directories in the tree
- Information stored persistently on local disk in two files
  - **Namespace image** and the **edit log**

Tracking location of blocks comprising files

- Namenode knows about datanodes on which all blocks of a file are located
- The locations of the blocks are not stored persistently
  - Information **reconstructed** from datanodes during start up

Interacting with HDFS

- HDFS presents a **POSIX-like** file system interface
- Client code does not need to know about the namenode and datanode to function

Datanodes

- Store and retrieve blocks
  - Initiated by the client or the namenode
- **Periodically reports** back to the namenode with the *list of blocks* that they store

Failure of the namenode

- Decimates the filesystem
- **All files on the filesystem are lost**
  - No way of knowing how to reconstitute the files from the blocks

Guarding against namenode failures

- **Backup** files comprising the persistent state of the **filesystem metadata**
  - Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems
    - Writes are synchronous and atomic
- Run a **secondary** namenode
  - Does not act as a namenode
  - Periodically merges namespace image with edit log

Secondary namenode

- Runs on a separate physical machine
  - Requires as much memory as the namenode to perform the merge operation
- Keeps a copy of the merged namespace image
  - Can be used if the namenode fails
- However, the secondary namenode **lags** the primary
  - Data loss is almost certain

HDFS Federation (introduced in 0.23)

- On large clusters with many files, memory is a limiting factor for scaling
- HDFS federation allows scaling with the addition of namenodes
  - Each manages a portion of the filesystem namespace
    - For e.g., one namenode for `/user` and another for `/share`

HDFS Federation [1/2]

- Each namenode manages a namespace volume
  - Metadata for the namespace and block pool
- Namespace volumes are **independent** of each other
  - No communications between namenodes
  - Failure of one namenode does not affect availability of another

HDFS Federation [2/2]

- Block pool storage is **not partitioned**
- Datanodes register with each namenode in the cluster
  - Store blocks from multiple blockpools

Recovering from a failed namenode [1/2]

- Admin starts a new primary namenode
  - With one of the filesystem metadata replicas
  - Configure datanodes and clients to use this namenode
- New namenode unable to serve requests until:
  1. Namespace image is **loaded** into memory
  2. **Replay** of edit log is complete
  3. Received enough **block reports** from datanodes to leave safe mode

Recovering from a failed namenode [2/2]

- Recovery can be really long
  - On large clusters with many files and blocks this can be about 30 minutes
- This also impacts routine maintenance

HDFS High Availability has features to cope with this

- Pair of namenodes in active standby configuration
- During failure of active namenode, standby takes over the servicing of client requests
  - In 10s of seconds

HDFS High-Availability: Additional items to get things to work

- Namenodes use a highly-available **shared storage** to store the *edit log*
- Datanodes must send block reports to **both** namenodes
  - Block mappings stored in memory not disk
- Clients must be configured to handle namenode failover

HDFS HA: Dealing with ungraceful failovers

- Slow network or a network partition can trigger failover transition
  - Previously active namenode thinks it is ***still*** the active namenode
- The HDFS HA tries to avoid this situation using **fencing**
  - Previously active namenode should be prevented from causing corruptions

Fencing mechanisms: To shutdown previously active namenode

- Kill the namenode's process
- Revoking access to the shared storage directory
- Disabling namenode's network port
  - Using the remote management command
- STONITH
  - Use specialized power distribution unit to forcibly power down the host machine

Basic Filesystem Operations

- Type **`hadoop fs -help`** to get detailed help on commands
  - We are invoking Hadoop's filesystem shell command **`fs`** which supports other subcommands
- Start copying a file from the local filesystem to HDFS

```
% hadoop fs -copyFromLocal input/docs/quangle.txt
        /user/tom/quangle.txt
```

Basic Filesystem Operations

- Copy file back to the local filesystem

```
%hadoop fs –copyToLocal /user/tom/quangle.txt
  input/docs/quangle.copy.txt
```

- Verify if the movement of the files have changed the files in any way

```
% openssl md5 quangle.txt quangle.copy.txt
```


Basic Filesystem Operations

```
% hadoop fs -mkdir books
% hadoop fs -ls .
Found 2 items
drwxr-xr-x - tom supergroup 0 2019-04-02 22:41 /user/tom/books
-rw-r--r-- 1 tom supergroup 118 2019-04-02 22:29 /user/tom/quangle.txt
```

- Directories are treated as metadata and **stored by the namenode** not the datanodes

HADOOP FILE SYSTEMS

COMPUTER SCIENCE DEPARTMENT
COLORADO STATE UNIVERSITY

31

Hadoop filesystems

- Hadoop has an abstract notion of filesystem
- HDFS is just one implementation
  - Others include HAR, KFS (Cloud Store), S3 (native and block-based)
- Uses URI scheme to pick correct filesystem instance to communicate with

  `% hadoop fs -ls  file://` to communicate with local file system

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | HADOOP | L26.32

32

Interacting with the filesystem

- Hadoop has a `FileSystem` class
- HDFS implementation is accessible through the `DistributedFileSystem`
  - Write your code against the `FileSystem` class for maximum portability

Reading data from a Hadoop URL

```
InputStream in = null;
try {
    in = new URL("hdfs://host/path").openStream();
    // process in
} finally {
    IOUtils.closeStream(in);
}
```

Make Java recognize Hadoop's URL scheme

- Call `setURLStreamHandlerFactory()` on URL with an instance of `FsURLStreamHandlerFactory`
- Can only be called once per JVM, so it is typically executed in a static block

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT HADOOP L26.35

35

Displaying files from a Hadoop filesystem

```
public class URLCat {
  static {
       URL.setURLStreamHandlerFactory(
                      new FsUrlStreamHandlerFactory());
  }

 public static void main(String[] args) throws Exception {
    InputStream in = null;
    try {
       in = new URL(args[0]).openStream();
       IOUtils.copyBytes(in, System.out, 4096, false);
    } finally {
      IOUtils.closeStream(in);
    }
 }
}
```

Buffer size used for copying

Close streams after copying is complete?

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT HADOOP L26.36

36

A sample run of the `URLCat`

```
% hadoop URLCat hdfs://localhost/user/tom/quangle.txt

On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT HADOOP L26.37

37

The contents of this slide set are based on the following references

- *Tom White. Hadoop: The Definitive Guide. 3rd Edition. Early Access Release. O'Reilly Press. ISBN: 978-1-449-31152-0.* Chapters [2 and 3].

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT HADOOP L26.38

38