The last line of the program ensures that a line is written for the last key group in the input. We can now simulate the whole MapReduce pipeline with a Unix pipeline (which is equivalent to the Unix pipeline shown in Figure.

```
% cat input/ncdc/sample.txt | ch02/src/main/ruby/max_temperature_map.rb | \
  sort | ch02/src/main/ruby/max_temperature_reduce.rb
1949    111
1950    22
```

The output is the same as the Java program, so the next step is to run it using Hadoop itself. The hadoop command doesn't support a Streaming option; instead, you specify the Streaming JAR file along with the jar option. Options to the Streaming program specify the input and output paths, and the map and reduce scripts. This is what it looks like:

```
% hadoop jar $HADOOP_INSTALL/contrib/streaming/hadoop-*-streaming.jar \
  -input input/ncdc/sample.txt \
  -output output \
  -mapper ch02/src/main/ruby/max_temperature_map.rb \
  -reducer ch02/src/main/ruby/max_temperature_reduce.rb
```

When running on a large dataset on a cluster, we should set the combiner, using the -combiner option. From release 0.21.0, the combiner can be any Streaming command. For earlier releases, the combiner had to be written in Java, so as a workaround it was common to do manual combining in the mapper, without having to resort to Java. In this case, we could change the mapper to be a pipeline:

```
% hadoop jar $HADOOP_INSTALL/contrib/streaming/hadoop-*-streaming.jar \
  -input input/ncdc/all \
  -output output \
  -mapper "ch02/src/main/ruby/max_temperature_map.rb | sort |
    ch02/src/main/ruby/max_temperature_reduce.rb" \
  -reducer ch02/src/main/ruby/max_temperature_reduce.rb \


   -file ch02/src/main/ruby/max_temperature_map.rb \
   -file ch02/src/main/ruby/max_temperature_reduce.rb
```

Note: also the use of -file, which we use when running Streaming programs on the cluster to ship the scripts to the cluster.

Python

Streaming supports any programming language that can read from standard input and write to standard output, so for readers more familiar with Python, here's the same example again. The map script is in Example 1, and the reduce script is in Example 2.

Example 1. Map function for maximum temperature in Python

```
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
  val = line.strip()
  (year, temp, q) = (val[15:19], val[87:92], val[92:93])
  if (temp != "+9999" and re.match("[01459]", q)):
    print "%s\t%s" % (year, temp)
```

Example 2-11. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python

import sys

(last_key, max_val) = (None, 0)
for line in sys.stdin:
  (key, val) = line.strip().split("\t")
  if last_key and last_key != key:
    print "%s\t%s" % (last_key, max_val)
    (last_key, max_val) = (key, int(val))
  else:
    (last_key, max_val) = (key, max(max_val, int(val)))

if last_key:
  print "%s\t%s" % (last_key, max_val)
```

We can test the programs and run the job in the same way we did in Ruby. example, to run a test:

```
% cat input/ncdc/sample.txt | ch02/src/main/python/max_temperature_map.py |
  sort | ch02/src/main/python/max_temperature_reduce.py
1949    111
1950    22
```

3.5 THE HADOOP DISTRIBUTED FILE SYSTEM

File systems that manage the storage across a network of machines are called distributed file systems. Since they are network-based, all the complications of network programming kick in, thus making distributed file systems more complex than regular disk file systems. Hadoop comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem. It is a filesystem designed for storing very large files (means files that are hundreds of megabytes, gigabytes, or terabytes in size) with streaming data access patterns, running on clusters of commodity hardware.

Features of HDFS

The Features of HDFS are as follows :

1. **Highly Scalable -** HDFS is highly scalable as it can scale hundreds of nodes in a single cluster.
2. **Replication -** Due to some unfavorable conditions, the node containing the data may be loss. So, to overcome such problems, HDFS always maintains the copy of data on a different machine.
3. **Fault tolerance -** In HDFS, the fault tolerance signifies the robustness of the system in the event of failure. The HDFS is highly fault-tolerant that if any machine fails, the other machine containing the copy of that data automatically become active.
4. **Distributed data storage -** This is one of the most important features of HDFS that makes Hadoop very powerful. Here, data is divided into multiple blocks and stored into nodes.
5. **Portable -** HDFS is designed in such a way that it can easily portable from platform to another.

Goals of HDFS

The goals of HDFS are as follows :

1. **Handling the hardware failure** - The HDFS contains multiple server machines. Anyhow, if any machine fails, the HDFS goal is to recover it quickly.
2. **Streaming data access -** The HDFS applications usually run on the general-purpose file system. This application requires streaming access to their data sets.
3. **Coherence Model -** The application that runs on HDFS require to follow the write-once-ready-many approach. So, a file once created need not to be changed. However, it can be appended and truncate.

Where to use HDFS

1. **Very Large Files:** Files should be of hundreds of megabytes, gigabytes or more.
2. **Streaming Data Access:** The time to read whole data set is more important than latency in reading the first. HDFS is built on write-once and read-many-times pattern.
3. **Commodity Hardware:**It works on low cost hardware.

Where not to use HDFS

1 **Low Latency data access**: Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.

2 **Lots Of Small Files:**The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.

3 **Multiple Writes:**It should not be used when we have to write multiple times.

3.5.1 The Design of HDFS

HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. Let's examine this statement in more detail:

1. Very Large Files

"Very large" in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.

2. Streaming Data Access

HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

3. Commodity hardware

Hadoop doesn't require expensive, highly reliable hardware to run on. It's designed to run on clusters of commodity hardware (commonly available hardware available from multiple vendors3) for which the chance of node failure across the cluster is high, at least

for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. It is also worth examining the applications for which using HDFS does not work so well. While this may change in the future, these are areas where HDFS is not a good fit today:

A. Low-latency data access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. HBase is currently a better choice for low-latency access.

B. Lots of small files

Since the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory. While storing millions of files is feasible, billions is beyond the capability of current hardware

C. Multiple writers, arbitrary file modifications

Files in HDFS may be written to by a single writer. Writes are always made at the end of the file. There is no support for multiple writers, or for modifications at arbitrary offsets in the file. (These might be supported in the future, but they are likely to be relatively inefficient.)

3.5.2 HDFS Concepts

The main components/ Concepts of HDFS are

1. **Blocks:** A Block is the minimum amount of data that it can read or write.HDFS blocks are 128 MB by default and this is configurable. Files n HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a file system, if the file is in HDFS is smaller than block size, then it does not occupy full block?s size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only. The HDFS block size is large just to minimize the cost of seek.
2. **Name Node:** HDFS works in master-worker pattern where the name node acts as master. Name Node is controller and manager of HDFS as it knows the status and the

metadata of all the files in HDFS; the metadata information being file permission, names and location of each block. The metadata are small, so it is stored in the memory of name node, allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently, so all this information is handled bya single machine. The file system operations like opening, closing, renaming etc. are executed by it.

3. **Data Node:** They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

3.5.2.1 Blocks

A disk has a block size, which is the minimum amount of data that it can read or write. File systems for a single disk build on data in blocks, which are an integral multiple of the disk block size. HDFS, too, has the concept of a block, but it is a much larger unit– 64 MB by default. Like in a file system for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units.

Benefits of Blocks

The benefits of Blocks are as follows :

1. A file can be larger than any single disk in the network. There's nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster.
2. Making the unit of abstraction a block rather than a file simplifies the storage subsystem. The storage subsystem deals with blocks, simplifying storage management (since blocks are a fixed size, it is easy to calculate how many can be stored on a given disk) and eliminating metadata concerns.
3. Blocks fit well with replication for providing fault tolerance and availability. To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (typically three).

Why Is a Block in HDFS So Large?

HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost of seeks. By making a block large enough, the time to transfer the data from the disk can be made to be significantly larger than the time to seek to the start of the block. The

the time to transfer a large file made of multiple blocks operates at the disk transfer rate. A quick calculation shows that if the seek time is around 10 ms, and the transfer rate is 100 MB/s, then to make the seek time 1% of the transfer time, we need to make the block size around 100 MB. The default is actually 64 MB, although many HDFS installations use 128 MB blocks. This figure will continue to be revised upward as transfer speeds grow with new generations of disk drives.

3.5.2.2 Name Node

An HDFS cluster has two types of node operating in a master-worker pattern: a namenode (the master) and a number of datanodes (workers). A client accesses the filesystem on behalf of the user by communicating with the namenode and datanodes. The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree.

This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.

3.5.2.3 Data Node

Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing. Without the namenode, the filesystem cannot be used. it is important to make the namenode resilient to failure, and Hadoop provides two mechanisms for this.

The first way is to back up the files that make up the persistent state of the filesystem metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems. It is also possible to run a secondary namenode, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large.

The secondary namenode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the namenode failing.

IMPORTANT QUESTIONS

1. Explain the a brief history of hadoop and apache hadoop and the hadoop ecosys
2. What is mapreduce? Explain about analyzing the data with hadoop.
3. Write briefly about map and reduce, java mapreduce.
4. Write briefly about scaling out and data flow.
5. Explain about combiner functions, running a distributed mapreduce job.
6. Discuss briefly hadoop streaming and the hadoop distributed file system.
7. Explain about the design of hdfs, hdfs concepts.
8. Write briefly blocks, name nodes and data nodes.