Introduction to HDFS

Ethans
Learn From experts

- HDFS is built on the principle of write-once, read-many-times pattern.
- A dataset generated or copied to the HDFS can be used to perform various analyses over time.
- Each time data analysis will involve a large proportion of dataset, sometimes only few proportions of it, so the time to read the whole dataset is more important than the latency in reading the first Record.

HDFS Features

Ethans
Learn From experts

Commodity hardware

- Hadoop doesn't require expensive, highly reliable hardware.
- It's designed to run on clusters of commodity hardware for which the chance of node failure across the cluster is high.
- HDFS is designed to carry on working without a noticeable interruption to the user in the face of failure

HDFS is not a good fit today:

- Low-latency data access
- Lots of small files

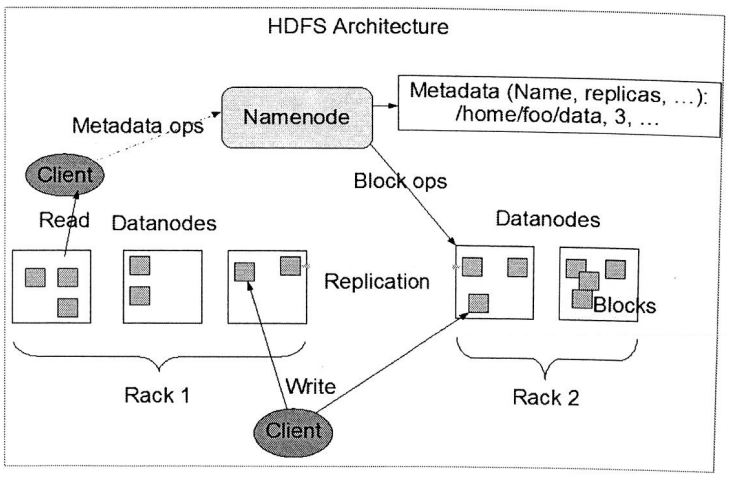
Why HDFS?

Ethans
Learn From experts

- Fault tolerance by detecting faults and applying quick, automatic recovery
- Data access via MapReduce streaming
- Simple and robust coherency model
- Processing logic close to the data, rather than the data close to the processing logic
- Portability across heterogeneous commodity hardware and operating systems
- Scalability to reliably store and process large amounts of data
- Economy by distributing data and processing across clusters of commodity personal computers
- Efficiency by distributing data and logic to process it in parallel on nodes where data is located
- Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures.

HDFS Architecture

Ethans
Learn From experts



Design of Name and Data Node

Ethans
Learn from experts

- HDFS has a master/slave architecture.
- HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients
- There are a number of DataNodes one per node which manage storage attached to the nodes that they run on
- A file is split into one or more blocks and these blocks are stored in a set of DataNodes

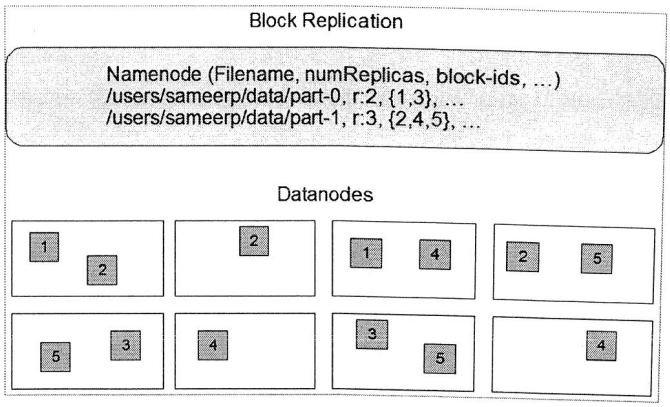
Replication Factor

Ethans
Learn from experts

- HDFS is designed to reliably store very large files across machines in a large cluster.
- It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
- The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.
- The NameNode makes all decisions regarding replication of blocks.
- It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster

Block Replication

Ethans
Learn From experts

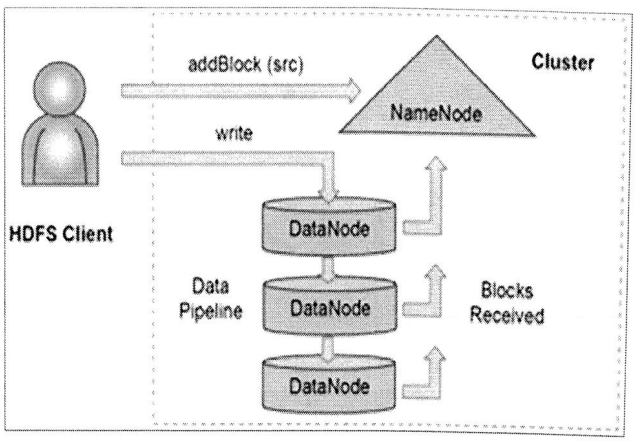


Slide 367

www.ethans.co.in

Data Flow

Ethans
Learn From experts



Slide 368

www.ethans.co.in

Block Size

Ethan's Learn from experts

- Disk has a block size, which is the minimum amount of data that it can read or write.
- HDFS block is much larger unit—128 / 256 MB by default
- Files in HDFS are broken into block-sized chunks, which are stored as independent units
- File in HDFS that is smaller than a single block does not occupy a full block's storage
- HDFS blocks are large to minimize the cost of seeks
- If the block is large enough, the time it takes to transfer the data from the disk can be longer than the time to seek to the start of the block
- Ex. if the seek time is around 10 ms and the transfer rate is
- 100 MB/s, to make the seek time 1% of the transfer time, we need to make the block size around 100 MB. The default is actually 128 MB

Block Size

Ethan's Learn from experts

- File can be larger than any single disk in the network
- Making the unit of abstraction a block rather than a file simplifies the storage subsystem eliminating metadata concerns
- Blocks fit well with replication for providing fault tolerance and availability.
- A block that is no longer available due to corruption or machine
- failure can be replicated from its alternative locations to other live machines to bring the replication factor back to the normal level
- HDFS's fsck command understands blocks
- % hdfs fsck / -files –blocks
- list the blocks that make up each file in the filesystem

Process Data

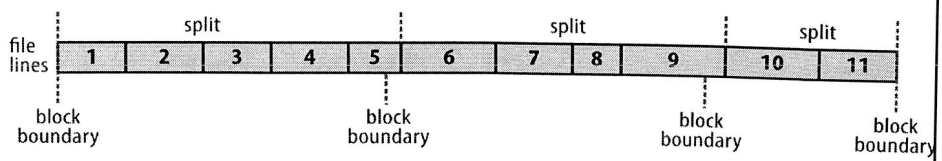
Ethans
Learn From experts

- HDFS breaks down very large files into large blocks and stores three copies of these blocks on different nodes in the cluster.
- Hadoop uses a logical representation of the data stored in file blocks, known as input splits
- The number of input splits that are calculated for a application determines the number of mapper tasks. Each of these mapper tasks is assigned to a slave node where the input split is stored.

Relation between HDFS and split

Ethans
Learn From experts

- Blocks are physical chunks of data store in disks where as InputSplit is not. It is a Java class with pointers to start and end locations in blocks
- A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information
- Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split



HDFS COMMANDS DEMO

Ethans
Learn from experts

HDFS COMMANDS DEMO

MapReduce in Python

Ethans
Learn from experts

- There are many ways to create a map reducer program in python, some of the frequent used module and ways are mentioned below:
- Each of the process have it own advantage and drawback, in this course we mainly focus on Streaming and Mrjob.

- Hadoop Streaming
- Mrjob
- dumbo
- hadoopy
- pydoop

Hadoop Streaming

Ethans
Learn From experts

- Hadoop Streaming is the way of supplying any executable to Hadoop as a mapper or a reducer, including standard Unix commands or Python scripts.
- The executable must read from stdin and write to stdout. One of the disadvantages of using Streaming directly is that while the inputs to the reducer are grouped by key, they are still iterated over line-by-line, and the boundaries between keys must be detected by the user.

Create Python Mapper

Ethans
Learn From experts

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

import sys

for line in sys.stdin:
    line = line.strip()
    keys = line.split()
    for key in keys:
        value = 1
        print( "%s\t%d" % (key, value) )
```

Create Python Reducer

Ethans
Learn from experts

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

import sys

last_key = None
running_total = 0
for input_line in sys.stdin:
    input_line = input_line.strip()
    this_key, value = input_line.split("\t", 1)
    value = int(value)

    if last_key == this_key:
        running_total += value
    else:
        if last_key:
            print( "%s\t%d" % (last_key, running_total) )
        running_total = value
        last_key = this_key

if last_key == this_key:
    print( "%s\t%d" % (last_key, running_total) )
```


Steps involved in streaming

Ethans
Learn from experts

Step 1: Create Mapper and Reducer code

Step 2: Test it on local file system

cat words.txt | ./mapper.py | sort | ./reducer.py > output.txt

Step 3: Copy the files to HDFS

hadoop dfs -mkdir wordcount

hadoop dfs -copyFromLocal ./words.txt wordcount/words.txt

hadoop dfs -ls wordcount/

Step 4: Test the hadoop streaming jar

Steps involved in streaming

Ethans
Learn From experts

```
hadoop jar
    /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2- cdh3u0.jar \
   -Dmapred.reduce.tasks=1 \
   -input wordcount/words.txt \
  -output wordcount/output \
  -mapper cat \
  -reducer "wc -l"
```

Step 5:

```
hadoop jar /usr/lib/hadoop-0.20/contrib/streaming/hadoop-streaming-0.20.2-
cdh3u0.jar \
 -Dmapred.reduce.tasks=1 \
 -mapper mapper.py \
 -reducer reducer.py \
 -input wordcount/words.txt \
 -output wordcount/output \
 -file mapper.py \
 -file reducer.py
```


MRJOB

Ethans
Learn From experts

mrjob lets you write MapReduce jobs in Python 2.6+/3.3+ and run them on several platforms.

You can:
- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)
- Run in the cloud using Google Cloud Dataproc (Dataproc)

To Install MRjob

```
sudo pip install mrjob
```

```
sudo apt-get install python-mrjob
```

MRJOB

Ethans
Learn From experts

mrjob is the easiest route to writing Python programs that run on Hadoop. If you use mrjob, you'll be able to test your code locally without installing Hadoop or run it on a cluster of your choice.

Here are a number of features of mrjob that make writing MapReduce jobs easier:

- Keep all MapReduce code for one job in a single class
- Easily upload and install code and data dependencies at runtime
- Switch input and output formats with a single line of code
- Automatically download and parse error logs for Python tracebacks
- Put command line filters before or after your Python code

Rating Program

Ethans
Learn From experts

```
from mrjob.job import MRJob

class MRRatingCounter(MRJob):
    def mapper(self, key, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer(self, rating, occurences):
        yield rating, sum(occurences)

if __name__ == '__main__':
    MRRatingCounter.run()
```

Objective – Module 14

Ethans
Learn from experts

Parallelism

- GIL
- Multithreading in Python
- Create Threads with parameters
- Create Daemon and Non Daemon processes
- Multiprocessing in Python

Multiple Thread Demo

Ethans
Learn from experts

```
import time

start = time.time()
def start_Single_thread(i):
    while i:
        i -= 1
    return

Total_iter = 50000000 # 50 million iteration
start_Single_thread(Total_iter)

end = time.time()
print "Single thread: "
print "Program took %.2f seconds to execute" %(end-start)
```

Multiple Thread Demo

Ethans
Learn from experts

```
import time, threading
start = time.time()
def start_Single_thread(i):
    while i:
        i -= 1
    return

Total_iter = 50000000 # 50 million iteration
t1 = threading.Thread(target=start_Single_thread,
                      args=(Total_iter/2,))
t2 = threading.Thread(target=start_Single_thread,
                      args=(Total_iter/2,))
#start the thread
t1.start(); t2.start()
# It will wait for main thread to wait
t1.join();t2.join()
end = time.time()
print "Multi thread: "
print "Program took %.2f seconds to execute" %(end-start)
```


Multiple Thread Demo

Ethans
Learn from experts

```
C:\Users\jatin\Desktop\Ethans\Python\
py
Single thread:
Program took 2.66 seconds to execute
```

```
C:\Users\jatin\Desktop\Ethans\Python\

Multi thread:
Program took 10.87 seconds to execute
```

Multiple Thread Demo

Ethans
Learn From experts

```
C:\Users\jatin\Desktop\Ethans\Python\
py
Single thread:
Program took 2.66 seconds to execute
```

```
C:\Users\jatin\Desktop\Ethans\Python\

Multi thread:
Program took 10.87 seconds to execute
```


Python GIL

Ethans
Learn From experts

Global Interpreter Lock

- Impossible to allow multiple native treads to execute code in a parallel.
- It imposes various restrictions on threads
- Namely, you can't utilize multiple CPUs
- Exception: C extension like numpy can release GIL

Python GIL

Ethans
Learn from experts

Applications written in programming languages with a GIL can be designed to use separate processes to achieve full parallelism, as each process has its own interpreter and in turn has its own GIL.

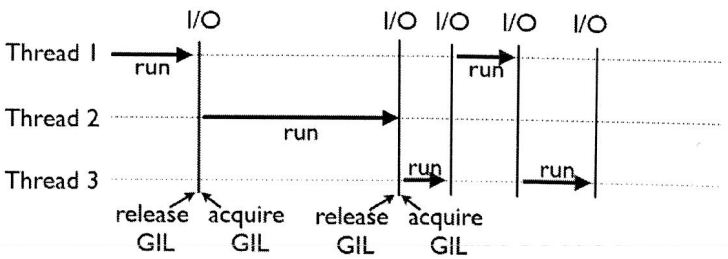
Benefits of the GIL

- Increased speed of single-threaded programs.
- integration of C libraries that usually are not thread-safe.

How Threads runs

Ethans
Learn from experts

- With the GIL, you get cooperative multitasking



- When a thread is running, it holds the GIL
- GIL released on I/O (read,write,send,recv,etc.)

Why/When to/not use threads?

Ethans
Learn from experts

- Not to use threads on CPU bound process
- Not to use when application make use of multiple cores simultaneously for performance reasons
- Not to use thread in distributed environment.

- Threads run in the same memory space (threads use the same memory, precautions have to be taken or two threads will write to the same memory at the same time. This is what the global interpreter lock is for)
- Great option for I/O-bound applications
- Great option of UI and non CPU bound processes.

Multithreading

Ethans
Learn from experts

- Module named threading is used to create threads.

They shared memory between processes

Each thread is running in parallel sequence.

There support GIL

Thread function

Ethans
Learn From experts

- Thread function in threading module used to create thread.
- Target is the parameter in Thread function.

```
import threading

def start_thread():
    """thread function"""
    print 'Function for thread'
    return

threads = []
for i in range(5):
    t = threading.Thread(target=start_thread)
    threads.append(t)
    t.start()
```


Thread function

Ethans
Learn From experts

- Args is the parameter in thread function to pass the argument

```
import threading

def start_thread(arg):
    """thread function"""
    print 'Function for thread ', arg
    return

threads = []
for i in range(5):
    t = threading.Thread(target=start_thread, args=(i,))
    threads.append(t)
    t.start()
```

Get the Thread name

Ethans
Learn from experts

- Threading.currentThread().getName()

```
import threading
import time

def func1():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(2)
    print threading.currentThread().getName(), 'Exiting'

def func2():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(3)
    print threading.currentThread().getName(), 'Exiting'

t1 = threading.Thread(target=func1, name='MyThread 1')
t2 = threading.Thread(target=func2, name='MyThread 2')
t3 = threading.Thread(target=func2) # use default name

t1.start()
t2.start()
t3.start()
```


Use Logging Module

Ethans
Learn from experts

```
import logging
import threading
import time

logging.basicConfig(level=logging.DEBUG,
                    format='[%(levelname)s] (%(threadName)-10s) %(message)s',
                    )

def func1():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

def func2():
    logging.debug('Starting')
    time.sleep(3)
    logging.debug('Exiting')
```

Create Daemon Thread

Ethan's Learn From experts

```
def daemon_process():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon_process', target=daemon_process)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)
```


Wait for Daemon to complete

Ethan's Learn From experts

```
def daemon_process():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon_process', target=daemon_process)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()
d.join()
```

Multiprocessing

Ethans
Learn from experts

- Module named multiprocessing is used to create processes.

No shared memory between processes

Each process run 1 thread each

There is a support for managed shared data.

Multiprocessing Example – Using Pool

Ethans
Learn from experts

```
from multiprocessing import Pool
from time import sleep

def start(n):
    sleep(1)
    funtion_result = n * n
    return funtion_result

if __name__ == '__main__':
    pool = Pool(processes=5)          # Using Map function
    results = pool.map(start, range(20), chunksize=10)
    print results
```

Multiprocessing Example – Using Process

Ethans
Learn From experts

```
from multiprocessing import Process
from os import getpid

def target_function():
    print getpid()

if __name__ == '__main__':
    p = Process(target=target_function, args=())
    p.start()
    p.join()
    p1 = Process(target=target_function, args=())
    p1.start()
    p1.join()
```


Multiprocessing Communication

Ethans
Learn From experts

Process Queues

Effective use of multiple processes usually requires some communication between them, so that work can be divided and results can be aggregated.

Process Pipe

The two connection objects returned by Pipe() represent the two ends of the pipe. Each connection object has send() and recv() methods (among others).

Note that data in a pipe may become corrupted if two processes (or threads) try to read from or write to the *same* end of the pipe at the same time

Multiprocessing Communication

Ethans Learn from experts

```
from multiprocessing import Process, Pipe
from os import getpid
from time import sleep

def target_function(proc, fmsg):
    count = 0
    while count < 10:
        msg = proc.recv()
        print "Process %r got message: %r" %(getpid(), msg)
        sleep(1)
        proc.send(fmsg)
        count += 1

if __name__ == '__main__':
    parent, child = Pipe()
    p = Process(target=target_function, args=(child, 'Message From child Proces
    p.start()
    parent.send('Message from Parent')
    target_function(parent, 'Message from Parent')
    p.join()
```


Multiprocessing Communication

Ethans Learn from experts

```
from multiprocessing import Process, Queue

def target_function(q):
    q.put(['First', 'Second'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=target_function, args=(q,))
    p.start()
    print q.get()
    p.join()
```

Synchronized processes

Ethans
Learn From experts

Lock on the process is use to ensure that only one process prints to standard output at a time:

```
from multiprocessing import Process, Lock

def func(l, i):
    l.acquire()
    print 'hello world', i
    l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        p = Process(target=func, args=(lock, num))
        p.start()
```


Synchronized processes

Ethans
Learn From experts

```
from multiprocessing import Process, Lock

def func(l, i):
    l.acquire()
    print 'hello world', i
    l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        p = Process(target=func, args=(lock, num))
        p.start()
        p.join()
```

Shared Memory

Ethans
Learn from experts

It is possible to create shared objects using shared memory which can be inherited by child processes.

Data can be stored in a shared memory map using Value or Array

Shared Memory

Ethans
Learn from experts

```
from multiprocessing import Process, Value, Array
import ctypes

def func(n, list1):
    n.value = 'M'
    for i in range(len(list1)):
        list1[i] = -list1[i]

if __name__ == '__main__':
    num = Value('c', 't') # 'c' indicates a character
    arr = Array('i', range(10))  # 'i' indicates integer.

    p = Process(target=func, args=(num, arr))
    p.start()
    p.join()

    print num.value
    print arr[:]
```

Shared Memory

Ethans Learn From experts

A manager object returned by Manager() controls a server process which holds Python objects and allows other processes to manipulate them using proxies.

A manager returned by Manager() will support types list, dict, Namespace, Lock Rlock etc

Shared Memory

Ethans Learn From experts

```
from multiprocessing import Process, Manager

def func(details, managers):
    details['Age'] = 35
    details['Name'] = 'Jatin'
    details['Dept'] = 'IT'
    managers.reverse()

if __name__ == '__main__':
    manager = Manager()

    details = manager.dict()
    managers = manager.list(['Aakash', 'rahul', 'Akshay'])

    p = Process(target=func, args=(details, managers))
    p.start()
    p.join()

    print details
    print managers
```