# Methods and tools for big data

Manuel – Summer 2022

Notes



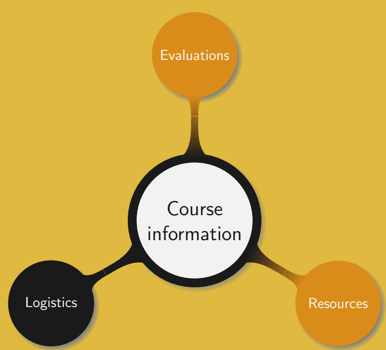


Notes

## 0. Course information

Notes






Notes

## Basic information

Teaching team:

- Instructor: Manuel (charlem@sjtu.edu.cn)
- Teaching assistants:
  - Yangyang (wangyangyang@sjtu.edu.cn)
  - TBD (TBD)

Important rules:

- When contacting a TA for an important matter, CC the instructor
- Prepend [VE472] to the subject, e.g. `Subject: [VE472] Grades`
- Use SJTU jBox service to share large files (> 2 MB)

**Never send large files by email**

Notes

## Course schedule

Course arrangements:

- Lectures:
  - Tuesday 16:00 – 17:40
  - Thursday 16:00 – 17:40
- Labs: Wednesday 18:20 – 20:40

Office hours:

- Anytime on Piazza
- On appointment

Notes

## Course objectives

Primary goals:

- Understand how big data sets are analysed in practice
  - Be able to use Hadoop
  - Learn how to work in the Hadoop ecosystem
- Be able to performed advanced data analysis on large data sets
  - Get good foundations on big data analysis
  - Be able to design, implement, and use advanced algorithm in Spark

*Be able to analyse any given dataset, regardless of there size*

Notes

## Course workflow

Learning strategy:

- Course side:
  1. Understand the new issues appearing as datasets grow
  2. Be able to setup a Hadoop cluster and use it
  3. Understand why traditional algorithms fail on big data
  4. Be able to implement advanced algorithms for big data
- Personal side:
  1. Derive algorithms for big data
  2. Use and work "inside" Hadoop, Drill, and Spark
  3. Relate known strategies to new problems
  4. Perform extra research

Notes

9

## Course outcomes

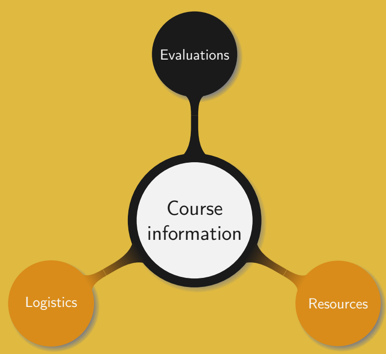
Detailed goals:

- Understand the basic logic behind Hadoop
- Have a general knowledge of the Hadoop ecosystem
- Be familiar with the basic Hadoop components: HDFS, YARN, and MapReduce
- Understand the structure of Drill and Spark
- Be able to work in Hadoop and "extend" its functionalities
- Know what tool to use for common specific purposes related to the study of big data
- Be familiar with common dimension reduction techniques
- Understand the limitations when facing "real" big data
- Be able to run basic data analysis on big data

Notes

10

## Chapter organisation




Notes

11

## Assignments

Homework:

- Total: 5 or 6
- Content: basic Hadoop, algorithms, Spark

Labs:

- Total: 12
- Content: guided sessions to setup and work with Hadoop, and Spark

Projects:

- Total: 1
- Content: analysis of some big dataset

Challenge:

- Total: 1
- Content: compare theory and practice in Hadoop and Spark implementations

Notes

12

## Grading policy

Grade weighting:

- Midterm exam: 15%
- Final exam: 15%
- Quizzes: 20%
- Projects: 30%
- Homework: 10%
- Labs: 10%

Assignment submissions: $-10\%$ per day, not accepted after 3 days

*Grades will be curved with the median in the range* $[\![B, B+]\!]$

Notes

13

## Honor Code

General rules:

- Not allowed:
  - Reuse the code or work from other students or groups
  - Reuse the code or work from the internet
  - Share too many details on how to complete a task
- Allowed:
  - Reuse part the course or textbooks and quoting the source
  - Share ideas and understandings on the course
  - Provide hints on where or how to find information

Notes

14

## Honor Code

Documents allowed during the exams:

- Midterm: none
- Final: a single A4 paper sheet with original handwritten notes

Group works:

- Every student in a group is responsible for his group's submission
- If a student breaks the Honor Code, the whole group is guilty

Notes

15

## Special circumstances

Contact us as early as possible when:

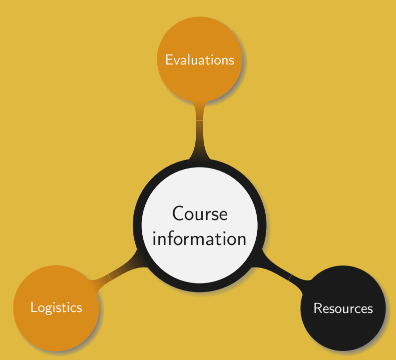
- Facing special circumstances, e.g. full time work, illness
- Feeling late in the course
- Feeling to work hard without any result

Any late request will be rejected

Notes

16

## Chapter organisation




Notes

## Canvas

Information and documents available on the Canvas platform:

- Course materials:
  - Syllabus
  - Lecture slides
  - Homework
  - Labs
  - Projects
- Course information:
  - Announcements
  - Notifications
  - Grades
  - Polls

Notes



## References

Useful places where to find information:

- *Hadoop the definitive guide*
- *Spark the definitive guide*
- *Machine learning, an algorithmic perspective*
- *Introduction to Data Mining*, by Tan et al..
- *Mining of Massive Datasets*, by Leskovec et al.. by White
- Search information online, i.e. $\{\textit{websites} \setminus \{\textit{non-English websites}\}\}$

Notes



## Key points

- Work regularly, do not wait the last minute/day
- Respect the Honor Code
- Go beyond what is taught
- Do not learn, understand
- Keep in touch with us
- Advice and suggestions are always much appreciated

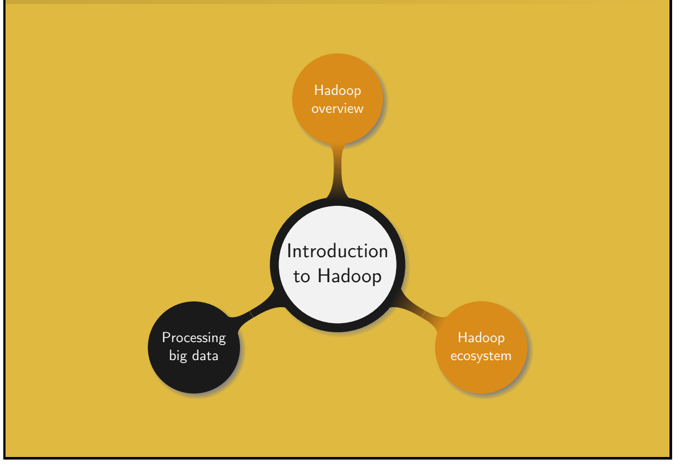
Notes

# 1. Introduction to Hadoop

Notes

## Chapter organisation

21




## Notes

## Data analysis

22

Generated data is often:

- Stored, e.g. in databases
- Preprocessed, e.g. cleaned
- Analysed, e.g. machine learning

Most common advanced analytics:

- Supervised learning: predict a label based on some features
- Recommendation: suggest product based on users' behaviour
- Unsupervised learning: discover structure in the data
- Graph analytics: searching for patterns

## Notes

## Von Neumann bottleneck

23

Problem for a regular computer:

- Fast CPU
- Large memory
- Limited throughput

Mitigating the problem:

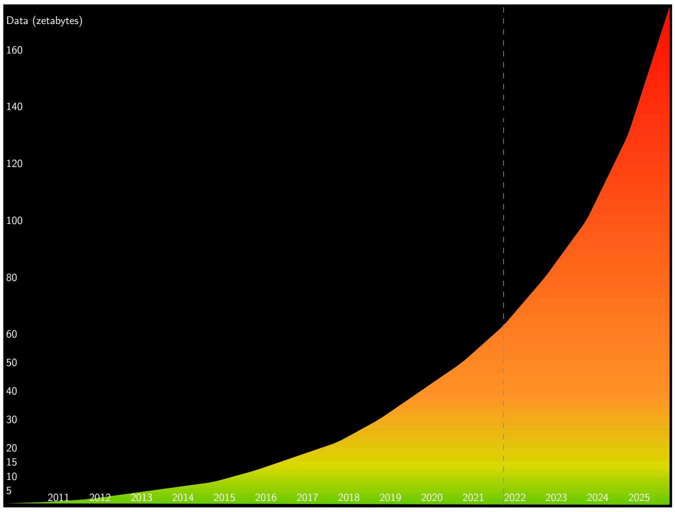
- Use caching
- Apply branch prediction
- Parallel read using RAID

**Example.** The speed of a disc read decreased relatively over time:

- 1990: 1.5 GB HDD at 4.4 MB/s
- Today: 1 TB HDD at 100 MB/s

*Scanning a whole disc in 1990 took 5 min, today it takes over 2.5 h!*

## Notes




## Notes

How to store and process data

as it grows very big?

Notes

26 Storing data

Relational Database Management Systems:

- Data size: gigabytes
- Access: interactive and batch
- Update: read|write small proportions of the data
- Structure: schema defined at writing time
- Efficiency: low-latency retrieval for small amount of data

Limitations of databases:

- Hard drive seek time increases slower than data transfer rate
- Data is often unstructured
- Slow to process as designed for read|write many times

Notes

27 Processing data

High-performance computing (HPC):

- Distributes computation across a cluster of machines
- Uses message passing interface
- Fits compute-bound jobs
- Data-flow controlled by programmer

Limitations of HPC:

- Handling of node or process failure
- Require very high network bandwidth
- Expensive infrastructures, complex to extend
- Low level APIs

Notes

28 Sequential vs. parallel algorithms

Random Access Machine (RAM) model:

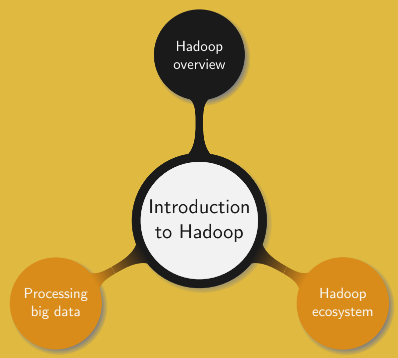
- A processor with a memory attached to it
- Each operation has a constant cost
- Runtime is proportional to the number of operations

Parallel Random Access Machine (PRAM) model:

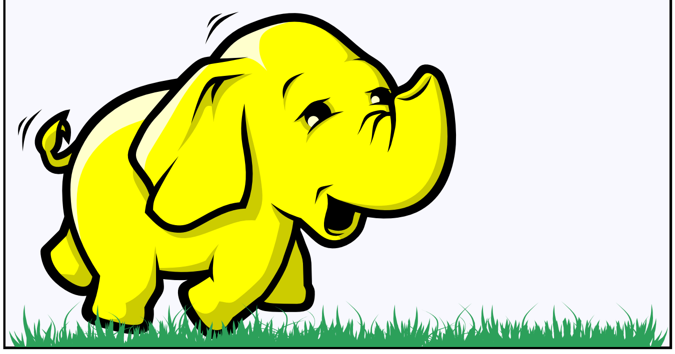
- Several processors with one or more memory modules attached
- Need to specify how how to deal with concurrent writes
- Each operation has a constant cost
- Runtime is defined when the slowest processor completes

*When dealing with "real" big data we need a distributed system*

Notes




Notes



Notes



The birth of Hadoop:

- 2002: Nutch, an open source web search engine
- 2003: paper describing Google File System (GFS)
- 2004:
  - NDFS: open source implementation of GFS for Nutch
  - Paper describing data processing on large clusters (MapReduce)
- 2005: open source implementation of MapReduce for Nutch
- 2006:
  - NDFS and MapReduce moved out of Nutch
  - Hadoop 0.1.0 released
  - Hadoop is run in production at Yahoo!

Notes



The adolescence of Hadoop:

- 2007 – 2008:
  - Number of companies using Hadoop jumps from 3 to over 20
  - Creation of Cloudera, first Hadoop distributor
- 2009:
  - MapR, new Hadoop distributor
  - HDFS and MapReduce become separate projects
- 2010 – 2011:
  - Many new "components" added to the Hadoop ecosystem
  - Receive two prizes at the Media Guardian Innovation Awards

Notes

## A short history



The maturity of Hadoop:

- 2012:
  - Hadoop 1.0 released
  - YARN ready to replace MapReduce (Hadoop 2.0)
- 2013 – 2014:
  - More than half of the Fortune 50 use Hadoop
  - Spark and Drill added to the Hadoop ecosystem
- 2017: Hadoop 3.0 released

Notes

## Hadoop's goals



Context where to adopt Hadoop:

- Massive amount of data to analysed
- Data stored over hundreds or thousands of computers
- Computation must be completed even if some nodes fail
- Cluster composed of commodity or high-end hardware

Hadoop's records:

- 2006: sort 1.8 TB of data in less than 48 h
- 2008: sort 1 TB of data in 209 s
- 2009: sort 1 TB of data in 62 s
- 2014: sort 100 TB of data in less than 23 min 30s

*The end goal is to efficiently analyse massive amount of data*

Notes

## Hadoop framework



Hadoop is composed of core modules:

- Hadoop common: base libraries and utilities used by other modules
- Hadoop Distributed File System (HDFS): distributed file system
- Hadoop MapReduce: implementation of the MapReduce model
- Apache Yet Another Resource Negotiator (YARN): manages the cluster resources and schedules the user's tasks

Languages:

- Mainly Java
- Some C
- Shell scripts for command line utilities

Notes

## HDFS overview



Characteristics of HDFS:

- Large files: at least hundreds of megabytes to terabytes
- Streaming data access: write once, read many times
- Commodity hardware: inexpensive common hardware

Limitations of HDFS:

- High throughput at the expense of latency
- The "Master node" keeps the filesystem metadata in memory
- Write always in append mode, by a single writer

Notes

## MapReduce overview

Programming paradigm composed of three main steps:

- Map:
  - A master node distributes the work and ensures exactly one copy of the redundant data is processed
  - Each worker node considers its local data and transforms it into key-value pairs
- Shuffle: each worker node redistributes its pairs based on the keys
- Reduce: each worker node combines a set of pairs into a smaller one

Notes



## MapReduce overview

MapReduce requirements:

- Mapping operations must be independent of each others
- Parallelism is limited by the number of sources and nearby CPUs
- Either all the output sharing the same key must be processed by a single reducer or the reduction must be associative

MapReduce benefits:

- Highly scalable on commodity hardware
- Possible to recover for partial failure
- Great efficiency due to parallelism

Notes



## YARN overview

A *container* is an environment with restricted resources where application-specific processes are run

YARN provides two types of daemons:

- Resource manager:
  - One per cluster
  - Manages the resources for the whole cluster
- Node manager:
  - One per cluster node
  - Launches and monitors containers

Notes



## MapReduce vs. YARN

In Hadoop 1, MapReduce:

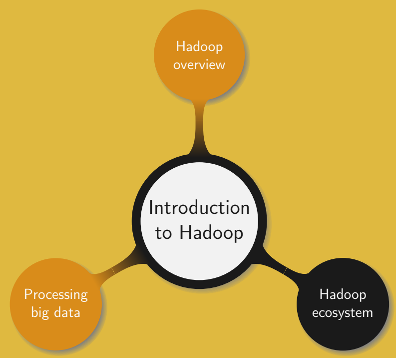
- Directly interacts with the filesystem
- Manages resources

In Hadoop 2, YARN:

- Manages the resources
- Interacts with the filesystem
- Hides low level details from the user
- Offers an intermediate layer supporting many other distributed programming paradigms

Notes

41 Chapter organisation




Notes

42 Mesos

Goal: global scalable resource manager, not restricted to Hadoop

Mesos scheduling:

- Determine the available resources
- Offer "various options" to an application scheduler
- Allow any number of scheduling algorithm to be developed, plugged, and used simultaneously
- Each framework decides what scheduling algorithm to use
- Mesos allocates resources across the schedulers, resolves conflicts, and ensures a fair share of the resources

Notes

43 Myriad

Goal: use Mesos to manage YARN resource requests

Simplified strategy:

1. A job requests resources to YARN
2. YARN uses the Myriad scheduler to allocate resources
3. Myriad scheduler matches requests to Mesos' resources offers
4. YARN allocates the resources

Benefits:

- Get the best from both worlds
- Give more flexibility to YARN

Notes

44 Spark

Goals:

- Be a full replacement for MapReduce
- Efficiently support multi-pass applications
- Write and read from the disk as little as possible
- As much as possible take advantage of the memory

Main ideas:

- Resilient Distributed Dataset (RDD): contains the data to be transformed or analysed
- Transformation: modifies an RDD into a new one
- Action: analyses an RDD

Notes

## Drill

Goals:

- Integrate into Hadoop as a MapReduce replacement
- Be an interactive ad-hoc analysis system for read-only data
- Be easily expandable using storage plugins
- Enjoy data agility

Main ideas:

- Columnar execution: shredded, in-memory columnar data representation
- Runtime compilation and code generation: compile and re-compile queries at runtime
- Optimistic execution: stream data in memory to minimise disk usage

Notes

## Spark vs. Drill

When to use Spark or Drill:

- Drill is an ANSI SQL:2003
- Spark has SQL query capabilities
- Drill allows fine grained security at the file level
- Drill is best used as a distributed SQL query engine
- Spark is best used to perform complex math, statistics, or machine learning

Notes

## Flink

Basics on Flink:

- Allows the execution of dataflow programs following a data-parallel and pipelined approach
- Provides a high-throughput and low-latency streaming engine
- Nicely handles node failures
- Can connect to various storage types

Notes

## Tez

Basics on Tez:

- Targets batch and interactive data processing applications
- Intends to improve MapReduce paradigm
- Exposes more simple framework and API to write YARN applications
- Expresses computation as a dataflow graph

Notes

## HBase

49

Basics on HBase:

- NoSQL database system for distributed filesystems
- Low latency access to small amount of data in a large data set
- Fast scan across tables
- Random access to rows

Common use cases:

- Applications requiring sparse rows
- Not good for relational analytics and transactional needs

Notes

## Hive, Spark SQL, and Presto

50

Basics on Hive:

- Access SQL data in HDFS using an SQL-like query language HQL
- Convert queries to MapReduce, Tez, or Spark jobs
- Warning: does not fully comply to ANSI-standard SQL

Basics on Spark SQL (formerly Shark):

- Was initially a port of Hive to Spark
- Follows Spark in-memory computing model
- Is "mostly" compatible with HQL

Basics on Presto:

- Supports ANSI-SQL standard
- Uses a custom engine, not based on MapReduce
- Can access various data sources through storage plugins

Notes

## Serialization and storage

51

Avro:

- Input: a schema describing the data and the data
- Output: generates the code to read/write data

Parquet:

- Columnar storage format
- Complex to handle

Java Script Object Notation (JSON):

- Not part of Hadoop
- Often preferred to XML by Hadoop community
- Represent data using key-value pairs

Notes

## Management and monitoring

52

Ambari:

- Production-ready, easy to use web-based GUI for Hadoop
- Eases the installation and monitoring of a cluster

Zookeeper:

- Effective mechanism to store and share small amounts of states and configuration across the cluster
- Not a replacement for any key-value store
- Has built-in protections to prevent using it as large data-store
- Used as a coordination service

Notes

## Analytics helpers

53

Major analytics helpers:

- Pig: high-level language to speak to MapReduce
- Hadoop streaming: write mappers/reducers in any language
- Mahout: set of scalable machine-learning algorithms for Hadoop
- MLlib: similar to Mahout, based on Spark (maintenance mode)
- Spark ML: similar to MLlib based on a higher level API
- Hadoop Image Processing Interface: package allowing to examine images and determine their differences and similarities

Notes

## Data transfer

54

Moving data to and from Hadoop:

- Sqoop: transfer data between HDFS and relational databases
- Flume: distributed system for collecting, aggregating, and moving large amount of data from various sources into HDFS
- Distributed Copy (DistCP):
  - Part of basic Hadoop tools
  - Used to move data between the clusters
  - Is the basis for more advanced Hadoop recovery tools

Notes

## From batch to realtime processing

55

Lambda data architecture:

- Setup three layers:
  - Batch layer: store all incoming data and batch process it
  - Speed layer: analyse incoming data in real time
  - Serving layer: serve curated data that can be analysed by other tools
- Drawback: maintain two code sets for batch and speed layers

Kappa data architecture:

- Not a replacement but an alternative to lambda architecture
- Layers: batch layer is removed compared to lambda architecture
- Suitable for systems with strict end-to-end latency requirements
- Drawback: replay the whole stream in case of error

Notes

## Realtime processing

56

Apache Storm:

- Distributed system for real-time processing of streaming data
- Able to process over a million records per second per cluster node
- Relies on Zookeeper for coordinating the nodes

Apache Kafka:

- Distributed platform used to create real-time streaming data pipelines
- Heavily relies on zerocopy (OS kernel level) to move data around
- Commonly used together with Spark, Flink, or Storm

Remote Dictionary Server (Redis):

- In-memory key-value data-store
- Extremely fast, simple, and versatile
- Benchmarked as the fastest DB in the world

Notes

## AI and ML

57

Ray:

- Spark's goal was to replace Hadoop and Ray's goal is to replace Spark
- Run fast machine learning or deep learning-based applications
- Hope to reach MPI power and granularity levels

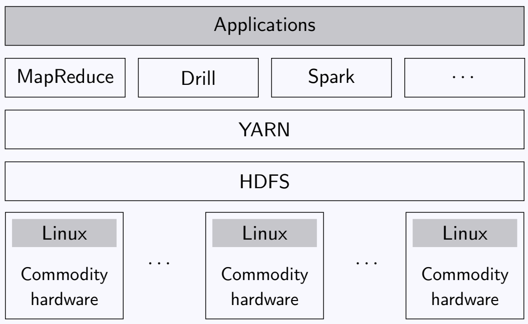
TensorFlow:

- Python-friendly library for fast and easy machine learning computation
- Data is stored is a tensor
- Create dataflow graphs where the edges are tensors and vertices mathematical operations
- Can run on Hadoop (TonY), Spark, and Ray

Notes

## Hadoop ecosystem in VE472

58




*Refer to Hadoop ecosystem table for more details*

Notes

## Key points

59

- What is big data?
- List limitations of common solutions when analysing large data sets?
- What is Hadoop?
- What are Hadoop's goals?
- Briefly describe the Hadoop ecosystem

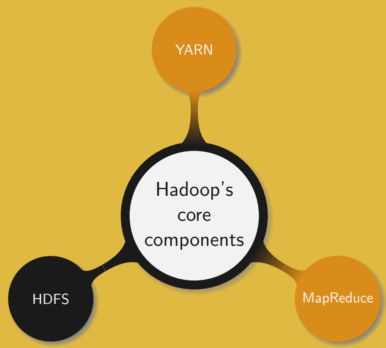
Notes

## 2. Hadoop's core components

Notes

61

## Chapter organisation




Notes

62

## Distributed filesystems

Regular filesystems on a computer:

- Partition
- Hard drive
- LVM

Distributed filesystems:

- Spans several computers
- Has to deal with potential network issues

*Idea behind HDFS summarised on slide* 1.36

Notes

63

## File blocks

Blocks in HDFS:

- Default size of 128 MB
- Files smaller than a block size do not occupy the whole block
- A file can be larger than a whole disk
- Data and metadata handled separately
- Easy to implement fault tolerance and availability

Notes

64

## Nodes

Two types of node in a cluster:

- Namenode:
  - Maintains FS tree and metadata for all files and directories
  - Locally stores information in namespace image and edit log
  - Knows on which datanodes the blocks of a file are located
- Datanode:
  - Stores and retrieve blocks
  - Regularly reports the list of stored blocks to namenode
  - Can store certain blocks in cache

Notes

65

## Finding a file

A namenode has no persistent copy of where blocks are:

- Each datanode announces the blocks it has
- All the information is kept in memory by the namenode
- When a write occurs an entry is added to the edit log

*What to do if the namenode fails?*

Notes

66

## Memory limitations

A namenode stores all the blocks of all the files in its memory:

- Assume 1 GB of memory for 1 million blocks
- 200 nodes cluster, 24 TB each: ∼12 GB of memory
- Cluster at Yahoo!: 25 PB → ∼64 GB of memory
- Cluster at Facebook: 60 PB → ∼156 GB of memory

*How about having more namenodes?*

Notes

67

## HDFS federation

Allowing more namenodes:

- Split the filesystem over several independent namenodes
- Each namenode has a namespace
- Each namespace has its own pool of blocks
- A namespace with a block pool is called namespace volume
- A datanode is not attached to a specific namespace volume

Notes

68

## High availability

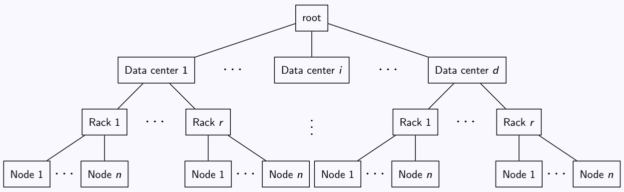
Two namenodes in an active-passive mode:

- Passive node takes over in case of failure of the active one
- The two namenodes share the same edit log
- Only the active namenode can write to the edit log
- Passive namenode reads entries when written in edit log
- Datanodes send block reports to both namenodes
- Passive namenode also works as secondary namenode
- Clients must be configured to handle namenode failures

Notes

## Cluster balancing

The goal is to evenly spread blocks across the whole cluster:

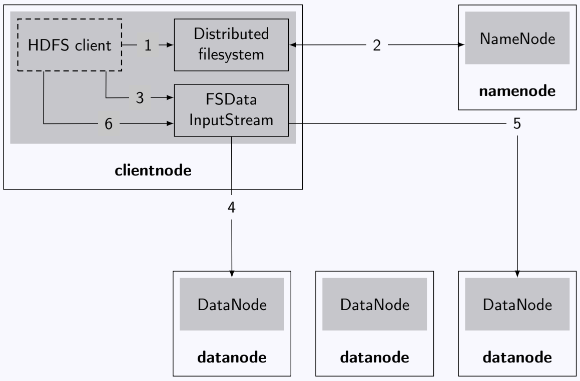



Replicas' location:

- First: same node as the client
- Second: random, different rack from the first
- Third: same rack as the second but different node
- Others: random nodes in the cluster

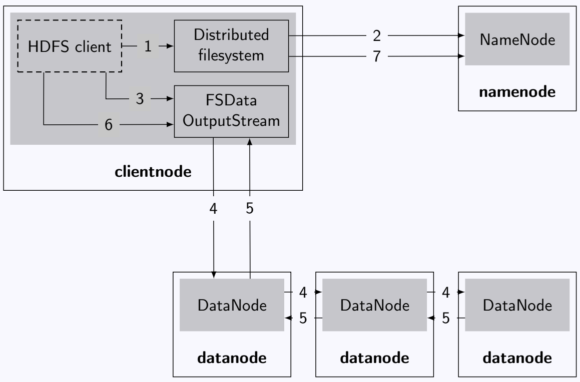
## Notes

## File read




## Notes

## File write




## Notes

## Failing file write

When write on a data node fails:

1. Close the pipeline
2. Add packets in front of the acknowledgment queue
3. Inform the name node of the failing data node
4. Remove the faulty data node from the pipeline
5. Construct a new pipeline using only the healthy data nodes
6. Complete the writing of the block across the pipeline
7. Arrange for the replication of under-replicated blocks

## Notes

## Other filesystems

Alternative filesystems built on top of Hadoop abstract filesystem:

- HAR: pack HDFS files into an archive
- View: used to create mount points for federated FS
- FTP: FS backed by an FTP server
- S3: FS based on Amazon S3
- Azure: FS developed by Microsoft Azure

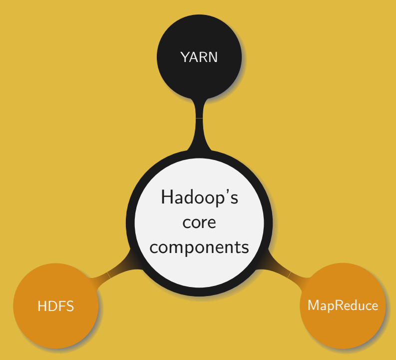
Interfaces to access HDFS data:

- libhdfs: C library with API similar to the Java one
- NFS: use Hadoop NFS gateway
- FUSE: based on libhdfs

Notes



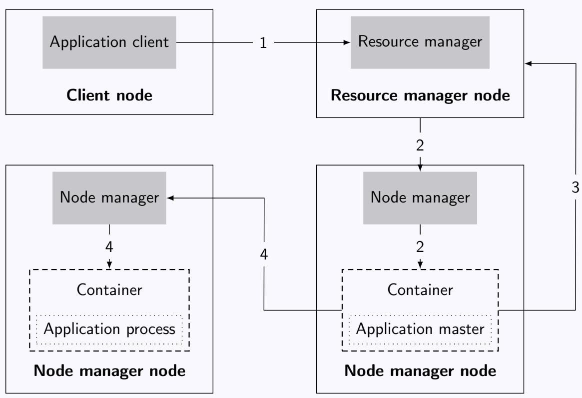
## Chapter organisation




Notes



## Running an application




Notes



## Requesting resources

A request has two components:

- Amount of resources for each container
- Preferred location of the containers

Common strategy for a request:

- Announce all the resources requests at the beginning
- Dynamically request resources based on the needs

Notes

## Categories of application

77

YARN can be used in three ways by applications:

- One application per user job: simplest model
- One application per user session:
  - Containers can be reused between jobs
  - Possibility to cache data between jobs
- Long-running application shared among users:
  - Application master is always "on"
  - Application master acts as a coordinator for other applications

Notes

## YARN schedulers

78

Three schedulers available in YARN:

- FIFO: request served one by one in a queue
- Capacity:
  - Define queues based on the "size" of the jobs to complete
  - All the jobs start early
  - Resources are wasted when unused
- Fair:
  - Resources are dynamically balanced over all the jobs
  - All the resources are fully used
  - Delay due the resource reallocation

Notes

## Capacity scheduler

79

Capacity scheduler setup:

- Each queue is handled as a FIFO
- Queue elasticity
  - A single job cannot exceed the capacity of the queue
  - The capacity can be exceeded when several jobs wait and resources are available
- Containers are not preempted
- Can control:
  - The number of resources per user/application
  - The number of applications that can be run at a time
  - Access Control Lists (ACL) on the queues

*The challenge is to find a reasonable trade-off for the capacity*

Notes

## Fair scheduler

80

Fair scheduler queues:

- One or more queues allowed:
  - Single queue: resources are fairly shared among all applications
  - Several queues, each having:
    - Its own scheduling policy
    - A max/min resources and number of applications
- Queues can be precisely configured using an allocation file
- By default a queue is dynamically created for each user

Notes

81

## Fair scheduler

Preemption in the fair scheduler:

- Efficiency is reduced as killed containers must be restarted
- Two timeout settings $t_1$ and $t_2$ to trigger preemption:
  - Minimum share: if a queue waits longer than $t_1$
  - Fair share: if a queue remains below half of its fair share for longer than $t_2$
- Timeouts can be set per queue or globally

Notes

82

## Delay scheduling

Locality problem when scheduling:

- An application requests a specific node
- The node is busy
- Should the application wait for the node or loosen its request?

YARN schedulers' approach:

- Every second each node manager sends a heartbeat reporting the running containers and available resources
- Capacity scheduler: wait for a predefined number of heartbeats before loosening the requirement
- Fair scheduler: wait for a predefined portion of nodes in the cluster to offer opportunities before loosening the requirement

Notes

83

## Dominant resource fairness

*How to fairly share resources between applications when they do not use the same type of resources?*

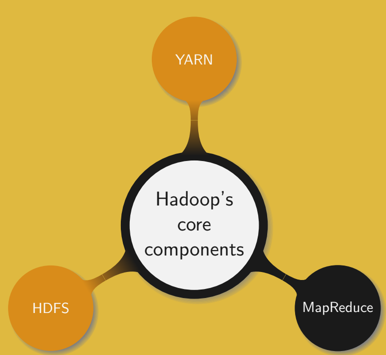
Basic idea for two applications:

- Consider the proportion of resources requested for a container by each application
- Call the largest proportion the dominant resource and use it as measure of cluster usage
- Proportionally offer less containers to the more demanding application

Notes

84

## Chapter organisation




Notes

## MapReduce job

Parties involved in a MapReduce job:

- A client which initiates the job
- YARN resource manager
- YARN node manager
- MapReduce application master
- HDFS

## Notes



## Job initialisation

Starting a MapReduce job:

1. Request a new application ID to the resource manager
2. Check the job parameters
3. Split the job into subtasks
4. Copy the splits and other necessary information to run the job onto the shared FS
5. Effectively submit the job on the resource manager

## Notes



## Job startup

Running a MapReduce job:

1. YARN scheduler allocates a container
2. Application master launched by the resource manager
3. Setup the tasks
4. Retrieve the splits from the shared FS
5. Create a Map task for each split and specify the number of tasks for the Reduce part
6. Resources for
   - a. Small tasks: run on the same node
   - b. Large tasks: contact the resource manager for more containers
     - i. Request resources for the maps (high priority)
     - ii. Request resources for the reducers when enough maps have completed
7. Locate the data on the distributed FS and start the task

## Notes



## Job failure

Potential points of failure:

- Task failure
- Application master failure
- Node manager failure: YARN level (no heartbeat received)
- Resource manager failure: YARN level (high availability mode)

Protection and progress monitoring:

- Tasks are run in a separate JVM: avoid crashing namenode
- Each task has a status and some counters
- Each task reports its progress to the application master
- Client application polls the application master every second to retrieve the latest status

## Notes

## Task failure

Task failure occurrences and solutions:

- The map or reduce task fails:
  - When receiving a failure notice the application master marks the task as failed
  - The container is freed and resources released
- The JVM crashes: the node manager notices the application manager of the failure
- A task hangs:
  - Tasks marked as failed if no report is received
  - The JVM is killed by the application master

*Failed tasks are rescheduled on a different nodemanager*

## Notes



## Application master failure

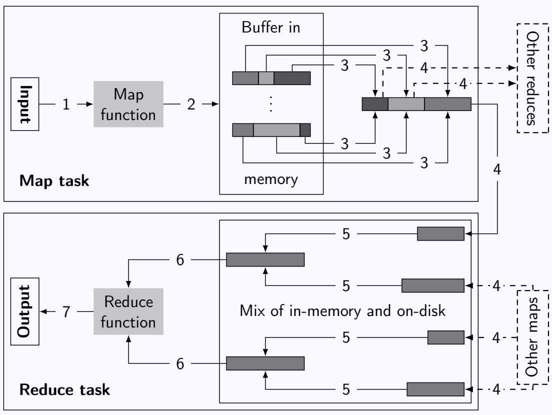
Failure detection and recovery:

- The application master sends periodic heartbeats to the resource manager
- On failure a new instance is run on a new container
- The tasks progress is known so it is possible to resume without re-running the completed tasks
- Each task caches the application master's address
- Use a timeout after which the resource manager is contacted for the new application's master address

## Notes



## Map and Reduce




## Notes



## Speeding up a MapReduce job

Optimized configuration setup:

- Provide shuffle with as much memory as possible
- Keep enough memory for map and reduce functions
- Optimize the code with respect to memory consumption
- Minimize the number of spills for the map part
- As much as possible keep intermediate reduce data in memory

Speculative execution:

- A task is detected as much slower than average
- Re-run it on a different node
- Kill all the other duplicates as soon as one completes

## Notes

93

## Key points

- Explain the basic structure of Hadoop
- What are the main principles behind HDFS?
- How is YARN used by higher level applications?
- Describe the structure of a MapReduce job

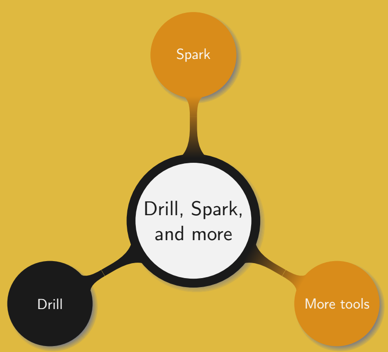
Notes

# 3. Drill, Spark, and more

Notes

95

## Chapter organisation




Notes

96

## Drill job

Parties always involved in a Drill job:

- A client which initiates the job
- Zookeeper

Parties optionally involved in a Drill job:

- YARN
- HDFS
- Hive
- HBase

Notes

97

## Drill on YARN workflow

Running Drill as a YARN application:

1. Start Drill on the client machine
2. Upload resources to the FS and request resources for the application master
3. Ask a node manager to prepare and start a container for the application master
4. The application master contacts the resource manager to obtain more containers

## Notes

98

## Drill on YARN workflow

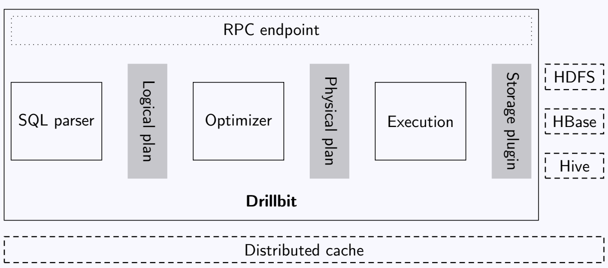
Running Drill as a YARN application:

5. Request the start of Drill software on each assigned node
6. Start a "Drill process" called a *drillbit*
7. Each drillbit starts and registers with Zookeeper
8. The application master checks the health of each drillbit through Zookeeper
9. Use Zookeeper to retrieve information on the drillbits, run queries, etc.

## Notes

99

## Drillbit




Foreman drillbit:

- Drillbit that receives the query
- It drives the entire query

## Notes

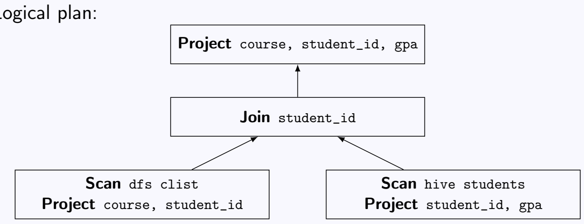
100

## Logical plan

Initial SQL query:

```sql
SELECT course, student_id, gpa
FROM clist.json l, hive.students s
WHERE l.student_id = s.student_id
```

Logical plan:




## Notes

101

## Drill query execution

Major fragment:

- Concept representing a phase of the query execution
- Composed of minor fragments

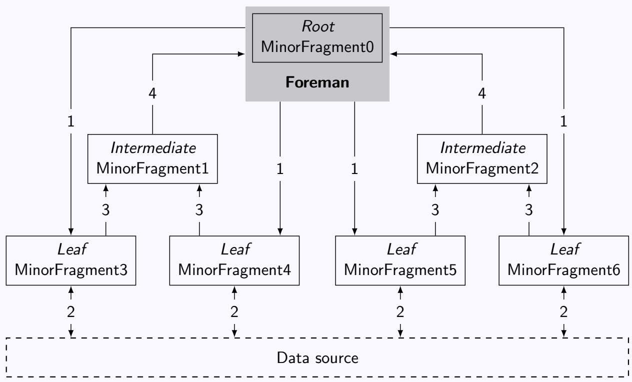
Minor fragment:

- Logical unit of work running in a thread
- Contain one or more relational operators
- Usually as numerous as the number of available drillbits
- Scheduled based on data locality when possible and round-robin otherwise

Notes

102

## Minor fragments execution tree




Notes

103

## Drill's strength

Architecture:

- No central server, no master-slave concept
- Each drillbit contains all the services and capabilities of Drill
- Nodes can be added or removed at no cost

Columnar execution:

- Avoid access for columns not involved in the query
- Directly performs SQL processing on columns

Notes

104

## Drill's strength

Optimistic query execution:

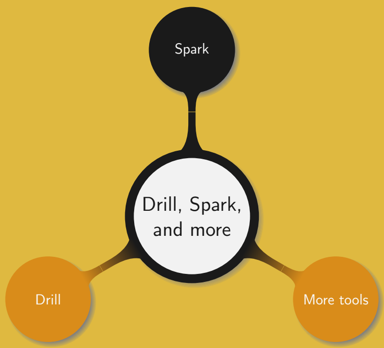
- Assume no failure will occur during query execution
- Rerun the query in case of failure
- Only write on disk when memory overflows

Vectorization: allow the CPU to operate on vectors

Runtime compilation: generate efficient code for each query

Notes

105 Chapter organisation




Notes

106 Spark job

Spark organisation:

- Application: user program built on Spark and composed of:
  - A driver program: process running the main function
  - Executors: processes launched for an application on worker nodes
- The driver program connects to a cluster manager
  - Standalone: cluster manager provided with Spark
  - YARN
  - Mesos
  - Kubernetes

Notes

107 Spark on YARN workflow

Two modes available:

- Client mode:
  - Driver runs in the client
  - Required in the case of interactive programs
  - Useful when building a Spark program
- Cluster mode:
  - The entire application runs in the cluster
  - Appropriate from production jobs
  - YARN application master failure strategy (slide 2.90) is applied

Notes

108 YARN client mode

Spark job workflow:

1. Start the driver program on a client node
2. The driver requests a container to the resource manager
3. A container starts and runs an Executor Launcher application master
4. The Executor Launcher requests more resources to start Executor backends processes in new containers
5. Each Executor Backend registers with the driver

Notes

## YARN cluster mode

109

Workflow similar to client mode but:

- The driver program runs in a YARN application master process
- The client submit a job but does not run any user code
- The application master starts the driver program
- The driver program "replaces" the Executor Launcher

Remark. Data locality:

- Executors are launched before data locality information is available
- The driver can optionally specify preferred locations

Notes

## Resilient Distributed Dataset

110

Resilient Distributed Dataset (RDD):

- Core abstraction in Spark
- Collection of objects distributed across a cluster:
  - Read-only: do not alter a dataset, transform it into a new one
  - Resilient: no disk write, reconstruct the RDD in case of partition loss
- Loaded as input:
  - Created from an external dataset
  - From an existing RDD
  - Parallelising an existing collection

Notes

## RDD operations

111

Two types of operations on an RDD:

- Transformation:
  - Create a new dataset from an existing one
  - Only compute the result when an action is run
  - Do not return any result to the driver program
- Action:
  - Run a computation on a dataset
  - Return the value to the driver program

Benefits of this approach:

- Transformed RDD is in memory when performing an action
- No large dataset to send back to the driver program

Notes

## RDD persistence

112

Datasets are cached in memory across operations:

- An RDD is stored on the node where it was computed
- An old RDD is dropped following the LRU algorithm
- A lost RDD is automatically recomputed if needed

Caching levels:

- Memory only: no compression, lost partitions are recomputed
- Memory and disk: partitions that do not fit in the memory are spilled on disk
- Memory only serialized: compression enabled
- Replication: all the above but also replicate on another node

Notes

## Functions and variables

113

Serialization of data and functions:

- Used to share information among the executors
- Transparent to the user

Task closure:

- Cannot share variables among executors
- Determine what variables and methods an executor needs
- Serialize this closure and send it to the executor
- Each executor receives a copy of the original variable
- Variables are not updated on the driver

Notes

## Shared variables

114

Broadcast variables:

- Read-only variables broadcasted to each executor
- Data sent in an efficient way to minimize traffic
- Useful for data needed over several stages of the computation

Accumulators:

- Variables that can be added to using associative and commutative operations
- The driver can retrieve their value
- They are only updated on action tasks
- Update only occurs once, even if an action is rerun

Notes

## Running of a Spark job

115

Job submission and execution:

- A job is submitted when an action is performed on an RDD
- The transformations on the RDD are organised into a logical execution plan
- Spark DAG scheduler transforms the logical plan into a execution physical plan
- The physical plan defines stages, split into tasks
- Spark task scheduler constructs a mapping of tasks to executors
- The executor runs the task
- Executors send status updates to the driver when a task is completed or has failed

Notes

## Higher level abstractions

116

Newer and higher level abstractions relying on RDD:

- Datasets:
  - Foundational type of the structured APIs
  - Provide type-safety and allow much flexibility
  - Only available in Java and Scala and comes at a performance cost
- DataFrames:
  - Similar to a spreadsheet split into partitions
  - Internally defined as DataSets of type `Row`
  - Most common structured API, supported in all languages
- SQL Tables:
  - Data structure similar to DataFrames but defined within a database
  - Unmanaged table: defined from a file on the disk
  - Managed table: imported in and managed by Spark

Notes

## Spark's strength

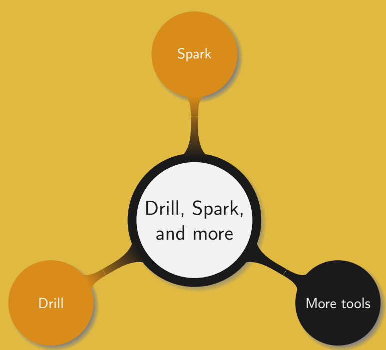
Advantages of DAG:

- Any lost RDD can easily be recovered
- Offers more possibilities than a simple Map and Reduce approach
- Transformation on RDDs are not directly applied:
  - Allows better optimizations
  - Decreases disk writes and data transfer

Remark. Spark generalises the MapReduce approach, is much faster, and features many more high-level operators

Notes

## Chapter organisation




Notes

## Scaling up software

Simple observations:

- Over 20 billions devices are connected to the internet
- The complexity of software keeps increasing
- New security challenges need to be addressed
- Package manager dependencies are complex to handle

Alternative package management systems:

- Flatpak, Snap, AppImage:
  - Distribution agnostic packages
  - Applications are sandboxed, i.e. isolated from each others and the host
- Nix: *all* packages are isolated from each others

Notes

## Linux containers

LinuX Containers (LXC):

- Operating-system-level virtualization method
- Relies on the kernel's *cgroup* and *namespace* isolation functionalities
- Concurrently run multiple isolated Linux OS on a machine
- Each container must be individually maintained
- Containers can be either privileged or unprivileged
- Containers access the bare machine and rely on the host kernel

*LXC requires the setup of a whole OS for each container*

Notes

## Docker



Docker uses a different approach than LXC:

- Initially based on LXC but *now* completely independent
- A daemon manages the docker containers
- A container is an encapsulated environment that runs applications
- An image containing an application and its dependencies is built based on a “configuration file”
- To update simply replace the old image with a new one

*Docker needs to be “manually” deployed, managed, and scaled*

## Notes

## Kubernetes



Kubernetes is a container orchestration tool:

- Mostly used with, but not limited to Docker
- Initially developed as an internal Google project
- Kubernetes is a cluster application that handles a cluster:
  - *Master:* controls all other machines in the cluster
  - *Nodes:* the machines onto which applications are running
  - *Pods:* single instance of an application or running process

## Notes

## Kubernetes



Main tasks of Kubernetes:

- Monitor the health of the running applications
- Balances the load
- Manages hardware resources allocation
- Eases the deployment of preconfigured applications
- Allows access to storage in the same way as any other resources

During the lifespan of an application:

- Containers can live, die, be resurrected
- Kubernetes handles everything without any human interaction

*Kubernetes can be coupled to Hadoop or work independently*

## Notes

## Key points



- When to use Drill?
- Explain the organisation of a drillbit
- When to use Spark?
- How does Spark interact with YARN?
- What are containerized solutions?

## Notes

Thank you, enjoy the Summer break!

## Notes

## References I

1.30 https://upload.wikimedia.org/wikipedia/commons/0/0e/Hadoop_logo.svg

## Notes