General Distributed Data Processing

What if you need to perform an analysis that is not supported by your (distributed) database?

- Breadth-First search
- PageRank
- k-means

Possible Solution

Implement the analysis in your favorite language.
Certain tasks however need to be handled in every implementation:

- Parallelization/synchronisation
- Distribution of computation
- Distribution of data
- Communication between nodes
- Node failures

$\Rightarrow$ Overall, creating a custom solution takes time and is error prone.

Alternative Solution

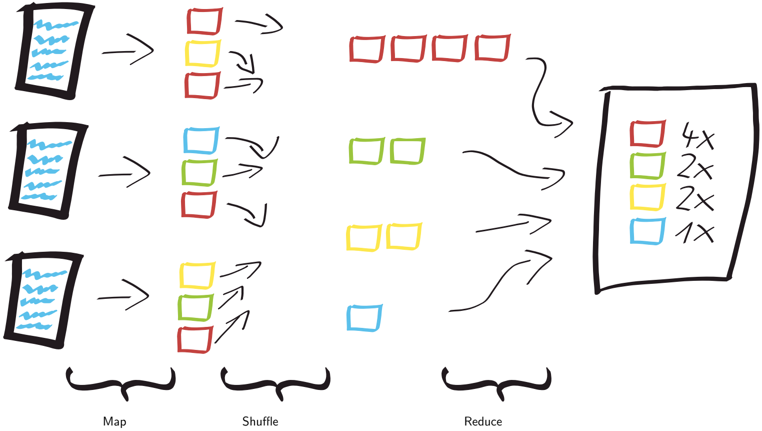
Idea

- Use a functional abstraction that specifies data-parallel computation: map and reduce functions
- A runtime systems manages
  - Program execution
  - Data distribution
  - Persistence
  - Node failures
  - Communication

Program formulation with

- **Map** all input records to key-value pairs
- **Reduce:** Computation on all pairs with the same key

Word frequency with MapReduce



- map produces (⟨`word`⟩, `1`) pairs
- reduce sums up `1` of all pairs with same `word`

Map-Reduce

A MapReduce program is defined by at least two functions:

$$\begin{aligned} \textit{map} &: (k_1, v_1) \longrightarrow [(k_2, v_2)] \\ \textit{reduce} &: (k_2, [v_2]) \longrightarrow [(k_3, v_3)] \end{aligned}$$

Map is called for all input pairs and for each produces a list of key/value pairs. Reduce is called once for each key that was produced by any map call with a list of all corresponding values.

Word frequency:

```
def map(name, doc):
  for word in '␣'.split(doc):
    emit(word, 1)

def reduce(key, values):
  emit(key, sum(values))
```

Using MapReduce

- PageRank
- Join two relations

MapReduce Implementations

- Google has a proprietary implementation in C++
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, now an Apache project
  - Designed for very large compute clusters to handle very large data sets
  - Used in Production at Yahoo, Facebook, Twitter, LinkedIn, Netflix,...
  - The de facto big data processing platform
- Losts of custom research implementations

Hadoop Architecture

Core modules

- Common
- Distributed File System
- YARN, scheduling and cluster resource management
- MapReduce, system for parallel processing of large data sets

YARN manages the cluster

- Resource manager
- Node manager

Hadoop MapReduce is an application on the cluster that negotiates resources with YARN.

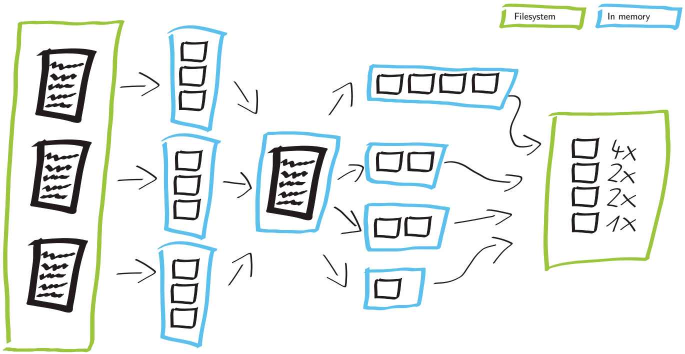
- JobTracker globally manages jobs, gives tasks to
- TaskTracker to execute on their local nodes

Lifecycle of a MapReduce request

- User submits job configuration JobTracker.
- JobTracker submits mapper tasks to TaskTrackers, thus schedules computation on all available cluster nodes.
- Each task works on the data local to its node.
- While mappers produce output, the shuffle process moves the output among the nodes in the cluster.
- Once the mappers and shuffling finishes, the JobTracker schedules reduce tasks to all nodes.

How is data shared between nodes?

- Hadoop uses the Hadoop distributed filesystem (HDFS) for durability and data distribution
- Map input is read from HDFS
- Map output is pre-sorted and written to HDFS
- Shuffle phase moves data between nodes and merges pre-sorted data
- Results of reduce phase are written to HDFS



Fault Tolerance

HDFS is able to cope with failing nodes

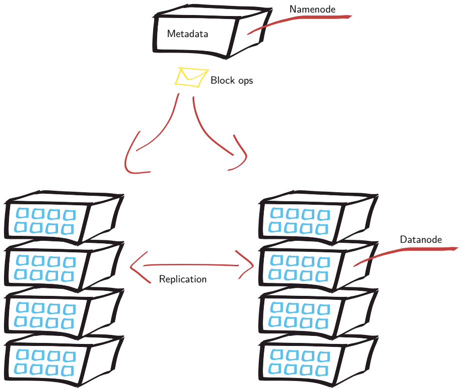
- Files are split into blocks (default size 64MB)
- Blocks are replicated onto different machines (default: 3 replicas)
- In case of node failure, missing replicas are reconstructed on other nodes

JobTracker is able to cope with failing nodes

- On failure, JobTracker will notice that one TaskTracker is non-responsive.
- If the job is still in the mapping phase, tasks of the failed tracker will be distributed to TaskTrackers on other nodes.
- If job is in reduce phase, those reduce tasks that did not finish are restarted on other TaskTrackers.

HDFS Architecture

- **Name node** manages file system and regulates access
- **Data nodes** manage attached storage
- HDFS exposes append-only files to users
- Internally, files are split into blocks and replicated across name nodes for failure tolerance



Implementing Word Frequency in Hadoop MR

Mappers must extend `org.apache.hadoop.mapreduce.Mapper` and implement the map function:

```
public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>
  public void map(Object key, Text value,
    Context context)
    throws IOException, InterruptedException {
      // write to output with
      context.write(key, value);
  }
}
```

Reducers must extend `org.apache.hadoop.mapreduce.Reducer`:

```
  public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable
   public void reduce(Text key, Iterable<IntWritable>
                         Context context
                         ) throws IOException, Interrupt
```

Optimizations

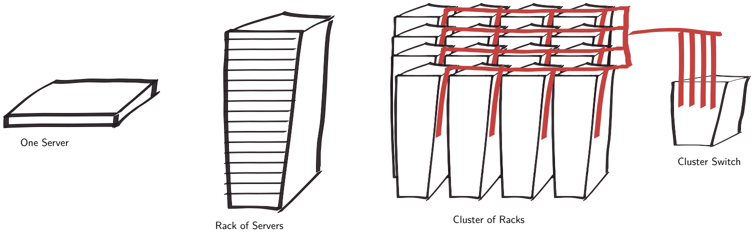
- Add combiner to reduce amount of transferred data
- Fold map and reduce steps to save on disk writes

Programming for a data centre

We have seen the MapReduce abstraction for scalable programming. But how can we create programs that run well on large clusters?

- Need to understand the design of warehouse-sized computers
  - Different techniques for different setting
  - Requires quite a bit of rethinking
- MapReduce algorithm design
  - How to express everything in terms of map(), reduce(), combine(), and partition()?
  - Are there any design patterns we can leverage?

Data centre architecture



- Computers in racks
- Racks in clusters, connected via cluster switch

Storage hierarchy

Accessing data on remote machines comes at a price:

- Local DRAM, 16GB, 100ns, 20GB/s
- Local Disk, 2TB, 10ms, 200MB/s
- Rack local DRAM (80 servers), 1TB, 300μs, 100MB/s
- Rack local Disk (80 servers), 160TB, 11ms, 100MB/s
- Cluster DRAM (30 racks), 30 TB, 500μs, 10MB/s
- Cluster Disk (30 racks), 4.8PB, 12ms, 10MB/s

Scaling up vs. out

- In case no single machine is large enough
  - Should we use a cluster of large (expensive) machines or a large cluster of (cheap) commodity machines?
- Nodes need to talk to each other!
  - Intra-node latencies: ca. 100ns
  - Inter-node latencies: ca. 100μs
- Let's model communication overhead

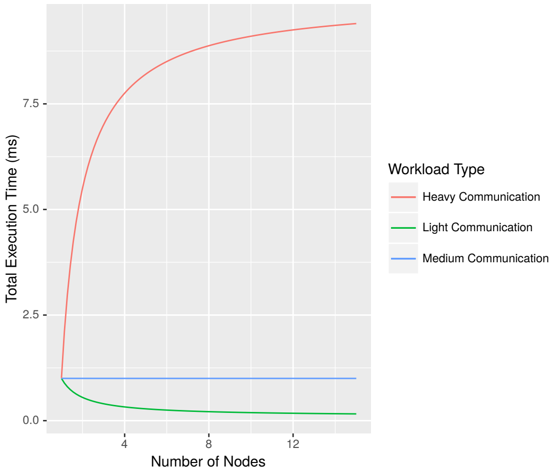
Modelling communication overhead

- Simple execution cost model
  - $TotalCost = CostOfComputation + CostToAccessGlobalData$
  - Fraction of local access is inversely proportional to size of cluster
    - $1/n$ of the data is local
  - n nodes in cluster (ignore cores for now)
    - $TotalCost = 1ms/n + f \cdot [100ns \cdot (1/n) + 100\mu s \cdot (1 - 1/n)]$
  - Three scenarios:
    - Light communication: $f = 1$
    - Medium communication: $f = 10$
    - Heavy communication: $f = 100$
- What is the cost of communication?

Communication cost

- Workloads with light communication benefit from larger clusters
- Large amounts of random communication impede performance

⇒ avoid random reads from remote!



Seeks vs. scans

- Consider a 1TB database with 100 byte records
  - We want to update 1% of the records
- Scenario 1: random access
  - Each update takes 30ms (seek, read, write)
  - 10^8 updates = 35 days
- Scenario 2: rewrite all records
  - Assume 100MB/s throughput
  - Time = 5.6 hours

$\Rightarrow$ avoid random reads from disk!

HDFS Design Considerations

Hardware Failure	Hardware failure is the norm rather than the exception.
Streaming Data Access	Applications that run on HDFS need streaming access to their data sets.
Large Data Sets	Applications that run on HDFS have large data sets.
Simple Coherency Model	HDFS applications need a write-once-read-many access model for files.
Move Computation, Not Data	A computation requested by an application is much more efficient if it is executed near the data it operates on.
Portability	HDFS has been designed to be easily portable from one platform to another.

Hadoop MapReduce use case

Hadoop MapReduce was built with the following assumptions

- The data sets are much larger than available main memory
- Processing happens on very large clusters of commodity machines

Therefore,

- intermediate results are written to disk
- shuffle phase is followed by a merge-sort phase

This design makes Hadoop suitable for big-data use cases, however it is not necessarily a good fit for workloads with a lot of computation and/or smaller data sets.

What is Spark?

Spark is an alternative more expressive fault-tolerant runtime for distributed batch data analytics.

Architecture

Programming Model	Resilient Distributed Datasets (RDDs), DataFrame API, Dataset API
Runtime	Spark
Cluster Manager	Hadoop Yarn, Builtin, Apache Mesos
Distributed Filesystem	HDFS, Local files, ...

Why a New Programming Model?

- The primitives map and reduce only allow very naive algorithm implementations. Hadoop thus also has to provides additional hooks to override: Sorting, Partitioner, GroupingComparator, ...
  Use-cases:
  - Outputting globally by key sorted data.
  - Increasing data-locality by using special partitioners, instead of hash-partitioning.
  - Answering window queries by sorting the values per group for the reducer.
- A single Map/Reduce job is only rarely enough, need to express more complex data pipelines.
- Lightweight recovery mechanisms. Heavyweight mechanisms contribute more than $> 70\%$ of total job cost in Hadoop.

The Spark Stack

Further convenience APIs on top of spark:

- Spark SQL - Express queries using SQL
- Spark Streaming - Allows for streaming data analysis instead of only batch analysis
- GraphX - An API especially tailored towards analyzing graphs and also implementing custom graph algorithms
- MLIB - Provides machine learning algorithms

Evolution of the Spark Programming Model: RDD

Key idea: resilient distributed datasets (RDDs)

- + Distributed collection of objects that can be cached in memory across the cluster
- + Manipulated through parallel functional operators (no need to write tons of boilerplate classes)
- + Chain transformations easily and ability to store pipelines into variables for easy composition
- + Materialization of intermediate steps optional as data can just be recomputed on node failure.

Limitations

- - Datamodel still opaque blobs, no optimization Possible
- - Garbage collection overhead when keeping objects in memory

Programming interface

- Functional APIs in Scala, Java, Python
- Interactive use from Scala shell

Evolution of the Spark Programming Model: DataFrame

Key idea: More declarative API - Spark SQL and DataFrame API

- $+$ Declarativity allows query plan optimization
- $+$ Strongly typed data model allows for optimized storage
- $+$ Query compilation avoid interpretation overhead

Limitations

- - No custom lambdas possible, DataFrames first need to be converted to RDDs
- - Spark SQL has no syntax checking and DataFrame API also limited

Evolution of the Spark Programming Model: Dataset API

Key idea: Combination of RDD and DataFrames

- + When fully using strong typing everything is checked during compile time
- + Optionally also weakly typed objects usable
- + Retains optimizability and code compilation

Footnote: The DataFrame and Dataset API's internally are mapped to RDDs and can use linage based recovery methods.

Using the Dataset API

Basic principle:

1. Create initial dataset.
2. Analyze it by calling the corresponding methods, e.g. limit(...), filter(...). Each method again returns a Dataset object.
3. Evaluation only starts once the result is required, e.g. by using count(), show() or collect().

Supported transformations

- agg
- distinct
- except
- filter
- flatMap
- groupBy
- intersect
- joinWith
- limit
- map
- orderBy
- sample
- select
- sort
- union

Examples

- Word-Count (Aggregation)

Extra Dataset functions

Managing Datasets:

- persist
- unpersist
- explain
- printSchema

Collecting results:

- describe
- first
- count
- show
- collect
- foreach

Examples

- Word-Count (Aggregation)
- TPC-H Join (other SQL constructs)
- PageRank

Fault Tolerance

RDDs track lineage information to rebuilt lost data.
Snapshots are automatically created after repartitioning transformations and managed using an LRU cache to speed up recovery.

Spark Detail: Optimizer

Steps:

1. Logical plan from Dataset / DataFrame / Spark SQL
2. Apply transformation rules to get optimized logical plan
3. Generate physical plan and select cheapest by using cost model

Spark Detail: Tungsten Runtime

Spark's runtime focuses on three main pain points:

- Memory Management and Binary Processing: leveraging application semantics to manage memory explicitly and eliminate the overhead of deserialization into the JVM object model and garbage collection
- Cache-aware computation: algorithms and data structures to exploit memory hierarchy (columnar storage and compression)
- Code generation: using code generation to exploit modern compilers and CPUs

Examples: Code compilation

Whole stage generation can be identified by calling explain(). Entries marked with (*) are using it.

- Word-Count (Aggregation)

What Makes it Faster?

- Support for general DAGs
- Optimizer and Binary / Code Generation Runtime
- Lower-latency engine (Spark OK with 0.5s jobs)
- Column-oriented storage and compression

What it Means for Users

Separate frameworks on Hadoop:

- HDFS read, ETL, HDFS write
- HDFS read, train, HDFS write
- HDFS read, query, HDFS write

Spark:

- HDFS read, ETL, train, query (combine Python, Scala, Java)

Conclusion

- Big data analytics is evolving to include
  - More **complex** analytics (e.g. machine learning)
  - More **interactive** ad-hoc queries
  - More **real-time** stream processing
- Spark is a fast platform that unifies these applications
- More info: http://spark.apache.org

Cloud Computing

For distributed data mangling, a cluster of machines is necessary. However configuring your own cluster is not.
Many providers offer these services for rent:

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

Services

Infrastructure	Rent compute resources, storage capacity, network connectivity, mobile devices
Platform	Rent platform software: databases, webservers, CDNs, cluster manager, user tracking
Software	Rent user software: developer tools, data analytics tools, mail services, office packages and much more

Service Providers

Biggest players:

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform

Example: Amazon Services

Infrastructure Services:

- EC2: virtual machines
- S3: distributed storage
- VPC: virtual private network

Platform as a service:

- RDS: relational databases
- EMR: MapReduce cluster, e.g. to run Spark
- Athena, Data Pipeline, ...: analytics tools
- dozens of more services available, see aws.amazon.com/documentation

Amazon EC2

Provision as many virtual machines as you need. Instances specs.[1]:

- With RAM from 0.5 to 1952 GiB
- With 1 to 128 vCPUs (vCPU is a hyperthread most of the time)
- Up to 16 GPUs (e.g. NVIDIA K80 GPUs with 16GB RAM)
- Up to 8 FPGAs
- Up to 8 x 800 GB SSDs or 24 x 2000 GB HDDs

Prices depend on your usage:

Smallest instance: $0.0059/h

Largest general purpose instance: $3.447/h

Largest compute instance: $1.591/h

Largest memory instance: $13.338/h

[1]https://aws.amazon.com/ec2/instance-types/

Amazon S3

S3 is a distributed file storage. In S3 lingo:

- buckets are used to store objects
- objects are uniquely identified by bucket, object key and version ID
- HTTP verbs are used for data modification and retrieval: GET, HEAD, PUTS, DELETES

S3 Features

Amazon Web Services promises that S3

- stores an infinite amount of data in a bucket
- scales past trillions of objects
- offers object versioning
- offers access management
- yields 99.999999999% durability

To achieve this scalability, S3 only offers eventual consistency. This means especially, that stale reads are possible.

How much data does Amazon store?

Not known, but let their offerings speak for themselves:

- AWS Snowmobile, a solution to transport 100 Petabyte
- https://aws.amazon.com/snowmobile/

How to experiment with AWS

Amazon offers a free tier with

1. 750 hours monthly for micro-instance usage
2. 5GB S3 storage with 20000 GET-requests and 2000-PUT requests
3. 750 hours of RDS (relational database service)

This gives you a nice opportunity to be master of your own compute cluster!
If you want more free resources, consider applying for aws educate.