Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing
2. Understanding MapReduce
   - the word count problem
   - more examples

MCS 572 Lecture 24
Introduction to Supercomputing
Jan Verschelde, 17 October 2016

Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing
2. Understanding MapReduce
   - the word count problem
   - more examples

extract meaning from the data avalanche

goal: applying complex analytics to large data sets

complementary technology is the use of cloud computing, in particular: Amazon Web Services.

Assumptions for writing MapReduce applications:

1. comfortable writing Java programs; and
2. familiar with the Unix command-line interface.

What is the value of data?

- Some questions are relevant only for large data sets.

  For example: movie preferences are inaccurate when based on just another person, but patterns can be extracted from the viewing history of millions.

- Big data tools enable processing on larger scale at lower cost.
- Additional hardware is needed to make up for latency.
- The notion of what is a database should be revisited.

Realize that one no longer needs to be among the largest corporations or government agencies to extract value from data.

data processing systems

Two ways to process large data sets:

1. *scale-up*: large and expensive computer (supercomputer).

   We move the same software onto larger and larger servers.

2. *scale-out*: spread processing onto more and more machines (commodity cluster).

Limiting factors:

- complexity of concurrency in multiple CPUs;
- CPUs are much faster than memory and hard disk speeds.

There is a third way:

3. *Cloud computing*: provider deals with scaling problems.

Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing
2. Understanding MapReduce
   - the word count problem
   - more examples

principles of Hadoop

1. *All roads lead to scale-out*, scale-up architectures are rarely used and scale-out is the standard in big data processing.
2. *Share nothing*: communication and dependencies are bottlenecks, individual components should be as independent as possible to allow to proceed regardless of whether others fail.
3. *Expect failure*: components will fail at inconvenient times.
   See our previous exercise on multi-component expected life span; resilient scale-up architectures require much effort.
4. *Smart software, dumb hardware*: push smarts in the software, responsible for allocating generic hardware.
5. *Move processing, not data*: perform processing locally on data.
   What gets moved through the network are program binaries and status reports, which are dwarfed in size by the actual data set.
6. *Build applications, not infrastructure.*
   Instead of placing focus on data movement and processing, work on job scheduling, error handling, and coordination.

History of Hadoop

Who to thank for Hadoop?

1. Google released two academic papers on their technology:
   in 2003: the google file system; and in 2004: MapReduce.
   The papers are available for download from google.research.com.
2. Doug Cutting started implementing Google systems,
   as a project within the Apache open source foundation.
3. Yahoo hired Doug Cutting in 2006 and supported Hadoop project.

From Tom White: *Hadoop: The Definitive Guide*, published by O'Reilly:

Definition (The name Hadoop is ...)

The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such.

components of Hadoop

Two main components of Hadoop are

1. The Hadoop Distributed File System (HDFS)
   stores very large data sets across a cluster of hosts,
   - optimized for throughput instead of latency,
   - achieving high availability through replication instead of redundancy.

2. MapReduce is a data processing paradigm that takes a
   specification of input (map) and output (reduce)
   and applies this to the data.

MapReduce integrates tightly with HDFS,
running directly on HDFS nodes.

common building blocks

- run on commodity clusters, scale-out: can add more servers
- have mechanisms for identifying and working around failures
- provides services transparently, user can concentrate on problem
- software cluster sitting on physical server controls execution

the Hadoop Distributed File System (HDFS)

HDFS spreads the storage across nodes.

Features:

- files stored in blocks of 64MB (typical file system: 4-32 KB)
- optimized for throughput over latency, efficient at streaming, but poor at seek requests for many smale ones
- optimized for workloads that are read-many and write-once
- storage node runs process DataNode that manages blocks, coordinated by master NameNode process running on separate host
- replicates blocks on multiple nodes to handle disk failures

MapReduce

A new technology build on fundamental concepts:

- functional programming languages offer map and reduce;
- divide and conquer breaks problem into multiple subtasks.

The input data goes to a series of transformations:

- the developer defines the data transformations,
- Hadoop's MapReduce job manages parallel execution.

What is the point?

Unlike relational databases the required structure data,
the data is provided as a series of key-value pairs
and the output of the map is another set of key-value pairs.

Most important point: *the Hadoop platform takes responsibility for every aspect of executing the processing across the data.*

The user is unaware of the actual size of data and cluster,
the platform determines how best to utilize all the hosts.

Challenge to the user: break the problem into

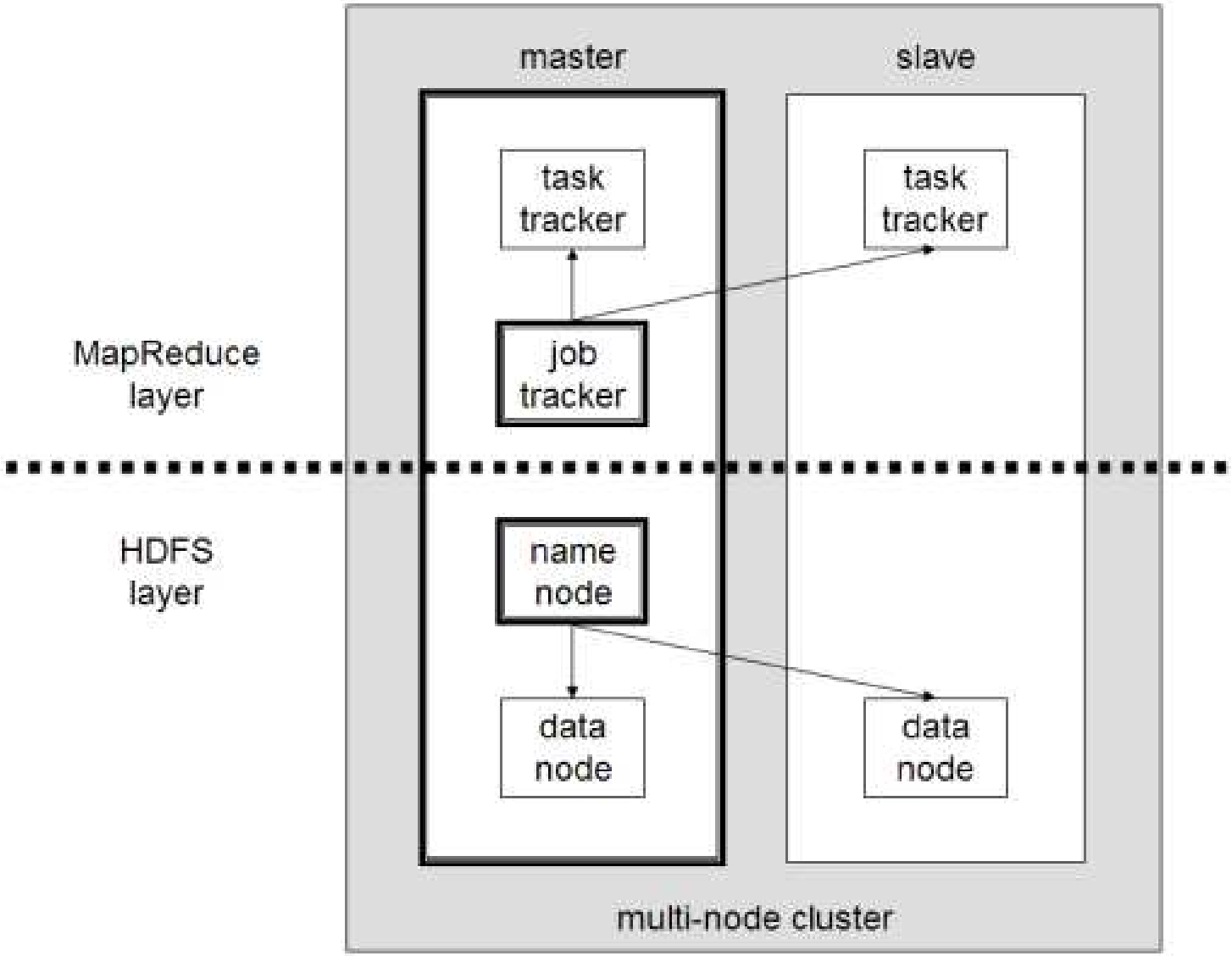
1. the best combination of chains of map and reduce functions, where the output of one is the input of the next; and
2. where each chain could be applied independently to each data element, allowing for parallelism.

software clusters

The common architecture of HDFS and MapReduce are software clusters:

- cluster of worker nodes managed by a coordinator node;
- master (NameNode for HDFS and JobTracker for MapReduce) monitors clusters and handles failures;
- processes on server (DataNode for HDFS and TaskTracker for MapReduce) perform work on physical host, receiving instructions for master and reporting status.

a multi-node Hadoop cluster



copied from wikipedia

what is it good for, and what not

Strengths and weaknesses:

- \+ Hadoop is flexible and scalable data processing platform;
- − batch processing not for real time, e.g.: serving web queries.

Application by Google:

1. Web crawler retrieves updated webpage data.
2. MapReduces processes the huge data set.
3. The web index is then used by a fleet of MySQL servers for search requests.

Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing

2. Understanding MapReduce
   - the word count problem
   - more examples

cloud computing

Cloud computing with Amazon Web Services is another technology.

Two main aspects:

1. new architecture option; and
2. different approach to cost.

Instead of running your own clusters, all you need is a credit card.

This is the third way:

1. scale-up: supercomputer
2. scale-out: commodity cluster
3. cloud computing: the provider deals with the scaling problem.

using cloud computing

How does it work? Two steps:

1. user develops software according to published guidelines or interface;
2. deploy onto the cloud platform and allow the service to scale on demand.

Service-on-demand aspect:

- start application on small hardware and scale up,
- off loading infrastructure costs to cloud provider.

Major benefit: great reduction of cost of entry for organization to launch an online service.

Amazon Web Services (AWS)

AWS is an infrastructure on demand

A set of cloud computing services:

- Elastic Compute Cloud (EC2): a server on demand.
- Simple Storage Service (S3): programmatic interface to store objects.
- Elastic MapReduce (EMR): Hadoop in the cloud.

In most impressive mode: EMR pulls data from S3, process on EC2.

Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing

2. Understanding MapReduce
   - the word count problem
   - more examples

key/value transformations

MapReduce as a series of key/value transformations

```
            map                          reduce
{K1, V1} --> {K2, List<V2>} ------> {K3, V3}
```

Two steps, three sets of key-value pairs:

1. The input to the map is a series of key-value pairs, called `K1` and `V1`.
2. The output of the map (and the input to the reduce) is a series of keys and an associated list of values that are called `K2` and `V2`.
3. The output of the reduce is another series of key-value pairs, called `K3` and `V3`.

Hadoop's hello world

The word count problem:

- Input: a text file.
- Output: the frequency of words.

Project Gutenberg (at `www.gutenberg.org`) offers many free books (mostly classics) in plain text file format.

Doing a word count is a very basic data analytic,
same authors will use the same patterns of words.

solving the word count problem with MapReduce

Every word on the text file is a key.

Its value is the count, its number of occurrences.

- Keys must be unique, but values need not be.
- Each value must be associate with a key,
  but a key could have no values

One must be careful when defining what is a key,
e.g. for the word problem do we distinguish between lower and upper case words?

a MapReduce job

A complete MapReduce Job for the word count problem:

1. Input to the map:

   `K1/V1` pairs are in the form
   `< line number, text on the line >`.

   The mapper takes the line and breaks it up into words.

2. Output of the map, input of reduce:

   `K2/V2` pairs are in the form `< word, 1 >`.

3. Output of reduce:

   `K3/V3` pairs are in the form `< word, count >`.

pseudo code for the word count problem

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
    for each v in values:
      result += ParseInt(v);
    Emit(AsString(result));
```

Introduction to Hadoop

1. What is Hadoop?
   - the big data revolution
   - extracting value from data
   - cloud computing

2. Understanding MapReduce
   - the word count problem
   - more examples

grep and counting URLs

Distributed Grep:

- The map function emits a line if it matches a supplied pattern.
- The reduce function is an identity function that just copies the supplied intermediate data to the output.

Count of URL Access Frequency:

- The map function processes logs of web page requests and outputs `(URL, 1)`.
- The reduce function adds together all values for the same URL and emits a `(URL, total count)` pair.

reverse web-link graph

Reverse Web-Link Graph:

- The map function outputs (`target`, `source`) pairs
  for each link to a target URL found in a page named `source`.
- The reduce function concatenates the list of all source URLs
  associated with a given target URL and emits the pair:
  (`target`, *list*(`source`)).

term-vector per host

Term-Vector per Host:

A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word, frequency) pairs.

- The map function emits a (`hostname`, `term vector`) pair for each input document (where the hostname is extracted from the URL of the document).
- The reduce function is passed all per-document term vectors for a given host.

  It adds these term vectors together, throwing away infrequent terms, and then emits a final (`hostname`, `term vector`) pair.

inverted index

Inverted Index:

- The map function parses each document, and emits a sequence of (`word`, `document ID`) pairs.
- The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (`word`, *list*(`document ID`)) pair.

The set of all output pairs forms a simple inverted index.

It is easy to augment this computation to keep track of word positions.

distributed sort

Distributed Sort:

- The map function extracts the key from each record, and emits a `key`, `record`) pair.
- The reduce function emits all pairs unchanged.

This computation depends

- on the partitioning facilities: hashing keys; and
- and the ordering properties: within a partition key-value pairs are processed in increasing key order.

suggested reading

- Garry Turkington. Hadoop Beginner's Guide. `www.it-ebooks.info`.
- Papers available from `research.google.com/archive` (URL):
  - Luiz Barroso, Jeffrey Dean, and Urs Hoelzle: **Web Search for a Planet: The Google Cluster Architecture.** At URL/`googlecluster.html`.
  - Jeffrey Dean and Sanjay Ghemawat: **MapReduce: Simplified Data Processing on Large Clusters.** At URL/`mapreduce.html`. Appeared in OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.
  - Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: **The Google File System**. At URL/`gfs.html`. Appeared in 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

Summary and Exercises

Hadoop and Cloud Computing reduce the entry cost for large scale data processing.

Exercises:

1. Install Hadoop on your computer and run the word count problem.
2. Consider the pancake problem we have seen in Lecture 23 and formulate a solution within the MapReduce programming model.

Applying Hadoop to a nontrivial problem could be a topic for an interesting final project.