Big Data processing with Hadoop

Luca Pireddu

CRS4—Distributed Computing Group

April 18, 2012

Outline

Motivation

Data set sizes are growing. But why?

Big Data

Data set sizes are growing. But why?

Incentive:

- Larger sizes tend to improve the sensitivity of analyses

Ability:

- More easily accessible sources of data
  - e.g., Internet, Twitter firehose
- Technology enables more ambitious science
  - e.g., LHC, whole-genome sequencing
- Cheaper and faster acquisition/tracking methods
  - e.g., cell phones, RFID tags, customer cards at the stores

Big Data computational challenges

- Data sets can grow so big that it is difficult or impossible to handle them with conventional methods
  - Too big to load into memory
  - Too big to store on your desktop workstation
  - Too long to compute with a single CPU
  - Too long to read from a single disk

Problems that require the analysis of such data sets have taken the name of *"Big Data" problems*

Big Data parallelization

- Many big data problems are loosely-coupled and are easily parallelized
- They may require high I/O throughput as large quantities of data is read/written
- Do not require real-time communication between batch jobs
- How should we parallelize them?

Poor man's parallel processing

Poor man's parallel processing

- manual data splitting into batches
- ad hoc scripting to automate, at least partially
- queueing system to distribute jobs to multiple machines
- shared storage to pass intermediate data sets

Poor man's parallel processing

Presents many weaknesses

- High effort, low code re-use
- No robustness to equipment failure
- Failures typically require human intervention to recover
  - raises operator effort and therefore operating costs
- Usually less-than-desirable parallelism
  - Getting high-parallelism (especially more than per-file) can get complicated
- I/O done to/from shared storage
  - Limits scalability in number of nodes
  - Storage can become the bottleneck; alternatively, storage becomes very expensive
  - High network use as data is typically read and written remotely
  - Raises infrastructure costs

MapReduce and Hadoop

MapReduce

- A programming model for large-scale distributed data processing
- Aims to solve many of the issues just mentioned

- Breaks algorithms into two steps:
  1. *Map*: map a set of input key/value pairs to a set of intermediate key/value pairs
  2. *Reduce*: apply a function to all values associated to the same intermediate key; emit output key/value pairs
- Functions don't have side effects; (k,v) pairs are the only input/output
- Functions don't share data structures

Consider a program to calculate word frequency in a document.

The quick brown fox ate the lazy green fox.

Word	Count
ate	1
brown	1
fox	2
green	1
lazy	1
quick	1
the	2

A possible MapReduce algorithm:

Map

- Input: part of text
- For each word write a tuple *(word, 1)*

Reduce

- Input: word *w*, list of 1's emitted for *w*
- Sum all 1's into count
- Write tuple *(word, count)*

MapReduce Example – Word Count

The quick brown fox ate the lazy green fox.

Here's some pseudo code for a MapReduce word counting algorithm:

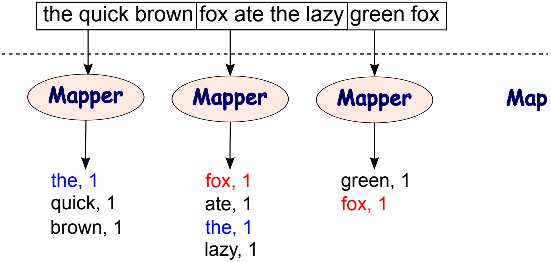
Map

```
map(key, value):
  foreach word in value:
    emit(word, 1)
```

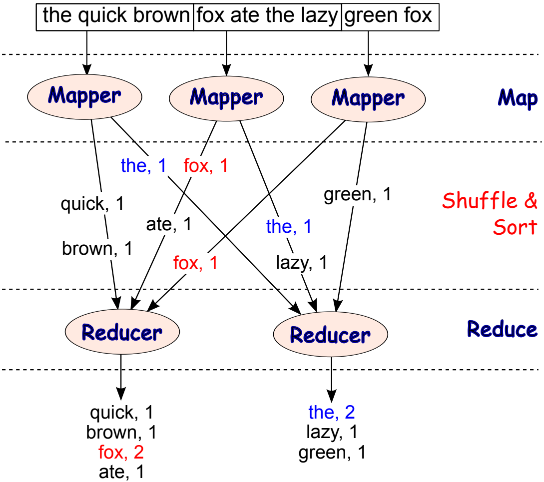
Reduce

```
reduce(key, value_list):
  int wordcount = 0
  foreach count in value_list:
    wordcount += count
  emit(key, wordcount)
```

MapReduce Example – Word Count



MapReduce Example – Word Count



MapReduce

The lack of side effects and shared data structures is the key.

- No multi-threaded programming
- No synchronization, locks, mutexes, deadlocks, etc.
- No shared data implies no central bottleneck.
- Failed functions can be retried—their output only being committed upon successful completion.

MapReduce allows you to put much of the parallel programming into a reusable framework, outside of the application.

Hadoop MapReduce

- The MapReduce model needs an implementation
- Hadoop is arguably the most popular open-source MapReduce implementation
- Born out of Yahoo! Currently used by many very large operations

Hadoop DFS

A MapReduce framework goes hand-in-hand with a distributed file system

- Multiplying the number of nodes poses challenges
    - multiplied network traffic
    - multiplied disk accesses
    - multiplied failure rates

Hadoop DFS

A MapReduce framework goes hand-in-hand with a distributed file system

- Multiplying the number of nodes poses challenges
  - multiplied network traffic
  - multiplied disk accesses
  - multiplied failure rates

Hadoop provides the Hadoop Distributed File System (HDFS)

- Stores blocks of the data on each node.
  - Move computation to the data and decentralize data access
- Uses the disks on each node
  - Aggregate I/O throughput scales with the number of nodes
- Replicates data on multiple nodes
  - Resistance to node failure

Components

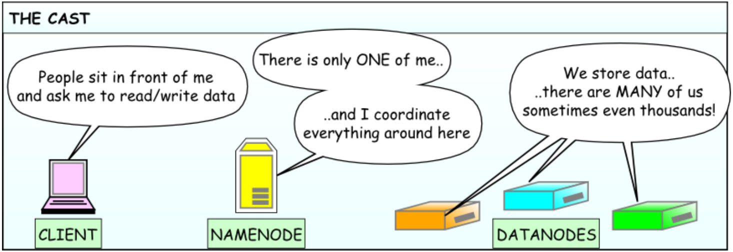
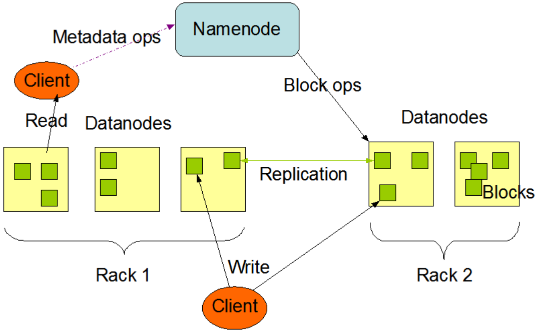


Image courtesy of Maneesh Varshney

Hadoop DFS: Architecture

- Files split into large blocks (e.g., 64 MB)
- Namenode maintains file system metadata
  - Directory structure
  - File names
  - The ids of the blocks that compose the files
  - The locations of those blocks
  - The list of data nodes
- Datanode stores, serves and deletes blocks

Hadoop DFS: Architecture



Cloud resources

- So, you have a big data problem
- You've written the next great MapReduce application
- You need a few hundred machines to run it... now what?

- So, you have a big data problem
- You've written the next great MapReduce application
- You need a few hundred machines to run it... now what?

Rent them!

IaaS

- Lately there's been a growth of Infrastructure as a Service (IaaS)
- Rent infrastructure from companies that specialize in providing and maintaining them
  - e.g., Amazon Web Services (AWS), IBM
- You can rent as many nodes as you need for as long as you need
  - even as little as one hour
  - pay as you go
  - elastic
- Makes sense in many cases
  - peaky loads or temporary requirements—i.e., low average use
  - need to quickly grow capacity
  - don't want to create an HPC group within the company

How popular is IaaS?

- In April 2011 Amazon suffered a major service outage

www.pcworld.com/article/225877/amazon_outage_crashes_reddit_quo

Amazon Outage Crashes Reddit, Quora, and Other Websites

By David Daw, PCWorld Apr 21, 2011 11:05 AM

A disruption at one of Amazon's datacenters has led to service disruptions in the company's EC2 or Elastic Cloud computing service. The outage, which started at Starting at 1:41 a.m. PDT, in turn brought down major websites such as Reddit and Foursquare.

Amazon's EC2 service allows business to rent processing time on Amazon's computers, which is useful for sites that need a lot of processing power. The disruption seriously affected sites that are constantly working through a lot of data. Dynamic sites like Hootsuite and Quora that are constantly processing new data are down completely. Other sites remain up but are still affected; Reddit, for instance, is operating in read-only mode as it cannot process new stories or votes.

Amazon hasn't responded to our requests for more information but the company's status dashboard has provided periodic updates throughout the morning. As of 10:50 a.m. PDT, the service disruptions are continuing and there is not yet an estimate for when service will be restored. An update provided to NetworkWorld at 8:54 AM PDT states that the issue had to do with a re-mirroring of EBS volumes at the datacenter.

Simplified Hadoop

Scripting with Hadoop

- At CRS4 we've written a Python API for Hadoop called **Pydoop**
- Allows one to access most of Hadoop's features with the simplicity of Python
- Lets you bridge Hadoop and C code
- Lets you script!!

Pydoop script to turn text to lower case in Hadoop

```python
def mapper(k,value, writer):
    writer.emit("", value.lower())
```

Scripting with Hadoop

- simple text-processing jobs reduced to two Python functions in a module
- makes it easy to solve simple problems
- makes it feasible to write simple (even throw-away) *parallel* programs

Pydoop wordcount script

```
def mapper(k, text, writer):
  for word in text.split():
    writer.emit(word, 1)

def reducer(word, count, writer):
  writer.emit(word, sum(map(int, count)))
```

- Then run it with:
  `pydoop_script wordcount.py hdfs_input hdfs_output`

Pydoop available on entu/oghe

- If you're interested, Pydoop is available on entu/oghe
- Use the python installation on els5
- At the next release we'll ask our kind administrators to install it centrally

Other high-level tools

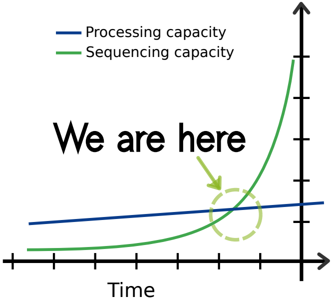
Other high-level Hadoop-based tools exist as well. E.g.,

- Pig
- Hive
- Cascading
- Cascalog
- Scalding
- Scrunch
- Spark

Sample Hadoop use case: high throughput sequencing

Genomics data growth

- Trend in sequencing technologies:
  - Lower costs
  - Increasing speed
  - Higher resolution
- Sequencing rate is growing exponentially
- Processing capacity is *not*



HT Sequencing at CRS4

CRS4 Sequencing and Genotyping Platform

- Currently the largest sequencing center in Italy
- Created to enable a number of studies on the Sardinian population

Equipment: 4 HiSeq2000 and 2 GAIIx Illuminas

Capacity: about 7000 Gbases/month

As its sequencing capacity grew, the operation faced scalability problems in its processing pipeline.

- Used "traditional" programs (some multi-threaded, not distributed)
- Data shared exclusively through Lustre volume
- Based on the "poor man's parallelism", with the consequential shortcomings

Seal

To solve those problems we began working on *Seal*

Seal

To solve those problems we began working on *Seal*

Seal is:

- a suite of distributed tools for processing HT sequencing data
- based on a proven technology: the Hadoop MapReduce framework
- used in production at the CRS4 Sequencing Center
- Released under GPLv3 license
- Web site: `http://biodoop-seal.sf.net`

Key goals

Scalable	• In cluster size • In data size
Robust	• Resilient to node failure and transient cluster problems

Seal: Features

Currently featured tools

Seal currently has tools to perform distributed:

- read demultiplexing (Demux)
- read alignment (Seqal)
- sorting of read alignments (ReadSort)
- compute base quality statistics (RecabTable).

- These tools are implemented as MapReduce programs that run on Hadoop

Seal: Features

Important features:

- distributed
- scalable
- robust
- open source

- Part of these benefits are a direct consequence of Seal being based on Hadoop
- Others are thanks to implementation details
  - algorithm design
  - shared (read-only) memory
  - etc.

Evaluation

Important criteria

- throughput per node
- efficiency of the distribution mechanism
- scalability w.r.t. nodes and data size

Evaluation steps:

1. Establish a single-node baseline throughput measure
2. Compare throughput/node of baseline, old CSGP workflow and Seal equivalent
3. Compare wall-clock runtimes
4. Evaluate scalability characteristics

Baseline

Reflects what can be easily achieved on a workstation with no programming effort.

- Use multi-threaded programs where available
- 8-disk GPFS volume used for storage

Data sets

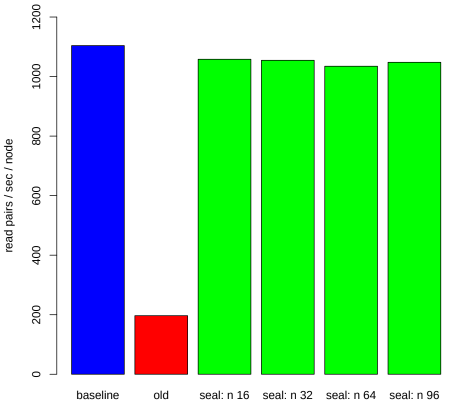
Baseline input data sets

Dataset	No. tiles	No. pairs	Size (GB)
Dataset B3	10	$3.6 \cdot 10^7$	15.7

Realistic data sets

Dataset	No. lanes	No. pairs	Size (GB)
Dataset MR1	1	$1.2 \cdot 10^8$	51
Dataset MR3	3	$3.3 \cdot 10^8$	147
Dataset MR8	8	$9.2 \cdot 10^8$	406

Throughput per Node



- Baseline: B3 dataset
- Old CSGP: MR3 (16 nodes)
- Seal: MR3

- Nodes used efficiently (mainly because of improved parallelism)
- The overhead payed by Seal for distributing the work is minimal

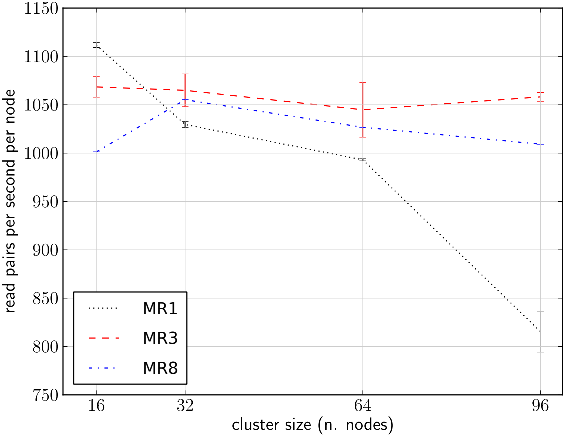
Runtime

Scenario	No. nodes	Runtime (h)
Old CSGP	16	29.1
Seal	16	5.4
Seal	32	2.7
Seal	64	1.4
Seal	96	0.9

Table: Wall clock times, Dataset MR3.

- Significant speed-up over old workflow on same number of nodes (16)
- Evident linear scalability

Scalability



- Ideal system would produce a flat line
- 1-lane case starves at more than 16 nodes

Publications

Related publications:

- S. Leo and G. Zanetti. Pydoop: a Python MapReduce and HDFS API for Hadoop. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pages 819–825, 2010.
- L. Pireddu, S. Leo, and G. Zanetti. MapReducing a genomic sequencing workflow. In Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing, pages 67–74, June 2011.
- Pireddu,L., Leo,S. and Zanetti,G. (2011). SEAL: a Distributed Short Read Mapping and Duplicate Removal Tool. Bioinformatics.

- Various posters. . .

In addition, we've been invited to the SeqAhead Next Generation Sequencing Data Analysis Network.

Conclusion

MapReduce

- Is often a good solution for Big Data problems that present loosely coupled parallelism
- Especially true for I/O-bound problems
- Simplifies development of parallel programs
    - Especially true when using Pydoop
- Successfully used in Seal and many companies
- The robustness added of the system is essential for automation
- Automation is very important for scaling sample throughput and maximizing the R.O.I. in any large-scale operation.

MapReduce

- Is often a good solution for Big Data problems that present loosely coupled parallelism
- Especially true for I/O-bound problems
- Simplifies development of parallel programs
  - Especially true when using Pydoop
- Successfully used in Seal and many companies
- The robustness added of the system is essential for automation
- Automation is very important for scaling sample throughput and maximizing the R.O.I. in any large-scale operation.

Questions?