Principles of Computer Science II

Large Scale Computation

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 17

Problem: Lots of data

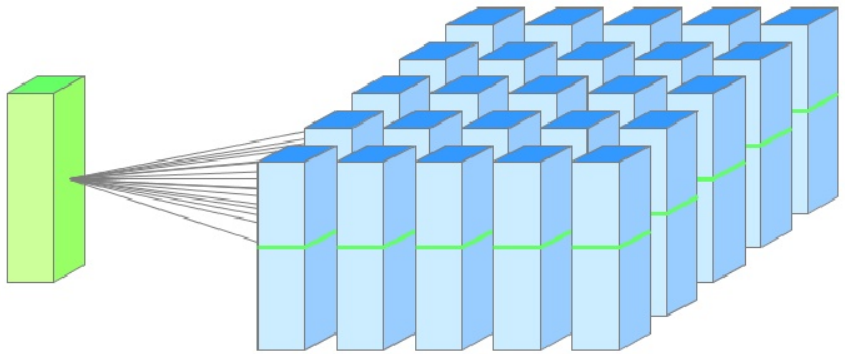
- Example: Homo sapiens high coverage assembly GRCh37
  - 27478 contigs
  - contig length total 3.2 Gb.
  - chromosome length total 3.1 Gb.
- One computer can read 30-35MB/sec from disc
  - $\sim$ 10 months to read the data
- $\sim$ 100 hard drives just to store the data in compressed format
- Even more to do something with the data.

Spread the work over many machines

- Good news: same problem with 1000 machines: $\leq$ 1 hour
- Bad news: concurrency
  - communication and coordination
  - recovering from machine failure
  - status reporting
  - debugging
  - optimization
- Bad news 2: repeat for every problem you want to solve

Computing Clusters

- Many racks of computers
- Thousands of machines per cluster
- Limited bandwidth between racks



Computing Environment

- Each machine has 2-4 CPUs
  - Typically quad-core
  - Future machines will have more cores
- 1-6 locally-attached disks
  - $\sim$ 10TB of disk
- Overall performance more important than peak performance of single machines
- Reliability
  - In 1 server environment, it may stay up for three years (1000 days)
  - If you have 10000 servers, expect to lose 10 each day
- Ultra reliable hardware still fails
  - We need to keep in mind cost of each machine

Map Reduce Computing Paradigm

- A simple programming model
  - Applies to large-scale computing problems
- Hides difficulties of concurrency
  - automatic parallelization
  - load balancing
  - network and disk transfer optimization
  - handling of machine failures
  - robustness
  - improvements to core libraries benefit all users of library

A typical problem

- Read a lot of data
- Map: extract something important from each record
- Shuffle and sort
- Reduce: aggregate, summarize, filter or transform
- Write the results

In more details

- Programmer specifies two primary methods:
  - map$(k, v) \rightarrow < k', v' > *$
    - Takes a key-value pair and outputs a set of key-value pairs
    - There is one Map call for every (k, v) pair
  - reduce$(k', < v' > *) \rightarrow < k', v' > *$
    - All values v with same key k are reduced together and processed in v order
    - There is one Reduce function call per unique key k
- All v' with same k' are reduced together, in order.

An example: Frequencies in DNA sequence

A typical exercise for a new engineer in his/her first week:

- Input files with one document per record
- Specify a map function that takes a key/value pair
  - key = document URL
  - value = document contents
- Output of map function is (potentially many) key/value pairs.
- In this case, output:
  (word, 1) once per word in the document

"document 1", "CTGGGCTAA"
converted to
(C, 1), (T, 1), (G, 1), ...

An example: Frequencies in DNA sequence

- MapReduce library gathers together all pairs with the same key (shuffle/sort)
- The reduce function combines the values for a key
- In this example:

key = "A"	key = "G"	key = "C"	key = "T"
values = 1, 1	values = 1, 1, 1	values = 1, 1	values = 1, 1
summarize	summarize	summarize	summarize
2	3	2	2

- Output of reduce paired with key and saved

(A, 3), (G, 3), (C, 2), (T, 2)

An example: Frequencies in DNA sequence

- Python threads are defined by a class

```
s = 'CTGGGCTAA'
seq = list(s)  # ['C', 'T', 'G', 'G', 'G', 'C', 'T', '
    A', 'A']
sc.parallelize(seq)\
  .map(lambda symbol: (symbol, 1))\
  .reduceByKey(add)\
  .collect()
```

- Output

```
[('A', 2), ('C', 2), ('G', 3), ('T', 2)]
```

Fault tolerance: handled via re-execution

- On worker failure:
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress map tasks
  - Re-execute in progress reduce tasks
  - Task completion committed through master
- On master failure:
  - Restart execution

Apache Spark

- Download latest version:
  `http://spark.apache.org/downloads.html`
- Install with PySpark

```
pip install pyspark
```

- Download dataset
  `http://hplgit.github.io/bioinf-py/data/yeast_chr1.txt`

Apache Spark

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("Frequencies").setMaster("
    local")
sc = SparkContext(conf=conf)

raw_data = sc.textFile("/home/ichatz/Local/psc2/lec17/
    yeast_chr1.txt")
print(raw_data.take(5))
print(raw_data.count())
print(raw_data.first())
```

Apache Spark

```
def splitLine(line):
  pairs = []
  symbols = list(line)
  if len(symbols) > 1:
    for symbol in symbols:
      pairs.append([symbol, 1])

  return pairs

pairs = raw_data.flatMap(splitLine)
print(pairs.take(10))

final = pairs.reduceByKey(lambda a,b: a + b)
print(final.collect())
```