MapReduce (in the cloud)

How to painlessly process terabytes of data

by Irina Gordei
MapReduce Presentation Outline

- What is MapReduce?
- Example
- How it works
- MapReduce in the cloud
- Conclusion
- Demo
**Motivation: Large Scale Data Processing**

- Process many terabytes of raw data
  - documents found by a web crawl
  - web requests log

- Produce various kinds of derived data
  - inverted indices
  - various representations of graph structure of documents
  - summaries of number of pages crawled per host
  - most frequent queries in a given day

- Problem characteristics
  - Input data is large
  - Need to parallelize computations (with thousands of CPUs)
  - Need to distribute the data
  - How to handle failures?
**Solution: MapReduce Programming Model**

- Restricted parallel programming model meant for large clusters
  - User implements Map() and Reduce()
- Parallel computing framework
  - Libraries take care of EVERYTHING else
    - Parallelization
    - Fault Tolerance
    - Data Distribution
    - Load Balancing
- Useful model for many practical tasks
Map and Reduce

- Functions borrowed from functional programming languages (eg. Lisp)

- Map()
  - Process a key/value pair to generate intermediate key/value pairs

- Reduce()
  - Merge all intermediate values associated with the same key

- Types
  - map  \((k1, v1) \rightarrow \text{list}(k2, v2)\)
  - reduce  \((k2, \text{list}(v2)) \rightarrow \text{list}(v2)\)
Example: Counting Words

- Map()
  - Input <filename, filetext>
  - Parses file and emits <word, count> pairs
    - eg. <"hello", 1>

- Reduce()
  - Sums all values for the same key and emits <word, totalcount>
    - eg. <"hello", (3 5 2 7)> => <"hello", 17>
Example Use of MapReduce

- Counting words in a large set of documents

```java
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: word
  //values: list of counts
  int results = 0;
  for each v in values
    result += ParseInt(v);
  Emit(AsString(result));
```
How MapReduce works

- User’s To Do List:
  1. Indicate
     - In/Out-files
     - M: number of map tasks
     - R: number of reduce tasks
     - W: number of machines
  2. Write map and reduce functions
  3. Submit the job

- This requires NO knowledge of parallel/distributed systems!!
- What about everything else?
**Implementation: Google Computing Environment**

= *large clusters of commodity PCs connected together with switched Ethernet*

- A typical cluster contains thousands of machines
- Machines are dual-processor x86's running Linux with 2-4GB memory/machine
- **Commodity networking**
  - Typically 100 Mbs or 1 Gbs/sec
- IDE disks connected to individual machines
  - Distributed file system (GDFS)
Figure 1: Execution overview
Fault Tolerance: Failures

- Worker fails
  - master pings workers periodically
  - if no response after a while, master marks worker as failed
  - a map task for a failed worker is reset to idle state; might be rescheduled

- Master fails
  - writes periodic checkpoints of master data structures
  - if master dies, another could be recreated from the check-pointed copies of its state
Fault Tolerance

- **Locality**
  - Network bandwidth is a scarce commodity
  - Input file blocks stored on multiple machines (3 copies)
  - Master attempts to assign worker onto a machine that contains a replica of input data or near replica (i.e., on machine attached to the same network switch)

- **Backup Tasks (coping with “stragglers”)**
  - When computation almost done, reschedule in-progress tasks (significantly reduces time for large MapReduce computations)
Refinements

- Partitioning function
  - typically $\text{hash}(key) \mod R$ to give balanced partitions
    - eg. scenario: word counting on 10 documents with 5 map and 2 reduce functions

- Combiner function
  - when there is significant repetition in intermediate keys
    - eg. $<$the, 1$>$

- Ordering
  - within a given partition, all intermediate values processed in order (simplifies creating sorted output)
MapReduce in the cloud

- Open Source
  - Apache Hadoop
  
  - Cloud MapReduce (built on top of the Amazon cloud OS using cloud services such as S3/SQS/SimpleDB)

- Amazon Elastic MapReduce
Conclusions

- Simplifies large-scale computations that fit this model

- Allows user to focus on the problem without worrying about details

- Computer architecture not very important
  - Portable model
References

- Hadoop MapReduce Tutorial:  
  http://hadoop.apache.org/comon/docs/current/mapred_tutorial.html
- J. Mellor-Crummey. *Data Analysis with MapReduce:*
  
  http://www.owlnet.rice.edu/~comp422/lecture-notes/comp422-2010-Lecture24-MapReduce.pdf