CHRYSAOR: FINE-GRAINED, FAULT-TOLERANT MAPREDUCE

Pedro Sá da Costa
Fernando M. V. Ramos, Miguel Correia
Motivation

- MapReduce is a tool for **processing massive datasets**
Motivation

- MapReduce is a tool for **processing massive datasets**
- Programming framework used by Google, or Yahoo (most likely here too)
Motivation

• MapReduce is a tool for **processing massive datasets**
• Programming framework used by Google, or Yahoo (most likely here too)
• Targeted a **single data center** and tolerates **crash faults**
Motivation

- MapReduce is a tool for **processing massive datasets**
- Programming framework used by Google, or Yahoo (most likely here too)
- Targeted a **single data center** and tolerates **crash faults**
- The **unprecedented data growth** brought the need to **scale-out computation** across clouds
Motivation

Google: DRAM error rates vastly higher than previously thought

PCs will likely require error correction code in the future due to DRAM issues

A study released this week by Google Inc. and the University of Toronto showed that data error rates on dynamic RAM memory modules are vastly higher than previously thought and may be more responsible for system shutdowns and service interruptions.

MORE LIKE THIS

Tips for using RAS features for storage systems

Chapter 1: Network Overview

Report: Intel’s Braidwood flash memory module could kill SSD market
GCreep: Google Engineer Stalked Teens, Spied on Chats (Updated)

Adrian Chen
09/14/10 02:23PM Filed to: EXCLUSIVE

We entrust Google with our most private communications because we assume the company takes every precaution to safeguard our data. It doesn’t. A Google engineer spied on four underage teens for months before the company was notified of the abuses.

David Barksdale, a 27-year-old former Google engineer, repeatedly took advantage...
Motivation

Internet outage disrupts Twitter and Spotify, reportedly due to cyber attack

LAUREN HOLTER  @laurenholter  10.21.16  10:34 am
Motivation

What are the advantages of using multiple clouds?

• It increases resilience by avoiding single points of failure.
Motivation

What are the advantages of using multiple clouds?

• It increases resilience by avoiding single points of failure.
• It can improve performance (data locality, computing and network diversity)
Motivation

What are the advantages of using multiple clouds?

• It increases resilience by avoiding single points of failure.
• It can improve performance (data locality, computing and network diversity)
• It can improve security, for instance by exploring the interaction between public and private clouds.
Motivation

What are the advantages of using multiple clouds?

• It *increases resilience* by avoiding single points of failure.
• It can *improve performance* (data locality, computing and network diversity)
• It can *improve security*, for instance by exploring the interaction between public and private clouds.
• It may help in *reducing costs*
Outline

- MapReduce
- Contribution
- MapReduce Execution
  - MapReduce Job
  - Logical MapReduce Jobs
  - WordCount Example
- Evaluation
  - Experimental Setup
  - Experimental Results
- Conclusions
Hadoop MapReduce

Input

Splitting

Map

Shuffle & Sort

Reduce

Output

Deer Beer River
Car Car River
Deer Beer River

Deer Beer River
Car Car River
Deer Beer River

Deer, 1
Beer, 1

Car, 1

Car, 1

Deer, 1

Beer, 1

Beer, 1

Car, 1

Car, 1

Deer, 1

River, 1

River, 1

Beer, 2

Car, 3

Car, 3

Deer, 2

Deer, 2

River, 2

Beer, 2

Car, 3

Deer, 2

River, 2

Job execution

Map task

Reduce task
Hadoop MapReduce

Input
- Deer Beer River
- Car Car River
- Deer Beer River

Splitting
- Deer Beer River
- Car Car River
- Deer Beer River

Map
- Dear, 1 Beer, 1 River, 1
- Car, 1 Car, 1
- Deer, 1 Deer, 1
- River, 1 River, 1

Shuffle & Sort
- Beer, 1 Beer, 1
- Car, 1 Car, 1
- Deer, 1 Deer, 1
- River, 2 River, 1

Reduce
- Beer, 2
- Car, 3
- Deer, 2
- River, 2

Output
- Deer, 2
- Car, 3
- Deer, 2
- River, 2

Job execution

Map task
Reduce task
Hadoop MapReduce

Input

Splitting

Map

Shuffle & Sort

Reduce

Output

Job execution

Map task

Reduce task
Hadoop MapReduce

Input

Splitting

Map

Shuffle & Sort

Reduce

Output

Deer Beer River
Car Car River
Deer Beer River

Deer Beer River

Car Car River

Deer Beer River

Dear, 1
Beer, 1
River, 1

Car, 1
Car, 1
Car, 1

Dear, 1
Beer, 1
River, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Beer, 1
Beer, 1

Beer, 2

Car, 3

Beer, 2
Car, 3
Deer, 2
River, 2

Deer, 2

River, 2

Job execution

Map task
Reduce task
Hadoop MapReduce

Input → Splitting → Map → Shuffle & Sort → Reduce → Output

Job execution

Map task → Reduce task
Hadoop MapReduce

Input

Splitting

Map

Shuffle & Sort

Reduce

Output

Job execution

Map task

Reduce task
Hadoop MapReduce

Cloud A
Hadoop MapReduce

- Tolerates **crash faults** by re-launching tasks
Hadoop MapReduce

- Tolerates **crash faults** by re-launching tasks
- Add **checksums to detect data corruptions**
Hadoop MapReduce

- Tolerates **crash faults** by re-launching tasks
- Add **checksums to detect data corruptions**
- Vulnerable to **cloud outage, arbitrary and malicious faults**
Outline

- MapReduce
- Contribution
- MapReduce Execution
  - MapReduce Job
  - Logical MapReduce Jobs
  - WordCount Example
- Evaluation
  - Experimental Setup
  - Experimental Results
- Conclusions
Contribution

- Execute **replicated jobs** in different clouds
Contribution

• Execute **replicated jobs** in different clouds
• Tolerate **faults at the task level** and relaunch the faulty tasks.
Contribution

- Execute **replicated jobs** in different clouds
- Tolerate **faults at the task level** and relaunch the faulty tasks.
- Use **digests in the tasks** to validate computation
Contribution

What do we want?
• A system that tolerates the above-mentioned classes of faults at reasonable cost
Contribution

What do we want?
• A system that \textit{tolerates the above-mentioned classes} of faults at reasonable cost
• That requires \textit{minimal modifications} to the users' applications
Contribution

What do we want?
• A system that tolerates the above-mentioned classes of faults at reasonable cost
• That requires minimal modifications to the users' applications
• and it does not change Hadoop source code
Outline

• Contribution
• **MapReduce Execution**
  • MapReduce Job
  • Logical MapReduce Jobs
  • WordCount Example
• Evaluation
  • Experimental Setup
  • Experimental Results
• Conclusions
Logical Job

- Cannot **pause** a job with MapReduce API
Logical Job

- Cannot **pause** a job with MapReduce API
- One logical job will be **dedicated to the map tasks**, and the other to the **reduce tasks**
Logical Job

- Cannot **pause** a job with MapReduce API
- One logical job will be **dedicated to the map tasks**, and the other to the **reduce tasks**
- Create the concept of **Identity map tasks**
Logical Job

• Cannot **pause** a job with MapReduce API
• One logical job will be **dedicated to the map tasks**, and the other to the **reduce tasks**
• Create the concept of **Identity map tasks**
• **Hadoop viewpoint**, each logical job is a complete MapReduce job
Cannot **pause** a job with MapReduce API

One logical job will be **dedicated to the map tasks**, and the other to the **reduce tasks**

Create the concept of **Identity map tasks**

**Hadoop viewpoint**, each logical job is a complete MapReduce job.

**Chrysaor viewpoint**, one logical job corresponds to the map tasks of the Chrysaor job, and the other to the reduce tasks
Logical Job

Input

- Deer Beer River
- Car Car River
- Deer Beer River

Splitting

- Deer Beer River
- Car Car River
- Deer Beer River

Map

- Deer, 1
- Beer, 1
- River, 1
- Car, 1
- Beer, 1
- River, 1
- Deer, 1
- Beer, 1
- River, 1

1st Job execution

Map task
Reduce task
Logical Job

Identity Map

Shuffle & Sort

Reduce

Output

2nd Job execution

Map task

Reduce task
Logical Job (Bird’s Eye View)

Actions:
1. launch 1st logical job
2. fetch digests from map outputs
3. vote map output
4. launch 2nd logical job
5. fetch digests from reduce outputs
6. vote reduce output

Chrysaor

Cloud A

Cloud B

Input data
Output data
Execution of the job
Logical Job (With Malicious Faults)

Actions:
1. launch 1st logical job
2. fetch digests from map outputs
3. vote map output
4. relaunch 1st logical job
5. fetch digests from map output
6. launch 2nd logical job
7. fetch digests from reduce outputs
8. vote reduce output

Legend:
- Input data
- Output data
- Wrong data
- Wrong output data

Graph:
- Execution of the job
- Re-execution of the job
public static void main(String[] args) throws Exception {
    // first map tasks
    JobConf conf = new JobConf(MyWordCount.class);
    conf.setJobName("wordcount");
    (...)
    job.setMapperClass(MyMap.class);
    job.setReducerClass(MyReducer.class);
    conf.setClass("mapreduce.job.map.identity.class",
                  MyFullyIndentityMapper.class, Mapper.class);
    (...)
    JobClient.runJob(job);
}
public static class MyFullyIdentityMapper extends Mapper<Object, Text, Text, IntWritable>{
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()){
            word.set(itr.nextToken());
        }
    }
}
Experimental Setup

- Evaluate the performance using Hadoop Gridmix Benchmark
Experimental Setup

- Evaluate the performance using Hadoop Gridmix Benchmark
- Amazon EC2 testbed
Experimental Setup

- Evaluate the performance using Hadoop Gridmix Benchmark
- Amazon EC2 testbed
- Setup 3 clouds located in different sites
Experimental Setup

• Evaluate the performance using Hadoop Gridmix Benchmark
• Amazon EC2 testbed
• Setup 3 clouds located in different sites
• Each cloud is composed by 1 Resource Manager, and 4 Node managers
Experimental Setup

- Evaluate the performance using Hadoop Gridmix Benchmark
- Amazon EC2 testbed
- Setup 3 clouds located in different sites
- Each cloud is composed by 1 Resource Manager, and 4 Node managers
- $f=1$
WordCount Performance

The diagram shows the execution performance of Medusa and Chrysaor against different input data sizes (in MB). The y-axis represents execution time in seconds, ranging from 0 to 1000. The x-axis represents the input data size in MB, with values at 1000, 2000, 4000, and 8000 MB. The performance trend indicates that both tools scale differently with increasing input size, with Chrysaor showing a notably higher execution time compared to Medusa.
WordCount Performance

Execution Performance (sec.)

5%-27% slower

Input data size (MB)
WordCount Performance

![Graph showing execution performance with varying input data sizes. The graph compares Medusa with arbitrary faults, Chrysaor with arbitrary faults map side, and Chrysaor with arbitrary faults reduce side. The graph indicates a 41% faster performance for Chrysaor with arbitrary faults reduce side.](image-url)
WordCount Performance

Execution Performance (sec.)

- Medusa with arbitrary faults
- Chrysaor with arbitrary faults map side
- Chrysaor with arbitrary faults reduce side

23% slower
WordCount Performance

The diagram shows the execution performance of Medusa and Chrysaor with and without malicious faults for different input data sizes (MB). The graph indicates that Medusa with malicious faults is 32% faster compared to Chrysaor with malicious faults, especially in the reduce side.
WordCount Performance

41% slower
Sort Performance

16% faster
Sort Performance

64% faster
Sort Performance

22% faster
Conclusion

- Hadoop fault tolerance mechanism cannot deal with arbitrary faults
Conclusion

• Hadoop fault tolerance mechanism cannot deal with arbitrary faults
• Chrysaor application to scale out MapReduce applications
Conclusion

- Hadoop fault tolerance mechanism cannot deal with arbitrary faults
- Medusa application to scale out MapReduce applications
- It achieves great level of fault tolerance at a reasonable cost