Datacenter Simulation Methodologies: Spark

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi and Benjamin C. Lee

This work is supported by NSF grants CCF-1149252, CCF-1337215, and STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA.
<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 - 09:15</td>
<td>Introduction</td>
</tr>
<tr>
<td>09:15 - 10:30</td>
<td>Setting up MARSSx86 and DRAMSim2</td>
</tr>
<tr>
<td>10:30 - 11:00</td>
<td>Break</td>
</tr>
<tr>
<td><strong>11:00 - 12:00</strong></td>
<td><strong>Spark simulation</strong></td>
</tr>
<tr>
<td>12:00 - 13:00</td>
<td>Lunch</td>
</tr>
<tr>
<td>13:00 - 13:30</td>
<td>Spark continued</td>
</tr>
<tr>
<td>13:30 - 14:30</td>
<td>GraphLab simulation</td>
</tr>
<tr>
<td>14:30 - 15:00</td>
<td>Break</td>
</tr>
<tr>
<td>15:00 - 16:15</td>
<td>Web search simulation</td>
</tr>
<tr>
<td>16:15 - 17:00</td>
<td>Case studies</td>
</tr>
</tbody>
</table>
Agenda

• Objectives
  • be able to deploy data analytics framework
  • be able to simulate Spark engine, tasks

• Outline
  • Learn Spark with interactive shell
  • Instrument Spark for simulation
  • Create checkpoints
  • Simulate from checkpoints
What is Spark?

Apache Spark is a fast and general engine for large-scale data processing

- Efficiency sources
  - General execution graphs
  - In-memory storage
- Usability sources
  - Rich APIs in Python, Scala, Java
  - Interactive shell

http://spark.apache.org/
Spark History

- Started in 2009
- Open sourced in 2010
- Many companies use Spark
  - Yahoo!, Intel, Adobe, Quantifind, Conviva, Ooyala, Bizo and others
- Many companies are contributing to Spark
  - Over 24 companies
- More information: http://spark.apache.org/
Spark Stack

- Spark is a part of the Berkeley Data Analytics Stack
- Spark unifies multiple programming models on same engine
  - SQL, streaming, machine learning, and graphs

![Spark Stack Diagram]

[https://www.safaribooksonline.com](https://www.safaribooksonline.com)
Benefits of Unification

- For the engine
  - Reduction in engine code size
  - Improvement in engine performance

- For users
  - Composition of different models
    (e.g. run SQL query then PageRank on results)
  - Fast composition (no writing to disk)
  - Easy status inspection with Spark shell
Why Spark?

• MapReduce simplifies big data analysis

• However, it performs poorly for:
  • Complex, multi-pass analytics (e.g. ML, graph)
  • Interactive ad-hoc queries
  • Real-time stream processing
Data Sharing in MapReduce

- Mapreduce model is slow
  - replication, serialization, and disk IO

![Diagram showing data sharing in MapReduce]
• Spark is fast
  
  • In-memory accesses 10-100x faster than network and disk
Key Idea: Resilient distributed datasets (RDDs)

- Fault-tolerant collection of elements
  - Can be cached in memory
  - Can be manipulated through parallel operators

- Two ways to create RDDs
  - Parallelizing existing RDD
  - Referencing a dataset in external storage system
    - E.g., shared filesystem, HDFS, HBase, ...
Generality of RDDs

- RDDs are coarse-grained interfaces,
- RDDs can express many parallel algorithms
  - Kmeans, SVMs, logistic regression, SVD, PCM, ...
- RDDs capture many current programming models
  - Data flow models: MapReduce, Dryad, SQL, ...
  - Iterative models: BSP (Pregel), iterative MapReduce, bulk incremental, ...
- And new models that these models do not
### Some Important Spark Operations

| Transformations (define a new RDD) | map  
|                                 | filter  
|                                 | sample  
|                                 | groupByKey  
|                                 | reduceByKey  
|                                 | sortByKey  
| Actions (return a result to driver program) | collect  
|                                 | reduce  
|                                 | count  
|                                 | save  
|                                 | lookupKey  
| flatMap  
| union  
| join  
| cogroup  
| cross  
| mapValues  

[biglearn.spark.noVideo.pdf]
Scheduling Process

RDD Objects
- `rdd1.join(rdd2)`
- `groupBy(...)`
- `filter(...)`

build operator DAG

DAGScheduler
- split graph into stages of tasks
- submit each stage as ready
- agnostic to operators!

TaskScheduler
- launch tasks via cluster manager
- retry failed or straggling tasks
- doesn’t know about stages

Worker
- execute tasks
- store and serve blocks

[Spark Internals (http://www.slideshare.net)]
Conclusion

• Spark provides faster framework for big data analytics
  • Complex analytics (e.g. machine learning)
  • Interactive ad-hoc queries
  • Real-time stream processing
• Spark unifies different models and enables sophisticated apps
Datacenter Simulation Methodologies
Getting Started with Spark

Tamara Silbergleit Lehman, Qiuyun Wang, Seyed Majid Zahedi
and Benjamin C. Lee

This work is supported by NSF grants CCF-1149252, CCF-1337215, and STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA.
Agenda

- Objectives
  - be able to deploy data analytics framework
  - be able to simulate Spark engine, tasks

- Outline
  - Experiment with Spark
  - Instrument Spark tasks for simulation
  - Create checkpoints
  - Simulate from checkpoints
• Launch Qemu emulator

```
$ qemu-system-x86_64 -m 4G -nographic -drive file=micro2014.qcow2,cache=unsafe
```

• Install Java (may take \( \sim 15 \text{min} \))

```
# apt-get update
# apt-get install openjdk-7-jdk openjdk-7-jre
```

• Download pre-built Spark

```
# wget http://d3kbcqa49mib13.cloudfront.net/spark-1.1.0-bin-hadoop1.tgz
# tar -xvf spark-1.1.0-bin-hadoop1.tgz
```
Interactive Analysis with the Spark Shell

- Launch Spark interactive Python interpreter
  
  ```bash
  # cd spark-1.1.0-bin-hadoop1
  # ./bin/pyspark
  ```

- Create RDD from input file
  
  ```python
  >>> textFile = sc.textFile("README.md")
  ```

- Count number of items in RDD
  
  ```python
  >>> textFile.count()
  ```

- See first item in RDD
  
  ```python
  >>> textFile.first()
  ```
More on RDD Operations

- Filter all lines with "Spark"

```python
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

- Count number of lines with "Spark"

```python
>>> linesWithSpark.count()
```

- Find maximum number of words in lines:

```python
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if(a > b) else b)
```
• Do same thing in different way:

```python
>>> def max(a, b):
...     if a > b:
...         return a
...     else:
...         return b

>>> textFile.
     map(lambda line: len(line.split())).
     reduce(max)
```
WordCount Example

• Count words with map and reduce functions

```python
>>> wordCounts = textFile.
    flatMap(lambda line: line.split()).
    map(lambda word: (word, 1)).
    reduceByKey(lambda a, b: a + b)
```

• Return all the elements of the dataset

```python
>>> wordCounts.collect()
```
• Spark supports cluster-wide in-memory caching

```scala
>>> linesWithSpark.cache()
```

```scala
>>> linesWithSpark.count()
```

```scala
>>> linesWithSpark.count()
```

• Very useful when data is accessed repeatedly
  • e.g., querying a small hot dataset
  • e.g., running an iterative algorithm like PageRank
Prepare Spark Simulation

- Exit python shell

  >>> exit()

- Copy “ptlcalls.h”

  # scp user@domain:/path/to/marss/ptlsim/tools/ptlcalls.h .

- Create ptlcalls.cpp file

  # vim ptlcalls.cpp
#include <iostream>
#include "ptlcalls.h"
#include <stdlib.h>

extern "C" void create_checkpoint(){
    char *ch_name = getenv("CHECKPOINT_NAME");
    if(ch_name != NULL) {
        printf("creating checkpoint %s\n", ch_name);
        ptlcall_checkpoint_and_shutdown(ch_name);
    }
}

extern "C" void stop_simulation(){
    printf("Stopping simulation\n");
    ptlcall_kill();
}
Build lib Library

- Install necessary packages
  
  ```bash
  # apt-get install gcc g++ build-essential
  ```

- Compile C++ code
  
  ```bash
  # g++ -c -fPIC ptlcalls.cpp -o ptlcalls.o
  ```

- Create shared library for Python
  
  ```bash
  # g++ -shared -Wl,-soname,libptlcalls.so -o libptlcalls.so ptlcalls.o
  ```

- Copy the library
  
  ```bash
  # cp libptlcalls.so ./bin/
  ```
Create Checkpoint for WordCount

- Include C++ library in WordCount python code
  (../examples/src/main/python/wordcount.py)

```python
from ctypes import cdll
lib = cdll.LoadLibrary('./libptlcalls.so')
```

- Call C++ function to create checkpoint for reduceByKey phase

```python
counts = lines.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1))
output = counts.collect()
lib.create_checkpoint()
lib.stop_simulation()
```
```python
counts = counts.reduceByKey(sample)
output = counts.collect()
lib.stop_simulation()
```
Create WordCount Checkpoint

- Shutdown Qemu emulator and run Marssx86’s Qemu

```
# shutdown -h now

$ ./qemu/qemu-system-x86_64 -m 4G -drive file=image/spark.qcow2,cache=unsafe -nographic
```

- Export CHECKPOINT_NAME

```
# export CHECKPOINT_NAME=wordcount
```

- Run wordcount.py example

```
# cd spark-1.1.1-bin-hadoop1/bin
# ./spark-submit
  ../examples/src/main/python/wordcount.py
  ../README.md
```
Simulate "wordcount" Checkpoint

- Check wordcount checkpoint
  
  ```
  $ qemu-img info ~/micro2014.qcow2
  ```

- Two ways to run from checkpoints
  - Manual run using terminal commands
  - Batch run using run_bench.py code
Manual Run

- Prepare wc.simcfg:

```bash
-logfile wordcount.log
-run
-machine single_core
-corefreq 3G
-stats wordcount.yml
-startlog 10M
-loglevl 1
-kill-after-run
-quiet
-dramsim-device-ini-file file ini/
    DDR3_micron_32M_8B_x4_sg125.ini
-dramsim-results-dir-name wordcount
```
• Run terminal command:

```bash
$ ./qemu/qemu-system-x86_64 -m 4G -drive file=/path/to/image,cache=unsafe -nographic -simconfig wc.simcfg -loadvm wordcount
```
Prepare util.cfg:

```
[DEFAULT]
marss_dir = /path/to/marss/directory
util_dir = %(marss_dir)s/util
img_dir = /path/to/image
qemu_bin = %(marss_dir)s/qemu/qemu-system-x86_64

default_simconfig = -kill-after-run -quiet

[suite spark]
checkpoints = wordcount

[run spark_single]
suite = spark
images = %(img_dir)s/spark.qcow2
memory = 4G
simconfig = -logfile %(out_dir)s/%(bench)s.log
    -machine single_core
    -corefreq 3G
    -run
    -stats %(out_dir)s/%(bench)s.yml
    -dramsim-device-ini-file ini/DDR3_micron_32M_8B_x4_sg125.ini
    -dramsim-results-dir-name %(out_dir)s_%(bench)s
    -startlog 10M
    -loglevel 1
    %(default_simconfig)s
```
• Run run_bench.py

```
$ ./util/run_bench.py -d run/wordcount_test
  spark_single
```

• More information:
  http://marss86.org/~marss86/index.php/Batch_Runs
Other Libraries and Real Data Sets

- Libraries
  - Correlations (Correlation between label and features)
  - Kmeans
  - Decision Tree
  - Logistic Regression

- Data Sets
  - http://www.umass.edu/statdata/statdata/stat-logistic.html
  - http://cs.joensuu.fi/sipu/datasets/
  - http://www.limfinity.com/ir/
Agenda

• Objectives
  • be able to deploy data analytics framework
  • be able to simulate Spark engine, tasks

• Outline
  • Experiment with Spark
  • Instrument Spark tasks for simulation
  • Create checkpoints
  • Simulate from checkpoints
Questions?
## Tutorial Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 - 09:15</td>
<td>Introduction</td>
</tr>
<tr>
<td>09:15 - 10:30</td>
<td>Setting up MARSSx86 and DRAMSim2</td>
</tr>
<tr>
<td>10:30 - 11:00</td>
<td>Break</td>
</tr>
<tr>
<td>11:00 - 12:00</td>
<td>Spark simulation</td>
</tr>
<tr>
<td><strong>12:00 - 13:00</strong></td>
<td><strong>Lunch</strong></td>
</tr>
<tr>
<td>13:00 - 13:30</td>
<td>Spark continued</td>
</tr>
<tr>
<td>13:30 - 14:30</td>
<td>GraphLab simulation</td>
</tr>
<tr>
<td>14:30 - 15:00</td>
<td>Break</td>
</tr>
<tr>
<td>15:00 - 16:15</td>
<td>Web search simulation</td>
</tr>
<tr>
<td>16:15 - 17:00</td>
<td>Case studies</td>
</tr>
</tbody>
</table>