HiBench Introduction

Carson Wang
(carson.wang@intel.com)
Agenda

• Background
• Workloads
• Configurations
• Benchmark Report
• Tuning Guide
Background

**WHY**
- Why we need big data benchmarking systems?

**WHAT**
- What is HiBench?

**HOW**
- How to use HiBench?
Big data ecosystem is complex

- **Hadoop**
  - MR1
  - MR2

- **Spark**
  - Scala
  - Java
  - Python

- **Streaming**
  - Spark
  - Storm
  - Storm-Trident
  - Samza

- **Deployment**
  - Standalone
  - YARN
  - Kafka
  - Zookeeper

- **Application**
  - SQL
  - MachineLearning
  - Graphx
Frequently Asked Questions from Customers

• Which framework is better?
  – Hadoop, Spark
  – Spark scala/java/python
  – SparkStreaming, Storm, or Samza
  – Standalone/YARN

• What is the performance difference between two deployments?

• How many resources are needed?
  – CPU cores, memory, network bandwidth

• Is the cluster configured properly?
  – Spark executor number, partition number tuning
HiBench Roadmap

HiBench 1.0 (2012.6)
- initial release

HiBench 2.0 (2013.9)
- CDH, hadoop2 support

HiBench 3.0 (2014.10)
- YARN support, Sparkbench

HiBench 4.0 (2015.3)
- Workload abstraction framework

HiBench 5.0 (2015.10)
- StreamingBench
Workload abstraction

- Micro Benchmark
- Machine Learning
- Web Search
- SQL
- Streaming
- HDFS Benchmark
Micro Benchmarks

- **Sort**
  - Sorts its text input data generated by RandomTextWriter.

- **WordCount**
  - Counts the occurrence of each word in the input data, which are generated using RandomTextWriter.

- **TeraSort**
  - TeraSort is a standard benchmark created by Jim Gray. Its input data is generated by Hadoop TeraGen example program.

- **Sleep**
  - Sleeps a few of seconds in each task to test framework scheduler.
Machine Learning

• **Bayes**
  – Benchmarks NaiveBayesian Classification implemented in Spark-MLlib/Mahout examples. A popular classification algorithm for knowledge discovery and data mining.

• **K-Means**
  – Tests the K-means (a well-known clustering algorithm for knowledge discovery and data mining) clustering in Spark-MLlib/Mahout.
Web Search

• **PageRank**
  - An implementation of Google’s Web page ranking algorithm.

• **Nutch Indexing**
  - Tests the indexing sub-system in Nutch, a popular open source (Apache project) search engine.
SQL

- Scan
- Join
- Aggregation

Adapted from the Pavlo’s benchmark "A Comparison of Approaches to Large-Scale Data Analysis" and HIVE-396. It contains queries performing the typical OLAP queries described in the paper.
HDFS Benchmark

• Enhanced DFSIO
  – Tests the HDFS throughput of the Hadoop cluster by generating a large number of tasks performing writes and reads simultaneously. It measures the average I/O rate of each map task, the average throughput of each map task, and the aggregated throughput of the HDFS cluster.
Streaming

- Identity
- Sample
- Project
- Grep
- Wordcount
- Distinctcount
- Statistics
Quick Start

1. Build HiBench with Maven

   mvn clean package –D spark1.5 –D MR2

2. Add basic configurations in conf/99-user_defined_properties.conf

   hibench.hadoop.home The Hadoop installation location
   hibench.spark.home The Spark installation location
   hibench.hdfs.master hdfs://sr555:8020
   hibench.spark.master yarn-client
Quick Start

3. Generate the input data and run the workload

workloads/wordcount/prepare/prepare.sh
workloads/wordcount/mapreduce/bin/run.sh
workloads/wordcount/spark/scala/bin/run.sh
workloads/wordcount/spark/java/bin/run.sh
workloads/wordcount/spark/python/bin/run.sh

4. View the report

View report/hibench.report to see the workload input data size, execution duration, throughput(bytes/s), throughput per node.
Configuration mechanism

• Hierarchical configuration
  – Global level conf/
  – Workload level workloads/<name>/conf/
  – Framework level workloads/<name>/mapreduce/hadoop.conf
  – Language level workloads/<name>/spark/scala/scala.conf
## Configuration - Parallelism

- **Parallelism tuning**
  - Parallelism of workload

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hibench.default.map.parallelism</td>
<td>Mapper numbers in MR, default parallelism in Spark</td>
</tr>
<tr>
<td>hibench.default.shuffle.parallelism</td>
<td>Reducer number in MR, shuffle partition number in Spark</td>
</tr>
</tbody>
</table>

- **Executor numbers & cores**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hibench.yarn.executors.num</td>
<td>Number executors in YARN mode</td>
</tr>
<tr>
<td>hibench.yarn.executors.cores</td>
<td>Number executor cores in YARN mode</td>
</tr>
</tbody>
</table>
Configuration - Memory

• Memory

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.executors.memory</td>
<td>Executor memory, standalone or YARN mode</td>
</tr>
<tr>
<td>spark.driver.memory</td>
<td>Driver memory, standalone or YARN mode</td>
</tr>
</tbody>
</table>
Configuration – Data Scale

• Data scale profile definition:
  – conf/10-data-scale-profile.conf

#Wordcount
hibench.wordcount.tiny.datasize 32000
hibench.wordcount.small.datasize 320000000
hibench.wordcount.large.datasize 3200000000
hibench.wordcount.huge.datasize 32000000000
hibench.wordcount.gigantic.datasize 320000000000
hibench.wordcount.bigdata.datasize 1600000000000

• Data scale profile selection:

  hibench.scale.profile large
Benchmark Report

• Each report/workload/framework/language has a dedicated folder which contains:
  – Console output log
  – System utilization log
  – Configurations files
    • Bash environment conf file used by workload scripts: <workload_name>.conf
    • All-in-one property based conf generated by all confs: sparkbench.conf
    • Spark/Samza/… related conf extracted from all confs: spark.conf, samza.conf, …
Report Example

- ./report
  - ./report/hibench.report
    - Summarized workload report, including workload name, execution duration, data size, throughput per cluster, throughput per node
  - ./report/<workload>/<language_api>/bench.log
    - Driver log when running this workload/api
  - ./report/<workload>/<language_api>/monitor.html
    - System utilization monitor chart on CPU, Disk, Network, Memory, Proc Load
  - ./report/<workload>/<language_api>/conf/<workload>.conf
    - Environment variable configuration generated for this workload
  - ./report/<workload>/<language_api>/conf/sparkbench/sparkbench.conf
    - Properties configuration generated for this workload
  - ./report/<workload>/<language_api>/conf/sparkbench/spark.conf
    - Spark properties configuration generated for this workload
Spark Workloads Tuning Guide

- Determine executor cores and numbers
- Determine parallelism
- Determine partition size
- Monitor system utilization
Determine Executor cores and numbers

- Set $N=$ Number of virtual CPU cores in one node.
- Set $M=$ Number of worker nodes in the cluster.
- Set executor cores to a number around 5:
  - $\text{Executor\_cores}=5$
- Calculate executor number running in each node:
  - $\text{Executor\_number\_in\_node} = \frac{(N-1)}{\text{Executor\_cores}}$
- Calculate total executor numbers
  - $\text{Executor\_number} = \text{Executor\_number\_in\_node} \times M$
Determine Parallelism

• Calculate Parallelism:
  \[ P = \text{Executor}\_\text{cores} \times E\text{xecutor}\_\text{number} \]

• Calculate memory for each executor:
  – \[ \text{Executor}\_\text{memory} \leq \frac{\text{Memory}\_\text{node}}{\text{Executor}\_\text{number}\_\text{in}\_\text{node}} \times 0.9 \]
Determine partition size

- Partition size should \( \geq P \) (parallelism)
  - But, a right partition size will vary with real cases.
  - Low parallelism:
    - Partition size less than \( P \): not enough tasks for all vcores
    - Partition size too large: Tasks finished too fast. Task scheduler can’t feed enough tasks.
  - Out of memory:
    - Partition size too small.
- Guideline
  - Increase partition size starting from \( P \) until performance begin to drop.
Tuning Example

Hardware Specs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>E5-2699 v3</td>
</tr>
<tr>
<td>CPU Core#</td>
<td>18</td>
</tr>
<tr>
<td>CPU Socket#</td>
<td>1</td>
</tr>
<tr>
<td>Threads Per Core#</td>
<td>2</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>256 GB</td>
</tr>
<tr>
<td>Disk (SSD)</td>
<td>4 x 400 GB</td>
</tr>
<tr>
<td>NIC</td>
<td>10 Gb</td>
</tr>
<tr>
<td>Cluster Size</td>
<td>3 Slaves</td>
</tr>
</tbody>
</table>

N = 36
M = 3
Executor Cores = 5
Executor Number in one node = (36 - 1) / 5 = 7
Total Executor Number = 7 * 3 = 21
Executor Memory <= 256 / 7 x 0.9 = 32 GB
Parallelism P = 5 * 21 = 105
Partition Size = 105, 210, 315, etc
Spark Tuning Example - TeraSort
Spark Tuning Example - KMeans

Low CPU usage. Too less input Data!
RDDs produced by textFile (sequenceFile, etc) have the partitions determined by input data size and HDFS block size
HiBench 6.0 What’s Next

• More real-world workloads
• More flexible build and configuration
• A better resource monitoring service
Q & A

Thanks