Big Data Benchmarking
On ARM

Apr. 2019
Guodong Xu
About Me:

Guodong Xu  - guodong.xu@linaro.org

Joined Linaro in 2012

Worked on Linux kernel, and Big Data

My Shared Page:

[1] Bigtop (v1.3.0) Build, Smoketest, and Deploy on Multiple Physical Machines
[2] How to 'Consume' a Bigtop release - to get a repository URL
[4] Issues I met when running NutchIndexing and How I fixed them

Pages from Linaro Big Data Team:

[1] https://collaborate.linaro.org/display/BDTS/Big+Data+Team+Space
Prelude - 1/

Running Big Data application on ARM Servers was a challenging job. Up until recently

In this session, I will cover:

- **Bigtop**
- Running the entire set of **HiBench** & **Yahoo streaming bench** benchmarking tools
  - Covering multiple level of Big Data application.
- Performance tuning examples, taking from **TeraSort**, **NutchIndexing** in Hibench, and **Flink latency** in Yahoo streaming bench.
Prelude - 2/ Challenges of Big Data benchmarking

- **Variety / Disparate sources**
  - Tend to host multiple use cases from diverse application domains. Sometimes mutually exclusive
  - Challenge is to be representative and portable.

- **Rapid Landscape Evolution and Expansion**
  - Constant and rapid evolution
  - Ensuring *benchmarking tools keep sufficient pace* with such rapid evolution to remain relevant

- **Landscape Diversity and Scalability**
  - Often involves multiple, distributed components
  - To be truly scalable and adequately capture the multidimensional scaling paradigm

- **System Complexity**
  - Distributed nature of big data system makes it extremely challenging

Ref: "Benchmarking Big Data", Page 9, by Ganesh Raju and Naresh Bhat
Running Big Data on ARM Server

Ready, Steady, Go!

1. Arm64 Server readiness - Hardware, booting firmware, OS
   - Arm ServerReady ®
   - Silicon vendors
2. Big Data deployments on Arm Servers. - Software stacks
3. Benchmarking, as a measure of multiple dimension, pass and excel.
Prelude - 3/

- Various distributions of Big Data solutions.
  - Such as CDH, MapR, HDP (hortonworks), and more.

- Numerous tools exist for benchmarking Big Data.
  - Such as TPCx-HS, HiBench, Spark-Bench, MRBench, NNBench... etc.

Why BigTop?

Why HiBench?
Part I - Bigtop

**Bigtop** is an umbrella project for many Big Data components.

In its original release and CI, only available for x86 and powerpc.

Old Ages:
Before that, to run on Arm server, need lots of quirks, and manual tuning. See

- Linaro Big Data team webpage, [https://collaborate.linaro.org/display/BDTS/Documentation](https://collaborate.linaro.org/display/BDTS/Documentation)
- What Linaro added to bigtop's source code tree.

```
git clone https://git.linaro.org/leg/bigdata/bigtop-trunk.git -b erp18.06
```
1.3.0 is the first to officially support Arm.

Contains:
- Hadoop: 2.8.4
- Spark: 2.2.1
- Flink: 1.4.2
- more ...

Pros:
- Roles defined in site.yaml
- Start/stop as a system service
  - 'sudo systemctl ...'

Further reading: Bigtop (v1.3.0) Build, Smoketest, and Deploy on Multiple Physical Machines
Part II - HiBench

- HiBench is a big data benchmark suite that helps evaluate different big data frameworks
  - Hadoop
  - Spark
  - Streaming (Spark Streaming, Flink, Storm, and Gearpump)

- 19 workloads, in 6 categories
  - Micro,
  - Machine Learning,
  - SQL,
  - Graph,
  - Websearch,
  - Streaming.

Eg.:
Sort, WordCount, TeraSort, Sleep,
SQL scan, join,
PageRank, Nutch indexing,
Bayes, Kmeans, NWeight,
enhanced DFSIO, etc.
HiBench

As a benchmarking suite, some comments:

- **Pros:**
  - Simple configuration file to connect your Big Data env.
    - `conf/hadoop.conf`, `conf/spark.conf`
  - Simple scripts to run many popular benchmarks in one line. Life becomes easier.
    - `bin/run_all.sh`
    - `conf/benchmarks.lst`
  - Each test case is splitted into two parts: prepare.sh and run.sh
  - Multiple frameworks, hadoop, spark, flink.
    - `conf/frameworks.lst`

- **Cons:**
  - Result only duration time. It's not useful for some Benchmarks.

Further reading: [HiBench 7.0 Build and Run, a step-by-step guidance](#)
HiBench - 2/ Running on Arm Server

All benchmarks supported on HiBench have been run on my Arm server cluster.

[ Benchmarking Coverage ]

Note: Linaro does not publish any benchmarking result. Please ask specific Arm server vendors for detail benchmarking data.
<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmark</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hadoop</strong></td>
<td>HadoopWordcount</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopSort</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopTerasort</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopSleep</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopDfsioe-read</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopDfsioe-write</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Machine Learning</strong></td>
<td>HadoopBayes</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopKmeans</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>HadoopScan</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopJoin</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopAggregation</td>
<td>Pass</td>
</tr>
<tr>
<td><strong>Websearch</strong></td>
<td>HadoopPagerank</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>HadoopNutchindexing</td>
<td>Pass</td>
</tr>
</tbody>
</table>
## HiBench - 4/ Test Coverage on Arm Server

<table>
<thead>
<tr>
<th>Spark</th>
<th>Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScalaSparkWordcount</td>
<td>Pass</td>
</tr>
<tr>
<td>ScalaSparkSort</td>
<td>Pass</td>
</tr>
<tr>
<td>ScalaSparkTerasort</td>
<td>Pass</td>
</tr>
<tr>
<td>ScalaSparkSleep</td>
<td>Pass</td>
</tr>
<tr>
<td>ScalaSparkBayes</td>
<td>Pass</td>
</tr>
<tr>
<td>ScalaSparkKmeans</td>
<td>Pass</td>
</tr>
<tr>
<td>LogisticRegression</td>
<td>Pass</td>
</tr>
<tr>
<td>Alternating Least Squares (ALS)</td>
<td>Pass</td>
</tr>
<tr>
<td>GradientBoostingTree</td>
<td>Pass</td>
</tr>
<tr>
<td>LinearRegression</td>
<td>Pass</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation (LDA)</td>
<td>Pass</td>
</tr>
<tr>
<td>Principal Components Analysis (PCA)</td>
<td>Pass</td>
</tr>
<tr>
<td>RandomForest</td>
<td>Pass</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>Pass</td>
</tr>
<tr>
<td>Singular Value Decomposition (SVD)</td>
<td>Pass</td>
</tr>
<tr>
<td>Spark</td>
<td>SQL</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td>ScalaSparkScan</td>
</tr>
<tr>
<td></td>
<td>ScalaSparkJoin</td>
</tr>
<tr>
<td></td>
<td>ScalaSparkAggregation</td>
</tr>
<tr>
<td>Websearch Benchmarks</td>
<td>ScalaSparkNWeight</td>
</tr>
<tr>
<td></td>
<td>ScalaSparkPagerank</td>
</tr>
<tr>
<td>Graph Benchmark</td>
<td>NWeight (nweight)</td>
</tr>
</tbody>
</table>
Part III - Benchmarking Workloads

TeraSort
TeraSort Setup

- 3 Arm servers each with multi-cores (Cortex-A72), 2.4 GHz, 11 * 4T SATA, 512G/256G DDR, 10Gbps ethernet
- BigTop 1.3.0, HiBench 7.0
TeraSort w/ Hadoop

- Tuning Terasort is a combat with CPU load
  - Quite easy to reach a stage where CPU load per core > 1.5
- Disk spindles too.
  - `iostat -xz`: `avgqu_sz > 1 (40+)`

Left: Impact of Parallel Jobs
Right: Tendency over workload increase.
TeraSort - How well our friends can do?

Out of Curiosity,

1TB: 494 seconds using MapR 4.0.1, sort rate: 0.1GB/sec/node.

1TB: 822 seconds using CDH 5.10, sort rate: 0.06GB/sec/node.

Test Environment Details
Number of Nodes: 20+1 node for NameNode/CLDB + YARN Resource Manager
   RAM: 128GB
   Disks: 11 Disks—110 GB
   CPU: 2x16 cores
   Network: 10 GbE

Ref: https://mapr.com/whitepapers/terasort-benchmark-comparison-yarn/

Disclaimer: Deliberate effort to avoid mentioning details regarding CPU model/GHz. not directly compatible with Arm Servers.
# TeraSort - Parameters tuning

<table>
<thead>
<tr>
<th>File</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn-site.xml</td>
<td>yarn.nodemanager.resource.memory-mb</td>
<td>514048 or 251904</td>
</tr>
<tr>
<td></td>
<td>yarn.schedulermimum-allocation-mb</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>yarn.schedulermimum-allocation-mb</td>
<td>16384</td>
</tr>
<tr>
<td></td>
<td>yarn.schedulermimum-allocation-mb</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>yarn.nodemanager.local-diirs</td>
<td>spread to all available physical drives to maximize parallelity. Eg. /mnt/sda2/yarn,/mnt/sdc1-yarn,/mnt/sdd1/yarn,/mnt/sde1/yarn,</td>
</tr>
<tr>
<td>mapred-site.xml</td>
<td>mapreduce.job.maps</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>mapreduce.job.reduces</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>yarn.app.mapreduce.am.resource.mb</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td>mapreduce.map.memory.mb</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td>mapreduce.reduce.memory.mb</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td>mapreduce.map.java.opts</td>
<td>-Xmx6400m</td>
</tr>
<tr>
<td></td>
<td>mapreduce.reduce.java.opts</td>
<td>-Xmx6400m</td>
</tr>
<tr>
<td></td>
<td>mapreduce.task.io.sort.mb</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>mapreduce.map.cpu.vcores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>mapreduce.reduce.cpu.vcores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>mapreduce.cluster.local.dir</td>
<td>spread to all available physical drives to maximize parallelity. Eg. /mnt/sda2/yarn,/mnt/sdc1-yarn,/mnt/sdd1/yarn,/mnt/sde1/yarn,</td>
</tr>
<tr>
<td>hdfs-site.xml</td>
<td>dfs.datanode.data.dir</td>
<td>spread to all available physical drives to maximize parallelity.</td>
</tr>
<tr>
<td></td>
<td>dfs.namenode.name.dir</td>
<td>spread to all available physical drives to maximize parallelity.</td>
</tr>
</tbody>
</table>
Part IV - Benchmarking Workloads

NutchIndexing - Websearch benchmarks
NutchIndexing w/ Hadoop

- This workload tests the indexing sub-system in Nutch, a popular open source (Apache project) search engine.

- However, it gave me the 'Biggest' trouble ever met in tuning a job
  - It runs very very slowly,
    - CPU load is low
    - Containers quit
    - AM timeout

- I wrote a blog page to list all errors I met while preparing nutchindexing to run.

Further reading: Issues I met when running NutchIndexing and How I fixed them

It turns out ...
Localization of data ... memory ... matters ...

What bottleneck'ed?

Memory Allocation:
- Yarn
- Local

For nutchindexing, conclusion: The size of memory used as cache must be larger than the size of all MR local filecache.

And when the memory cannot cache the whole localization data, the mapper tasks will frequently read data from local hard disk. It reads so frequent and so large (90MBytes/sec) that mappers cannot make meaningful progress at all.
NutchIndexing - Parameters Tuning

Here to list Yarn parameters which have significant impact on overall performance.

- **Tools for debugging**
  - Jstack
  - top -H -p <pid>

- **Parameters old /new value, and why.**
  - memory for yarn nodemanager
  - timeout of AM
  - timeout of task containers
  - memory for mapreduce I/O sort

**Concerns:** No one configuration file can fill all type of tasks. In order to reach maximum performance, have to tune one after one.
Part V - Streaming Bench

Benchmark Tool:

- Yahoo Stream Bench

Computation engine tested:

- Flink
Flink - Running on Arm server

2 Data Producers, 1 Kafka node, 1 Flink node.

Ref: Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming, from IEEE International Parallel and Distributed Processing Symposium Workshops
Key conclusions Yahoo team drew from x86 server stays true on Arm servers.

- The percentile latency for all Kafka emit rates are relatively the same.
- The percentile latency rises linearly until around the 99th percentile, where the latency appears to increase exponentially.

Ref: Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming, from IEEE International Parallel and Distributed Processing Symposium Workshops. Also available here.

Note: Yahoo team in their IEEE reports uses 40 nodes, and each with two Intel E5530 processors running at 2.4GHz and a total of 16 cores (8 physical, 16 hyperthreading) per node. Each node has 24GB of memory.
## Flink - Parameter Tuning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskmanager.numberOfTaskSlots</td>
<td>56</td>
</tr>
<tr>
<td>taskmanager.heap.mb</td>
<td>196608</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num.io.threads</td>
<td>8</td>
</tr>
<tr>
<td>linger.ms</td>
<td>0</td>
</tr>
<tr>
<td>log.dirs</td>
<td>12 physical disks with 1 disk for system OS usage. Allocate 11 physical disks for Kafka log.dir usage.</td>
</tr>
<tr>
<td>num.partitions</td>
<td>12</td>
</tr>
<tr>
<td>num.network.threads</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusion

● Ecosystem for Big Data on Arm is maturing.
● Arm servers at current state are powerful for real-world production usage.
● Efforts from Linaro and ecosystem, tuning on Arm servers for each of the tasks is no different compared to x86.
  ○ Same toolset, similar configuration changes, same skill set.

Thanks to:

● Linaro Big Data team, Ganesh Raju (tech lead)
● Linaro members.

Questions?

Contact: guodong.xu@linaro.org
END