Spark on large Hadoop cluster and evaluation from the viewpoint of enterprise Hadoop user and developer

Masaru Dobashi (NTT DATA)
Who am I?

- I’m Masaru Dobashi
- Chief Engineer of NTT DATA in Japan
  
  One of leading solution provider in Japan

- My team focusing on Open Source Software solutions
- I’ve been integrated several Hadoop systems for 5+years
  
  The largest one is a 1000+ nodes cluster

- In these years, also utilize Spark, Storm, and so on.
NTT Group

- Total Asset: ¥19.6,536 trillion
- Operating Revenues: ¥10.7007 trillion
- Number of Employees: 227,168
- Number of Consolidated Subsidiaries: 827

Regional Communications Business

- NTT EAST
  - NIPPON TELEGRAPH AND TELEPHONE EAST CORPORATION
    - [100%]

- NTT WEST
  - NIPPON TELEGRAPH AND TELEPHONE WEST CORPORATION
    - [100%]

Long-Distance and International Communications Business

- NTT Communications Corporation
  - [100%]

- Dimension Data Holdings plc.
  - [100%]

Mobile Communications Business

- NTT DOCOMO, INC.
  - [66.7%]

Data Communications Business

- NTT DATA CORPORATION
  - [54.2%]

Net Sales: USD 13.2 billion
  - (June, 2014; USD 1 = JPY 102)

Employees: 75,000
  - (January, 2014)

• Planning management strategies for the NTT Group.
• Encouraging fundamental R&D efforts

※ NTT’s Voting Rights Ratio (as of Mar. 31, 2013)
Our motivation and expectation for Spark

Characteristics of its performance with GBs, TBs and tens of TBs of data.

Tips for the people who are planning to use Spark
Hadoop has become the base of data processing

- We started to use Hadoop 6 years ago
- Hadoop enables us to process massive data daily and hourly

The system image 6 years ago:

- Data loader
- Hadoop (MapReduce, HDFS)
- Hive
- Pig
- Outer system
  - Daily/Hourly
  - Batch
Example of massive data processing

NTT DOCOMO supports growth in society and industry

NTT DOCOMO’s project supports the research of society and industry using large-scale operational data.

Incoming data
30 billion + / day
1 TB / day

Generated data
PBs of data

Demands for data processing have diversified

- **Handle variety of requirements for data processing**
  - Both throughput and low latency
  - APIs useful for data analysis

  This should be achieved by Spark

- **Make the data management simple**
  - Want to run different types of frameworks on one HDFS
  - Because multi clusters themselves impose complexity and inefficiency in data management

  This should be achieved by Hadoop2.x and YARN
Spark and other data frameworks collaborate with each other

- Flume, Fluentd
- RabbitMQ, Kafka
- Spark processes data on HDFS and run on YARN with other framework

Outside service

Messaging

MapReduce
Hive
Pig

KVS/cache

Hadoop (YARN, HDFS)

HBase

on-memory

Spark

stream

Dataflow

Storm

Spark Streaming

batch

Outside service

Visualization service
The evaluation of Spark on YARN
Four essential points we wanted to evaluate

**Basic viewpoint**

The basic characteristics about scale-out. Especially about TBs and tens of TBs of data.

<table>
<thead>
<tr>
<th>#</th>
<th>Points we wanted to evaluate</th>
<th>Apps used for evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Capability to process tens of TBs of data <em>without unpredictable decrease of performance nor unexpected hold</em></td>
<td>WordCount</td>
</tr>
<tr>
<td>2</td>
<td>Keep reasonable performance when data is bigger than total memory available for caching</td>
<td>SparkHdfsLR (Logistic Regression)</td>
</tr>
<tr>
<td>3</td>
<td>Keep reasonable performance of shuffle process with tens of TBs of data</td>
<td>GroupByTest (Large shuffle process)</td>
</tr>
<tr>
<td>4</td>
<td>Easy to implement the multi-stage jobs (from our business use-case)</td>
<td>POC of a certain project</td>
</tr>
</tbody>
</table>
The specification of the cluster

Total cluster size

- 4k+ Core
- 10TB+ RAM

<table>
<thead>
<tr>
<th>Item</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>E5-2620 6 core x 2 socket</td>
</tr>
<tr>
<td>Memory</td>
<td>64GB 1.3GHz</td>
</tr>
<tr>
<td>NW interface</td>
<td>10GBase-T x 2 port (bonding)</td>
</tr>
<tr>
<td>Disk</td>
<td>3TB SATA 6Gb 7200rpm</td>
</tr>
</tbody>
</table>

Software stuck

- Spark 1.0.0
- HDFS & YARN(CDH5.0.1)
- CentOS6.5
We ran tests two times per each data size. OS cache was cleared before the each execution started. The process time seemed to be linear per input data size.

We found reasonable performance, even if all of data cannot be held on cache.
Input data: 27TB

[CPU Usage]
blue: user
green: system

[Network usage]
red: in
black: out

[Disk I/O]
black: read
pink: write

Because of locality, there’re few NW I/O. This is ideal situation.

About 400MB/sec/server
WordCount’s performance depends on Map-side process. Reduce-side process may not be bottleneck. This is because Map-side outputs small data.

On this task, we confirmed reasonable performance, even if the input data exceeded the total memory amount.

Tasks had the locality for data, we observed the stable throughput, (i.e. time vs. data processed)

I will talk about the case which a task lost locality, later.
We ran the logistic regression with 3 cycles of computation. We found ideal difference between cycles. The difference of process time between cycle 1 and others seemed to depend on the cache usage.

Available cache size per server (16 GB * 3 * 0.6 = 26GB)

Cache is effective, even if executor cannot hold all of data.

Using memory and disk

Cache is effective

100% cached

73% cached

33% cached

16% cached

Theoretical input data size (per server)
Point2: Resource usage of a certain slavenode

8GB input per server

[CPU Usage]
blue: user
green: system

[Network usage]
red: in
black: out

[Disk I/O]
black: read
pink: write

<table>
<thead>
<tr>
<th>Time</th>
<th>600 sec</th>
<th>89 sec</th>
<th>85 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>16GB</td>
<td>1229 sec</td>
<td>316 sec</td>
<td>297 sec</td>
</tr>
</tbody>
</table>

Cache mechanism prevented the disk usage

Part of data use disk, because cache cannot hold all of data
The cache mechanism of Spark worked for iterative applications

RDD’s cache mechanism works consistently, and enhances throughput while the amount of input data is bigger than the total memory available for caching

It is important to minimize boxing overhead when storing data object into RDD
Point3: Process time of GroupByTest

Starting fetched shuffle-data spilling to disks, but no impact on total Elapsed Time

We didn’t find drastic changes of gradient, when we processed these size of data.
Actually, we saw the bottleneck of disk I/O as well as the bottleneck of NW. This is typical when we ran shuffle test whose map tasks generated massive output data.

The network resource usage of a certain slavenode when we ran variety patterns of tests

- Small shuffle on one rack
  We saw no spill to disk.

- Large shuffle on one rack
  We saw spill to disk.

- Large shuffle on cluster
  We saw bottleneck of the core switch
The process time seemed to be linear per input size of shuffle.

When the shuffle data spills out to the disk, the disk access would compete among shuffle related tasks, such as *ShuffleMapTask*(WRITE), *Fetcher*(READ), etc. Then, the competition deteriorate the performance.
We categorized existing Hadoop applications in a certain project and made the mock application which represents major business logics of the project.

Groups by different categories

These computation is different from each applications. For example, calculating difference of values between data records.

This application resembles the log analysis to find the feature of web users.
Point4: Lesson learned from this POC

**Tips**

- Use cache mechanism efficiently
- Prevent skew of task allocation in the start
- Prevent too large partition size
- Practices for heap tuning
- Use RDD to manage data rather than own arrays
- Practices for implementation of DISTRIBUTE BY

**Issues**

- Missing data locality of tasks
- Error of web UI when we ran large jobs
We can use the cache mechanism efficiently by minimizing object stored in MemoryStore or the data store of the cache mechanism.

The convenience and the efficiency of data size may have trade-off relationship. But the implicit conversion of Scala can solve it in a certain case.

Rich data format (like case class) + Simple function

Simple data format (Byte, Int, ...) + Rich function

The cost of computation of data in memory is not consequence compared with the disk I/O
It takes a little to start all of containers when we run large jobs on large YARN cluster.

In this case, the allocation of tasks starts before all containers are available, so that some tasks are allocated on non-data-local executors.

Our workaround

```scala
val sc = new SparkContext(sparkConf)
Thread.sleep(sleeptime)
```

We inserted a little sleep time. This reduces total processing time as a result. But...This is really workaround.

Ultimately, we should implement the threshold to start the task allocation. For example, the percentage of containers ready for use may be useful for this purpose.
Prevent skew of task allocation in the start(2)

Input data: 27TB

[CPU Usage]
blue: user
green: system

[Network usage]
red: in
black: out

[Disk I/O]
black: read
pink: write

Example of slavenode’s resource usage

Because of lack of locality, data transfer occurred.
Future work and conclusion
Future work

Find the good collaboration between Spark and YARN. Here are some issues to be resolved.

- Overhead for starting containers
- Avoid skew of task allocation when starting applications
- If we can use I/O resource management in the future, it will realize rigorous management.

Ensure traceability from a statement of application to the framework of Spark.
- This is used for performance tuning and debugging.
Conclusion

Expectation 1
Can scalably process tens of TBs of data without unpredictable decrease of performance nor unexpected hold

Impression
Good! ... but we need some technique for scale out

Expectation 2
Keep reasonable performance when data is bigger than total memory available for caching

Impression
Good! ... but we need some technique to efficiently use the cache

Expectation 3
Capability to run an application on YARN

Impression
We're evaluating now and it is under development right now.
Spark is a young product and has some issues to be solved. But these issues should be resolved by the great community member. We also contribute it!