Spark on Kubernetes Best Practice

Chen, Junjie and Shao, Saisai
About Spark

A unified analytics engine for large-scale data processing.

• DAG based task scheduler
• Explicit Cache API & In-memory computation
• Catalyst as the optimizer & Tungsten project for native execution acceleration
Spark Execution Model
Spark Cluster Manager

- **Spark Standalone**
  - Lightweight built-in cluster manager

- **Apache Mesos**
  - Spark support Apache Mesos in early stage

- **Apache Hadoop Yarn**
  - A cluster resource manager comes from Hadoop 2.0

- **Kubernetes**
  - Start from Spark 2.3.0
Why Kubernetes

Kubernetes vs Yarn and Mesos

• Better fine grained resource management
• Better centralized management and monitoring
• Lots of addon plugins: Logging, Authentication, Authorization, Monitoring and etc.
• Large OSS community
• Multiple version

Industry trend
• Serverless service: require more powerful orchestration framework.
Spark on k8s Architecture

Steps:

1. Submit application through spark-submit
2. K8s create driver node
3. Driver ask k8s scheduler to create executor pods
4. Executor pods receive task and execute
5. Executor send result to driver node
Spark on Kubernetes Status

Implemented
✓ Native Kubernetes support
✓ Static allocation
✓ Cluster mode
✓ Staging server

Working in Progress
• Client Mode, Spark shell and SparkSQL
• Dynamic allocation + External shuffle services

https://spark.apache.org/docs/2.3.0/running-on-kubernetes.html
Use Cases
Sparkling Cloud Data Warehouse

A PB scale EDW with benefits of fast deployment, resource elasticity, high performance and cost-effective.
Sparkling Deployment

Three types of nodes: Master, Core and Elastic
Scale in/out for elastic nodes
Problems

Hard to support auto scaling

High Availability

Flexibility

Multiple tenancy
Tencent Kubernetes Engine

TKE provides container-centric, highly scalable and high performance container management service.

- Fully compatible with Kubernetes native API
- Integrate Tencent Cloud plugin CBS and CLB
- Security insurance through configurable VPC
- Elastic cluster hosting
Sparkling + TKE

How to play?

✓ Allocate cluster through TKE cloud API
✓ Auto scale in/out via TKE cloud API
✓ Administration out of box
✓ Logging auditor
✓ HA
✓ Multiple tenacy
Architecture Overview
What about storage?

- Data locality consideration
- Inside Kubernetes cluster
- Outside Kubernetes cluster
Performance (TBD)
Benchmark

TPC-DS

• Allocate cluster through TKE cloud API
• Hands on scaling via Kubernetes API
• Administration via TKE
• Logging

Need some custom design and optimization
## Cluster Configuration (TBD)

<table>
<thead>
<tr>
<th></th>
<th>Sparkling cluster</th>
<th>Bare Metal</th>
<th>Kubernetes cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>slaves</td>
<td>20</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>vcores</td>
<td>32 x 20</td>
<td>80 x 9</td>
<td>32 x 20</td>
</tr>
<tr>
<td>Memory</td>
<td>128G x 20</td>
<td>256G x 9</td>
<td>128G x 20</td>
</tr>
<tr>
<td>Disk</td>
<td>4T x 20</td>
<td>10T x 9</td>
<td>4T x 20</td>
</tr>
</tbody>
</table>
OS level tuning

<table>
<thead>
<tr>
<th>OS configuration</th>
<th>Default</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU power frequency policy</td>
<td>power save</td>
<td>performance</td>
</tr>
<tr>
<td>Number of network connections</td>
<td>128</td>
<td>1024</td>
</tr>
<tr>
<td>Transparent Huge Page Compaction</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>Restricting memory swappiness</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Max file open number</td>
<td>10240</td>
<td>65535</td>
</tr>
</tbody>
</table>
Kubernetes Configuration
Hadoop Configuration

Before Tuning | After Tuning
---|---
T_LOAD | 109.63% | 113.35% | 105.09% | 109.17%
T_PT | 20.00% | 40.00% | 60.00% | 80.00%
BBQpm@3000 | 0.00% | 20.00% | 40.00% | 60.00%

Hadoop configuration

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Default</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfs.datanode.handler.count</td>
<td>10</td>
<td>128</td>
</tr>
<tr>
<td>dfs.namenode.handler.count</td>
<td>10</td>
<td>128</td>
</tr>
<tr>
<td>dfs.namenode.service.handler.count</td>
<td>10</td>
<td>128</td>
</tr>
<tr>
<td>dfs.datanode.max.xcievers</td>
<td>256</td>
<td>8192</td>
</tr>
<tr>
<td>dfs.datanode.max.transfer.threads</td>
<td>4K</td>
<td>64K</td>
</tr>
<tr>
<td>Sparkling configuration</td>
<td>Default</td>
<td>Updated</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>spark.executor.cores</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>spark.executor.memory</td>
<td>1G</td>
<td>14G</td>
</tr>
<tr>
<td>spark.executor.memoryOverhead</td>
<td>1.6G</td>
<td>2G</td>
</tr>
<tr>
<td>spark.driver.memory</td>
<td>1g</td>
<td>16G</td>
</tr>
<tr>
<td>spark.driver.cores</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>spark.dynamicAllocation.enabled</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>spark.reducer.maxSizeInFlight</td>
<td>48m</td>
<td>128m</td>
</tr>
<tr>
<td>spark.shuffle.spill.compress</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>spark.shuffle.service.enabled</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>spark.shuffle.registration.timeout</td>
<td>5000</td>
<td>600000</td>
</tr>
<tr>
<td>spark.shuffle.registration.maxAttempts</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>spark.network.timeout</td>
<td>120s</td>
<td>1200s</td>
</tr>
<tr>
<td>spark.sql.shuffle.partitions</td>
<td>200</td>
<td>4096</td>
</tr>
<tr>
<td>spark.files.fetchTimeout</td>
<td>60</td>
<td>600s</td>
</tr>
<tr>
<td>spark.port.maxRetries</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>spark.rpc.numRetries</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>
THANKS!