Spark Applications and Optimizations

Lesson 7

Hortonworks

Learning Objectives

- After you complete this lesson you should be able to:
  - Create a spark standalone application
  - Know the important configuration settings
  - Be able to submit an application to Yarn
  - Joe’s best practices

Hortonworks

Creating a Spark Standalone Application

- We need to “set up” the application
  - The shells do a few things for us when we start it up

- We need to create the SparkConf
- We need to create the SparkContext
- We need to import the necessary libraries

Set up - Python

```
from pyspark import SparkContext
from pyspark import SparkConf

if __name__ = “__main__”
    //do your argument length/checks

    sconf = SparkConf() \
        .setAppName(“My App”)
    sc = SparkContext(conf=sconf)
    //Everything else is like the shell
```

Set up - Scala

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object MyApp {
    val sconf = new SparkConf()
        .setAppName(“My App”)
    val sc = new SparkContext(sconf)
    //Everything else is like the shell
```

Building the Spark Application – Scala Only

- Supports MVN and SBT
- Include the dependencies like any other java project

Submitting the Application

- Python:

```
spark-submit MyApp.py
```

- Scala/Java

```
spark-submit --class MyApp MyApp.jar
```

Setting configs in app

- Some configs have setters, some use a generic method

```
sconf=SparkConf()\
    .setAppName(“Word Count”)\
    .set(“configuration”, “value”)

sc = SparkContext(conf=sconf)
```

Setting configs at runtime

- We just pass them as parameters before the jar/py file
- Python

```
spark-submit --master yarn-client MyApp.py
```

- Scala/Java

```
spark-submit --class MyApp --conf <key>=<value> MyApp.jar
```

Hortonworks

Important Configuration Items

Pre Defined Configs

```
--num-executors 20
--executor-mem 8g
--executor-cores 2
--master yarn-client
--driver-memory 1g
```

Set using --conf key=value

```
spark.shuffle.memoryFraction
spark.storage.memoryFraction
spark.default.parallelism
spark.speculation
```

Hortonworks

Updating your serializer to Kryo

```
sconf=SparkConf
    .set(“spark.serializer”,“org.apache.spark.serializer.KryoSerializ
er”)
    .registerKryoClasses(Array(classOf[MyClass1],
    classOf[MyClass2])

sc = SparkContext(conf=sconf)
```

Joe’s Best Practices: Parrallelism

- Parrallelism should be at least a little less then 2x the number of total cores for your application
  - Generally want slightly less than a multiple of the number of cores so we can take advantage of speculative execution and minimize the idle time of resources
- Tasks should take >100ms (via web ui)
  - Scheduling tasks usually takes less then 20ms, don’t want this to be a large factor in your processing time
- Not uncommon for number of partitions to be between 100 and 10000

Job Submission

- Keep as much out of the app as possible to maintain flexibility (leave Kryo config in the app)
- Yarn-Client when doing testing/dev
- Yarn-Cluster when running production

Joe’s Best Practices: Executors

- Memory:
  - 4gb-16gb
- Cores
  - Usually about 1 core per 4gb memory, always atleast 2
- Number of executors depends on the application
  - Memory of executors = file size if not caching
    - Don’t forget to set memory fraction to .05
  - 2x Memory of executors = file size if caching
  - This is not a standard, but a good starting place

Joe’s best Practices: Misc

- speculative execution is good usually
- Kryo for EVERYTHING
- Use persist over cache to force developer to be aware of memory usage
  - MEMORY_ONLY_SER is best 95% of the time

Hortonworks

Conclusion and Key Points

Hortonworks