Spark Tuning

Taming the lightening fast framework

Agenda

- Memory configuration
- Executors
- Shuffle
- Partitioning
- Caching
- Tungsten
- Broadcasting
- Spark streaming receivers

Configuration API

- Spark uses configurations to tune different aspects of system
- We can specify configurations using
  - spark-env.sh
  - SparkConf in the driver program
  - Parameters to the spark-submit
- The configuration set in the driver takes precedence compared to the ones in set in the configuration files

Memory configurations

- Setting right memory for both driver and executor is important
- The driver memory is set using
  - spark.driver.memory - Default is 1g
- It’s good practice to set around 4gb
- The executor memory is set using
  - spark.executor.memory - Default is 1g
- Set it to around 2g in a 8gb 4 core machine
- 60% of the executor memory is used for caching. It can be controlled using spark.storage.memoryFraction

Deciding on number of executors

- It is always good to have multiple small executors on a single worker node rather than having single big executor
- Executors with 2gb ram and 2 executors per core give optimal results
- You can control no of cores per executor using
  - spark.driver.cores
  - spark.executor.cores
- Specifying the executor cores allows to run multiple executors on single worker node

Shuffle

- Shuffle places very important role determining the performance of a spark application
- Always try to avoid/ minimize the shuffle needed
- Multiple techniques can be used to achieve the minimization of shuffle
- Some of them are
  - Filtering
  - Compression
  - Broadcasting

Filtering

- Always try to filter the data from the input source which only keeps needed data for transformation
- If you are using rdd, make sure you call filter explicitly before any reduce or join operations
- If you are using dataframe, use explain command to make sure filter is pushed to as low in the chain as possible
- Make use of smart sources which supports predicate push

Compression

- Always try to use the compression for the shuffle data similar to the compression for map output
- You can set compressing using

  spark.io.compression.codec

- Use kryo serializer with the combination of above compression to further reduce to the shuffle size
- Try avoiding spilling shuffle to disk.

  conf.set(“spark.shuffle.memoryFraction”,”0.3”) // 0.2 by default

Partitioning

- Number of partitions determines the parallelism of you application
- Having right kind number of partitions helps you to balance good load across the cluster
- Few suggestions for good partitioning
  - Avoid empty partitions . Use coalesce to combine multiple of those
  - Always specify no of partitions for reduce operations like reduceByKey

Partitions (contd..)

- Make use custom partitioner if the hash partitioner creates skewed partitions
- Make use of mapParititons API to speed up the partition wise operations
- Make use of lookup API to access partitioned data rather than using filter API
- In smart sources, make sure data source can understand the underneath partition scheme
- Control block time for partitions in Spark streaming

Speculative execution

- Speculative running of tasks allows to handle failed or slow running tasks effectively
- You can enable it using

```
conf.set(“spark.speculation”,”true”)
```

- Do Not enable speculation in
  - When non distributed storages like RDBMS used.
  - When HBase is used as the source

Caching

- Do Not cache all the RDD’s you create. Choose only the ones which are going to use repetitively.
- Choosing right storage level helps in performance
- Call unpersist when you know you are no more using the RDD
- The different storage has different performance characteristics

Storage levels in Cache

- MEMORY_ONLY - Keeps data only in memory with deserialized format
  - Faster access
  - Increased GC pressure
  - Need to recompute when there is fault
- MEMORY_ONLY_SER - Serialized format
  - Slower access
  - Lesser GC as less number of objects
  - Need to recompute when there is fault

Storage levels

- MEMORY_AND_DISK(SER)
  - Slower than only memory
  - Do Not need compute when there is fault
- MEMORY(_AND_DISK)_2
  - Keeps 2 copies of data on the cache
  - Better fault tolerance
- OFF_HEAP
  - Tachyon is used as caching layer
  - Higher latency for read
  - Share cache across multiple contexts

Tungsten

- From 1.4, spark uses off heap memory system tungsten for implementing its cache.
- This greatly increases the performance as there will be less usage of memory and less garbage collection
- Data Frames makes heavy use of the tungsten data types for performance
- Enabled by default in 1.5. Enable using below property in 1.4

```
conf.set(“spark.sql.tungsten.enabled”,”true”)
```

Broadcasting

- Try to use broadcasting for lookup data rather using default scala closures
- Using broadcasting decrease size of the serialize function size and also optimizes the shuffle of data
- Use broadcasted joins whenever possible
- Configure broadcast join size for spark sql using

```
spark.sql.autoBroadcastJoinThreshold  // default is 10 mb
```

Spark streaming receivers

- Every receiver runs in a single worker node. It’s good to have multiple receiver to increase parallelism in the data collection
- Use director connector for kafka stream rather than receivers
- Enable WAL for receiver failure using

  spark.streaming.receiver.writeAheadLog.enable

- Enable the backpressure (1.5)

  spark.streaming.backpressure.enabled