CS x55: Distributed Systems [Spark Streaming]

Shrideep Pallickara
Computer Science
Colorado State University

Computer Science Department
Colorado State University

1

Topics covered in this lecture

- Spark Streaming
  - Performance considerations
  - Example

Colorado State University
Professor: Shrideep Pallickara
Computer Science Department
Spark Streaming
L33-B.2

2

PERFORMANCE CONSIDERATIONS IN SPARK STREAMING

Performance considerations

- Batch size
  - **500 milliseconds** is considered a good minimum size
  - Start with a large batch size (~10 seconds) and work down to a smaller batch size
    - If processing times remain consistent, explore decreasing the batch size
    - If the processing times increase? You have reached the limit
- Window size
  - Has a great impact on performance
  - Consider increasing this for expensive operations

Garbage collections and memory usage

- Cache RDDs in serialized form
  - Using Kryo for serialization reduces this even more
    - Reduces space for in-memory representations
- By default, Spark uses an in-memory cache
  - Can also evict RDDs older than a certain time-period
    - spark.cleaner.ttl
    - This preemptive eviction of RDDs also reduces the garbage collection pressure

Levels of parallelism in data receiving [1/4]

- Each input DStream creates a single receiver that receives a single stream of data
  - Receiving multiple data streams possible by creating multiple input DStreams
    - Each Dstream must be configured to receive different partitions of the data stream from the source(s)
- For a Kafka DStream receiving data on two topics?
  - Split into two DStreams each receiving one topic
    - Two receivers would run and receive data in parallel

Levels of parallelism in data receiving [2/4]

- Another approach is to tune the receiver's **block interval**
  - Determined by `spark.streaming.blockInterval`
- For most receivers, received data is **coalesced** into blocks of data before storing in memory
- The number of blocks in each batch determines the number of tasks used to process the received data in a map-like transformation
- Number of tasks per batch?
  - Batch interval/block interval

Levels of parallelism in data receiving [3/4]

- Number of tasks per batch?
  - Batch interval/block interval
- Block interval of 200 ms will create 10 tasks per 2 second batches
- If the number of tasks is too low?
  - All available cores might not be available to use all the data
- To increase number of tasks for a given batch interval?
  - Reduce the block interval

Levels of parallelism in data receiving [4/4]

- What if you did not want to receive data with multiple input streams?
  - Explicitly **repartition** the input data stream
- Repartitioning is done using the `inputStream.repartition(<number of partitions>))`
  - Distributes the received batches of data across the specified number of machines in the cluster *before* further processing

Data serialization [1/2]

- Data received through receivers is stored with StorageLevel.MEMORY_AND_DISK_SER_2
  - Data that does not fit in memory spills over to disk
- Input data and persisted RDDs generated by DStream transformations are automatically cleared
  - If you are using a window operation of 10 minutes, then Spark Streaming will keep the last 10 minutes of data, and actively throw away older data
  - Data can be retained for a longer duration by setting `streamingContext.remember`

Data serialization [2/2]

- RDDs generated by streaming computations may be persisted in memory
  - Persisted RDDs generated by streaming computations are persisted with StorageLevel.MEMORY_ONLY_SER
- If you are using batch intervals of a few seconds and no window operations?
  - You can try disabling serialization in persisted data
    - Reduce CPU overheads due to serialization, without excessive GC overheads.

PROCESSING TWITTER STREAMS USING SPARK

Spark-streaming example [1/5]

- Step-by-step approach to finding the top 10 hashtags from a stream of tweets using counts [Every second there is an output over data from the last 300 seconds]
- Step-1: Create a SparkStream context and Twitter credential setup

```
SparkConf sparkConf = new SparkConf().setAppName("Spark-
streaming-twitter-trends");

/*
Twitter authentication details … [Not included here]
*/
//JavaStreamigContext
JavaStreamingContext jssc =
      new JavaStreamingContext(sparkConf, new Duration(1000));

//Discretized stream of tweets
JavaDStream<Status> twitterStream = (JavaDStream<Status>)
TwitterUtils.createStream(jssc);
```


Spark-streaming example [2/5]

- Step-2: Map Input DStream of `Status` to `String`

```
//Discretized stream of Strings
JavaDStream<String> statuses = twitterStream.map(
    new Function<Status, String>() {
    public String call(Status status) {
        return status.getText();
    }
  }
);

statuses.print();

//trigger the execution of code
jssc.start();
jssc.awaitTermination();
```

Spark-streaming example [3/5]

- Step-3: Stream of hashtags from stream of tweets

```
//Tokenize words from status
JavaDStream<String> wordsFromStatuses = statuses.flatMap(
    new FlatMapFunction<String, String>() {
       public Iterable<String> call(String input) {
          return Arrays.asList(input.split(" "));
       }
    }
);

//Extract hashtags
JavaDStream<String> hashTags = wordsFromStatuses.filter(
    new Function<String, Boolean>() {
       public Boolean call(String word) {
          return word.startsWith("#");
       }
    }
);
```


Spark-streaming example [4/5]

- Step-4: Count the hashtag over 5 min window

```
//Mapping to tuple of (hashtag,1) in order to count
JavaPairDStream<String, Integer> hashtagtuples = hashTags.mapToPair(
new PairFunction<String, String, Integer>() {
   public Tuple2<String, Integer> call(String input) {
      return new Tuple2<String, Integer>(input, 1);
   }
});
//Aggregating over window of 5 min and silde of 1s
JavaPairDStream<String, Integer> counts =
hashtagtuples.reduceByKeyAndWindow(
   new Function2<Integer, Integer, Integer>() {
      public Integer call(Integer int1, Integer int2) {
         return int1 + int2;
      }
   },  new Function2<Integer, Integer, Integer>() {
          public Integer call(Integer int1, Integer int2) {
             return int1 - int2;
          }
   },  new Duration(60 * 5 * 1000), new Duration(1 * 1000));
```

Spark-streaming example [5/5]

- Step-5: Find top 10 hashtags according to counts

```
JavaPairDStream<Integer, String> swapCounts = counts.mapToPair(
new PairFunction<Tuple2<String, Integer>, Integer, String>() {
  public Tuple2<Integer, String> call(Tuple2<String, Integer> input)
    return input.swap();
  }});
JavaPairDStream<Integer, String> sortedCount = swapCounts.transformToPair(
new Function<JavaPairRDD<Integer, String>,JavaPairRDD<Integer, String>>(){
public JavaPairRDD<Integer, String> call(JavaPairRDD<Integer, String> input)
throws Exception {
  return input.sortByKey(false);
}});
sortedCount.foreach(new Function<JavaPairRDD<Integer, String>, Void> () {
  public Void call(JavaPairRDD<Integer, String> rdd) {
    String out = "\nTrending hashtags:\n";
    for (Tuple2<Integer, String> t: rdd.take(10)) {
      out = out + t.toString() + "\n";
}
System.out.println(out);
return null;}});
```


The contents of this slide-set are based on the following references

- *Learning Spark: Lightning-Fast Big Data Analysis. 1st Edition. Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. O'Reilly. 2015. ISBN-13: 978-1449358624.*
  [Chapter 10]
- Spark Streaming Programming Guide: http://spark.apache.org/docs/latest/streaming-programming-guide.html#memory-tuning
- Processing Twitter Streams using Spark: https://databricks-training.s3.amazonaws.com/realtime-processing-with-spark-streaming.html