Performance Tuning

Levels of Parallelism

- Example: Kafka streams can be split up, received and processed in parallel, and then recombined in spark.

```
numStreams = 5
kafkaStreams = [KafkaUtils.createStream(...) for _ in range (numStreams)]
unifiedStream = streamingContext.union(*kafkaStreams)
unifiedStream.pprint()
```

- Lagging in streaming or processing can be ameliorated by using under-utilized receivers and executors.

Data Serialization

- Input Data gets Serialized and gets stored in executors' memory with StorageLevel.MEMORY_AND_DISK_SER_2. This serialization obviously has overheads – the receiver must deserialize the received data and re-serialize it using Spark's serialization format.
- RDDs generated by streaming computations may be persisted in memory. RDDs generated by streaming computations are persisted with StorageLevel.MEMORY_ONLY_SER by default to minimize GC overheads.

```
from pyspark import StorageLevel
dir(StorageLevel)
rdd.persist(StorageLevel.MEMORY_ONLY_SER)
```

Memory Tuning

- Batch interval can have big effect on performance
  - App might be able to keep up with 2 second interval, but not 500 ms
- data received through receivers is stored with StorageLevel.MEMORY_AND_DISK_SER_2
  - If it doesn’t fit there, you need to cut down on memory usage
- Memory usage varies by transformation
  - updateStateByKey() can take up a lot, map() not so much
- Garbage collection can free up memory, but might cause some lags
  - Reconsider if you need ultra-low latency

Additional Techniques

```
conf.set("spark.rdd.compress", "true")
sc = SparkContext(appName="Name", conf=conf)
props.append("spark.rememberDuration","10")
```

- further reduction in memory usage can be achieved with `spark.rdd.compress`
- setting `streamingContext.remember` to specify how long to keep old data
- Use of the concurrent mark-and-sweep GC is strongly recommended for keeping GC-related pauses consistently low. Even though concurrent GC is known to reduce the overall processing throughput of the system, its use is still recommended to achieve more consistent batch processing times.

```
import gc
gc.collect()
```