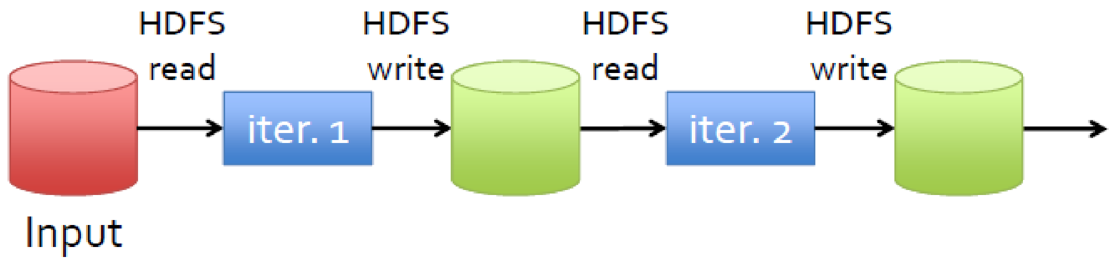
Spark介绍

为什么需要Spark

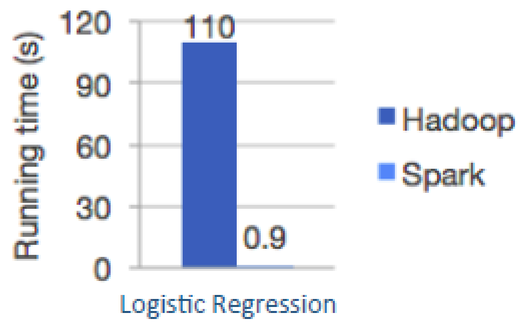
- Hadoop write intermediate data into HDFS



- Map/Reduce task launch overhead, so do Hive
  - Task launch delay 5 ~ 10 seconds
- No analysis function in Hbase, without Hive
- Low memory and CPU utilization, I/O is the bottle neck
- Can not support the ad-hoc queries

Spark与Hadoop MapReduce的对比

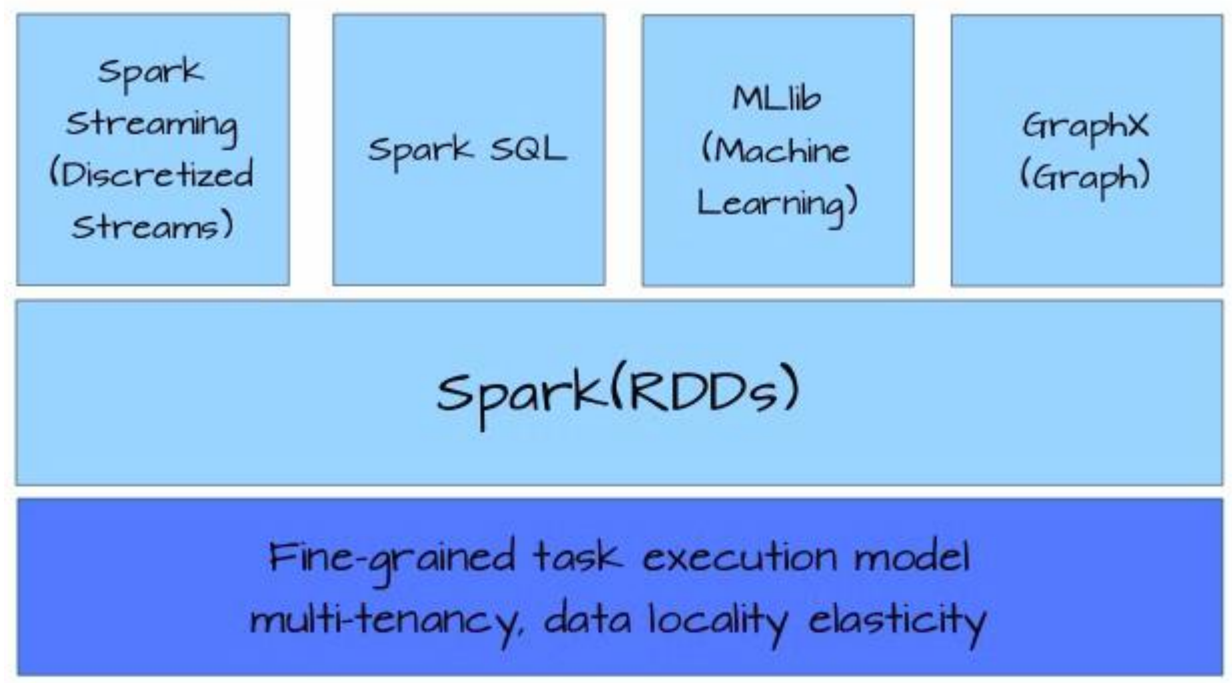
- Spark takes the concepts of MapReduce to the next level
  - Higher level API = faster, easier development
  - Low latency = near real-time processing
  - In-memory data storage = up to 100x performance improvement



```
sc.textFile(file) \
  .flatMap(lambda s: s.split()) \
  .map(lambda w: (w,1)) \
  .reduceByKey(lambda v1,v2: v1+v2)
  .saveAsTextFile(output)
```

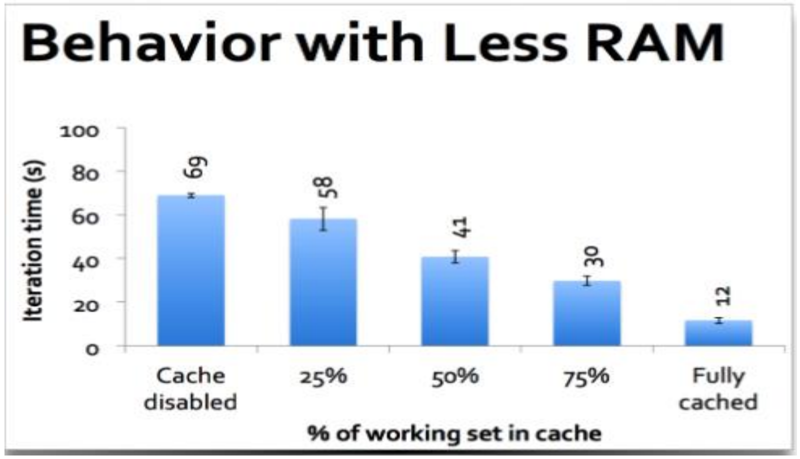
```
public class WordCount {
  public static void main(String[] args) throw
   Job job = new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}

public class WordMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
public void map(LongWritable key, Text value,
Context context) throws IOException, InterruptedException {
    String line = value.toString();
    for (String word : line.split("\\W+")) {
      if (word.length() > 0)
        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}

public class SumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
  public void reduce(Text key, Iterable<IntWritable>
  values, Context context) throws IOException, InterruptedException {
    int wordCount = 0;
    for (IntWritable value : values) {
      wordCount += value.get();
    }
    context.write(key, new IntWritable(wordCount));
  }
}
```

Spark的软件栈


Spark Streaming (Discretized Streams)
Spark SQL
MLlib (Machine Learning)
GraphX (Graph)
Spark(RDDs)
Fine-grained task execution model
multi-tenancy, data locality elasticity

支持的语言：Java, Python, Scala

内存对Spark的影响



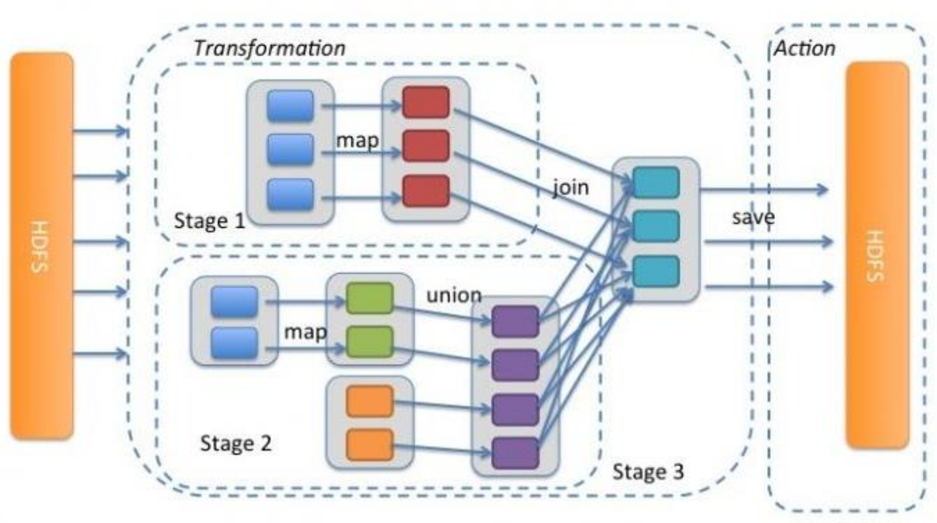
Performance without Memory

	Hadoop Record	Spark 100TB	Spark 1PB
Data Size	102.5TB	100TB	1000TB
Time	72 min	23 min	234 min
# Cores	50400	6592	6080
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Environment	Dedicate	Cloud (EC2)	Cloud (EC2)

RDD (Resilient Distributed Datasets)

特点

- 容错性
- 并行
- 迭代操作

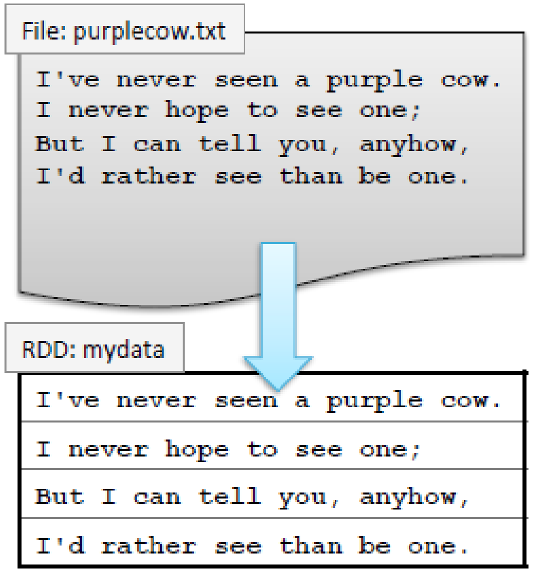


生成RDD

- 从文件生成
- 从来自内存的数据生成
- 从其它RDD

File-Based RDDs

- For file-based RDDS, use `SparkContext.textFile`
  - Accepts a single file, a wildcard list of files, or a comma-separated list of files
  - Examples
    - `sc.textFile("myfile.txt")`
    - `sc.textFile("mydata/*.log")`
    - `sc.textFile("myfile1.txt,myfile2.txt")`
  - Each line in the file(s) is a separate record in the RDD

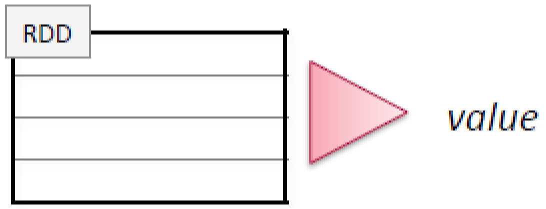


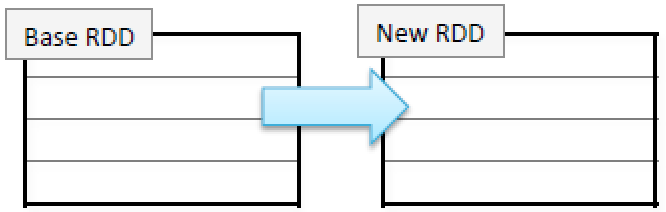
```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.

RDD Operations

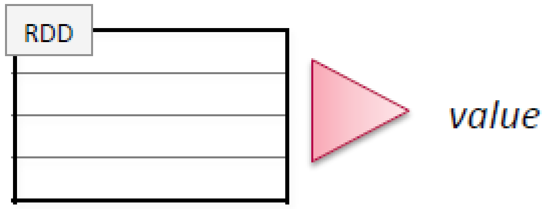
- Two types of RDD operations
  - Actions – return values
  - Transformations – define a new RDD based on the current one(s)





RDD Operations: Actions

- Some common actions
  - `count()` – return the number of elements
  - `take(`*n*`)` – return an array of the first *n* elements
  - `collect()` – return an array of all elements
  - `saveAsTextFile(`*file*`)` – save to text file(s)

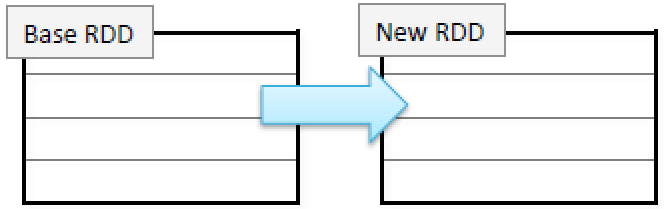


```
> mydata =
   sc.textFile("purplecow.txt")

> mydata.count()
4

> for line in mydata.take(2):
     print line
I've never seen a purple cow.
I never hope to see one;
```

```
> val mydata =
   sc.textFile("purplecow.txt")

> mydata.count()
4

> for (line <- mydata.take(2))
    println(line)
I've never seen a purple cow.
I never hope to see one;
```

RDD Operations: Operations

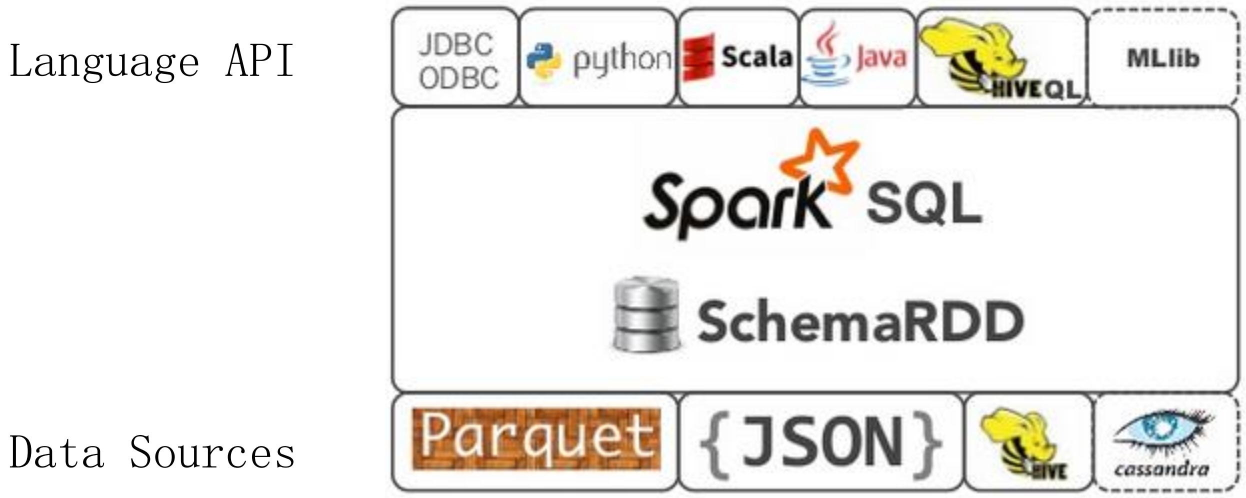
- **Transformations create a new RDD from an existing one**



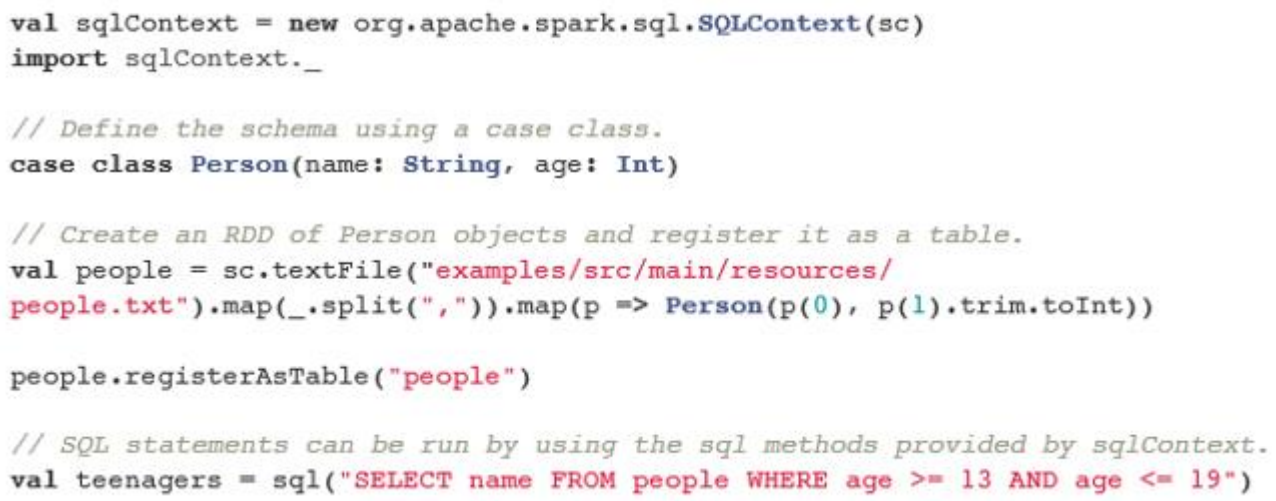
- **RDDs are immutable**
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
- **Some common transformations**
  - `map(`*`function`*`)` – creates a new RDD by performing a function on each record in the base RDD
  - `filter(`*`function`*`)` – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

Spark SQL: Relational Data Processing in Spark

- **Integrated** – Seamlessly mix SQL queries with Spark programs.
- **Unified Data Access** – Load and query data from a variety of sources.
- **Hive Compatibility** – Run unmodified Hive queries on existing warehouses.
- **Standard Connectivity** – Connect through JDBC or ODBC.



Query on Object

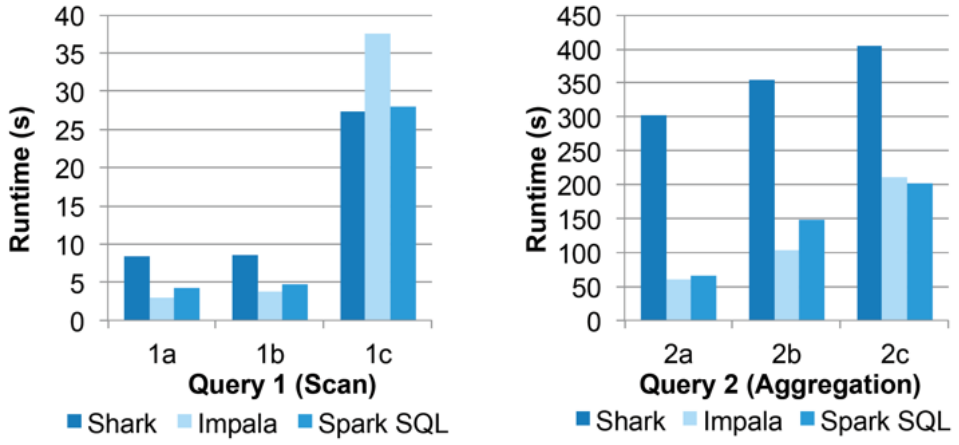


```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)

// Create an RDD of Person objects and register it as a table.
val people = sc.textFile("examples/src/main/resources/
people.txt").map(_.split(",")).map(p => Person(p(0), p(1).trim.toInt))

people.registerAsTable("people")

// SQL statements can be run by using the sql methods provided by sqlContext.
val teenagers = sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

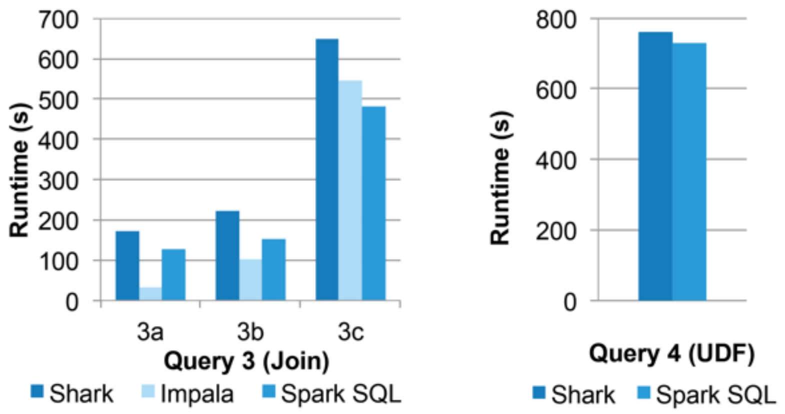
Spark SQL – Query in HiveQL

```
//val sc: SparkContext // An existing SparkContext.
//NB: example on laptop lacks a Hive MetaStore
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

// Importing the SQL context gives access to all the
// public SQL functions and implicit conversions.
import hiveContext._

hql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
hql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kvl.txt' INTO TABLE src")

// Queries are expressed in HiveQL
hql("FROM src SELECT key, value").collect().foreach(println)
```

Spark SQL Performance



110GB of data after columnar compression with Parquet

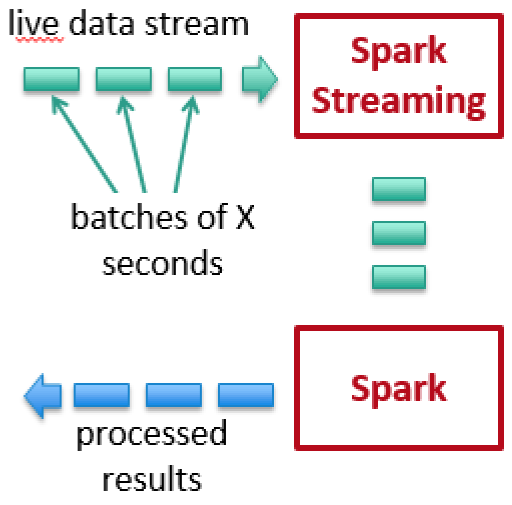


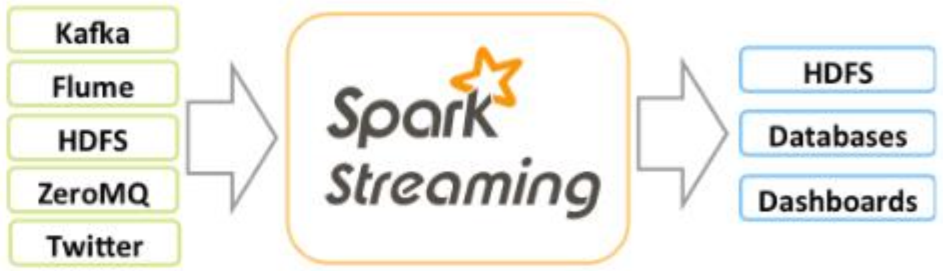
databricks

Spark Streaming

Run a streaming computation as a series of very small, deterministic batch jobs

- Batch sizes as low as ½ second, latency of about 1 second
- Potential for combining batch processing and streaming processing in the same system





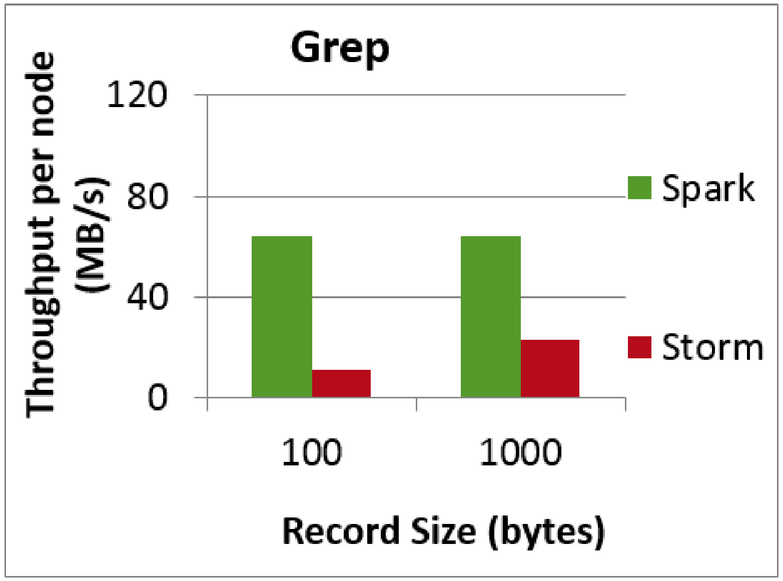
Spark Streaming Example

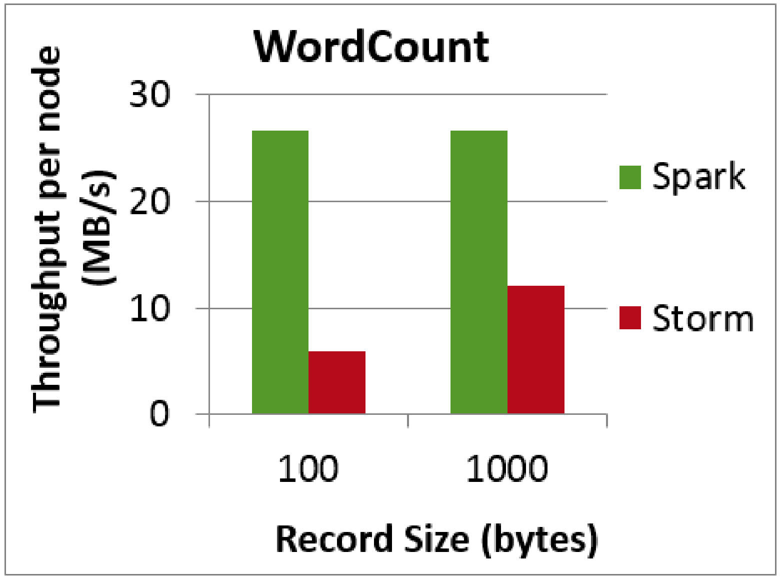
```
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

// Create a StreamingContext with a SparkConf configuration
val ssc = new StreamingContext(sparkConf, Seconds(10))

// Create a DStream that will connect to serverIP:serverPort
val lines = ssc.socketTextStream(serverIP, serverPort)

// Split each line into words
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
```

Comparison with Storm and S4

Higher throughput than Storm

- Spark Streaming: **670k** records/second/node
- Storm: **115k** records/second/node
- Apache S4: 7.5k records/second/node


Grep
Throughput per node (MB/s)
Record Size (bytes)
🟩 Spark
🟥 Storm


WordCount
Throughput per node (MB/s)
Record Size (bytes)
🟩 Spark
🟥 Storm