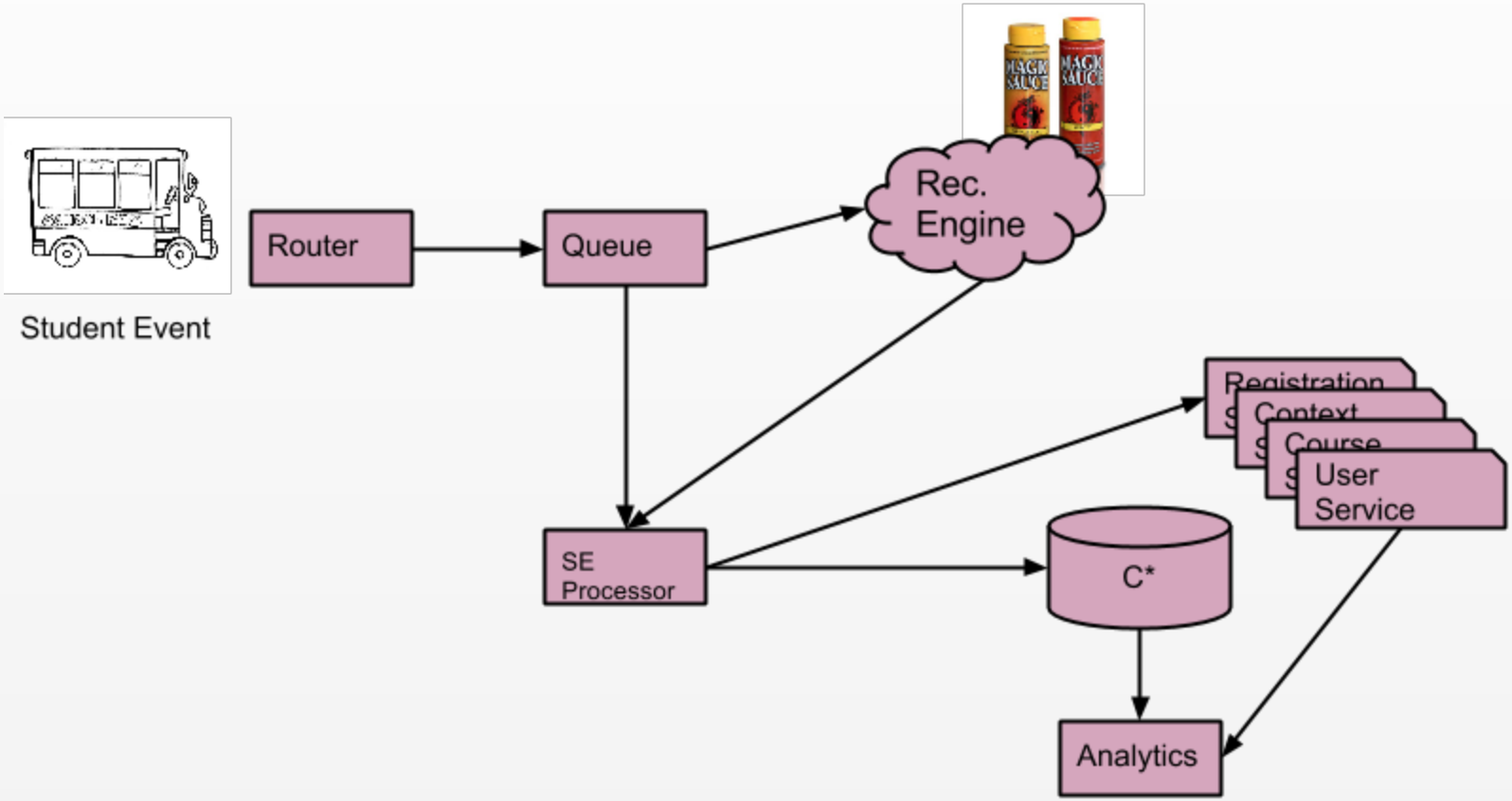
Hadoop and Cassandra

July 2013

Giannis Neokleous
www.giann.is
@yiannis_n

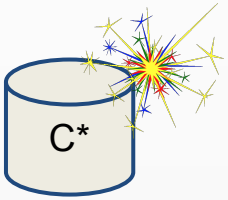
Cassandra at Knewton

- Student Events (At different processing stages)
- Parameters for models
- Course graphs
- Deployed in many environments ~ 14 clusters


Student Event
Router
Queue
Rec.
Engine
Registration
Context
Course
User
Service
SE
Processor
C*
Analytics

Getting data in and out of Cassandra in bulk efficiently

Why?



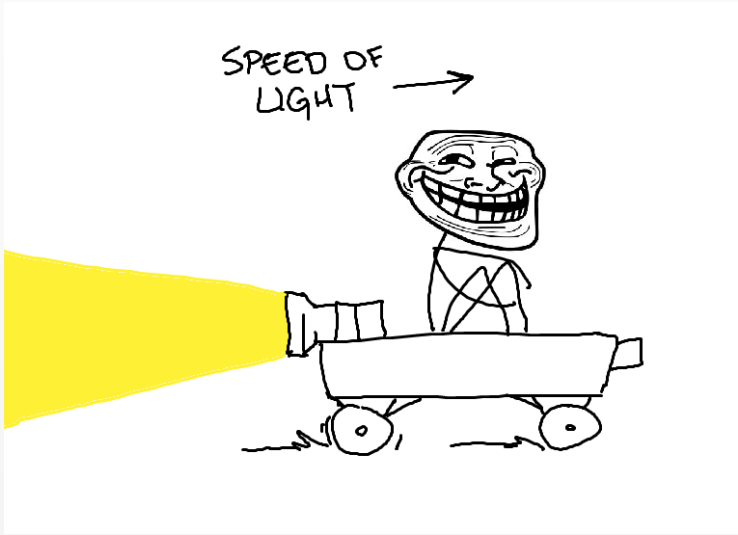
- Lots of data sitting in shiny new clusters
  - Want to run Analytics
- You suddenly realize your schema is not so great
- The data you're storing could be more efficient
- Think you've discovered an awesome new metric

Stuck!

How do you get data out

efficiently and fast?

No slow-downs?



Solutions

- Cassandra comes packaged with sstable2json tool.
- Using the thrift API for bulk mutations, gets.
  - Can distribute reads or writes to multiple machines.
- ColumnFamily[Input|Output]Format - Using Hadoop
  - Needs a live cluster
  - Still uses the thrift API


hadoop

+


Cassandra

Why is MapReduce a good fit for C?*

- SSTables are sorted
  - MapReduce likes sorted stuff
- SSTables are immutable
  - Easy to identify what has been processed
- Data is essentially key/value pairs
- MapReduce can partition stuff
  - Just like you partition data in your Cassandra cluster
- MapReduce is Cool, so is Cassandra

Does it work?

Yes! But where?

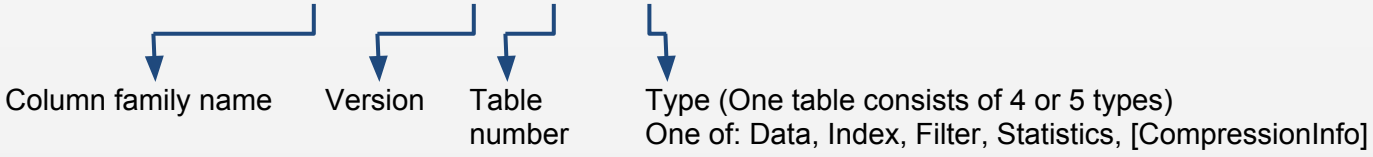
- Been using bulk reading in production for a few months now
  - *Works great!*
- Been using bulk writing into Cassandra for almost two years
  - *Works great too!*

How?!!1

Reading in bulk

A little about SSTables

- Sorted
  - Both row keys and columns
- Key Value pairs
  - Rows:
    - Row value: ***Key***
    - Columns: ***Value***
  - Columns:
    - Column name: ***Key***
    - Column value: ***Value***
- Immutable
- Consist of 4 parts
  - ColumnFamily-hd-3549-Data.db



A little about MapReduce

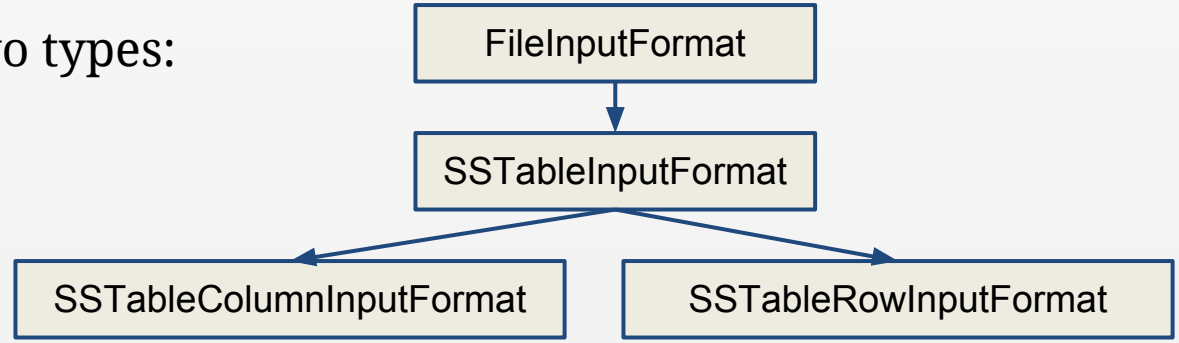
- InputFormat
  - Figure out where the data is, what to read and how to read them
  - Divides the data to record readers
- RecordReader
  - Instantiated by InputFormats
  - Do the actual reading
- Mapper
  - Key/Value pairs get passed in by the record readers
- Reducer
  - Key/Value pairs get passed in from the mappers
  - All the same keys end up in the same reducer

A little about MapReduce

- OutputFormat
  - Figure out where and how to write the data
  - Divides the data to record writers
  - What to do after the data has been written
- RecordWriter
  - Instantiated by OutputFormats
  - Do the actual writing

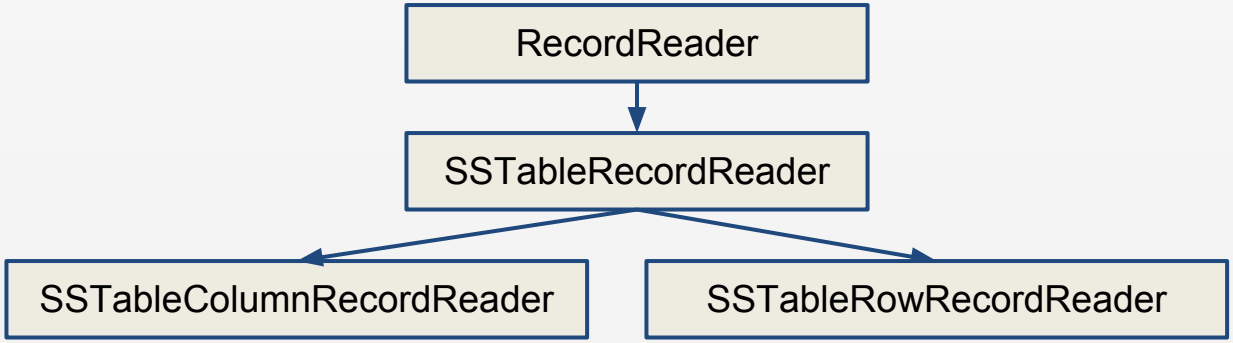
SSTableInputFormat

- An input format specifically for SSTables.
  - Extends from FileInputFormat
- Includes a `DataPathFilter` for filtering through files for *-Data.db files
- Expands all subdirectories of input - Filters for ColumnFamily
- Configures Comparator, Subcomparator and Partitioner classes used in ColumnFamily.
- Two types:

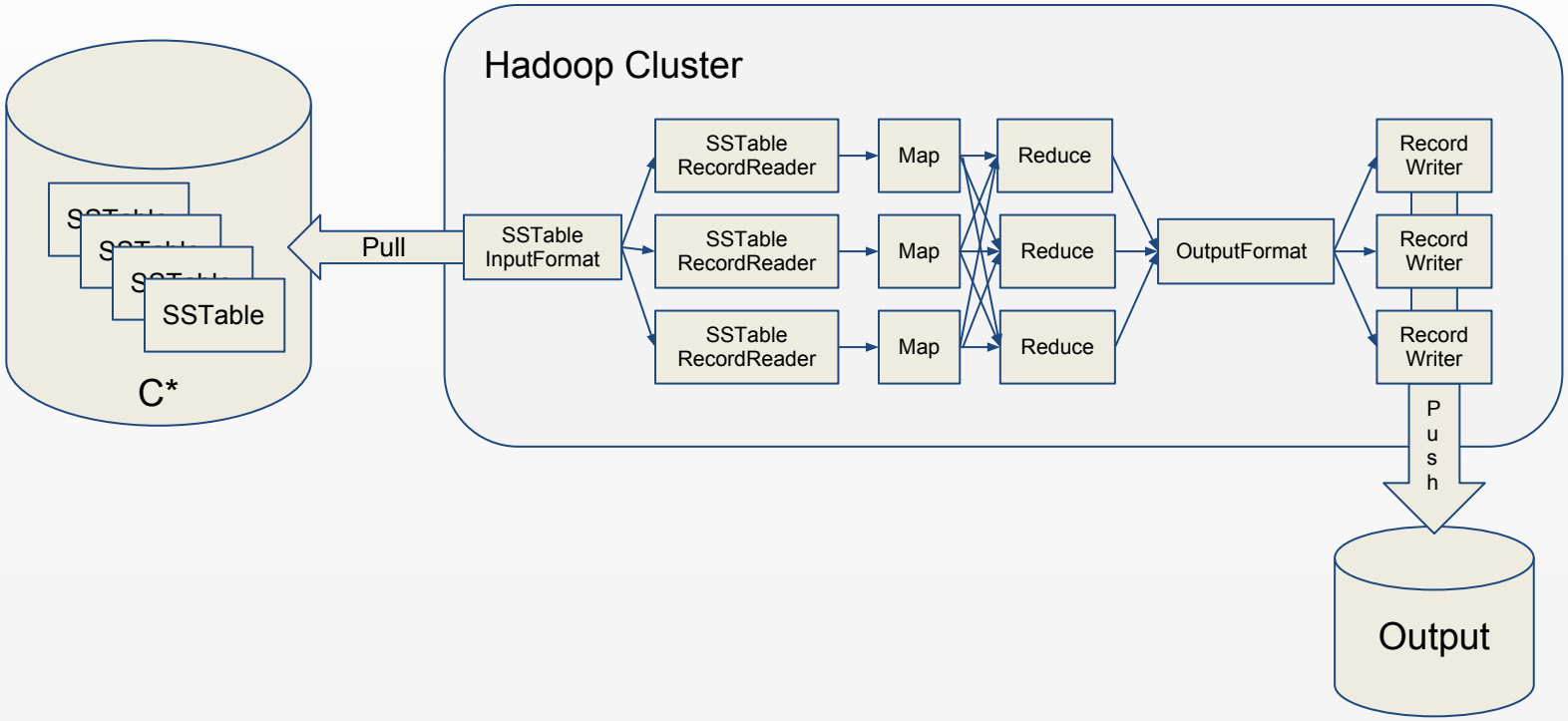


SSTableRecordReader

- A record reader specifically for SSTables.
- On init:
  - Copies the table locally. (Decompresses it, if using Priam)
  - Opens the table for reading. (Only needs Data, Index and CompressionInfo tables)
  - Creates a TableScanner from the reader
- Two types:



Data Flow


Hadoop Cluster
SSTable
C*
Pull
SSTable
InputFormat
SSTable
RecordReader
SSTable
RecordReader
SSTable
RecordReader
Map
Map
Map
Reduce
Reduce
Reduce
OutputFormat
Record
Writer
Record
Writer
Record
Writer
Push
Output

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf);
    ClassLoader loader = SSTableMRExample.class.getClassLoader();
    conf.addResource(loader.getResource("knewton-site.xml"));
    SSTableInputFormat.setPartitionerClass(RandomPartitioner.class.getName(), job);
    SSTableInputFormat.setComparatorClass(LongType.class.getName(), job);
    SSTableInputFormat.setColumnFamilyName("StudentEvents", job);

    job.setOutputKeyClass(LongWritable.class);
    job.setOutputValueClass(StudentEventWritable.class);
    job.setMapperClass(StudentEventMapper.class);
    job.setReducerClass(StudentEventReducer.class);

    job.setInputFormatClass(SSTableColumnInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    SSTableInputFormat.addInputPaths(job, args[0]);
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
}
```

Load additional properties from a conf file

Define mappers/reducers. The only thing you have to write.

Read each column as a separate record.

```
public class StudentEventMapper extends SSTableColumnMapper<Long, StudentEvent, LongWritable, StudentEventWritable> {
  @Override
  public void performMapTask(Long key, StudentEvent value, Context context) {
        //do stuff here
  }
  // Some other omitted trivial methods
}
```

Sees row key/column pairs.
Remember to skip deleted columns (tombstones)

Replication factor

- Data replication is a thing
- Have to deal with it:
  - In the reducer
  - Only read (num tokens) / (replication factor) - if you're feeling brave

Priam

- Incremental backups
  - No need to read everything all the time
- Priam usually snappy compresses tables
- Works good if you want to use EMR
  - Backups already on S3

Writing in bulk

Writing in bulk

- Define custom output format
- Define custom record writer
  - Uses the SSTableSimpleWriter
    - Expects keys in sorted order (Tricky with MapReduce - More about this later)
- Nothing special on the Mapper or Reducer part

What happens in the OutputFormat?

- Not much…
  - Instantiates and does basic configuration for RecordWriter

```
public abstract RecordWriter<K, V> getRecordWriter(TaskAttemptContext context)
        throws IOException, InterruptedException;
```

What happens in the RecordWriter?

- Writes Columns, ExpiringColumns (ttl), CounterColumns, SuperColumns
  - With the right abstractions you can reuse almost all of the code in multiple Jobs
- On close SSTables written by the record writer get ***sent**** to Cassandra

```
public SSTableSimpleWriter(File directory,
        CFMetaData metadata, IPartitioner partitioner)
protected void writeRow(DecoratedKey key,
        ColumnFamily columnFamily)
private void addColumn(Column column)
```

```
public class RecordWriter<K, V> {
    public abstract void write(K key, V value)
        throws IOException, InterruptedException;
    public abstract void close(TaskAttemptContext context)
        throws IOException, InterruptedException;
    .....
}
```

How exactly do you send the SSTables to Cassandra?

How do SSTables get sent? - Part I

- sstableloader introduced in 0.8 using the BulkLoader class
  - Starts a gossiper occupies ports
  - Needs coordination - Locking
- Not convenient to incorporate the BulkLoader class in the code
- Gossiper connects the sender to the cluster
  - Not part of the ring
  - Bug in 0.8 persisted the node in the cluster

How do SSTables get sent? - Part II

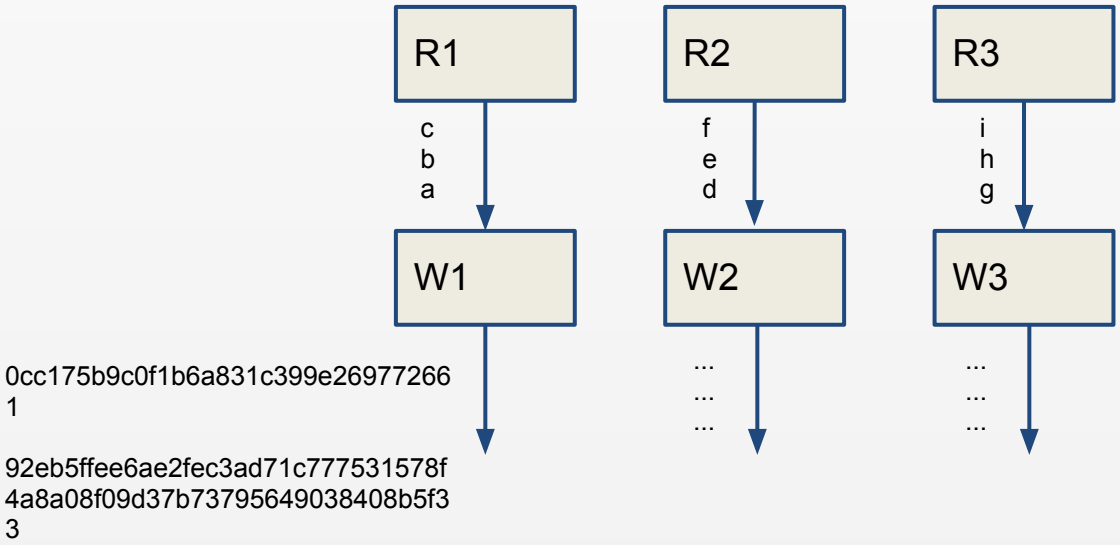
- Smart partitioning in Hadoop, then scp
  - No Gossiper
  - No coordination
  - Each reducer is responsible for handling keys specific to 1 node in the ring.
- Needs ring information beforehand
  - Can be configured
    - Offline in conf
    - Right before the job starts

Decorated Keys

- Keys are decorated before they're stored.
  - Faster to work with - Compares, sorts etc.
  - RandomPartitioner uses MD5 hash.
- Depends on your partitioner.
  - Random Partitioner
  - OrderPreservingPartitioner
  - etc? custom?
- When reading the `SSTableScanner` de-decorates keys.
- Tricky part is when writing tables.

Decorated Keys

- Columns and keys are sorted - After they're decorated.
- Don't partition your keys in MapReduce before you decorate them.
  - Unless you're using the unsorted table writer.

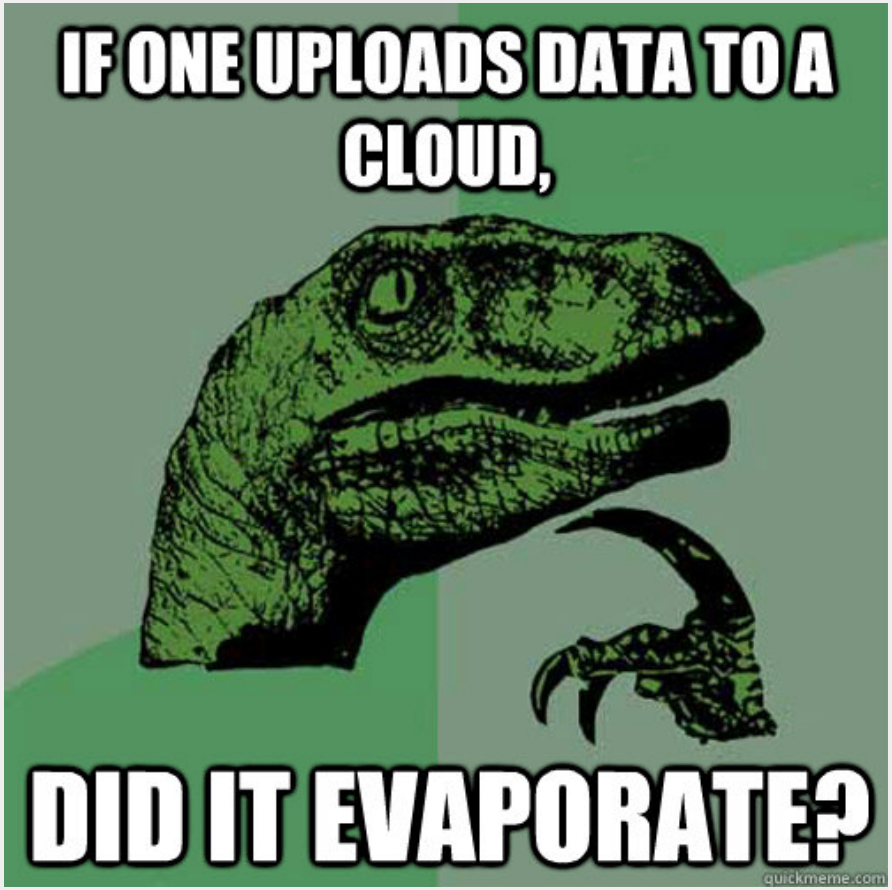


KassandraMRHelper

- Open sourcing today!
  - github.com/knewton/KassandraMRHelper
- Has all you need to get started on bulk reading SSTables with Hadoop.
- Includes an example job that reads "student events"
- Handles compressed tables
- Use Priam? Even better can snappy decompress priam backups.
- Don't have a cluster up or a table handy?
  - Use `com.knewton.mapreduce.cassandra.WriteSampleSSTable` in the test source directory to generate one.

```
usage: WriteSampleSSTable [OPTIONS] <output_dir>
 -e,--studentEvents <arg>   The number of student events per student to be
                            generated. Default value is 10
 -h,--help                  Prints this help message.
 -s,--students <arg>        The number of students (rows) to be generated.
                            Default value is 100.
```

Thank you



Giannis Neokleous
www.giann.is
@yiannis_n

Questions?