

Using Hadoop: Best Practices

Casey Stella

January 22, 2012

Table of Contents

Introduction

Designing a Map Reduce Algorithm

Performance

Testing

Debugging

Conclusion

Introduction

- ▶ Hi, I'm Casey

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician
- ▶ Caveat emptor
 - ▶ Not all problems can be solved with Map Reduce
 - ▶ This is a batch processing system (i.e. not realtime)

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician
- ▶ Caveat emptor
 - ▶ Not all problems can be solved with Map Reduce
 - ▶ This is a batch processing system (i.e. not realtime)
- ▶ I'm going to talk about some of the best practices that I've seen

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician
- ▶ Caveat emptor
 - ▶ Not all problems can be solved with Map Reduce
 - ▶ This is a batch processing system (i.e. not realtime)
- ▶ I'm going to talk about some of the best practices that I've seen
 - ▶ Some of these are common knowledge

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician
- ▶ Caveat emptor
 - ▶ Not all problems can be solved with Map Reduce
 - ▶ This is a batch processing system (i.e. not realtime)
- ▶ I'm going to talk about some of the best practices that I've seen
 - ▶ Some of these are common knowledge
 - ▶ Some of these don't show up until you've been up 'til 3AM debugging a problem.

Introduction

- ▶ Hi, I'm Casey
 - ▶ I work at Explorys
 - ▶ I work with Hadoop and the Hadoop ecosystem daily
 - ▶ I am a recovering Mathematician
- ▶ Caveat emptor
 - ▶ Not all problems can be solved with Map Reduce
 - ▶ This is a batch processing system (i.e. not realtime)
- ▶ I'm going to talk about some of the best practices that I've seen
 - ▶ Some of these are common knowledge
 - ▶ Some of these don't show up until you've been up 'til 3AM debugging a problem.
- ▶ These are my opinions and not necessarily the opinions of my employer.

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase
 - ▶ Serialize your objects tightly (e.g. not using Java Serialization)

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase
 - ▶ Serialize your objects tightly (e.g. not using Java Serialization)
 - ▶ Key/values emitted from the map phase had better be linear with a **small** constant..preferably below 1

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase
 - ▶ Serialize your objects tightly (e.g. not using Java Serialization)
 - ▶ Key/values emitted from the map phase had better be linear with a **small** constant..preferably below 1
- ▶ Strategies
 - ▶ Intelligent use of the combiners

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase
 - ▶ Serialize your objects tightly (e.g. not using Java Serialization)
 - ▶ Key/values emitted from the map phase had better be linear with a **small** constant..preferably below 1
- ▶ Strategies
 - ▶ Intelligent use of the combiners
 - ▶ Use Local Aggregation in the mapper to emit a more complex value.

Performance considerations

- ▶ Setup and teardown costs, so keep the HDFS block size large
- ▶ Mappers, Reducers and Combiners have memory constraints
- ▶ Transmission costs dearly
 - ▶ Use Snappy, LZO, or (soon) LZ4 compression at every phase
 - ▶ Serialize your objects tightly (e.g. not using Java Serialization)
 - ▶ Key/values emitted from the map phase had better be linear with a **small** constant..preferably below 1
- ▶ Strategies
 - ▶ Intelligent use of the combiners
 - ▶ Use Local Aggregation in the mapper to emit a more complex value.
 - ▶ Ensure that all components of your keys are necessary in the sorting logic. If any are not, push them into the value.

Unit/Integration Testing Methodologies

- ▶ First off, do it.

Unit/Integration Testing Methodologies

- ▶ First off, do it.
- ▶ Unit test individual mappers, reducers, combiners and partitioners
 - ▶ Actual unit tests. This will help debugging, I promise.
 - ▶ Design components so that dependencies can be injected via polymorphism when testing

Unit/Integration Testing Methodologies

- ▶ First off, do it.
- ▶ Unit test individual mappers, reducers, combiners and partitioners
 - ▶ Actual unit tests. This will help debugging, I promise.
 - ▶ Design components so that dependencies can be injected via polymorphism when testing
- ▶ Minimally verify that keys
 - ▶ Can be serialized and deserialized
 - ▶ *hashCode()* is sensible (Remember: the *hashCode()* for enum is not stable)
 - ▶ *compareTo()* is reflexive, symmetric and jives with *equals()*

Unit/Integration Testing Methodologies

- ▶ First off, do it.
- ▶ Unit test individual mappers, reducers, combiners and partitioners
 - ▶ Actual unit tests. This will help debugging, I promise.
 - ▶ Design components so that dependencies can be injected via polymorphism when testing
- ▶ Minimally verify that keys
 - ▶ Can be serialized and deserialized
 - ▶ *hashCode()* is sensible (Remember: the *hashCode()* for enum is not stable)
 - ▶ *compareTo()* is reflexive, symmetric and jives with *equals()*
- ▶ Integration test via single user mode hadoop if you like, but I think it's mostly pretty useless.

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR
 - ▶ If not, then sample the dataset and verify with R, Python or whatever.

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR
 - ▶ If not, then sample the dataset and verify with R, Python or whatever.
- ▶ Do outlier analysis and thresholding based QA

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR
 - ▶ If not, then sample the dataset and verify with R, Python or whatever.
- ▶ Do outlier analysis and thresholding based QA
- ▶ Data analysis is hard and often requires specialized skills
 - ▶ Enter a new breed: the data scientist

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR
 - ▶ If not, then sample the dataset and verify with R, Python or whatever.
- ▶ Do outlier analysis and thresholding based QA
- ▶ Data analysis is hard and often requires specialized skills
 - ▶ Enter a new breed: the data scientist
 - ▶ Stats + Computer Science + Domain knowledge

Quality Assurance Testing

- ▶ The output of processing large amounts of data is often large
- ▶ Verify statistical properties via MR
 - ▶ If statistical tests fit within Map Reduce, then use MR
 - ▶ If not, then sample the dataset and verify with R, Python or whatever.
- ▶ Do outlier analysis and thresholding based QA
- ▶ Data analysis is hard and often requires specialized skills
 - ▶ Enter a new breed: the data scientist
 - ▶ Stats + Computer Science + Domain knowledge
 - ▶ Often not a software engineer

Debugging Methodologies

- ▶ Better to catch it at the unit test level

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling
 - ▶ Take the data and integrate it into a unit test.

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling
 - ▶ Take the data and integrate it into a unit test.
- ▶ **DO NOT**
 - ▶ Use print statements to debug unless you're sure of the scope.

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling
 - ▶ Take the data and integrate it into a unit test.
- ▶ **DO NOT**
 - ▶ Use print statements to debug unless you're sure of the scope.
 - ▶ Use counters where the group or name count grows more than a fixed amount.

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling
 - ▶ Take the data and integrate it into a unit test.
- ▶ **DO NOT**
 - ▶ Use print statements to debug unless you're sure of the scope.
 - ▶ Use counters where the group or name count grows more than a fixed amount.
- ▶ **DO**
 - ▶ Use a single counter in the actual job if the job doesn't finish

Debugging Methodologies

- ▶ Better to catch it at the unit test level
- ▶ If you can't, I suggest the following technique
 - ▶ Investigatory map reduce job to find the data causing the issue.
 - ▶ Single point if you're lucky, if not then a random sample using reservoir sampling
 - ▶ Take the data and integrate it into a unit test.
- ▶ **DO NOT**
 - ▶ Use print statements to debug unless you're sure of the scope.
 - ▶ Use counters where the group or name count grows more than a fixed amount.
- ▶ **DO**
 - ▶ Use a single counter in the actual job if the job doesn't finish
 - ▶ Use a map reduce job that outputs suspect input data into HDFS

Sample Code

- ▶ I've created a simple Hadoop project in the NLP domain to illustrate some of these points
- ▶ Implements a Map Reduce algorithm that analyzes the senatorial speeches and generates the most statistically important words used, comparing across multiple political orientations
- ▶ It features
 - ▶ Full example of a Map Reduce algorithm that is not word count
 - ▶ Unit tests for the Keys and Values
 - ▶ Integration test that executes the full lifecycle
- ▶ <https://github.com/cestella/SenatorialSpeechInvestigation>

Conclusion

- ▶ Thanks for your attention
- ▶ Follow me on twitter @casey_stella
- ▶ Find me at
 - ▶ <http://caseystella.com>
 - ▶ <https://github.com/cestella>