Spindle - CloudCom 2014

Next-Generation Query Processing for Adobe Marketing Cloud

Brandon Amos* and David Tompkins, **Adobe Research**

*Adobe summer intern, Ph.D. Student at Carnegie Mellon University.

Presentation available online at
`http://adobe-research.github.io/spindle/pres`

2014/12/19

Motivation

- Adobe Marketing Cloud delivers marketers a web analytics platform with subsecond query response time.

Motivation

- Adobe Marketing Cloud delivers marketers a web analytics platform with subsecond query response time.
- Trending open source technologies such as *Apache Spark*, *Cloudera Impala*, and *Google Dremel* offer a general purpose alternative to in-house software at Adobe.

Motivation

- Adobe Marketing Cloud delivers marketers a web analytics platform with subsecond query response time.
- Trending open source technologies such as *Apache Spark*, *Cloudera Impala*, and *Google Dremel* offer a general purpose alternative to in-house software at Adobe.
- **Spindle** is an early investigation of the feasibility of Apache Spark for web analytics.

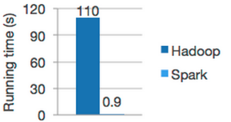
Motivation

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

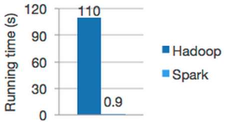
Motivation

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

Problem: Current performance studies do not show Spark's performance for interactive web analytics application.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- Spark is a trending distributed computing environment and favors in-memory processing.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- Spark is a trending distributed computing environment and favors in-memory processing.
- Documentation is currently biased towards writing Spark applications and does not provide best practices for running on a cluster.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- Spark is a trending distributed computing environment and favors in-memory processing.
- Documentation is currently biased towards writing Spark applications and does not provide best practices for running on a cluster.
- Spark provides 4 deployment modes: EC2, standalone, Mesos, and YARN.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- Spark is a trending distributed computing environment and favors in-memory processing.
- Documentation is currently biased towards writing Spark applications and does not provide best practices for running on a cluster.
- Spark provides 4 deployment modes: EC2, standalone, Mesos, and YARN.
- **Motivation:** Spark's standalone cluster mode seems simplest to configure and use, but requires files to be synchronized across the cluster, including the Spark installation directory.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- *Open source* ***Puppet*** *is a flexible, customizable framework available under the Apache 2.0 license designed to help system administrators automate the many repetitive tasks they regularly perform.*

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- *Open source **Puppet** is a flexible, customizable framework available under the Apache 2.0 license designed to help system administrators automate the many repetitive tasks they regularly perform.*
- ***Fabric** is a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.*

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- *Open source **Puppet** is a flexible, customizable framework available under the Apache 2.0 license designed to help system administrators automate the many repetitive tasks they regularly perform.*
- ***Fabric** is a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.*
- Organizations have open-sourced Puppet projects for HDFS and Spark on GitHub to manage the server configurations.

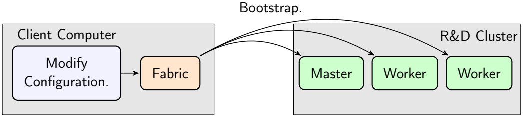
System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- *Open source **Puppet** is a flexible, customizable framework available under the Apache 2.0 license designed to help system administrators automate the many repetitive tasks they regularly perform.*
- ***Fabric** is a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.*
- Organizations have open-sourced Puppet projects for HDFS and Spark on GitHub to manage the server configurations.
- Use **Fabric** to synchronize and deploy to the servers, which simplifies configuration changes by keeping all configuration locally.

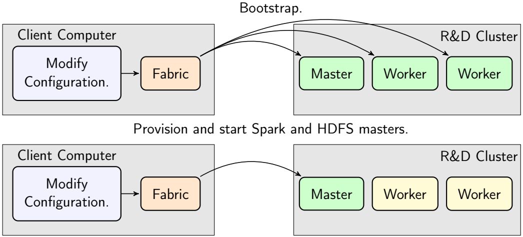
System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment



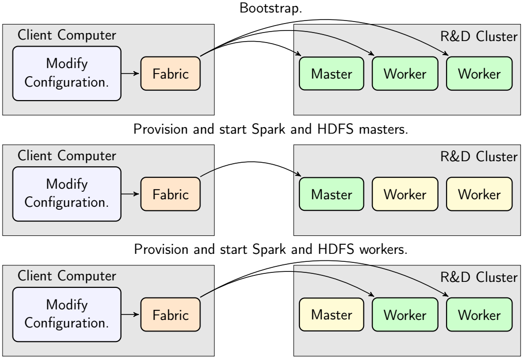
System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment



System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment



System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:
  - Application JAR files need to be synchronized across every node on the cluster.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:
  - Application JAR files need to be synchronized across every node on the cluster.
  - Does not provide a way to obtain application output.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:
  - Application JAR files need to be synchronized across every node on the cluster.
  - Does not provide a way to obtain application output.
- **Solution:** Utilize Fabric to deploy, get output, and kill jobs on a Spark cluster.

System Architecture

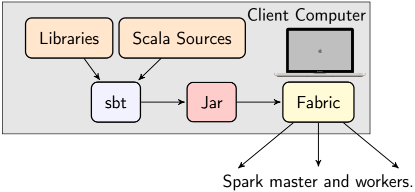
Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:
  - Application JAR files need to be synchronized across every node on the cluster.
  - Does not provide a way to obtain application output.
- **Solution:** Utilize Fabric to deploy, get output, and kill jobs on a Spark cluster.

System Architecture

Mini-Project: Deploying Spark. github/adobe-research/spark-cluster-deployment

- **Problem:** Deploying applications to a Spark cluster:
  - Application JAR files need to be synchronized across every node on the cluster.
  - Does not provide a way to obtain application output.
- **Solution:** Utilize Fabric to deploy, get output, and kill jobs on a Spark cluster.



System Architecture

Requests and responses with Spray.

Motivation: How can we use expose our analytics engine to other components of Adobe's infrastructure?

System Architecture

Requests and responses with Spray.

Motivation: How can we use expose our analytics engine to other components of Adobe's infrastructure?

System Architecture

Requests and responses with Spray.

Motivation: How can we use expose our analytics engine to other components of Adobe's infrastructure?

Solution: ***spray*** *is an open-source toolkit for building REST/HTTP-based integration layers on top of Scala and Akka Actors. Being asynchronous, actor-based, fast, lightweight, modular and testable it's a great way to connect your Scala applications to the world.*

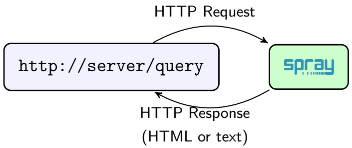
System Architecture

Requests and responses with Spray.

`http://server/query`

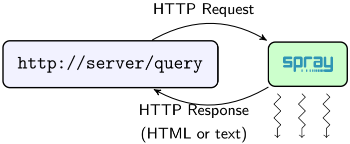
System Architecture

Requests and responses with Spray.



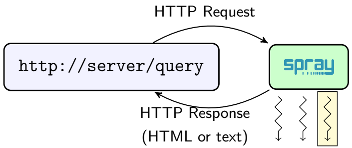
System Architecture

Requests and responses with Spray.



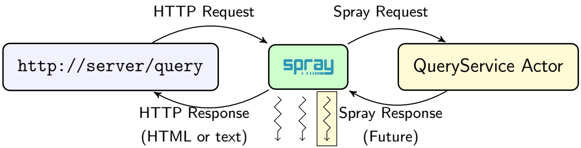
System Architecture

Requests and responses with Spray.



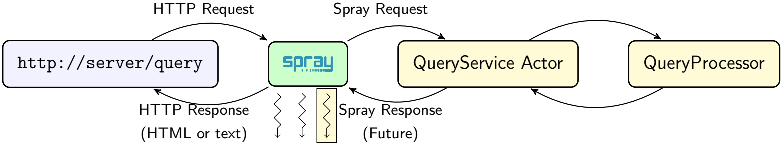
System Architecture

Requests and responses with Spray.



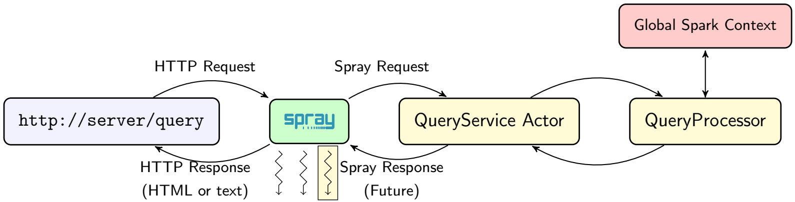
System Architecture

Requests and responses with Spray.



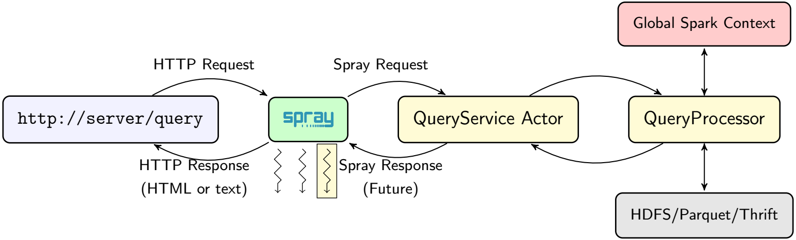
System Architecture

Requests and responses with Spray.



System Architecture

Requests and responses with Spray.



System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

- **Motivation:** Adobe's web analytics data is sparse and has approximately 250 columns, and queries implemented in Spindle use less than ten columns.

System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

- **Motivation:** Adobe's web analytics data is sparse and has approximately 250 columns, and queries implemented in Spindle use less than ten columns.
- **Solution:** Store the data in columnar store databases separated by day with **Thrift** and **Parquet** on **HDFS**.

System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

- ***Apache Thrift*** *allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages.*

System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

- ***Apache Thrift*** *allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages.*
- *We created* ***Parquet*** *to make the advantages of compressed, efficient columnar data representation available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model, or programming language.*

System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

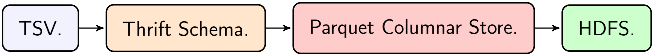
- Load each day into a Parquet database on HDFS.

TSV. → Thrift Schema. → Parquet Columnar Store. → HDFS.

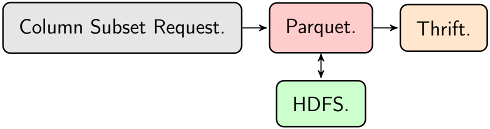
System Architecture

Mini-Project: Columnar data store. github/adobe-research/spark-parquet-thrift-example

- Load each day into a Parquet database on HDFS.



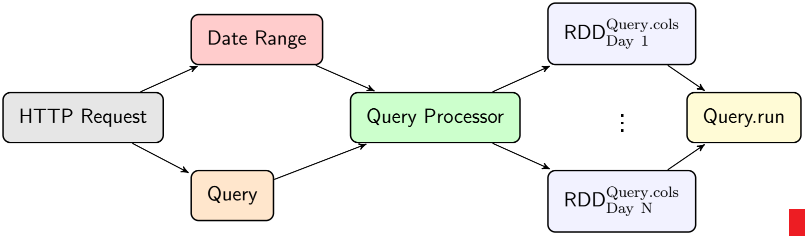
- Retrieve data by specifying column subsets.



System Architecture

Mini-Project: Columnar data store.

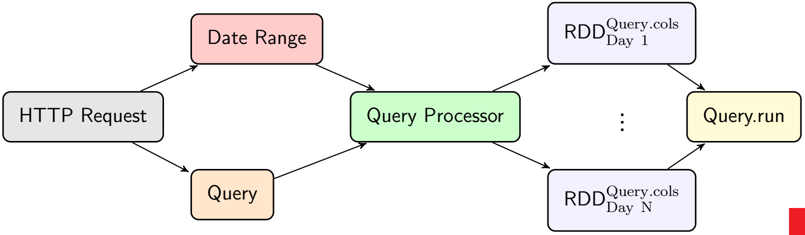
- Load RDD's into a Scala `Seq`.



System Architecture

Mini-Project: Columnar data store.

- Load RDD's into a Scala `Seq`.
- Only works well for a single timezone of data.



System Architecture

Simple caching policy.

- **Motivation:** Loading gigabytes or terabytes of data is a performance bottleneck for many applications, and web analytics queries are likely to be on the same data set.

System Architecture

Simple caching policy.

- **Motivation:** Loading gigabytes or terabytes of data is a performance bottleneck for many applications, and web analytics queries are likely to be on the same data set.
- **Possible Solution:** ***Tachyon*** *is a memory-centric distributed file system that runs on top of HDFS and caches working set files in memory. Tachyon is Hadoop compatible and can be used with Spark or MapReduce programs without any code change.*

System Architecture

Simple caching policy.

```
Tachyon Summary

      Started:                 07-09-2014 17:47:55:462
      Uptime:        0 day(s), 23 hour(s), 24 minute(s), and 3 second(s)
      Version:                          0.4.1
  Running Workers:                        5

Cluster Usage Summary

     Memory Capacity:        5120.00 MB
   Memory Free / Used:    5120.00 MB / 0.00 B
    UnderFS Capacity:        5258.67 GB
   UnderFS Free / Used:  4978.05 GB / 4508.00 KB
```

System Architecture

Simple caching policy.

```
Tachyon Summary

      Started:            07-09-2014 17:47:55:462
      Uptime:      0 day(s), 23 hour(s), 24 minute(s), and 3 second(s)
      Version:                     0.4.1
   Running Workers:                  5

Cluster Usage Summary

     Memory Capacity:         5120.00 MB
   Memory Free / Used:    5120.00 MB / 0.00 B
    UnderFS Capacity:         5258.67 GB
   UnderFS Free / Used: 4978.05 GB / 4508.00 KB
```

- Parquet optimizes disk accesses and only reads specific blocks while loading files and never entire file. However, Tachyon 0.4.1 cannot cache random file block accesses and has the severe limitation of only caching files when all blocks are read.

System Architecture

Simple caching policy.

```
Tachyon Summary

      Started:              07-09-2014 17:47:55:462
      Uptime:       0 day(s), 23 hour(s), 24 minute(s), and 3 second(s)
      Version:                      0.4.1
  Running Workers:                    5

Cluster Usage Summary

    Memory Capacity:          5120.00 MB
  Memory Free / Used:     5120.00 MB / 0.00 B
   UnderFS Capacity:          5258.67 GB
  UnderFS Free / Used: 4978.05 GB / 4508.00 KB
```

- Parquet optimizes disk accesses and only reads specific blocks while loading files and never entire file. However, Tachyon 0.4.1 cannot cache random file block accesses and has the severe limitation of only caching files when all blocks are read.
- Confirmed by emailing Tachyon's mailing list.

System Architecture

Simple caching policy.

- **Workaround Solution:** Provide a `cache` parameter that saves RDD's in memory between query calls.

System Architecture

Simple caching policy.

- **Workaround Solution:** Provide a `cache` parameter that saves RDD's in memory between query calls.
- This works well for benchmarking single queries on the same data, but requires substantial engineering effort to make production-ready.

System Architecture

Simple caching policy.

- **Workaround Solution:** Provide a `cache` parameter that saves RDD's in memory between query calls.
- This works well for benchmarking single queries on the same data, but requires substantial engineering effort to make production-ready.
  - Two queries could be submitted on the same date range that request overlapping, but not identical, column subsets. How should these data sets with partially overlapping values be cached?

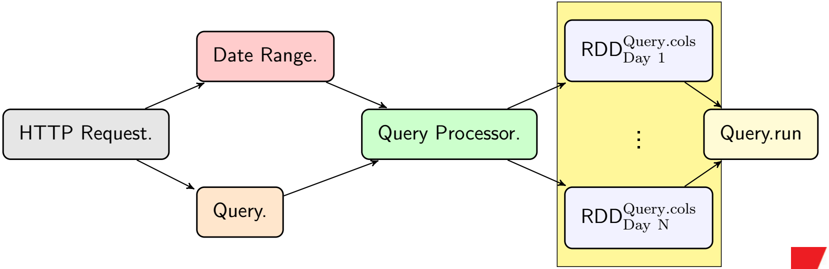
System Architecture

Simple caching policy.

- **Workaround Solution:** Provide a `cache` parameter that saves RDD's in memory between query calls.
- This works well for benchmarking single queries on the same data, but requires substantial engineering effort to make production-ready.
  - Two queries could be submitted on the same date range that request overlapping, but not identical, column subsets. How should these data sets with partially overlapping values be cached?
  - What if one of the queries is called substantially more times than the others? How can the caching policy recommend the data required by these queries be cached?

System Architecture

Simple caching policy.



System Architecture

Queries.

Shorthand	Name
Q0	Pageviews
Q1	Revenue
Q2	RevenueFromTopReferringDomains
Q3	RevenueFromTopReferringDomainsFirstVisitGoogle
Q4	TopPages
Q5	TopPagesByBrowser
Q6	TopPagesByPreviousTopPages
Q7	TopReferringDomains

System Architecture

Queries.

Shorthand	Name
Q0	Pageviews
Q1	Revenue
Q2	RevenueFromTopReferringDomains
Q3	RevenueFromTopReferringDomainsFirstVisitGoogle
Q4	TopPages
Q5	TopPagesByBrowser
Q6	TopPagesByPreviousTopPages
Q7	TopReferringDomains

	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
post_pagename	×				×	×	×	
user_agent						×		
visit_referrer			×	×				
post_visid_high			×	×			×	×
post_visid_low			×	×			×	×
visit_num			×	×			×	×
visit_referrer								×
hit_time_gmt							×	
post_purchaseid		×	×	×				
post_product_list		×	×	×				
first_hit_referrer				×				

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
    val queryResult = allData.collect{
        case (root) if !root.post_pagename.isEmpty => (root.post_pagename, 1)
      }
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
    val queryResult = allData.collect{
        case (root) if !root.post_pagename.isEmpty => (root.post_pagename, 1)
      }
      .reduceByKey(_+_,numPartitions)
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
    val queryResult = allData.collect{
        case (root) if !root.post_pagename.isEmpty => (root.post_pagename, 1)
      }
      .reduceByKey(_+_,numPartitions)
      .top(10) {
        Ordering.by((entry: ((String, Int))) => entry._2)
      }.toSeq
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
    val queryResult = allData.collect{
        case (root) if !root.post_pagename.isEmpty => (root.post_pagename, 1)
      }
      .reduceByKey(_+_,numPartitions)
      .top(10) {
        Ordering.by((entry: ((String, Int))) => entry._2)
      }.toSeq
    if (c.profile)
      "[" + queryResult.map("\"" + _.toString + "\"").mkString(", ") + "]"
```

System Architecture

Queries.

```
case class QueryConf(
  sc: SparkContext, data: Array[RDD[SiteCatalyst]], profile: Boolean,
  daysInRange: Seq[String], dailyRows: Seq[Long], targetPartitionSize: Long
)

object TopPages extends Query {
  def colsNeeded = Seq("post_pagename")
  def run(c: QueryConf) = {
    val allData = c.sc.union(c.data); val numAllRows = c.dailyRows.reduce(_+_)
    val numPartitions = (numAllRows/c.targetPartitionSize).toInt
    val queryResult = allData.collect{
        case (root) if !root.post_pagename.isEmpty => (root.post_pagename, 1)
      }
      .reduceByKey(_+_,numPartitions)
      .top(10) {
        Ordering.by((entry: ((String, Int))) => entry._2)
      }.toSeq
    if (c.profile)
      "[" + queryResult.map("\"" + _.toString + "\"").mkString(", ") + "]"
    else {
      html.TopPages("TopPages", c.daysInRange, queryResult).toString
    }
  }
}
```

System Architecture

Queries.

Online demo is available at `http://adobe-research.github.io/spindle/`

System Architecture

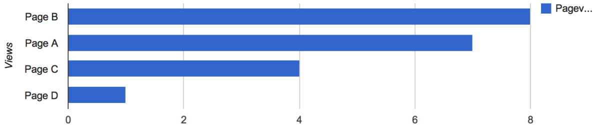
Queries.

Online demo is available at `http://adobe-research.github.io/spindle/`

TopPagesByBrowser

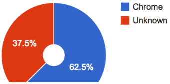
Run again and profile.

2014-08-14 - 2014-08-16



Page B

Total Pageviews: 8



Chrome	5
Unknown	3

System Architecture

Ad hoc queries.

- Spark SQL is an alpha feature in Spark and allows relational queries in SQL to be executed with Spark.

System Architecture

Ad hoc queries.

- Spark SQL is an alpha feature in Spark and allows relational queries in SQL to be executed with Spark.
- SiteCatalyst Parquet on HDFS can be loaded directly into the Spark SQL data types and queried with SQL strings.

System Architecture

Ad hoc queries.

Ad Hoc Queries.

2014-08-14 - 2014-08-16 **Number of Output Results** 3

The command line interface below demonstrates Spark SQL on the analytics data set. View the possible columns with the `columns` command and run a sql query with the `sql` command. The data for the dates specified is loaded into tables `data_2014_08_14` through `data_2014_08_16`, and all the data is unioned into a table named `all_data`. For example, `sql select count(*) from all_data` will output the total number of events through the specified date range.

```
Press <tab> to see a list of available commands.
> sql select count(*) from all_data
[20]
> sql select post_pagename, hit_time_gmt from data_2014_08_16 order by hit_time_gmt
[Page D,1408187379]
[Page A,1408187380]
[Page B,1408187380]
>
```

Empirical Results

Experimental Data

- Query date range is Jan 1, 2014 to Jan 7, 2014, inclusively.

Empirical Results

Experimental Data

- Query date range is Jan 1, 2014 to Jan 7, 2014, inclusively.
- Each Spark worker has 24 cores and 20GB of memory.

Empirical Results

Experimental Data

- Query date range is Jan 1, 2014 to Jan 7, 2014, inclusively.
- Each Spark worker has 24 cores and 20GB of memory.
- The entire Parquet databases on HDFS for each day totals 13.1GB.

Empirical Results

Experimental Data

- Query date range is Jan 1, 2014 to Jan 7, 2014, inclusively.
- Each Spark worker has 24 cores and 20GB of memory.
- The entire Parquet databases on HDFS for each day totals 13.1GB.
- Benchmarking is done with a Python script, and plots are produced with pyplot.

Empirical Results

Ideal number of partitions.

- Using 6 workers, vary the target partition sizes for each query that uses partitioning.

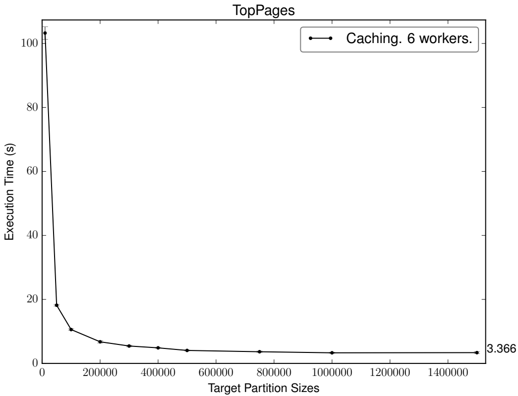
Empirical Results

Ideal number of partitions.

- Using 6 workers, vary the target partition sizes for each query that uses partitioning.
- Don't cache data and sample each target partition size 4 times.

Empirical Results

Ideal number of partitions.



Empirical Results

Ideal number of partitions.

- **Conclusion:** Target 1.5M records in each partition for best performance on this data set.

Query	Best Execution Time	Execution Time at 1.5M Target Size
Q2	16853.50	16853.50
Q3	16934.25	16934.25
Q4	3241.25	3241.25
Q5	14862.00	14877.25
Q6	35554.50	37034.00
Q7	6597.25	6597.25

Empirical Results

Caching.

- How much better performance does caching give?

Empirical Results

Caching.

- How much better performance does caching give?
- Leverage all 6 workers and target 1.5M records in each partition.

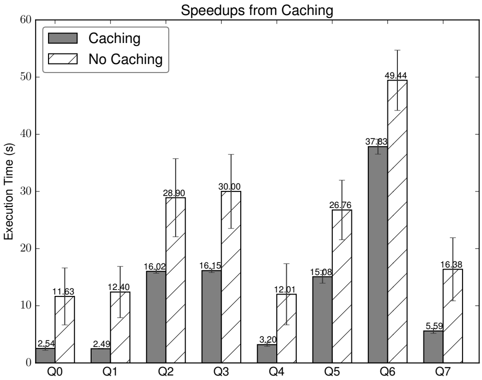
Empirical Results

Caching.

- How much better performance does caching give?
- Leverage all 6 workers and target 1.5M records in each partition.
- Sample each point 4 times.

Empirical Results

Caching.



Empirical Results

Benchmarking concurrent queries.

- Multiple users can utilize an analytics system like SiteCatalyst concurrently, and Spark can be used in multithreaded applications.

Empirical Results

Benchmarking concurrent queries.

- Multiple users can utilize an analytics system like SiteCatalyst concurrently, and Spark can be used in multithreaded applications.
- **Research Question:** How much will Spark's performance degrade if multiple users are utilizing it at the same time?

Empirical Results

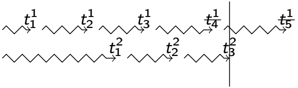
Benchmarking concurrent queries.

- Multiple users can utilize an analytics system like SiteCatalyst concurrently, and Spark can be used in multithreaded applications.
- **Research Question:** How much will Spark's performance degrade if multiple users are utilizing it at the same time?
- **Solution:** Assume users submit the same query at the same time and observe the average query response time.

Empirical Results

Benchmarking concurrent queries.

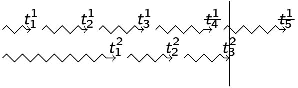
- A Python script using threading manages threads for each number of concurrent queries.



Empirical Results

Benchmarking concurrent queries.

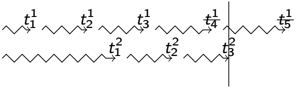
- A Python script using threading manages threads for each number of concurrent queries.
- Consider the example below for 2 threads and 3 repeated queries per thread. The first thread finishes before the second, and remains loaded until the second thread finishes. t_4^1 and t_5^1 are not used in the average.



Empirical Results

Benchmarking concurrent queries.

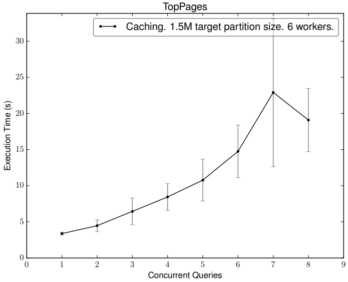
- A Python script using threading manages threads for each number of concurrent queries.
- Consider the example below for 2 threads and 3 repeated queries per thread. The first thread finishes before the second, and remains loaded until the second thread finishes. t_4^1 and t_5^1 are not used in the average.
- This experiment runs queries on the date range of Jan 1, 2014 to Jan 7, 2014 and each thread calls the query 4 times.



Empirical Results

Benchmarking concurrent queries.

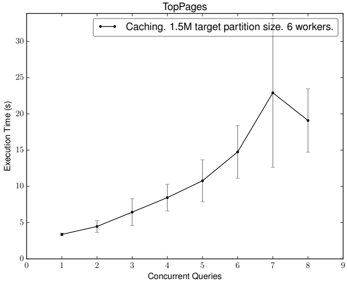
- Speedups for all queries are similar to the following.



Empirical Results

Benchmarking concurrent queries.

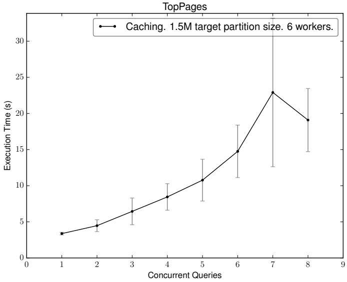
- Speedups for all queries are similar to the following.

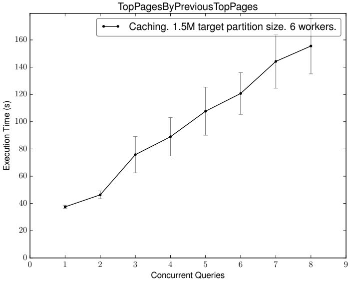


Empirical Results

Benchmarking concurrent queries.

- Speedups for all queries are similar to the following.





Conclusions

- Spark is a good candidate for real-time analytics processing.

Conclusions

- Spark is a good candidate for real-time analytics processing.
- Spindle is a research prototype and benchmarkable analytics engine.

Conclusions

- Spark is a good candidate for real-time analytics processing.
- Spindle is a research prototype and benchmarkable analytics engine.
- Spindle's future work is on caching and preprocessing data.

Questions?

Spindle Project	`http://github.com/adobe-research/spindle`
Brandon Amos	`http://github.com/bamos`
David Tompkins	`http://github.com/DavidTompkins`