Project Proposal

Hadoop and Hive as scalable alternatives to DBMS business intelligence solutions for Big Data

Marissa Hollingsworth

Department of Computer Science
College of Engineering
Boise State University

A term applied to data sets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time. -Wikipedia

A term applied to data sets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time. -Wikipedia

- *"More data usually beats better algorithms."*

  -Anand Rajaraman

A term applied to data sets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time. -Wikipedia

- *"More data usually beats better algorithms."*
  -Anand Rajaraman
- High-cost and challenges make it hard for smaller companies to take advantage of the business intelligence insights it can provide.

A term applied to data sets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time. -Wikipedia

- *"More data usually beats better algorithms."*
  -Anand Rajaraman
- High-cost and challenges make it hard for smaller companies to take advantage of the business intelligence insights it can provide.
- As data sets grow the cost of traditional database approach increases non-linearly.

Hadoop provides an open source solution for reliable and scalable distributed computing for Big Data.

Hadoop provides an open source solution for reliable and scalable distributed computing for Big Data.

- Runs on cheap commodity hardware vs expensive single machine

Hadoop provides an open source solution for reliable and scalable distributed computing for Big Data.

- Runs on cheap commodity hardware vs expensive single machine
- We will use the following Hadoop sub-projects for the proposed project:
  - *HDFS*: A distributed file system that provides high throughput access to application data.
  - *MapReduce*: A software framework for distributed processing of large data sets on compute clusters.
  - *Hive*: A data warehouse infrastructure with SQL ad-hoc querying.

An inherently parallel programming model and associated implementation for processing and generating large data sets.

An inherently parallel programming model and associated implementation for processing and generating large data sets.

- The user defines map and reduce functions:

An inherently parallel programming model and associated implementation for processing and generating large data sets.

- The user defines map and reduce functions:
  - `map`: processes raw input data to generate a set of key/value pairs.

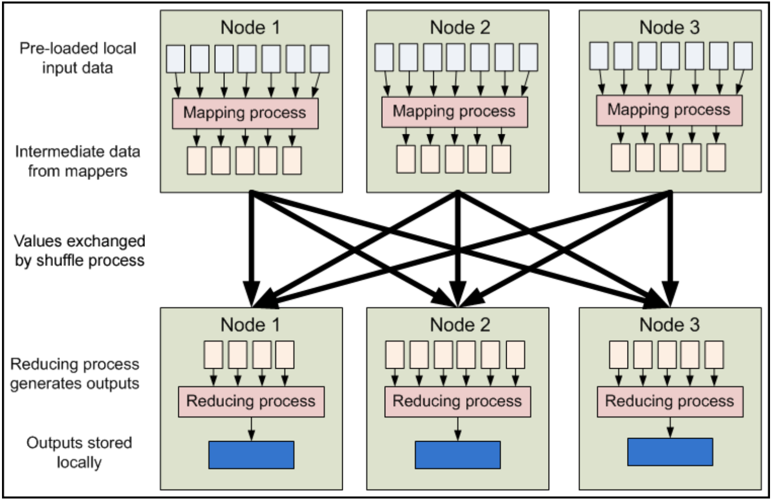
An inherently parallel programming model and associated implementation for processing and generating large data sets.

- The user defines map and reduce functions:
  - `map`: processes raw input data to generate a set of key/value pairs.
  - `reduce`: merges intermediate values associated with the same key to produce desired result.

An inherently parallel programming model and associated implementation for processing and generating large data sets.

- The user defines map and reduce functions:
  - `map`: processes raw input data to generate a set of key/value pairs.
  - `reduce`: merges intermediate values associated with the same key to produce desired result.
- Complex problems can use multiple map and reduce phases with dependencies between them.

hadoop MapReduce



A framework developed by Facebook for data warehousing on top of Hadoop.

A framework developed by Facebook for data warehousing on top of Hadoop.

- Analysts with strong SQL skill can easily run queries on huge volumes of data.

A framework developed by Facebook for data warehousing on top of Hadoop.

- Analysts with strong SQL skill can easily run queries on huge volumes of data.
- Query the data using a SQL-like language called HiveQL.

A framework developed by Facebook for data warehousing on top of Hadoop.

- Analysts with strong SQL skill can easily run queries on huge volumes of data.
- Query the data using a SQL-like language called HiveQL.
- Allows custom mappers and reducers when it is inconvenient or inefficient to express logic in HiveQL.

A Big Data Problem: Payment Analysis

Predict customer payment behavior based on payment histories.

A Big Data Problem: Payment Analysis

Predict customer payment behavior based on payment histories.

- Requires a batch process to analyze structured and unstructured datasets.

A Big Data Problem: Payment Analysis

Predict customer payment behavior based on payment histories.

- Requires a batch process to analyze structured and unstructured datasets.
- Rapid expansion of data size as client base expands.

A Big Data Problem: Payment Analysis

Predict customer payment behavior based on payment histories.

- Requires a batch process to analyze structured and unstructured datasets.
- Rapid expansion of data size as client base expands.
- Solution should be low-cost, easily manageable, and scalable.

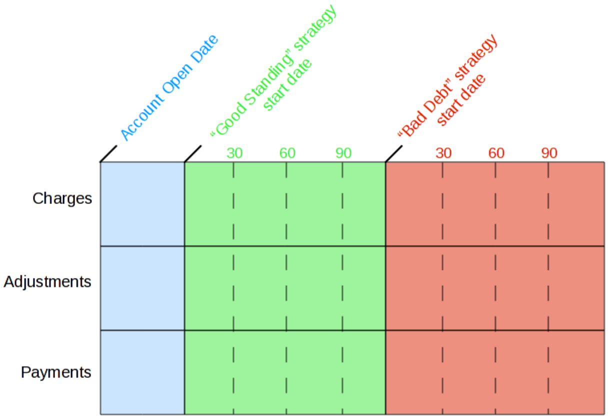
A Big Data Problem: Payment Analysis

Predict customer payment behavior based on payment histories.

- Requires a batch process to analyze structured and unstructured datasets.
- Rapid expansion of data size as client base expands.
- Solution should be low-cost, easily manageable, and scalable.

	Traditional RDBMS	*MapReduce*
Data size	GB-TBs	TBs-PBs
Access	Interactive and batch	Batch
Updates	Read/write many times	Write once, read many times
Structure	Static Schema	Dynamic Schema
Integrity	High	Low
Scaling	Nonlinear	Linear

A Big Data Problem: Payment Analysis



Customer

CustomerNumber	FirstName	LastName	Ssn	ZipCode3

StrategyHistory

AccountNumber	StratName	StratStartDate

Account

AccountNumber	OpenDate	Customer Number

Transaction

AccountNumber	TranDate	TranAmount	TranType

Figure: Aggregate charges, adjustments, and payments for each account.

Methods: Requirements Phase

The project requirements will be established as follows:

1. Meet with consultant.

Methods: Requirements Phase

The project requirements will be established as follows:

1. Meet with consultant.
2. Clearly outline the details and constraints.

Methods: Requirements Phase

The project requirements will be established as follows:

1. Meet with consultant.
2. Clearly outline the details and constraints.
3. Work with consultant to verify details and constraints.

Methods: Requirements Phase

The project requirements will be established as follows:

1. Meet with consultant.
2. Clearly outline the details and constraints.
3. Work with consultant to verify details and constraints.
4. Consistently verify that the requirements are being met and remain applicable during all phases of the project.

Methods: Design Phase

- **Custom writable classes**.
  1. `Customer` - object representing customer tuple.
  2. `Account` - object representing an account tuple.
  3. `Transaction` - object representing a transaction tuple.
  4. `StrategyHistory` - object representing a strategy history tuple.

Methods: Design Phase

- **Custom writable classes**.
    1. `Customer` - object representing customer tuple.
    2. `Account` - object representing an account tuple.
    3. `Transaction` - object representing a transaction tuple.
    4. `StrategyHistory` - object representing a strategy history tuple.
- **MapReduce job flow**. The complexity of the payment analysis problem requires several stages of data aggregation to achieve the final results. We will need to design the details and algorithms to achieve the output for the following jobs.

Methods: Design Phase

- **Custom writable classes**.
  1. `Customer` - object representing customer tuple.
  2. `Account` - object representing an account tuple.
  3. `Transaction` - object representing a transaction tuple.
  4. `StrategyHistory` - object representing a strategy history tuple.
- **MapReduce job flow**. The complexity of the payment analysis problem requires several stages of data aggregation to achieve the final results. We will need to design the details and algorithms to achieve the output for the following jobs.
- **Hive dataset structure**. The schema used to store the data in HDFS can have profound effects on the efficiency of Hive queries.

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:
    1. MapReduce solution.

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:
  1. MapReduce solution.
  2. Sample data generation.

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:
  1. MapReduce solution.
  2. Sample data generation.
  3. Hive solution.

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:
    1. MapReduce solution.
    2. Sample data generation.
    3. Hive solution.
    4. Benchmark test cases.

Methods: Implementation Phase

- Hadoop 0.18 series API and Java 1.6 API
- Eclipse, MapReduce plug-in, MySQL Workbench
- Implementation phases:
  1. MapReduce solution.
  2. Sample data generation.
  3. Hive solution.
  4. Benchmark test cases.

Methods: Testing Phase

- Test phase requires the following test cases:
  - Read and write accuracy of each `WritableComparable` type.
  - Output of each MapReduce job and final MapReduce result.
  - Accuracy of each HiveQL query and final Hive result.

Methods: Testing Phase

- Test phase requires the following test cases:
  - Read and write accuracy of each `WritableComparable` type.
  - Output of each MapReduce job and final MapReduce result.
  - Accuracy of each HiveQL query and final Hive result.
- Most testing during development will occur on a small data set on a single-node in *pseudo-distributed* mode.

Methods: Benchmarking Phase

- Benchmarks include:

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance
- Considerations:

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance
- Considerations:
  - Size of datasets: from 10GB to terabytes.

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance
- Considerations:
  - Size of datasets: from 10GB to terabytes.
  - HDFS cluster design and DBMS system specs:

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance
- Considerations:
  - Size of datasets: from 10GB to terabytes.
  - HDFS cluster design and DBMS system specs:
    - A *fully-distributed* HDFS cluster running on varied number of data nodes (3 to 16).

Methods: Benchmarking Phase

- Benchmarks include:
  - MapReduce versus DBMS: scalability and performance
  - Hive versus DBMS: scalability and performance
  - MapReduce versus Hive: performance
- Considerations:
  - Size of datasets: from 10GB to terabytes.
  - HDFS cluster design and DBMS system specs:
    - A *fully-distributed* HDFS cluster running on varied number of data nodes (3 to 16).
    - The estimated cost of the DBMS system and HDFS cluster will be comparable for comparison tests.

Project Schedule

December 2010	- Meet with consultant to define problem.
January 2011	- Obtain specification documents and start application design phase.
February 2011	- Solidify application requirements and design. - Begin implementation and test phases of MapReduce solution.
March 2011	- Finalize MapReduce solution. - Begin implementation phase of sample data generation.
April 2011	- Use sample data to compare MapReduce implementation to MySQL solution. - Begin implementation and test phases of Hive solution. - Write report sections for MapReduce solution.
May 2011	- Finalize Hive solution. - Use sample data to compare Hive implementation to MapReduce and MySQL implementations. - Write report sections for Hive solution. - Finalize report.