Hadoop and No SQL

Nagaraj Kulkarni

Nagaraj.Kulkarni@compegence.com
Process, Data and Domain Integrated Approach

Decision Excellence
Competitive Advantage lies in the exploitation of:

- More detailed and specific information
- More comprehensive external data & dependencies
- Fuller integration
- More in depth analysis
- More insightful plans and strategies
- More rapid response to business events
- More precise and apt response to customer events

Ram Charan’s Book: What The CEO Wants You To Know: How Your Company Really Works
Touching Upon

Context For Big Data Challenges

Data Base Systems – Pre Hadoop Strengths and Limitations

What is Scale, Why No SQL

Think Hadoop, Hadoop Eco system

Think Map Reduce

Nail Down Map Reduce

Think GRID (Distributed Architecture)

Deployment Options

Map Reduce Not and Map Reduce Usages

Nail Down HDFS and GRID Architecture

Objective: Understand MapReduce Paradigm
Systemic Changes

- Connected Globe
- Customer Centric
- Agility and Response Time
- Boundary less ness
- Best Sourcing
- Interlinked Culture
- Demand Side Focus
- Bottom Up Innovation
- Empowered employees
- Leading Trends
- Responsiveness
- Speed, Agility, Flexibility
Landscape To Address

Data Explosion

Manageability
Scalability
Performance

Information Overload

Agility
Decision Making
Time to Action

Interlinked Processes & Systems

Boundaryless
Systemic Understanding
Collaborate and Synergize
Simplify and Scale
Information Overload

A wealth of information creates a poverty of attention.

Herbert Simon, Nobel Laureate Economist
How do we scale

**Traditional System - How they achieve Scalability**

- Multi Threading
- Multiple CPU – Parallel Processing
- Distributed Programming – SMP & MPP
- ETL Load Distribution – Assigning jobs to different nodes
- Improved Throughput
### Scale – What is it about?

<table>
<thead>
<tr>
<th>Company</th>
<th>Monthly Users</th>
<th>Monthly Views</th>
<th>Monthly Content</th>
<th>New Data/day</th>
<th>Cluster Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>500 Million</td>
<td>500 Billion</td>
<td>25 Billion</td>
<td>15 TB</td>
<td>1200 m/cs, 21 PB Cluster</td>
</tr>
<tr>
<td>eBay</td>
<td>90 Million</td>
<td>10 Billion</td>
<td>220 million</td>
<td>40 TB</td>
<td>80 + nodes</td>
</tr>
<tr>
<td>Twitter</td>
<td>1 TB</td>
<td>50 Million</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yahoo</td>
<td>82 PB</td>
<td>247 Billion</td>
<td>126 Million</td>
<td>40 PB</td>
<td></td>
</tr>
</tbody>
</table>

**Facebook**
- 500 Million Active Users per Month
- 500 Billion+ Page Views per month
- 25 Billion+ Content per month
- 15 TB New Data / day
- 1200 m/cs, 21 PB Cluster

**eBay**
- 90 Million Active Users
- 10 Billion Requests per day
- 220 million+ items on sale
- 40 TB + / day
- 40 PB of Data

**Twitter**
- 1 TB plus / day
- 80 + nodes

**Yahoo**
- 82 PB of Data
- 25000+ nodes

**Estimated Data**
- 1.73 Billion Internet Users
- 247 Billion emails per day
- 126 Million Blogs
- 5 Billion Facebook Content per week
- 50 Million Tweets per day
- 80% of this data is **unstructured**

**Facebook Content**
- Estimated 800 GB of data per user (million Petabyte!)
Thinking of Scale - Need for Grid

**Think Numbers**

1000 Nodes / DC
10 DC
1K byte webserver log record
1 second / row

In one day

\[ 1000 \times 10 \times 1K \times 60 \times 60 \times 24 = 864 \text{ GB} \]

Storage for a year

\[ 864 \text{ GB} \times 365 = 315 \text{ TB} \]

To store 1 PB – 40K * 1000 = **Millions $**

To process 1 TB = 1000 minutes ~ 17 hrs

**Think Agility and Flexibility**
Scale – What is it about?

What scaling means

Volume
Speed
Integration level
more…

Does it scale linearly with data size and analysis complexity

money
We would not have no issues…

If the following assumptions Hold Good:

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.
New Paradigm: Go Back to Basics

✓ Divide and Conquer (Divide and Delegate and Get Done)
✓ Move Work or Workers?
✓ Relax Constraints (Pre defined data models)
✓ Expect and Plan for Failures (avoid n address failures)
✓ Community backup
✓ Assembly Line Processing
  ✓ (Scale, Speed, Efficiency, Commodity Worker)
✓ The “For loop”
✓ Parallelization (trivially parallelizable)
✓ Infrastructure and Supervision (Grid Architecture)
✓ Manage Dependencies
✓ Ignore the Trivia (Trivia is relative!)

Charlie Munger’s Mental Models

Hadoop and No SQL
<table>
<thead>
<tr>
<th>Map Reduce Paradigm</th>
<th>Grid Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Divide and Conquer</td>
<td>✓ Split and Delegate</td>
</tr>
<tr>
<td>✓ The “for loop”</td>
<td>✓ Move Work or Workers</td>
</tr>
<tr>
<td>✓ Sort and Shuffle</td>
<td>✓ Expect and Plan for Failures</td>
</tr>
<tr>
<td>✓ Parallelization (trivially parallelizable)</td>
<td>✓ Assembly Line Processing (Scale, Speed, Efficiency, Commodity Worker)</td>
</tr>
<tr>
<td>✓ Relax Data Constraints</td>
<td>✓ Manage Dependencies and Failures</td>
</tr>
<tr>
<td>✓ Assembly Line Processing Scale, Speed, Efficiency, Commodity Worker)</td>
<td>✓ Ignore the Trivia (Trivia is relative!)</td>
</tr>
</tbody>
</table>

Map Reduce History
- Lisp
- Unix
- Google FS

Replication, Redundancy,
Heart Beat Check, Cluster rebalancing,
Fault Tolerance, Task Restart,
Chaining of jobs (Dependencies),
Graceful Restart,
Look Ahead or Speculative execution,
Hbase/Cassandra for huge data volumes- PBs.
• Hbase fits in well where Hadoop is already being used.
• Cassandra less cumbersome to install/manage

MongoDB/CouchDB
Document oriented databases for easy use and GB-TB volumes. Might be problematic at PB scales

Neo4j like graph databases
for managing relationship oriented applications- nodes and edges

Riak, redis, membase like Simple key-value databases
for huge distributed in-memory hash maps
Let us Think Hadoop

Hadoop Ecosystem Map

1. Unstructured Data
   - Flume
   - Scribe

2. Structured Data
   - hiho
   - Sqoop

3. File System
   - hadoop

4. Engine + Logic
   - Oozie
   - Cascading

5. Workflow

6. Workflow Support

7. Monitor/Manage Hadoop ecosystem
   - Hue
   - Ganglia
   - HBase

8. High Level Interfaces
   - JAQL

9. More High Level Interfaces
   - Mahout
   - Amazon Web Services

10. Unstructured Data

11. Structured Data

12. Support

13. Monitor/Manage Hadoop ecosystem

14. OLTP
## RDBMS and Hadoop

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data size</strong></td>
<td>Gigabytes</td>
<td>Petabytes</td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>Interactive and batch</td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Structure</strong></td>
<td>Fixed schema</td>
<td>Unstructured schema</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>SQL</td>
<td>Procedural (Java, C++, Ruby, etc)</td>
</tr>
<tr>
<td><strong>Integrity</strong></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Nonlinear</td>
<td>Linear</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>Read and write</td>
<td>Write once, read many times</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
Apache Hadoop Ecosystem

**Hadoop Common**: The common utilities that support the other Hadoop subprojects.

**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.

**Pig**: A high-level data-flow language and execution framework for parallel computation.

**HBase** / Flume / Scribe: A scalable, distributed database that supports structured data storage for large tables.

**Hive**: A data warehouse infrastructure that provides data summarization and ad hoc querying.

**ZooKeeper**: A high-performance coordination service for distributed applications.

**Flume**: Message Que Processing

**Mahout**: scalable Machine Learning algorithms using Hadoop

**Chukwa**: A data collection system for managing large distributed systems.
Apache Hadoop Ecosystem

ETL Tools
- Pig (Data Flow)
- HBase (Key-Value store)

BI Reporting
- Hive (SQL)

RDBMS
- Sqoop

MapReduce (Job Scheduling/Execution System)
- HDFS (Hadoop Distributed File System)

Zookeeper (Coordination)

Avro (Serialization)

GRID Architecture

Avro (Serialization)
HDFS – The BackBone

Hadoop Distributed File System

HDFS

- The Backbone

Hadoop

Large Data

NameNode

Data Nodes

HDFS Architecture

Metadata (Name, replicas, ...): /home/foo/data, 3, ...

Namenode

Datanodes

Replication

Block ops

Write

Read

Client

Rack 1

Rack 2
Map Reduce – The New Paradigm

Transforming Large Data

MapReduce Basics

- Functional Programming
- List Processing
- Mapping Lists

Mappers

Reducers
PIG – Help the Business User Query

Pig: Data-aggregation functions over semi-structured data (log files).

- Pig Latin Programs
- Query Parser
  - Semantic Checking
  - Logical Optimizer
- Logical to Physical Translator
- Physical To M/R Translator
- Map Reduce Launcher
- Logical Plan
- Optimized Logical Plan
- Physical Plan
- MapReduce Plan
Pig Latin Example

Pig Latin

A = LOAD 'myfile'
   AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE 
x, COUNT(B);
STORE D INTO 'output';

pig.jar:
- parses
- checks
- optimizes
- plans execution
- submits jar
to Hadoop
- monitors job progress

Execution Plan
Map:
  Filter
Reduce:
  Count

Do you like the Flexibility?
A high level interface on Hadoop for managing and querying structured data
- Interpreted as Map-Reduce jobs for execution
- Uses HDFS for storage
- Uses Metadata representation over hdfs files

Key Building Principles:
- Familiarity with SQL
- Performance with help of built-in optimizers
- Enable Extensibility – Types, Functions, Formats, Scripts
FLUME – Distributed Data Collection

- Distributed Data / Log Collection Service
- Scalable, Configurable, Extensible
- Centrally Manageable

- Agents fetch data from apps, Collectors save it
- Abstractions: Source -> Decorator(s) -> Sink
Oozie – Workflow Management

An Oozie Workflow
Understanding Map Reduce Paradigm

Logical Architecture

- Pre-loaded local input data
- Intermediate data from mappers
- Values exchanged by shuffle process
- Reducing process generates outputs
- Outputs stored locally
Understanding Map Reduce Paradigm
Map Reduce Paradigm

Job
Configure the Hadoop Job to run.

Mapper
map(LongWritable key, Text value, Context context)

Reducer
reduce(Text key, Iterable<IntWritable> values, Context context)
MapReduce is a functional programming model and an associated implementation model for processing and generating large data sets.

Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs,

and

a reduce function that merges all intermediate values associated with the same intermediate key.

Many real world tasks are expressible in this model.
Input & Output: each a set of key/value pairs

Programmer specifies two functions:

map (in_key, in_value) -> list(out_key, intermediate_value)
  • Processes input key/value pair
  • Produces set of intermediate pairs

reduce (out_key, list(intermediate_value)) -> list(out_value) Combines all intermediate values for a particular key
  • Produces a set of merged output values (usually just one)
  • Inspired by similar primitives in LISP and other languages
Word Count Example

A simple MapReduce program can be written to determine how many times different words appear in a set of files.

What does Mapper and Reducer do?

Pseudo Code:

**mapper** (filename, file-contents):
for each word in file-contents:
emt (word, 1)

**reducer** (word, values):
sum = 0
for each value in values:
    sum = sum + value
emt (word, sum)
Example: Count word occurrences

map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_value:
       EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
   // output_key: a word
   // output_values: a list of counts
   int result = 0;
   for each v in intermediate_values:
       result += ParseInt(v);
   Emit(AsString(result));

Pseudocode: See appendix in paper for real code
Map – Reduce Execution Recap

- Master-Slave architecture

- Master: JobTracker
  - Accepts MR jobs submitted by users
  - Assigns Map and Reduce tasks to TaskTrackers (slaves)
  - Monitors task and TaskTracker status, re-executes tasks upon failure

- Worker: TaskTrackers
  - Run Map and Reduce tasks upon instruction from the Jobtracker
  - Manage storage and transmission of intermediate output
Understanding Map Reduce Paradigm

Map – Reduce Paradigm Recap

Example of map functions –
  Individual Count, Filter, Transformation, Sort, Pig load

Example of reduce functions –
  Group Count, Sum, Aggregator

A job can have many map and reducers functions.
How are we doing on the Objective

Objective:

Understand

MapReduce

Paradigm