Big Data Integration and Analysis

Requirements Specification

Prepared By:

Gaurav Kumar

Héctor Ugarte

Miguel Mármol

Tina Boroukhian

University of Bonn

5 May 2015

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This document presents Requirements Specification for Big Data Integration and Analysis using Apache Hadoop and Apache Spark. It describes the scope of the system, an overall description and both functional and non-functional requirements for the software.

1.2 Project Scope

These days data streams from each and every activity of daily life: from phones and credit cards and televisions and computers; from sensor-equipped buildings,GPS, trains, buses, planes, bridges, and factories. The data flows so fast that the total accumulation of the past two years—a zettabyte—dwarfs the prior record of human civilization. This huge amount of data is very important as it contains a lot of useful information and considering the volume, velocity and variety of data, cleaning and analysing big data is a big challenge.

A real example of such challenge can be seen in E-commerce company such as Amazon where they have huge amount of customer related data. This data is very important for company so that they can analyse data for future predictive modeling about most sought product by customers.

In our project, we are aiming for Big Data Integration and Analysis.

1.3 Roles definition

- **Client:** is the person or group of people that want to integrate and analyze certain data set.

2. Overall description

2.1 Product perspective

The Big Data Integration and Analysis project intend to use Apache Hadoop 2.6, The process is explained as below:

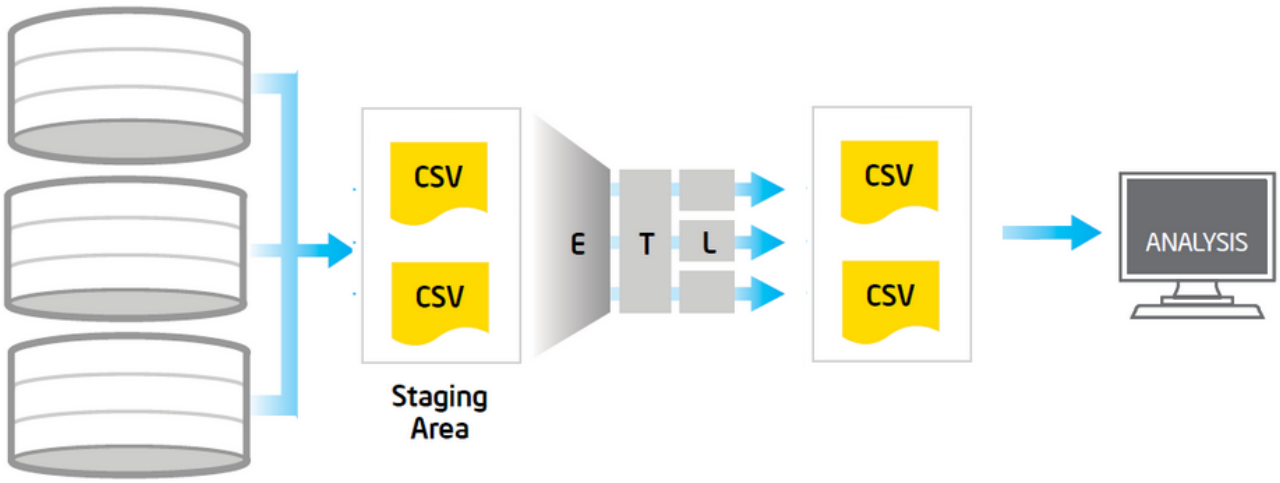

Fig 1. Overall process to be done.

- Extract data from sources. During the extract step, data is collected from different source systems and in CSV file format (flat files with delimiters) which is the default file format of Apache Hadoop.
- Transform that data into a standard format. The transform step may include multiple data manipulations, such as moving, splitting, translating, merging, sorting, pivoting, and more. For example, a customer name might be split into first and last name, or dates might be changed to the standard ISO format (e.g., from 07-24-13 to 2013-07-24). Often this step also involves validating the data against data quality rules.
- Analyse the data using analytic queries on above formatted data. Analytic queries contain aggregation functions such as finding max, min, average or count.

2.2. Apache Hadoop

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.

The core of Apache Hadoop consists of:

- Storage part: Hadoop Distributed File System (HDFS).
- Processing part: Hadoop MapReduce.

Hadoop splits files into large blocks and distributes them amongst the nodes in the cluster. To process the data, Hadoop MapReduce transfers packaged code for nodes to process in parallel, based on the data each node needs to process.

The base Apache Hadoop framework is composed of the following modules:

- Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
- Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the

cluster;

- Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and
- Hadoop MapReduce – a programming model for large scale data processing.

A MapReduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation. It follows these steps:

- "Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.
- "Shuffle" step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- "Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

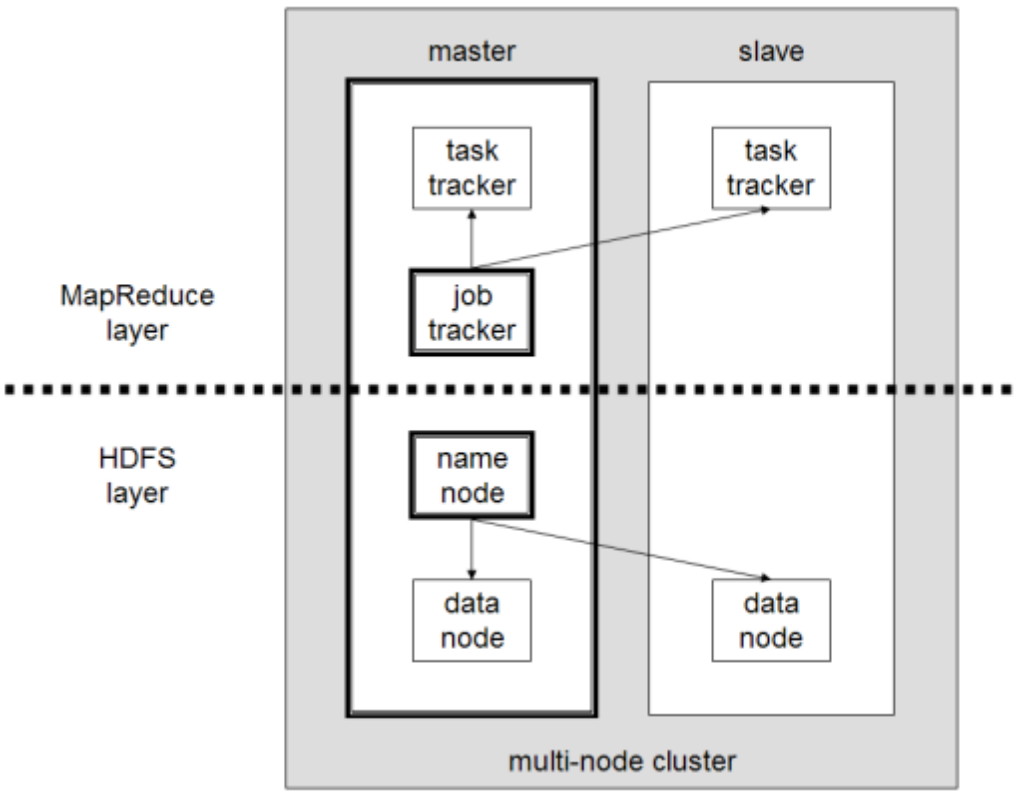

Fig 2. A multi-node Hadoop cluster architecture.

2.3. Apache Spark

Apache Spark is an open-source cluster computing framework. It provides high-level APIs in Java, Scala and Python. In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's in-memory primitives provide performance up to 100 times

faster.

Spark runs locally on each node and executes in memory when possible. Based on Spark's Resilient Distributed Datasets (RDD), Spark can employ RAM for dataset persistence. Spark stores files for chained iteration in memory as opposed to using temporary storage in HDFS, as Hadoop does. Contrary to Hadoop, Spark utilizes multiple threads instead of multiple processes to achieve parallelism on a single node, avoiding the memory overhead of several JVMs.

The fundamental programming abstraction is called Resilient Distributed Datasets(RDD), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory.

It has two components:

- Driver
- Workers

Driver

- ➢ defines and invokes actions on RDDs
- ➢ tracks the RDDs' lineage

Workers

- ➢ store RDD partitions
- ➢ perform RDD transformations

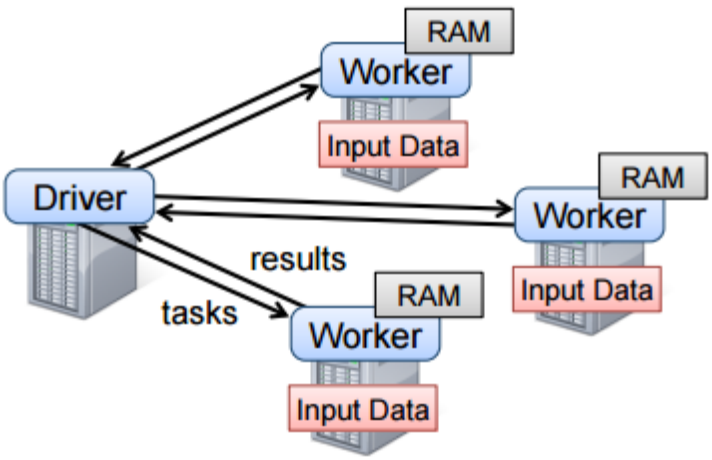

Fig 3. Spark runtime.

Spark Runtime

The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

3. Specific Requirements

The following section illustrates the requirements for the project. Those requirements are divided between functional and non-functional.

3.1. Functional Requirements

The functional requirements are grouped according to use case model.

A requirement has the following properties:

- Requirement ID uniquely identifies requirement.
- Title defines the functional group the requirement belongs to. It gives the requirement a symbolic name.
- Description the definition of the requirement.
- Priority defines the order in which requirements should be implemented. Priorities are designated (highest to lowest) "1", "2", and "3". Requirements of priority 1 must be implemented in the first productive system release. The requirements of priority 2 and lower are subject of special release agreement, which is out of scope of this document.
- Risk specifies risk of not implementing the requirement. It shows how the particular requirement is critical to the system. Next are the risk level and the associated impact to the system if the requirement is not implemented or implemented incorrectly:

  Critical (C) – will break the main functionality of the system. The system can not be used if this requirement is not implemented.

  High (H) – will impact the main functionality of the system. Some functions of the system could be inaccessible, but the system can be generally used.

  Medium (M) – will impact some system's features, but not the main functionality. System can be used with some limitation.

  Low (L) – the system can be used without limitation, but with some workarounds.

Req ID. Title Description Priority Risk References	R01. **Get raw sample data in CSV files.** Get or collect data from our sources. Input data must be in CSV file format. 1 C -

Req ID.	R02.
Title	**Move data to HDFS.**
Description	Move data to HDFS (hadoop Distributed file system) so that data can be processed with MapReduce programming model.
Priority	1
Risk	C
References	R01.

Req ID.	R03.
Title	**Clean data.**
Description	Clean and format data in a proper manner such as standard date and correct number formatting.
Priority	1
Risk	C
References	R01, R02.

Req ID.	R04.
Title	**Reduce columns.**
Description	Reduce the columns so that we keep only needed columns for desired results.
Priority	1
Risk	M
References	R02.

Req ID.	R05.
Title	**Analyze data.**
Description	Use analytic queries on above formatted data. Analytic queries contain aggregation functions such as finding max, min, average or count.
Priority	1
Risk	M
References	R04.

Req ID.	R06.
Title	**Get results.**
Description	get desired result and move output from HDFS to local machine.
Priority	1
Risk	M
References	R05.

4. Non-functional Requirements

4.1 Scalability

The system should be able to scale to many nodes capable of processing increasing sizes of data within a reasonable span of time.

4.2 Performance

System should be working in high performance manner so that It can process huge amounts of data within a reasonable span of time.

4.3 Fault Tolerant

System should be able to behave as fault tolerant, i.e. in case of a failure of one or more machines, it will continue working.

4.4 Distributed Environment

System should be working in a distributed computing environment, where processes can run in parallel across multiple machines.