WILFRID LAURIER UNIVERSITY
LAURIER
Inspiring Lives.

Computer Algebra Research Group
of
Wilfrid Laurier University

Integer Matching Using Cluster Computing Frameworks

By: Mohamed Mohamedtaki, Directed Research Student

Supervised by: Dr. Ilias Kotsireas, Professor, Wilfrid Laurier University

March 23, 2016

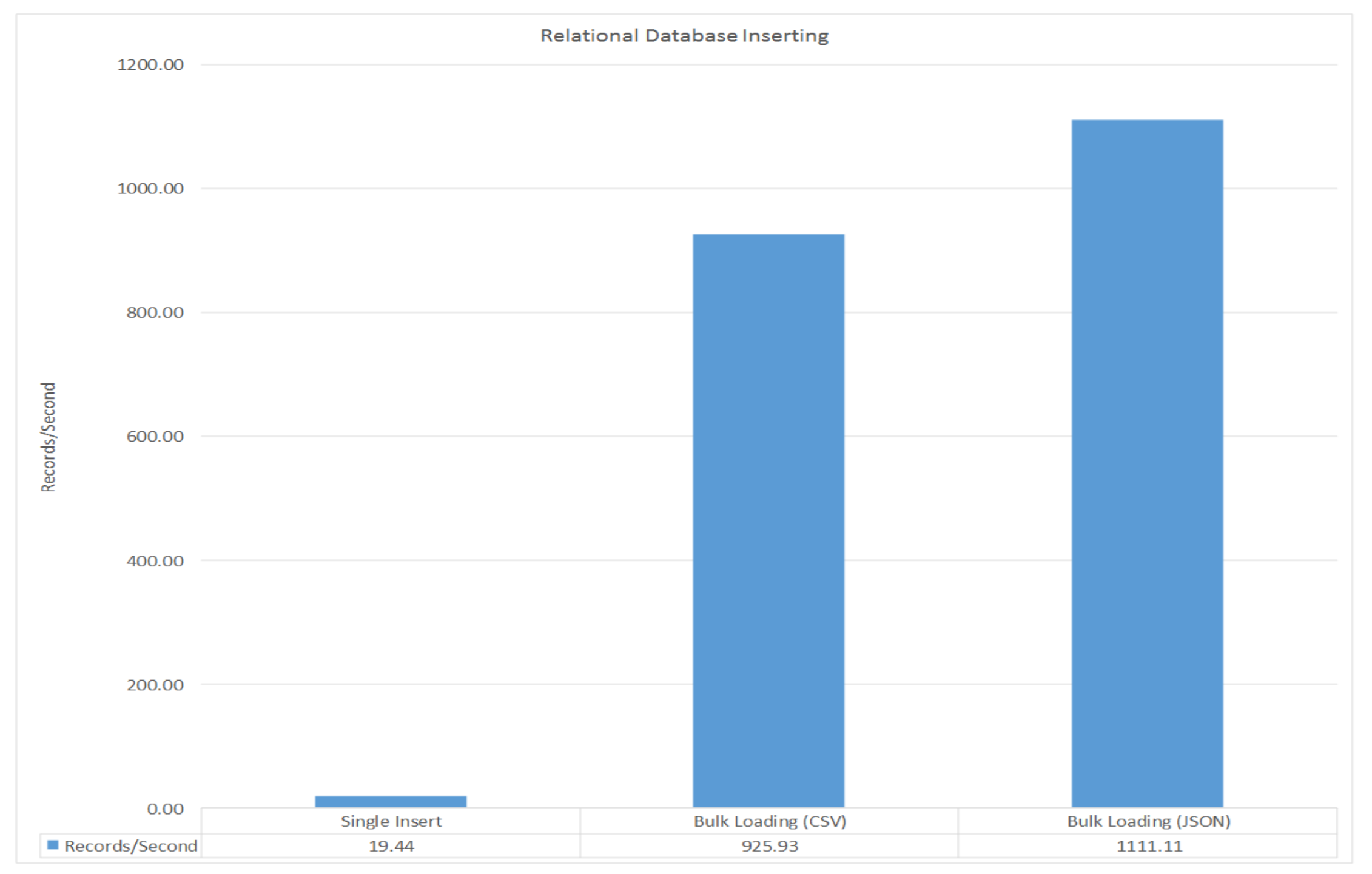
NSERC
CRSNG

SHARCNET™

Abstract

Relational databases rely on schemas, allowing data to be separated into entities. Data is broken down into a structured form, and loaded into databases where it can be analyzed and stored [4]. But the very nature of data is not structured, and relational databases, can manage unstructured data in small sets before having the need to scale up, requiring a costly investment in more powerful servers and licenses [1].

A large matrix was computed consisting of 62 columns and more than 10 million rows of non-negative integer data. Its output was separated into separate files for portability, having a total file size of more than 1.5 terabytes.



	Single Insert	Bulk Loading (CSV)	Bulk Loading (JSON)
■ Records/Second	19.44	925.93	1111.11

Figure 1: The inserting hurdle for large datasets

Loading data into a relational database management system is tedious, and a time consuming. The project's aim is to use **open cluster frameworks** in order to eliminate the time consuming inserting hurdle, and efficiently find a matching between various files given a total for the match.

1) Selecting a Framework

Network clusters can be used for processing various jobs in parallel. Servers in the cluster work in unison, allowing for data redundancy and redirection of work; in the event that a server is taken offline [2].

1.1) Hadoop

Apache Hadoop consists of 3 main functions, **Hadoop core**, the main engine running Hadoop, a storage system known as **Hadoop Distributed File System** (HDFS), and a processing framework, **MapReduce** [1].

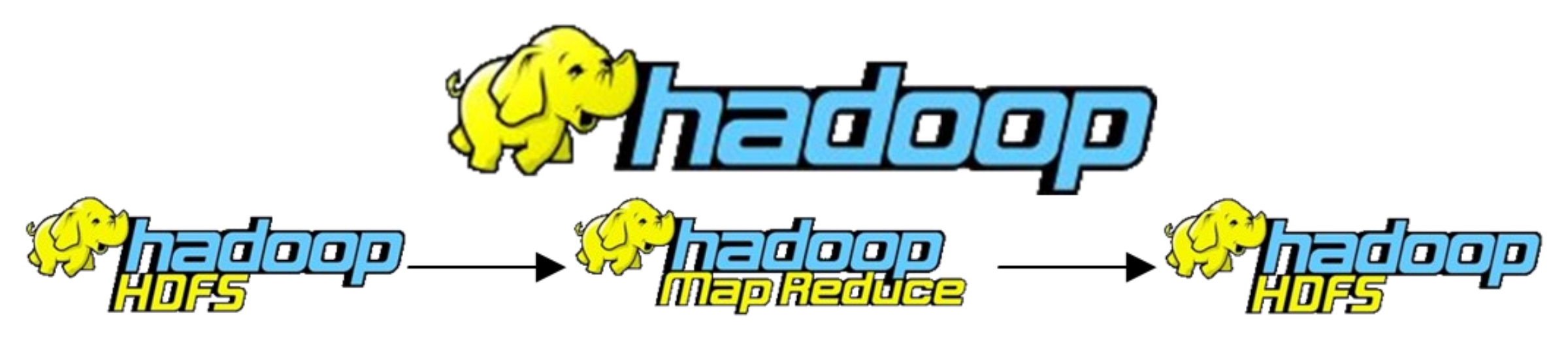


Figure 2: Hadoop processing cycle [5]

- **HDFS** works by splitting files into large blocks, distributing them across nodes in a network cluster and spreading them across a cluster to be used through out [1].
- **MapReduce** is a model for processing data, it first Maps the data by sorting, and filtering, then Reducing it into a summary. The implementation is separated into 2 tasks, and computed in parallel. Hadoop heavily uses the hard disk to perform it's computations [1].

1.2) Spark

Spark was developed to provide a more versatile engine than Hadoop, with a focus on computing performance, development speed, and diverse support for various programming languages. Many data sources can be read by Spark, and separated into Resilient Distributed Datasets (RDDs) [5].

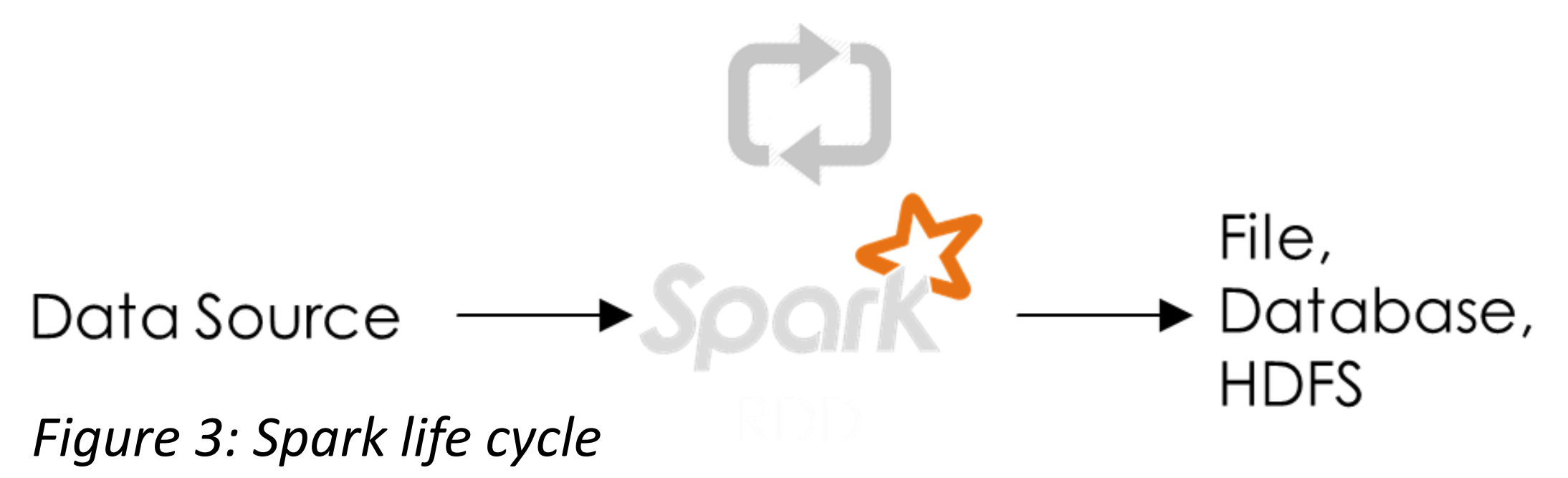


Figure 3: Spark life cycle

Spark processes RDDs through set operations, applying a series of **transformations,** followed by **actions.** Transformations are used to manipulate, and reduce a data set, whereas actions are used for analyzing the reduced set. Once an action has been performed, data can be further transformed, and other actions can be applied. Spark core is capable of efficiently managing memory, and performs faster than Hadoop core [5].

2) Matching

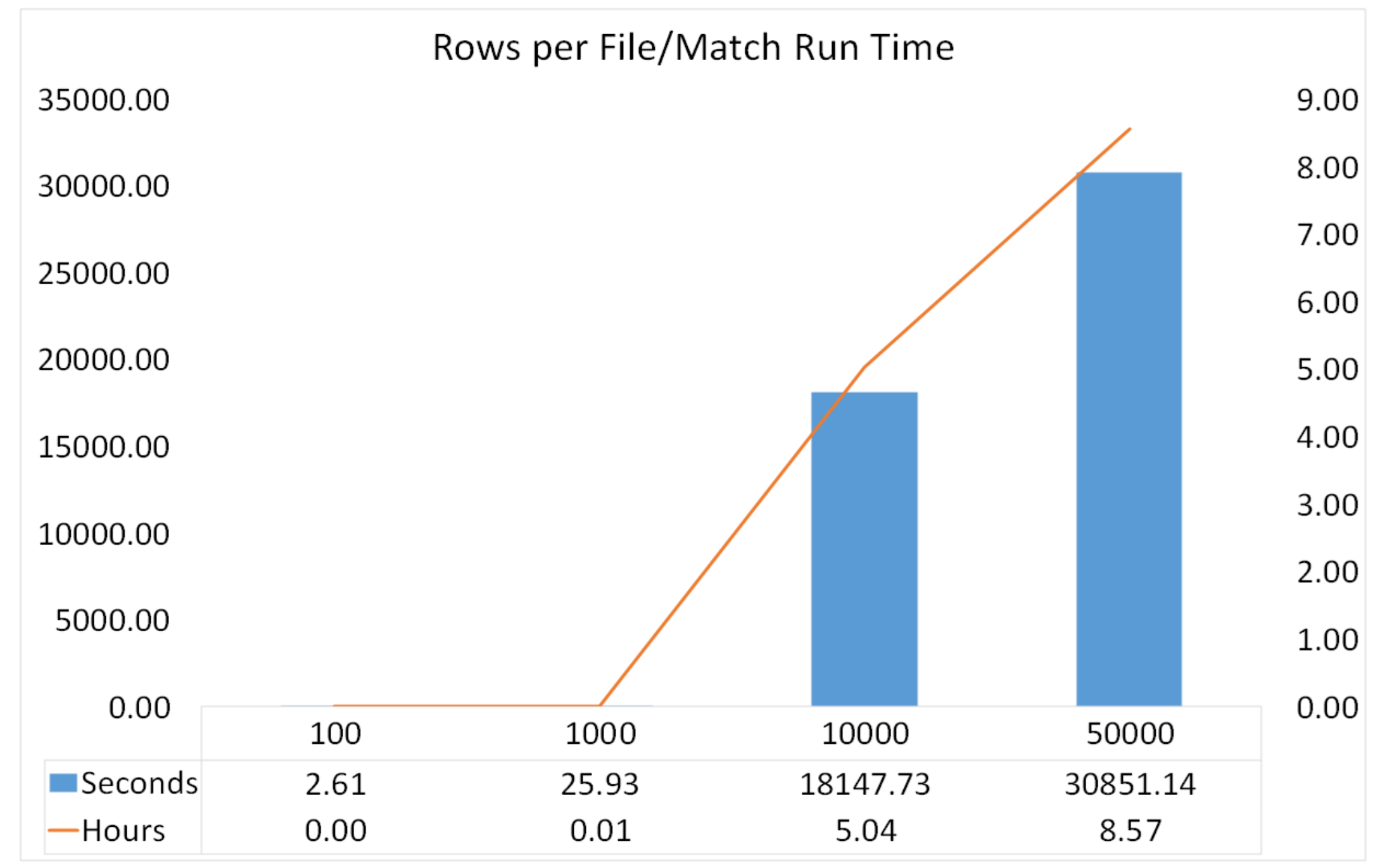
There exists two rows in the separate files where the sum of each overlapping columns is equal to some given λ.

$$(a_1 + b_1) = \lambda \ldots \ (a_n + b_n) = \lambda, \text{ a and b are files}$$

The following procedure was implemented using Spark, and run on various sizes of data [2]:

- Create **mappings** to use for each row in the **RDD**
- Spark core **infers** the schema, and maps it to its respective file
- The data is stored as a temporary "table"
- Using **Spark SQL** a matching is performed to find where the result exists within the two files

Efficiency for a 2-way matching using Apache Spark using 1 master node and only 3 worker nodes is summarized below:



	100	1000	10000	50000
■ Seconds	2.61	25.93	18147.73	30851.14
— Hours	0.00	0.01	5.04	8.57

Figure 4: Spark 2-way matching efficiency

As the volume of data increases, the job will require scaling, in order to effectively determine a 2-way match with a very large data set.

Another challenge appears as the number of files increases. Finding a match between 3 or more files requires a different algorithm. The proposed solution relies on λ.

If a match exists between n files with k columns, then there is some row in each of the n files where a sum exists that is equal to λ.

For example, for a 3-way matching, there are 3 files with k columns.

$$f_1, f_2, f_3$$

If all the values of the rows in f_3 are greater than zero, then the sum of the columns in the row in f_1 and f_2, which sum with a row in f_3 to yield λ, will be less than λ.

$$(f_{1,1} + f_{2,1}) < \lambda \ldots (f_{1,i} + f_{2,i}) < \lambda \ldots (f_{1,k} + f_{2,k}) < \lambda$$

Then there must be a range of integers less than λ, in which the immediate sum of $f_{1,i}$ and $f_{2,i}$ falls within, which can be used to further reduce the RDD to a smaller set.

$$\alpha < \beta < \lambda$$

and

$$(f_{1,1} + f_{2,1}) \in [\alpha,\beta] \ldots (f_{1,i} + f_{2,i}) \in [\alpha,\beta] \ldots (f_{1,k} + f_{2,k}) \in [\alpha,\beta]$$

Once reduced, this new RDD can be matched against f_3 in order to find the rows from f_1, f_2, and f_3 who's individual column sum is λ.

A similar approach can be taken with a 4-way matching algorithm, where two sets of files are reduced by range, and their resulting RDDs are summed to equal λ, and their rows are returned.

Conclusion

- Relational databases require too much structure, and cannot be efficiently used for handling unstructured data.
- Hadoop, and Spark can both be used to analyze large unstructured data sets, harnessing the power of a cluster, Spark is (10x-100x)[3] faster than Hadoop
- 2 Way matching can be performed relatively quickly with Apache Spark on small sets of data
- Next steps include optimizing the matching algorithm, to work quickly on larger data sets, and implementing the 3+ matching

References

[1] Zikopoulos, Paul. Big Data beyond the Hype: A Guide to Conversations for Today's Data Center. N.p.: McGraw-Hill Education, 2015. Print.

[2] Karau, Holden, Andy Konwinski, Patrick Wendell, and Matei Zaharia.Learning Spark. Sebastopol, CA: O'Reilly Media, 2014. Print.

[3] "Apache Spark." Apache Spark. Apache Software Foundation, n.d. Web. <http://spark.apache.org>.

[4] Salehnia, Ali. "Comparisons of Relational Databases with Big Data: A Teaching Approach." South Dakota State University, 2015. Web. <https://www.asee.org/documents/zones/zone3/2015/Comparisons-of-Relational-Databases-with-Big-Data-a-Teaching-Approach.pdf>.

[5] Intro to Apache Spark Training - Part 1. Perf. Pacco. Youtube. Databricks, n.d. Web. <https://www.youtube.com/watch?v=VWeWViFCzzg>.