THE UNIVERSITY OF SYDNEY

A STUDY ON IMPLEMENTING ITERATIVE ALGORITHMS USING BIGDATA FRAMEWORKS

Jack Galilee
Dr. Ying Zhou
School of Information Technologies
FACULTY OF ENGINEERING & INFORMATION TECHNOLOGIES

BACKGROUND

Iterative Algorithms

- Common in the fields of machine learning, and data mining.
- Repeatedly read (*iterate*) over the input dataset, applying one or more functions to improve, or add to, the final solution.
- Terminate (*converge*) on some condition, e.g. once a maximum number of iterations is reached, or when further iterations would yield little or no improvement to the final solution.

BigData Frameworks

- Provide tools to implement and execute complex algorithms in parallel on clusters of networked machines.
- Programmers do not have to manage the parallel execution of their algorithm on the cluster, or faults with individual machines.

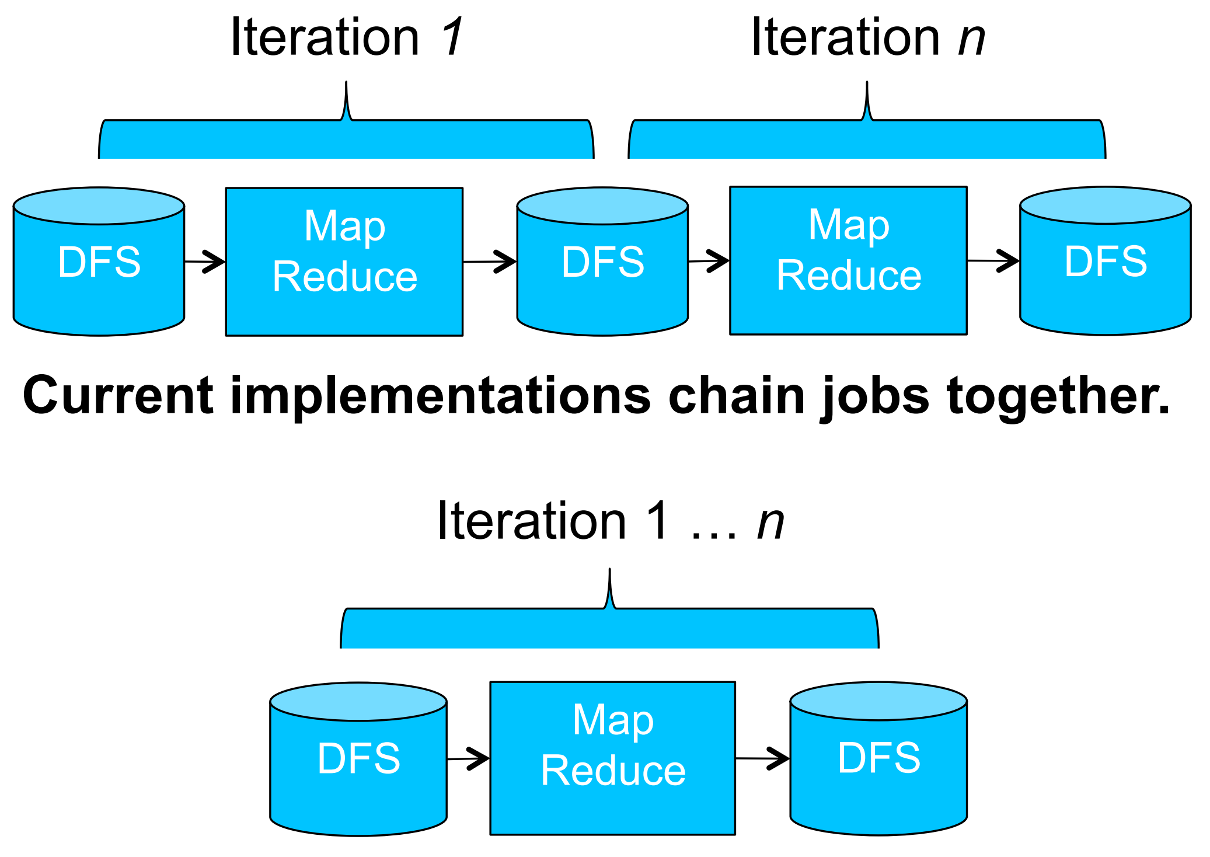
MapReduce

- Programming model breaks computation into *map* and *reduce* phases.
- Designed for batch processing, not iterations.
- Popularised by the Hadoop [3] big data framework.

PROBLEM AND MOTIVATION

Iterations with MapReduce

- Iterations can be implemented by chaining MapReduce jobs together. However, this can be inefficient.



Current implementations chain jobs together.

Ideal implementations runs as one job

Fig 1. Current and ideal iterative MapReduce implementations. Adapted from [6].

- Two approaches have been taken to address this inefficiency.
  1. Improve MapReduce adding support for iterations [6].
  2. Replace MapReduce with new programming models that support iterations by default [4,5].

Motivation

- New big data frameworks Spark [5] and Flink [2] still in their infancy.
- Support for iterative algorithms is designed and implemented in an ad hoc manner. It is not easy to compare the features and performance of Spark and Flink against one another and Hadoop.

METHODOLOGY

- Select two new big data frameworks – Spark and Flink – in addition to Hadoop MapReduce.
- Implement two popular iterative machine learning algorithms; the K-Means clustering algorithm and the Apriori frequent item set mining algorithm.
- Compare how each framework supports the implementation of these algorithms.
- Investigate each framework's runtime performance and fault tolerance features in a cluster environment.

CONTRIBUTIONS

- Evaluation of MapReduce and these new big data frameworks. Comparison of their support for; *sharing state, convergence, execution, and fault tolerance.*
- Recommendation guide examining a range of criteria such as; *understandability, usability, practicality,* and *performance.*

IMPLEMENTATIONS

Requirements and Support

- *State* – is shared state available throughout the *algorithms* execution.
- Convergence – does the framework provide mechanism to *manage* convergence, or must it be *custom* built.
- Execution – is the algorithm run at an *iteration* level where the programmer must collect information from each iteration before starting the next, or does the framework execute the entire *algorithm* and handle iterations as part of its API.
- Fault Tolerance – is the framework able to recover the iterative algorithm's state.

	State	Convergence	Execution	Fault
Hadoop	Iteration	Custom, Look back	Iteration	Iteration
Spark	Algorithm	Custom, Look back	Iteration	Iteration
Flink	Operator	Custom and Managed	Algorithm	None

No system-provided fault tolerance for the algorithm implementation's driver.

EXPERIMENT CONFIGURATION

Nodes

- Amazon EC2 M3.large [1]
- 2 x vCPU, Intel Xeon E5-2670 (2.5-2.6GHZ)
- 7.5GiB Memory
- 1 x 32GB SSD, 1 x 10GB EBS SSD

Cluster

- Hadoop YARN 2.2.0
- K-Means experiment ran on two clusters using 50 and 100 nodes respectively.
- Apriori ran on a 10 node cluster,.

Data

- Synthetic transactions 2.5GB. Average transaction length of five.
- Synthetic two dimensional points. 50GB (25 centroids). 100GB (50 centroids).
- Fixed set of initial centroids defined to ensure same execution across all runs.

PERFORMANCE RESULTS

K-Means

- Flink consistently performed best.
- Spark faster than Hadoop on 50GB, 50 nodes. Slower on 100GB, 100 nodes.

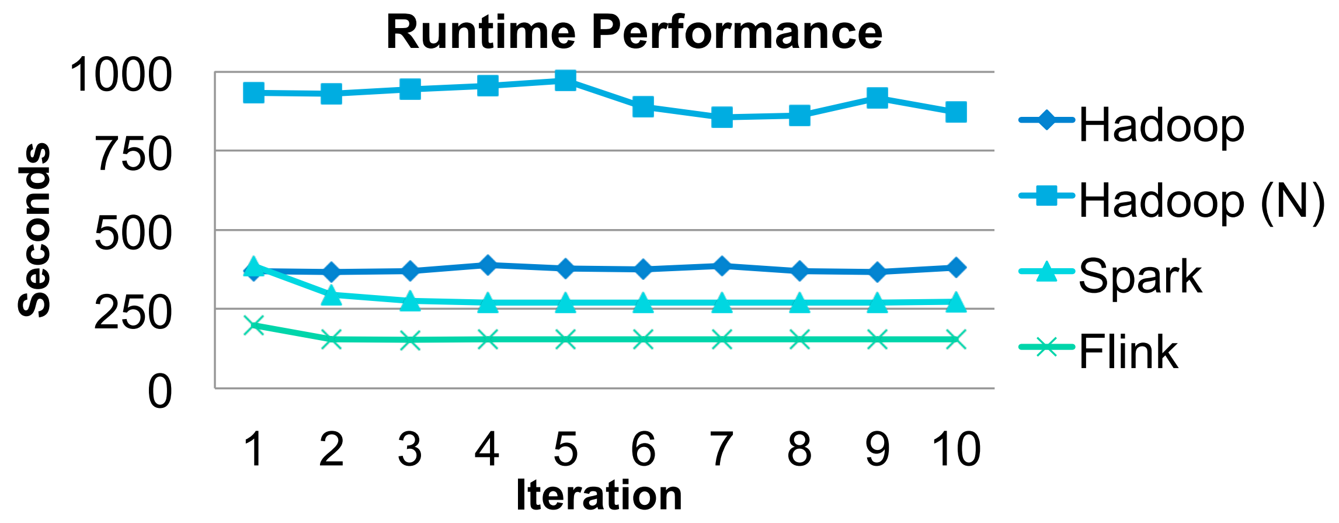


Fig 2. Average of three runs. 50 nodes on 50GB synthetic point dataset. (N) Hadoop with no combiner.

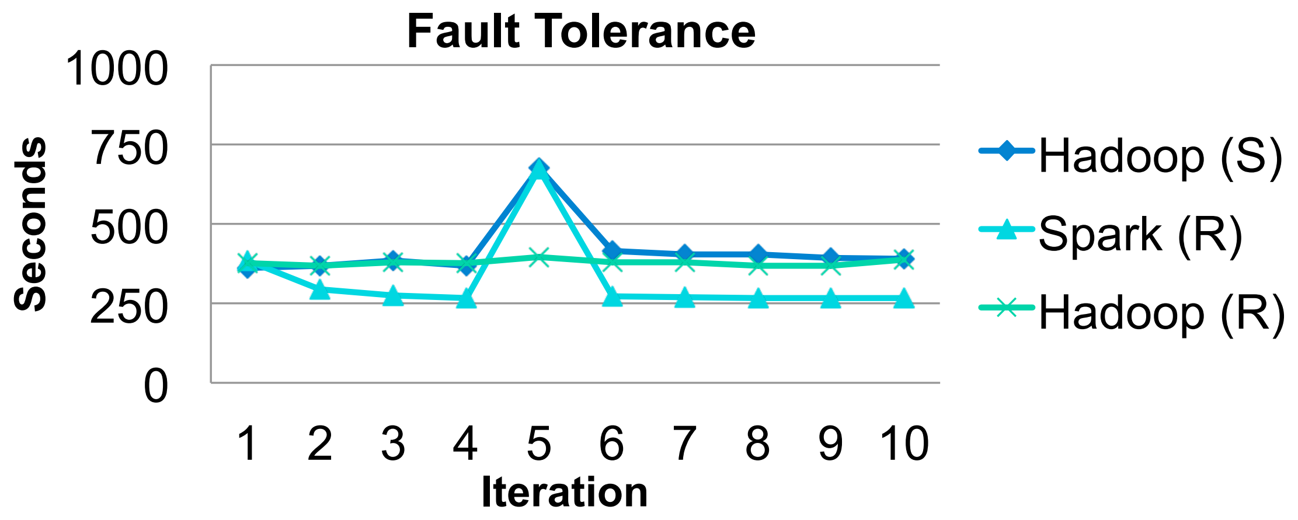


Fig 3. Repeat of Fig 2. In the fifth iteration a fault is created. (S) shutdown a node, (R) restarted a node.

Apriori

- Spark performed 10x slower than Hadoop.
- Flink crashed, exhausting memory, despite working on small datasets.

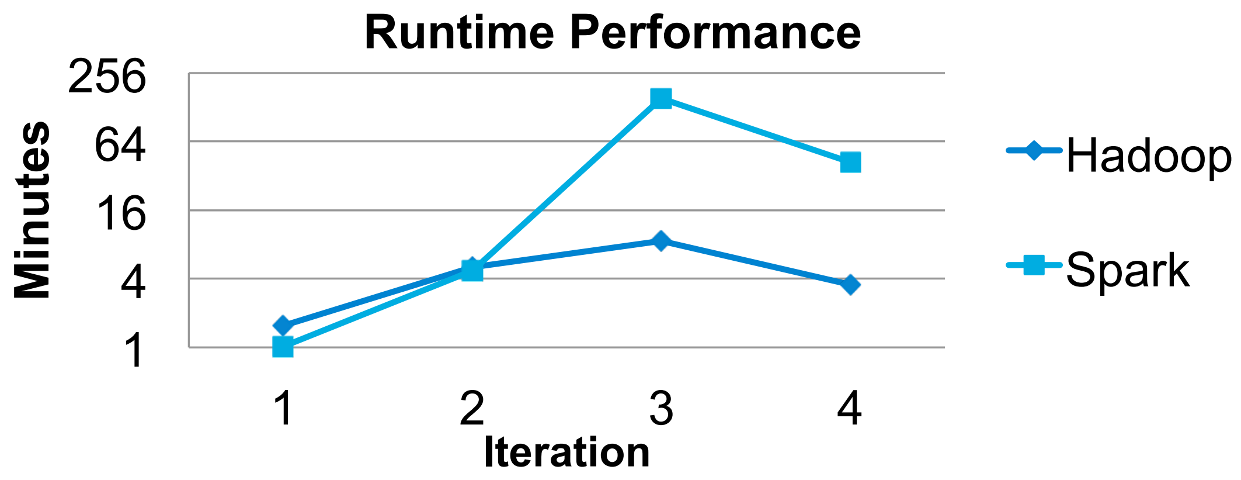


Fig 4. Average runtime of three runs. 10 nodes on 2.5GB synthetic transaction dataset.

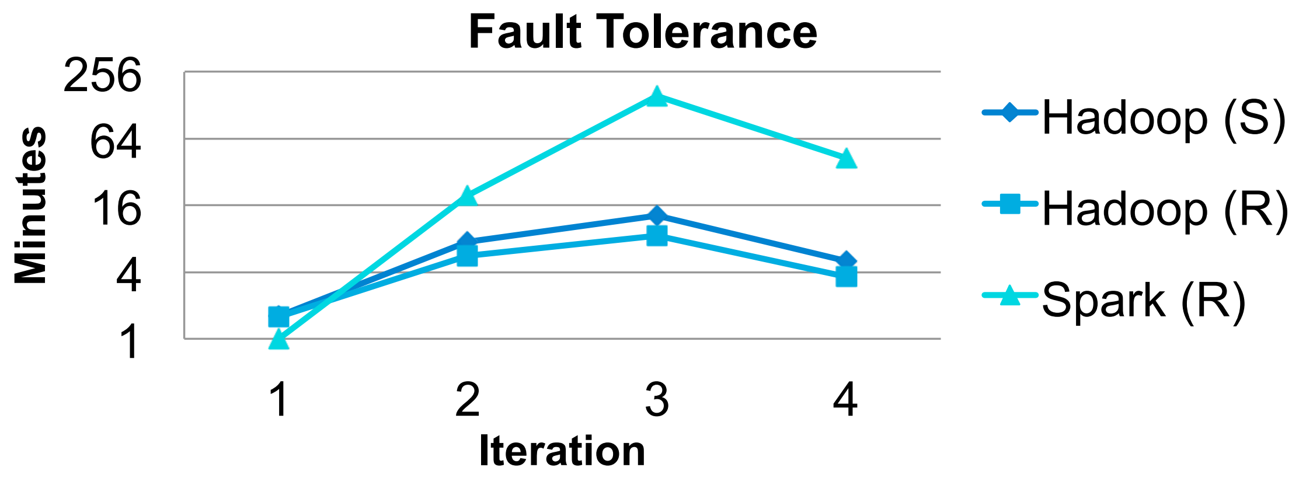


Fig 5. Repeat of Fig 4. In the second iteration a fault is created. (S) shutdown a node, (R) restarted a node.

RECOMMENDATIONS

	Usability	Understandability	Performance	Practicality
1	Spark	Hadoop	Flink	Spark
2	Hadoop	Spark	Spark	Hadoop
3	Flink	Flink	Hadoop	Flink

- *Usability* –the amount of programming effort required to implement iterative algorithms.
- *Understandability* – the amount of effort required to design an algorithm using the runtime control provided by the framework.
- *Performance* – execution time in terms of cluster size and data size.
- *Practicality* – suitability of running the framework in a shared environment.

REFERENCES

[1] Amazon. AWS EC2 Instances. 2014. URL: http://aws.amazon.com/ec2/instance-types/. (Accessed on: 20th Sep. 2014).
[2] *Apache Flink*. Apr. 2014. URL: http://flink.incubator.apache.org/ (Accessed on: 26th Aug. 2014).
[3] *Apache Hadoop*. Apr. 2014. URL: http://hadoop.apache.org/ (Accessed on: 20th Apr. 2014).
[4] Stephan Ewen et al. "Spinning fast iterative data flows". In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1268–1279.
[5] Matei Zaharia et al. "Spark: cluster computing with working sets". In: Proceedings of the 2nd
USENIX conference on Hot topics in cloud computing. 2010, pp. 10–10.
[6] Yanfeng Zhang et al. "imapreduce: A distributed computing framework for iterative computation". In: *Journal of Grid Computing* 10.1 (2012), pp. 47–68.