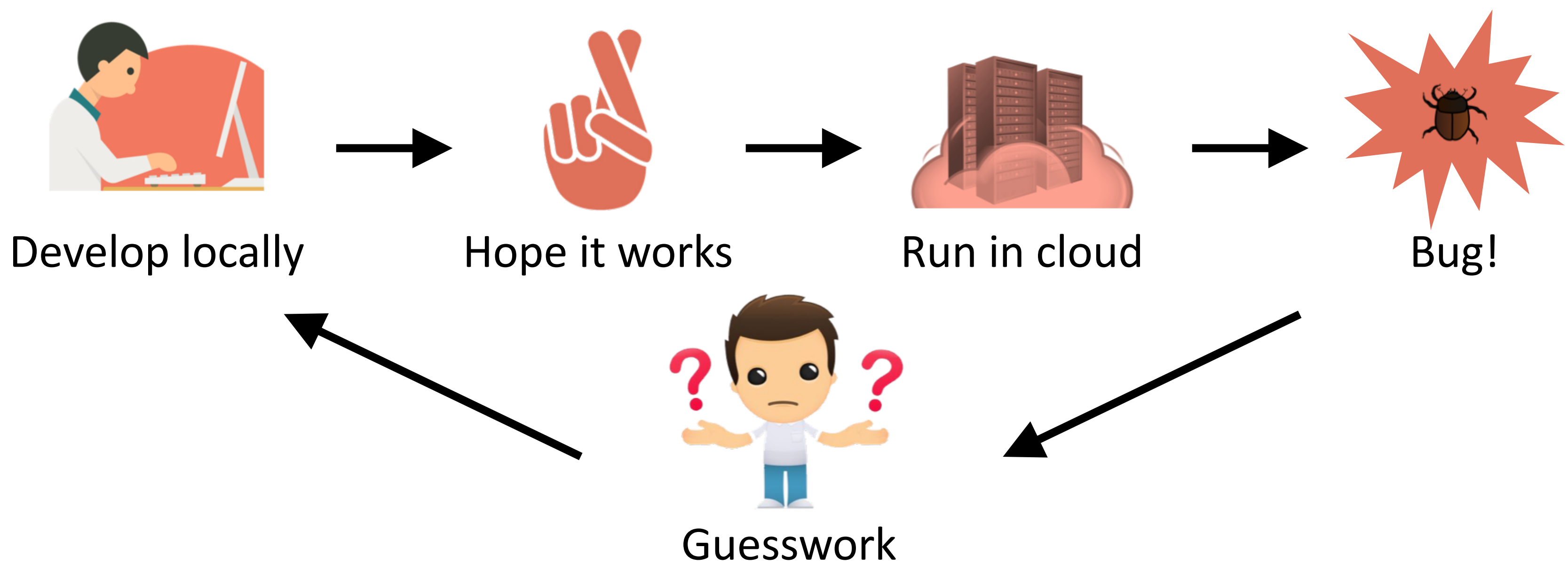
Automated Debugging of Big Data Analytics in Apache Spark Using BigSift

Muhammad Ali Gulzar   Miryung Kim
University of California, Los Angeles

#Res3SAIS

Big Data Debugging in the Dark



Google Map Reduce

hadoop

Spark

HIVE

BigDebug: Debugging Primitives for Interactive Big Data Processing

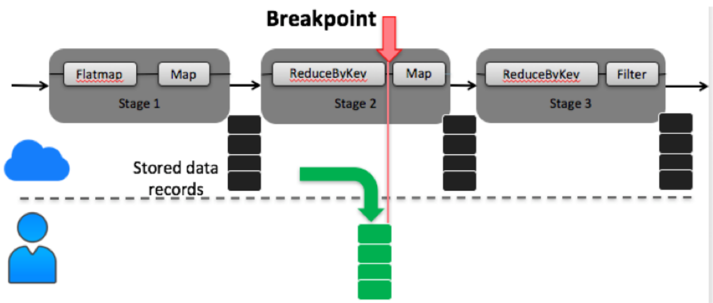
ICSE 2016

Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali Tyson Condie, Todd Millstein, Miryung Kim

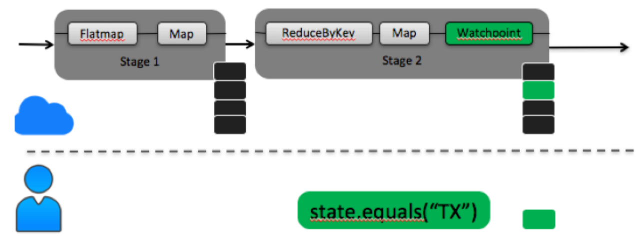
Presented at [Spark Summit 2017]

Interactive Debugging with BigDebug

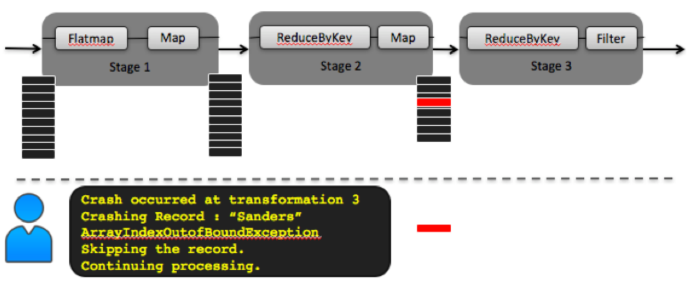
1. Simulated Breakpoint



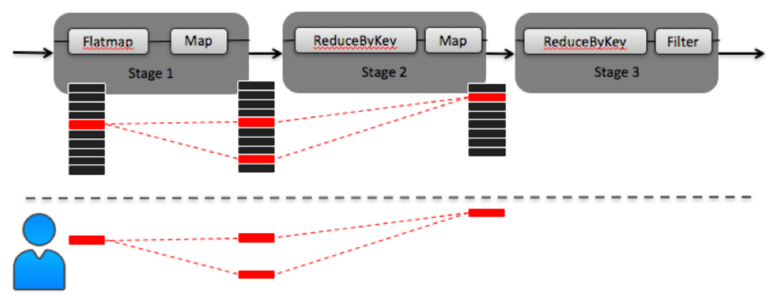
2. On Demand Guarded Watchpoint



3. Crash Culprit Identification



4. Backward and Forward Tracing



Titian: Data Provenance support in Spark

VLDB 2015

Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, Tyson Condie

Data Provenance – in SQL

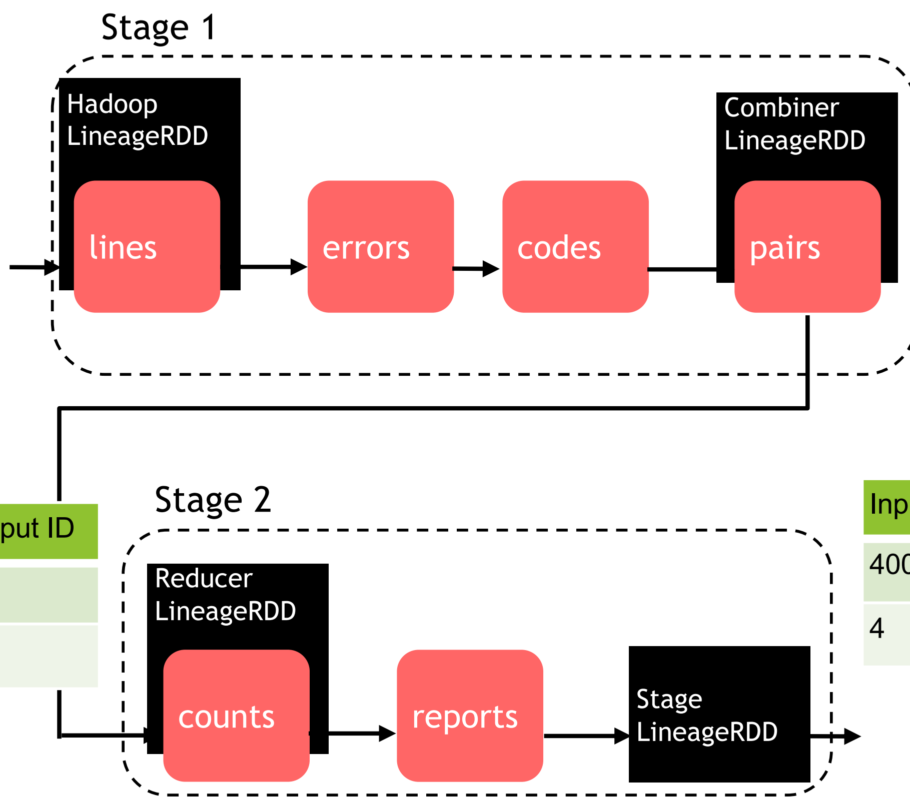
SELECT time, AVG(temp)
FROM sensors
GROUP BY time

Sensors			
Tuple-ID	Time	Sendor-ID	Temperature
T1	11AM	1	34
T2	11AM	2	35
T3	11AM	3	35
T4	12PM	1	35
T5	12PM	2	35
T6	12PM	3	100
T7	1PM	1	35
T8	1PM	2	35
T9	1PM	3	80

Why ID-2 and ID-3 have those high values?

Result-ID	Time	AVG(temp)	
ID-1	11AM	34.6	
ID-2	12PM	56.6	Outlier
ID-3	1PM	50	Outlier

Step 1: Instrument Workflow in Spark



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
[p1, p2]	400
[p1]	4

Input ID	Output ID
400	id1
4	id2

Step 2: Example Backward Tracing

Hadoop

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Combiner

Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
p1	400

Worker1

Worker3

Reducer

Input ID	Output ID
[p1, p2]	400
[p1]	4

Stage

Input ID	Output ID
400	id1
4	id2

Hadoop

Input ID	Output ID
offset1	id1
…	…

Combiner

Input ID	Output ID
{ id1, …}	400

Input ID	Output ID
p1	400

Worker2

Reducer.Output ID $\bowtie$ Stage.Input ID

Step 2: Example Backward Tracing

Hadoop

Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Combiner

Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
p1	400

Combiner.Output ID $\bowtie$ Reducer.Output ID

Hadoop

Input ID	Output ID
offset1	id1
…	…

Combiner

Input ID	Output ID
{ id1, …}	400

Input ID	Output ID
p1	400

Combiner.Output ID $\bowtie$ Reducer.Output ID

Step 2: Example Backward Tracing

Hadoop	
Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Combiner	
Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Hadoop.Output ID $\bowtie$ Combiner.Input ID

Hadoop	
Input ID	Output ID
offset1	id1
…	…

Combiner	
Input ID	Output ID
{ id1, …}	400

Hadoop.Output ID $\bowtie$ Combiner.Input ID

BigSift: Automated Debugging of Big Data Analytics in DISC Applications

SoCC 2017

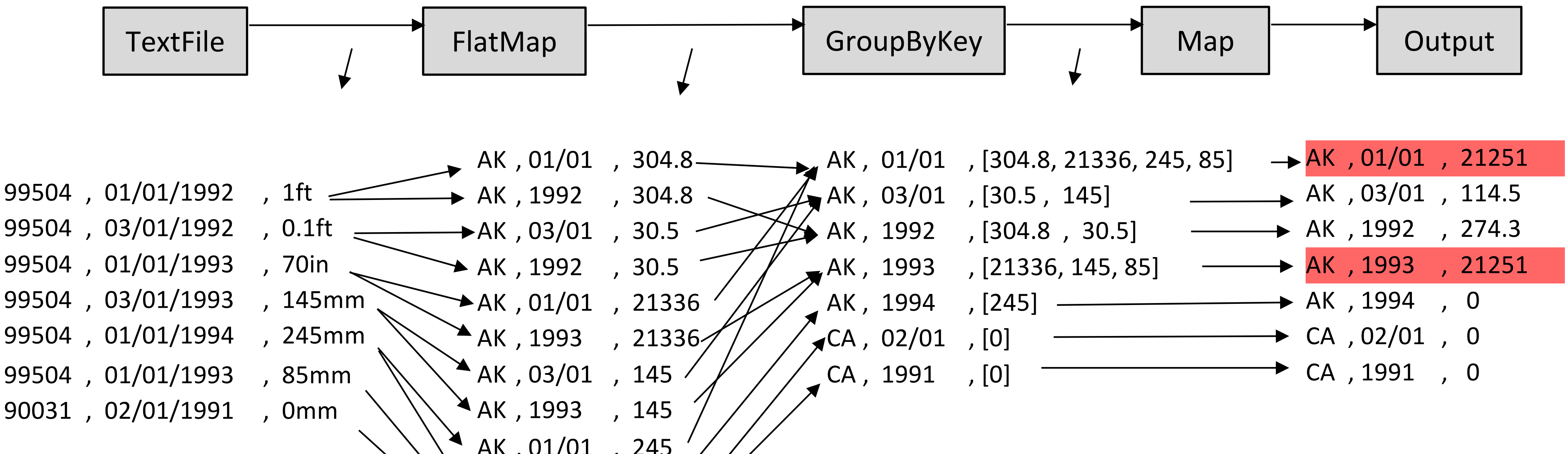
Muhammad Ali Gulzar, Matteo Interlandi, Xueyuan Han, Tyson Condie, Miryung Kim

Motivating Example for Automated Debugging with BigSift

- Alice writes a Spark program that identifies, **for each state** in the US, the **delta between the minimum and the maximum** snowfall reading for **each day of any year** and **for any particular year**.
- An input data record that measures 1 foot of snowfall on January 1st of Year 1992, in the 99504 zip code (Anchorage, AK) area, appears as

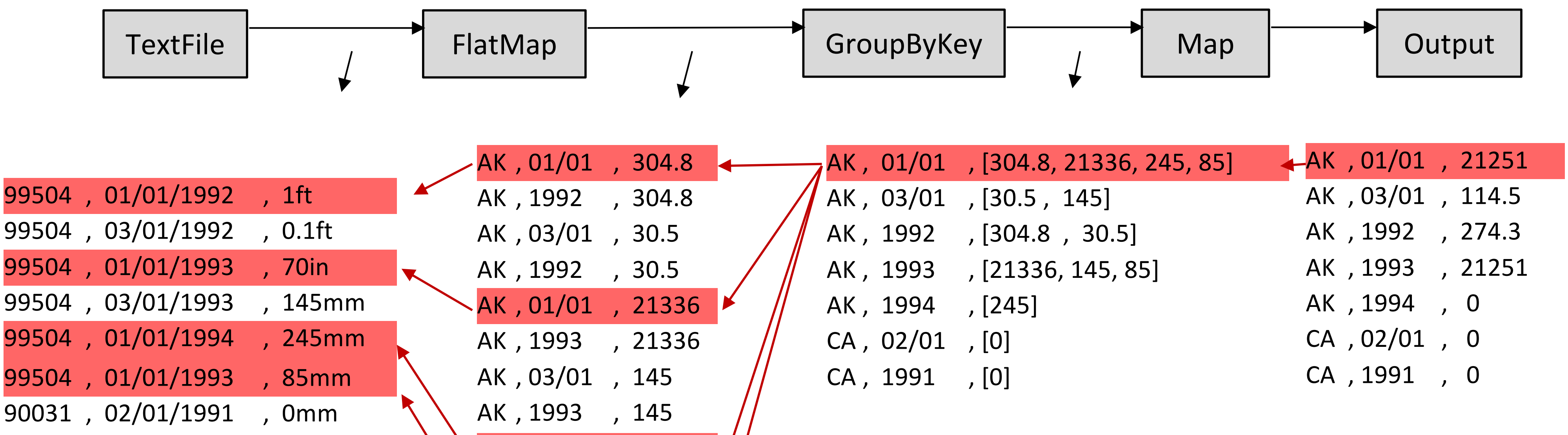
99504 , 01/01/1992 , 1ft

Debugging Incorrect Output



It is challenging because the input records is very large and the program involves computing min and max and a unit conversion

Data Provenance with Titian [VLDB '15]



It over-approximates the scope of failure-inducing inputs *i.e.* records in the faulty key-group are all marked as faulty

Delta Debugging

- Alice tests the output from different input configurations using a test function

TextFile → FlatMap → GroupByKey → Map → Output

```scala
def test(key:String, delta: Float) : Boolean = {
        delta < 6000
}
```

1
99504 , 01/01/1992
99504 , 03/01/1992
99504 , 01/01/1993 , 70in

2
99504 , 03/01/1993 , 145mm
99504 , 01/01/1994 , 245mm
99504 , 01/01/1993 , 85mm
90031 , 02/01/1991 , 0mm

AK , 01/01 , 304.8
AK , 1993 , 21336
AK , 03/01 , 145
AK , 1993 , 145
AK , 01/01 , 245
AK , 1994 , 245
....

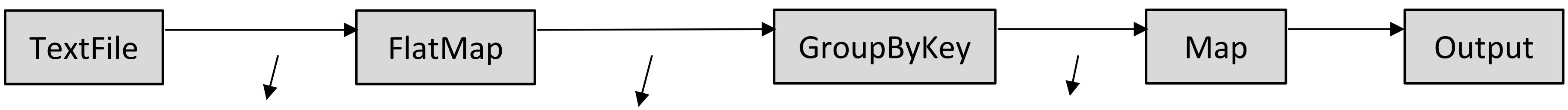
AK , 03/01 , [30.5 , 145]
AK , 1992 , [304.8 , 30.5]
AK , 1993 , [21336, 145, 85]
AK , 1994 , [245]
CA , 02/01 , [0]
CA , 1991 , [0]

AK , 01/01 , 21251
AK , 03/01 , 114.5
AK , 1992 , 274.3
AK , 1993 , 21251
AK , 1994 , 0
CA , 02/01 , 0
CA , 1991 , 0

Run 1 fails the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.



1	99504	01/01/1992	1ft
	99504	03/01/1992	0.1ft
2	99504	01/01/1993	70in

AK	01/01	304.8
AK	1992	304.8
AK	03/01	30.5
AK	1992	30.5
AK	01/01	21336
AK	1993	21336

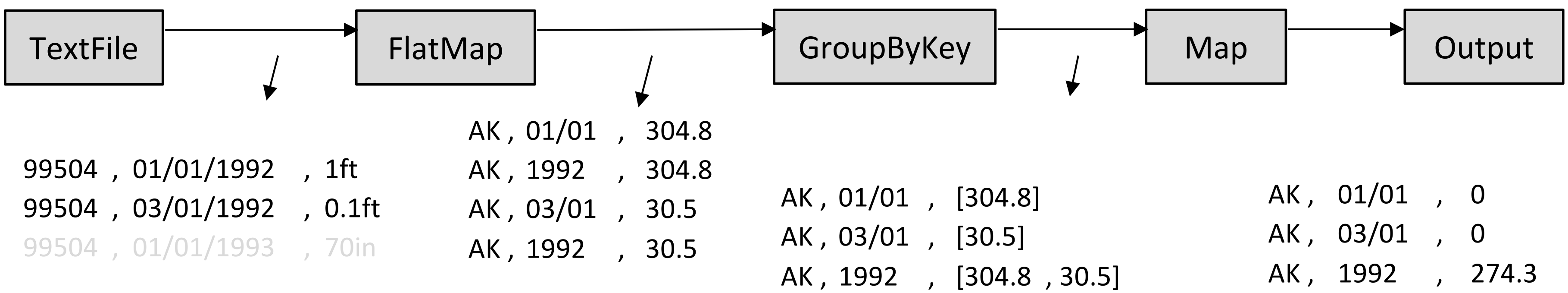
AK	01/01	[304.8, 21336]
AK	03/01	[30.5]
AK	1992	[304.8 , 30.5]
AK	1993	[21336]

AK	01/01	21031.2
AK	03/01	0
AK	1992	274.3
AK	1993	0

Run 2 fails the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.



Run 3 passes the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.

TextFile → FlatMap → GroupByKey → Map → Output

99504 , 01/01/1992 , 1ft
99504 , 03/01/1992 , 0.1ft
99504 , 01/01/1993 , 70in

AK , 01/01 , 21336
AK , 1993 , 21336

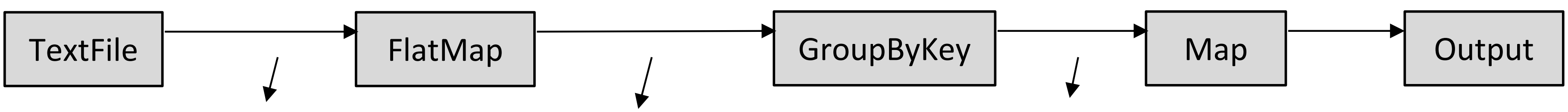
AK , 01/01 , [21336]
AK , 1993 , [21336]

AK , 01/01 , 0
AK , 1993 , 0

Run 4 passes the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.



99504 , 01/01/1992 , 1ft
99504 , 03/01/1992 , 0.1ft
99504 , 01/01/1993 , 70in

AK , 01/01 , 304.8
AK , 1992 , 304.8

AK , 01/01 , [304.8]
AK , 1992 , [304.8]

AK , 01/01 , 0
AK , 1992 , 0

Run 5 passes the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.

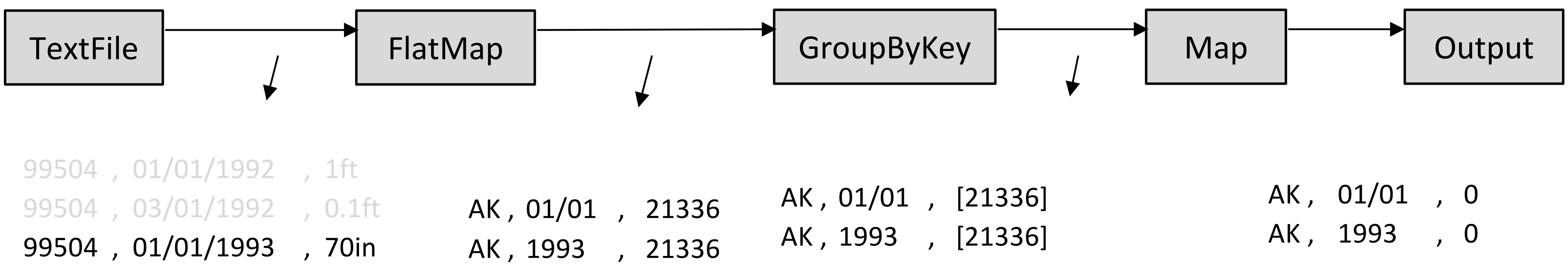
TextFile → FlatMap → GroupByKey → Map → Output

99504 , 01/01/1992 , 1ft			
99504 , 03/01/1992 , 0.1ft	AK , 03/01 , 30.5	AK , 03/01 , [30.5]	AK , 03/01 , 0
99504 , 01/01/1993 , 70in	AK , 1992 , 30.5	AK , 1992 , [30.5]	AK , 1992 , 0

Run 6 passes the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.



Run 7 passes the test

Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.

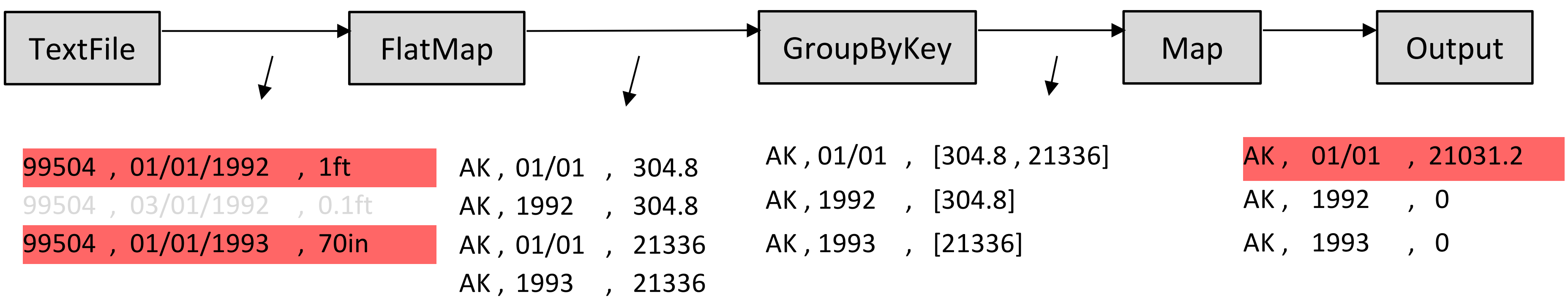
TextFile → FlatMap → GroupByKey → Map → Output

99504 , 01/01/1992 , 1ft	AK , 03/01 , 30.5	AK , 03/01 , [30.5]	AK , 03/01 , 0
99504 , 03/01/1992 , 0.1ft	AK , 1992 , 30.5	AK , 1992 , [30.5]	AK , 1992 , 0
99504 , 01/01/1993 , 70in	AK , 01/01 , 21336	AK , 01/01 , [21336]	AK , 01/01 , 0
	AK , 1993 , 21336	AK , 1993 , [21336]	AK , 1993 , 0

Run 8 passes the test

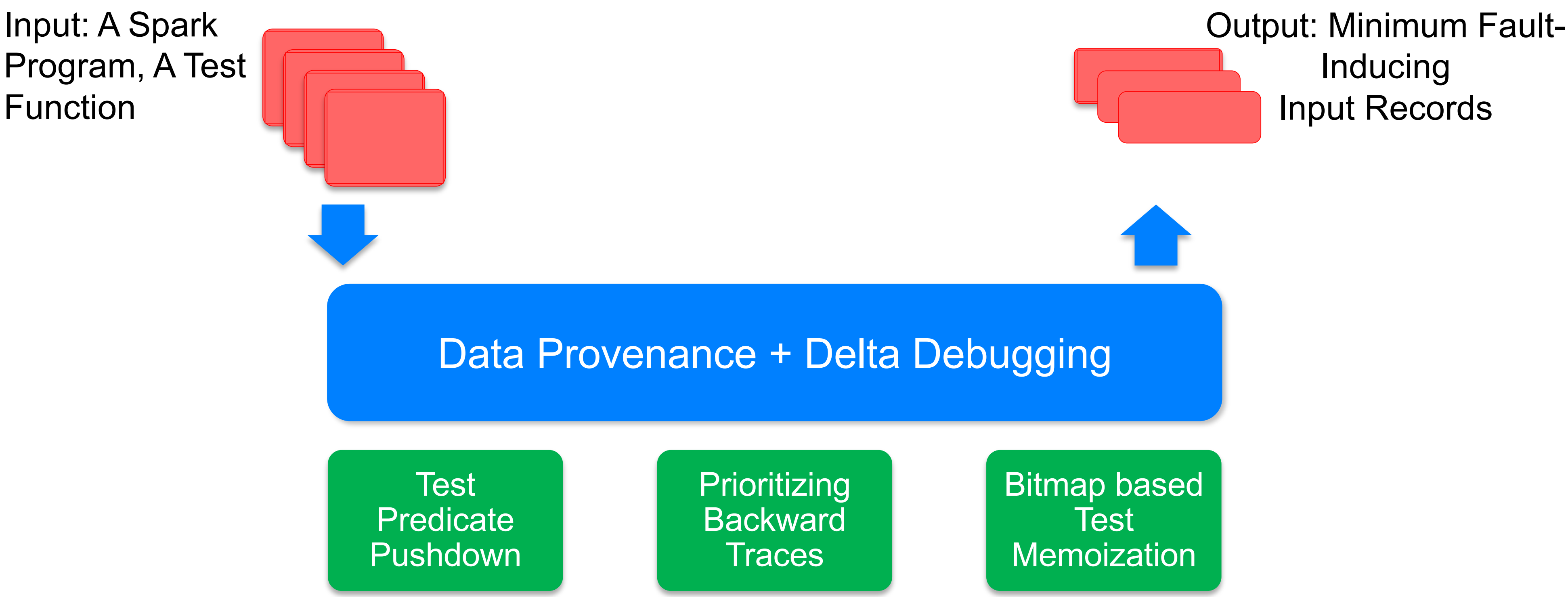
Delta Debugging

- DD is an systematic debugging procedure guided by a test function. DD is **inefficient** because it does not eliminate irrelevant input records upfront.



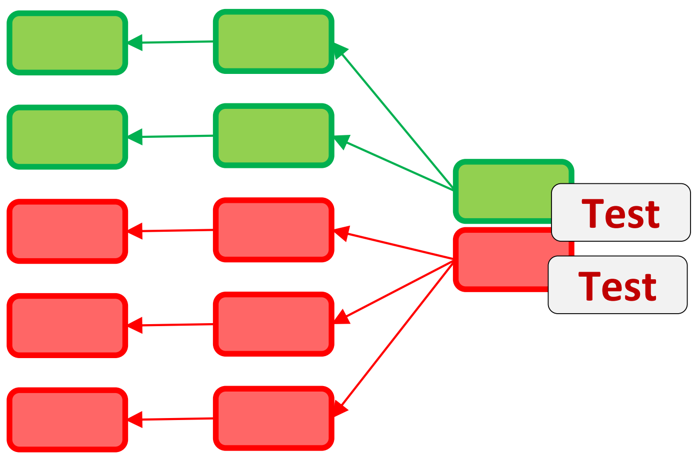
Run 9 fails the test

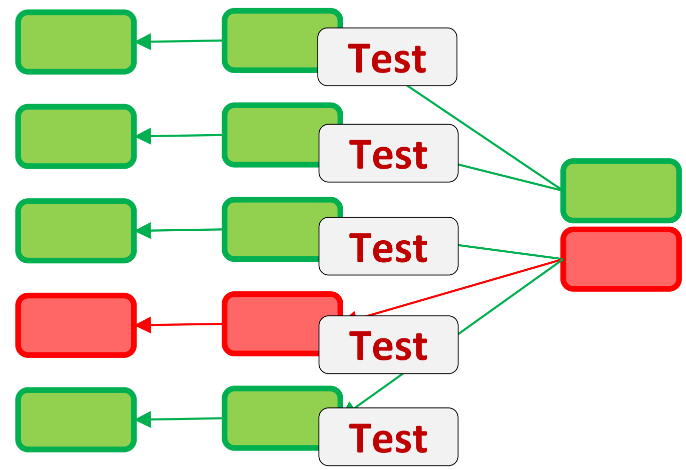
Automated Debugging in DISC with BigSift



Optimization 1: Test Predicate Pushdown

- **Observation:** During backward tracing, data provenance traces through all the partitions even though only a few partitions are faulty

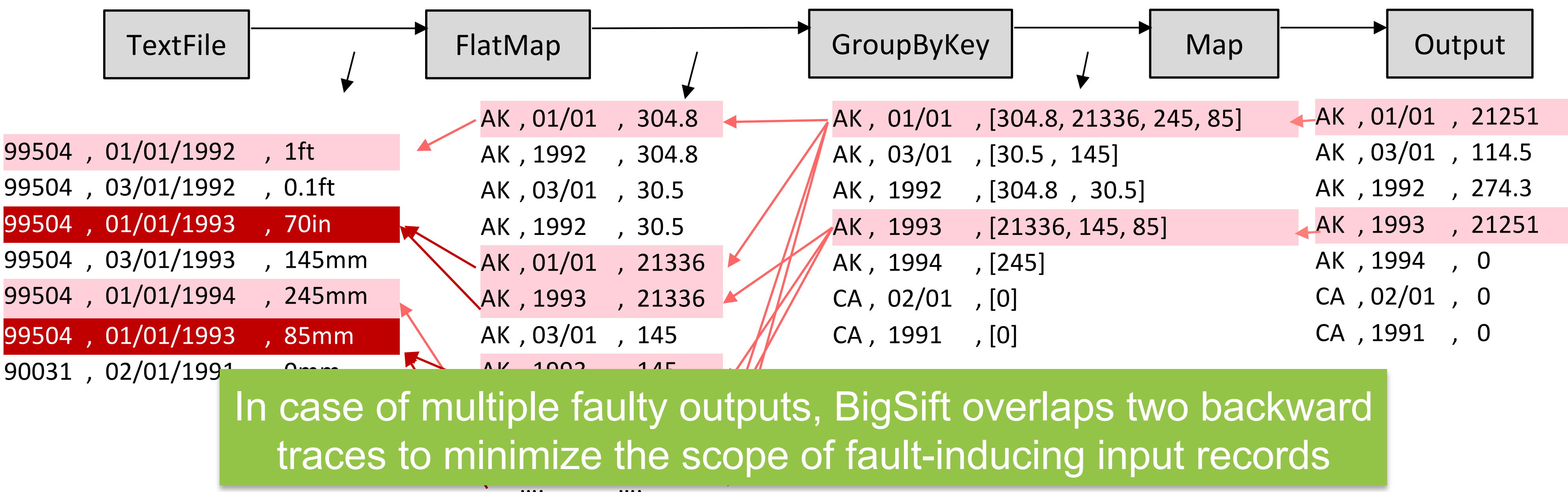




If applicable, BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.

Optimization 2 :Prioritizing Backward Traces

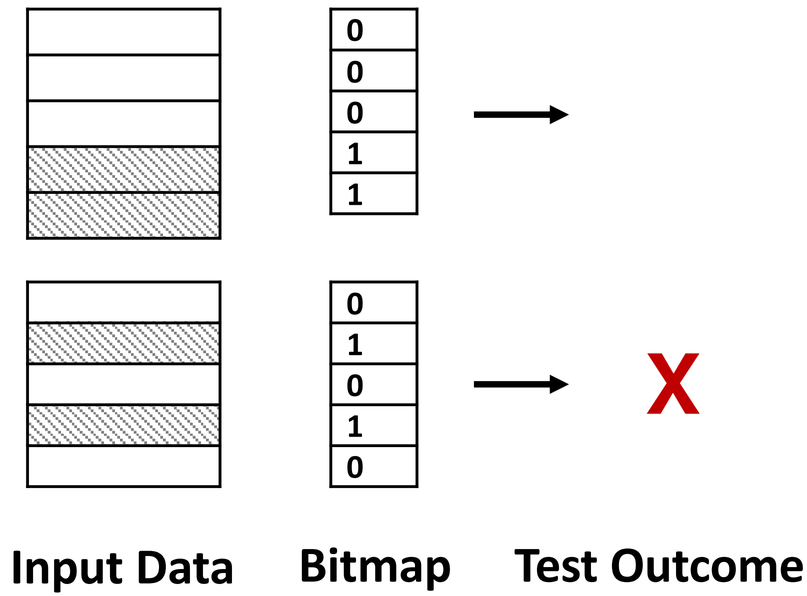
- **Observation:** The same faulty input record may contribute to multiple output records failing the test.



In case of multiple faulty outputs, BigSift overlaps two backward traces to minimize the scope of fault-inducing input records

Optimization 3: Bitmap based Test Memoization

- **Observation:** Delta debugging may try running a program on the same subset of input redundantly.
- BigSift leverages bitmap to compactly encode the offsets of original input to refer to an input subset



We use a bitmap based test memoization technique to avoid redundant testing of the same input dataset.

Demo

Debugging Time

Subject Program		Running Time (sec)	Debugging Time (sec)		
Subject Program	Fault	Original Job	DD	BigSift	Improvement
Movie Histogram	Code	56.2	232.8	17.3	13.5X
Inverted Index	Code	107.7	584.2	13.4	43.6X
Rating Histogram	Code	40.3	263.4	16.6	15.9X
Sequence Count	Code	356.0	13772.1	208.8	66.0X
Rating Frequency	Code	77.5	437.9	14.9	29.5X
College Student	Data	53.1	235.3	31.8	7.4X
Weather Analysis	Data	238.5	999.1	89.9	11.1X
Transit Analysis	Code	45.5	375.8	20.2	18.6X

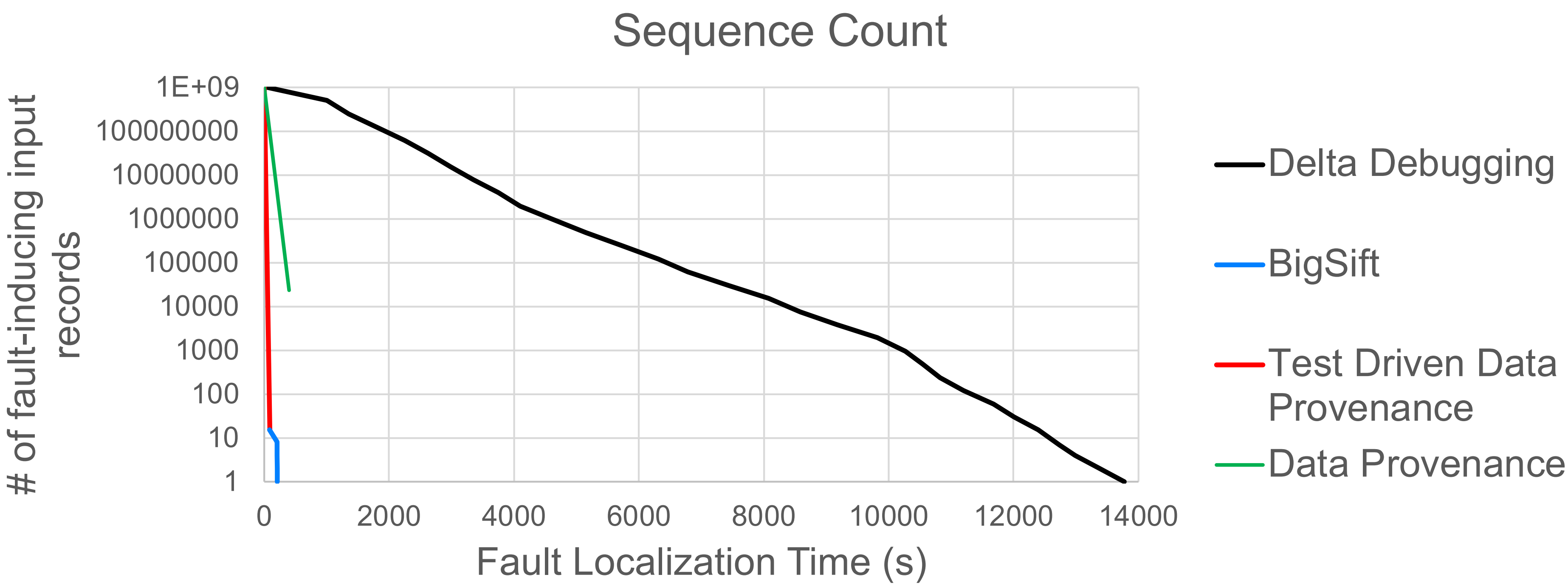
BigSift provides up to a 66X speed up in isolating the precise fault-inducing input records, in comparison to the baseline DD

Debugging Time

Subject Program		Running Time (sec)	Debugging Time (sec)		
Subject Program	Fault	Original Job	DD	BigSift	Improvement
Movie Histogram	Code	56.2	232.8	17.3	13.5X
Inverted Index	Code	107.7	584.2	13.4	43.6X
Rating Histogram	Code	40.3	263.4	16.6	15.9X
Sequence Count	Code	356.0	13772.1	208.8	66.0X
Rating Frequency	Code	77.5	437.9	14.9	29.5X
College Student	Data	53.1	235.3	31.8	7.4X
Weather Analysis	Data	238.5	999.1	89.9	11.1X
Transit Analysis	Code	45.5	375.8	20.2	18.6X

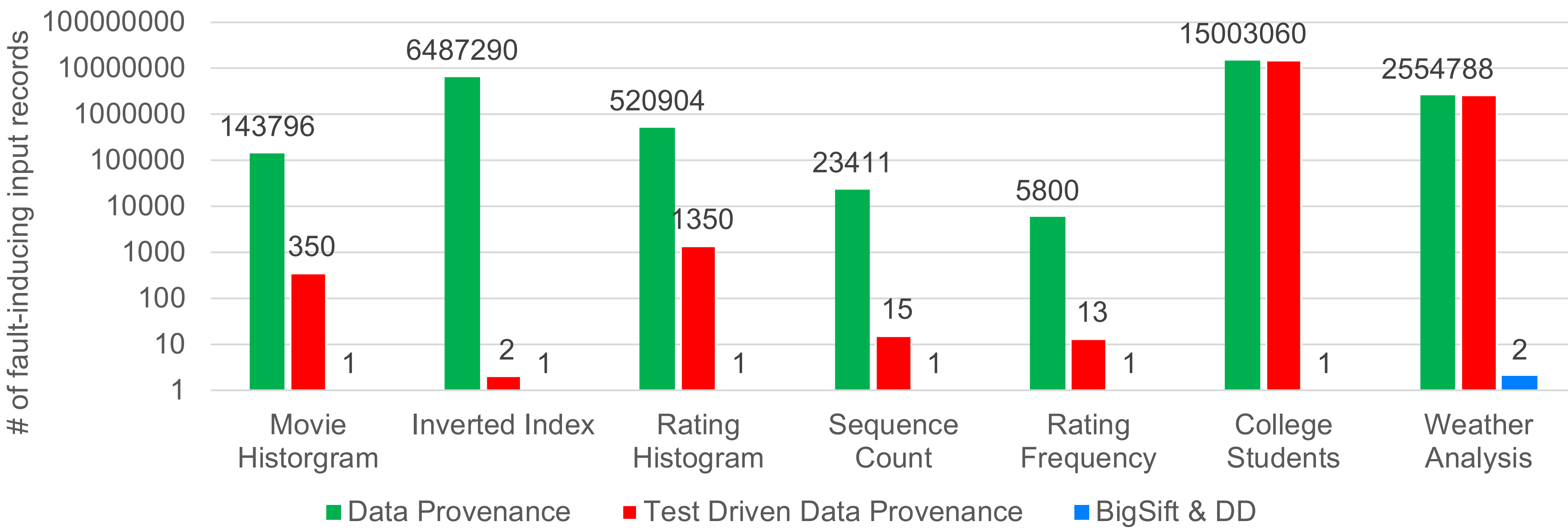
On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

Debugging Time



On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

Fault Localizability over Data Provenance



BigSift leverages DD after DP to continue fault isolation, achieving several orders of magnitude 103 to 107 better precision.

Summary

- BigSift is the first piece of work in automated debugging of big data analytics in DISC.
- BigSift provides **10^3X – 10^7X more precision** than data provenance in terms of fault localizability.
- It provides up to **66X speed up** in debugging time over baseline Delta Debugging.
- In our evaluation we have observed that, on average, BigSift finds the faulty input in **62% less** than the original job execution time.