BigDebug: Interactive Debugger for Big Data Analytics in Apache Spark

MUHAMMAD ALI GULZAR, MATTEO INTERLANDI,
TYSON CONDIE, MIRYUNG KIM

UNIVERSITY OF CALIFORNIA, LOS ANGELES

Developing Big Data Analytics

- Data scientists gather insights from massive quantities of data using data intensive scalable computing systems

Google MapReduce

Spark

HIVE

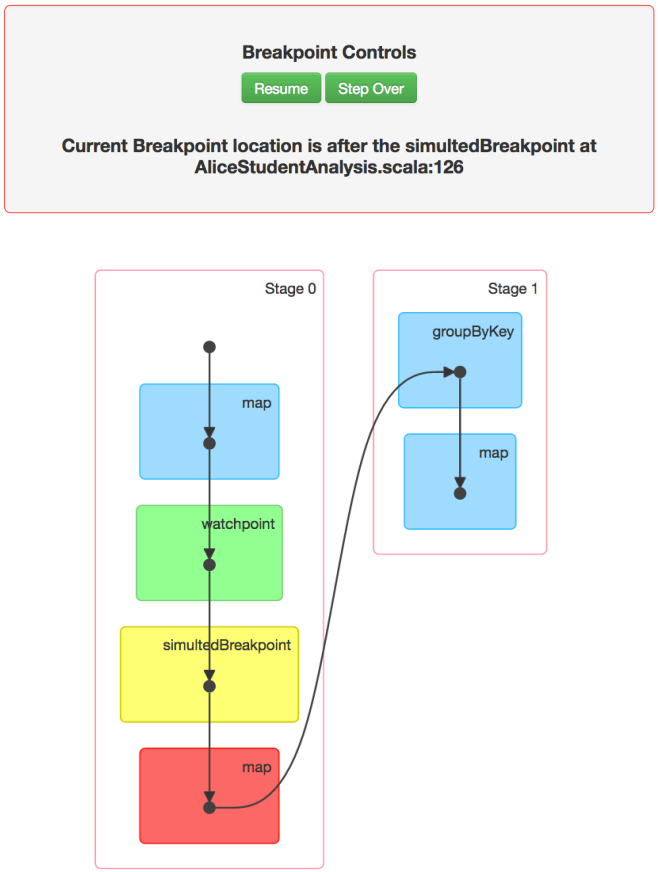
- Debugging of big data analytic workflows on the cloud is time consuming and error-prone

Traditional Debugging in Big Data Analytics

Enabling interactive debugging requires us to **re-think the features of traditional debugger** such as GDB

- Pausing the entire computation on the cloud could reduce throughput
- It is clearly infeasible for a user to inspect billion of records through a regular watchpoint
- Even launching remote JVM debuggers to individual worker nodes cannot scale for big data computing

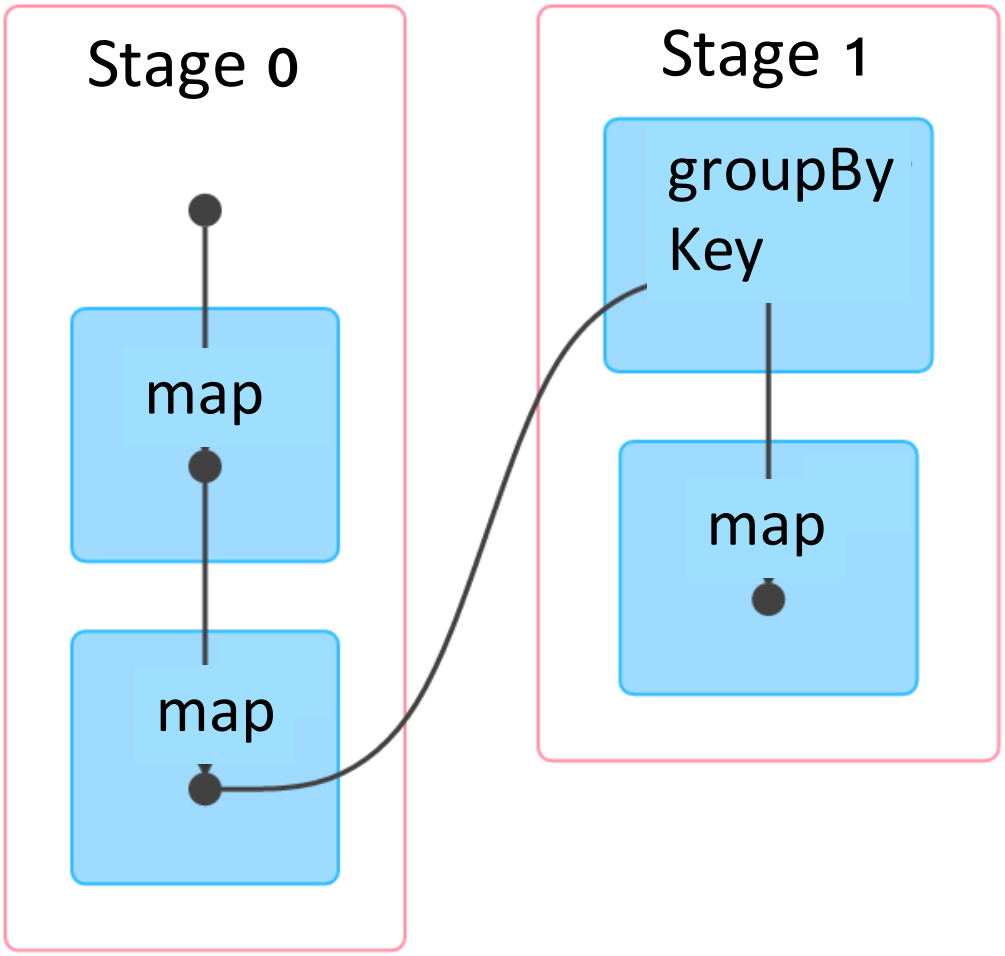
BigDebug : Interactive Debugger for Apache Spark [ICSE 2016]



AliceStudentAnalysis.scala

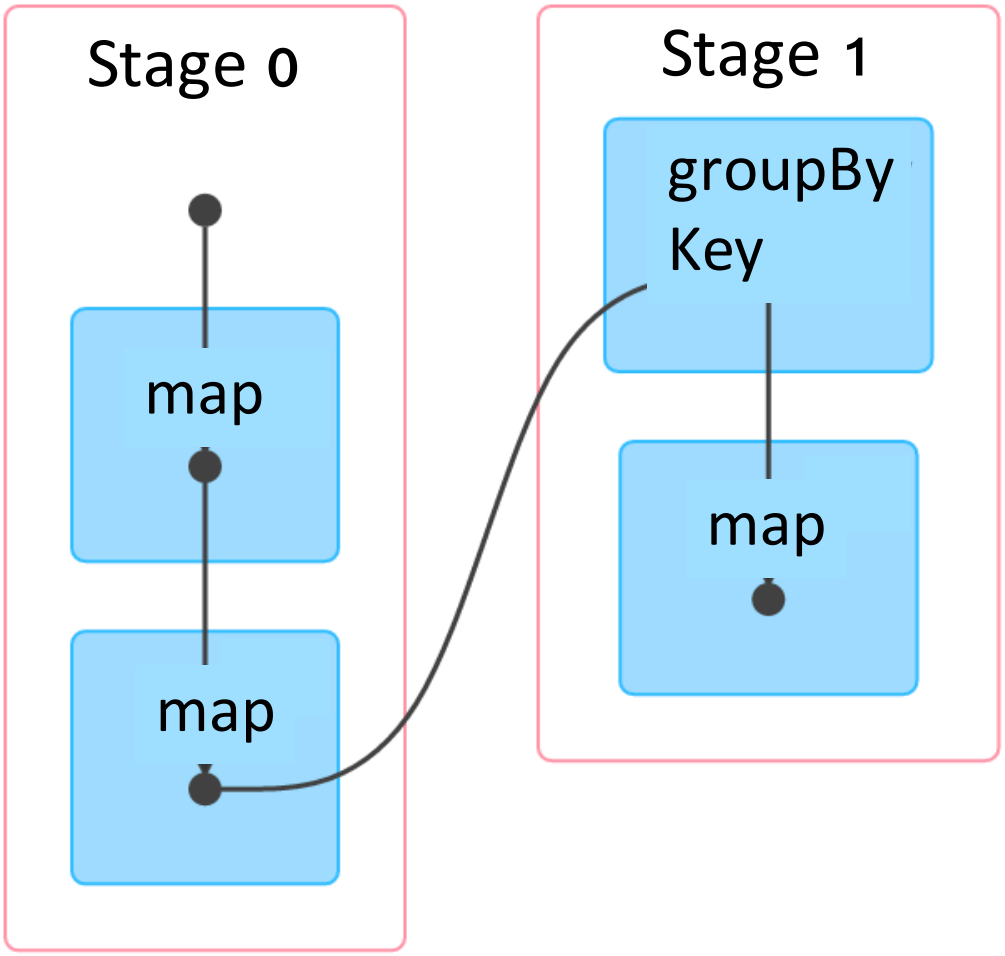
```
110
111 object AliceStudentAnalysisWP {
112
113   val COLLEGEYEAR = List("Sophomore" , "Freshman" , "Junior", "Senior")
114   def main(args: Array[String]): Unit = {
115
116     //set up spark configuration
117     val sparkConf = new SparkConf()
118     val bdconf = new BigDebugConfiguration
119     bdconf.setFilePath("/home/ali/work/temp/git/dsbigdebug/spark-lineage/examples/src/mai
120     //set up spark context
121     val ctx = new SparkContext(sparkConf)
122     ctx.setBigDebugConfiguration(bdconf)
123     //spark program starts here
124     val records = ctx.textFile("/home/ali/Desktop/myfile.txt", 1)
125       .watchpoint(s=> !COLLEGEYEAR.contains(s.split(" ")(2)))
126       .simultedBreakpoint()
127>    val grade_age_pair = records.map(line => {
128       val list = line.split(" ")
129       (list(2), list(3).toInt)
130     })
131     val average_age_by_grade = grade_age_pair.groupByKey
132       .map(pair => {
133       val itr = pair._2.toIterator
134       var moving_average = 0
135       var num = 1
136       while (itr.hasNext) {
137         moving_average = moving_average + itr.next()
138         num = num + 1
139       }
140       (pair._1, moving_average/num)
141     })
142     val out = average_age_by_grade.collect()
143     out.foreach(println)
144   }
```

Feature 1: Simulated Breakpoint



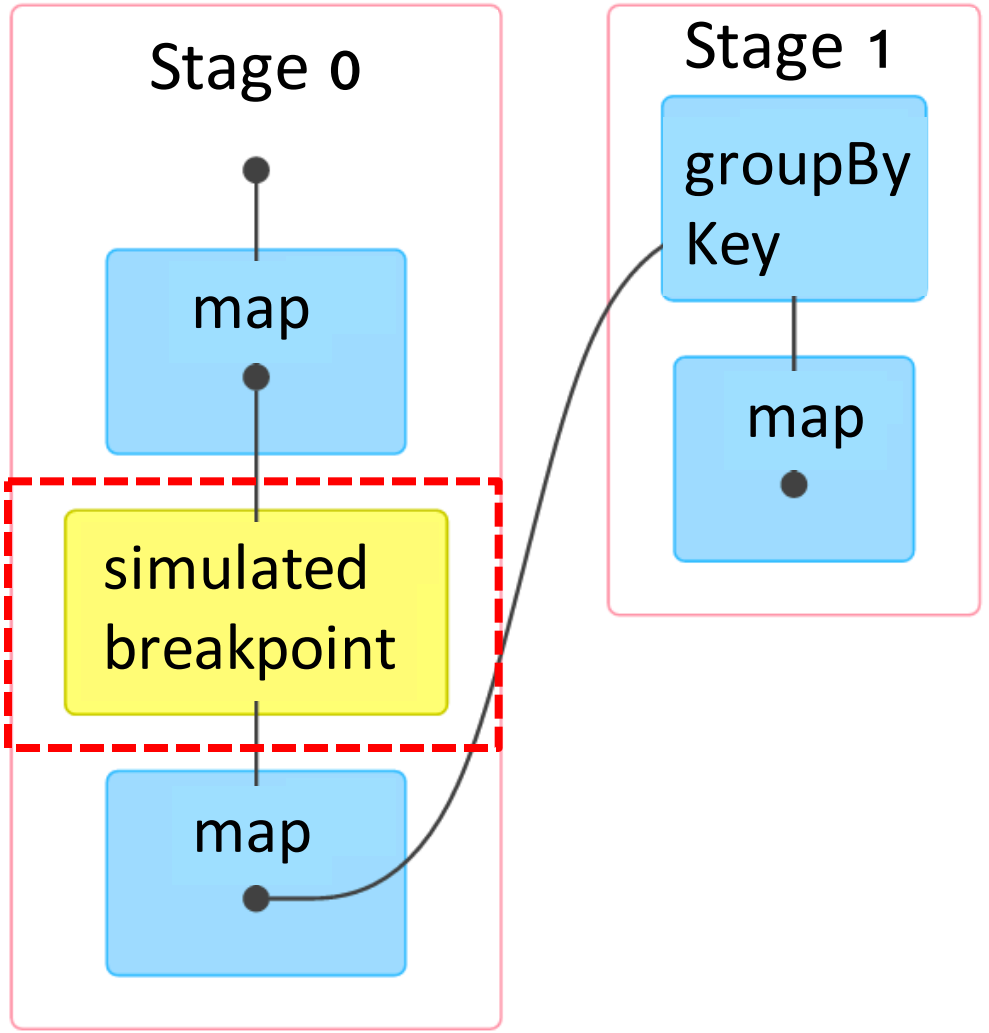

```
125     val records = ctx.textFile("/home/ali/Deskt
126       .simulatedBreakpoint()
127 >   val grade_age_pair = records.map(line => {
```

Feature 1: Simulated Breakpoint



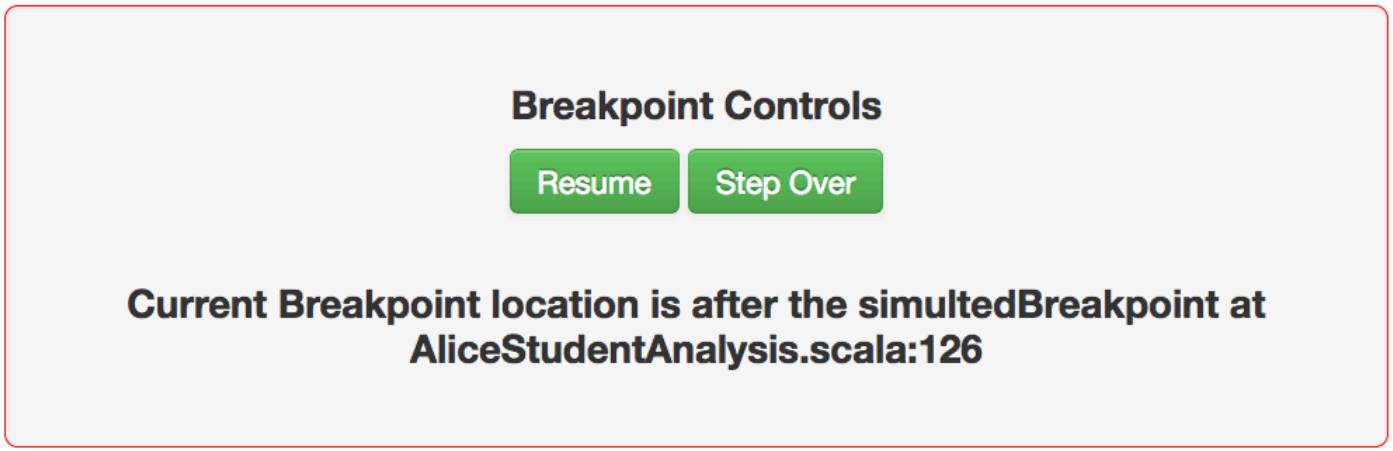

```
125     val records = ctx.textFile("/home/ali/Deskt
126       .simulatedBreakpoint()
127 >   val grade_age_pair = records.map(line => {
```

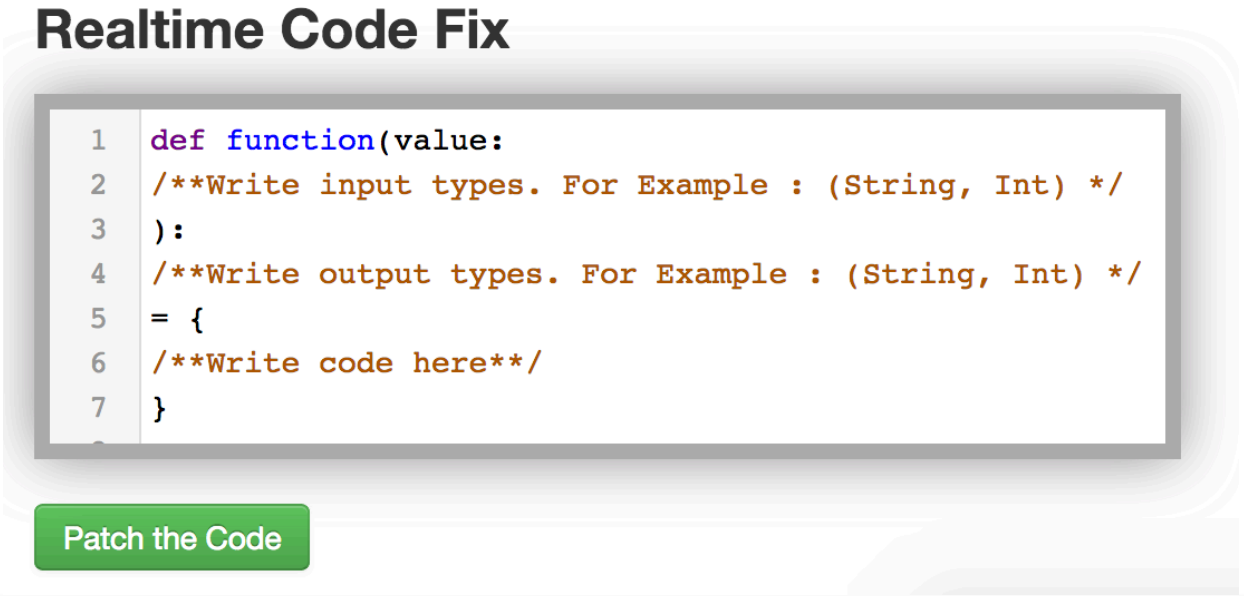
Feature 1: Simulated Breakpoint




```
125     val records = ctx.textFile("/home/ali/Deskt
126       .simulatedBreakpoint()
127 >   val grade_age_pair = records.map(line => {
```

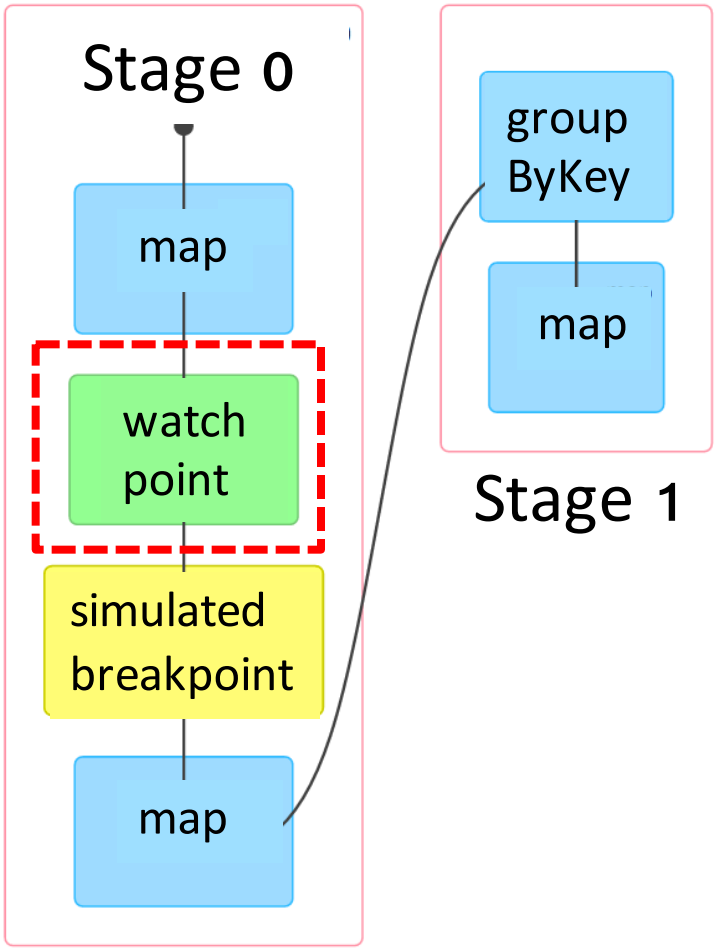
Feature 1: Simulated Breakpoint





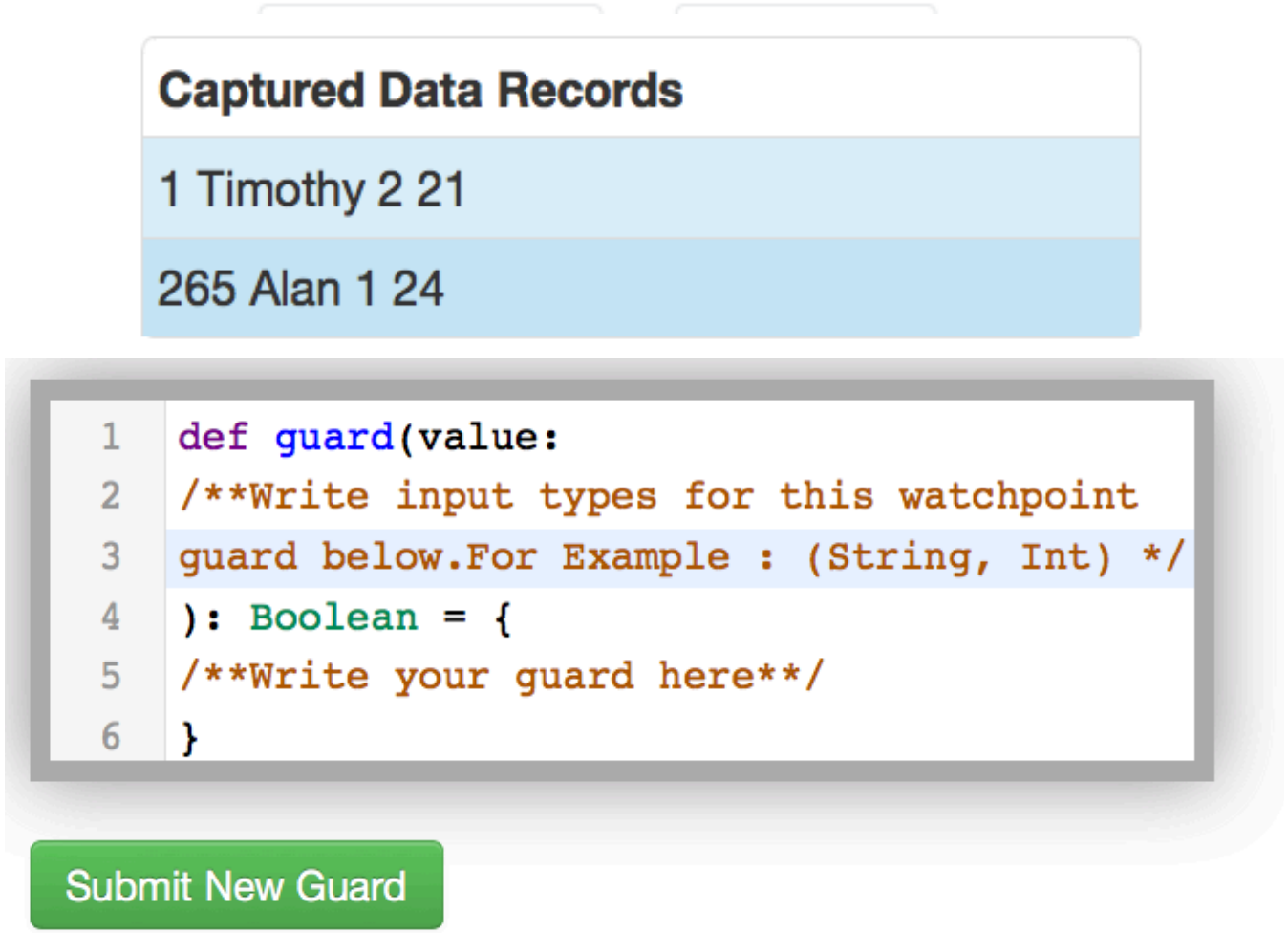
Simulated breakpoint enables user to inspect intermediate program state without pausing the computation

Feature 2: On Demand Guarded Watchpoint



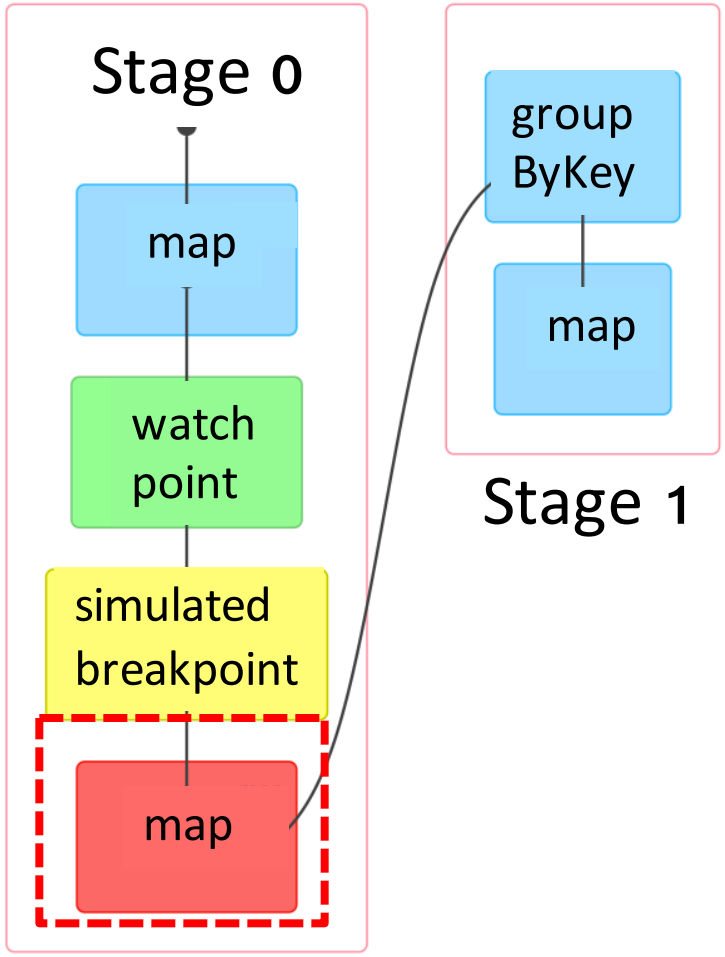

```
124  val records = ctx.textFile("/home/ali/Desktc
125     .watchpoint(s=> !COLLEGEYEAR.contains(s.sp
126     .simulatedBreakpoint()
```

Feature 2: On Demand Guarded Watchpoint



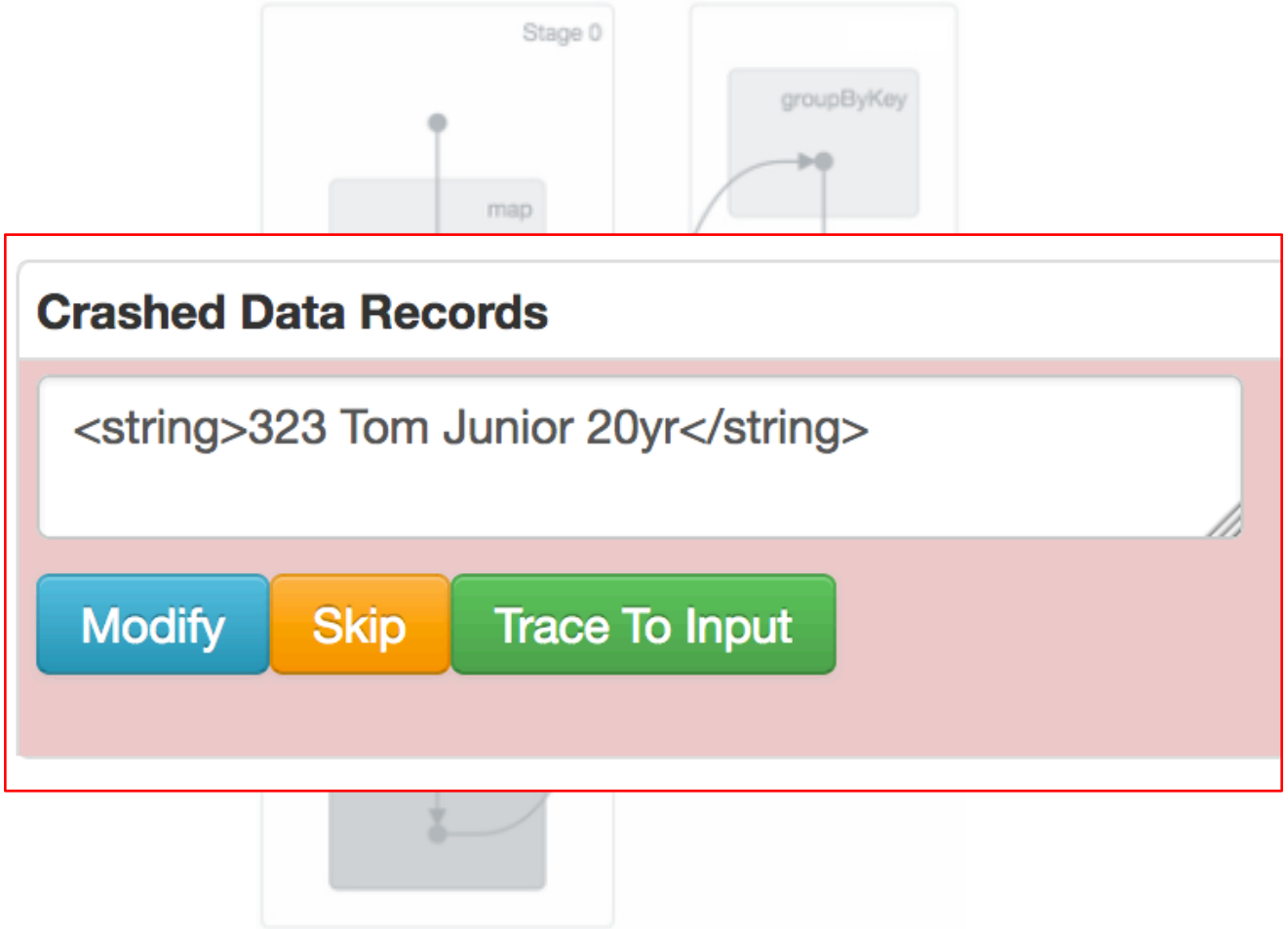
A user can inspect intermediate data using a guard and also update it on the fly

Feature 3: Crash Culprit Identification and Remediation



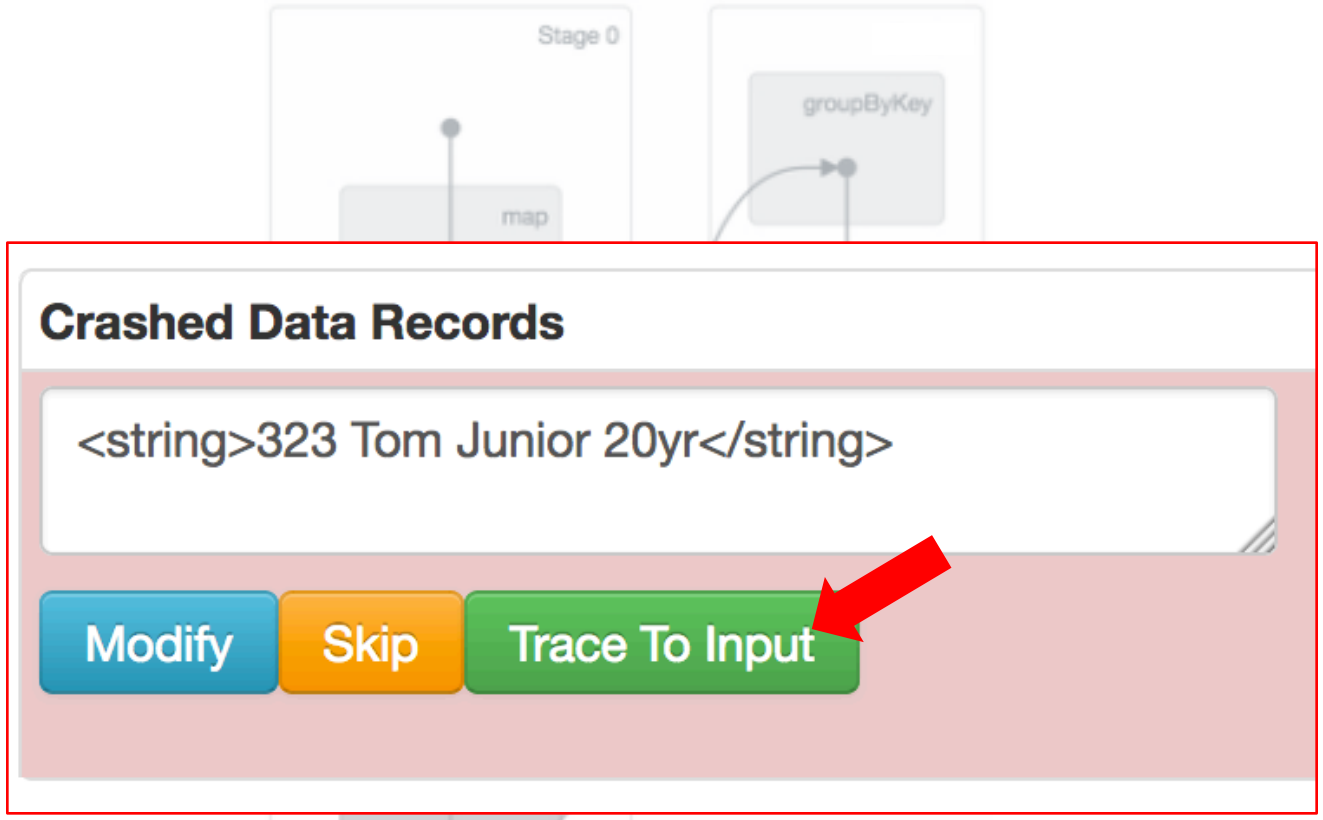

```
125     .watchpoint(s=> !COLLEGEYEAR.contains(s.split(" ")(2)))
126     .simultedBreakpoint()
127   val grade_age_pair = records.map(line => {
```

Feature 3: Crash Culprit Identification and Remediation



A user can use BigDebug to identify the crashing records and remediate from the failure

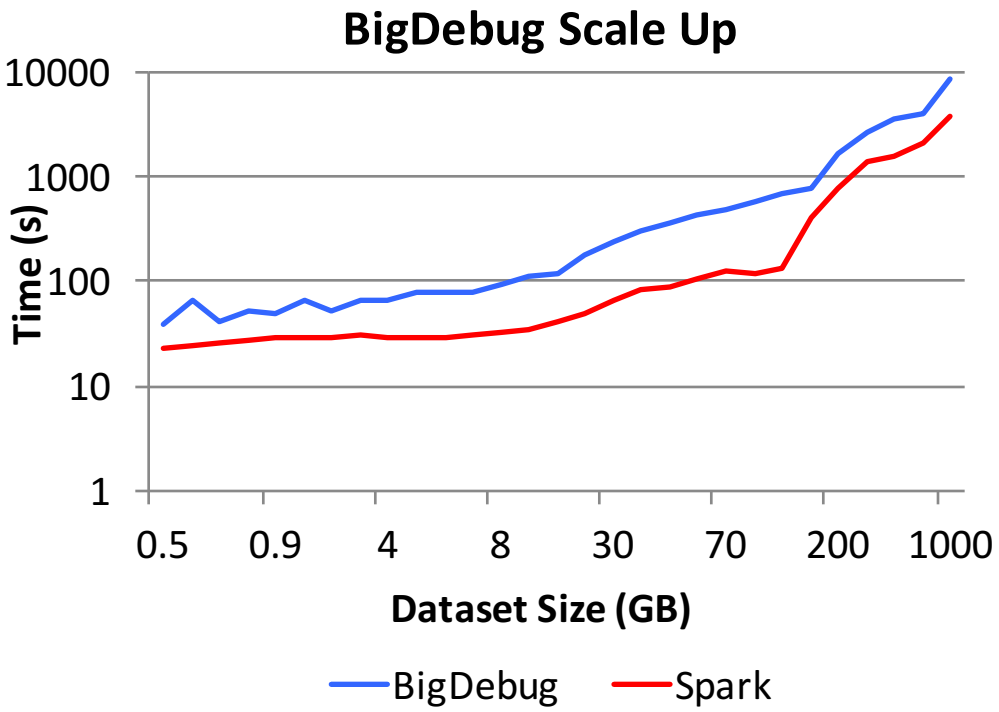
Feature 4: Forward and Backward Tracing [VLDB '15]



Data provenance enables users to identify crash inducing inputs records

```
125
126     .simultedBreakpoint()
127   val grade_age_pair = records.map(line => {
```

Performance Evaluation [ICSE '16]



With maximum instrumentation, BigDebug takes 2.4X the time of baseline Spark while the average case is at 1.34X

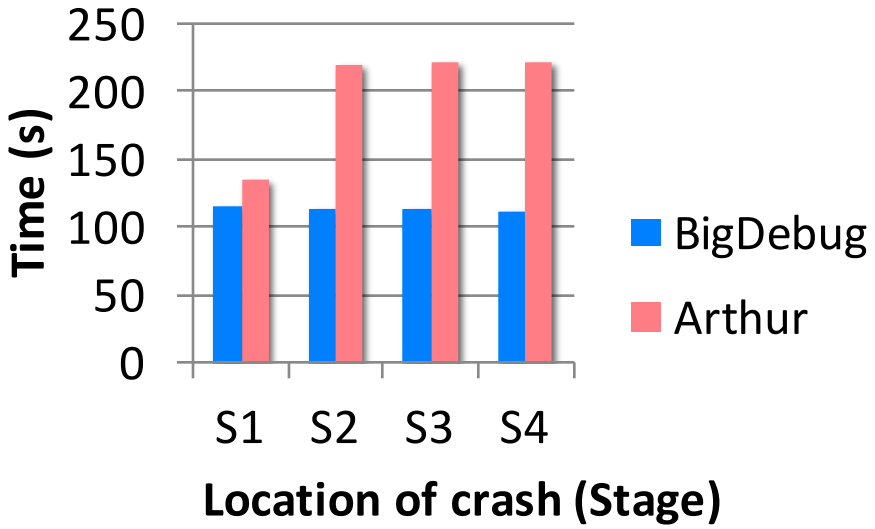
Time Saving

Arthur [Dave et al. 2013]

The first run crashes

The second run instruments all records leading to a crash

The third run removes the crash.



BigDebug

A single run can detect and remove the crash and resumes the job

BigDebug finds a crash inducing record with 100% accuracy and saves upto 100% time saving through runtime crash remediation

In Summary

- BigDebug provides primitives to enable interactive debugging on the cloud without sacrificing the performance
- Using data provenance a user can understand how errors propagate through data processing steps
- On average, BigDebug poses 34% overhead and saves 100% of time in case of crash
- BigDebug is publicly available at

https://sites.google.com/site/sparkbigdebug/