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

```scala
object AliceStudentAnalysisWP {
  val COLLEGEYEAR = List("Sophomore", "Freshman", "Junior", "Senior")
  def main(args: Array[String]): Unit = {
    // set up spark configuration
    val sparkConf = new SparkConf()
    val bdconf = new BigDebugConfiguration()
    bdconf.setFilePath("/home/ali/work/temp/git/debugdebug/spark-lineage/examples/src/main")
    // set up spark context
    val ctx = new SparkContext(sparkConf)
    ctx.setBigDebugConfiguration(bdconf)
    // spark program starts here
    val records = ctx.textFile("/home/ali/Desktop/myfile.txt", 1)
    .watchpoint(s => COLLEGEYEAR.contains(s.split(" ")(2))
    .simultedBreakpoint)
    val grade_age_pair = records.map(line => {
      val list = line.split(" ")
      (list(2), list(3).toInt)
    })
    val average_age_by_grade = grade_age_pair.groupByKey
      .map(pair => {
        val itr = pair._2.iterator
        var moving_average = 0
        var num = 1
        while (itr.hasNext) {
          moving_average = moving_average + itr.next()
          num = num + 1
        }
        (pair._1, moving_average/num)
      })
    val out = average_age_by_grade.collect()
    out.foreach(println)
  }
}
```
Feature 1: Simulated Breakpoint

val records = ctx.textFile("/home/ali/Desk: .simulatedBreakpoint()

val grade_age_pair = records.map(line => {

Feature 1: Simulated Breakpoint

val records = ctx.textFile("/home/ali/Desk...
   .simulatedBreakpoint()

val grade_age_pair = records.map(line => {

Feature 1: Simulated Breakpoint

```
val records = ctx.textFile("/home/ali/Desk:
.simulatedBreakpoint()
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

Stage 0
- map
- watchpoint
- simulated breakpoint
- map

Stage 1
- group ByKey
- map

```
val records = ctx.textFile("/home/ali/Desktop...
.watchpoint(s=> !COLLEGEYEAR.contains(s.split...)
.simulatedBreakpoint()
```
Feature 2: On Demand Guarded Watchpoint

```
1    def guard(value:
2    /**Write input types for this watchpoint
guard below. For Example: (String, Int)*/
3    ): Boolean = {
4    /**Write your guard here**/
5    }
```

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.
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 up to 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/