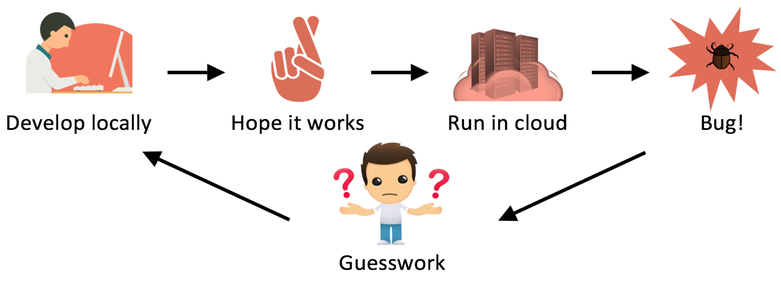
Interactive and Automated Debugging for Big Data Analytics

Muhammad Ali Gulzar

Advised by Miryung Kim

University of California, Los Angeles

Challenges in Debugging and Testing of Big Data Analytics



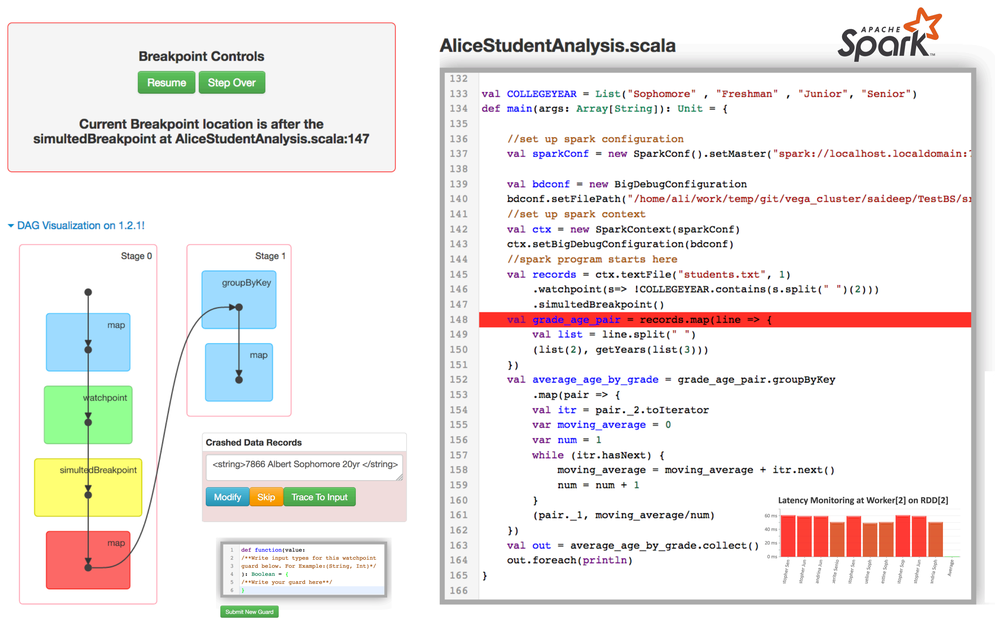
- Debugging big data processing jobs is **time consuming and error-prone**.
- Developers are notified of runtime failures or incorrect outputs after many hours of **wasted computing cycles** on the cloud.
- Finding the **root cause** of a test failure among billions of input records is almost **impossible.**
- Testing big data applications is expensive and random sampling for test data results in **in-adequate code coverage.**

BigDebug: Debugging Primitives for Interactive Big Data Processing, *ICSE '16*

Debugging Primitives

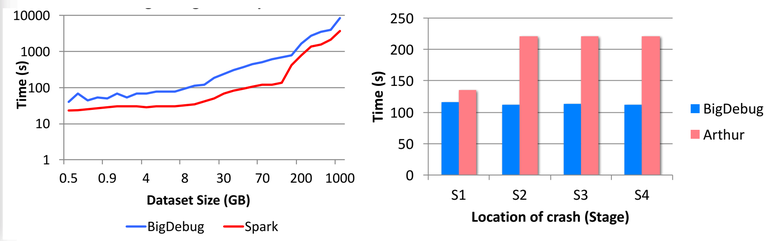
- **Simulated Breakpoint :** A user can inspect a program state in a remote node without pausing the computation.
- **On Demand Guarded Watchpoint :** BigDebug delivers filtered program states in a streaming fashion to the user, on demand.
- **Crash Culprit Remediation:** It reports crash-inducing records without terminating the job and allows user to take actions on the fly.
- **Backwards and Forward Tracing:** A user can trace crashing record back to the input data in order to isolate the root cause of a problem.
- **Fine-Grained Latency Monitoring:** It notifies the user with the records that are taking longer than usual to process.

BigDebug Interactive User Interface



Performance Evaluation

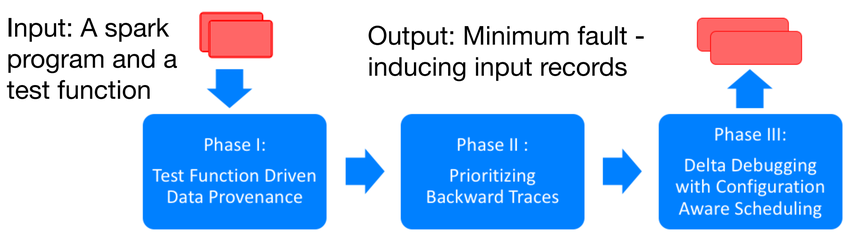
- With maximum instrumentation BigDebug, on average, takes 2.5X longer than the baseline Spark.
- When latency profiling is disabled, the overhead reduces to just **34%**, on average.
- BigDebug provides upto **100% time saving** over Spark through runtime crash remediation.



BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark. M Gulzar, M Interlandi, S Yoo, S Tetali, T Condie, T Millstein, M Kim. Proceedings of 38th IEEE/ACM International Conference on Software Engineering, pages 784-795

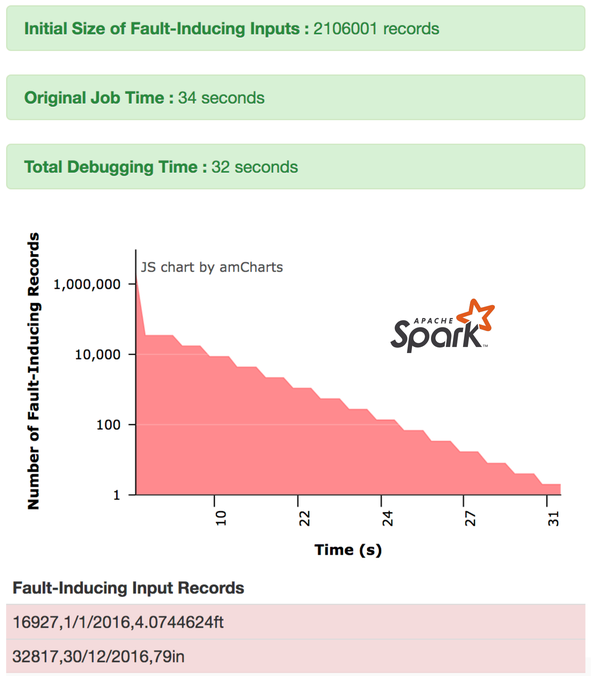
BigSift: Automated Debugging for Big Data Analytics, *SoCC '17*

- Given a test function, BigSift automatically finds a **minimum set of fault-inducing input** records responsible for a faulty output.



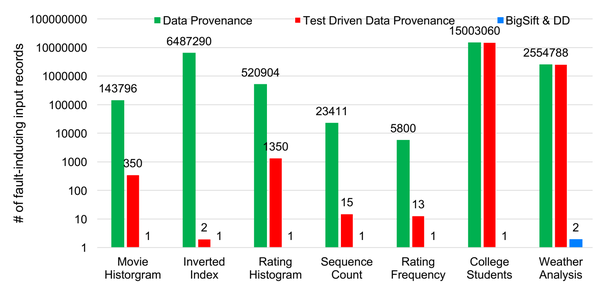
- **Optimization 1:** BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.
- **Optimization 2:** BigSift overlaps two backward traces to minimize the scope of fault-inducing input records.
- **Optimization 3:** It uses bitmap based test memoization technique to avoid redundant testing of the same input dataset.

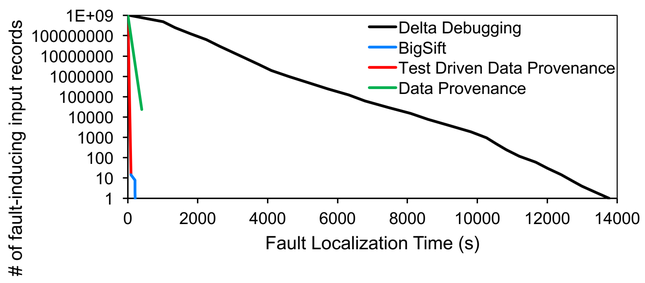
BigSift User Interface



Performance Evaluation

- On average, BigSift takes **62% less time** than the original job to debug a single fault.
- BigSift provides **7X to 66X** speed up in the debugging time compared to the baseline *Delta Debugging*.
- BigSift achieves **10^3 to 10^7X** better precision than state of the art data provenance *i.e* Titian





Automated Debugging in Data Intensive Scalable Computing. M Gulzar, M Interlandi, X Han, M Li, T Condie, M Kim. Proceedings of Symposium of Cloud Computing 2017. 15 Pages.

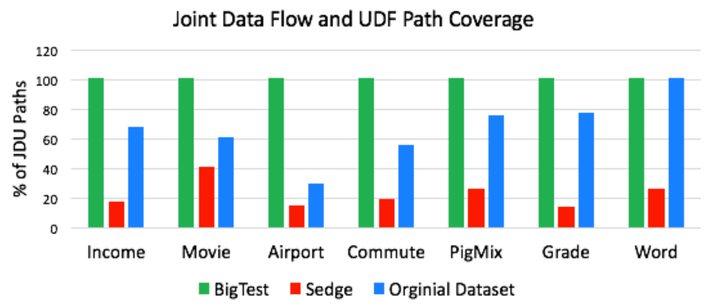
BigTest: White-box Testing of Data Intensive Scalable Computing Applications, *Ongoing*

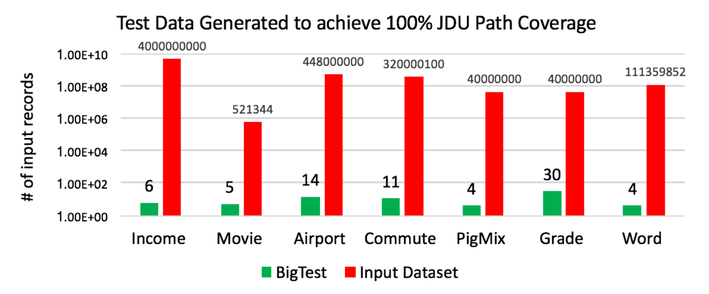
Challenges in Testing DISC Applications

- How can we select the **minimal sample** of an input dataset to perform efficient testing of DISC applications?
- How can we generate test cases that **exercise all program paths** of a DISC application to maximize code coverage?
- Due to **dataflow operators** and complex **user defined functions** in DISC application, it is extremely hard to answer the two questions.

Preliminary Results

- BigTest provides **100%** Joint Dataflow and UDF (JDU) path coverage by generating testing data which is several **orders of magnitude** (10^6 to 10^{10}) smaller than the original input dataset.



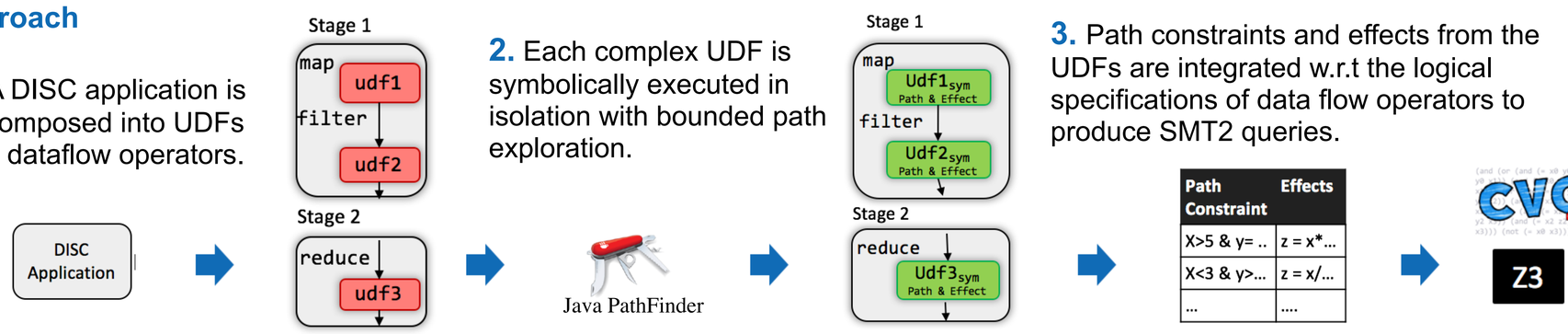


Approach

1. A DISC application is decomposed into UDFs and dataflow operators.

2. Each complex UDF is symbolically executed in isolation with bounded path exploration.

3. Path constraints and effects from the UDFs are integrated w.r.t the logical specifications of data flow operators to produce SMT2 queries.



4. Test data is generated using off-the-shelf theorem solvers such as CVC4 or Z3.