CS 5914 Project:
Silent Data Corruption in Apache Spark/HDFS Tested with PageRank

Sam Williams

Overview

In this project, I examine silent data corruption (SDC) in Apache Spark with Hadoop File System (HDFS) tested with PageRank.

Task 1: Create a fault injector for Apache Spark to simulate SDC.

Task 2: Run a basic PageRank application with **p** probability of experiencing an SDC using the fault injector and analyze correctness.

Presentation Outline

1. Background (SDC, Spark & Hadoop, PageRank)
2. Related Work
3. Project Design
4. Implementation & Results
5. Summary
6. Future Work
7. Additional Slides (Practical Issues, Discussion Q's, extra info)

Background

Silent Data Corruption

- Incorrect data generated by or calculated in hardware that is passed to the software layer
- e.g.
    - ALU incorrectly calculates 2+2 = 5
    - Bit flips
        - Cosmic radiation
        - Faulty hardware

Impacts of SDC

- Can cause faults
- … Or can go undetected
  - Big issue
  - Can render invalid results

How to identify and prevent SDC?

- Single bit flips can be detected with cyclic redundancy checks (CRCs)
    - And fixed with error correction codes (ECCs)
        - Not always implemented in caches, registers and ALUs
    - But double flips can go undetectable
- Redundancy
    - Detection requires at least dual redundancy
    - Correction requires at least triple
    - … $$$

Apache Spark/Hadoop

- Provide
  - Distributed data and application framework
  - Fault tolerance
- But still susceptible to SDC
  - In Spark RDD transformations that are distributed across nodes

PageRank

- Algorithm that determines probability user clicks on a page
- Distributable/Parallelizable application
- Interesting property of convergence
    - PageRank scores converge ~50 iterations [1]

Related Work

- RedMPI - SDC analysis and fault injector for MPI [2]
- Impacts of SDC at scale (Google, Facebook) and approaches of detection and correction [3, 4]
    - Facebook found SDC occurrence using Spark [3]
- Nothing found on fault injectors for Spark

Project Design

Design Overview

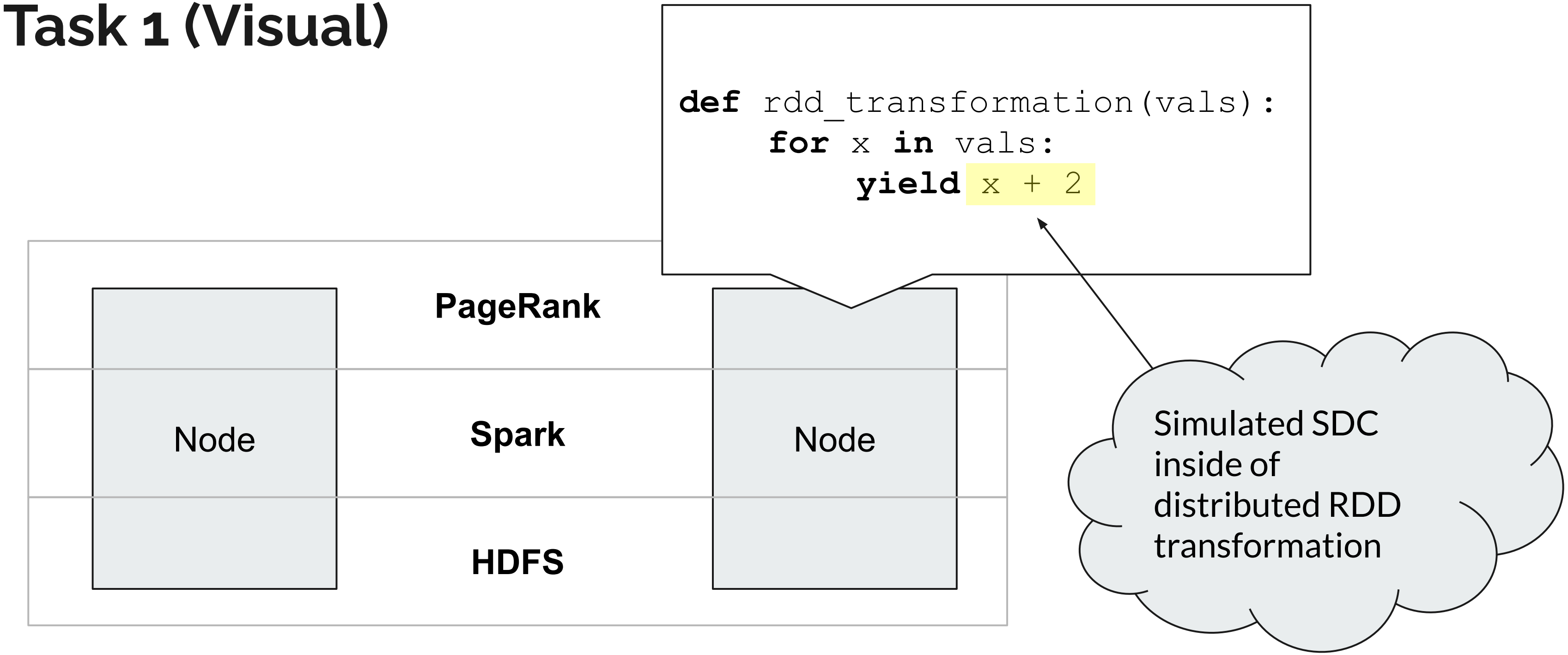
Task 1: Create a fault injector for Apache Spark applications to simulate SDC.

Task 2: Run a basic PageRank application with **p** probability of experiencing an SDC using the fault injector and analyze correctness.

Task 1 (Fault Injector)

- Define SDC as a single flipped bit in data
  - Simplification
- Performed during RDD transformation operations
  - Tasks are distributed across worker nodes
  - Uniform probability of SDC for all nodes
  - Entry point somewhere data value is calculated
- Python Spark apps (PySpark)

Task 1 (Visual)



```
def rdd_transformation(vals):
    for x in vals:
        yield x + 2
```

PageRank

Spark

HDFS

Node

Node

Simulated SDC
inside of
distributed RDD
transformation

Task 2 (Analyze PageRank with SDC)

- Use PageRank PySpark app from Spark examples
- Use parameters:
  - # PageRank iterations
  - Probability of SDC occurring
- Run trials and calculate correctness
  - Relative error
  - Average deviations

Results

Task 1 (Fault Injector)

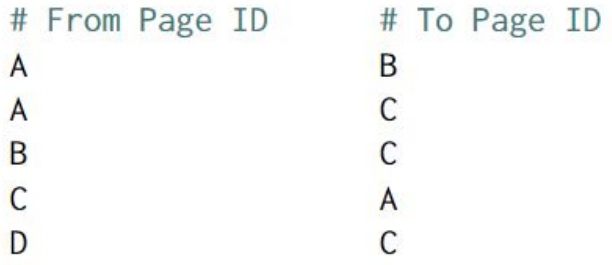
- Simple PySpark API Call
  - … `return SIMULATE_SDC( val, <probability> )`
- Works with primitives (i.e. floats for PageRank)
  - Python's ***struct*** and ***random*** modules
  - Bit arithmetic
- Bash script file pre-pends API implementation to a file

Task 2 (Analyze PageRank with SDC)

- Used with Spark's example PageRank PySpark app
  - Slightly modified with for loops...
- Independent Variables
  - # of PageRank iterations (10, 25, 50)
  - Probability SDC occurs (0.001, 0.01, 0.1)
- 50 Trials of each combination

Task 2 (PageRank Instance)

- 3 Pages
  - A, B and C
  - (Page D is omitted by PageRank implementation)



# From Page ID	# To Page ID
A	B
A	C
B	C
C	A
D	C

Results (Relative Error)

Relative Error: $$\frac{|\,average - actual\,|}{average}$$

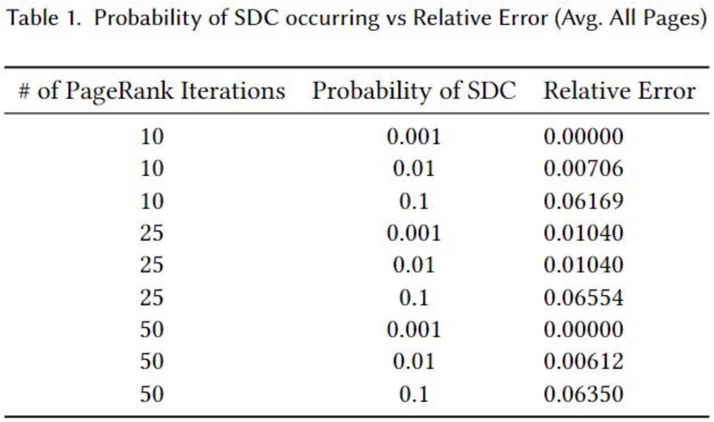
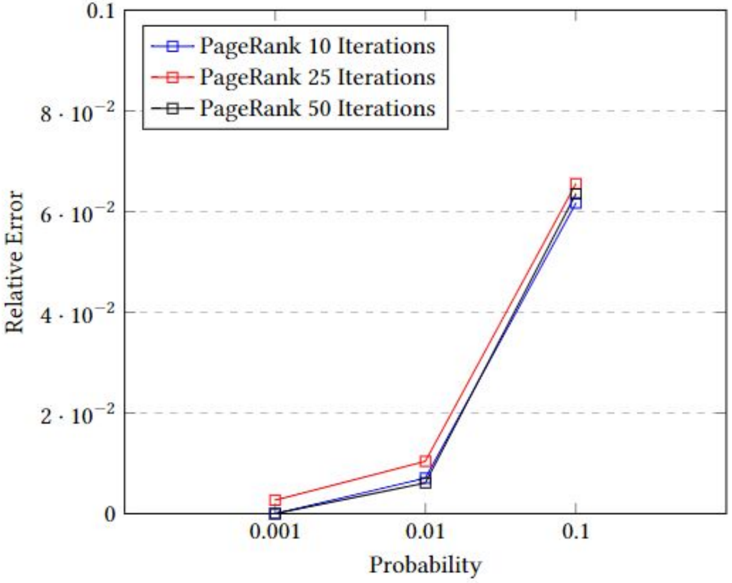


Table 1. Probability of SDC occurring vs Relative Error (Avg. All Pages)

# of PageRank Iterations	Probability of SDC	Relative Error
10	0.001	0.00000
10	0.01	0.00706
10	0.1	0.06169
25	0.001	0.01040
25	0.01	0.01040
25	0.1	0.06554
50	0.001	0.00000
50	0.01	0.00612
50	0.1	0.06350

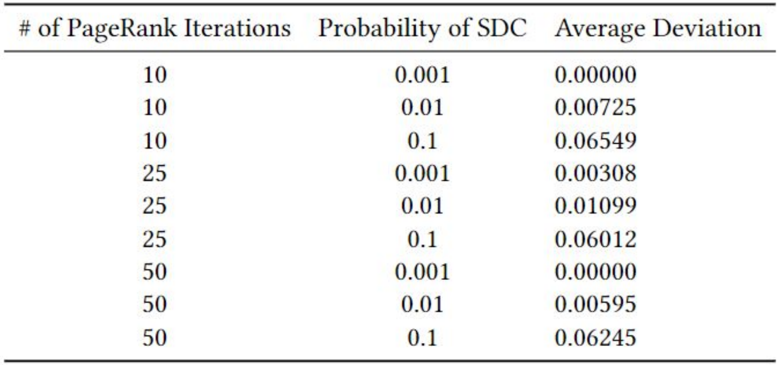
Figure 1. Probability of SDC occurring vs Relative Error (Table 1)



Results (Average Deviation)

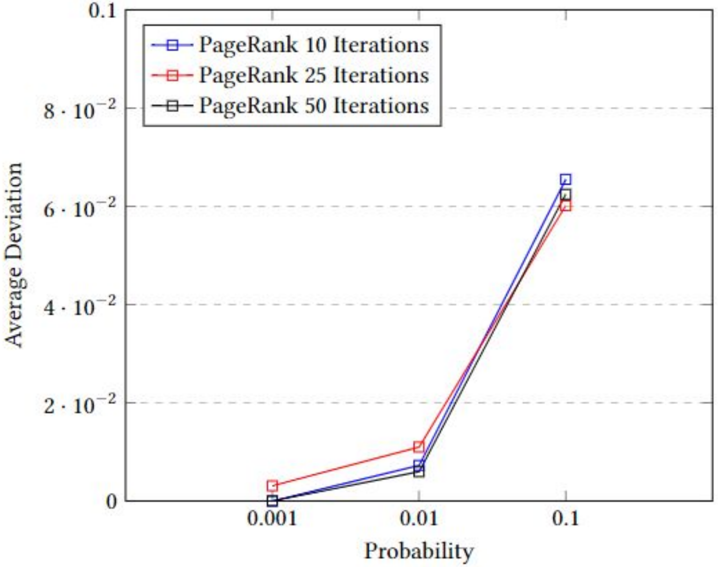
Average Deviation: $$\frac{\Sigma \mid \textit{actual - measured} \mid}{\textit{\# trials}}$$

Table 2. Probability of SDC occurring vs Average Deviation (Avg. All Pages)



# of PageRank Iterations	Probability of SDC	Average Deviation
10	0.001	0.00000
10	0.01	0.00725
10	0.1	0.06549
25	0.001	0.00308
25	0.01	0.01099
25	0.1	0.06012
50	0.001	0.00000
50	0.01	0.00595
50	0.1	0.06245

Figure 2. Probability of SDC occurring vs Average Deviation (Table 2)



Results (Takeaways)

- Relative error and average deviation are similar
- Correctness of PageRank affected by SDC is **not** improved by increasing the number of iterations
- Slightly higher relative error and average deviation experienced during 25 iterations for low probability of SDC occurring, over the 10 and 50 iteration test

Summary

- Created SDC simulator (fault injector) for PySpark apps
- Tested with PageRank PySpark app with variable number of iterations and probability of SDC occurring
    - Found correctness of PageRank affected by SDC is not improved by increasing the number of iterations
        - Though potentially interesting findings with 25 iterations over 10 and 50

Future Work Ideas

- Convert the fault injector code to an official Python Module instead of pre-pending to existing file
- Expand the fault injector framework into other Spark-supported languages like Java, Scala and R
- Explore the 25 iteration phenomenon
- PageRank input with significantly more pages should be analyzed for accuracy.
    - Parallelization

Practical Issues

- Methodology did not take advantage of parallelization
    - PageRank with 4 pages is not worth smart partitioning
- … or Spark
    - Spark Application startup/shutdown times > runtime
- Testing on VirtualBox

Discussion Questions

- Does utilizing fault tolerance via lineage lead to less SDC than redundancy?
- Are there any advanced forms of SDC prevention you’ve used/read about?
- Are there better ways of simulating SDC than my approach?

Additional info

- Relative errors and average deviations for each PageRank score were averaged for all pages
- 2 Ubuntu 22.04.1 LTS virtual machines simulating a cluster in a VirtualBox NatNetwork that were running Apache Spark 3.3.0 with Hadoop 3.3.4
- See report for implementation questions

References

1. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66. Stanford InfoLab. http://ilpubs.stanford.edu:8090/422/ Previous number = SIDL-WP-1999-0120.
2. David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. Detection and correction of silent data corruption for large-scale high-performance computing. In SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. 1–12.
3. David F. Bacon. 2022. Detection and Prevention of Silent Data Corruption in an Exabyte-scale Database System. In The 18th IEEE Workshop on Silicon Errors in Logic – System Effects.
4. Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale.