1 Background

Large-scale distributed systems are designed to be highly available, but like all human-coded systems, inevitably fall prey to bugs and experience outages. Outages in such large-scale systems however, can be very impactful, as seen for example in the 2011 Amazon Web Services outage that brought down major sites like the New York Times, Reddit, and Quora. The paper “Simple Testing Can Prevent Most Critical Failures” from the University of Toronto presents this problem in great detail, outlining the major causes of these critical failures and outlining a proof-of-concept tool to warn developers about potential sources for bugs. Paper link: https://zoo.cs.yale.edu/classes/cs426/2017/bib/yuan14simple.pdf.

It has been found that over 90% of critical failures in distributed data-intensive systems result from incorrect failure handling code. In Java code, this typically corresponds to incorrect or bad practice implementation of the catch block in try-catch blocks. Of these failures, about half of these errors consist of the following:

- An empty error handler
- The exception is too general, causing the system to abort in more cases than necessary
- The error handler contains "TODO" or "FIXME" statements
- Obviously incorrect logic in the error handling that would have been caught by simple testing

To address these types of errors researchers at the University of Toronto (see paper linked above) have developed a simple static checker called Aspirator that is
built using Chord, a program analysis framework for Java. In basic terms, Aspirator scans Java bytecode in order to identify the situations outlined above in order to alert developers about bugs related to error handling. This tool however, mainly performs basic functionality like finding and reporting empty error handlers or handlers containing a single comment. It would be interesting to extend such a tool in order to better defend against failures to make it more practical and thus more widely adopted by developers. Ideally, code-generation to actually fix simple cases of these bugs would be most beneficial.

2 Proposal

The proposed project is to extend the existing Aspirator tool to cover more cases and be more beneficial to developers. The first step in the project would be to first download the open source Aspirator repository from GitHub (https://github.com/diy1/aspirator) and install Chord as per the instructions detailed in the README found in the repository. Next, the tool would need to be run on an open-source distributed system like HDFS or Cassandra in order to replicate some of the results described in the paper and fully understand how the tool works and what can be improved. In addition to implementing the extension itself, the project would include researching open source issue trackers and determining patterns in bugs reports to further extend Aspirator. Some ideas for the direction of this extension include but are not limited to:

- Further identifying cases of \texttt{switch} fall-through and calls to \texttt{abort()} that could be generalized to languages outside of Java alone.

- More fine-grained treatment of cases of exception over-catch, which may include identifying cases of \texttt{Throwable} or overhead calls to \texttt{System.exit()}, when a more precise \texttt{Exception} should be used.

- The paper notes that there are multiple cases of false positives, and introduces a limited set of tactics to address these cases. One extension could be to test ways in which added conditions affect performance.

- Label or sort findings and warnings by some perceived level of criticality.

- Adding the functionality of Aspirator to an existing widely used static analysis tool, like FindBugs (http://findbugs.sourceforge.net/).

- The extension could include an analysis of Java source code in addition to the basic bytecode analysis performed in the original tool.
• Time and feasibility permitting, it would be ideal to be able to identify common patterns in the bugs found, and research ways to apply code-generation techniques in order to automatically fix incorrect handling code and simple bugs. Examples of this include situations where the error handler is empty, but the fix would be to insert a simple retry or method call.

3 Deliverables

Deliverables for this project would include an implementation of source code for an extension of the existing Aspirator tool. This extension would be written primarily in Java, and the code being analyzed would be in Java as well. The extension might also include a tool to analyze Java source code as well as byte code. The final product would also include a writeup of what approaches were taken to improve the existing tool, what was successful and unsuccessful, a comparison of the extended tool to the original one, as well as challenges and interesting bug patterns found along the way.