**Logjam.Next**

Log Analysis Tool for NetApp, Inc.

**Interim Progress Report**

*Initial Requirements, Design,  
Implementation, and Testing Plan*

**Team #20**

Nathaniel Brooks

Daniel Grist

Jeremy Schmidt

Wenting Zheng

North Carolina State University

Department of Computer Science

Senior Design Fall 2019

**September 13th , 2019**

# Executive Summary

*Nathaniel*

NetApp tasked our team this semester with improving a log analysis tool called Logjam. Logjam was originally designed by a former NCSU Senior Design Team after NetApp approached NC State with a problem they were facing. NetApp needed a way for their StorageGRID Tech Support employees to perform useful analysis on StorageGRID logs housed inside a large unstructured networked file server. Only a small fraction of logs stored on the networked file server were relevant to StorageGRID and there was no easy way to query the server for only StorageGRID logs. In addition, once all StorageGRID logs were found there was no easy way for a Tech Support employee to analyze the logs and gain useful information. These were the primary problems that Logjam attempted to solve: to accurately extract and index StorageGRID log files and to allow analysis of the log files.

Based on the outcomes of the previous Logjam project, NetApp approached NC State with ideas for improving it in a new iteration called Logjam.Next. The original Logjam implementation required a considerable amount of memory to track which files it had processed andNetApp wanted to process files at a faster rate. Additionally, the original Logjam implementation was difficult to install . Lastly, the original Logjam implementation had a user interface that was non-intuitive for NetApp’s Tech Support employees. Therefore our goals for this second iteration of Logjam became the following: modify Logjam so that it uses less memory when performing incremental scans, create an easier installation and execution procedure, and present log analysis through a cleaner user interface. We were given the codebase and documentation that the previous team maintained as a starting point.

We began our work by attempting to install and run the previous team’s codebase. Our difficulties with installation eventually lead to us disassembling their code and running only parts of it. We had to clarify with the NetApp team a lot of questions about the previous application because it appeared to perform operations contradictory to the given goal. After clearing up our confusion, we removed the offending lines from the previous team’s implementation and began performing correct scans over sample file structures. We also set up a Virtual Machine (VM) cluster at NCSU that closely mimicked the production environment that NetApp would use to test Logjam. Due to legal obligations between NetApp and its customers, they could not allow us to work on the full networked file server.

At this point in time we are working hard to get a full Logjam system up and running with support for both the ingestion of log files and viewing analysis through a web browser.

# Project Description

*Nathaniel*

## Sponsor Background

NetApp is a cloud data company that manages data storage and provides services to install and maintain client controlled data storage. They develop multiple hardware and software products including a product called StorageGRID, developed by Morgan Mears’s team in Research Triangle Park. StorageGRID is a distributed object storage solution that provides data storage redundancy across multiple locations with an efficient global namespace for all objects. StorageGRID systems are installed on both NetApp controlled hardware and client controlled hardware.

Morgan Mears  
Principal Engineer  
Morgan.Mears@netapp.com

Sam Fink  
StorageGRID Developer  
Samuel.Fink@netapp.com

Arpita Awasthi  
Software Developer  
Arpita.Awasthi@netapp.com

## Problem Statement

Each of NetApp’s StorageGRID systems stores a large collection of operational log files. Whenever there is a problem with StorageGRID systems, clients contact NetApp Tech Support and send them an archive of log files. Normally, only log files pertinent to the problem are sent to to NetApp Tech Support . Tech Support then takes these bundles and uploads them to a interal Network File System (NFS) repository. Over the years this NFS repo has grown to over 1000 Terabytes, with only a small fraction relevant to StorageGRID. NetApp needs a way for Tech Support to effectively utilize the StorageGRID log data stored on that NFS repo.

In 2018, a NCSU Senior Design team began formulating a solution for this problem that they called Logjam. Their solution was designed to scan the NFS repo for StorageGRID logs, index those logs inside ElasticSearch, and provide analysis through a web interface served by Kibana. However, parts of their program did not work as intended and NetApp saw areas for improvement. Logjam was able to scan an entire directory of files, but it had no way to safely stop a scan midway. It kept track of which log files it had previously scanned from the inspection directory, but at a large memory cost inside an SQL database. The web interface was able to search log files, but it did not do so in a clear way and it did not provide adequate tools to analyze log files. Getting Logjam booted up and running involved a long and error-prone process.

## Project Goals & Benefits

NetApp wanted Logjam to incrementally scan the inspection directory with a lower memory footprint that did not rely on an SQL database to store scanned paths. Doing so would allow Logjam to run faster and use less resources while running. They also desired for Logjam to have a more user-friendly installation process that was automated as much as possible. This would help their Tech Support teams get Logjam started without exerting unnecessary work. Finally, they wanted Logjam to have a cleaner and more useful web interface, with diagrams that showed comparison statistics to existing logs. This improvement would also assist their Tech Support teams in quickly identifying the severity and cause of StorageGRID error logs.

## Development Methodology

Our team worked on Logjam in two week iterations with rotating leaders and note-takers. Everyone fulfilled the role of leader and note-taker during the course of the project. During the iterations we would meet with our sponsors every Tuesday from 10:30 - 11:30 to show our progress, ask clarifying questions about the project, and agree on goals for the next meeting. We would plan out the meetings days before, and use that time to decide on specific goals we think are possible to achieve by the next meeting.

We used a DevOps strategy for development, whereby we planned to deliver small sets of functional features over our many iterations. Each iteration’s features were accompanied by the requirements, design, and possible testing of those features. Once we got something running, we tried to always keep it running without regression of any features. We wanted to have a deliverable at each meeting that showcased some of the new features for the iteration.

## Challenges & Resolutions

One of our primary technical challenges was our collective inexperience with the tools used in Logjam. None of us had worked with the Elastic Stack before and only one of our members had experience with Docker. We had to learn these tools while working on the project.

Working with Logjam required a machine with considerable disk storage and RAM. Initially we attempted to use Virtual Machines (VM) and dual Linux boots on our personal laptops, but it proved to be insufficient for the needs of Logjam. We had to request a VM from NCSU with storage space for 400GB.

Logjam was designed to use 3rd party libraries to unzip various compressed file formats such as .tar, .zip, and .7z. As with any unknown 3rd party library, this can present legal concerns if the library performs something nefarious with the data it is given. We carefully reviewed each of the 3rd party libraries that the previous team chose to use and evaluated the reliability of each. All of the 3rd party libraries we reviewed were open source.

Due to the privacy concerns of NetApp’s client data, we were not able to work directly with the NFS repo. We could only work with smaller sanitized data sets that the NetApp team provided. This was a legal problem that did not affect us directly while working, but it indirectly influenced how we tested our program. It forced us to rely on the NetApp team to test our program on the full NFS repo and report back any errors that it encountered.

# 

# Requirements

*Wenting*

## Functional Requirements

FR1: heuristic/separating

* Logjam.Next shall recurse through a root directory and apply a heuristic to find StorageGRID log files. It shall unzip compressed directories and recursively search them.
  + Assumptions
    - The root directory will not change structure drastically
    - Top level directories will include a unique case number
    - New files may appear in previously scanned directories
    - Most files should be log type

FR2: indexing

* Upon detecting a StorageGRID log file, Logjam.Next shall parse the file and index it. All other log files shall be ignored. It shall store the case number with each indexed log file.
  + Assumptions
    - The case number in the pointed directory is unique (data from America and Europe is put in different directories)

FR3: anomaly detection

* Logjam.Next Web Interface shall allow StorageGrid support technician to view a risk assessment for inputting log indicating how abnormal the log is.
  + **User Story**
    - As a StorageGrid support technician, I want to visualize how frequent the customer log is in the filtered data so that I can identify if a customer’s log data is abnormal. I will paste the log and use three filters (release level, platform type, and encoding type) to show how frequently this log appears in the system.
  + Assumptions
    - The input data should not be identical to any previously ingested by the system

## Non-Functional Requirements

NFR1: performance

Logjam.Next should ingest and process files at an absolute minimum rate of 1GB log files per hour.

NFR2: security

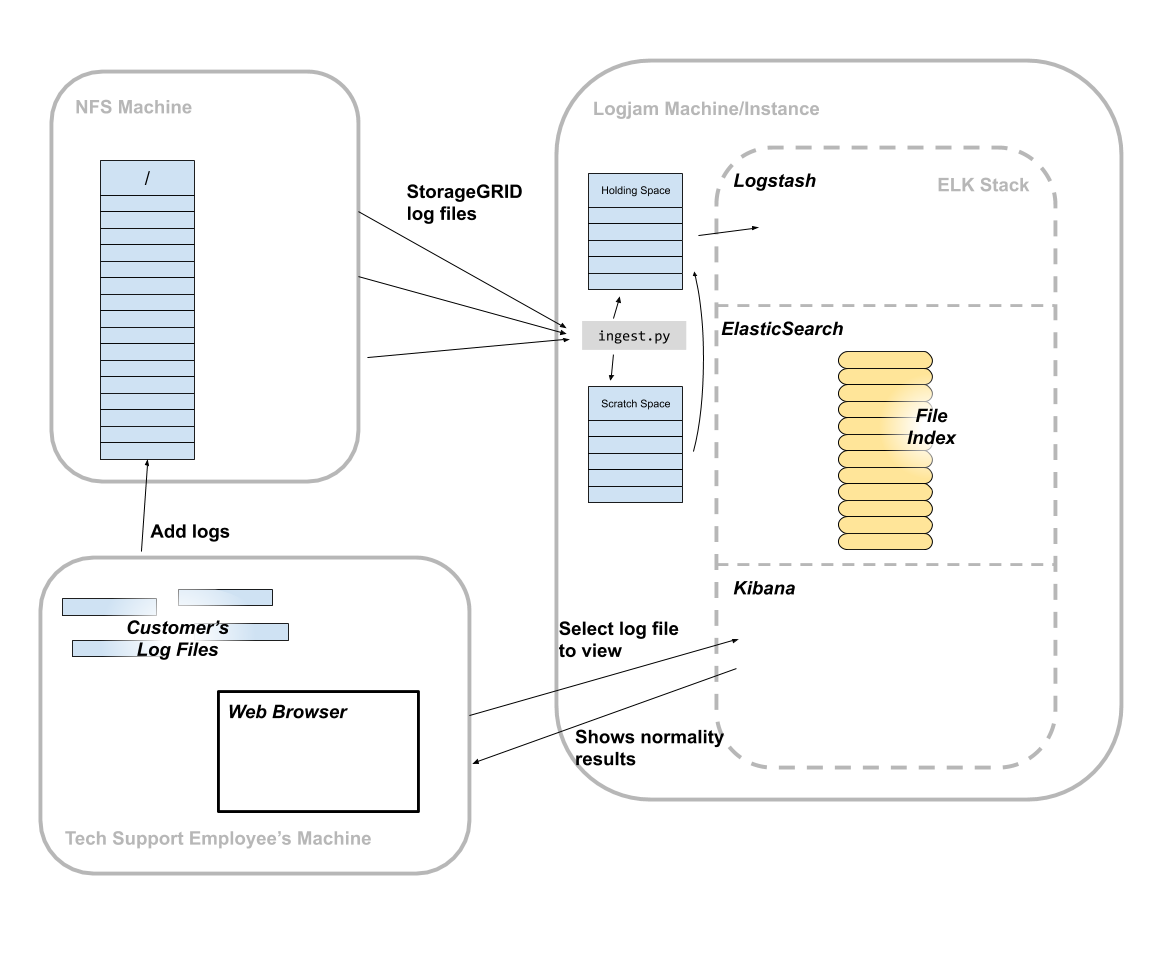
Logjam.Next should treat the entire source repository as read only. The temporary data is stored in scratch space.

## Constraints

* Use Elastic Stack (LogStash for ingesting, ElasticSearch for indexing, Kibana for visualization)
* Logjam.Next should be able to install and run on Ubuntu LTS 18.04 with minimal, reproducible setup
* NCSU GitHub repository will be used to host code and track code changes

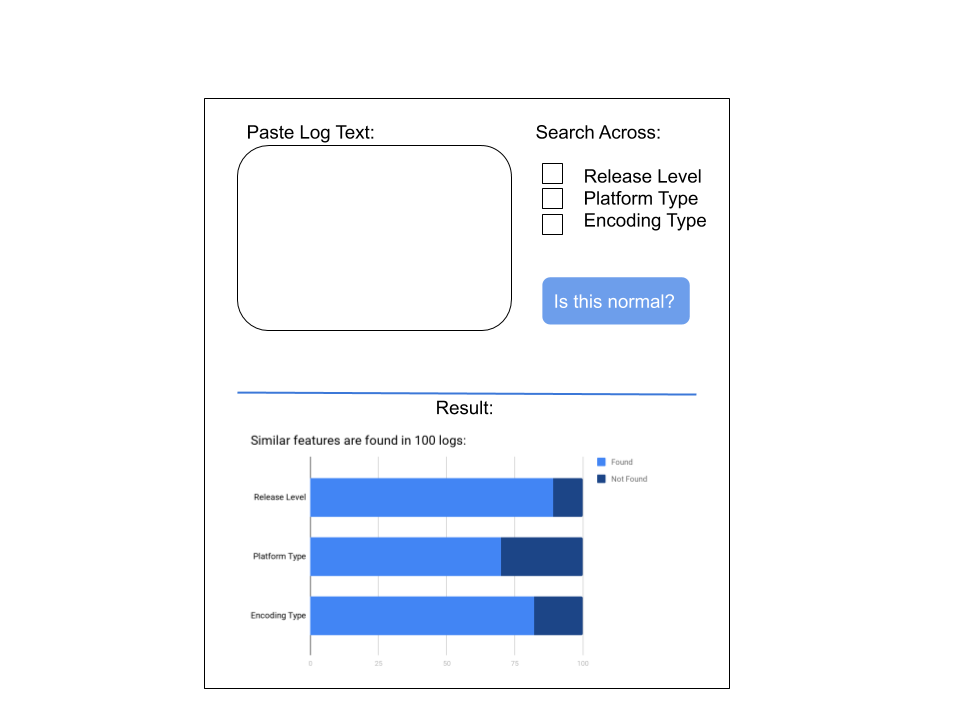
# Design

*Daniel*



**Figure 1: High Level Diagram**

Logjam.Next is divided into three main components: the user, an NFS (Network File System) repository, and a logjam instance (see Figure 1). The user is in charge of adding customer’s log files to the NFS repository to be stored. The NFS repository is a storage unit that allows a system to share files with others through a network. Its main functionality is to store files that can be accessed by anyone in the company. Lastly, the logjam instance is where our application runs. It contains a Python script, ingest.py, that with the help of two scratch spaces, it gets files from the NFS repository that belong to StorageGRID and hands them off to the ELK stack if they have been modified since the last check. The ELK stack, or Elastic Stack, is an open source suite that is used for log ingestion and analysis. It has three main components: Logstash, Elasticsearch, and Kibana. Logstash is in charge of parsing and structuring raw log data. Elasticsearch indexes the log data in a way that is easily accessible, and Kibana provides a web interface for querying and visualizing the data from the log files. Once the recently modified StorageGRID log files reach the ELK stack, they will go through Logstash, where key data is extracted before it is sent to ElasticSearch. The data gets indexed during the ElasticSearch phase and it becomes available for Kibana to use and query data that will be displayed for simple examination.



**Figure 2: UI Diagram**

As seen in Figure 2, the graphical interface will be composed of four different parts. Support engineers will be able to paste log data in the big box at the top. The second part is a selection of checkboxes where the support engineer can choose where to search across for specific logs and what to filter by. The “Is this Normal?” button is there to send a query and search for similar logs to the one pasted in the textbox with the specified fields. The last component of the graphical interface is the results area, which appears once the normalcy button is pressed. The result area shows a graphical display of how often the log appears in the database, and how it compares to those similar logs.

# Testing

*Jeremy*

## Testing Status

The previous team provided a set of four functional tests which ran scenarios on specific input data and asserted certain outputs from the ingest.py script. In our efforts to run the tests provided, we found them very finicky and hard to reproduce. Because of this difficulty and because we anticipate some divergence in our design, we do not plan to keep any of these tests and will instead move forward with our own set of unit tests and acceptance tests.

At this time, no new tests have run against our project.

## Testing Strategy

Our test plan will consist of white-box unit tests and black-box acceptance tests. The aim for the unit tests will be to test the lower level operations of our program without using external resources or requiring complicated setup. Our acceptance tests will be more involved and involve using the system in as close to a simulated end-user scenario as possible. Acceptance tests may be automated if time permits.

The nature of our project presents certain testing challenges. Because our input consists of extremely large quantities of relatively unstructured data, it is important that our test input data covers appropriate equivalence cases rather than trying to test for *any* possible input.

Our unit tests will be written in Python using the built-in UnitTest library. This framework provides a robust testing setup with a small amount of boilerplate code. To avoid contaminating project directories, tests which run file operations will use the pyfakefs library to mock a file system in memory. We will use Coverage.py to determine unit test code coverage.

## Unit Tests

We anticipate achieving unit test coverage of at least 85% for code portions of our project.

Example unit tests:

* Input archives of each of the types: zip, gz, tar, 7z; verify they are decompressed without issues
* Input a double-compressed archive (possibly with mixed compression types); verify files are extracted from all levels
* Input a corrupt archive file to decompress; verify the program can handle it gracefully without crashing

## Acceptance Tests

### Required inputs:

These acceptance tests will run using the sample input data provided by Morgan Mears located in Google Drive. The folder “01” from the data set will be referred to in the test plan as “sample-01”.

### Test Setup

1. Create a Python virtual environment for the project:

python3 -m venv ./venv

source ./venv/bin/activate

1. Install Python dependencies:

pip install -r requirements.txt

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| ingest1 | Ingest the dataset sample-01 into logjam by running `ingest.py <path-to-data> -v` | Full dataset is processed without errors and data is available for query in elastic search (Test query TBD) |  |
| ingest2 | **Preconditions:**   * Elasticsearch index is empty * All logs previously input into logjam have been deleted   **Steps:**   1. Start ingesting the data set sample-01 as described in ingest1, 2. After a few seconds, interrupt the program by pressing ctrl-C 3. Start the ingest script again in the same way. | Logjam should pick up where it left off as indicated by the program's log output |  |
| analyze1 | **Preconditions:**   * Sample-01 data set has previously been ingested   **Steps:**  Paste a section of log into the UI which does have similarities to existing logs (exact input data TBD) | logjam reports logs with similar features. (“not abnormal”) (Exact report info TBD) |  |
| analyze2 | **Preconditions:**   * Sample-01 data set has previously been ingested   **Steps:**  Input a section of log with an error type which is *unlike* those previously ingested (exact input data TBD) | Logjam indicates this input *is* abnormal (exact report info TBD) |  |

Discussion on test result:

# 

# Task Plan

*Wenting*

|  |  |  |  |
| --- | --- | --- | --- |
| **Team 20 Task Plan** | | | |
| **Item** | **Owner(s)** | **Due Date** | **Status** |
| Draft requirement   * Learn ELK Stack * Setup Github repository * Investigate existing installation process * Get example data from Morgan | All | 9/3/2019 | Complete |
|
| Develop a simple prototype   * Update ingestion script * Update Python version and contact IT Support for VM * Research on how log files are input to elastic search * Research on modification time | All | 9/10/2019 | Complete |
| Jeremy, Nathaniel |
| Jeremy |
| Daniel |
| Wenting |
| Draft IPR   * Executive Summary and Project Description * Design * Testing * Requirements and Task Plan | All | 9/13/2019 | Complete |
| Nathaniel |
| Daniel |
| Jeremy |
| Wenting |
| Draft OPR Slides | All | 9/13/2019 | Complete |
| Technical Exploration of ELK Stack | All | 9/17/2019 | In Progress |
| Low-Level Design & Assign Tasks | All | 9/19/2019 |  |
| Iteration 1 Initial Implementation:   * Walk through directory and ingest log data * Index data so that it’s easily searchable * Test manually   IPR feedback & sponsor updates |  | 9/24/2019 |  |
| Design/Code Review | All | 9/26/2019 |  |
| Iteration 1 Implementation & Simple Demo | All | 10/1/2019 |  |
| Final IPR & Completed Iteration 1 | All | 10/4/2019 |  |
| OPR 2 & Demo |  | 10/8/2019 |  |