[[1]](#footnote-2)

Leveraging Cloud Services for Large Scale Fuzzy Hashing(July 2022)

Adam Alidra, *MSDS Candidate*, Leonardo Leal Filho., *MSDS Candidate*, and Josh Mitchell., *MSDS Candidate*

*Abstract*— **Our project proposal is centered on optimizing fuzzy hashing for use by enterprises at scale, by way of leveraging hybrid-cloud resources. We will start by describing common fuzzy hashing algorithms and how they work to find files with similarities to other documents. We will then discuss the inherent scale issues of using fuzzy hashing techniques and explore ways to address these issues using SQL DML and clustering techniques. First, we will demonstrate this process using ‘****on-premise’ resources, in other words, using MySQL on our local machines. Then, we will perform this process using cloud services, namely, Azure Database for MySQL, demonstrating inherent cloud benefits, including scalability, elasticity, high availability, security, and a cost-optimized** **approach to scaling compute and storage for enteprises.**

# INTRODUCTION

I

dentifying computer file anomalies (or similarities) and the potential security risks they present is a ubiquitous challenge for enterprises. Moreover, with the proliferation of applications and the data they generate, in both, varying structures and types, the problem has become increasingly complex and costly to manage at scale.

For this project, we are using the hashproject database. This is a default database we programmatically developed using Python in Visual Studio Code packaged with MySQL. Contained within the database is our data model designed to support ssDeep fuzzy hashing, a similarity digest algorithm used to determine similarities between hashes that represent compared files. The relations embedded within the data model include:

* Fuzzy\_hash: hash\_id, ssdeep\_hash
* Ssdeep\_chunk: chunk\_id, chunk\_size, chunk
* Crypto\_hash\_table: idcrypto, md5\_hash, sha1\_hash, sha256\_hash

# Background

ssDeep and other fuzzy hashing algorithms aim to identify (or delineate) between similar (or dissimilar, respectively) files using derived hashes to compare against. These algorithms produce scoring ranges that tell us how close or different a file is from one another, zero signifying no or negligible similarity to 100, demonstrating almost or even an exact match. ssDeep computes a fuzzy hash (typically referred to as ‘compute’) based on each bit of data inputted, which can range from strings to files to videos and more. Once we have ‘compute’ for the inputs (files) we can run a comparison method to compare the hashes- “…this allows for simple high-level comparisons without the need to compare each file byte by byte,” (Wallace, 2015).

ssDeep, at scale, however, comes with challenges- the ssDeep compare function needs to be called for each hash being tested against, which presents issues when the hashes are queried collectively from a database as part of a lookup. This becomes even more technically challenging and resource intensive when an ssDeep hash needs to be compared against all other hashes.

# Approach

Our team focused on testing the optimization method we executed enabling ssDeep to scale. This testing, which involves comparing ssDeep hashes at scale, requires a reductive technique. It effectively decreases the search vector of similar hashes- this technique is called the ‘Lookup’ task or ‘Matching Mode’.

## Chunksize

The first step in the optimization of ssDeep hashes, which lends to its practical application of fuzzy hashing, is creating ssDeep hashes or a ‘chunksize’. This is an integer which is a programmatic representation of the underlying file it corresponds with. The characters within the chunk each correspond with a part of the original file.

**Hash 1:**

Chunk:

set([['v7XINhX', '7XINhXz', 'XINhXzn', 'INhXznV', 'NhXznVJ', 'hXznVJ8', 'XznVJ8C', 'znVJ8CC', 'nVJ8CC1',

'VJ8CC1r', 'J8CC1rB', '8CC1rBX', 'CC1rBXd', 'C1rBXdo', '1rBXdo0', 'rBXdo0z', 'BXdo0ze', 'Xdo0zek',

'do0zekX', 'o0zekXU', '0zekXUd', 'zekXUd3', 'ekXUd3C', 'kXUd3Cd', 'XUd3CdP', 'Ud3CdPJ', 'd3CdPJx',

'3CdPJxB', 'CdPJxB7', 'dPJxB7m', 'PJxB7mN', 'JxB7mNm', 'xB7mNmD', 'B7mNmDZ', '7mNmDZk', 'mNmDZkU',

'NmDZkUK', 'mDZkUKM', 'DZkUKMK', 'ZkUKMKZ', 'kUKMKZQ', 'UKMKZQb', 'KMKZQbF', 'MKZQbFT', 'KZQbFTi',

'ZQbFTiK', 'QbFTiKK', 'bFTiKKA', 'FTiKKAZ', 'TiKKAZT', 'iKKAZTy']])

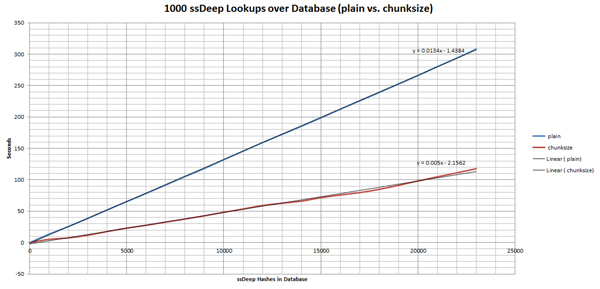
We then took the chunksize but computed it by a factor of 2 i.e. doublechunk= chunksize\*2 to enable ssDeep hashes computed with adjacent chunksizes to be compared. Doing so enabled us to perform comparisons of chunksizes such that no other value than zero was returned unless “the chunksize of the other hash is n/2, n, or 2\*n,” (Wallace, 2015).

Double chunk:

set(['ShT8C+f', 'hT8C+fu', 'T8C+fui', '8C+fuio', 'C+fuioH', '+fuioHq', 'fuioHq1', 'uioHq1K',

'ioHq1KE', 'oHq1KEF', 'Hq1KEFo', 'q1KEFoA', '1KEFoAU'])

Effectively, this was the first optimization in scenarios where comparisons were being made on vast volumes of unique datasets. A benchmark depiction of the unoptimized versus optimized result:



# Schema Creation

We created a MySQL database on our local machine and subsequent schema. This was an important step in our process as we needed to store our ssDeep hashes, including their chunksizes for each corresponding file in our analysis. This would eventually allow us to both, compute ssDeep comparisons and query against the database to find anomalies in our search. Although we took a more programmatic approach using functions to deploy our database, we are including a simple example of the code which can be used to achieve the same result:

Text

Description automatically generated

# Querying

Now that a database and schema was created, for each created hash it was uploaded to the database. The ssDeep hashes are represented in the ‘fuzzy\_hash\_table’ as the primary key, ‘hash\_id.’ These are then tokenized and entered into the chunk table, ‘ssdeep\_chunk\_table.’ As you can see below, the foreign key, ‘fuzzy\_hash\_table\_hash\_id’ has a relationship to the ssDeep hashes table.

A screenshot of a computer

Description automatically generated with medium confidence

When you execute the hash\_and\_upload.py script you effectively generate cryptographic hashes which are used to uniquely identify a file- these are stored in the ‘crypto\_hash\_table’. When we want to query the database, we use the db\_query.py script and provide the ssDeep full hash. After connecting to the database, we execute several queries to ascertain possible relationships with the inputted ssDeep hash. Next, we utilize ‘Matching Mode’ on the results to determine the hashes that best match the input hash. Matching mode is powerful in that it takes the hashes from input files and compares them against all known hashes in the database. Finally, we reuse the query to obtain the cryptographic hashes associated with the ssDeep hash corresponding to the file in question. In the example below, you will notice a score based on a comparison between Hash 1 and Hash 2 results in a value greater than zero. Also, the comparisons between Hash 1 and Hash 3, and between Hash 2 and Hash 3, demonstrate no difference in comparisons.

>>> ssdeep.compare("768:v7XINhXznVJ8CC1rBXdo0zekXUd3Cd

PJxB7mNmDZkUKMKZQbFTiKKAZTy:ShT8C+fuioHq1KEFoAU", "768:C7XINhXznVJ8CC1rBXdo0zekXUd3CdPJxB7mNmDZkUKMKZQbFTiKKA

ZTV6:ThT8C+fuioHq1KEFoAj6")

97

>>> ssdeep.compare("768:v7XINhXznVJ8CC1rBXdo0zekXUd3Cd

PJxB7mNmDZkUKMKZQbFTiKKAZTy:ShT8C+fuioHq1KEFoAU", "768:t2m3D9SlK1TVYatO/tkqzWQDG/ssC7XkZDzYYFTdqiP1msdT1OhN7UmSaED7Etnc:w7atyfzWgGEXszY

YF4iosdTE1zz2+Ze")

0

>>> ssdeep.compare("768:C7XINhXznVJ8CC1rBXdo0zekXUd3Cd

PJxB7mNmDZkUKMKZQbFTiKKAZTV6:ThT8C+fuioHq1KEFoAj6",

"768:t2m3D9SlK1TVYatO/tkqzWQDG/ssC7XkZDzYYFTdqiP1msdT1

OhN7UmSaED7Etnc:w7atyfzWgGEXszYYF4iosdTE1zz2+Ze")

0

Other fuzzy Hashing Techniques

TLSH, otherwise known as Trend Micro Locality Sensitive Hash, is another fuzzy matching library. Like ssDeep, TLSH generates a hash value which can be used for similarity comparisons. Our approach, outlined above herein, can be applied towards TLSH and other fuzzy hashing methods.

ssDeep in the cloud

For purposes of planning for scale, we looked to cloud resources. We opted to leverage Azure Database for MySQL, a Platform as a Service offering. In addition to the ability to scale up and down the resource at will, thereby further optimizing the costs associated with the service, we, as users also get the added benefit of increased security, high availability, backup and restore, monitoring and alerting, and replication. It is also important to note the flexibility in which developers, data scientists, and IT professionals can deploy, secure, and/or interact with resources- with your option of programming languages in the console, including Azure CLI, PowerShell, and Bash, the ability to run your code from Visual Studio and other developer tools, and varying methods to connect to your locally hosted resources, enterprises benefit from the added benefit of leveraging their existing skill sets to support their organization, including in the instance of the ssDeep use case, mitigating threats and risks.

Graphical user interface, text, application, email

Description automatically generated

*MySQL in Azure Portal illustration*

References

1. B. Wallace, “Optimizing ssdeep for use at scale,” Virus Bulletin :: Optimizing ssDeep for use at scale, 27-Nov-2015. [Online]. Available: <https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale#:~:text=ssDeep%20%5B1%5D%20is%20a%20fuzzy,file%20are%20considered%20highly%20similar>. [Accessed: 26-Jul-2022].
2. J. Kornblum, H. Grohne, and T. OI, “ssdeep - Fuzzy hashing program,” *ssdeep Project*, 11-Apr-2018. [Online]. Available: https://ssdeep-project.github.io/ssdeep/#docs. [Accessed: 26-Jul-2022].
3. B. Wallace, “Bwall/SSDC: Ssdeep based Clustering Tool,” *GitHub*, 17-Aug-2016. [Online]. Available: https://github.com/bwall/ssdc. [Accessed: 26-Jul-2022].
4. “Python wrapper for ssdeep fuzzy hashing library,” *GitHub*, 21-Dec-2021. [Online]. Available: https://github.com/DinoTools/python-ssdeep. [Accessed: 26-Jul-2022].
5. “Azure database for mysql - managed mysql database: Microsoft azure,” *Managed MySQL Database | Microsoft Azure*, 2022. [Online]. Available: https://azure.microsoft.com/en-us/services/mysql/#overview. [Accessed: 26-Jul-2022].
6. *Trendmicro/tlsh*, 22-Sep-2021. [Online]. Available: https://github.com/trendmicro/tlsh. [Accessed: 26-Jul-2022].
7. “A locality sensitive hash,” *TLSH*, 26-Nov-2021. [Online]. Available: https://tlsh.org/. [Accessed: 26-Jul-2022].
8. J. Kornblum and T. OI, “Getting Started with ssdeep,” *ssdeep Project*, 25-Sep-2017. [Online]. Available: https://ssdeep-project.github.io/ssdeep/usage.html. [Accessed: 26-Jul-2022].
9. D. R. White, “Ssdeep datasets,” *Software Quality Group*, 15-Nov-2019. [Online]. Available: https://www.nist.gov/itl/ssd/software-quality-group/ssdeep-datasets. [Accessed: 26-Jul-2022].
10. J. Abrahamy, “Intezer community tip: How to optimize ssdeep comparisons with Elasticsearch,” *Intezer*, 19-Sep-2017. [Online]. Available: https://www.intezer.com/blog/malware-analysis/intezer-community-tip-ssdeep-comparisons-with-elasticsearch/. [Accessed: 26-Jul-2022].
11. A. Alidra, L. Leal, and J. Mitchell, “dbfuzzhash,” Fuzzy Hash search project using databases, 27-7-2022. [Online]. Available: https://github.com/adama323/dbfuzzhash

1. This work was submitted on July 26, 2022. Adam Alidra, Leonardo Leal Filho, and Josh Mitchell are Master of Science Data Science Candidates at the Southern Methodist University, as of the submitted date of this paper. [↑](#footnote-ref-2)