A Report of the Implementation of AdHASH, An Incremental Hash Algorithm

Ching-Chan Lee
Computer Science Department
San José State University
San José, CA 95192
408-924-1000
ching-chan.lee@sjsu.edu

ABSTRACT

An incremental hash algorithm was first proposed by Bellare et al. in 1997, and the emergence of cloud computing and distributed computing usually uses hash codes to check the data consistency. A fast and robust hash algorithm for checking the consistency between files is desired. The incremental hash algorithm seems to be a perfect candidate when the files are modified in a relatively small portion compared to the previous version of files. Recent applications of the incremental hash algorithm are mentioned by Zhao et al. in 2017.

1. INTRODUCTION

Hash codes are often used for checking the consistency of the data and the signature of a file, to name a few. In [2], the application of the incremental hash algorithm has its advantage in a certain context.

However, people are looking for a hash algorithm that satisfies a few properties where one-way and collision-free are the popular properties. The balance between performance and security is the key when comparing a traditional hash algorithm and security concerns.

This report summarizes the implementation of AdHash, an incremental hash algorithm proposed in [1]. The report details explain the paradigm of the incremental hash algorithm and a brief experiment showing the differences in the performance between the traditional approach and the incremental approach.

2. MOTIVATION

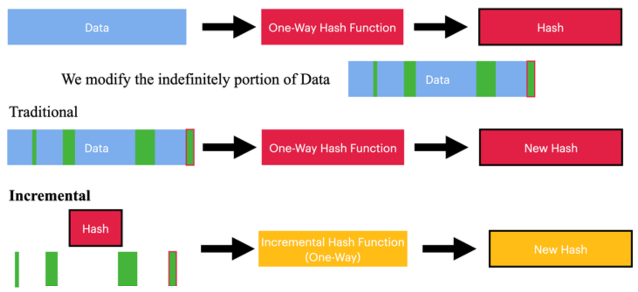


Figure 1. Concepts of the tradition hash algorithm and incremental hash algorithm

The traditional one-way hash function takes the whole inputted data to compute the hash code. However, when the file is modified in a relatively small portion, re-computing the hash code by inputting the whole file is inefficient. The ideas of the incrementality proposed by Bellare et al. [1] are computing the new hash code corresponding to the latest modified file by updating the hash code with the changes of the data. The update is either appending new hash codes or removing the hash codes corresponding to the modified blocks and appending the new hash codes.

Figure1. illustrates the aforementioned concept where the green rectangles represent the changed blocked of data. With the incremental approach, the new hash code can be generated faster and also maintains a collision-free property as proposed by [1].

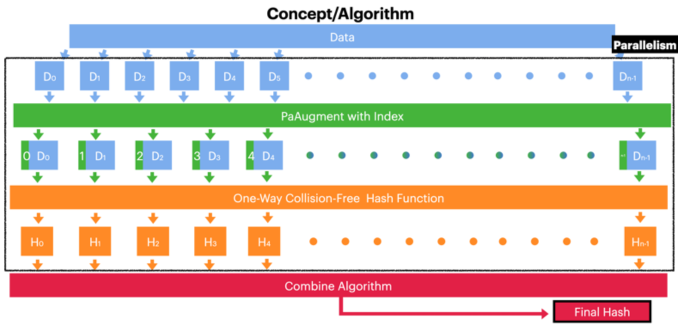

Concept/Algorithm
Data
Parallelism
D_0 D_1 D_2 D_3 D_4 D_5 D_{n-1}
PaAugment with Index
0 D_0 1 D_1 2 D_2 3 D_3 4 D_4 D_{n-1}
One-Way Collision-Free Hash Function
H_0 H_1 H_2 H_3 H_4 H_{n-1}
Combine Algorithm
Final Hash

Figure 2. Paradigm of an incremental hash algorithm

3. INCREMENTAL HASH ALGORITHM

Figure 2 shows the paradigm of an incremental hash algorithm. The paradigm also shows the instruction to implement an incremental hash algorithm. Additionally, we can obviously see that applying parallelism to it is possible.

3.1 Data Blocks

The first step is to chop the data into blocks with a predefined *BLOCK_SIZE*. Not only the *BLOCK_SIZE* needs to be defined, but also a constant *N* needs to be defined as well. *N* is the maximum number of the blocks can be inputted to the algorithm. We expect *N* = 2^n.

3.2 Data Augmentation

This step is critical to ensure that the reordering the data blocks would not have the same hash code after the combination algorithm. (See Figure 3)

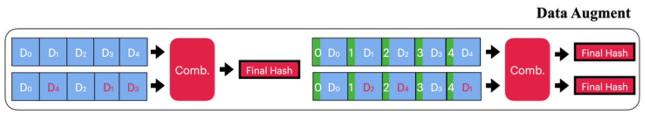

Data Augment
D_0 D_1 D_2 D_3 D_4
D_0 D_4 D_2 D_1 D_3
Comb.
Final Hash
0 D_0 1 D_1 2 D_2 3 D_3 4 D_4
0 D_0 1 D_2 2 D_4 3 D_3 4 D_1
Comb.
Final Hash
Final Hash

Figure 2. Data Augmentation. Left without the augmentation. Right with the augmentation

Once we chop the data into blocks, each block has an index to indicate its position. The data augmentation process is a two-step process. First, encode the index into bits with a length lg(N) where N is predefined in the previous process. Later, we concatenate the index in bits with the corresponding block. Eventually, the length of each data block is $L = BLOCK\_SIZE + \lg(N)$.

This process is parallelable.

3.3 One-way Hash Algorithm

The collision-freeness is affected by the chosen one-way hash algorithm. The candidates need to satisfy two properties. The first one is the computed hash codes need to be a fixed size. The other is the algorithm needs to be collision-free with a fixed-length inputted data. Once the candidates are set, we use a random generator that serves as an oracle to choose the one-way hash algorithm.

We input each data block into the same chosen one-way hash algorithm independently. As a result, we will have a collection of hash codes.

This process is parallelable.

3.4 Combination Algorithm

After the previous process, the final step of the incremental hash algorithm is the combination process. The authors propose three approaches to combine the hash codes to obtain a final hash code. Each of the approaches has its own concerns regarding security. However, we focus on the implementation of AdHash algorithm and its security concerns.

The combination approaches can be many, but one major property they must satisfy. That is reversible. The operations of the combination need to be reversible since, in some cases, we need to remove the hash codes based on the modified data blocks.

4. AdHASH

4.1 Algorithm

AdHash is one of the proposed incremental hash algorithms in [1]. Basically, the combination algorithm is adding. We add every hash code produced by the chosen one-way hash function in the prime field, a special Galois Field, as the following mathematical expression from [1]. (See Expression 1.)

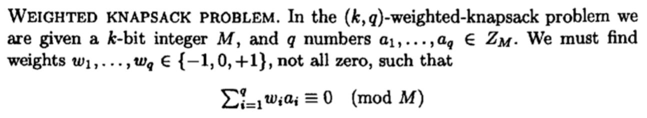
$$\mathrm{AdHASH}^{h}_{M}(x_1 \ldots x_n) \;=\; \textstyle\sum_{i=1}^{n} h(\langle i \rangle . x_i) \bmod M$$

Expression 1. AdHash Expression where h(<i>.xi) means the hash code of the data block i.

The k-bit integer M needs to be decided, and it needs to be a prime number and large enough to satisfy the conditions for a prime field.

4.2 Security

The authors address the security issues with a weighted subset sum problem. (See Figure 4.)



WEIGHTED KNAPSACK PROBLEM. In the (k, q)-weighted-knapsack problem we are given a k-bit integer M, and q numbers $a_1, \ldots, a_q \in Z_M$. We must find weights $w_1, \ldots, w_q \in \{-1, 0, +1\}$, not all zero, such that

$$\textstyle\sum_{i=1}^{q} w_i a_i \equiv 0 \pmod{M}$$

Figure 4. Weighted Subset Sum Problem defined in [1]

We can refer to Prof. Yazdankhah's lecture to understand this problem and the hardness of this problem. Briefly speaking, to solve a knapsack problem when the series of numbers is not super-increasing knapsack, it is a hard problem. The time complexity is O(2^n) where n is the number of data blocks. However, in the defined problem, the weight belongs to a set of {-1, 0, 1}. Therefore, the time complexity is O(3^n), where n is the number of data blocks.

However, as long as there is enough time to execute the brute-force algorithm to find a collision, the security is broken. The authors redefine the meaning of collision-freeness in [1]. We say in a limited time t, the collision-finder algorithm C can find a collision in a limited time of attempt q with a probability of p. We define a Function F is (t, q, p)-collision-free if there is no collision-finder breaks F.

5. EXPERIMENT

The experiment is a comparison between the traditional approach and the incremental approach under the same overhead of the implementation. Both the traditional and incremental results are based on AdHash algorithm. The experiment is not compared with MessageDigest in Java Standard Library because the overhead is different due to the implementation. The implemented algorithm is not optimized.

5.1 Append-Only Files

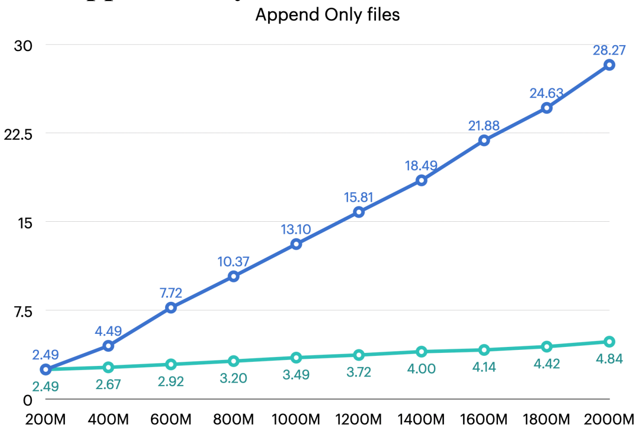


Figure 5. Comparison while applying append-only files

Figure 5 shows that although both approaches have linear growth, but the growth rate is much different. The traditional one is much higher than the incremental one.

5.2 Modified Files

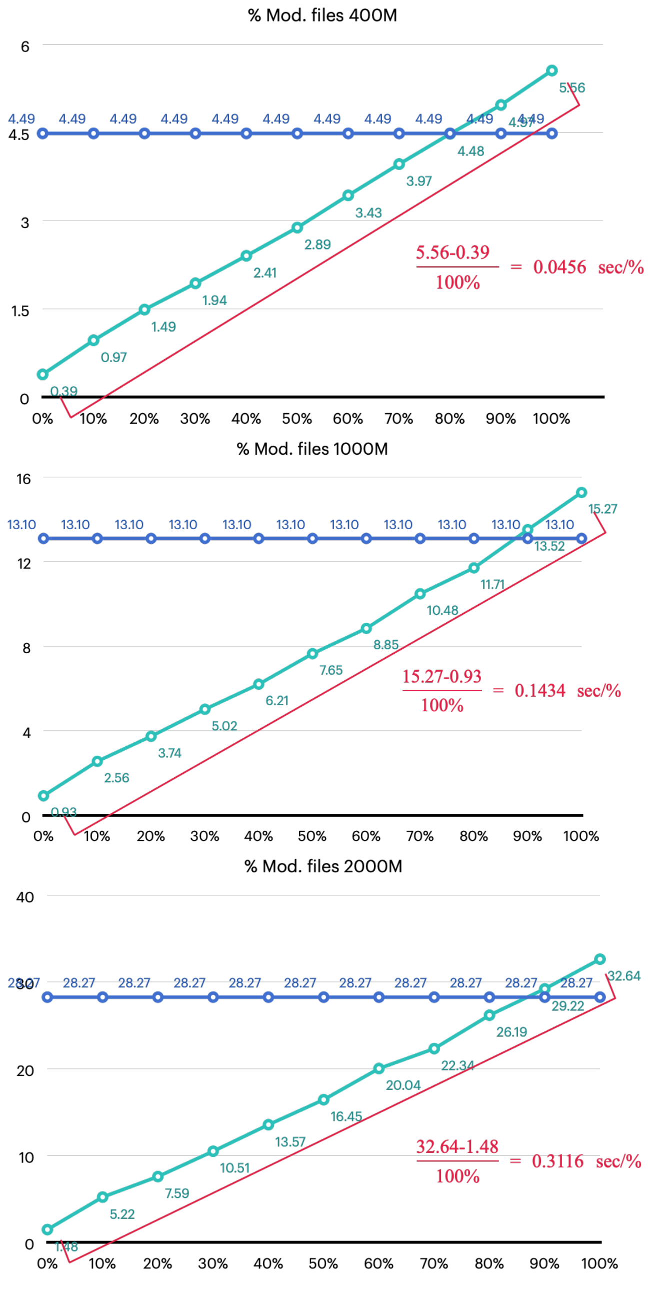


Figure 6. Comparison while applying modification-only files in a fixed size

Figure 6 shows the corresponding time that the algorithms take to compute the new hash code when the files with different percentages of modified portions are inputted. For example, when a file is 50% modified with a file size of 1000MB, the traditional approach and the incremental approach have the results of 13.10 seconds and 7.65 seconds, respectively. We could see that the incremental approach still has a better result when 80% of the file is modified.

5.3 Append and Modified Files

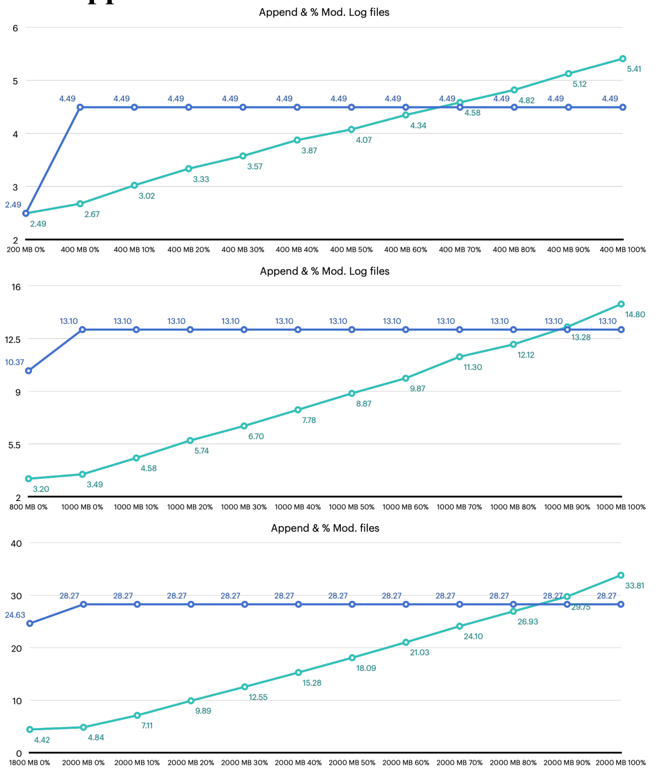


Figure 7. Comparison while applying append and modification files in a fixed total size

Figure 7 shows the corresponding time that the algorithms take to compute the new hash code when the files with different percentages of modified portion, and a fixed size of new data is appended to the file are inputted to the algorithms. For example, when the old file is 50% modified, and a fixed new data of 200MB is appended to the new file with a total size of 2000MB, the traditional approach and the incremental approach have the result of 28.27 seconds and 18.09 seconds, respectively. We could see that the incremental approach still has a better result when 80% of the file is modified.

5.4 Growth Rate on Incremental Algorithm

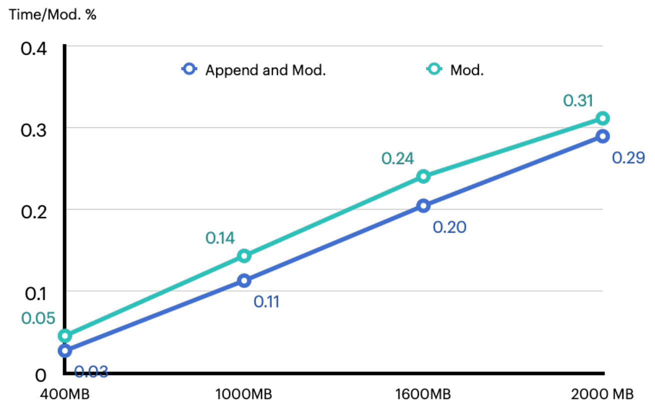


Figure 8. The cost of updating files with different file sizes

Figure 8 shows in either case of the files, the cost for computing the new hash has a linear growth based on the sizes of files.

5.5 Block Size

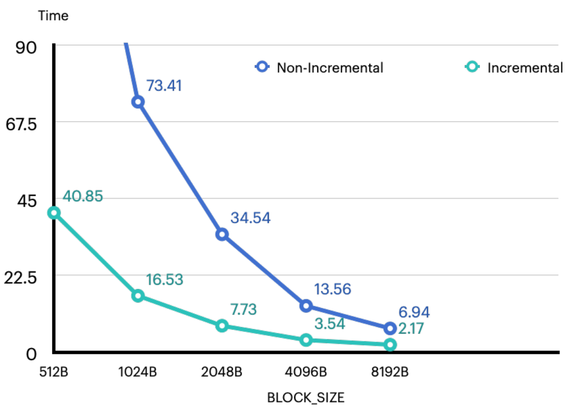


Figure 9. Comparison while applying modification-only files in a fixed size

We can conclude from Figure 9 that the *BLOCK_SIZE* affects the computing time significantly. The difference between the traditional approach and the incremental approach becomes smaller when the *BLOCK_SIZE* becomes greater, which also implies when the larger the *N* is, the poorer the performance the incremental approach is. However, we can also analyze this result with the overhead caused by the actual implementation.

6. SUMMARY

The experiment result indeed shows the fact that the incremental approach is more efficient in most of the cases. The security is guaranteed as long as the weighted subset sum problem is still hard, as stated by the authors [1]. Although there are many improvement and optimization can be done with the current implementation in Java to reduce the overhead in order to render a more advanced experiment result. However, because the original research does not address many details on the security part, further analysis and researches regarding the potential attack are desirable.

7. REFERENCES

[1] Mihir Bellare and Daniele Micciancio. 1997. A new paradigm for collision-free hashing: incrementality at reduced cost. In Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'97). Springer-Verlag, Berlin, Heidelberg, 163–192.

[2] Yongjun Zhao and Sherman S.M. Chow. 2017. Updatable Block-Level Message-Locked Encryption. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17). Association for Computing Machinery, New York, NY, USA, 449–460. DOI:https://doi.org/10.1145/3052973.3053012