2014 IEEE 28-th Convention of Electrical and Electronics Engineers in Israel

Accelerating Duplicate Data Chunk Recognition Using NN Trained by Locality-Sensitive Hash

Amit Berman, Yitzhak Birk, Avi Mendelson
Electrical Engineering Department
Technion – Israel Institute of Technology
Haifa, Israel
{bermanam@tx, birk@ee, avim@ee}.technion.ac.il

Abstract— **Deduplication is often used in storage systems in order to save storage space, communication bandwidth, write energy, and recovery and error-protection infrastructure. However, deduplication overhead increases latency and computation energy. Determining whether a data chunk is already stored by comparing signatures constitutes a significant fraction of this deduplication overhead. In this paper, we propose a statistical chunk classifier based on a neural network. Our technique is based on learning the patterns of locality-sensitive hashing of the data. Our experiments show an acceleration of chunk processing, leading to reduction in deduplication overhead.**

Index Terms— Deduplication; Chunking; Cloud Storage; Neural Network; Machine Learning; Locality-Sensitive Hashing;

I. INTRODUCTION

Data is generated and stored at an ever increasing rate, yet, we have come to expect to pay less for more over time. Although the device-level cost per stored bit is low and declining, this is only partly reflected in system cost due to the need to satisfy demands such as data robustness, availability and performance. For example, storing all data in solid state drives (SSD) increases performance, but also dramatically increases cost. Therefore, cost-effective data storage and management are of utmost importance.

In some of the dominant storage-intensive applications, the same data is stored multiple times. This is sometimes done for fault tolerance, but in many cases it is the result of applications. For example, mail sent to multiple recipients may be stored multiple times; different users of the same shared computer or cloud system may have copies of the same system files. Finally, programs that periodically back up systems may store identical or similar copies of the same file [1, 2, 3].

The above observations have given rise to the area of deduplication (dedup). This is basically lossless compression. However, the vast amounts of data involved and the fact that there isn't necessarily any "proximity" between copies, have led researchers to very different approaches. Generally speaking these are coarse grain approached. For example, looking for identical blocks or files rather than some adaptive dictionary as in lossless compression (e.g. [4]). In fact, compression and deduplication may sometimes interfere with each other, compressing identical blocks of data in different compression contexts would produced different compressed versions and would go undetected by a deduplication algorithm. (Similarly, encryption interferes with both compression and deduplication.)

Both the performance requirements and the cost elements of deduplication depend both on application needs (and related service level agreements) and on the overall system architecture. For example, deduplication onto a small removable storage device may be latency sensitive (as the owner wants to pull out the device), while in a large storage system the main issue is throughput because buffering is possible. Similarly, in a centralized system the main cost element is computation, whereas in a distributed cloud system the required communication may be of great importance [5].

Deduplication is almost inherently sub-optimal, in that one could achieve greater compression at the cost of much more computation and time. For a dedup scheme to be correct, it only needs to ensure that an incoming chunk is not falsely determined to be identical to an existing one. Deduplication schemes thus vary dramatically: the simplest ones only avoid duplication of identical files, often based on examining their metadata; others operate at block granularity, but any change that causes blocks to become different (e.g., deleting the first character of a multi-block text file, thereby causing a shift of all remaining characters by one position, causes all blocks to become different from their earlier versions); more sophisticated ones may "slide across" an incoming chunk of data in order to determine whether any block-size contiguous set of bits is identical to a stored block. Dedup schemes can be compared based on the achieved space savings, based on performance, cost of any combination thereof.

Deduplication entails three main steps: 1) dividing the data into variable or fixed chunks (chunking); 2) calculating a signature for each chunk and comparing it to the ones which are already stored (processing). In case of match an apparent match, validation is carried out via direct comparison of the two chunks; 3) chunk storage or metadata update to point to existing chunk.

Our focus in this paper is on accelerating chunk processing (step 2). Specifically, we focus on the computation and amount of metadata required in order to determine whether a new

Authorized licensed use limited to: HOLON ACADEMIC INSTITUTE OF TECHNOLOGY. Downloaded on January 27,2021 at 07:30:40 UTC from IEEE Xplore. Restrictions apply.

chunk is identical to an already stored one. This also affects memory traffic and possibly other communication.

Several approaches have developed to accelerate duplicate chuck recognition. Content-addressable storage (CAS) [6] stores a chunk in an address equal to a hash value computed from its content. However, it is inflexible and makes scaling and difficult because addresses are pre-allocated and "consumed" even if not in use. (For example, 8 Peta-bytes would be required in support of a 40 bit hash in order to store a pointer to the actual stored chunk for each possible hash value. Of course, additional levels of indirection could mitigate this requirement.) Moreover, hash collisions are possible.

Bloom filter [7] enables a probabilistic duplication search. However, its efficiency decreases and implementation complexity increases as the search population grows.

In hash comparison and Bloom filter approaches, the search of whether the hash is already stored and its locating is performed in the same transaction. This may lead to unnecessary complex search when hash is not stored, as shown in Fig. 1. New approaches decouple the chunk existence and location searches, in order to avoid such spare searches.

Statistical classifiers [8, 9, 10] were proposed for detecting duplicates based on neural networks and active learning of past data chunks. Active learning is used to select data sets that provide high information gain to the learner. The main advantage of such classifiers is the decoupling of duplication estimation from total storage size. The disadvantage of those techniques is the limited classification accuracy.

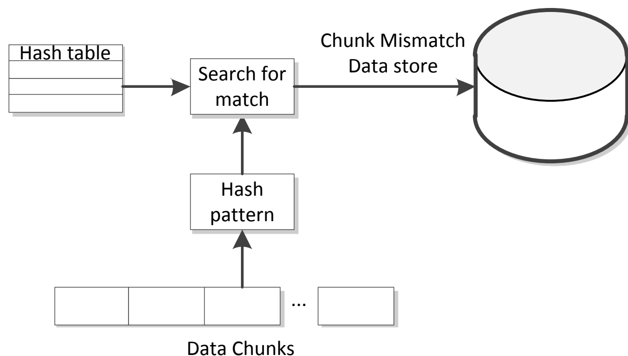


Fig. 1. Spare chunk search during deduplication.

In this paper, we propose a neural network classifier, trained by locality-sensitive hash for duplicate chunk recognition. Our method is based on low complexity hash functions and neural network architecture. Our method differs from [8, 9, 10] by utilizing locality-sensitive hash similarity model.

II. Chunk Processing with Neural-Network

We design a probabilistic neural-network based chunk classifier to identify chunks that are already stored. Chunk size is expected to reach several kilo-Bytes, so we reduce the computation complexity by examining the hash value of each chunk. The framework of deduplication system design consists of the following steps:

- Hash function selection
- Neural network size and architecture
- Neural network training
- Classification threshold determination
- Error estimation

We next discuss each step in detail.

A. Hash Function Selection

In order to increase the efficiency of neural network, similar chunks have to produce higher values than unique chunks. Therefore, similar chunks have to produce similar hash values. In order to achieve this, locality-sensitive hashing (LSH) functions are utilized [11]. In our design, LSH function is implemented by bit sampling at the required hash length. The bit indices are constant, selected randomly at system initialization time.

B. Neural network size and architecture

Neural networks (NN) enable to learn large amounts of features. Therefore, any increase of NN size would reduce classification error. However, such an increase may cause additional computation and latency. We used a 3-layer NN (Fig. 3). Input hash size is 32 bits, so the input layer includes 33 neurons (32 bit hash value and bias unit); there are 33 neurons in the hidden layer, and a single neuron in the output layer. Each neuron is connected to all neurons in the next layer.

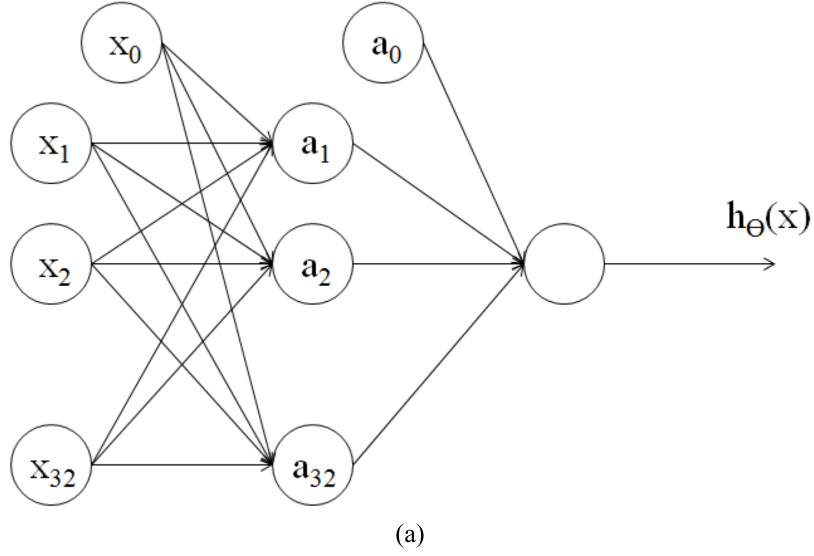
C. Neural Network Training

In order to set adaptive statistical classifier, we have two-phase training procedure. In the 1st phase, referred to as pre-training, we train the NN with given pass data. This data consists of the hashes of the stored data. Second phase learning is ongoing during deduplication system run: if a chunk is marked as stored by NN but actual storage search did not result in finding a duplicate, the NN is trained with the chunk's hash. The pre-training is done by the following steps: random initialize weights Θ, implement forward propagation algorithm to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$. Compute cost function $J(\Theta)$. Perform backpropagation and forward propagation with gradient decent to try to minimize $J(\Theta)$ as a function of Θ. All those are known NN algorithms. In the second learning phase, we perform only the last step, namely just backpropagation and forward propagation with gradient decent according to new added hash.

D. Classification Threshold Determination

The NN outputs a probability that the chunk is already stored. A threshold value is required in order to decide whether to store the chunk as there is no identical stored chunk or to speculate that a duplicate does exist and go on to find it and verify, then store a pointer to it. The choice of the threshold value represents a trade-off between false positive (resulting in extra computation) and false negative (resulting in wasted storage space) decision probabilities. (Note that a mistake does not result in an actual error: false negative results in storing an extra chunk, and false positive merely results in extra work because a full comparison is carried betweenthe new chunk and the one believed to be identical to it as a means of perfect verification.) In order to determine the threshold, we collect

Authorized licensed use limited to: HOLON ACADEMIC INSTITUTE OF TECHNOLOGY. Downloaded on January 27,2021 at 07:30:40 UTC from IEEE Xplore. Restrictions apply.



(a)

$$a_1 = g\left(\Theta^{(1)}_{1,0}x_0 + \Theta^{(1)}_{1,1}x_0 + \ldots + \Theta^{(1)}_{1,32}x_{32}\right)$$

$$a_2 = g\left(\Theta^{(1)}_{2,0}x_0 + \Theta^{(1)}_{2,1}x_0 + \ldots + \Theta^{(1)}_{2,32}x_{32}\right)$$

$$\ldots$$

$$a_{32} = g\left(\Theta^{(1)}_{32,0}x_0 + \Theta^{(1)}_{32,1}x_0 + \ldots + \Theta^{(1)}_{32,32}x_{32}\right)$$

$$h_\Theta\left(X\right) = g\left(\Theta^{(2)}_{1,0}a_0 + \Theta^{(2)}_{1,1}a_1 + \ldots + \Theta^{(2)}_{1,32}a_{32}\right)$$

$$g\left(z\right) = \frac{1}{1+e^{-z}}\ ;\ \Theta^{(1)} \in R^{32x33}\ ;\ \Theta^{(2)} \in R^{1x32}$$

(b)

$$J\left(\Theta\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log\left(h_\Theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\Theta\left(x^{(i)}\right)\right)\right] +$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{2}\sum_{i}\sum_{j}\left(\Theta^{(l)}_{ji}\right)^2$$

(c)

Fig. 2. Neural network model. (a) NN architecture (b) $a_i^{(j)}$-activation of unit i in layer j, $\Theta^{(j)}$ – matrix of weights contolling function mapping from layer j to layer j+1. (c) cost function. NN performance is optimized when $J(\Theta)$ is minimized.

$X=x_1x_2\ldots x_{32}$ is the hash value, m is the number of training chunks, each is a pair $(x^{(i)},y^{(i)})$ where $x^{(i)}$ is the hash value, $y^{(i)}=0$ if chunk is not in storage and 1 if stored. λ is the regularization parameter and can be arbitrary number.

Authorized licensed use limited to: HOLON ACADEMIC INSTITUTE OF TECHNOLOGY. Downloaded on January 27,2021 at 07:30:40 UTC from IEEE Xplore. Restrictions apply.

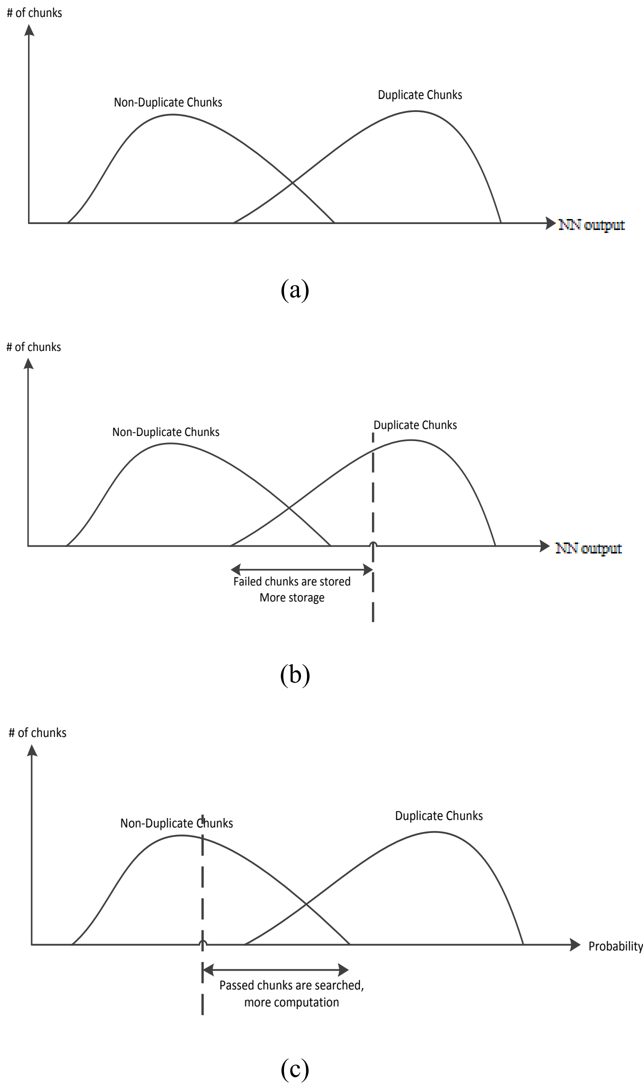


Fig. 3. Classification threshold determination. (a) pass and fail chunk distributions. (b) threshold is in favor of non-duplicate chunks results in unessessary chunk storage (c) threshold is in favor of duplicate chunks results in unessessary chunk search comupatation.

two probability distributions. The "pass" distribution is the collection of all chunks hashes and their corresponding probabilities that are already stored. Similarly, "fail" distribution is the collection of chunk hashes that are not stored and their NN assign probabilities. The "pass" and "fail" distributions are shown in Fig. 2(a) right and left curves accordingly. The threshold is a probability value, in which above it or equal would result in chunk search, and below it chunk processing results in immediate chunk storage. Fig. 2(b) shows an example of the high threshold (in favor of non-duplicate chunks), where chunk duplicates are marked as new and stored (unnecessary chunk storage). Fig. 2(c) shows an example of low threshold (in favor of duplicate chunks), where new chunk (non-duplicate) are marked as duplicate and searched in the storage system (unnecessary computation).

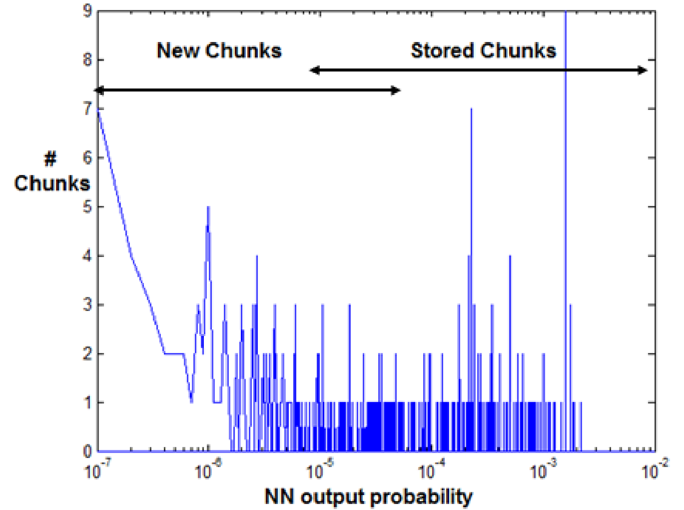


Fig. 4. Probability distribution of pass and fail chunks in dataset experiment. Error probability ranges from 0.7 to 0.9 depends on dataset and threshold location.

E. *Error Estimation*

Error probability is calculated as the overlap area between duplicate and non-duplicate chunks distributions. Detailed analysis assigns different error weights for storage and computation according to cost or other metric. For example, the energy spent on unnecessary chunk search is modeled to be half of spare chunk storage. The error probability is calculated apostriori and accumulated as more data is stored. Threshold is dynamically adjusted for error minimization.

The suggested NN-based chunk processing scheme is depicted in Fig. 4.

III. Experimental Results

We tested our chunk processing scheme with a dataset of 200MB, consists of 100MB training and duplicate data and 100MB of non-duplicate data. The dataset is a our own hard drive data that contains various file types including operating system, text, audio, pictures, video, web browsing and software tools such as matlab.

Dataset file sizes range from 4Kbyte to 10Mbyte. Each file is divided into fixed 4Kbyte chunks, hashed by locality-sensitive random bit sampling, and applied to NN chunk classifier. The resulting distribution of pass and fail chunks are shown in Fig. 5. The overlap region is between 10^{-6}-10^{-5}.

Our search in replacing the threshold is found to be $3\cdot10^{-5}$ which leads to 0.3 error probability. Latency is also accelerated.

IV. Conclusions and Future Work

In this work we build neural-network based chunk processing scheme for probabilistic classification between duplicate and non-duplicate chunks in data deduplication. We improved NN chunk duplication detection by using locality-sensitive hash

Authorized licensed use limited to: HOLON ACADEMIC INSTITUTE OF TECHNOLOGY. Downloaded on January 27,2021 at 07:30:40 UTC from IEEE Xplore. Restrictions apply.

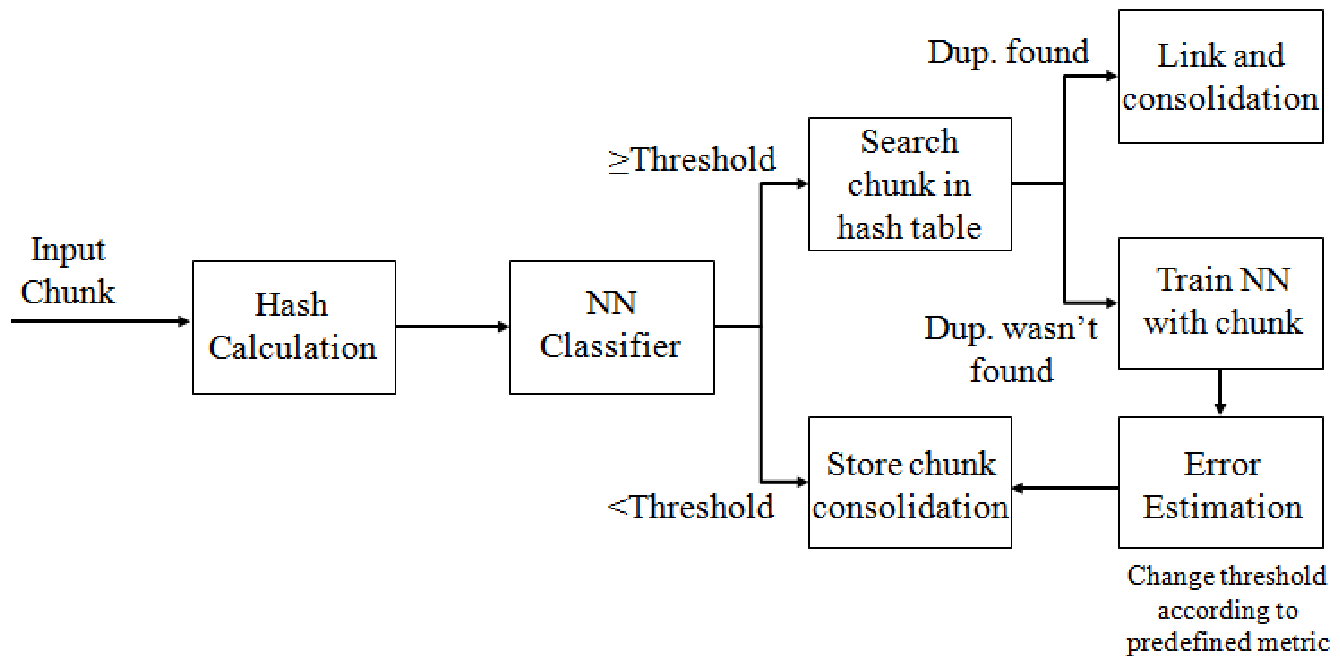


Fig. 5. Suggested NN-based chunk processing system.

as a training set. Our learning-based classifier is scalable due to the decoupling between the duplicate chunk detection and the amount of storage space (unlike Bloom filter and other processing schemes). We described the design steps and considerations to build the NN statistical classifier. We implemented and demonstrate the classifier on real dataset and achieve relatively high detection results.

Future work includes examination of the classifier in various datasets, and threshold parameter optimization techniques to minimize error rate, computation and memory resources, and exploration of the precision and recall trade-off.

Acknowledgment

This work was supported in part by HPI-Technion research school for scalable computing.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., et al. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58. ACM.

[2] N. M. Chowdhury and R. Boutaba, "A survey of network virtualization," Computer Networks, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[3] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," Security Privacy, IEEE, Nov. 2010.

[4] J. Ziv, A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Tran. On Information Theory, Vol. IT-23 1977.

[5] Netherlands, June 2009.D. Chappell. Introducing the Azure services platform. White paper, Oct. 2008.

[6] Ungureanu et-al., "HydraFS: a high throughput file system for the HYDRAstor CAS system", FAST'10.

[7] G. Lu, Y. Nam, and D. Du, "BloomStore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash," in Proceedings of the 28th IEEE Symposium on Massive Data Storage (MSST/SNAPI), 2012.

[8] P. Christen, "Probabilistic data generation for deduplication and data linkage", Proceedings of IDEAL 2005.

[9] J. Liu, "Data Deduplication using Neural Networks", Proceedings of SPIE Vol. 2304, 1994.

[10] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discoveryand Data Mining (KDD-2002), Edmonton, Alberta, 2002.

[11] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Localitysensitive hashing scheme based on p-stable distributions. In Proc. ACM Symp. on Computational Geometry, pages 253–262, 2004.

Authorized licensed use limited to: HOLON ACADEMIC INSTITUTE OF TECHNOLOGY. Downloaded on January 27,2021 at 07:30:40 UTC from IEEE Xplore. Restrictions apply.