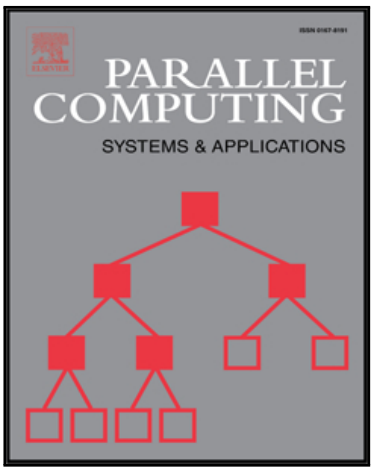

Vectorizing Disks Blocks for Efficient Storage System via Deep Learning

Dong Dai, Forrest Sheng Bao, Jiang Zhou, Xuanhua Shi, Yong Chen

PII: S0167-8191(18)30080-2
DOI: 10.1016/j.parco.2018.03.003
Reference: PARCO 2438

To appear in: *Parallel Computing*

Received date: 20 September 2016
Revised date: 19 November 2017
Accepted date: 25 March 2018

Please cite this article as: Dong Dai, Forrest Sheng Bao, Jiang Zhou, Xuanhua Shi, Yong Chen, Vectorizing Disks Blocks for Efficient Storage System via Deep Learning, *Parallel Computing* (2018), doi: 10.1016/j.parco.2018.03.003

Vectorizing Disks Blocks for Efficient Storage System via Deep Learning

Dong Dai[a], Forrest Sheng Bao[b], Jiang Zhou[a], Xuanhua Shi[c], Yong Chen[a]

[a]*Dept. of Computer Science, Texas Tech University, Lubbock, Texas*
[b]*Dept. of Computer Science, Iowa State University, Ames, Iowa*
[c]*School of Computer, Huazhong University of Science and Technology, Wuhan, China*

Abstract

Efficient storage systems come from the intelligent management of the data units, i.e., disk blocks in local file system level. Block correlations represent the semantic patterns in storage systems. These correlations can be exploited for data caching, pre-fetching, layout optimization, I/O scheduling, etc. to finally realize an efficient storage system. In this paper, we introduce Block2Vec, a deep learning based strategy to mine the block correlations in storage systems. The core idea of Block2Vec is twofold. First, it proposes a new way to abstract blocks, which are considered as multi-dimensional vectors instead of traditional block Ids. In this way, we are able to capture similarity between blocks through the distances of their vectors. Second, based on vector representation of blocks, it further trains a deep neural network to learn the best vector assignment for each block. We leverage the recently advanced word embedding technique in natural language processing to efficiently train the neural network. To demonstrate the effectiveness of Block2Vec, we design a demonstrative block prediction algorithm based on mined correlations. Empirical comparison based on the simulation of real system traces shows that Block2Vec is capable of mining block-level correlations efficiently and accurately. This research and trial show that the deep learning strategy is a promising direction in optimizing storage system performance.

Keywords:
Storage System, Block Correlation, Deep Learning, Word2Vec, Natural Language Processing, Intelligent Storage

1. Introduction

In the Big Data era, the performance of storage systems has become increasingly important. High-performance distributed file systems have drawn increasing attention for both high-performance computing and cloud computing platforms [1, 2, 3, 4]. Although in distributed file systems, factors like data placement [5], metadata management [6], fault tolerance [3], and scheduling [7] all affect the overall performance, the I/O requests will eventually arrive at individual disks at the back-end storage servers and rely on their performance to deliver an overall optimal performance. Hence, to achieve the best I/O performance in distributed and parallel file systems, block-level access in local disks is also critical to be optimized.

It is acknowledged that the correlations between disk blocks are critical to the improvement of storage disk performance [8, 9, 10]. The correlation can be used for re-organizing data blocks to maximize the sequential access [11, 12], pre-fetching blocks to reduce I/O latency, or increasing the cache-hits [13, 14]. The capability of being able to detect the correlations between blocks and use them to direct local storage optimization is important to not only local systems but also large-scale distributed systems.

The correlations between blocks in storage systems come from several sources. First, the sequential blocks are more likely to be accessed together (i.e., the locality principle [15]), indicating close correlations among them. Second, applications may introduce temporal correlations among blocks that are not physically close to each other. For example, the B-tree or B*-tree data structures in database systems will introduce continuous accesses between blocks that store parent nodes and child nodes. The Ext3 or Ext4 file systems also introduce temporal correlations between blocks where the file data is stored and the blocks the corresponding *inode* [16] is stored. Last but not the least, multiple applications that interact with each other also introduce block correlations. Such applications commonly exist in the form of workflow in scientific computing.

As correlated blocks tend to be accessed relatively close to each other in an access stream, it is possible to detect them by analyzing the block access stream. Generally, there are two types of block access streams, at *application-level* and at *system-level*. The *application-level* access stream is collected in a per-application way, and a *system-level* access stream is collected at the system level as a reflection of data accesses from all applications and even the operating system (OS) itself. In this research, we focus on the system-level access stream for several reasons. First, system-level block accesses are easier to collect compared to per-application block access collection, as applications access data at the file level, which needs to be mapped to blocks and also differentiated from other applications and the OS. Second, system-level streams are more accurate in terms of the access order than application-level streams, because they are not affected by operating system I/O scheduling. Third, system-level access streams can further reflect the correlations generated from interactions among applications and can be more useful to optimize the storage system.

Given a continuous block access stream at the system level, the objective of this research is to infer accurate block correlations based on access context. In fact, there has been a large number of research efforts focusing on this specific or relevant problem. The probability graph [17, 18] was designed to record the access closeness of blocks as a graph, whose edges denote how many times two blocks are shown together. Such graph can be further used to predict next IO access based on the current one. Semantic distance [12] leverages the knowledge of how the file system uses the disk system including on-disk data structure to infer relationship of blocks. Small artificial network, which contains 13 input units, 10 output units, and 12 units in the hidden layer, was proposed in [19] to learn the sequence of I/O accesses. The Markov model and Hidden Markov Model [20] abstract the access sequence as a series of changing states and use them to predict next possible access. Pattern signatures [21, 22] were used to capture application signatures based on their I/O behaviors. Some complex data mining algorithms, especially frequency mining, were also proposed to mine the frequent patterns in I/O access traces for correlations [23, 24].

In this study, we propose *Block2Vec*, a strategy that applies the advanced deep learning (DL) technique to efficiently and accurately mine the block correlations from large-size block access streams. The core idea of Block2Vec contains two aspects. First, we propose a new way to *abstract disk blocks as multi-dimensional vectors instead of traditional indexes to better abstract their features*. Second, we propose to *train a deep neural network to learn the best vector value for each block by training on real-world block access traces*. The training algorithm is inspired by the word embedding algorithm called word2vec [25, 26]. To the best of our knowledge, Block2Vec is the first approach to leverage the deep learning techniques into block correlation mining. It is also the first study that models disk blocks into high-dimensional vectors instead of consecutive, single dimensional block indexes. We conduct extensive evaluations through a common use case, disk access prediction, to analyze the benefits of exploiting block-level correlations via the newly proposed approach. Compared to existing block correlation detection algorithms including sequential prediction (SP) and probability graph (PG), Block2Vec achieves impressive and stable prediction accuracy with a low overhead. These evaluations based on real-world I/O traces also confirm that Block2Vec is able to run efficiently with reasonable resources as an effective block correlation mining method for modern storage systems.

The rest of this paper is organized as follows. In Section II, we introduce the core ideas of this research study: a generic model for mining block correlations, which contains a statistical block access model and a new vector-based representation of disk blocks. In Section III, before discussing the detailed design and implementation of proposed approach, we give a brief introduction of deep learning (DL) techniques and the highly relevant word embedding tasks in natural language processing (NLP). In Section IV, we present the detailed design and implementation of Block2Vec. In Section V, we further introduce a simple correlation-based block prediction algorithm and a semi-supervised block clustering algorithm for block reorganization as two applications of detected block correlation. We also use them as evaluation metrics in the later section. Section VI presents our experimental results. Section VII concludes the paper and discusses future work.

2. General Model for Mining Block Correlations

2.1. Statistical Block Access Model

It is observed that correlated blocks are accessed close to each other because of their data relevance. Therefore, if two blocks are almost always accessed together within a short time interval, it is very likely that these two blocks are correlated to each other. In other words, it is possible to automatically infer block correlations in a storage system

by analyzing the access stream. This observation plays the foundation of mining block correlations. However, it is not trivial to quantitatively measure such correlations from this intuitive observation. The similar problem also exists in natural language processing (NLP), where similar words also show in the similar context or closely with each other, e.g., "Obama" with "President". To be able to quantitatively measure such word similarity, statistical language model [27] is proposed and widely used. Inspired by the statistical language model, we propose to use statistical model to describe the block accesses in this work.

The statistical model describes the probability of a block, say b_4, being the next one to be accessed, given an existing block access sequence, say $\{b_1, b_2, b_3\}$. This value reflects the block correlations among b_4 and b_1, b_2, b_3, which can be mathematically calculated as the conditional probability of sequence $\{b_1, b_2, b_3, b_4\}$ given $\{b_1, b_2, b_3\}$. In this way, mining block correlation can be considered as building a statistical model to estimate the probability of a certain group of blocks. The larger the probability is, the closer the group of blocks are. For example, if $P\{b_4|b_3\} = 0.9$, then we know that b_3 and b_4 are closely correlated.

In a formal description, the statistical model of block access calculates the probability that a given block sequence appears. This can be calculated based on the probability of each block and its conditional probability in the sequence as follows.

$$\begin{aligned} P(b_1, \ldots, b_m) &= P(b_1, \ldots, b_{m-1}) \cdot P(b_m|b_1, \ldots, b_{m-1}) \\ &= \prod_{i=1}^{m} P(b_i|b_1, \ldots, b_{i-1}) \end{aligned}$$

Here, each conditional probability can be calculated by counting corresponding access pattern based on given block access traces. However, as a block access sequence can easily contain many blocks, it will quickly become impractical to compute such a value. We can approximate it by only considering a fixed number of preceding blocks, for example n. This approximation is widely used in statistical language model too, known as the N-gram [28]. In this case, the probability of observing a block sequence $b_1, \ldots, b_m$ is given as:

$$\begin{aligned} P(b_1, \ldots, b_m) &= P(b_{m-n+1}, \ldots, b_{m-1}) \cdot \\ &\quad P(b_m|b_{m-n+1}, \ldots, b_{m-1}) \\ &= \prod_{i=1}^{m} P(b_i|b_{i-n+1}, \ldots, b_{i-1}) \end{aligned}$$

Here, it is assumed that the probability of observing the i-th block b_i in the context history of $i-1$ preceding blocks can be approximated by the probability of observing it in the shortened context history of the preceding $n-1$ blocks (n-th order Markovian property). With the capability of calculating such probability for any block sequence, we can find block b_m with the maximal likelihood of forming the entire sequence and choose it as the next prediction.

As we have described, it is not difficult to observe that the statistical block access model is similar to its natural language cousin, the statistical language model, which is a probability distribution over sequences of words. In fact, such similarity behind these two fields comprises the rationale and motivation for this work. In this research, for the first time, we explore applying those techniques, used to build statistical language model in NLP, onto block correlation mining for storage systems.

2.2. Vector Representation of Blocks

Most existing block correlation detection strategies treat each block as an atomic unit and represent them as indexes in the total physical blocks (e.g., block Id b_i). Such representation is similar to "one-hot" representation [29] in NLP. In this way, mining block correlation becomes learning the similarity between two unique integer Ids based on block access sequences. This representation of blocks is simple and straightforward, but it has several limitations. First, the representation itself (e.g., block Id) does not contain the notion of correlation similarity as they are just integer indexes of blocks, indicating the location of blocks. For example, even block b_1 and b_{200} are closely correlated, their representations $(1, 200)$ cannot show it. Second, it suffers the *curse of dimensionality* problem whenever a machine

learning or pattern detection algorithm is applied on the block accesses traces to learn the pattern [30]. Specifically, as a storage system usually contains a large number of blocks, if we just use an integer Id to represent each block, the number of all possible sequences of block accesses will increase exponentially with the number of the blocks. This indicates a huge number of combinations of values (i.e., block accesses) that must be discriminated from each other by the pattern mining algorithm, which significantly increases the computation complexity. However, using a vector representation of blocks, the learning algorithm can generalize one combination of values (vectors) to possibly a large number of combinations of similar vectors, hence improve the training accuracy.

In this research, for the first time, we propose to use a vector instead of an index to represent a block. It will associate each block with a continuous-valued vector, instead of only representing the physical location through a block Id. The vector representation has two obvious advantages. First, the vector indicates a tuple of features that characterize the block. Once the vector representations are assigned to blocks, each block corresponds to a point in a high-dimensional feature space. One can imagine that each dimension of that space is related to a characteristic of the block. For example, the physical location of block, the file it belongs to, the user who owns the file, the creation time, the access time, etc. The rationale is that the correlated blocks will get to be closer to each other in that space. In this way, a sequence of block accesses can thus be transformed into a sequence of these feature vectors, which can be efficiently learned by a deep neural network. Second, the vector representation allows the model to generalize the access sequences that have not yet shown in the existing block access history, but are similar to existing ones in terms of their features, i.e., their vector representations. For example, assume that the vector representations of blocks b_1 and b_{10} are similar, and those of b_2 and b_{20} are also similar. In the block access history, the sequence of (b_1, b_2) occurs multiple times and is always followed by b_3. Later, if we see the sequence (b_{10}, b_{20}), which never occurs in history, using their vector representations can help generalize them to the known sequence (b_1, b_2) and help make a prediction about next visit, b_3 or a block similar to b_3. The ability to generalize block access sequences into combinations of high-dimensional vectors will significantly alleviate the problem of the *curse of dimensionality* as many different combinations of block accesses are merged into similar vectors.

The vector representation of blocks can be trained through a neural network [29]. But, it still indicates a high computational complexity, to train on a large number of block accesses. For example, a normal 1TB disk may contain billions of blocks (4KB block size). Recently, in NLP, new architectures and training models have been proposed to reduce the complexity of training a complex neural network on large datasets [25, 26]. This plays a foundation for our research. In the next section, we will give a brief introduction on the deep neural network and the progress in NLP before describing our proposed block correlation mining approach.

3. Deep Learning and Word Embedding

Deep learning is a rapidly emerging machine learning technique originally proposed as an extension to the traditional artificial neural network (ANN) [31]. The word "deep" comes from not only the fact that such an extended ANN normally has multiple hidden layers forming a deep hierarchy as compared to traditional ANN, but also its capability to perform complicated transformations to reach a high-level abstraction of the data.

An ANN transforms input data to output, usually called *prediction*. The powerfulness of ANNs is rooted in their capability to mimic any function numerically under a tolerable precision, if given enough layers and/or nodes/neurons. An ANN usually has a fixed architecture, i.e., the connections between nodes. By changing the weights between connections, the ANN can be reconfigured to mimic different functions. Known as the *training* process, an ANN usually uses an iterative process to update those weights based on many pairs of input data and expected output (sometimes called the *label* or *target*). Given an input data, an untrained ANN most likely produces a prediction output that is very off from the target. As long as there is a large enough error between the target and the prediction, the training will keep going and in each iteration of the training, the weights of all connections are updated. In most cases, the more complex the network is, the more data is needed to train the ANN. Therefore, the numbers of data (thus, iterations to train an ANN) and weights to update (thus, computational cost at each iteration) require massive computational power to train a highly complex ANN. This makes deep learning impractical and unpopular until recently, especially after the wide use of Graphical Processing Unit (GPU) in scientific research.

Numerous research results have shown that DL, compared with traditional neural networks, can build much more accurate and robust models. The reason, as mentioned earlier, is because a more complex ANN can mimic a more complex function. Therefore, DL is currently widely used in many fields including NLP. One of the most exciting uses

of DL in NLP is word embedding [32], which maps words to high-dimensional vectors of real numbers by training a deep artificial neural network [29]. Such a vector is called a "distributed representation" of an object. Unlike its common meaning in super or parallel computing, here the word "distributed" means vectorized, i.e., an objected is presented using more than one real numbers. Representing (i.e., embedding) sparse objects (e.g., words) into vectors makes many tasks convenient. For example, the similarity or correlation between two objects can be calculated by the cosine of the angle between the two corresponding vectors.

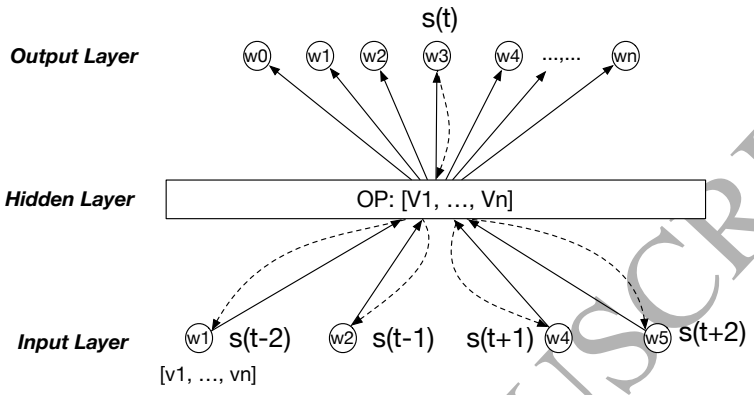

Figure 1: A neural network example for calculating word embedding for English words (e.g., top 50,000) by solving the "filling-the-blank" problem. The training is done by using numerous 5-word sequences, e.g., "*the cat sits on mat*", from English text. The input is 4 words in the 5-word sequence except the middle word, e.g., "*the cat __ on mat*". Each neuron in the output layer corresponds to one of the n words. A correct prediction will yield the highest probability for the word "*sits*". A wrong prediction will trigger the neural network to update vector representation of words and the weights between neurons.

Fig. 1 shows an example deep neural network that can generate the embedding value for each word by training on a set of 5-word sequences, such as "the cat sits on mat" or "a boy eats his dinner". For each input sequence, we calculate the possibility of the middle word $s(t)$, knowing its context $s(t-2)$, $s(t-1)$, $s(t+1)$, $s(t+2)$. The input layer contains nodes correspond to the 4 context words where each context word is a high-dimensional vector $[v_1, \ldots, v_n]$ connected to the hidden layer. The hidden layer normally applies a simple operation, e.g., summation, on all vectors from the input layer and sends results to the output layer. The output layer is huge as each node corresponds to one word in the vocabulary. Each output node produces an output between 0 and 1, i.e., the probability that the corresponding word is the middle word $s(t)$, and the sum of all these output values will add up to 1. The output-layer node that corresponds to the middle word $s(t)$ is expected to have the largest output (the probability).

The training rationale is straightforward. Initially, each word is represented as a random vector. Hence, at the early stage of training, the prediction is very likely to be wrong. For any given training sequence, the expected word (target) should have the maximal probability in the output layer. If the hidden layer yields a word other than the target, the weights and vectors will be updated by back-propagation [33]. Afterwards, the updated vectors of input words will be copied to the output layer. In this way, a network can be trained to both predict the possibility of word and generate accurate vector representation for each word. After training the network using a huge amount of word sequences, we will be able to obtain the vectors to represent words.

After words are represented in vectors, many tasks can be done easily by operations on vectors. For example, vec("Germany") + vec("capital") is close to vec("Berlin"). This can help many tasks such as question answering (e.g., what is the capital of Germany?) or knowledge base construction (e.g., if we know that Berlin is the capital of Germany, and we also know vec("France") + vec("capital") is close to vec("Paris"), we can gain new knowledge that Paris is the capital of France).

Training such a neural network for the accurate vector representation of words is an active research topic [34] in NLP with broad applications, such as machine translation [35, 36] and sentiment analysis [37]. Word2Vec [25] is a representative one due to its optimized network architecture and training models, which save a huge amount of computation.

In this study, inspired by word embedding, we propose to use high-dimensional vectors to represent disk blocks.

Like Word2Vec which allows studying the similarity between words by vector operations, our method, known as *Block2Vec*, allows accurate mining of the correlation between blocks. To the best of our knowledge, our study is the first to use distributed/vector representation and deep learning techniques for such a purpose.

4. Block2Vec Design and Implementation

In this section, we introduce the design and implementation of *Block2Vec*, a deep neural network-based solution to learn the vector representation of blocks from a sequence of block accesses. Its overall architecture is shown in Figure 2. The whole system begins with a stream of I/O traces, which will be pre-processed and formatted into training sets. These training datasets then are used to train a deep neural network to obtain the vector representation of each block. Utilizing such a vector representation, the distance between any two vectors can be calculated. This distance indicates the block correlations, which can be further used to improve storage system efficiency towards new coming I/O requests. We will describe the data pre-processing, the neural network architecture, the training algorithm, and discuss its efficiency in this section.

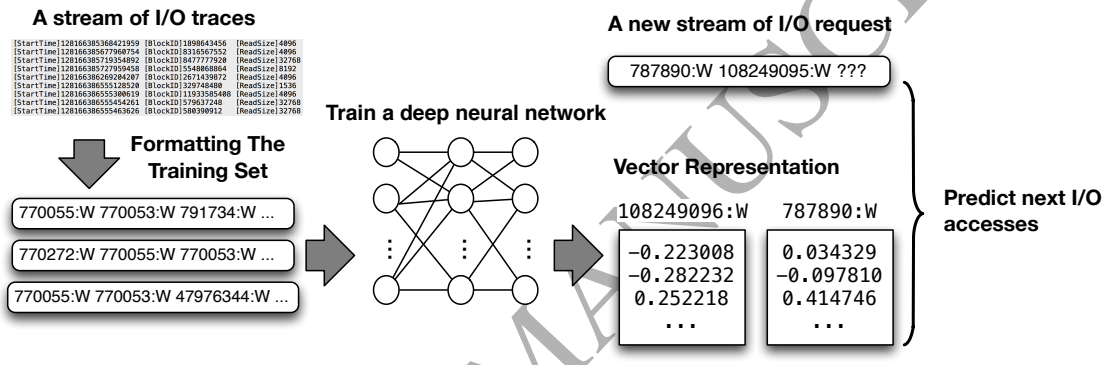

Figure 2: Overall architecture of Block2Vec.

4.1. Data Pre-Processing: Cutting Window

The first challenge of training a deep neural network for mining block correlation is the training set. The system-level block access sequence is a continuous sequence of accessed blocks without any boundary. We are not interested in the correlation of two blocks distant from each other in terms of access time. From a storage system's point of view, it is much more interesting and useful to consider block accesses that are not far apart. To address this, in *Block2Vec*, we use access distance to divide continuous block access trace into multiple closely related block accesses. Specifically, we examine how far two continuous block accesses depart by measuring the time elapsed between them. We set a maximal threshold for such distance, denoted as cutting window max_{win}. Once a block access happens later than max_{win} after the previous one, we consider a new "sequence" has started. In this way, the history block access trace is sliced into size-variant shorter sequences. Each sequence contains accesses that are temporally related.

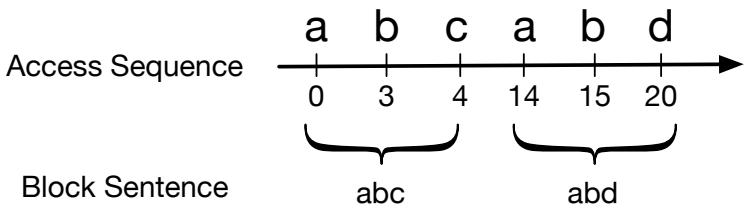

Figure 3: Splitting block sequences into multiple block sentences. The maximal cutting window is 5 time units in this example.

Fig. 3 illustrates how the access sequence *abcabd* is divided into separate block sequences with $max_{win} = 5$. Note that the time interval is calculated based on two continuous block accesses, not from the start of the sequence. For example, accessing *d* is 6 intervals after visiting *a*, but they are still in the same sequence because it is only 5 intervals later than visiting its previous *b*. In this way, it is possible that a single block sequence contains a large number of block accesses. In real-world block traces, the maximal time difference max_{win} is measured in milliseconds and its best value is based on the usage of the block correlations. For read-ahead pre-fetching, max_{win} should be relatively small to cover the closest related block accesses only. For data re-organization, the max_{win} can be relatively large to find more subtle correlations to structure the data blocks. In this study, we set $max_{win} = 1000$ *ms* except otherwise noted.

4.2. Data Pre-Processing: Block Access Translation

After splitting the single sequence of block accesses into multiple relevant sequences, the next challenge is that each block operation may access various sizes of data. Fig. 4 shows a piece of real world block trace snippets from MSR-Cambridge trace [38] (detailed description of such trace is in Section 6). Here, the 6th column shows the size of each block access in Bytes. We can see different access sizes across those items (specifically marked with red boxes): some of them are 4096 (indicating a whole 4KB block access), some are smaller (e.g., 1536 Bytes), and some are much larger (e.g., 32768 Bytes). Other block-level I/O traces have the similar property.

```
128166386533632472,hm,0,Write,11812225024,4096,2099
128166386555128520,hm,0,Read,329748480,1536,68413
128166386555194255,hm,0,Write,3163787264,28672,2679
128166386555196007,hm,0,Write,3154132992,4096,927
128166386555300619,hm,0,Read,11933585408,4096,208813
128166386555454261,hm,0,Read,579637248,32768,55170
128166386555463626,hm,0,Read,580390912,32768,45806
```

Figure 4: A piece of real-world block access trace (MSR-trace).

For large I/O operations, we use an aggregated way to translate them, in which only one block Id is included in the final sequence no matter how many bytes are read from that block. Thus, each block access is actually considered as an object access, whose data accesses may vary in size.

In addition, a system-level I/O trace contains both read and write operations as shown in Fig. 4. In *Block2Vec*, we train the neural network using both because both operations reflect the correlations of blocks, which actually come from the data stored on them. We differentiate them by appending an *operation* bit at the end of their identifications. For example, the "Write" on block `579637248` is translated to `579637248W` and the "Read" on the same block is translated to `579637248R`. This also helps future prediction. By splitting them, we are able to predict next block to be read or written, which is important for different use cases. For example, only future reads need to be pre-fetched, but both read and write are important for data re-organization.

4.3. Block2Vec Neural Network Architecture

Block2Vec leverages deep neural network to learn the vector representation of blocks. Its neural network architecture is shown in Fig. 5. Each time, it takes *k* (called the *context window*) recent block accesses as the training set. As described, each block is represented as an *n*-dimensional vector $[v_1, ..., v_n]$. All input vectors are connected to the hidden layer. The hidden layer **sums** all input vectors into a single vector. Each dimension of such vector connects to the output layer, which, similar to word2vec, is a Huffman tree. The connections are built between each dimension of the vector to each node on the Huffman tree.

The Huffman tree is built from a full scan of the entire block accesses. All nodes have different path lengths to the root of the tree depending on their occurring frequencies. While building the Huffman tree, the frequency of each block is recorded for an elimination procedure later. Infrequent blocks of fewer than min_f occurrences are ignored as correlations involving these rarely accessed blocks are considered not stable. For example, if a block is only accessed once in the whole trace, it will not be persuasive to make a conclusion that it is correlated with other

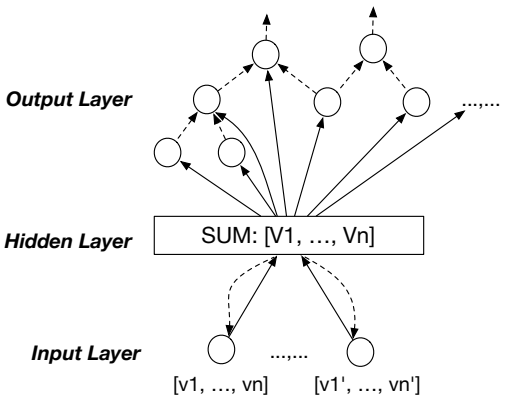

Figure 5: Neural network architecture of Block2Vec.

blocks. The blocks not filtered out are called *active blocks*. To be analyzed in Section 4.5, the number of active blocks significantly affects the computation complexity of *Block2Vec*. In this work, we use an empirical value $min_f = 5$ for a good balance between complexity and block coverage.

Once the output layer of the neural network is built and the training set has been pre-processed, *Block2Vec* will train the neural network. Vector representation of each block and all weights in the neural network are randomly initialized. Each time, a sequence of block accesses inside a context window is sent to train. According to the training model, both input and expected output are generated and fed into the network. Given the current weights in the network, an output can be calculated. If the output matches the expected one, the training will move to the next context window. If the outputs are not correct, a back-propagation will happen to adjust all weights from the hidden layer to the input layer, and most importantly, the vector values of all input blocks.

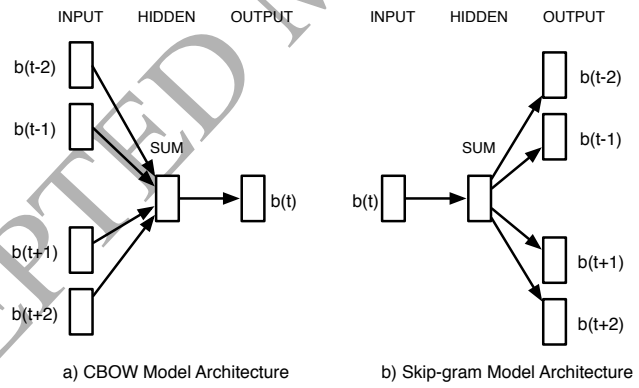

Figure 6: Two training models in Block2Vec: CBOW and Skip-gram.

Block2Vec contains two different training models, both of which have been widely used in NLP [25, 26]. Shown in Fig. 6(a), the first model called *Continuous Bag-of-Words* (CBOW) is almost identical to what we have seen in the "filling-the-blank" example earlier in Fig. 1. The training criteria are to correctly classify the current (middle) block given several future and history blocks as the input. It does not consider the order of blocks in the trace history as both future and history block accesses are summed together in the hidden layer. The second one is called *Skip-gram* model, as shown in Fig. 6(b). Instead of predicting the current block based on the context, it tries to predict the context, i.e., two preceding blocks and two succeeding blocks in this example, based on the current block. The Skip-gram model considers the order of block accesses as it considers the input as a single block and the output forms a moving window on the block accesses sequence. This is critical for our use case since the order is important in block correlations.

Also, Skip-gram is also able to obtain more accurate vector representation of blocks than CBOW simply because Skip-gram model allows more "targetted" update to the vectors of input block. Specifically, in Skip-gram model, only the vector of one block is updated by one back-propagation, while in CBOW model the change triggered by one back-propagation is distributed to the vectors of all input blocks. In the evaluation section (Section 6), we report a detailed computation time and accuracy comparison between these two models. *Block2Vec* allows users to choose either one as the training model. The default model is Skip-gram.

4.4. Time Sensitive Training

As described in the previous section, to train Block2Vec neural network, each time we choose a set of blocks in the context window to generate inputs and the expected outputs based on the chosen model (i.e., CBOW or Skip-gram) to feed the neural network. However, this does not consider the time sensitivity of block accesses. In a natural language, the time interval between two continuous words is not considered important, and they will not affect the correlation between the words. However, such interval is important for block correlations. For example, consider two block accesses: $[b_1, b_3]$ and $[b_2, b_3]$ with access times $[0, 4]$ and $[5, 6]$, respectively. The correlation between b_1 and b_3 should be considered less strong than that between b_2 and b_3 since the former has longer interval. A long interval indicates the higher possibility of irrelevant data accesses.

Existing training algorithms only consider the distance of the order of block accesses. For example, word2vec utilizes a random sampling to differentiate the closeness of words. Specifically, each time for a word s_t, instead of directly using the words within the context window k, one could have a random window size as $rand(k)$. Since the trace file will be fed multiple times to train the network, randomly sampling will give less weight to those distant words. However, such sampling strategy does not work well for block accesses since it does not reflect the time intervals between block accesses. In *Block2Vec*, we use a different way to sample the context window. Specifically, we also consider the elapsed time between block accesses in the context window. Two sets of blocks in the same order but with different time intervals will be considered differently. Fig. 7 shows how it works for CBOW model. In this example, if the current block is c, we first choose a, b, d, e as they are inside the initial time window (i.e., 8ms). Then, we check block Ids inside a smaller context window (cut in half), and find out the inputs (within 4ms). Repeating this step for another half context window (i.e., 2ms), we have b, d as the input again. In this way, Block2Vec maximally leverages the closely sequential block accesses and also amplify the impact of closeness between block accesses.

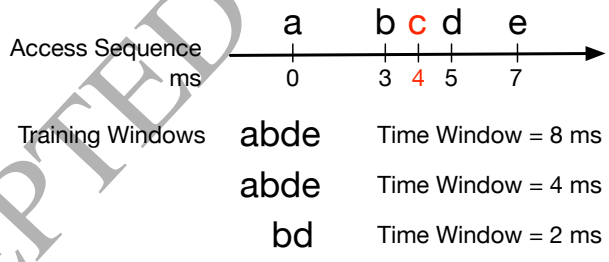

Figure 7: Time sensitive training in Block2Vec.

4.5. Efficiency of Block2Vec

Block2Vec is the first method to consider each block as a high-dimensional vector and to train a deep neural network to learn such vector representation accurately. By leveraging the recent progress in deep learning, it delivers this promise with reasonable resource requirement.

4.5.1. Memory Consumption

Block2Vec requires memory space to store the Huffman tree, weights of the neural network, and other auxiliary data structures. But, all memory consumption depends on the size of *active blocks*, which have high access frequency (larger than min_f) in the trace during the collection period. In fact, although a modern hard-disk can have billions of blocks (4TB / 4KB $\approx$ 1 Billion), the frequently accessed (active) blocks during a period are usually limited. For

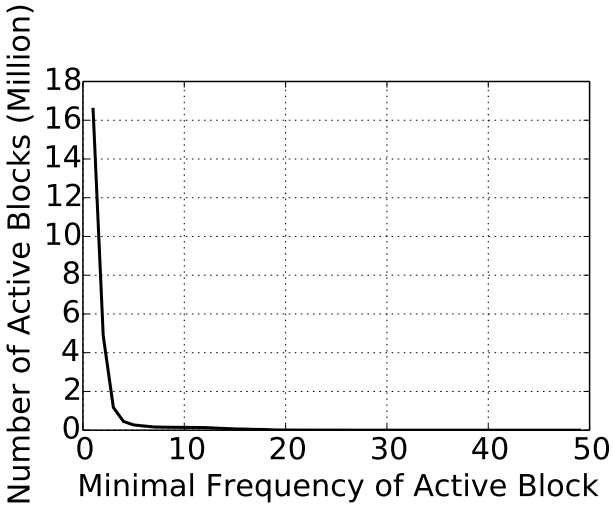

Figure 8: The number of active blocks v.s. min_f in `Project2` trace.

example, the MSR `Project2` traces [38], collected in 2007 on a active commercialized server (their project building server) for a week's time period, only access around 16 million different blocks, much smaller than the total number of blocks in a typical disk at that time (500GB/4KB $\approx$ 134 Million). Moreover, as Block2Vec only considers blocks that are frequent in the trace (frequency above the threshold min_f), it will further reduce the number of blocks and hence reduce the memory consumption. Fig. 8 shows the number of active blocks v.s. the minimal frequency (min_f) on the `Project2` trace. We can easily observe that the number of active blocks drops significantly as the minimal frequency increases. Due to these reasons, Block2Vec is more space efficient than probability graphs [18] because they do not need to maintain the information of edges between any two close blocks, which has up to $O(N^2)$ space complexity, during the graph building process.

4.5.2. Computational Complexity

The training complexity of CBOW model is $O(N \times D + D \times \log_2 V)$ [39]. Here, N is the size of the context window, D is the dimension of the vector, and V is the size of active blocks. As V is much smaller than a total number of blocks in disks as we have described, the training is reasonably fast. In fact, the training phase itself is faster than block correlation mining algorithms such as probability graph [18] or frequency mining like C-Miners [24]. The training phase of a probability graph needs to update N vertexes each time when a new block is accessed. To iterate all active blocks, the computation complexity becomes $N \times V$, which is also the computation complexity of C-Miners. The training complexity of Skip-gram model is $O(N \times (D + D \times \log_2 V)$ [39]. Note that, building the Huffman tree is not considered a part of the training phase. It costs $O(V \times \log_2 V)$ to build such a tree from scratch. But, this is not needed for each run. Also, Block2Vec is designed for off-line training, which could happen with a fixed interval, for example, one week. We will give detailed training time in the evaluation section.

5. Applications and Use Cases

5.1. Correlation-based Block Prediction

Thanks to the vector representation by Block2Vec, block correlations can be quantitatively measured by their vector distances in the high-dimensional space. Such correlation can be used to predict future data block accesses given a sequence of current block accesses. The rationale for the prediction comes from the fact that the neural network is trained in a way that similar blocks should appear in the similar context. The simple correlation-based block prediction strategy works in this way: while I/O block accesses are being issued, the prediction algorithm keeps a look-back window to keep track past block accesses. The look-back window is as wide as the context window used

to train the neural network. For each block access in the look-back window, starting from the latest to the oldest, we retrieve the nearest k blocks whose distances are closest to each block. As more distant block accesses have less impact on future accesses, we weight a factor α on the distances of older block accesses. In *Block2Vec*, we choose $\alpha = 1.1$ based on empirical analysis. All blocks are reversely ordered based on their distances and top k blocks are picked as the predictions.

Note that, we do not use the trained neural network to predict the next block. The major reason comes from the Huffman tree optimization. Each time when we train the neural network, *Block2Vec* does not update all weights from the hidden layer to all nodes in the output layer. This helps reduce the computation time. It only updates the weights between the hidden layer and the nodes that represent the predicted outputs. Weights between the hidden layer to other (incorrect) nodes of the Huffman tree are ignored and not suppressed. So, they might mislead the prediction algorithm when the same sequence occurs for prediction.

In real world, block prediction algorithms can be sophisticated. Many research efforts have been done in this space and many others are also actively going on. In this work, these sophisticated prediction algorithms are not our focus. Instead, in our implementation and evaluation, we just use the accuracy of the basic sequential/look-ahead prediction to measure different correlation mining algorithms.

5.2. Correlation-based Block Re-Organization

In this section, we introduce another possible usage for vector representations of blocks: re-organizing data blocks. We show the rational and key techniques for implementing it. However, due to its complexity and a huge number of relevant parameters, this usage is not a good candidate to evaluate the benefit of *Block2Vec* itself. Hence, in this research, we do not include this use case in the evaluation. Future work is planned to systematically evaluate this use case.

Block2Vec produces vector representations to indicate block correlations. Hence, these blocks can be clustered into different correlated groups according to their distances in high-dimensional spaces. All blocks in the same group are considered closely correlated with each other regarding the accessing order. Such a result can be used to re-organize data blocks to improve the sequential read. Since we have already assigned each block a vector value, it will be non-trivial to apply clustering algorithm like K-Means on them, as Algorithm 1 shows. Here, *thr* represents the maximal distance between blocks that are put into the same group. The key challenge is that *thr* is unknown and not easy to infer: a large *thr* will end up with missing correlations, but too small *thr* may combine non-relevant blocks together. We can use a semi-supervised training to get the proper *thr* value as our previous work used [40].

Algorithm 1 Clustering Algorithm

```
1:  procedure CLUSTER(blocks, groups, thr)
2:      for each b_i ∈ blocks do
3:          g = get_nearest_group(groups, b_i);
4:          dist = distance(g, b_i);
5:          if dist leqthr then
6:              add_to_group(g b_i);
7:          else
8:              new_group = create_group(b_i);
9:              add_to_groups(new_group, groups);
10:         end if
11:     end for
12: end procedure
```

The semi-supervised algorithm works as Algorithm 2 shows. We defined the learning factor (α) as the adjustment of *thr* each time. The adjustment can either increase or decrease the *thr* value. During training, the function will first call $cluster\_accuracy(groups, trace)$ to calculate the accuracy of current clustering based on *thr*. If a better accuracy is got (i.e., $a > ref$), then it means the previous adjustment is achieving better results. So we will repeat the adjustment again (i.e., keep_previous_adjust()). If a worse accuracy is gotten, we will reverse previous adjustments (i.e., reverse_previous_adjust()) with a smaller learn factor $\frac{\alpha}{2}$. The whole training loop will stop once α becomes too

small. The function *cluster_accuracy*(*groups*, *trace*) is to calculate the accuracy of each group by replaying the block access trace. Each time, it looks at the k context blocks for the current block (b_i) and checks whether these next blocks belong to the group of b_i. If yes, there is a *hit* increased; or else, something is missing in the correlation. In the end, the function will return the ratio of hit compared to the total checks. According to block correlations, it is expected to obtain a better accuracy if we are able to cluster blocks more accurately

Algorithm 2 Semi-supervised Training Algorithm

```
1:  procedure TRAIN(blocks, groups, thr)
2:      groups = {}; thr = rand(0, 1); ref=0;
3:      curr_adjust = incr;
4:      while true do
5:          groups = cluster(blocks, groups, thr);
6:          a = cluster_accuracy(groups, trace);
7:          if a > ref then
8:              keep_previous_adjust(thr, α);
9:          else
10:             reverse_previous_adjust(thr, α/2);
11:         end if
12:         if adjust_stop then
13:             break;
14:         end if
15:     end while
16: end procedure
17:
18: procedure CLUSTER_ACCURACY(groups, trace)
19:     var total = 0; hit = 0;
20:     for each b_i ∈ trace do
21:         bc = blocks_context(b_i, trace);
22:         for next ∈ bc do
23:             if next ∈ get_group(b_i) then
24:                 hit++;
25:             end if
26:             total += k;
27:         end for
28:     end for
29:     return hit/total;
30: end procedure
```

6. Experimental Results

Block correlation mining is an intensively explored topic. There already exists multiple research as mentioned in Section II. In this evaluation, we compare *Block2Vec* with two well-accepted methods: probability graph (PG) [17] and the sequential prediction (SP). The PG implemented is based on [18] except that the file vertex in original paper is changed to block here.

6.1. Data Trace Description

We use trace-driven simulations with several large disks traces collected in real systems. Specifically, we use the MSR Cambridge Traces [38] (MSR-Cambridge) from SNIA [41] as the test data set in this research. The MSR-Cambridge trace contains 1-week block I/O traces in enterprise servers at MSR-Cambridge. There are totally 36 I/O traces from 36 different volumes on 13 servers. We chose two of them from the servers `Project2` and `Proxy1` as the

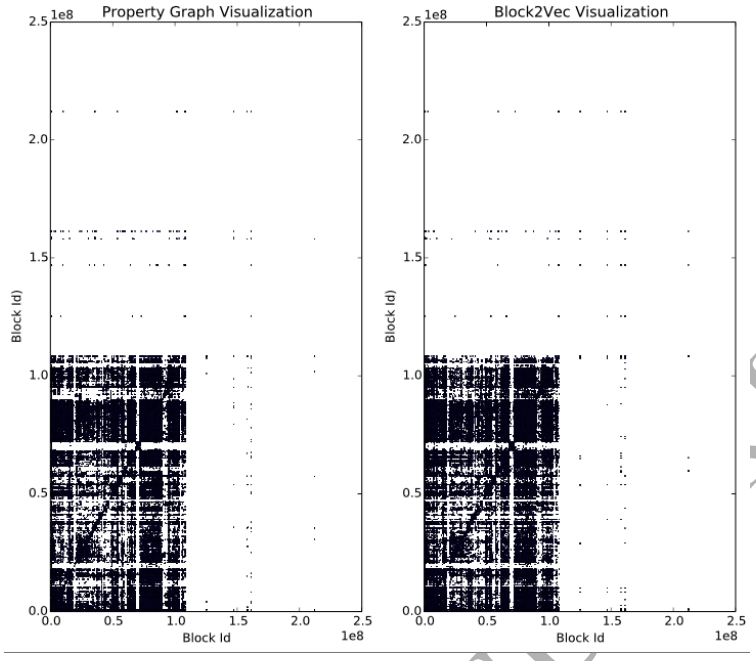

Figure 9: Block correlations visualization of PG (left) and Block2Vec (right).

datasets for all evaluations. `Project2` trace has approximately 29 million block I/O accesses, while `Proxy1` trace has more (around 133 million) block I/O accesses. Evaluations on other trace files show the similar results. Among all block operation, we select 90% of the traces as the training set and the remaining 10% as the verification set.

In MSR-Cambridge trace files, each I/O trace contains multiple fields: *timestamp* indicates the issue time of the I/O request; *type* can be either `READ` and `WRITE` indicating the I/O operation type; *offset* is the starting offset of the I/O in bytes from the start of the logical disk; *size* is the transfer size of the I/O request in bytes. All fields are shown as follow:

$$\{timestamp, hostname, disk\_id, type, offset, size, iotime\}$$

Based on this trace, we first calculate the block Id based on *offset* as:

$$blockId = \lfloor \frac{offset}{4096} \rfloor$$

All the block accesses will be ordered based on their timestamps. The timestamps used in this trace is *windows file time*, which is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 A.M. Jan. 1st, 1601 UTC.

6.2. Visualization of Block Correlations

Before the accuracy comparison of *Block2Vec* and other algorithms, we first visualize mined block correlations based on PG and Block2Vec (Skip-gram model) in Fig. 9. The baseline SP is not shown as it is just a diagonal line.

Both x-axis and y-axis are the block Ids. If two blocks are correlated, a tiny point will be plot. In this way, we show the overall detected correlations from different algorithms. From these figures, we observe: 1) the line close to diagonal, which represents the locality principle, is identified well by both algorithms; 2) although there are some differences, the overall visualization is similar for the two algorithms that blocks are grouped into different parts, which can be used to improve data organization. In evaluations below, we will quantitatively compare the accuracy of block correlations detected through block prediction.

6.3. Block2Vec Training Performance

A reasonable speculation about using deep learning techniques like *Block2Vec* into block correlation mining is their performance. In this subsection, we report the performance of training in different models, parameters, and datasets. The hardware platform for this evaluation contains a 1.7GHz Intel Core i7 CPU, an 8GB memory, and a 256GB SSD. No GPU or other accelerator is used.

Fig. 10 compares training speed of both CBOW and Skip-gram models against different vector dimension (i.e., from 30 to 100) on two traces (`Project2` and `Proxy1`). The *y*-axis shows training speed in K blocks/thread/second. As we can see, longer vector decreases the training speed. Also, the training of Skip-gram is much slower than that of CBOW for both traces.

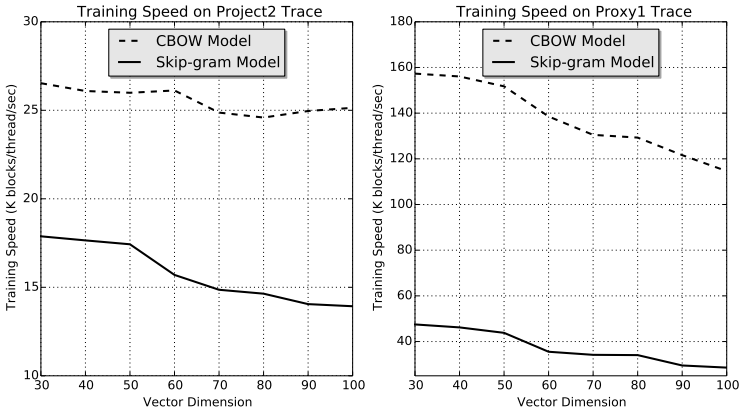

Figure 10: Block2Vec training speed v.s. different vector dimension on different training models (CBOW and Skip-gram) and datasets.

We also evaluate the training speeds of both CBOW and Skip-gram models against different context windows (i.e., from 5 to 10) on both traces. The context window indicates the number of block accesses we used to feed the neural network each time (described in Section 4.3). According to Fig. 11, longer context window indicates slower training speed. One thing worth noting is the performance difference for CBOW model is not significant as in the case of vector size. But, the difference for Skip-gram model is much larger.

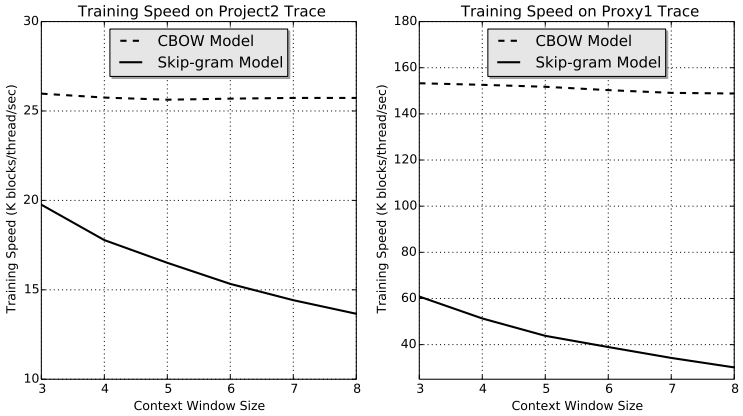

Figure 11: Block2Vec training speed v.s. different context windows on different training models (CBOW and Skip-gram) and datasets.

The training time on different datasets and training models are provided in Table 1. It is interesting to see that although Fig. 10 indicates a much higher training speed on `Proxy1` trace than on `Project2` trace. But the training

time for `Proxy1` is much longer. This is mainly because `Project2` contains 271K active blocks and 2.96 million total training sequences, while `Proxy1` contains smaller (38K) active blocks and much more (13.3 million) total training sequences. A small number of active blocks significantly boosts the training speed. But, the training time largely increases for larger training set.

Table 1: Training Time with vector size 50 and context window 5

	`Project2`	`Proxy1`
CBOW Model	120.12 s	881.95 s
Skip-gram Model	183.91 s	2862.5 s

The time here is recorded as a single iteration through the training set. We normally run multiple iterations to achieve the best accuracy. Note that both traces are collected from a week-long run of the production server, whereas the maximal training time is less than 1 hours. Hence, it is feasible to run *Block2Vec* regularly in the real system.

6.4. Block2Vec Prediction Accuracy

As mentioned earlier, larger vector size and context window will slow down the training phase. Not surprisingly, they also impact the accuracy of block correlations. We further show the accuracy of mined correlations towards those different parameters in this subsection.

To evaluate the accuracy of *Block2Vec*, we use block prediction strategy described in Section 5 and measure the percentage of correct predictions. The prediction criteria is largely simplified: each time, we predict next block access with k candidates; if the real block access belongs to the k candidates, we consider it is a *hit*, or else a *miss*. The accuracy is calculated as $Accuracy = \frac{hit}{hit+miss}$. Each time, we predict $k = 30$ blocks (the results of other possible k values are also reported in later evaluation) to testify the accuracy. In this evaluation, we use `Project2` dataset as an example. Again, the first 90% of the total data trace was used to train the model and then the remaining 10% was for prediction.

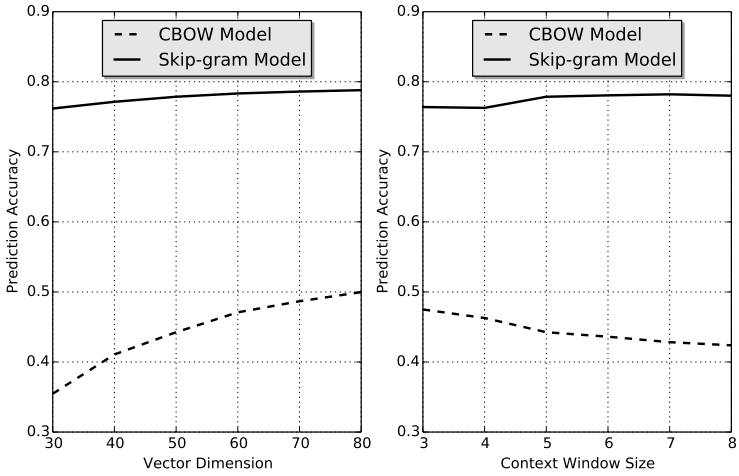

Figure 12: Prediction accuracy of two training models towards different vector dimensions and context window size on `Project2` trace.

The results are given in Fig. 12. There are several key observations. First, the Skip-gram model has a significantly higher accuracy than CBOW model, mainly because the Skip-gram model is able to capture the order information as described in Section 4.3. Second, the accuracy of the Skip-gram models increases for higher vector dimensions and larger context windows. But the change is fairly small, under 3%. Considering the increase of computation complexity shown in Fig. 10, a medium vector dimension (e.g., 50) and context window (e.g., 5) will be appropriate for real-world usage. Third, the accuracy of the CBOW model is not only limited (typically under 50%), it is also

more obviously affected by vector dimension and context window size. In addition, it is interesting to see that its accuracy decreases with larger context window. This is mainly because that the order information is ignored in the CBOW model. So that, the longer context window it chooses, the more order information is lost.

6.5. Block2Vec Comparison Evaluation

In the end, we compare Block2Vec with the widely adopted probability graph (PG) strategy and the basic sequential prediction (SP).

To predict the next block, the PG algorithm simply locates the current block in the graph and returns the top k neighbors whose edges have the largest co-occurrences. For a fair comparison, we use the same min_f value to filter out the inactive blocks to accelerate training and also improve the accuracy. The SP method does not have training phase. Each time, we just select next k consecutive blocks as the predictions based on the current visit. Similar to PG, we also exclude the inactive blocks in the accuracy evaluation.

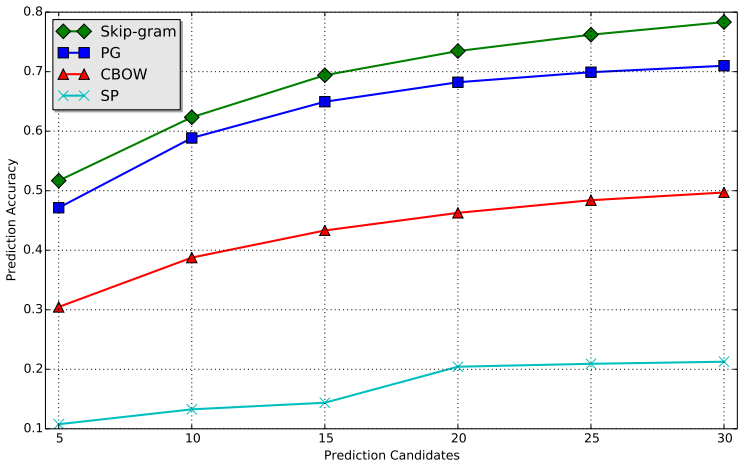

Figure 13: Training accuracy of Block2Vec (CBOW and Skip-gram), PG (probability graph), and SP (sequential prediction) based on different k.

Fig. 13 reports the prediction accuracy on the `Project2` trace based on different algorithms and k ranging from 5 to 30. The proposed *Block2Vec* with the Skip-gram model achieves the best accuracy among all algorithms, with 8% performance improvement than PG. The CBOW model is much less accurate than PG. SP has the bottom performance, indicating that most block accesses in this trace file are not sequential. For such a non-sequential trace, *Block2Vec* is able to achieve an impressive accuracy under such a simple prediction scheme.

Another point worth noting is that PG requires much longer training time than *Block2Vec* due to the time-consuming co-occurrence updates on edges. In fact, to train on `Project2` trace, which has around 2.9 million block access, the algorithm costs more than 100 minutes to finish.

7. Conclusion and Future Work

In this research, we propose to use high-dimensional vectors to represent blocks. Based on such an approach, we design and implement *Block2Vec*, a deep learning strategy to learn the best vector representations of blocks by training a deep neural network. To the best of our knowledge, this is the first time that a deep learning technique for vectorizing block objects is used in block correlation mining. We introduce and discuss the design and implementation details of *Block2Vec* and present detailed evaluations on both training cost and accuracy, with comparison to two other well-known methods, probability graph and sequential prediction. These results confirm that *Block2Vec* is a practical and accurate way to learn block correlation in storage systems. In the future, we will further apply *Block2Vec* to more use cases and also use deep learning strategy on object-based parallel file systems. We also plan to combine the vector representations and other sequence prediction algorithms like the recursive neural network to conduct more accurate block predictions.

8. Acknowledgement

This research is supported in part by the National Science Foundation under grants CCF-1409946 and CNS-1338078.

References

[1] P. Schwan, Lustre: Building a file system for 1000-node clusters, in: Proceedings of the 2003 Linux Symposium, Vol. 2003, 2003.
[2] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, 2006, pp. 307–320.
[3] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, in: ACM SIGOPS operating systems review, Vol. 37, ACM, 2003, pp. 29–43.
[4] D. Dai, X. Li, C. Wang, M. Sun, X. Zhou, Sedna: A Memory Based Key-Value Storage System for Realtime Processing in Cloud, in: CLUSTER Workshops, 2012.
[5] Z. Jiang, X. Wei, N. Jason, E. Kace, C. Yong, Suora: A scalable and uniform data distribution algorithm for heterogeneous storage systems, in: Proceedings of the 11th IEEE International Conference on Networking, Architecture, and Storage, 2016.
[6] D. Dai, R. B. Ross, P. Carns, D. Kimpe, Y. Chen, Using Property Graphs for Rich Metadata Management in HPC Systems, in: Parallel Data Storage Workshop (PDSW), 2014 9th, IEEE, 2014.
[7] Y. Lu, Y. Chen, R. Latham, Y. Zhuang, Revealing Applications' Access Pattern in Collective I/O for Cache Management, in: Proceedings of the 28th ACM international conference on Supercomputing, ACM, 2014.
[8] J. Choi, S. H. Noh, S. L. Min, Y. Cho, Towards Application/File-level Characterization of Block References: a Case for Fine-grained Buffer Management, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 28, ACM, 2000, pp. 286–295.
[9] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, C. S. Kim, A Low-overhead High-performance Unified Buffer Management Scheme that Exploits Sequential and Looping References, in: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4, USENIX Association, 2000.
[10] A. J. Smith, Sequentiality and Prefetching in Database Systems, ACM Transactions on Database Systems (TODS) 3 (3) (1978) 223–247.
[11] J. Schindler, J. L. Griffin, C. R. Lumb, G. R. Ganger, Track-Aligned Extents: Matching Access Patterns to Disk Drive Characteristics., in: FAST, Vol. 2, 2002, pp. 259–274.
[12] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, Semantically-Smart Disk Systems., in: FAST, Vol. 3, 2003, pp. 73–88.
[13] H.-T. Chou, D. J. DeWitt, An Evaluation of Buffer Management Strategies for Relational Database Systems, Algorithmica 1 (1-4) (1986) 311–336.
[14] E. V. Carrera, E. Pinheiro, R. Bianchini, Conserving Disk Energy in Network Servers, in: Proceedings of the 17th annual international conference on Supercomputing, ACM, 2003, pp. 86–97.
[15] P. J. Denning, The locality principle, Communications of the ACM 48 (7) (2005) 19–24.
[16] S. Tweedie, Ext3, journaling filesystem, in: Ottawa Linux Symposium, 2000, pp. 24–29.
[17] T. M. Kroeger, D. D. Long, The case for efficient file access pattern modeling, in: Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on, IEEE, 1999, pp. 14–19.
[18] J. Griffioen, R. Appleton, Performance Measurements of Automatic Prefetching, in: Parallel and Distributed Computing Systems, 1995, pp. 165–170.
[19] T. M. Madhyastha, D. A. Reed, Learning to Classify Parallel Input/Output Access Patterns, Parallel and Distributed Systems, IEEE Transactions on 13 (8) (2002) 802–813.
[20] T. M. Madhyastha, R. D. A, Input Output access pattern classification using hidden Markov models, Proceedings of the fifth workshop on I/O in parallel and distributed systems (1997) 57–67.
[21] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, X.-H. Sun, I/O acceleration with pattern detection, in: Proceedings of the 22nd international symposium on High-Performance Parallel and Distributed Computing, ACM, 2013, pp. 25–36.
[22] Y. Liu, R. Gunasekaran, X. Ma, S. S. Vazhkudai, Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces, in: Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14), 2014, pp. 213–228.
[23] Z. Li, Z. Chen, Y. Zhou, Mining block correlations to improve storage performance, ACM Transactions on Storage (TOS) 1 (2) (2005) 213–245.
[24] Z. Li, Z. Chen, S. M. Srinivasan, Y. Zhou, C-miner: Mining block correlations in storage systems, in: FAST, pp. 173–186.
[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed Representations of Words and Phrases and Their Compositionality, in: Advances in neural information processing systems, 2013, pp. 3111–3119.
[26] Word2Vec, https://code.google.com/archive/p/word2vec/.
[27] R. Rosenfeld, Two decades of statistical language modeling: Where do we go from here?
[28] W. B. Cavnar, J. M. Trenkle, et al., N-gram-based text categorization, Ann Arbor MI 48113 (2) (1994) 161–175.
[29] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, J.-L. Gauvain, Neural Probabilistic Language Models, in: Innovations in Machine Learning, Springer, 2006, pp. 137–186.
[30] G. P. Hughes, On the mean accuracy of statistical pattern recognizers, Information Theory, IEEE Transactions on 14 (1) (1968) 55–63.
[31] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[32] Y. Bengio, R. Ducharme, P. Vincent, A neural probabilistic language model, in: NIPS'01, 2001.
[33] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Cognitive modeling 5 (3) (1988) 1.
[34] J. Pennington, R. Socher, C. D. Manning, Glove: Global vectors for word representation, in: Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

[35] W. Y. Zou, R. Socher, D. M. Cer, C. D. Manning, Bilingual word embeddings for phrase-based machine translation., in: EMNLP, 2013, pp. 1393–1398.
[36] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078.
[37] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the conference on empirical methods in natural language processing (EMNLP), Vol. 1631, Citeseer, 2013, p. 1642.
[38] D. Narayanan, A. Donnelly, A. Rowstron, Write off-loading: Practical Power Management for Enterprise Storage, ACM Transactions on Storage (TOS) 4 (3) (2008) 10.
[39] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.
[40] D. Dai, Y. Chen, D. Kimpe, R. Ross, Provenance-based object storage prediction scheme for scientific big data applications, in: Big Data (Big Data), 2014 IEEE International Conference on, 2014, pp. 271–280. doi:10.1109/BigData.2014.7004242.
[41] S. I. Repository, http://iotta.snia.org/traces/388.