EPiC Series in Computing

Volume 64, 2019, Pages 68–77

Proceedings of 28th International Conference
on Software Engineering and Data Engineering

Vertical Data Processing for Mining Big Data: A Predicate Tree Approach

Mohammad Hossain[1], Maninder Singh[2], Sameer Abufardeh[3]
[1,3]Math Science and Technology Department, University of Minnesota Crookston, MN, USA
[2]Department of Computer Science and IT, St Cloud State University, MN, USA
[1,3]{Hossain, sabufard}@crk.umn.edu,[2] msingh@stcloudstate.edu

Abstract

Time is a critical factor in processing a very large volume of data a.k.a 'Big Data'. Many existing data mining algorithms (supervised and unsupervised) become futile because of the ubiquitous use of horizontal processing i.e. row-by-row processing of stored data. Processing time for big data is further exacerbated by its high dimensionality (# of features) and high cardinality (# of records). To address this processing-time issue, we proposed a vertical approach with predicate trees (pTree). Our approach structures data into columns of bit slices, which range from few to hundreds and are processed vertically i.e. column by column. We tested and compared our vertical approach to traditional (horizontal) approach using three basic Boolean operations namely addition, subtraction and multiplication with 10 data sizes. The length of data size ranged from half a billion bits to 5 billion bits. The results are analyzed w.r.t processing speed time and speed gain for both the approaches. The result shows that our vertical approach outperformed the traditional approach for all Boolean operations (add, subtract and multiply) across all data sizes and results in speed-gain between 24% to 96%. We concluded from our results that our approach being in data-mining ready format is best suited to apply to operations involving complex computations in big data application to achieve significant speed gain.

1 Introduction

There has been massive increase in volume of data everyday with the advancement in technology [16]. In present era, the data is collected and stored from various sources like satellite, customer checkout from super stores, stock exchange, electronic communication, social media, images/video from surveillance etc. This massively production of data has introduced the need for advance storage and processing capabilities [3-4]. This type of large (high in cardinality) and complex (high in dimensionality) data is known as "*Big Data*". High cardinality refers to situation when 100's of thousands of data records are generated/stored every unit time interval such that processing each one of

them is not possible in a given unit time interval. High dimensionality means high-dimensional data that is beyond general 3-dimensions e.g. in text processing, every unique word represents a feature and there could be even more than hundred thousand features in some texts [28].

Problem Statement: Mining such enormous data that is generated at high velocity has some serious challenges associated to it i.e. capturing, storing, searching, processing and visualization [7]. High velocity of data has also raised high cardinality and high dimensionality [5, 8]. Furthermore, mining such big data with significant accuracy adds to above challenges. There have been many approaches and studies conducted in past to address these challenges in big data but to some extent they lack in scalability [6, 8].

Motivation: These un-addressed challenges left us with a motivation to attempt to solve scalability issue in big data using vertical bit vectors (comprised of 1's and 0's) also known as predicate trees (pTree) [9]. The attempt to arrange data into vertical data format and possibility to apply logical operations over a large chunk of data at once is our motivation [3-4, 9]. Another motivation behind vertical approach is its scalability, lossless and data mining ready format that ensures very less pre-processing work [10].

Proposed Approach: Our approach with pTrees uses data that is structured into vertically bit-sliced vectors. Vertical vectors are readily available to operate over any logical operation that lay the foundation of any complex calculation. A speed gain over underlying Boolean operations ensures speed gain for complex computations that are built over them and this proposed approach is being evaluated in this study. Our focus is to analyze basic logical operations over vertical (our proposed) and horizontal (traditional) approaches for big data to report computational time difference. The term pTree and vertical bit vector is interchangeable used in this paper (explained later in the paper).

2 Background

Afore mining process, it is often assumed that the data is structured horizontally in relational table in a DB or in a data cube in data warehouse [11] i.e. into records and tuples. Scan-based data processing over horizontally structured records is known to be ineffective for mining big data because of scalability issues as they do not scale with large datasets [12].

In last decade, there has been an increased focus over vertical databases (also known as column-oriented database) management systems (VDBMSs) [13]. Column-oriented DBMSs are different from the traditional ones in a way they store and access data [13]. These databases store data in vertical format i.e. in vertical bit slices and have extensive usage in various data warehouse applications because of their better performance in terms of read I/O versus conventional DBMSs. Some major open-source and column-oriented Database Management Systems are C-Store [14], Infobright [15], InfiniDB [16], MonetDB [19] and pTree [17-18].

C-Store [14] is a collaborative research project at MIT that has now been commercially developed and is known as Vertica. It brought a lot of novel and interesting features to the column-oriented architecture such as efficient packing of objects, use of overlapping projections to store the data, query optimizer and executor based on columns. Infobright [15] is a database and warehouse system available in both commercial as well as open source to community. Infobright uses knowledge grid: - a small metadata layer instead of regular indexes. InfiniDB [16] is an efficient, multi-threaded analytic database system based on column-oriented storage architecture. InfiniDB uses an automated mix of vertical and horizontal partitioning. While the vertical partitioning allows faster processing by bringing only selected columns in the memory, a logical horizontal partitioning helps in reducing overall I/O in horizontal and vertical direction. MonetDB [19] is one of the most mature column-oriented database system. PTree [17-18] are lossless because the vertical bit-wise partitioning that is used in the pTree technology guarantees that all information is retained completely i.e. there is no loss of information in

converting horizontal data to this vertical format. pTree vertical data structures have been exploited in various domains and data mining algorithms, ranging from classification [17-18], [20], clustering [24-25], association rule mining [21-22], to outlier analysis [23] as well as other data mining algorithms. Next section discusses about pTree approach in detail.

3 pTree Approach

In big data processing, one of the biggest challenge is the timely processing of given dataset. Using pTrees (i.e. vertical bit vectors), we propose a novel approach to overcome aforesaid challenge in big data processing. Supervised and unsupervised learning approaches require mathematical formulae like Manhattan distance, Euclidean distance, sum, variance etc. to learn from training data and to make predictions over new (training or unforeseen) instances [20 and 35]. Few Boolean operations are briefed in the following section.

3.1 Boolean Algebra for pTrees

Boolean algebra for a set of two elements B = {0, 1} is generally defined with 1 being the TRUE state and 0 being the FALSE state. The three basic operations i.e. AND, OR and NOT are explained with two sets X and Y $\in$ {0, 1}.

3.2 Definitions and Proofs

The following definitions and proofs are referred in this paper. The Lemma's discussed in this section lays the foundation of working with pTrees.

Definition 1 (2's Complement): In binary number system, 2's complement is used to compute negative equivalent of an integer. Assume a binary number N of n bits, 2's complement of N is defined as: 2^n–N. More details can be found at [26].

Definition 2 (1's Complement): Assume a binary number N of n bits then 1's complement of N is defined as: 2^n –N-1

Definition 3 (purity): The purity in pTrees refer to presence of either all 1's or all 0's in a vertical bit-vector i.e. if a vertical bit vector consists of all 1's or all 0's then it is pure-1 or pure-0 bit-vector respectively. Purity provides speedy lookup.

Definition 4 (Level-0 pTree): In the process of pTree formation, the vertical bit-vector that is created by either 1 or 0 is known as level-0 pTree. Number of bits (0 or 1) in a level-0 pTree is known as the length of pTree. For example in Figure 1.b, the vertical bit-vector 11110100 forms level-0 of pTree.

Definition 5 (Multi-Level pTree): The vertical bit-vector at level-0 is tested for pure-1/pure-0 to form subsequent levels of pTrees to construct a multi-level pTree. The fixed number of bits to check for purity is named as '*stride*' of pTree. In Figure 1.b, the stride length is 2 bits and root is at level-3.

Lemma 1. For any binary number N, 2's complement of N = (1's Complement of N) +1
Proof: Assume N has n bits. So according to the definition-1, 2's complement of N = 2^n–N => 2^n − N − 1 + 1 = 1's complement of N + 1 [using definition-2]
Lemma 2. Complement of the complement of a number gives the original number
Proof: Assume a binary number N of n bits. Assume 2's complement of N is M which will also be n-bit number. So according to definition 1, M = 2^n–N. Now, 2's complement of M = 2^n −M = 2^n − (2^n − N) = N, hence proved.

3.3 Formation of Vertical Bit Vector

In our proposed approach, vertical bit-vector (level-0 of multi-level pTree) records the truth value of a given predicate for a data set. The value of predicate is assigned 1 if the predicate condition is satisfied. For example in Figure 1, a data set that captures temperature of a place for 8 days (N=8); if a condition (is temperature freezing i.e. below 32-degree Fahrenheit?) for predicate '*Temp*' is chosen then all the values in *temp* column consists of either 1 (if temp is less than 32°F) or 0 (if temp is greater than 32°F). The predicate P (if the temperature is freezing) is true (i.e. 1) for 5 records while is false (i.e. 0) for 3 days. The vertical bit-vector obtained by using predicate P is 11110100 (see Figure 1.a).

Conversion of bit-vector to tree: The bit vector obtained from Figure 1.a (11110100) has a length of N bits (i.e. 8). The vertical bit-vector is processed with stride length of two bits to form multi-levels in the form of a tree based on stride purity (Definition 3-5). The grouping of bits is needed to compress bit vectors to make them be able to process faster (Figure 1.b).

Formation of pTrees: In Figure 1.b, we used stride length of two-bits to check purity (explained in III.B). With stride length of 2 bits and N total number of bits, the height of multi-level pTree can be at most $\log_2 N$. In Figure 1.b, there are 3 levels and the top level represents the root and leaf represents level-0 of pTree. If pure-1 or pure-0 is encountered from level-0 and up then final pTree is formed by pruning child branches of pTrees that contained pure bits (Figure 1.b). For example, in Figure 1.c, left branch of final pTree has only one bit (i.e. 1) and it signifies that all children nodes of this 1 are also pure-1. Similarly, there is one pure-0 in final pTree.

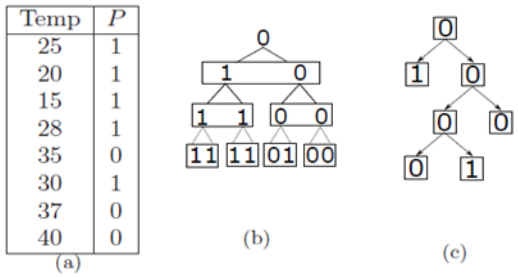

Figure 1: Construction of pTree: (a) Predicate for temperature (b) Conversion to tree (c) Final pTree

3.4 Formation of Vertical Bit Vectors from Data Set

Conversion of a relational table of horizontal records to a set of vertical bit-vectors (pTree at level-0) is created by decomposing each attribute in the table into separate bit slices. If an attribute has numeric value, we convert the data into binary (Figure 2). Each bit position of binary value generates one-bit slice but if an attribute has categorical value then a bitmap for each category is created followed by generation of bit vectors for each category. Such vertical partitioning guarantees that the information is not lost.

Next, consider a data set D with n-attributes such as $D = (A_1, A_2, \ldots, A_n)$. In order to represent this data set into pTrees we require *n*-pTrees sets as $\{P_1, P_2, \ldots, P_n\}$ such that attribute A_i will be represented by pTree set P_i. Suppose each value of A_i is presented by an N-bit binary number $a_{i,N-1}, a_{i,N-2} \ldots, a_{i,j}, \ldots, a_{i,0}$; then pTree $P_i[j]$ will represent the bit slice of $a_{i,j}$. $P_i[0]$ pTree represent the least significant bit slice of A_i and is called the Least Significant PTree (LSP) of P_i. Similarly, $P_i[N-1]$ is known as the Most Significant Ptree (MSP). Figure 2 shows the representation of data set into pTree. In Figure 2, we see that the dataset had three attributes. So, we needed three pTree sets to represent them and each pTree set required 3-bits to convert them to binary.

A_1	A_2	A_3
5	3	4
3	1	6
2	5	2
6	7	0
7	4	5
4	5	3
6	2	6
4	1	2

(a)

P_1			P_2			P_3		
$P_1[2]$	$P_1[1]$	$P_1[0]$	$P_2[2]$	$P_2[1]$	$P_2[0]$	$P_3[2]$	$P_3[1]$	$P_3[0]$
1	0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	0	1	0	1	0	1	0
1	1	0	1	1	1	0	0	0
1	1	1	1	0	0	1	0	1
1	0	0	1	0	1	0	1	1
1	1	0	0	1	0	1	1	0
1	0	0	0	0	1	0	1	0

(b)

Figure 2: pTree representation of Data Set: (a) Data Set of 3 attributes (b) Corresponding pTree Sets

3.5 Operations on Vertical Approach

Any Boolean operation discussed in section III.B can be applied on vertical bit vector. Suppose p and q are two level-0 vectors of length L and assume a binary operator OP_b that is applied on these two vectors. Then p OP_b q is calculated as p[i] OP_b q[i] $\forall i|i \ni (0 : L - 1)$ where p[i] and q[i] are the i^{th} bit of p and q vector. Similarly, if OP_u is a unary operator then OP_u(p) is calculated as OP_u(p[i]). So, the binary operations AND, OR, XOR and XNOR as well as unary operation NOT can be applied to vertical bit-vector (level-0 pTrees).

Various operations including sum, max, min, median/rank and top-k have been successfully implemented with pTrees in prior studies [25]. Vertical representation allows very fast execution for some mathematical expressions. For example, consider vertical bit-vector (11110100) from Figure 1.a to count the number of 1's. The resulting pTree from Figure 1.c provides faster counting by traversing through different branches of tree, we can calculate count i.e. 1 in left leaf of pTree is at level-2 (level-0 is leaf nodes) that accounts for 2^2 =4 counts of 1 and similarly, there is another 1 at level-0 that accounts for 2^0=1 count of occurrence of 1. So, a simple addition of all the values (4+1=5) gives the total count of 1 in predicate tree (pTree) instead of complete traversal for each record. It is worth to mention here that pTree formation is one-time process.

4 Experiments Results and Discussion

This section presents theoretical proofs, advantages, result and its discussion of pTree approach. The time comparison for various Boolean operations is presented in Figure 3 for horizontal versus vertical approach. X-axis in Figure 3 represents bit-width and Y-axis represents execution time of Boolean operation. The analysis and comparison between horizontal and vertical approach for remaining data set sizes is shown in Table 1. Boolean operations namely Addition, Subtraction and Multiplication are labelled with acronym A, S and M (these acronyms were chosen to best utilize available table space) in Table 1. The term 'K' in table means 1000 i.e. 12000 millisecond is written as 12K. Horizontal approach is analyzed over 4 bit-widths (8 bit, 16 bit, 24 bit and 32 bit) because of negligible difference in execution time across 4 to 8, 8 to 16 bit-widths and so on.

4.1 Performance analysis of speed between vertical and horizontal approaches

As mentioned before, the term '*pTrees*' referred in this section is vertical bit-vector at level-0 of pTree (Figure 1.b). Vertical approach (using pTrees) and trivial horizontal approach is compared over time taken (in milliseconds) to execute various Boolean operations using variable data-size (0.5 billion to 5 billion bits) as well as variable bit-widths (4 bits to 32 bits) in Figure 3. Due to space restriction, graphical representation of results has been shown (Figure 3) for two data sizes (0.5 and 5 billion bits)

across variable bit-width (4 to 32 bit-width with step width of 4 bits) over vertical and horizontal approach.

The performance of pTree approach over addition and subtraction operations outperform horizontal approach for each bit-width and each dataset size (refer Figure 3). In multiplication, the pTree performance is better for smaller bit-width as compared to horizontal approach under similar experimental conditions. The execution time for horizontal approach across variable bit-width remained almost similar (Figure 3) whereas it showed linear increase for pTree approach across variable bit-width. The complete performance results obtained for each data set size have been listed in Table 1. However, for multiplication operation, pTree approach couldn't outperform horizontal approach beyond 12 bit-width (refer Table 1 for M column) because during pTree multiplication with large number of bits, the number of internal operations at register level increases exponentially (proof is beyond the scope of this paper).

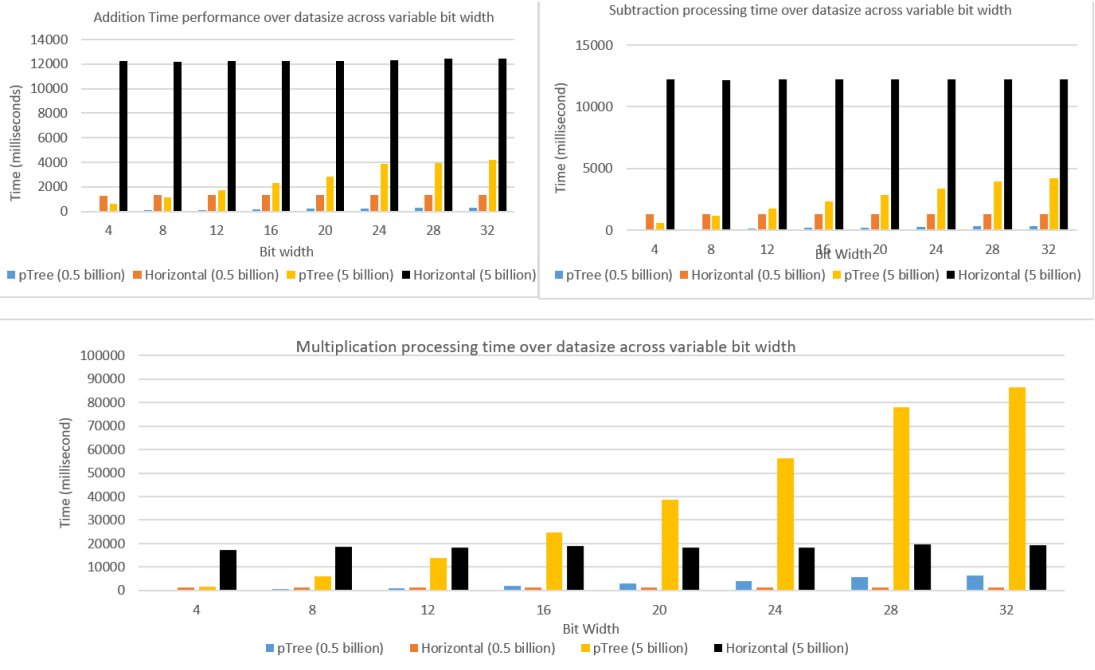

Figure 3: Results of various Boolean operations

4.2 Analysis of Space and Speed Gain by vertical approach over horizontal approach

This research question focuses on analysis of speed gain and space consumption by vertical approach. The experimental analysis of speed gain through execution of three Boolean operations is shown in Table 2. Three basic Boolean operations have been executed over 10 different data sets and for 8 different bit-widths. The resulting speed gain using the by pTree approach has been shown in Table 2 below. The speed gain was constant across variable data size but varied over bit-width. So, Table 2 present results for variable bit-width only. It is seen that the speed gain decreases as the number of bit-width increases but pTree approach still outperforms horizontal approach. Speed gain is highest (>90%) for all Boolean operations for data sets that have bit-width of 4. It is also seen that for Boolean operations Add and Subtract the speed gain range between 66% (for 32 bit-width data) to 95% (for 4 bit-width data). The discussion on space and speed gain are discussed in the following sections.

Table 1: The execution time of pTree and Horizontal approach in milliseconds for different data sets and bit-widths

	4 bit width			8 bit width			12 bit width			16 bit width		
Data	pTree			pTree			pTree			pTree		
Size	A	S	M	A	S	M	A	S	M	A	S	M
0.5	44	48	117	85	90	462	125	129	1K	167	173	1.8K
1.0	110	109	250	197	208	978	298	304	2.2K	386	377	4K
1.5	181	180	417	337	347	1.6K	508	499	3.5K	663	661	6.5K
2.0	245	244	561	486	475	2.1K	682	682	4.9K	1K	987	9.3K
2.5	311	328	699	598	647	2.8K	876	880	6.4K	1.2K	1.2K	12K
3.0	364	358	844	696	702	3.5K	1K	1K	7.9K	1.4K	1.4K	14.6K
3.5	425	420	1K	822	805	4K	1.2K	1.2K	9.3K	1.6K	1.6K	17.2K
4.0	500	486	1.1K	960	963	4.8K	1.4K	1.4K	11K	1.9K	1.9K	19.6K
4.5	554	539	1.3K	1.1K	1.1K	5.3K	1.6K	1.5K	12.2K	2.1K	2.1K	22.4K
5.0	607	597	1.5K	1.2K	1.1K	6K	1.7K	1.7K	14K	2.3K	2.3K	24.6K
	20 bit width			24 bit width			28 bit width			32 bit width		
Data	pTree			pTree			pTree			pTree		
Size	A	S	M	A	S	M	A	S	M	A	S	M
0.5	207	212	2.9K	247	254	4.2K	286	295	5.6K	309	313	6.5K
1.0	464	465	6.1K	564	575	9K	647	654	12K	676	708	14K
1.5	843	820	10.7K	971	972	15.3K	1.2K	1.2K	21K	1.2K	1.2K	25K
2.0	1.1K	1.1K	15.2K	1.4K	1.4K	21.4K	1.6K	1.6K	29K	1.7K	1.7K	35K
2.5	1.4K	1.5K	19.5K	1.7K	1.7K	27.3K	2K	2.1K	37K	2.1K	2.2K	44K
3.0	1.7K	1.7K	23K	2K	2K	33.4K	2.3K	2.3K	47K	2.6K	2.5K	52K
3.5	2K	2K	27.5K	2.4K	2.3K	38.8K	2.8K	2.8K	53K	3K	3K	62K
4.0	2.3K	2.3K	31.7K	2.8K	2.8K	45.3K	3.2K	3.2K	60K	3.4K	3.6K	71K
4.5	2.5K	2.6K	35.5K	3.2K	3.2K	49.6K	3.6K	3.7K	68K	3.8K	3.9K	80K
5.0	2.8K	2.8K	38.7K	3.9K	3.4K	56.3K	3.9K	4K	78K	4.2K	4.2K	87K
	8 bit width			16 bit width			24 bit width			32 bit width		
Data	Horizontal			Horizontal			Horizontal			Horizontal		
Size	A	S	M	A	S	M	A	S	M	A	S	M
0.5	1.3K	1.3K	1.4K	1.3K	1.3K	1.4K	1.3K	1.3K	1.4K	1.3K	1.3K	1.46K
1.0	2.5K	2.5K	3K	2.5K	2.5K	3K	2.5K	2.5K	3K	2.5K	2.5K	3.2K
1.5	3.7K	3.7K	4.9K	3.7K	3.7K	5K	3.7K	3.7K	5K	3.7K	3.7K	5.6K
2.0	4.9K	4.9K	6.6K	4.9K	4.9K	7.2K	4.9K	4.9K	7K	4.9K	4.9K	7.7K
2.5	6.1K	6.1K	8.6K	6.1K	6.1K	9.3K	6.1K	6.1K	9.2K	6.1K	6.2K	9.8K
3.0	7.3K	7.3K	10.5K	7.3K	7.3K	11.2K	7.3K	7.3K	11K	7.3K	7.4K	11.6K
3.5	8.5K	8.5K	12.4K	8.5K	8.5K	13.3K	8.6K	8.5K	12.7K	8.6K	8.6K	13.8K
4.0	9.7K	9.7K	14.7K	9.8K	9.7K	15.1K	9.8K	9.7K	14.9K	9.8K	9.8K	16K
4.5	11K	11K	16.5K	11K	11K	17.4K	11K	11K	16.8K	11K	11K	17.8K
5.0	12K	12K	18.7K	12K	12K	19.1K	12K	12K	18.9K	12K	12K	19.4K

4.3 Space Advantages of pTrees

Assume a dataset *S* consisting of *N* rows and *n* columns containing value of *m* bits. So, if we convert the dataset into pTrees we will get *mn* pTrees where the length of each tree will be *N* bits. In traditional approach, let the size of the dataset be S_{trad}. The size of each value will be = $\left\lceil \frac{m}{8} \right\rceil$ bytes and size of each column = $\left\lceil \frac{m}{8} \right\rceil N$ bytes that gives us the size of the whole dataset, S_{trad}= $\left\lceil \frac{m}{8} \right\rceil nN$ bytes.

In pTree approach, let the size of the dataset be S_{pTree}. Size of each pTree will be $\left\lceil \frac{N}{8} \right\rceil$bytes. So the size of whole dataset, $S_{pTree} = mn \left\lceil \frac{N}{8} \right\rceil$ bytes. In a large dataset where $N >> 8$, we can assume $N = 8 \times L$. We get $S_{trad} = 8 \left\lceil \frac{m}{8} \right\rceil nL$ bytes and $S_{pTree} = mnL$ bytes. Now if $1 < m < 8$ then $\left\lceil \frac{m}{8} \right\rceil = 1$. Therefore, we get $S_{pTree} = \frac{m}{8} S_{trad}$. However, when m = 8 we get $S_{pTree} = S_{trad}$ Now if $9 < m < 16$ then $\left\lceil \frac{m}{8} \right\rceil = 2$ and we get $S_{pTree} = \frac{m}{16} S_{trad}$. And when $m = 16$ we get $S_{pTree} = S_{trad}$. So, we conclude that $S_{pTree} < S_{trad}$ and $S_{pTree} = S_{trad}$ if m is a multiple of 8.

Table 2: Speed Gain by pTree approach

Bit Width	Add	Subtract	Multiply
4	95%	95%	91%
8	90%	91%	68%
12	86%	86%	24%
16	81%	81%	-29%
20	77%	77%	-113%
24	71%	71%	< -113%
28	68%	68%	< -113%
32	66%	66%	< -113%

4.4 Speed Advantages of pTrees

Assume that a logical operation (AND, OR, NOT, XOR) between two machine words (of size W) of memory takes T_{log} units of time. When we do such logical operations on two pTrees of length N we do it on L pairs of machine words where $L = \left\lceil \frac{N}{W} \right\rceil$. So, the logical operation on two pTrees takes LT_{log} unit of times. Assume an arithmetic operation (addition, subtraction, multiplication etc.) between two bytes of memory takes T_{arith} units of time. Suppose we will do such an arithmetic operation on two columns of our previously described data set, S. For simplicity assume each value in the dataset takes 1 byte of memory. So, each column has N bytes memory and the arithmetic operation will take $T_{trad} = NT_{arith}$ units of time. Suppose to get the same arithmetic operation using pTree we need to perform g number of logical operation on different pTrees. So, the process will take $T_{pTree} = gLT_{log}$ unit of time. For simplicity consider that N is multiple of W, so we can write

$$\frac{\text{TpTree}}{Ttrad} = \frac{g}{W} \frac{Tlog}{Tarith}$$

Let's call the factor $\frac{\text{Tlog}}{Tarith} = \alpha$. In old processors bit-wise logical operation would run faster than arithmetic operation like addition, multiplication, etc. resulting the factor α less than 1. However, in modern day processors the arithmetic operations are optimized in such a level that they run as fast as bit-wise operations resulting the factor α close to 1 [27]. Again, W represent the number of bits in a machine word, which the computers can handle at a time, is expected to be a large number comparing with g. As a result, if $g < W$ the T_{pTree} will be less than T_{trad} giving the speed gain of pTree based algorithm processing over traditional processing of the same algorithm.

5 Conclusion

From results and discussion, it can be concluded that the vertical approach outperformed the horizontal approach over data sets ranging between half-billion bits to 5 billion bits. The speed gain obtained with the vertical approach over the horizontal approach varies between 66% and 95% across variable bit-widths and is obtained for Boolean operation Add and Subtract. Data sets with 4 bit-width showed the highest speed gain (95%) and with 32 bit-width showed comparatively lower speed gain (66%). A positive speed gain (24%) is shown for multiplication operation up to bit-width of 12 and showed negative speed gain beyond 12 bits. We concluded from the discussion in section 4.4 that our

approach is able to process vertically bit vectors faster is most suited to address big data processing issues. Our approach showed that the vertical approach is faster than the horizontal approach over basic Boolean operations that are the basis of very complex calculations like Manhattan distance, Euclidean distance, sum, variance, etc. Using the vertical approach in complex calculations can further improve processing speed.

Reference

[1] Edd Dumbill. Making sense of big data. Big Data, 1(1):1–2, 2013.

[2] Chris Eaton, Dirk Deroos, Tom Deutsch, George Lapis, and Paul Zikopoulos. Under-standing big data, April 2012. URL:

http://public.dhe.ibm.com/common/ssi/ecm/en/iml14296usen/IML14296USEN.pdf

[3] Mohammad K Hossain, and Sameer Abufardeh. "A New Method of Calculating Squared Euclidean Distance (SED) Using pTree Technology and Its Performance Analysis." Proceedings of 34th International Conference on Computers and Their Applications (CATA-2019), Honolulu, Hawaii, USA, March 2019: 45-54.

[4] Mohammad K Hossain, Arjun Roy, Arijit Chatterjee, and William Perrizo. Algorithms to calculate the manhattan (l1) distance for vertical data represented in ptrees. In Proceedings of the 2012 ISCA 27th International Conference on Computers and Their Applications (CATA-2012), CATA-2012, Las Vegas, Nevada, USA, March 2012.

[5] S. Sagiroglu and D. Sinanc, "Big data: A review," *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, San Diego, CA, 2013, pp. 42-47.

doi: 10.1109/CTS.2013.6567202

[6] A. Katal, M. Wazid and R. H. Goudar, "Big data: Issues, challenges, tools and Good practices," *Contemporary Computing (IC3), 2013 Sixth International Conference on*, Noida, 2013, pp. 404-409. doi: 10.1109/IC3.2013.6612229

[7] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing as a service and big data. arXiv preprint arXiv:1301.0159, 2013.

[8] Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. 2013. Data warehousing and OLAP over big data: current challenges and future research directions. In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP* (DOLAP '13). ACM, New York, NY, USA, 67-70. DOI=http://dx.doi.org/10.1145/2513190.2517828

[9] Mohammad K Hossain and William Perrizo. Algorithm for shifting images stored in peano mask trees. International Journal of Electrical, Electronics and Computer Systems (IJEECS), 1(1):2221–7258, March 2011.

[10] Mohammad Kabir Hossain, Rajibul Alam, Abu Ahmed Sayeem Reaz, and William Perrizo. Bayesian classification for spatial data using p-tree. In Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International, pages 321–327. IEEE, 2004.

[11] J. Han, M. Kamber, and J. Pei. Data Mining: Concepts and Techniques. Morgan Kaufmann series in data management systems. Elsevier, 2012. ISBN 9789380931913.

[12] Arjun Roy, Mohammad K Hossain, Arijit Chatterjee, and William Perrizo. Column-oriented database systems : A comparison study. In Proceedings at the ISCA 27th International Conference on Computers and Their Applications, CATA-2012, Las Vegas, Nevada, USA, March 2012.

[13] Abadi DJ, Madden SR, Hachem N. Column-stores vs. row-stores: How different are they really?. InProceedings of the 2008 ACM SIGMOD international conference on Management of data 2008 Jun 9 (pp. 967-980). ACM.

[14] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and Stan Zdonik. C-store: A column oriented dbms. VLDB, pages 553–564, 2005.

[15] D. Slezak, J. Wroblewski, V. Eastwood, and P. Synak. Brighthouse: An analytic data warehouse for ad-hoc queries. VLDB, pages 1337–1345, 2008.

[16] Calpont infinidb concepts guide. URL http://www.calpont.com/phocadownload/documentation/2.0.0/CalpontInfiniDBConceptsGuide 20-1.pdf

[17] T. Abidin and William Perrizo. Smart-tv: A fast and scalable nearest neighbor based classifier for data mining. In Proceedings of the 21st Association of Computing Machinery Symposium on Applied Computing, SAC-06, Dijon, France, April 23-27 2006.

[18] T. Abidin, A. Dong, H. Li, and William Perrizo. Efficient image classification on vertically decomposed data. In IEEE International Conference on Multimedia Databases and Data Management, MDDM-06, Atlanta, Georgia, April 8 2006.

[19] P. Boncz and M. Kersten. Monet: An impressionist sketch of an advanced database system. In Basque International Workshop on Information Technology, San Sebastian, Spain, July 1995.

[20] M. Khan, Q. Ding, and William Perrizo. K-nearest neighbor classification on spatial data stream using ptrees. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 02, pages 517–528, Taipei, Taiwan, May 2002.

[21] I. Rahal, D. Ren, and William Perrizo. A scalable vertical model for mining association rules. Journal of Information and Knowledge Management (JIKM), 3(4):317–329, 2004.

[22] I. Rahal, M. Serazi, A. Perera, Q. Ding, F. Pan, D. Ren, W. Wu, and William Perrizo. Datamime. In Association of Computing Machinery, Management of Data, ACM SIGMOD 04, Paris, France, June 2004.

[23] D. Ren, B. Wang, and William Perrizo. Rdf: A density-based outlier detection method using vertical data representation. In Proceedings of the 4th IEEE International Conference on Data Mining, ICDM-04, pages 503–506, November 2004.

[24] E. Wang, I. Rahal, and William Perrizo. Davyd: an iterative density-based approach for clusters with varying densities. International Journal of Computers and Their Applications (IJCTA), 17(1):1–14, 2010.

[25] A. Perera, T. Abidin, M. Serazi, G. Hamer, and William Perrizo. Vertical set squared distance based clustering without prior knowledge of k. In International Conference on Intelligent and Adaptive Systems and Software Engineering, IASSE-05, pages 72–77, Toronto, Canada, July 2005.

[26] M.M. Mano. Digital Design. Prentice Hall international editions. Prentice Hall, 2002. ISBN 9781888325171.

[27] Kim, Joo-Young, and Hoi-Jun Yoo. "Bitwise competition logic for compact digital comparator." 2007 IEEE Asian Solid-State Circuits Conference. IEEE, 2007.

[28] Domingos. Pedro, "A few useful things to know about machine learning." Communications of the ACM 55.10 (2012):78-87.

[29] Weinberger, Kilian Q., John Blitzer, and Lawrence K. Saul. "Distance metric learning for large margin nearest neighbor classification." Advances in neural information processing systems. 2006.

[30] Xing, Eric P., et al. "Distance metric learning with application to clustering with side-information." Advances in neural information processing systems. 2003.

[31] Weinberger, Kilian Q., and Lawrence K. Saul. "Distance metric learning for large margin nearest neighbor classification." Journal of Machine Learning Research 10.Feb (2009): 207-244.