ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2016, Vol. 37, No. 3, pp. 255–260. © Pleiades Publishing, Ltd., 2016.

Join Decomposition Based on Fragmented Column Indices

E. Ivanova* **and L. B. Sokolinsky****

(Submitted by A. V. Lapin)

South Ural State University (National Research University),
prospekt Lenina 76, Chelyabinsk, 454080 Russia

Received March 16, 2015

Abstract—The paper is devoted to the issue of decomposition of the join relational operator with the aid of distributed column indices. Such decomposition allows one to utilize the modern many-core accelerators (GPU or Intel Xeon Phi) to speed up the query execution for very large databases. Column indices are the new kind of index structures, which exploits "key-value" technics. The paper describes the methods of column index fragmentation based on domain intervals. This technic allows organizing the parallel query processing without exchanges. All column index fragments are stored in main memory in compressed form to conserve space. This approach can be implemented as a coprocessor for relational database systems. The database coprocessor is able to perform resource-intensive operations much more faster than a conventional DBMS.

DOI: 10.1134/S1995080216030136

Keywords and phrases: *Very large databases, parallel query processing, column indices, domain-interval fragmentation.*

1. INTRODUCTION

Nowadays, human scientific and practical activities create the new challenges that demand big data processing. According to IDC study [1], the amount of digital data is doubling in size every two years, and by 2020 the digital universe—the amount of digital data created and replicated—will reach 44 zettabytes, or 44 trillion gigabytes. In 2013, only 22% of the information in the digital universe would be a candidate for analysis (useful if it were tagged) and less than 5% of that was actually analyzed. By 2020, the useful percentage could grow to more than 35%, mostly because of the growth of data from embedded systems.

Actually, the only way to process efficiently big data is using the parallel database system, which are able to process data in parallel on the high performance system with distributed memory [2–4]. The traditional approach for database storing is row-oriented representation. However, column-oriented database systems have been shown to perform more than an order of magnitude better than row-oriented database systems ("row-stores") on analytical workloads such as those found in data warehouses, decision support, and business intelligence applications. The elevator pitch behind this performance difference is straightforward: column-stores are more I/O efficient for read-only queries since they only have to read from disk (or from memory) those attributes accessed by a query [5]. Column-oriented databases are particularly well suited for compression because data of the same type is stored in consecutive sections. This makes it possible to use compression algorithms specifically tailored to patterns that are typical for the data type [6].

In recent years, more and more many-core processors are superseding sequential ones. Increasing parallelism, rather than increasing clock rate, has become the primary engine of processor performance growth, and this trend is likely to continue. Particularly, today's GPUs (Graphic Processing Units) and Intel's MIC (Many Integrated Cores), greatly outperforming traditional CPUs in arithmetic throughput

*E-mail: Elena.Ivanova@susu.ru
**E-mail: Leonid.Sokolinsky@susu.ru

and memory bandwidth, can use hundreds of parallel processor cores to execute tens of thousands of threads [7]. Recent trends in new hardware and architectures have gained considerable attention in the database community. Processing units such as GPU or MIC provide advanced capabilities for massively parallel computation. Database processing can take advantage of such units not only by exploiting this parallelism, e.g., in query operators (either as task or data parallelism), but also by offloading computation from the Central Processing Unit (CPU) to these coprocessors, saving CPU time for other tasks [8]. The many integrated cores of the Xeon Phi make this hardware accelerator a natural computing platform for an in-memory database engine or server. The database tables reside in the memory space of the MIC thus supporting fast in-memory database applications [9]. Main memory as the primary storage location is becoming increasingly attractive as a result of the decreasing cost/size ratio [6]. Main Memory Database (MMDB) eliminates disk access by storing and manipulating entire database in main memory. For performance-significant systems MMDB offer very low response time and very high throughput [10]. According to Gartner's 2013 Hype Cycle for Emerging Technologies report, in-memory database management system have 2 to 5 years until widespread adoption [11].

According to this, the problem of developing new efficient methods of parallel database processing in main memory on modern compute clusters with manycore accelerators using column-oriented representation and data compression is important. To meet this goal, we offer a special type of index structures called *distributed column indices*. Distributed column indices allow to perform a decomposition of relational operations, which admits the efficient parallel execution of them on computing cluster system, equipped with manycore accelerators. In this paper, we consider the decomposition for natural join. We will be using the notation from [12] for denoting the relation operators. The symbol $\circ$ we will use to denote the operation of concatenation of the tuples.

The paper is organized as follows. Section 2 introduces the notion of column index. Section 3 presents the definition of domain-interval fragmentation of column index. Section 4 describes a method of decomposing natural join of two relations based on domain-interval fragmentation of column indices built for the join attributes. The theorem about correctness of this decomposition method is proved. Section 5 summarises the results and outlines directions for future research.

2. COLUMN INDEX

Let $R\,(A^*, B_1, \ldots, B_u)$ be the R relation with A primary key and the following attributes: $B_1, \ldots, B_u$. Tuples of R have length of $u+1$ and form of $(a, b_1, \ldots, b_u)$, where $a \in \mathbb{Z}_{\geq 0}$ and $\forall j \in \{1, \ldots, u\}\left(b_j \in \mathfrak{D}_{B_j}\right)$. Here, $\mathfrak{D}_{B_j}$ is the domain of attribute B_j. Let $r.B_j$ denote a value of attribute B_j. Let $r.A$ denote a value of the primary key of tuple r: $r = (r.A, r.B_1, \ldots, r.B_u)$. The *primary key* of relation R has the property: $\forall r', r'' \in R\,(r' \neq r'' \Leftrightarrow r'.A \neq r''.A)$. Define *tuple address* as a primary key value of the tuple. To get the tuple by its address, we will use $\&_R$ *dereferencing function*: $\forall r \in R\,(\&_R(r.A) = r)$.

Definition 1. Let $R\,(A^*, B, \ldots)\,, T\,(R) = n$ be given. Let a linear order be defined on set $\mathfrak{D}_B$. The *column index* $I_{R.B}$ for attribute B of relation R is an ordered relation $I_{R.B}(A^*, B)$, which satisfies the following requirements:

$$T(I_{R.B}) = n, \pi_A\,(I_{R.B}) = \pi_A\,(R)\,; \tag{1}$$

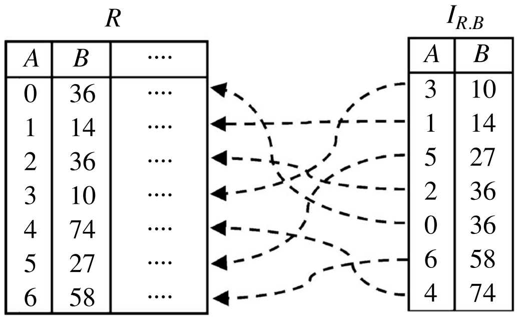
$$\forall x_1, x_2 \in I_{R.B}\,(x_1 \leq x_2 \Leftrightarrow x_1.B \leq x_2.B)\,; \tag{2}$$

$$\forall r \in R\,(\forall x \in I_{R.B}\,(r.A = x.A \Rightarrow r.B = x.B))\,. \tag{3}$$

Condition (1) means that the sets of primary keys of column index and indexed relation are equal. Condition (2) means that index elements are sorted in ascending order of values of attribute B. Condition (3) means that attribute A of an index element contains the address of tuple of R, which has the same value of B attribute as the corresponding element of column index has.

From the intensional point of view, the column index $I_{R.B}$ is a table with two columns A and B (see fig. 1). The number of rows in the column index is equal to the number of rows in the indexed table. Column B of index $I_{R.B}$ contains all the values of column B in table R (including duplicates). These values are sorted in ascending order inside column index.

The following lemma is useful for proving the correctness of decomposition of a relational operators.



R

A	B	....
0	36	....
1	14	....
2	36	....
3	10	....
4	74	....
5	27	....
6	58	....

$I_{R.B}$

A	B
3	10
1	14
5	27
2	36
0	36
6	58
4	74

Fig. 1. Column index.

Lemma 1. *Let relation schema $R(A^*, B, \ldots)$ be given. Let column index $I_{R.B}$ be defined for relation R. Then*

$$\pi_B(I_{R.B}) = \pi_B(R). \tag{4}$$

Proof. Choose an arbitrary $b \in \mathfrak{D}_B$. Let $T(\sigma_{B=b}(R)) = k$. Without loss of generality we may assume that $\forall r \in R\,(r.A < k \Leftrightarrow r.B = b)$. It follows using (1) and (3) that $\forall x \in I_{R.B}\,(x.A < k \Leftrightarrow x.B = b)$. Thus $T(\sigma_{B=b}(I_{R.B})) = k$. Therefore (4) is true. The lemma is proven.

3. DOMAIN-INTERVAL FRAGMENTATION

Definition 2. Let a total ordering relation be defined on domain $\mathfrak{D}_B$. Let a separation of set $\mathfrak{D}_B$ into $k > 0$ nonintersecting intervals is given:

$$\left.\begin{array}{l} V_0 = [v_0; v_1], V_1 = (v_1; v_2], \ldots, V_{k-1} = (v_{k-1}; v_k]; \\ v_0 < v_1 < \ldots < v_k; \\ \mathfrak{D}_B = \bigcup\limits_{i=0}^{k-1} V_i. \end{array}\right\} \tag{5}$$

The function $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$ is called *interval fragmentation function* for domain $\mathfrak{D}_B$ if the following condition holds:

$$\forall i \in \{0, \ldots, k-1\}\,(\forall b \in \mathfrak{D}_B\,(\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i)). \tag{6}$$

Definition 3. Let column index $I_{R.B}$ for relation $R(A^*, B, \ldots)$ with attribute B on domain $\mathfrak{D}_B$. The function

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\}$$

is called *domain-interval fragmentation function* if the following condition holds:

$$\forall x \in I_{R.B}\,(\varphi_{I_{R.B}}(x) = \varphi_{\mathfrak{D}_B}(x.B)). \tag{7}$$

Define ith fragment $(i = 0, \ldots, k-1)$ of index $I_{R.B}$ as follows:

$$I^i_{R.B} = \{x | x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\}. \tag{8}$$

It means that the ith fragment contains tuples, which have values of attribute B from the ith domain interval. This fragmentation is called the *domain-interval fragmentation*. The number of fragments is the *degree of fragmentation*.

The domain-interval fragmentation has the following fundamental properties, which follow directly from its definition:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B};$$

$$\forall i, j \in \{0, \ldots, k-1\} \quad \left(i \neq j \Rightarrow I^i_{R.B} \cap I^j_{R.B} = \emptyset\right).$$

The following lemma is useful for proving the correctness of decomposition of a relational operators.

Lemma 2. *Let a domain-interval fragmentation with degree k be given for column index $I_{R.B}$ of relation $R\left(A^*, B, \ldots\right)$. Then*

$$\forall i \in \{0, \ldots, k-1\} \left(\forall x \in I_{R.B} \left(x \in I^i_{R.B} \Leftrightarrow x.B \in V_i\right)\right). \tag{9}$$

Proof. First, letTs proof

$$\forall i \in \{0, \ldots, k-1\} \left(\forall x \in I_{R.B} \left(x \in I^i_{R.B} \Rightarrow x.B \in V_i\right)\right). \tag{10}$$

Let $x \in I^i_{R.B}$. It follows using (8) that $\varphi_{I_{R.B}}(x) = i$. By the (7), we have $\varphi_{\mathfrak{D}_B}(x.B) = i$. Hence, by the (6), $x.B \in V_i$, it follows that (10) is true.

Second, letTs proof

$$\forall i \in \{0, \ldots, k-1\} \left(\forall x \in I_{R.B} \left(x.B \in V_i \Rightarrow x \in I^i_{R.B}\right)\right). \tag{11}$$

Let $x \in I_{R.B}$ and $x.B \in V_i$. It follows using (6) that $\varphi_{\mathfrak{D}_B}(x.B) = i$. By the (7), we have $\varphi_{\mathfrak{D}_B}(x.B) = \varphi_{I_{R.B}}(x) = i$. Hence, by the (8), $x \in I^i_{R.B}$, it follows that (11) is true. The lemma is proven.

4. DECOMPOSITION OF NATURAL JOIN OF TWO RELATIONS

Let two relations be given:

$$R\left(A^*, B_1, \ldots, B_u, C_1, \ldots, C_v\right)$$

and

$$S\left(A^*, B_1, \ldots, B_u, D_1, \ldots, D_w\right).$$

Let the following two sets of column indices for attributes $B_1, \ldots, B_u$ be given:

$$I_{R.B_1}, \ldots, I_{R.B_u}; \quad I_{S.B_1}, \ldots, I_{S.B_u}.$$

Let the following domain-interval fragmentation with degree k be defined for all these indices:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; \tag{12}$$

$$I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}. \tag{13}$$

Let

$$P^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\, I^i_{S.B_j}.A \to A_S} \left(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j} \right) \tag{14}$$

for all $i = 0, \ldots, k-1$ and $j = 1, \ldots, u$. Define

$$P_j = \bigcup_{i=0}^{k-1} P^i_j. \tag{15}$$

Let

$$P = \bigcap_{j=1}^{u} P_j. \tag{16}$$

Define

$$Q = \left\{ r \circ (s.D_1, \ldots, s.D_w) | r \in R \wedge s \in S \wedge (r.A, s.A) \in P \right\}. \tag{17}$$

Theorem 1. $\pi_{*\backslash A}(Q) = \pi_{*\backslash A}(R) \bowtie \pi_{*\backslash A}(S)$.

Proof. First, let's proof

$$\pi_{*\setminus A}(Q) \subset \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S). \tag{18}$$

Let

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v, d_1, \ldots, d_w) \in Q. \tag{19}$$

The (17) implies that there exist tuples r and s such that

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v) = r \in R, \tag{20}$$

$$(a', {b_1}', \ldots, {b_u}', d_1, \ldots, d_w) = s \in S \tag{21}$$

and

$$(r.A, s.A) \in P. \tag{22}$$

It follows that $\exists p \in P(p.A_R = a \wedge p.A_S = a')$. Hence, by the (16) we get $\forall j \in \{1, \ldots, u\} \times (\exists p \in P_j(p.A_R = a \wedge p.A_S = a'))$. It follows using (15) that $\forall j \in \{1, \ldots, u\}(\exists i \in \{0, \ldots, k-1\} \times (\exists p \in P_j^i(p.A_R = a \wedge p.A_S = a')))$. Thus since (12)–(14), it follows that

$$\forall j \in \{1, \ldots, u\} \left(\exists x \in I_{R.B_j} \left(\exists y \in I_{S.B_j} \left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a' \right) \right) \right).$$

Hence, by the Definition 1 we get

$$\forall j \in \{1, \ldots, u\} \left(\exists \tilde{r} \in R \left(\exists \tilde{s} \in S \left(\tilde{r}.A = a \wedge \tilde{r}.B_j = \tilde{s}.B_j \wedge \tilde{s}.A = a' \right) \right) \right).$$

Since A is a primary key in R and S, using (20) and (21), it follows that $(a', b_1, \ldots, b_u, d_1, \ldots, d_w) \in S$, hence $(b_1, \ldots, b_u, c_1, \ldots, c_v, d_1, \ldots, d_w) \in \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Hence (18) holds.

Now, let's proof

$$\pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S) \subset \pi_{*\setminus A}(Q). \tag{23}$$

Let

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v) = r \in R \tag{24}$$

and

$$(a', b_1, \ldots, b_u, d_1, \ldots, d_w) = s \in S. \tag{25}$$

Therefore, by the column index definition, using (4), we get

$$\forall j \in \{1, \ldots, u\} \quad \left(\exists x \in I_{R.B_j} \left(\exists y \in I_{S.B_j} \left(x.A = a \wedge x.B_j = b_j = y.B_j \wedge y.A = a' \right) \right) \right).$$

Combining it with (9), we obtain

$$\forall j \in \{1, \ldots, u\}$$
$$\left(\exists i \in \{0, \ldots, k-1\} \left(\exists x \in I^i_{R.B_j} \left(\exists y \in I^i_{S.B_j} \left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a' \right) \right) \right) \right).$$

Now, by (14),

$$\forall j \in \{1, \ldots, u\} \quad \left(\exists i \in \{0, \ldots, k-1\} \left(\exists p \in P_j^i \left(p.A_R = a \wedge p.A_S = a' \right) \right) \right).$$

Using (15), we get

$$\forall j \in \{1, \ldots, u\} \quad \left(\exists p \in P_j \left(p.A_R = a \wedge p.A_S = a' \right) \right)$$

By (16), it follows that $(a, a') \in P$. Summing it with (17), (24) and (25), we get

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v, d_1, \ldots, d_w) \in Q,$$

so (22) holds. The theorem is proven.

5. CONCLUSION

In this article, we presented a method of the decomposition of the natural join operator based on the column indices and the domain-interval fragmentation. The correctness of this method was proven. We used the described method in a prototype of parallel DBMS coprocessor. This prototype was implemented on SMP with Xeon Phi accelerator. The computational experiments performed using our prototype confirmed the efficiency of proposed approach [13]. This approach can be generalized for many another relational operators. We plan to design the corresponding decomposition methods for intersection, grouping and some others relational operators.

ACKNOWLEDGMENTS

The study was supported by the Ministry of Education and Science of the Russian Federation under Federal targeted program "Research and development in priority fields of scientific and technological complex of Russia in 2014–2020" (Governmental contract No. 14.574.21.0035).

REFERENCES

1. V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, The Digital Universe of Opportunities: Rich Data and the creasing Value of the Internet of Things. IDC white paper. April 2014. Available: http://idcdocserv.com/1678 [November 06, 2014].
2. L. B. Sokolinsky, Programming and Computer Software **30** (6), 337–346 (2004).
3. C. S. Pan and M. L. Zymbler, Lecture Notes in Computer Science **8055**, LNCS, Pt. 1, 153–164 (2013).
4. K. Y. Besedin and P. S. Kostenetskiy, *Simulating of query processing on multiprocessor database systems with modern coprocessors*, 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) 2014, Opatija, Croatia, May 26–30, IEEE, 1835–1837 (2014).
5. D. J. Abadi, S. R. Madden, and N. Hachem, *Column-Stores vs. Row-Stores: How Different Are They Really? Proceedings of the 2008 ACM SIGMOD international conference on Management of data, June 912, 2008* (Vancouver, BC, Canada. ACM, 2008), p. 967–980.
6. H. Plattner and A. Zeier, *In-Memory Data Management: An Inflection Point for Enterprise Applications* (Springer, 2011), 254 p.
7. J. Fang, A. L. Varbanescu, and H. Sips, *Sesame: A User-Transparent Optimizing Framework for Many-Core Processors, Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2013), May 13–16, 2013* (Delft, Netherlands. IEEE, 2013), p. 70–73.
8. S. Breß, F. Beier, H. Rauhe, K.-U. Sattler, E. Schallehn, G. Saake, *Efficient Co-Processor Utilization in Database Query Processing*, Information Systems. **38** (8), 1084–1096 (2013).
9. M. Scherger, *Design of an In-Memory Database Engine Using Intel Xeon Phi Coprocessors, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'14), July 21–24, 2014* (Las Vegas, USA. CSREA Press, 2014), p. 21–27.
10. P. A. Deshmukh, *Review on Main Memory Database*, International Journal of Computer & Communication Technology **2**, Issue 7, 54–58 (2011).
11. H. LeHong and J. Fenn, Hype Cycle for Emerging Technologies. Gartner Inc. Research Report. August 2013. Available: http://www.gartner.com/doc/2571624 [December 16, 2014].
12. H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book (2nd Edition)* (Prentice Hall, 2008), 1224 p.
13. E. V. Ivanova and L. B. Sokolinsky, *Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices*, 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, 2015, Proceedings, IEEE, 2015, pp. 210–213.