Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices

Elena Ivanova and Leonid Sokolinsky
South Ural State University, Chelyabinsk, Russia
ivanovaev@susu.ac.ru, Leonid.Sokolinsky@susu.ru

Abstract - The paper describes decomposition of natural join relational operator based on the column indices and domain-interval fragmentation. This decomposition admits parallel executing the resource-intensive relational operators without data transfers. All column index fragments are stored in main memory in compressed form to conserve space. During the parallel execution of relational operators, compressed index fragments are loaded on different processor cores. These cores unpack fragments, perform relational operator and compress fragments of partial result, which is a set of keys. Partial results are merged in the resulting set of keys. DBMS use the resulting set of keys for building the resulting table. Described approach allows efficient parallel query processing for very large databases on modern computing cluster systems with many-core accelerators. A prototype of the DBMS coprocessor system was implemented using this technique. The results of computational experiments are presented. These results confirm the efficiency of proposed approach.

I. 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.

One of the popular ways 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] – [5]. 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 [6]. 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 [7].

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 and memory bandwidth, can use hundreds of parallel processor cores to execute tens of thousands of threads [8]. 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 [9]. 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 [10].

Main memory as the primary storage location is becoming increasingly attractive as a result of the decreasing cost/size ratio [7]. 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 [11]. According to Gartner’s 2013 Hype Cycle for Emerging Technologies report, in-memory database management system have 2 to 5 years until widespread adoption [12].

According to this, the problem of developing new efficient methods of parallel database processing in main memory on modern compute clusters with many-core 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 operators, which admits the efficient parallel execution of them on computing cluster system, equipped with many-core accelerators. In this paper, we consider the decomposition of the natural join operator. In this paper, we use the notation from [13]. The symbol $\circ$ we use to denote the operation of concatenation of the tuples.

II. Column Index

Let $R(A, B_1, \ldots, B_u)$ be the relation R with *virtual key* (virtual record identifier) A 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 virtual key of tuple r: $r = (r.A, r.B_1, \ldots, r.B_u)$. The *virtual key* of relation R has the property: $\forall r', r'' \in R\left(r' \neq r'' \Leftrightarrow r'.A \neq r''.A\right)$. Define *tuple address* as a virtual key value of the tuple. To get the tuple by its address, we will use $\&_R$ *dereferencing function*: $\forall r \in R\left(\&_R(r.A) = r\right)$.

Let $R\left(A^*, B, \ldots\right)$, $T(R) = n$ 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, which satisfies the following requirements:

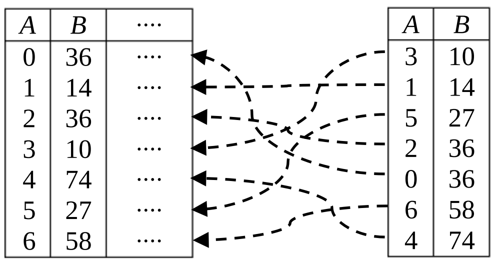
$$T(I_{R.B}) = n \text{ и } \pi_A\left(I_{R.B}\right) = \pi_A\left(R\right); \tag{1}$$

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

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

Condition (1) means that the sets of virtual 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 и 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.



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

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

Figure 1. Column index

III. Domain-Interval Fragmentation

Let a total ordering relation be defined on domain $\mathfrak{D}_B$. Divide $\mathfrak{D}_B$ into $k > 0$ nonintersecting intervals:

$$\left.\begin{array}{l} V_0 = [v_0; v_1], V_1 = (v_1; v_2], \ldots, V_{k-1} = (v_{k-1}; v_k]; \\ v_0 < v_1 < \ldots < v_k; \\ \mathfrak{D}_B = \bigcup\limits_{i=0}^{k-1} V_i. \end{array}\right\}. \tag{4}$$

Define *interval fragmented function* on domain $\mathfrak{D}_B$ as $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$. This function satisfies the following requirement:

$$\forall i \in \{0, \ldots, k-1\}\left(\forall b \in \mathfrak{D}_B\left(\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i\right)\right). \tag{5}$$

Let column index $I_{R.B}$ be given for relation $R\left(A^*, B, \ldots\right)$ with attribute B on domain $\mathfrak{D}_B$. Let interval fragmented function $\varphi_{\mathfrak{D}_B}$ be defined on domain $\mathfrak{D}_B$. The function

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\} \tag{6}$$

is *domain-interval fragmented function* for index $I_{R.B}$, if it satisfies the following requirement:

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

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

$$I_{R.B}^i = \left\{x \mid x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\right\}. \tag{8}$$

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:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I_{R.B}^i; \tag{9}$$

$$\forall i, j \in \{0, \ldots, \mathrm{k}-1\}\left(i \neq j \Rightarrow I_{R.B}^i \cap I_{R.B}^j = \varnothing\right). \tag{10}$$

IV. Decomposition of the Natural Join Operator

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

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

$$I_{S.B_1}, \ldots, I_{S.B_u}.$$

Let domain-interval fragmentation of degree k be defined for these indices:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; \qquad (11)$$

$$I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j}. \qquad (12)$$

Let

$$P^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\, I^i_{S.B_j}.A \to A_S} \left(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j} \right) \qquad (13)$$

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

$$P_j = \bigcup_{i=0}^{k-1} P^i_j. \qquad (14)$$

Let

$$P = \bigcap_{j=1}^{u} P_j. \qquad (15)$$

Define

$$Q = \{r.B_1, \ldots, r.B_u, r.C_1, \ldots, r.C_v, s.D_1, \ldots, s.D_w \mid r \in R \wedge s \in S \wedge (r.A, s.A) \in P\}.$$

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

Note that calculation of P^i_j by (13) can be done in parallel on *k* different processors without data exchange. It ensures a near-linear speedup.

V. Performance Evaluation

The described approach was implemented as a prototype of DBMS coprocessor system. The source code of the program is openly available in the public GitHub repository [15]. Column indices and domain-interval fragmentation were evaluated using this prototype. We performed natural join operator over fragmented column indices $I_{R.B}$ and $I_{S.B}$. Attribute B of column index $I_{R.B}$ is a virtual key. Attribute B of column index $I_{S.B}$ is a foreign key. Join operator was implemented by merge join algorithm.

Number of column index tuples were following: $T(I_{R.B})$ = 600 000 and $T(I_{S.B})$ = 60 000 000. Column indices $I_{R.B}$ and $I_{S.B}$ were fragmented using domain-interval fragmentation. Every fragment of column indices was compressed by Zlib data compression library [14]. In described experiments, attribute B of column index $I_{S.B}$ had uniform and nonuniform (rule: 80/20, 65/20 and 45/20) data distribution.

The experiments were done on Intel Xeon Phi accelerator with 61 cores. Join operator was performed in 1, 2 and 4 threads per core (see fig. 2). The results of this experiment show that we have maximum speedup if we use 1 thread per core. The second experiment (see fig. 3) shows that load balancing can be effectively managed by increasing the number of fragments.

The experiments have shown that approach on base column indices allow to perform resource-intensive join operator for $T(I_{R.B})$ = 600 000 and $T(I_{S.B})$ = 60 000 000 during 1 second on one Intel Xeon Phi accelerator. The described approach eliminates data transfer, so we can expect a near-linear speedup on computing cluster systems with million nodes equipped with many-core accelerators.

VI. Related Work

Binary table model was introduced in the paper [16]. On the basis of this model, several column-oriented DBMS were designed. As it was demonstrated by work [17] and [18], column-oriented systems offer an order-of-magnitude performance improvement over traditional row-oriented systems for analytical processing workloads, such as those found in data warehouses or decision support systems. One of the main disadvantages of column-oriented DBMS is lacking the optimization technique, which is intrinsic to relational (row-oriented) DBMS. The work [19] investigated column-oriented simulation in a relational DBMS via the following techniques: vertical partitioning, index-only plans and materialized views. The investigation

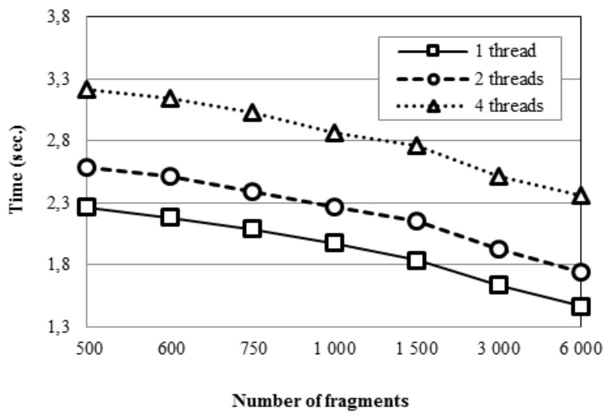


Figure 2. Dependence between time of calculation and number of fragments

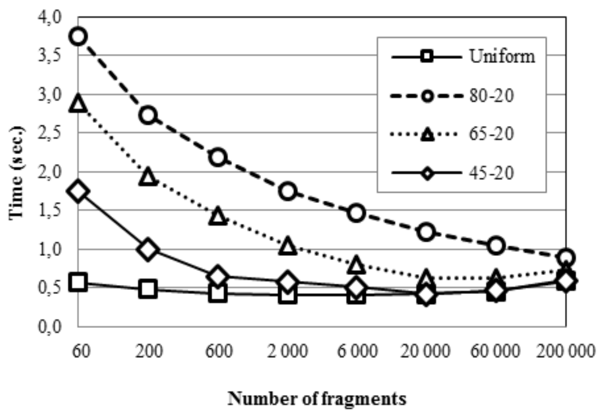


Figure 3. The influence of number of fragments on load balancing of Xeon Phi cores

showed that such techniques do not improve the performance of row stores for analytical processing workloads. To overcome the problems faced with work [19], the work [20] introduced two new operators: Index Merge and Index Merge Join. The algorithms presented in this paper were designed specifically to take advantage of parallel processing whenever possible. Another approach was proposed in work [21]. This paper introduced a new index type, column store indexes, where data is stored column-wise in compressed form. Column store indexes are intended for data-warehousing workloads where queries typically process large numbers of rows but only a few columns. To further speed up such queries, the paper [21] also introduced a new query processing mode, batch processing, where operators process a batch of rows (in columnar format) at a time instead of a row at a time.

VII. Conclusions

In this article, we presented a decomposition of the natural join operator based on the column indices and the domain-interval fragmentation. Our approach was evaluated using the prototype DBMS coprocessor system. Experiments showed its efficiency for a resource-intensive natural join operator. Proposed approach can be used on computing cluster systems with many-core accelerators. Described technique is suitable for data warehouse workloads as well as for OLTP workloads.

As a direction of a future research, we are going to use described approach for the decomposition of another relational operators and compare speedup with existing DBMS.

Acknowledgment

The study was supported by the Ministry of education and science of Russia 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, 2014. http://idcdocserv.com/1678 [January 20, 2015].

[2] L.B. Sokolinsky, "Survey of architectures of parallel database systems," Programming and Computer Software, vol. 30, No. 6, pp. 337-346, 2004.

[3] A.V. Lepikhov and L.B. Sokolinsky, "Query processing in a DBMS for cluster systems," Programming and Computer Software, vol. 36, No. 4, pp. 205-215, 2010.

[4] C.S. Pan, and M.L. Zymbler, "Taming elephants, or how to embed parallelism into PostgreSQL," in Processing of the 24th International Conference, DEXA 2013, Prague, Czech Republic, part I, vol. 8055, pp. 153-164, August, 2013.

[5] K.Y. Besedin, and P.S. Kostenetskiy, "Simulating of query processing on multiprocessor database systems with modern coprocessors," in Proceedings of the 37th International Convention, MIPRO 2014, Opatija, Croatia, pp. 1835-1837, May, 2014.

[6] D.J. Abadi, S.R. Madden, and N. Hachem, "Column-stores vs. row-stores: how different are they really?," in Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, pp. 967-980, June, 2008.

[7] H. Plattner, and A. Zeier, In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer, 2011.

[8] J. Fang, A.L. Varbanescu, and H. Sips, "Sesame: a user-transparent optimizing framework for many-core processors," in Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid2013, Delft, Netherlands, pp. 70-73, May, 2013.

[9] S. Breß, F. Beier, H. Rauhe, K.-U. Sattler, E. Schallehn, and G. Saake, "Efficient Co-Processor Utilization in Database Query Processing," Information Systems, vol. 38, No. 8, pp. 1084–1096, 2013.

[10] M. Scherger, "Design of an In-Memory Database Engine Using Intel Xeon Phi Coprocessors," in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'14, Las Vegas, USA, pp. 21-27.

[11] P.A. Deshmukh, "Review on Main Memory Database," International Journal of Computer & Communication Technology, vol. 2, No. 7, pp. 54–58, 2011.

[12] H. LeHong, and J. Fenn, "Hype Cycle for Emerging Technologies," Gartner Inc, Research Report, 2013. http://www.gartner.com/doc/2571624 [December 16, 2014].

[13] H. Garcia-Molina, J.D. Ullman, and J. Widom, Database Systems: The Complete Book (2nd Edition). Prentice Hall, 2008. 1224 p.

[14] G. Roelofs, J. Gailly, and M. Adler. Zlib. http://www.zlib.net/ [April 4, 2015].

[15] A prototype of the DBMS coprocessor system using colomn indices based on domain-Interval fragmentation. https://github.com/elena-ivanova/colomnindices [April 4, 2015].

[16] G.P. Copeland, and S.N. Khoshafian, "A decomposition storage model," in Proc. SIGMOD, 1985, pp. 268-279.

[17] P.A. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-Pipelining Query Execution," in Proc. CIDR, 2005, pp. 225-237.

[18] M. Stonebraker, D.J. 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 S. Zdonik, "C-Store: A Column-Oriented DBMS," in Proc. VLDB, 2005, pp. 553-564.

[19] D.J. Abadi, S.R. Madden, and N. Hachem, "Column-Stores vs. Row-Stores: How Different Are They Really?," in Proc. SIGMOD, 2008, pp. 967-980.

[20] A. El-Helw, K.A. Ross, B. Bhattacharjee, C.A. Lang and G.A. Mihaila, "Column-oriented query processing for row stores," in Proc. DOLAP, 2011, pp. 67-74.

[21] P. Larson, C. Clinciu, E.N. Hanson, A. Oks, S.L. Price, S. Rangarajan, A. Surna and Q. Zhou, "SQL server column store indexes," in Proc. SIGMOD Conference, 2011, pp. 1177-1184.