DEWS2004 5-A-03

PUB/SUB システムにおける UB-Tree インデクスの性能解析

張　旺[†]　　王　波涛[†]　　喜連川　優[†]

† 東京大学　生産技術研究所
〒１３５－８５０５　東京 目黒区駒場４－６－１
E-mail: †{zhangw,botaow,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　A publish/subscribe system dynamically routes and delivers information from information producers to interested users. Efficient event filtering (or event matching) algorithms are the kernel of publish/subscribe systems. And most of research efforts are focusing on the multiple one-dimension indexes in last several years. There are two main problems in such kind of systems: the heavy workload of the indices' maintenance and the poor performance of index searching process for the inequality operators $(>, <)$. We propose a multi-dimension predicate index based on the UB-tree, which has no limit on the using of operators $(=, <, >)$. Also we just use one index to reduce the workload of maintenance dramatically. In this paper, our proposal will be introduced and compared with other famous algorithms.

キーワード　UB-tree, Multi-dimension index, Event filtering, publish/subscribe system, predicate index

Performance Analysis of UB-tree Indexed Pub/Sub System

Wang ZHANG[†], Wang ZHANG[†], and Masaru KITSUREGAWA[†]

† Institute of Industrial Science, The University of Tokyo,
Meguro ward Komaba 4–6–1, Tokyo, 135–8505 Japan
E-mail: †{zhangw,botaow,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract　A publish/subscribe system dynamically routes and delivers information from information producers to interested users. Efficient event filtering (or event matching) algorithms are the kernel of publish/subscribe systems. And most of research efforts are focusing on the multiple one-dimension indexes in last several years. There are two main problems in such kind of systems: the heavy workload of the indices' maintenance and the poor performance of index searching process for the inequality operators $(>, <)$. We propose a multi-dimension predicate index based on the UB-tree, which has no limit on the using of operators $(=, <, >)$. Also we just use one index to reduce the workload of maintenance dramatically. In this paper, our proposal will be introduced and compared with other famous algorithms.

Key words　UB-tree, Multi-dimension index, Event filtering, publish/subscribe system, predicate index

1. Introduction

A publish/subscribe system connects together information producers, which publish events to the system, and information consumers, which subscribe to particular types of events within the system. The system is responsible for identifying the set of subscriptions that are matched by a published event (if any), and for notifying the corresponding subscribers. The earliest publish/subscribe systems were subject-based. In such systems, information consumers subscribe to one or more subjects and the system notifies them each time an event, classified as belonging to one of the subjects they subscribed to is published. Event matching is a straightforward task in these systems because events can be filtered only according to their subject. Any additional event filtering has to be done by the subscriber himself.

An attractive alternative to subject-based systems is content-based subscription systems. These systems appear to be more promising in meeting subscriber's needs of defining filtering criteria (conjunction of predicates) when they register their interest in receiving publications. Compared to subject-based systems, content-based systems allow subscribers to express a "query" against the content of a published event. Examples of content-based systems are Le Subscribe [23], and READY [13]. In general, most content-based subscriptions systems use quite similar publication and subscriptions languages. In these systems, an event is distinguished based on its event schema. An event schema defines the type of the

information contained in each event, and the system usually supports multiple event schemas.

In our study, we build a predicate-based index on UB-tree by dimension transform for efficient event matching. This paper is organized as following: In section 2, is the background of our study. In section 3 the UB-tree and dimension transformation will be introduced. Then our work on how to optimize the event filtering process will be introduced in section 4. And some the related works is introduced in section 5. At last, the experiments and comparisons among our proposal with other famous solutions will be introduced in section 6. The last section is our conclusion.

2. Background

2.1 Event Matching Problem

The event matching problem can be expressed as follows. Given an event e and a set of subscriptions S, determine all subscriptions in S that are matched by e. A subscription is a conjunction of predicates. A predicate is a triple consisting of an attribute, a constant, and a relational operator (<, <=, =, !=, >=, >). A subscription schema defines the type of the information to be supported by publish/subscribe system. The attributes are defined in subscription schema. For example, three attributes: *CompanyName, Price* and *ChangeRatio* with string, float and float types respectively can be defined for stock market.

Following is a subscription example of stock schema, (*CompanyName* = *Yahoo*) AND $(Price > 1000)$ AND $(Change - Ratio < 0.05)$. An event is an array of pairs of (Attribute, Constant). The size of array depends on subscription schema. Following is an event example of stock schema, (*CompanyName, Intel*), (*Price, 5000*), (*ChangeRatio, 0.03*). An event *e* matches a subscription S if all predicates in S are satisfied by some (Attribute, Value) pairs in e. For example, event (*CompanyName, Yahoo*), ($price$, 500), (*ChangeRatio, 0.1*) matches following subscription which is expressed as a conjunction of two predicates: $(CompanyName = Yahoo)$ AND $(Price < 1000)$.

Event matching algorithms in content-based publish/subscribe systems can be classified into two categories:

- Algorithms based on predicate index. The algorithms based on predicate indexing consist of two steps:
  - The first step determines all predicates that are satisfied by the event.
  - The second step finds all subscriptions that are matched by the events based on the results of the first phase.

Algorithms based on predicate indexing techniques use a set of one-dimensional index structures to index predicates in the subscriptions. They differ from each other by the way to select predicates from subscriptions, which are kept in the index structures [7] [10] [15] [17] [20] [23] [28].

Basically, the predicates are grouped based on all subscriptions. A predicate family consists of predicates having the same attribute. For each attribute, one predicate index is built. For example, for stock schema introduced previously, three predicate indexes will be built for $CompanyName$, $Price$, $ChangeRatio$.

- Algorithm based on subscription index [1] [18]. The techniques based on subscription index insert subscriptions into a matching tree. Events enter the tree from root node and are filtered through by intermediate nodes. An event that passes all intermediate testing nodes reaches leaf nodes where references of matching subscriptions are stored.

Our proposed solution is also designed for predicate index. Although there are many proposals for selection of predicates from subscriptions, [10] [15] [23] [28], the predicate index is essential while determines all the predicates that are satisfied by the event at the first step. From next introduction, we will concentrate on the predicate index. For details of different methods of predicates selection, please refer to [10] [15] [23] [28].

3. UB-Tree and Dimension Transformation

From viewpoint of search in high dimensional data space, event filtering of subscriptions using operators $<$ or $>$ can be regarded as the following two kinds of queries:

- Events are point enclosure queries and subscriptions are hyper cubes.
- Events are range queries and subscriptions are points. In this case, dimension transform is required.

Because the attributes used in subscriptions should not be fixed, there are lots of incomplete subscription hyper cubes which overlap each other heavily. So normally it is hard to use of multidimensional indices structure directly to build efficient indices on those subscriptions hyper cubes for point enclosure query. For this reason, we choose range query and do dimension transform in our design.

As introduced in [6] [9] [8] [12], many multidimensional index structures have been proposed for range query. Because besides efficient event filtering (search), publish/subscribe system requires both dynamic maintenance and space efficiency, not all multidimensional index structure can meet above requirements. For example, performance of R-tree [14] and R*-tree [4] suffer from region splitting and merging while updating index. R+-tree [26] cannot guarantee a minimal storage utilization; KD-tree [5] is sensitive to the order in which the points are inserted; quadtree [25] is unbalanced and sensitive to data density. UB-tree [2] [11] [24] is designed to perform multidimensional range query. It is a dynamic index structure based on B-tree and supports updates with logarithmic performance like B-tree with space complexity O(n). For above reasons, we choose UB-tree to perform range query in our design.

3.1 UB-Tree

The UB-tree [22] is a clustering index for multidimensional point data, which inherits all good properties of the B-tree. Logarithmic performance guarantees are given for the basic operations of insertion, deletion and point query. The UB-tree clusters data ac-

$$Z(x) = \sum_{i=0}^{s-1} \sum_{j=1}^{d} x_{j,i} \cdot 2^{i \cdot d + j - 1}$$

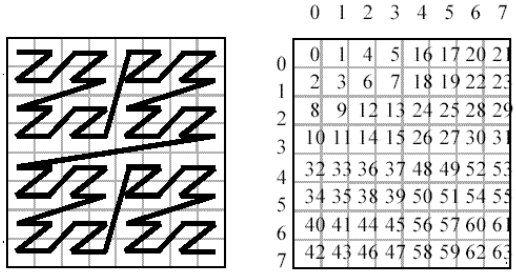


図 1　Z-curve and Z-address

cording to a space filling curve, which is named as the Z-curve and introduces the new idea of partitioning the data space into disjoint Z-regions, which are mapped into disk pages. The Z-regions are then indexed by a B-tree using last included Z-address as key, which is the ordinal of a point on the Z-curve. As shown in Fig.1, Z(x) is a bijective function that computes for every tuple x its Z-address, i.e., its position on the space filling Z-curve. The slide presents the Z-addresses (or Z-values) for an 8x8 universe in Fig.1(a). Z-values are efficiently computed by bit-interleaving as described in Fig.1(b).

These Z-regions in conjunction with a sophisticated algorithm for multidimensional range queries [3] and the Tetris algorithm [21] for sorted reading of multidimensional ranges offer excellent properties [22] for multidimensional applications like data warehousing, archiving systems, temporal data management, etc. The middle part, in the Fig.2, shows a Z-region partitioning (or also called UB-tree partitioning) which is a disjoint set of Z-regions whose union covers the entire multidimensional space. In this figure the partitioning consists of 5 Z-regions. Most Z-regions preserve spatial proximity, i.e., neighboring points of a given point are in the same region with a high probability. The region [21 : 35] consists of two disconnected parts. If a Z-region could consist of many disconnected parts, this would prevent Z-regions from being suitable for clustering. However, [22] gives a proof that regardless of the dimensionality of the Z-ordered space (i.e., not only for 2d) the number of not connected parts of a Z-region is at most two.

3.2 Dimension Transform for Event Filtering

For one attribute A with value range $[IMin, IMax]$, the corresponding predicate with format of $Istart <= A <= Iend$ can be represented as an interval of $[Istart, Iend]$. Given a corresponding event with value $Evalue$, if the predicate is satisfied, it means

$$Istart <= Evalue <= Iend$$

logically it is equal to

$(IMin <= Istart <= Evalue)$ AND $(Evalue <= Iend <= IMax)$

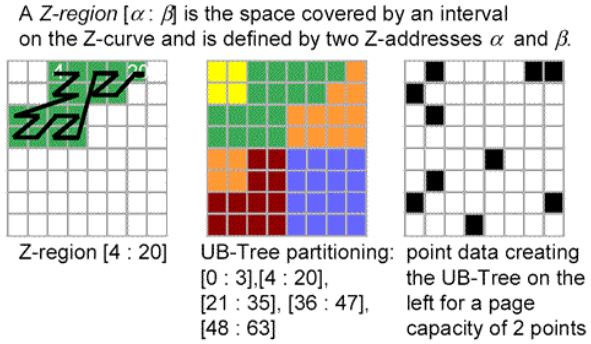


図 2　Z-regions and Z-space

By defining two new dimensions $AStart$ and $AEnd$ for $Istart$ and $Iend$, 1D dimensional point enclousre query can be transformed to 2D range query as shown in Fig.3.

For one attribute, after transform from 1D to 2D, event becomes range query and subscription becomes point data. The new 2D space has following properties:

- Event range. Event range is determined by two vertexes in 2D space. Upper left corner (IMin, IMax) is fixed. Lower right corner $(Evalue, Evalue)$ is always located on the diagonal of 2D space as shown in Fig. 3.
- Equality predicate point. It means $Istart == Iend$ is true, so it is located on the diagonal of 2D space.
- Half-interval predicate point. Half-interval predicate means only one operator is used, like $Istart <= A$ or $A <= Iend$. It logically equals to $Istart <= A <= IMax$ or $IMin <= A <= Iend$. The half-interval predicate point is located on the border of 2D space above the diagonal.
- $TRUE$. For uncompleted subscription, only parts of attributes are used. Because subscription is a conjunction of predicates, for the attributes not be used in the subscription, their related predicates should always be considered as $TRUE$. Logically $TRUE$ can be represented as $IMin <= A <= IMax$. Then $TRUE$ is a point with constant value $(IMin, IMax)$.
- Dead Space. Because $Istart <= Iend$, there is no data located in the space under diagonal space. It's called dead space. And we don't need to allocate the space for this area.

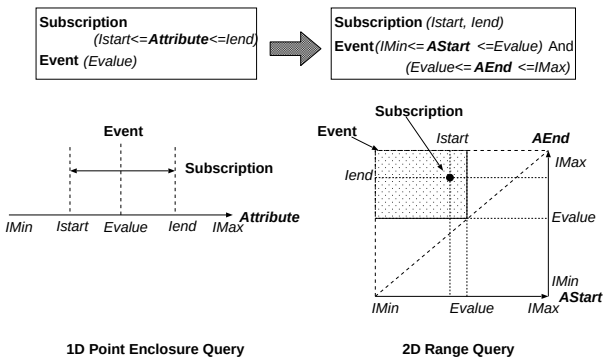


図 3　Transform 1D point enclosure query to 2D range query

For above properties, even the data in 1D space are distributed uniformly, it is very possible that data skew occurs after transform. Without data skew, the number of results of range query will be limited for its low selectivity on high dimensional space. Data skew depends on the percentage of kinds of predicates used in subscriptions. Its influence on performance of event filtering will be shown and discussed later in Fig.7(d) and Fig.7(e).

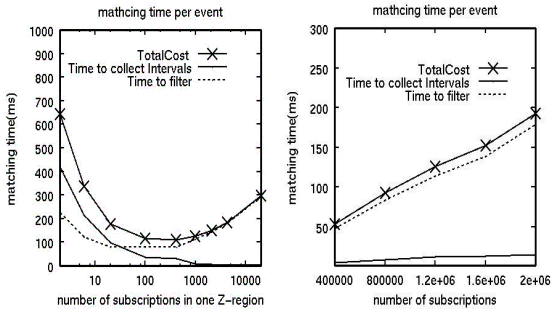
4. Performance Improvement

With the emergence of cheap computers with huge memory, more and more algorithms can be run in the main memory. Considering the performance, our solution is designed to be executed in the main memory too. In this section, after analyzing the workload distribution of UB-tree's range querying, we propose our optimizing methods focusing on reducing the cost of filtering operation [27].

4.1 Workload Distribution of Range Query

Because original UB-tree is designed for secondary storage, the performance is dominated by I/O cost. The main task of its range query is to calculate efficiently the set of one-dimensional intervals of Z-value (Z-regions). Our index is designed to run in main memory, for performance improvement, the workload distribution of calculating intervals of Z-value and filtering results from candidate objects kept in the selected Z-regions, should be considered comprehensively.

According to UB-tree's structure,workload distribution depends on the setting of the maximum number of objects kept in one Z-Region (corresponding to one leaf node of B+tree) as shown in Fig.4(a) (Please refer to Table.1 in Section 5. for detail information of test environment).



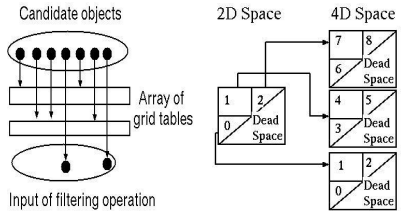
(a) (b)

図 4 Workload distribution

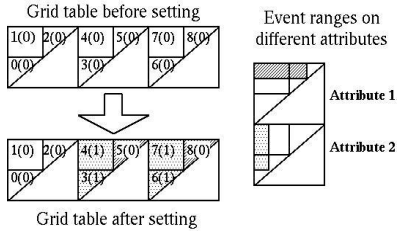
From Fig.4(a) we can find that there exists a value range (nearly from 400 to 900 here) where the best performance can be gotten. So we will do some optimization work within this range. As shown in Fig.4(b) (the maximum number is 700 there), the workloads of this two steps(filtering process and the intervals collecting process) are different. Cost on filtering operation accounts for major part of the total cost. And the cost changes linearly with the number of subscriptions (objects). Cost to collect intervals (Z-regions) is relatively small and stable. So the goal of the optimization of UB-tree in the main memory is to reduce the cost related to filtering operation. Two methods are proposed: one is reducing the input of filtering operation (Section 4. 2); another is improving the performance of filtering operation itself (Section 4. 3).

4.2 Reduce Input of Filtering Operation

The basic idea is shown in Fig.5(a), an array of grid tables is created dynamically to reduce input of filtering operation. Each dimension is equally divided by a linear hash and the total space is divided into grid similar to grid file. Grid table is an array of grid cells, which is not located in dead space. It records whether each cell is covered or not by the incoming event range. The sequence number of the item in grid table (cell array) is called cell ID. The structure of one grid table is shown in Fig.5(b). After dimension transformation, N attributes corresponds to one 2ND space. The left side of Fig.5(b) shows an example of cell ID setting in 2D space. The right side of Fig.5(b) shows an example of cell ID setting in 4D space. Here each dimension is divided equally into two parts. The links show the corresponding relation between the 2D space and 4D space when number of attributes changes from 1 to 2. The method of calculating cell ID is straightforward and skipped here.



(a) Basic idea (b) Cell Ids in different spaces



(c) Value setting of grid table in 4D space

図 5 The way to reduce input of filtering operation and Grid Table

In order to make use of grid table, the following extension should be done:

- While inserting a subscription point, its corresponding cell ID is calculated and kept inside the index with the subscription point. Because each subscription point corresponds to one cell of the grid, the subscription can compute its cell ID according to its coordinates on dimensions.

- Before searching index, the grid table should be reset and filled according to the range of input event. For the cell intersecting with the event range, its entry will be set to 1, otherwise left to 0 as shown in Fig.5(c). The content of each item is ”cell ID(value)”. Cell ID is not kept in the item.

As introduced previously, grid table is dynamically built and assigned. Because size of grid table increases exponentially with the increasing of the number of attributes, building one grid table on all attributes is impractical. In order to save the size of grid table, only parts of attributes with higher selectivity and larger value domain, are selected to build grid table. Further these attributes are divided into groups to reduce the exponential increment of the memory used caused by the large number of selected attributes. Each group corresponds to disjunctive lower dimensional space. That's the reason why an array of grid tables is used in Fig.5(a). In this case, corresponding different groups, multiple grid tables should be created and checked while do event filtering.

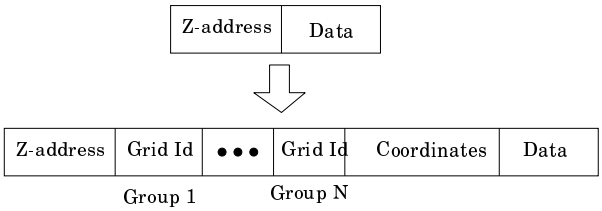


図 6　Entry extension of UB-tree node

Before filtering a candidate object (subscription point) of UB-tree, the cell ID(s) kept with the candidate object will be used to check whether the corresponding cells intersect with the input event range by looking up the grid table(s). If one of the cell values is 0, that means the corresponding cell doesn't intersect with input range, and then there is no need to send this candidate object to the filtering operation further. So the input of filtering operation is reduced.

Because this optimization method is not dependent on UB-tree, so it can be applied to other similar multidimensional index structure. It is the intersection of two orthogonal partitioning methods: space-filling curve and grid table. Even the adding of grid table is a kind of overhead, the cost of maintain an array of grid tables with smaller size, can be neglected compared with larger number of candidate objects in a large database. The effectiveness of the grid table will be shown later in Fig.8.

4.3 Improve Performance of Filtering Operation

As introduced before, Z-address is the key gotten by bit-interleaving of coordinates corresponding to all dimensions. Even the Z-address can be used to do filtering operation directly, the operation is much expensive than the bitwise operation. In order to improve the performance of filtering operation, the coordinates of subscription points are added in UB-tree like Z-addresses.

According to above two optimization methods, the structure of a node entry(注1) in UB-tree is extended as shown in Fig.6.

5. Performance Evaluation

In this section, we'll evaluate our proposed index and the effectiveness of optimizing method. At same time we compare it with 1) counting algorithm since it is used in many publish/subscribe systems and 2) bruteforce considering about the curse of dimensionality. 3) access-predicate based algorithm which has a high performance. 4) R-tree considering about the multidimensional indices' comparison for the original R-tree without dimension transformation.

5.1 Environment

The set of parameters used in simulation is listed in Table 1. In all of the experiments presented in the rest of the paper, the parameters take their default values except the parameter on horizontal axis. The type of simulated data is short integer with 16bits. B+tree is used to build UB-tree with fanout 4, and the order of one leaf node is 350(注2). Two grid tables are built for first 8 attributes. 4 attributes use one grid table. Each space is divided into 5 parts. We implemented a workload generator according to a workload specification. The events are created randomly. The hardware platform is Sun Fire 4800 with 4 900MHz CPUs and 16G memory. The OS is Solaris 8. We will do kinds of evaluations by changing one parameter and fixing other parameters.

To compare with the access-predicate based algorithm in the experiment, we extend the scenarios in the Le subscribe system [10]: the total number of attributes, which the subscription can choose, is 32. The domain size for each attribute is 10000. And the number of the fixed attributes is from 8 up to 28, the number of the unfixed attributes is only 1. For all the scenarios, in the fixed attributes, 2 of them are in equality. And for the one unfixed attribute, it is in equality. The number of subscription is 100000, and the number of event is 100.

Our simulated environment is a reasonable in the publish/subscribe environment. Notice that, there exists data skew for default distributions of equality predicates and inequality predicates after dimension transform as introduced in section 3. For each attribute after dimension transform, 80% equality predicates will locate on diagonal line and 10% half-interval predicates will locate on the border of 2D space.

5.2 Evaluation Results

Fig.7(a) shows that both original UB-tree and optimized UB-tree have better scalability. Fig.7(b) shows that dimension transform based algorithms are insensitive to the changes of selectivity. The reason is data skew about dimension transform because the access

---

（注1）：Leaf node of UB-tree based on B+-tree.

（注2）：Because binary search tree is the fast search algorithm based on tree structure in main memory, it has fanout 2. So the fanout is set as smaller as the implementation is allowed here. 350 means the maximum number of objects kept in a node is 700 where the best performance can be reached. Please refer to Fig.4(b).

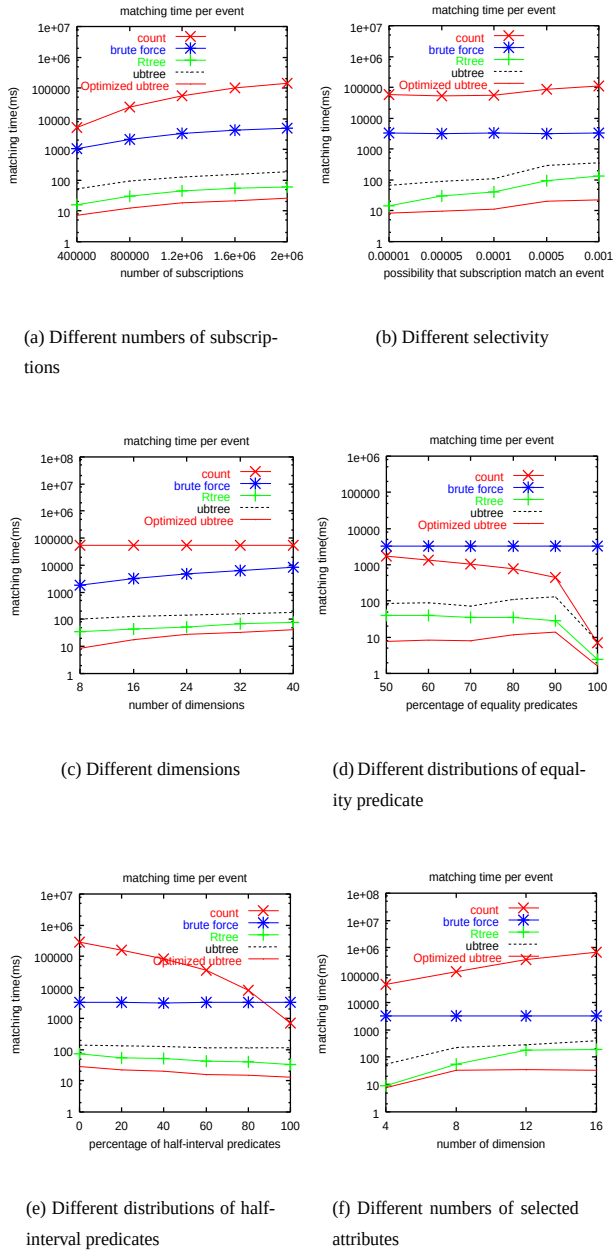
表 1　Simulated parameters

Parameter	Value range	Default value
Number of subscriptions	0-2000000	1200000
Number of dimensions (attributes)	8-40	16
Possibility of one attribute is used in subscription	0-100%	the first 3 attribute is 100%, the 4th-8th attributes is 70%, the others are 1%
Ratio of one subscription is satisfied	0-100%	0.01%
Ratio of equality predicates is used	0-100%	80%
Ratio of half-interval predicate among non-equality predicates	0-100%	50%

number of leaf node(Z-region) changes 4 times(from 79 to 221) for one event filtering while selectivity changes 100 times (from 0.00001 to 0.001). Fig.8(a) shows the same change trend of objects number in Z-regions and number of results in the same test.

Fig.7(c) shows the performance with different dimensions. The reason that counting algorithm performance is stable is that only the first 8 attributes have higher possibility to be used. Because the filtering operation is applied for every dimension, the time of dimension transform based algorithms and bruteforce become higher with the increment of dimensions. And the time of optimized algorithm grows a little quickly because the time saved by using of grid table is not dependent on the total attribute number of data space. Fig.7(d) shows performance of counting algorithm heavily depends on the distribution of equality predicates. Fig.7(e) shows the performance with different distributions of half-interval predicates. Again the performance of counting algorithm changes sharply. Fig.7(f) shows that with number increment of the selected attributes, time of counting algorithm rises when more and more one-dimensional indexes are used.

The representative effectiveness of grid table is shown in Fig.8 with different selectivity and dimensions. It also shows the changing of the input before and after using grid table. We can find that the lower the selectivity is, the higher the effectiveness of grid table is. The input amount is reduced by 1-2 orders of magnitude here. The difference is more than one order of magnitude that means the curse of dimensionality doesn't occur in our simulated environment. As introduced above, theoretically selectivity of such kind of range query decreases exponentially for uniformly distributed data when size of the dimension increases. But in practice, the selectivity is dependent on the definition of subscriptions like our simulated environment. The data skew occurs after dimension transform and the data mainly distribute on the hyper plane determined by border of space and its diagonal line, that's reason the cure of dimensionality



(a) Different numbers of subscriptions

(b) Different selectivity

(c) Different dimensions

(d) Different distributions of equality predicate

(e) Different distributions of half-interval predicates

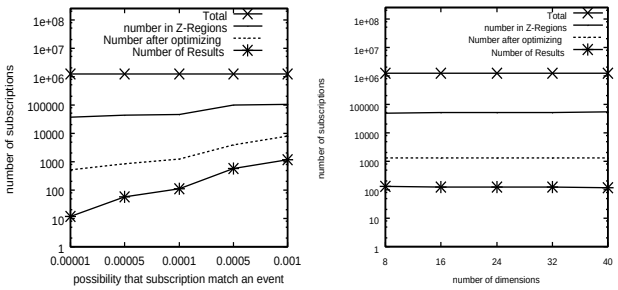
(f) Different numbers of selected attributes

図 7　Results of different simulations

is postponed.

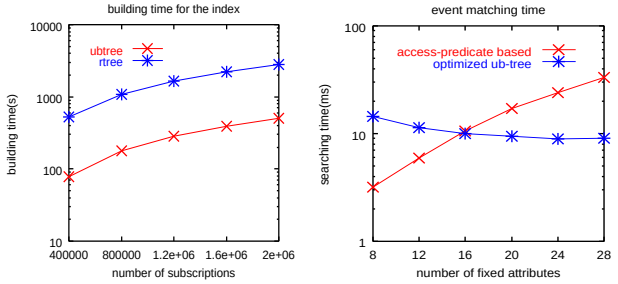
During the experiment, we find the original R-tree is much faster than the R-tree with dimension transform. So we choose the original R-tree here. According to the results in Fig.7, we find that the performance of R-tree is located between the performance of original UB-tree and optimized UB-tree. The optimized UB-tree is 3 times faster than the R-tree in the event filtering process. And from the Fig.9(a), we can find the index building time of the R-tree is much higher than that of UB-tree. This is another evidence of the poor insertion performance of the R-tree compared with UB-tree. So we think UB-tree is a better choice for publish/subscribe system, which needs dynamic maintenance also.

We also make some comparisons with the access-predicate based algorithm. In our experiment, we don't concern about the mem-



(a) Different selectivity (b) Different dimensions

図 8 Effectiveness of grid table



(a) Indices building time for UB-tree and R-tree

(b) Searching time for the UB-tree and access-predicate based algorithm

図 9 other experiments

ory space used. So we rebuild the access-predicate based algorithm to get the best performance. From the result in Fig.9(b), we find that with the number of the fixed attribute grows, the performance access-predicate based algorithm becomes worse quickly. For our solution, it becomes a little better when the dimension (in the reasonable range here) increased. And this is the performance comparison based on the similar scenario in the Le subscribe system [10]. With the limitation of the Le subscribe system, we can't use the access-predicated algorithm on the scenarios, described in the Table1, which is more practical.

6. Conclusion

In this paper, we proposed an UB-tree based predicate index for publish/subscribe system by dimension transform. It is more practical According to above experiments, our proposed optimized index is 4 order of magnitude faster than counting algorithm, and 1 order of magnitude faster than UB-tree without optimizing, and more than 3 times faster than the R-tree solution. For the high dimensional data, it can get more benefit. So we can say that our proposed index structure is efficient under reasonable size of dimension (Maximum is 40, default is 16).

文 献

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, T. D. Chandra. Matching Events in a Content-based Subscription System. Eighteenth ACM Symposium on Principles of Distributed Computing(PODC), 1999:53-61

[2] R. Bayer. The Universal B-tree for multidimensional Indexing. Technical Report TUM-I9637, November 1996

[3] R. Bayer and V. Markl. The UB-tree: Performance of Multidimensional Range Queries. Technical Report TUM-I9814, Institut fr Informatik, TU Mnchen, 1997.

[4] N. Beckmann, H.-P. Kriegel, R. Schneidar, B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD 1990:322-331

[5] J. L. Bentley. Multidimensional binary search trees used for asociative searching. Commun. ACM 18:509-517, 1975

[6] S. Berchtold, D. A. Keim. High-Dimensional Index Structure: Database Support for Next Decade's Application Tutorial. ICDE 2000

[7] S. Chandrasekaran, M. J. Franklin. Streaming Queries over Streaming Data. Proceedings of the 28th VLDB Conference, Hong Kong, 2002

[8] Y.-J. Chiang, R. Tamassai, "Dynamic Algorithms in Computational Geometry". Technial Report CS-91-24, Dept. of Computer Science, Brown Univ., 1991

[9] M. deBerg, M. V. Kreveld, M. Overmars, O. Schwarzkopf. "Computational Geometry-Algorithms and Applications". ISBN 3-540-65620-0 Springer. 1998

[10] F. Fabret, H.Arno Jacobsen, F. Llirbat, J. Pereira, K. A.Ross, D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. ACM SIGMOD 2001

[11] R. Fenk, V. Markl, R. Bayer. Inerval Processing with the UB-tree. IDEAS 2002:12-22

[12] V. Gaede, O. Gnther. Multidimensional Access Methods. In Computing Surverys 30(2):170-231. ACM Press, 1998

[13] R.Gruber, B. Krishnamurthy and E. Panagos. The architecture of the ready event notification service. In Proc. of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop, 1999.

[14] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. ACM SIGMOD 1984:47-57

[15] E. N. Hanson, M. Chaaboun, Chang-Ho, Y. Wang. A Predicate Matching Algorithm for Database Rule Systems. ACM SIGMOD 1990

[16] E. N. Hanson, T. Hohnson. Selection Predicate Indexing for Active Database Using Interval Skip List. TR94-017. CIS department, Univeristy of Florida, 1994

[17] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha. Scalable Trigger Processing. ACM SIGMOD 1999

[18] A. Hinze, S. Bittner. Efficient Distribution-Based Event Filtering. International Workshop on Distributed Event Based Systems. Austrai July 2002

[19] H. A. Jacobsen, F. Fabret. Publish and Subscribe Systems. Tutorial. ICDE 2001

[20] S. Madden, M. Shah, J. Hellerstein, V. Raman. Continuously Adaptive Continuous Queries(CACA) over Streams. ACM SIGMOD 2002

[21] V. Markl, M. Zirkel, and R. Bayer. Processing Operations with Restrictions in RDBMS without External Sorting: The Tetris Algorithm. Sydney, Austrialia, pages 562C571. IEEE Computer Society, 1999.

[22] V. Markl. Processing Relational Queries using a Multidimensional Access Technique. PhD thesis, DISDBIS, Band 59, Infix Verlag, 1999.

[23] J. Pereira, F. Fabret, F. Llirbat and D. Shasha. Efficient matching for web-based publish/subscribe systems. In Proc. of the Fifth IFCIS International Conference on CoopIS ¨ 2000, Eilat, Israel, September 2000.

[24] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhard, and R. Bayer. Integrating the UB-tree into a Database System Kernel. In Proceedings of International Conference on Very Large Data Bases, 2000, Cairo, Egypt, 2000.

[25] H. Samet. The quadtree and related hierarchical data structure. ACM

Computer Survery. 16(2):187-260, June 1984

[26] T. K. Sellis, N. Roussopoulos, C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. VLDB 1987:278-291

[27] B. Wang, W. Zhang, M. Kitsuregawa. UB-tree Based Efficient Predicate Index with Dimension Transfrom for Pub/Sub System. In DASFAA ¨ 04, 2004.

[28] T. W.Yan, H. Garcia-Molina. The SIFT Information Dissemination System. In ACM TODS 2000