UB-tree Based Efficient Predicate Index with Dimension Transform for Pub/Sub System

Botao Wang[1], Wang Zhang[1], and Masaru Kitsuregawa[1]

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

Abstract. For event filtering of publish/subscribe system, significant research efforts have been dedicated to techniques based on multiple one-dimensional indexes built on attributes of subscription. Because such kinds of techniques are efficient only in the case that operators used in predicates are equality operator (=) and attributes used in subscriptions are fixed, the flexibility and expressiveness of publish/subscribe system are limited.
Event filtering on subscriptions which include not only equality operator (=) but also non-equality operators (<=, =>) without fixed attributes, is similar to query in high dimensional data space. In this paper, considering dynamic maintenance and space efficiency of publish/subscribe system, we propose an index structure for event filtering based on UB-tree. There, by dimension transform, the event filtering is regarded as high dimensional range query. The feasibility of the proposed index is evaluated in simulated publish/subscription environment. Results show that in almost all the cases, the performance our proposed index is 4 order of magnitude faster than counting algorithm. Because our index can support both equality operator (=) and non-equality operators (<=, >=), we can conclude that our proposal is efficient and flexible for event filtering of publish/subscribe system under reasonable size of dimension.

1 Introduction

Publish/subscribe systems provide subscribers with the ability to express their interests in an event in order to be notified afterwards of any event fired by a publisher, matching their registered interests [12]. Index is crucial for efficient event filtering on subscriptions.

As far as we know, in the context of publish/subscribe system, the predicate indexing techniques have been widely applied [9] [10] [13] [17] [18] [22] [27] and significant research efforts have been dedicated to the techniques based on multiple one-dimensional indexes built on attributes of subscriptions. Because such kinds of techniques are efficient only in the cases that operators used in predicates are equality operator (=) and attributes used in subscriptions are fixed, the flexibility and expressiveness of publish/subscribe system are limited.

Predicates with non-equality operators <= or >= and subscription with unfixed selectable attributes are indispensable for subscriber to precisely define

their subscriptions. Although event filtering on such kinds of subscriptions can be regarded as queries on high dimensional space [20], according to our survey, there are very few research efforts on applying multidimensional index structure to publish/subscribe system. In this paper, we examine the feasibility of transforming point enclosure query into range query by dimension transform, where event filtering is regarded as high dimensional range query. Considering dynamic maintenance and space efficiency of publish/subscribe system, we design an UB-tree based predicate index for event filtering of publish/subscribe system.

The rest of this paper is organized as follows. Section 2 introduces the background and UB-tree. Section 3 introduces dimension transform for publish/subscribe data. In Section 4, we describe our optimization methods to improve performance. In Section 5, our proposal is evaluated in simulated publish/subscribe environment. Section 6 discusses the related work. Finally, conclusion is given in Section 7.

2 Background and UB-tree

2.1 Background

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 hypercubes.
- 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 exist lots of incomplete subscription hypercubes which overlap each other heavily. So it is hard to make use of multidimensional index structure directly to build efficient index on these subscriptions hypercubes for point enclosure query. For this reason, we choose range query and do dimension transform in our design.

As introduced in [7] [8] [11] [15], 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 [16] and R*-tree [4] suffer from region splitting and merging while updating index. R+-tree [26] cannot guarantee a minimal storage untilization; KD-tree [5] is sensitive to the order in which the points are inserted; quadtree [25] is unbalanced and sensitive to data density.[1] UB-tree [2] [14] [24] is designed to perform multidimensional range query. It is a dynamic

[1] It is out of the range of this paper to compare kinds of multidimensional index structures. We just want to note why UB-tree is selected in this paper. For the details of kinds of multidimensional index structures, please refer to [7] [8] [11] [15] which are comprehensive surveys and text book on this topic.

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.

2.2 UB-tree

The basic idea of the UB-tree [2] [3] [14] [21] [24] is to use a space filling curve to map a multidimensional universe to one dimensional space. Z-curve (Fig.1 [2]) is used to preserve multidimensional clustering. A Z-address (called Z-value

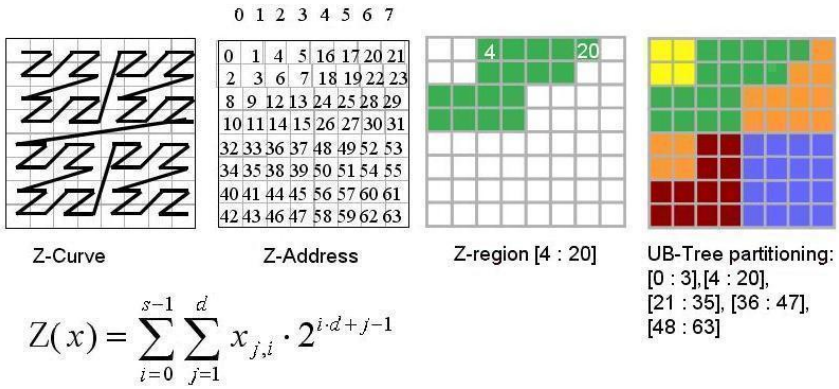


Fig. 1. Two-Dimensional Z-curve, Z-addresses Z-region and Z-partition

too) is the ordinal number of the key attributes of a tuple on the Z-curve, which can be computed efficiently by bit-interleaving of coordinates corresponding to dimensions. A standard B-tree is used to index the tuple by taking the Z-address of the tuple as key. The fundamental innovation of UB-Tree is the concept of Z-region (Fig.1), to create a disjunctive partitioning of the multidimensional space. Its Z-region partitioning adapts well to the data distribution, i.e., densely populated areas are finer partitioned. This allows very efficient processing of multidimensional range queries.

While processing a multidimensional range query, a UB-tree retrieves all Z-regions, which are properly intersected by the query box. Due to the mapping of the multidimensional space to Z-values, this results in a set of intervals on the B-Tree where Z-values are stored. The main task of the UB-tree range query algorithm is to calculate efficiently the set of one-dimensional intervals corresponding to Z-regions stored in B-tree in the sequence of Z-curve. After getting intersected intervals, the results will be filtered out from all objects in the Z-regions according to the range box. For more details of the search algorithm, please refer to [24].

[2] Extracted from http://mistral.in.tum.de/results/presentations/ppt/ubtree.ppt.

3 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

$$(IMin <= Istart <= Evalue) \text{ AND } (Evalue <= Iend <= IMax)$$

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.2. For one attribute, after transform from 1D to 2D, event becomes range

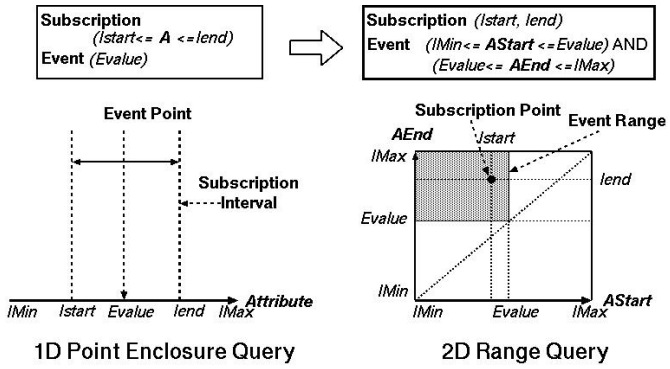


Fig. 2. Transform 1D point enclosure query to 2D range query

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. 2
- 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 incompleted subscription, only parts of attributes are used. Because subscription is a conjunction of predicates, for the attributes not be used in the subscription, their realted predicates are considered to be $TRUE$. Logically $TRUE$ can be represented as $IMin <= A <= IMax$. Then $TRUE$ is a point with constant value $(IMin, IMax)$.

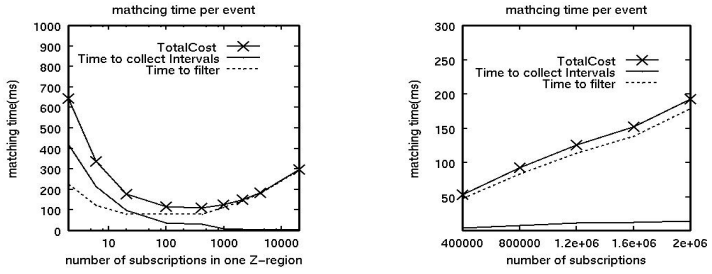
– Dead Space. Because $Istart <= Iend$, there is no data located in the space under diagonal space. It's called dead space.

For above properties, even the data in 1D space are distributed uniformly, it is very possible that data skew occurs after transform. 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.6-d and Fig.6-e.

4 Performance Improvement

With the emergence of cheap computers having very large size memory, more and more algorithms will run in main memory. Considering performance, our predicate index is designed to be executed in main memory. In this section, after analyzing workload distribution of UB-tree's range search, we will propose optimizing methods focusing on reducing the cost of filtering operation.

4.1 Workload Distribution of Range Query



(a) (b)

Fig. 3. Workload distribution

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.3-a (Please refer to Table.1 in Section 5.1 for detail information of test environment).

From Fig.3-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 optimization work in this range. As shown in Fig.3-b (the maximum number is 700 there), there is a big difference of the workloads of this two steps. Cost on filtering operation accounts for major part of the total cost. 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 is to reduce the cost related to filtering operation. Two ways 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).

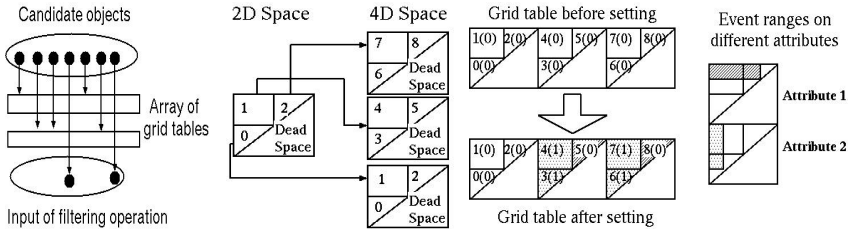
4.2 Reduce Input of Filtering Operation

The basic idea is shown in Fig.4-a, an array of grid tables is created dynamically to reduce input of filtering operation. There 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 not located in dead space. It records whether each cell is covered or not by input 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.4-b. After dimension transformation, N attributes corresponds to one 2ND space. The left side of Fig.4-b shows an example of cell ID setting in 2D space. The right side of Fig.4-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 straightfoward and skipped here.

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

- While inserting subscription point, its corresponding cell ID is calculated and kept inside the index with 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 is set to 1, otherwise left to 0 as shown in Fig.4-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 exponently with number of attributes, building one grid table on all attributes is impractical. In order 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 groupes to reduce the exponent incrementation of size caused by the 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.4-a. In this case, corresponding different groups, multiple grid table should be created and checked while do event filtering.



(a) Basic idea (b) Cell Ids in different spaces (c) Value setting of grid table in 4D space

Fig. 4. The way to reduce input of filtering operation and Grid Table

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 input event range by looking up grid table(s). If one of the cell values is 0, it means the corresponding cell doesn't intersect with input range, and then there is no need to send this candidate object to 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 intersection of two orthogonal partionings: space-filling curve and grid file. Even the adding of grid table is 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.7.

4.3 Improve Performance of Filtering Operation

As introduced in Section 2.2, 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 expensive bitwise operation. In order to improve the performance of filering 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[3] in UB-tree is extended as shown in Fig.5

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

[3] Leaf node of UB-tree based on B+-tree.

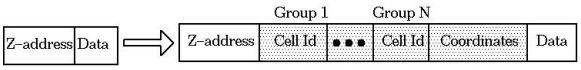


Fig. 5. Entry extension of UB-tree Node

is used in many publish/subscribe systems and 2) bruteforce considering about the curse of dimensionality.

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[4]. Two grid tables are built for first 8 attributes. Each grid table corresponds 4 attributes. 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.

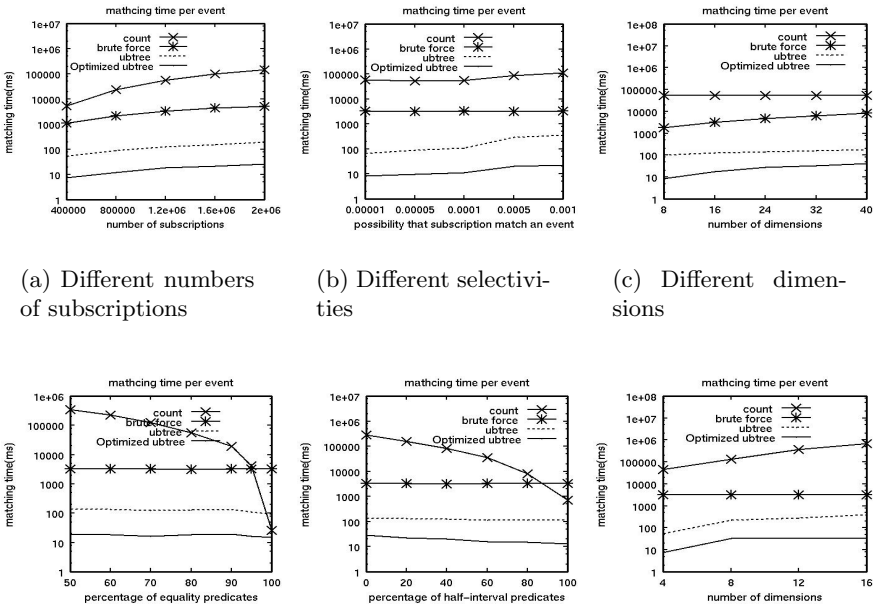
Table 1. Simulated parameters

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%

5.2 Evaluation Results

Fig.6-a shows that both original UB-tree and Optimized UB-tree have better scalability. Fig.6-b shows that dimension transform based algorithms are insen-

[4] 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.3-b.



(a) Different numbers of subscriptions

(b) Different selectivities

(c) Different dimensions

(d) Different distributions of equality predicate

(e) Different distributions of half-interval predicates

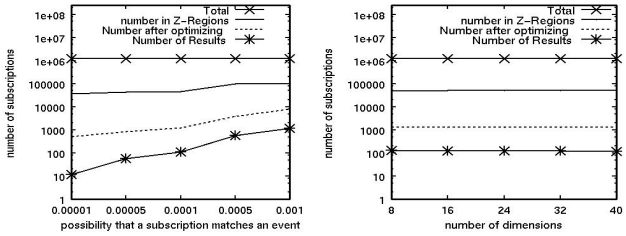
(f) Different numbers of selected attributes

Fig. 6. Results of different simulations

sitive to the changes of selectivities. Fig.6-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.6-d shows performance of counting algorithm heavily depends on the distribution of equality predicates. Fig.6-e shows the performance with different distributions of half-interval predicates. Again the performance of counting algorithm changes sharply. Fig.6-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.7 with different selectivities and dimensions. It also shows the amount 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 order of magnitude here.

According to above results, we can find that in almost all the cases in our simulated environment, our proposed optimized index is 4 order of magnitude faster than counting algorithm, and 1 order of magnitude faster than UB-tree without optimizing. So we can say that our proposed index structure is efficient under reasonable size of dimension (Maximum is 40, default is 16).



(a) Different selectivities

(b) Different dimensions

Fig. 7. Effectiveness of grid table

6 Related Work

A lot of algorithms related to event matching have been proposed. Some are proposed for publish/subscribe systems [1] [13] [19] [27] and continuous queries [9] [10] [22]; Some are proposed for active database [17] [18].

Predicate indexing techniques have been widely applied. There, a set of one-dimensional index structures to index the predicates in the subscriptions. Mainly, there are two kinds of predicate indexing based algorithms: counting algorithm [27] and Hanson algorithm [17] [18]. They differ from each other by whether or not all predicates in subscriptions are placed in the index structures. According to [20], it is hard to judge which one is better because counting algorithm depends on average probability of a predicate to be satisfied and Hanson algorithm depends on average probability of matching each access (selected) predicate. [27] is an Information Dissemination System (IDS) for document filtering. There, the predicate index is an inverted list which is built based on the vocabularies used in predicates. In [17] [18], algorithms related to rule management were proposed. The key component of the algorithm in [17] is the interval binary search tree (IBS-tree) for each attribute. The IBS-tree is designed for efficient retrieval of all intervals that overlap a point, while allowing dynamic insertion and deletion of intervals. "Expression Signature" is designed to group subscriptions and share computation in [18]. In [13], Hanson algorithm is extended by dynamic clustering. High performance is gotten compared to counting algorithm, but there fixed attributes are predefined and value domain of attributes is limited to 5-100.

The testing networking based techniques initially pre-processes the subscription into a matching tree. Different from predicate index, [1] and [19] built subscription trees based on subscription schema. In [1], each non-leaf node contains a test, and edges from the node represent results of that test. The test and result correspond to predicate. A leaf node contains a subscription. The matching is to walk the matching tree by performing the test prescribed by each node and following the edge according to the result of test. if number of matched subscription(s) is greater than one, multiple paths will be walked. In [19], Profile (subscription) tree is built, the height of tree is number of attributes defined in subscription schema. Each non-leaf level corresponds to one attribute of event schema. Each attribute domain is divided into non-overlapping subranges by the value of predicate. One leaf node contains multiple subscriptions whose predicates are satisfied by the values of attributes in the subranges. There is only a single path to follow in order to find the matched subscriptions.

Event filtering is one critical step of continuous queries. In [10], Expression signature is used to group queries for computation sharing. In [22], four data structures: a greater-than balanced binary tree, a less-than balanced binary tree, an equality hash-table, and an inequality hash-table were built. Counting algorithm was used in [10] and [22]. In [9], predicate index is built based on Red-Black tree, there algorithm is similar to bruteforce that scans the red-black for event filtering each time.

7 Conclusion

In this paper, we proposed an UB-tree based predicate index for publish/subscribe system by dimension transform. The index was evaluated by simulated publish/subscription environment with maximum 40 dimensions and 2 million subscriptions. The results show that in almost all the cases in our simulated environment, the performance of our proposed index, is 4 order of magnitude faster than counting algorithm, 1 order of magnitude faster than UB-tree without optimizing. Because the index can support event filtering on subscriptions which use both equality operator (=) and non-equality operators (<=, >=) without fixing attributes, we can conclude that our proposal is efficient and flexible for event filtering of publish/subscribe system under reasonable size of dimension.

References

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, V. Markl. The UB-Tree: Performance of Multidiemnsional Range Queries. Technical Report TUM-I9814, June 1998

4. N. Beckmann, H.-P. Kriegel, Ralf Schneidar, Berhhard 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, C. Bohm, H.-P. Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimesionality. ACM SIGMOD 1998:142-153
7. S. Berchtold, D. A. Keim. High-Dimensional Index Structure: Database Support for Next Decade's Application Tutorial. ICDE 2000
8. M. deBerg, M. V. Kreveld, M. Overmars, O. Schwarzkopf. "Computational Geometry-Algorithms and Applications". ISBN 3-540-65620-0 Springer. 1998
9. S. Chandrasekaran, M. J. Franklin. Streaming Queries over Streaming Data. Proceedings of the 28th VLDB Conference, Hong Kong, 2002:203-214
10. J. Chen, D. J. DeWitt, F. Tian, Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. ACM SIGMOD 2000:379-390
11. Y.-J. Chiang, R. Tamassai, "Dynamic Algorithms in Computational Geometry". Technial Report CS-91-24, Dept. of Computer Science, Brown Univ., 1991
12. P. T. Eugster, P. Felber, R. Guerraoui and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. Technical Report 200104, Swiss Federal Institute of Technology
13. F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. ACM SIGMOD 2001:115-126
14. R. Fenk, V. Markl, R. Bayer. Inerval Processing with the UB-Tree. IDEAS 2002:12-22
15. V. Gaede, O. Gnther. Multidimensional Access Methods. In Computing Surverys 30(2):170-231. ACM Press, 1998
16. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. ACM SIGMOD 1984:47-57
17. E. N. Hanson, M. Chaaboun, C.-H., Y.-W. Wang. A Predicate Matching Algorithm for Database Rule Systems. ACM SIGMOD 1990:271-280
18. E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha. Scalable Trigger Processing. ACM ICDE 1999:266-275
19. A. Hinze, S. Bittner. Efficient Distribution-Based Event Filtering. International Workshop on Distributed Event Based Systems. Austrai July 2002:525-532
20. H. A. Jacobsen, F. Fabret. Publish and Subscribe Systems. Tutorial. ICDE 2001
21. V. Markl. MISTRAL:Processing Relational Queries using a Multidimensional Access Tecnnique. Ph.D. Thesis, TU Munchen, 1999, published by infix Verlag, St.Augustin. DISDBIS 59, ISBN 3-89601-459-5, 1999
22. S. Madden, M. Shah, J. Hellerstein, V. Raman. Continuously Adaptive Continuous Queries(CACA) over Streams. ACM SIGMOD 2002:49-60
23. R. Motwani. Models and Issues in Data Stream Systems. Invited Talk. PODS 2002
24. F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, R. Bayer. Intergrating the UB-tree into a Database System Kernel. VLDB 2000:253-272
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. T. W. Yan, H. Garcia-Molina. The SIFT Information Dissemination System. In ACM TODS 24(4):529-565 1999