複数入力を持つ拡張 R-tree 検索アルゴリズムを用いた PUB/SUB システムの平均応答時間の改善

Adaptively Improving Average Response Time of Pub/Sub System Based on Extended R-Tree Search Algorithm with Multiple Inputs

王　波涛♠ 張　旺♦　喜連川　優♠

Botao WANG♠　Wang ZHANG♦

Masaru Kitsuregaw♠

Publish/subscribe システムはユーザに対して興味があるイベントを随時送信している。イベントが起こる確率は時間によって変化し、予測ができない。一定の時間内にイベントが何も起こらなかったり、複数のイベントが同時に発生することは良く起こる。複数のイベントが同時に発生した時、その平均レスポンスタイムは作業の処理順序に依存する。本論文は始に R-tree を用いた複数イベントのためのフィルタリングアルゴリズムを提案する。各々のイベントの負荷に関する情報を用いることで、処理時間が短いイベントから処理を行い、平均レスポンス時間を向上させる。さらに、インデックスが動的に変化する環境下で、サイズの異なるイベント集合のための自己適応モデルの提案と評価を行う。

Publish/subscribe system captures the dynamic aspect of the specified information by notifying users of interesting events as soon as possible. The rate of event arriving is time varying and unpredictable. It is very possible that no event arrives in an unit time and multiple events arrive in another unit time. When multiple events arrive at same time, the average response time of events filtering depends on the sequence of filtering events which have different workloads. In this paper, we first propose an event filtering algorithm with multiple inputs (multiple events) based on R-tree. With information of relative workload of each event, event by event filtering can be executed with short-job first policy to improve average response time of multiple jobs. Further a self-adaptive model is proposed and evaluated to filter set of events with different sizes on dynamically changing index.

1. Introduction

Efficient event filtering with a faster response time is very important for event processing with multiple steps like event join, and is one of important factors to provide good service for subscribers. Generally the rate of event arriving is time varying and unpredicatable. For example, traffic monitoring, ticket reservation, internet access, stock price, etc. In contrast to stable rate, it's very possible that a batch of events arrive in one unit time and no event arrives during another unit time.[1]

In the context of publish/subscribe system, even many index techniques such as multiple one-dimensional indexes based [1], [2], [3], [4], [5] and multidimensional index based [6], [7] have been proposed for event filtering, all these techniques are designed to filter events one by one. They can not deal directly with batch events in a fast average response time if those events arrive at same time with different workloads. Meanwhile, we found that event filtering based on multidimensional index [6] [7] is more efficient and flexible than that based on multiple one-dimensional indexes. In order to improve average response time of event filtering in the case that multiple events arriving at the same time, in this paper, we first propose a R-tree based event filtering algorithm to improve average response time of filtering multiple events arriving at same time. There a cost model to estimate relative workloads of these events is built to arrange the filter order of these events with Short-Job First (SJF) policy. Further, because the number of events arriving at the same time and index size change dynamically, an adaptive model is proposed to filter events with average response time always same as or close to the possible best time.

The rest of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 introduces the algorithm to improve average response time. Section 4 proposes the adaptive model. In Section 5, the event filtering algorithm and the adaptive model are evaluated and analyzed in a simulated environment. Section 6 discusses the related work. Finally, conclusion and future work are given out in Section 7.

2. Background and Motivation

As introduced in [6], [7], multidimensional index (R-tree or UB-tree) based event filtering is feasible[2] and is much efficient and flexible than that based on the popular multiple one-dimensional indexes based technique -count algorithm [3], [4]. Meanwhile, SJF is one well-known policy used to improve average response time while scheduling multiple jobs. The critical thing is to estimate workloads correctly.

♠ 正会員　東京大学生産技術研究所
{botaow,kitsure}@tkl.iis.u-tokyo.ac.jp
♦ 学生会員　東京大学情報理工学系研究科博士後期課程
zhangw@dblab.is.ocha.ac.jp

♠ 正会員　東京大学生産技術研究所
{botaow,kitsure}@tkl.iis.u-tokyo.ac.jp
♦ 学生会員　東京大学情報理工学系研究科博士後期課程
zhangw@dblab.is.ocha.ac.jp

[1] Even logically for most of the events, there exist absolutely different arriving times, in this paper, we regards the events arriving in the same unit time as the events arriving at same time.

[2] For details, please refer to [6], [7].

Because UB-tree partitions space with space filling curve, original UB-tree's search algorithm is depth-first and no Minimum Bounding Rectangle (MBR) information is required and kept inside its nodes, it's not easy to use MBR of event to calculate the number of multiple search paths at specified middle levels of original UB-tree without accessing leaf nodes. The structure of R-tree doesn't have this problem. For this reason, we choose R-tree as the basis of our proposal in this paper.

3. Improve Average Response Time

3.1 Basic Idea

The basic idea is that for the multiple events arriving at same time, the relative workloads are estimated respectively. The workload is defined as the number of search at a specified level.

There are two steps for the multiple events filtering algorithm (called BatchSearch later) 1) first based on their different numbers of search paths on R-tree by search R-tree to a specified level 2) and then do filtering (R-tree search from the specified level) event by event with SJF policy with assumption that the more the number is, the higher the workload is.

Because the algorithm used at step 2 is similar to original R-tree search algorithm, we introduce only the data structure and algorithm to estimate the workload in step 1. We assume the reader has enough background about R-tree.

3.2 Data Structures to Estimate Workload

WorkloadTable is an array of items with structure shown in Fig.1. Each item corresponds to one event. The Workload in Fig.1 is the number of nodes located at the ending of search paths stopped at the specified level. WorkloadTable is filled and sorted by function EstimateWorkload as introduced as Fig.2

A data structure named IntersectBuffer Fig.2 is used to record events whose MBRs intersect with those of items of one R-tree node which is located in the paths from root to the specified level.

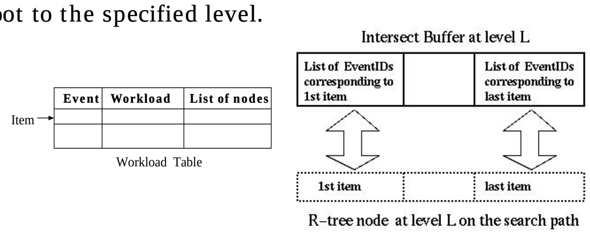


Fig.1 Data Structure used to Estimate Workload

The number of items in one intersect buffer is same as that of one R-tree node. The 1st item of intersect buffer corresponds the 1st item of R-tree node. The content of the 1st item of IntersectBuffer is the list of eventIDs whose MBRs intersect with that of the 1st item of the R-tree node. The others items have similar contents.

The algorithm to fill WorkloadTable is shown in Fig2. In function EstimateWorkload, line 1 initializes the IntersectBuffer of level 0 where there is only one item with one pointer pointing to the bf Root node and all events are assumed to intersect with MBR of this item. The SJF can not be benefited by the main algorithm BatchSearch with f Level valued 0, because all workloads have same value 1.Line 2 call a recursive function BatchIntersect to fill WorkloadTable, level 1 means checking from root node. Line 3 sorts WorkloadTable according to the number of search paths in ascending order.

```
Begin EstimateWorkload(Root, EventArray, EventNumber, Level)
1  Set IntersectBuffer of level 0;
2  BatchIntersect(Root, EventArray, 1, Level);
3  Sort WorkloadTable in ascending order;
End EstimateWorload

Begin BatchIntersect(CurrentNode, EventArray, CurrentLevel, Level)
1  Get the item which CurrentNode corresponds to from IntersectBuffer
2      of CurrentLevel-1 and read all eventIDs into EventList;
3  IF CurrentNode is leaf Node or CurrentLevel >= Level
4    Add WorkloadTable with CurrentNode and EventList;
5  ELSE
6    Reset IntersectBuffer of CurrentLevel;
7    For each item in CurrentNode
8      For each event in EventList
9        Check MBR intersection of current item with current event
10        If intersect
11          Insert eventID into the corresponding item of
12                IntersectBuffer of CurrentLevel;
13        ENDIf
14      ENDForLoop
15    ENDForLoop
16    For each item of CurrentNode
17      BatchIntersect(Item's SubNode, EventArray, CurrentLevel+1, Level);
18    ENDForLoop
19 ENDIf
End BatchIntersect
```

Fig.2 Algorithm to Estimate Workload

In function BatchIntersect, line 1-2, read one item from IntersectBuffer of last level (the level nearer to root) and gets all event IDs kept in the item. That item corresponds to the item of R-tree node at last level which includes the pointer pointing to CurrentNode. Line 3 checks the ending condition of recursive search and line 4 adds the WorkloadTable with the event IDs gotten at line 1-2 and CurrentNode. Line 6-15 fill intersect buffer of CurrentLevel. Line 16-18 search next level by accessing subnodes of CurrentNode.

4. Model of Adaptive Search

For same batch events, the performance changes with different possible values of R-tree Level. At the same time, the number of events arriving at same time is not fixed, the size of index changes dynamically also. In this section, we will propose a self-adaptive model in order to filter kinds of events with average response time same as or close to the possible best time.

4.1 Performance Analysis

While filtering multiple events arriving at same time, time cost to estimate workloads is overhead compared to the processing with original R-tree search algorithm event by event without the workload estimation. The overhead becomes larger with the value increment of Level. At the same time, because the higher the Level is, the more accurate of workload estimation is, the efficiency of SJF become more and more better with the value increment of Level also. For the same batch events, the average response time based on the BatchSearch is a function of Level L. Their relationships can be described in the left part of Fig.3 Because it has shape of concave as shown on the leaf of Fig.3 with mark "Ideal and logical". The best level exits for the batch events with same event number and it should be located between level 0 to level TreeHeight-1. The best level changes for different number of input events and size of index.

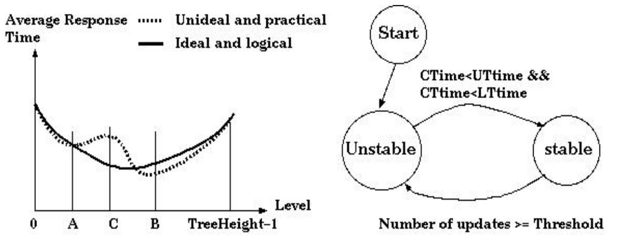


Fig.3 Changing of Average Response Time and Adaptive Model

In order to get best average response time, the BatchSearch should run with Level valued best level.

4.2 Adjust Best Level Dynamically According to Statistic Information

The adaptive model is shown on the right of Fig.3, it is built for filtering of batch events with same event number arriving at same time. For events with different sizes (numbers), their statuses will be kept in different buffers, for example, entries of a status buffer array for batch events with different sizes. The main function of the model is to adjust best level dynamically for filtering of different batch events with different numbers and sizes of index.

If the current level is best level, we call the system is stable. In stable status, the BatchSearch is executed with Level valued best level. The number of update operations (insert and delete) is monitored in stable status. After some update operations, the height of index tree or data distribution of index maybe be changed, it is necessary to check the best level or adjust it if it changes. The system becomes unstable then. The Threshold shown in Fig.3 is the number to determine the time when the system enters unstable status from stable status.

In unstable status, the best level can be checked by trying all levels with same events naively, but it's not acceptable for an dynamic system in practice. This overhead is not neglected for a higher index tree or batch events with larger number.Our model is that check current level and its upper and lower levels (totally 3 levels) based on the "Ideal and logical" changes of performance.

In unstable status, for events arriving batch by batch with same size at different time, BatchSearch process these batches of events with value changed in a way of round-robin loop.

The input contents of BatchSearch change in the sequence of arriving time like

(EventArrayNo1, CurrentLevel),
(EventArrayNo2, CurrentLevel -1),
(EventArrayNo3, CurrentLevel+1),
...,
(EventArrayNo3N, CurrentLevel+1)

N is the counter of loop.So in unstable status,system does events filtering with Level} values same as or close to the best level.

The average response times of three different levels (called CTime, UTime, LTime in Fig.3 which correspond to the average response time of current level, upper level,lower level) are summed up and checked when the loop ends. If CTime < UTime && CTime < Dtime) is true, the system will enter stable status, because the current level is the best level. Otherwise, moves the current level towards to the direction of best level according to the "Ideal and logical}" changes of BatchSearch performance and restarts a new loop.

Because for every events filtering ,the input EventArray of BatchSearch is different, so it's possible that the time gotten at different levels doesn't change "ideally and logically" when the loop counter N is very small, for example 1. In this case as the line of " Unideal and practical" shown in Fig.3, it is possible for system to enter stable status, even the current level(A) is not best level(B). It is also possible that

CTime > UTime) && CTime > LTime

is true as shown at level(C). The adaptive model can not work well in this case. But, if the value of loop counter N is bigger enough, the "Unideal and practical" line will change in the same concave shape or close up to " Ideal and logical" line statistically. The adaptive model is expected to work well if the loop number is big enough. It's managable for a long time running pub/sub system.

5. Results of Evaluation

5.1 Environment

The algorithm is designed for main memory structure and evaluated in a 12D space. Both subscriptions and events are created randomly as unsigned short. The algorithm is implemented on R*-tree[3] with index node capacity 10 and leaf node capacity 20 which are default values. The hardware platform is Sun Fire 4800 with 4900MHz CPUs and 16G memory. The OS is Solaris 8.

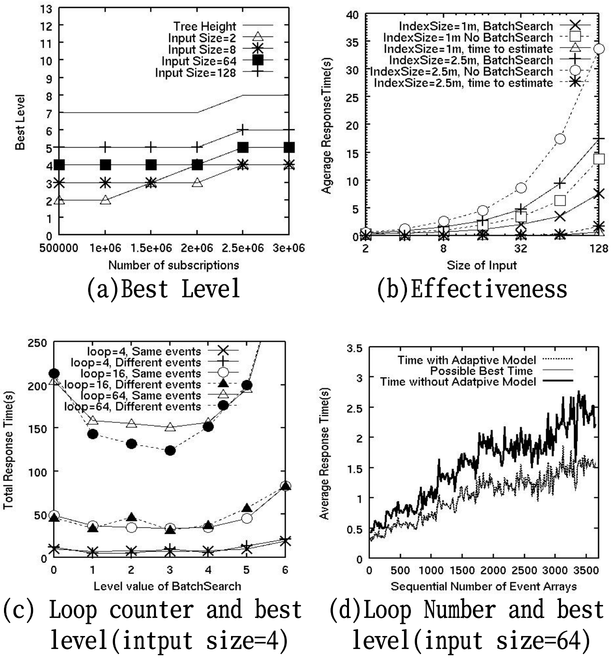
5.2 Evaluation of BatchSearch

The evaluation results are shown in Fig.4. Fig.4-a shows that the best level changes slowly with increment of index size. Fig.4-b compares the average response time of BatchSearch algorithm to that which just inputs events to original R*-tree in a random sequence. Further, it shows that the cost to estimate workload (algorithms shown in Fig.1 can be neglected compared to average response time. It also shows that the larger the size of input is, the more the average response time can be improved. The maximum is nearly up to 50% in our evaluation.

Fig.4-c and Fig.4-d show the changing of response time which are calculated with same and different events gotten at different levels. There the index size is 1.5 million and the height of tree is 7。It shows that with increment of loop counter, the trembling of response time marked "Unideal and practical" in Fig.3, iscaptured

[3] Version 0.62b. http://www.cs.ucr.edu/~marioh/spatialindex

clearly in Fig.4-c (loop=4 or loop=16, Different events), becomes more and more stable with shape of ideal concave and merge into the line gotten with the same events at each level.



(a)Best Level　(b)Effectiveness

(c) Loop counter and best level(intput size=4)　(d)Loop Number and best level(input size=64)

Fig.4 Evaluation Results of the BatchSearch

The l Fig.4-d shows the performance of unstable status compared to the performance without using the adaptive model (same as the "No BatchSearch" performances shown in Fig.4-b) and the possible best performance. There, the size of index changes from 0.5 million to 2.6 millions, the Threshold is 300,000, and the loop counter is 64. When the system becomes stable, 300,000 objects (subscriptions) are inserted into the index.

So the left part of Fig.5 shows the performance of unstable status.

We can find that performance with the adaptive model is much better than the performance without the adaptive model, the performance differences are almost at same level as those shown in Fig.4-b which are gotten at stable status. The performance of the adaptive model even in unstable status is very close to the possible best performance as shown in Fig.5-d. We can say that with the adaptive model, the arrays of events can be filtered with response time close to the possible best time. The difference can be neglected compared to the performance without adaptive model.

6. Related Work

A lot of algorithms related to event filtering have been proposed. They are proposed for publish/subscribe systems [2], [4], [5], [6], for continuous queries [1] [8] [9] and for active database [3].

Predicate indexing techniques have been widely applied. There,a set of one-dimensional index structures are used to index the predicates in the subscriptions, the representive algorithm is called counting algorithm [4] and Hanson algorithm[4]. They differ from each other by whether or not all predicates in subscriptions are placed in the index structures.

In [6] and [7], multidimensional index based event filtering is proved to be feasible and efficient. It's the basis of this paper.

Event filtering is one critical step of continuous queries. In [1], predicate index is built based on Red-Black tree, there algorithm is similar to bruteforce that scans the Red-Black tree for event filtering each time. Count algorithm was used in [8], [9].

[9] implemented routing policies to let faster operators filter out some tuples before they reach the slower operators. In [10], queries are optimized based on rate of input.

The problem of multiple events arriving at same time with different workloads is not considered in above techniques.

7. Conclusion

In this paper, for pub/sub system, we first proposed an event filtering algorithm with multiple events as input based on R-tree. Further an adaptive model is designed to filter multiple events for different event numbers and changing index size. According to the evaluation results, the average response time can be improved maximally up to nearly 50% with our algorithm and the adaptive model can work well with average response time same as or close to the possible best time in both stable and unstable statuses.

[文献]

[1] S. Chandrasekaran and M. J. Franklin: “Streaming queries over streaming data”, VLDB, pp.203-214(2002)

[2] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha: “Filtering algorithms and implementation for very fast publish/subscribe systems”, SIGMOD, pp.115-126(2001)

[3] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. B. Park, and A. Vernon: “Scalable trigger processing”, ICDE, pp.266-275(1999)

[4] T.W.Yan and H.Garcia-Molina: “The shift information dissemination system”. ACM Trans. Database Syst., 24(4), pp.529-565(1999)

[5] B.Wang, W. Zhang, and M. Kitsuregawa: “Design of b+tree-based predicate index for efficient event matching”, APWEB, pp.537-547(2003)

[6] B.Wang, W. Zhang, and M. Kitsuregawa: ” UB-Tree based effcient predicate index with dimension transform for pub/sub system”, DASFAA, pp63-74.(2004)

[7]W.Zhang. Performance analysis of Ub-tree indexed publish/subscribe system. Master’s thesis, Department of Information and Communication Engineer, The University of Tokyo, 2004

[8] J. Chen, D. J. DeWitt, F. Tian, and Y.Wang: ” Niagaracq: a scalable continuous query system for internet databases”, SIGMOD, pp. 379-390(2000)

[9] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman: ”Continuously adaptive continuous queries over streams”, SIGMOD, pp.49-60 (2002)

[10] S. Viglas and J. F. Naughton: “ Rate-based query optimization for streaming information sources”, SIGMOD, pp.37-48(2002)

王　波涛　Botao WANG

Research Fellow at Institute of Industrial Science, the University of Tokyo. He received the Ph.D degree in computer science in 2000 from Kyushu University. His research interests include spatial-temporal database, parallel processing and data stream. He is a member of DBSJ.

張　　旺　Wang ZHANG

Ph.D student of Graduate School of Information Science and Technology, the University of Tokyo. He received the Master degree in information engineering in the above gradate school. His research interests include data clustering and data stream. He is a student member of DBSJ.

喜連川 優　Masaru KITSUREGAWA

Professor and the director of center for information at Institute of Industrial Science, the University of Tokyo. He received the Ph.D degree in information engineering in 1983 from the University of Tokyo. His research interests include parallel processing and database engineering. He is a member of steering committee of IEEE ICDE, PAKDD and WAIM, and has been a trustee of the VLDB Endowment. Now he servers as general chair of ICDE2005. He was the chair of data engineering special interest group of Institute of Electronic, Information, Communication, Engineering, Japan, the chair of ACM SIGMOD Japan, Chapter. He is currently a trustee of DBSJ.