데이터 스트림에서 그래프 기반 기법을 이용한 슬라이딩 윈도우 다중 조인 처리

Processing Sliding Window Multi-Joins using a Graph-Based Method over Data Streams

장 량* / Liang Zhang, 유병섭** / Byeong-Seob You, 거준위*** / Jun-Wei Ge,
김경배**** / Gyoung-Bae Kim, 이순조***** / Soon-Jo Lee, 배해영****** / Hae-Young Bae

요 약

데이터 스트림 환경에서 셋 이상의 스트림들에 대한 조인연산을 위해 순서를 선택하는 기존 기법들은 항상 간단한 휴리스틱 방법을 이용하였다. 그러나 기존 기법들은 조인 선택도나 데이터 수신 비율과 같은 것만 고려하여 일반적인 응용에서 비효율적이며 낮은 성능을 갖는다. 본 논문에서는 최적의 조인 순서로 그래프 기반의 슬라이딩 윈도우 다중 조인 알고리즘을 제안한다. 이 기법에서 슬라이딩 윈도우 조인 그래프를 먼저 생성하는데, 정점(vertex)은 조인 연산으로 표현되고 엣지(edge)는 슬라이딩 윈도우들 사이의 조인관계를 나타낸다. 그리고 정점 가중치(vertex weight)와 엣지 가중치(edge weight)는 각각의 조인의 비용과 조인 연산들의 상호관계를 표현한다. 이때 데이터 스트림은 빠른 처리를 해야 하므로 메모리 기반의 그래프 기법을 사용한다. 이를 이용하여 최대값만을 이용하여 조인 연산을 수행하는 MVP 알고리즘을 개선하고 이의 그래프에서 최적의 조인 순서를 찾는다. 이를 통한 최종 결과는 중첩-루프(nested loop) 조인 계획을 수행하여 얻어진다. 성능비교를 통하여 제안기법이 기존 기법들보다 우수함을 증명한다.

Abstract

Existing approaches that select an order for the join of three or more data streams have always used the simple heuristics. For their disadvantage – only one factor is considered and that is join selectivity or arrival rate, these methods lead to poor performance and inefficiency in some applications. The graph-based sliding window multi-join algorithm with optimal join sequence is proposed in this paper. In this method, sliding window join graph is set up primarily, in which a vertex represents a join operator and an edge indicates the join relationship among sliding windows, also the vertex weight and the edge weight represent

■ 논문접수 : 2007.4.18 ■ 심사완료 : 2007.8.20

* 중경우전대 컴퓨터학과 석사과정(liuxuqingquan@gmail.com)
** 교신저자 인하대 컴퓨터정보공학과 박사과정(bsyou@dblab.inha.ac.kr)
*** 중경우전대 컴퓨터학과 교수(jwge@cqupt.ac.cn)
**** 서원대 컴퓨터교육과 조교수(gbkim@seowon.ac.kr)
***** 서원대학교 컴퓨터정보통신공학부 부교수(sjlee@seowon.ac.kr)
****** 인하대 컴퓨터공학부 교수(hybae@inha.ac.kr)

the cost of join and the reciprocity of join operators respectively. Then the optimal join order can be found in the graph by using improved MVP algorithm. The final result can be produced by executing the join plan with the nested loop join procedure. The advantages of our algorithm are proved by the performance comparison with existing join algorithms.

주요어 : 데이터 스트림, 그래프 이론, 윈도우 조인, 쿼리 최적화
Keyword : Data Stream, Graph Theory, Window Join, Query Optimization

1. 서론

Data stream [1] appears in many new applications recent few years. Data stream is fast, endless, continuous and real-time, which is different from traditional static relations stored in disk. Some representative applications include processing telephone call records [2], monitoring Internet traffic [3] and sensor data [4]. The system that can process data stream is called Data Stream Management System (DSMS). Each element in a data stream can be seemed as $\langle s, t \rangle$, where s is the data element and t indicates the corresponding timestamp.

It is impossible to store entire data for processing, as data stream is infinite and physical memory is limited. Therefore, sliding window [1] may be used for preserving the new arrival data that are more significant. The basic problem about sliding window is inserting and expiring tuples. So inserting and expiring tuples leads to two different re-execution strategies. The eager re-evaluation strategy generates new results after each new tuple arrives. The lazy one re-executes the query periodically, which is a more practical solution. The query in DSMS is being executed continuously that is called continuous query. Generally speaking, each sliding window maps to one data stream. A sliding window can be shared by many continuous queries, and also one continuous query could inquiry many sliding windows. The research in this paper is about the join sequence of many sliding windows in one query with lazy re-execution strategy.

A simple heuristic based on one parameter is used for appointing the join sequence of data stream in the literature, such as evaluating the most selective predicate first [5]. However, a graph-based model for relation join is proposed in [6]. The model is the weighted direction join graph that includes many significant factors and can indicate all possible query plans (various execution orders). Then using the heuristic MVP algorithm an effective spanning tree is found in the graph model. Each spanning tree is corresponding to a query plan.

As sliding window is a stream-to-relation operator [1], a correlative graph model can also be defined to representing sliding window join. In this graph a node represents a join operation and a directed edge indicates the existence of common sliding window between two nodes. Weight values with optimization required parameters including arrival rate, window size and join selectivity factor are imposed to each node and edge. The model can represent different kinds of sliding window join queries. Then we improve

the MVP algorithm according to the attribute of data stream to acquire a spanning tree with low cost. As the computing of execution plan is not exploited in [6], the nested loop join algorithm is used to execute the query plan. Finally, the whole procedure to process sliding window join is completed.

Several contributions have been made in this paper. First, we define a novel sliding window join graph and identify its convenience of analyzing join query. Then based on the sliding window join graph we develop a new integrated join algorithm including the step that arrange siding windows in an optimal join order. Finally we describe our implementation of the proposed algorithm and present results from a detailed performance study of the implementations.

Table 1 lists the symbols used in this paper and their meanings. The rest of the paper is organized as follows: Section 2 describes related work and section 3 shows how to use graph-based approach to process the window join. Section 4 presents experimental study of the algorithm. Finally, Section 5 gives the conclusion.

<Table 1> Definition of terms

Symbol	Meaning
S_j	Stream j
W_j	Sliding window related to stream j
λ_j	Stream j arrival rate
T_j	Stream j time window size
N_j	Number of tuples in W_j
B_j	Number of hash bucket in W_j
C_n	Cost of accessing one tuple in NLJ
C_h	Cost of accessing one tuple in HJ
M_i	Constant
$\|X\|$	The size of relation X
$\|\|X\|\|$	The tuple size of relation X
θ	Join Operator
τ	Re-execution interval
JSF	Join selectivity factor
JCF	Join concatenation factor
NOW	The time that execute the query plan

2. Related works

The related works are classified into two parts: stream join; the traditional graph model and MVP algorithm.

2.1 Stream join

Most recent works on joining processing over data streams are based on Symmetric Hash Join (SHJ) [7] which adopts the pipelining algorithm [8]. SHJ is extended to XJoin [9] by processing spill overflowing inputs to disk effectively when memory fills up. Data stream multi-joins operation also extends SHJ to two classifications that are those processing a series of pipelining binary joins [9] [10], and those defining a single, symmetric, multiway join operator, such as Mjoin [5], SteMs [11]. The proposed graph-based algorithm is related to appoint the sequence of many binary join operators.

[10] not only proposes incremental multiway nested loop join and hash join algorithms, also presents a sensible join ordering heuristic that is sorting the join first

whose join selectivity is small and assembling fast streams at or near the top of the query plan. [5] also researches the probing sequence of data streams in experiment. But all above papers do not discuss the join order of sliding window. Also no existing join algorithm includes the step of selecting join order. These problems are researched in this paper.

2.2 Graph model and MVP algorithm

[2] defines theweighted directed join graph (WDJG). In this graph, each node represents a join operation. If a common relation exists between two nodes, the edge is a physical edge (PE); if not, the edge is a virtual edge (VE). The direction of edge implies the join order. Edge weights indicate the impact of a join to the cost of the next join. E.g. if there are two join operators $\theta_j = R\theta S$ and $\theta_i = S\theta T$, the edge weight of $\overrightarrow{v_j v_i}$ is shown as:

$$\overrightarrow{w_{ji}} = \langle w^1_{ji}, w^2_{ji} \rangle = \begin{cases} \left\langle \frac{|R|\times|S|JSF_j}{|S|}, \frac{(\|R\|+\|S\|)\times JCF_j}{\|S\|} \right\rangle & \overrightarrow{v_j v_i} \text{ is PE} \\ \langle 1,1 \rangle & \overrightarrow{v_j v_i} \text{ is VE} \end{cases} \quad (1)$$

Vertex weights represent the accumulated impact of a join sequence to a next join operator. The weight of θ_i is shown as (θ_i is the next join of θ_i and $\overrightarrow{v_j v_i}$ is PE):

$$\overrightarrow{W_i} = \langle W_i^1, W_i^2 \rangle = \left\langle \prod_{\theta_j} W_j^1 \times w^1_{ji}, \sum_{\theta_j} ((W_j^2 - \frac{1}{w^2_{ji}}) \times JCF_i \right\rangle \quad (2)$$

A spanning tree becomes an execution plan in WDJG. For finding query plans faster, effective spanning tree (EST) is defined. An effective query plan can be found in shorter processing time by deleting ineffective spanning tree and reducing giant search space.

The Maximum Value Precedence (MVP) algorithm is a relatively low complexity and yet high effiency algorithm for optimizing in WDJG. This algorithm finds a near optimal solution using only $O(n^2)$ time, lower than existing algorithm. Its idea is to reduce the cost of expensive join operations as early as possible [2]. Thus two steps are included in the algorithm. The first step is to choose an edge to reduce the costly vertex weight. The second step is to select an edge that causes the edges minimum increase to the result of joins.

3. Design of graph-based sliding window multi-join

In this chapter, we first introduce the sliding window join graph and also analyze the detailed computing of the weight in this model. We then propose the sliding window join algorithm based on this model.

3.1 Sliding window join graph

A Sliding Window Weighted Directed Join Graph (SWWDJG) is a weighted complete graph. Each vertex represents a join operation of two sliding windows. Each vertex is connected to another vertex via two edges in opposite directions. Physical edges (PE) and virtual edges (VE) are two kinds of edges in

each SWWDJG. Physical edges $\overrightarrow{v_j v_i}$ and $\overrightarrow{v_i v_j}$ exist between vertexes v_i and v_j only if there is a common sliding window between them, otherwise virtual edges $\overrightarrow{v_j v_i}$ and $\overrightarrow{v_i v_j}$ exists. The direction of an edge indicates an execution order of θ_i after θ_j. Each vertex v_j and edge $\overrightarrow{v_j v_i}$ is associated with a weight, $\overrightarrow{w_j}$ and $\overrightarrow{w_{ji}}$ respectively. They are defined as (3) (4).

$$\overrightarrow{w_{ji}} = \left\langle w_{ji}^1, w_{ji}^2 \right\rangle = \begin{cases} \left\langle |W_j| \times JSF_j, \dfrac{(\|W_j\|+\|W_i\|)\times JCF_j}{\|W_i\|} \right\rangle & \overrightarrow{v_j v_i} \text{ is a PE} \\ \langle 1,1 \rangle & \overrightarrow{v_j v_i} \text{ is a VE} \end{cases} \quad (3)$$

$$\overrightarrow{w_j} = \text{cost}(\theta_j) \quad (4)$$

SWWDJG is rather different from WDJG. Firstly, the join operation represented by node is processed between sliding windows instead of static relations. Secondly, when calculating the edge weight– the cardinality and the tuple size of relation are replaced by the window size and the element size in data stream. And lastly, vertex weight in WDJG is dependent on its preceding vertex weights, which makes it difficult to specify. Thus the vertex weight in SWWDJG is set to the cost of each join operator, which makes us compute the join cost easier.

3.2 Computing edge weight and vertex weight

As the difference of two kinds of sliding windows [1]: time-based sliding window and count-based sliding window is the window size, we only researched the detailed computation of edge weight and vertex weight about time-based sliding window. The edge weight and vertex weight of count-based sliding window is similar to its. For each kind of weight, a computation formula is proposed.

3.2.1 Edge weight

For a virtual edge, the join operations of VE will not influence each other on the join cost. Hence, the weight of a VE is defined as <1,1> to show that the joins are independent.

Now consider the PE. The tuple size of a sliding window is constant written as $\|W_j\| = M_j$, $\|W_i\| = M_i$. The window size is represented as $|W_j| = \lambda_j T_j$, $|W_i| = \lambda_i T_i$. As the join selectivity factor and the join concatenation factor can be normally collected from statistical information, the edge weight can be represented by above factors. So (3) can be reshown as:

$$\overrightarrow{w_{ji}} = \left\langle w_{ji}^1, w_{ji}^2 \right\rangle = \begin{cases} \left\langle \lambda_j T_j JSF_j, \dfrac{(M_j+M_i)\times JCF_j}{M_i} \right\rangle & \overrightarrow{v_j v_i} \text{ is a PE} \\ \langle 1,1 \rangle & \overrightarrow{v_j v_i} \text{ is a VE} \end{cases} \quad (5)$$

3.2.1 Vertex weight

Vertex weight only implies the cost of the join operation itself. The cost of inserting and expiring tuples is ignored because it is not influenced by join ordering. The cost of

join operation over data stream is rather different from traditional database. For relational data join, only the I/O cost is contained (CPU cost is much less than I/O cost). Instead of storing in the disk, the data streams directly enter the input buffer then are insert to the sliding window for processing. The whole join procedure is performed in the memory. Thus only the CPU cost is calculated over data stream.

The data structure of sliding window can be hash table or queue, which is mapped to hash join or nested loop join respectively. Hence based on the unit-time cost model [9], the cost formulas of nested loop join and hash join about sliding window are shown as:

$$\begin{aligned} \overline{\overline{W}}_j &= Cost(\theta_j) \\ &= |W_j| \times \|W_i\| \times |W_j| \times \|W_i\| \times C_n \\ &= \lambda_j T_j \lambda_i T_i M_j M_i C_n \end{aligned} \quad (6)$$

$$\begin{aligned} \overline{\overline{W}}_j &= \mathrm{Cost}(\theta_j) \\ &= |W_j| \times \|W_j\| \times |W_i| \times \|W_i\| \times C_h \\ &= \frac{\lambda_j T_j \lambda_i T_i M_j M_i}{B_j B_i} C_h \end{aligned} \quad (7)$$

3.3 Graph-based join algorithm

Using the graph join model, we can deal with sliding window multi-joins. Figure 1 shows the graph-based sliding window join algorithm with lazy re-evaluation. As pointed out earlier, there are three phases in our algorithm: founding join model, selecting join query plan and executing query plan.

In the first phase, when setting up the SWWDJG before probing in step 03, the new arrival tuples in stream i are seemed as a new W_i to replace the original one, so that the output result can be updated continuously with optimal join order. This is similar to the pipelining algorithm [4] that the new arrival

```
Input: A continuous sliding window multi-join query
Output: The result tuples
Begin: Every time the query is re-executed
01: Insert new tuples into sliding windows;
02: For i=1…n;
03: Set up the SWWDJG_i
04: JoinOrder (SWWDJG_i);   Return (W_1 θ_1 W_2) θ_2 … θ_{n-1} W_n)
05: ∀t_1∈W_1 and ∀t_2∈W_2
        If t_1.attr θ t_2.attr   ……//loop from W_n to W_k
06:     ∀t_k∈W_k and NOW-τ ≤ t_k.ts ≤ NOW and t_k.ts-T_1 ≤ t_1.ts ≤ t_k.ts
          and t_k.ts-T_2 ≤ t_2.ts ≤ t_k.ts and ……
            If t_1.attr θ t_k.attr ……//loop from W_{k+1} to W_n
              ∀t_n∈W_n and t_k.ts-T_n ≤ t_n.ts ≤ t_k.ts
               If t_1.attr θ t_n.attr
07:            Return t_1 θ t_2 θ t_3 … θ t_n
08:          Endif……
09:     Endif……
10: Endif
End
```

<Figure 1> The Graph-based Join Algorithm

tuples are inserted to its corresponding sliding window, and then probe other windows.

In the second phase, we use the improved MVP algorithm [6] to find an optimal join query plan in step 04, as the cost of nested loop join about sliding window is different. In original MVP algorithm an inflowing edge $\overrightarrow{v_j v_i}$ whose weight $w^1_{v_j v_i}$ is less than 1 can reduce the cost of vertex v_j. That means if the intermediate result replaces the original relation the tuple number of join result will be reduced. But when processing sliding window, from (6) and (7) we can see the two weight vectors influence same to join cost. Thus the inflowing edge whose weight $w^1_{v_j v_i} w^2_{v_j v_i}$ is less than 1 can reduce the cost of sliding window join.

In the last phase, the query plan is executed through a nested loop join procedure where the timestamp of tuples have to satisfy certain condition to make sure the result is valid.

3.4 Example

In a four time-based sliding windows join query, θ_1 is on W_1 and W_2, θ_2 on W_2 and W_3, θ_3 on W_3 and W_4, θ_4 on W_4 and W_1. Assume that the tuple size of each sliding window is 100, and their parameters are shown in Table 2.

Using the graph-based algorithm we first set up the corresponding SWWDJG in Figure 2. Also the vertex weight (cost of each join operator) is calculated (here we assume $C_n = 1$).

<Table 2> Sliding Window Parameters

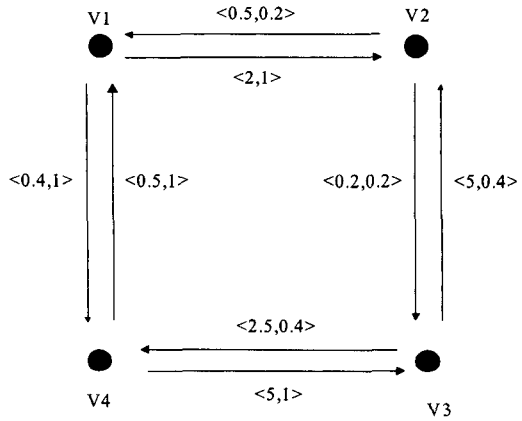
θ_i	JSF_i	JCF_i	λ_i	T_i	λ_{i+1}	T_{i+1}
θ_1	0.002	0.5	10	100	2	100
θ_2	0.001	0.1	2	100	5	100
θ_3	0.05	0.2	5	100	1	100
θ_4	0.005	0.5	1	100	10	100

$$\overrightarrow{W_1} = \text{cost}(\theta_1) = \lambda_1 T_1 \lambda_2 T_2 M_1 M_2 C_n = 2 \times 10^9$$

$$\overrightarrow{W_2} = \text{cost}(\theta_2) = \lambda_2 T_2 \lambda_3 T_3 M_2 M_3 C_n = 1 \times 10^9$$

$$\overrightarrow{W_3} = \text{cost}(\theta_3) = \lambda_3 T_3 \lambda_4 T_4 M_3 M_4 C_n = 0.5 \times 10^9$$

$$\overrightarrow{W_4} = \text{cost}(\theta_4) = \lambda_4 T_4 \lambda_1 T_1 M_4 M_1 C_n = 1 \times 10^9$$



<Figure 2> The SWWDJG of the Example Join

In this graph, a near optimal join sequence is selected by using the improved MVP algorithm which gives the result: $v_2 \rightarrow v_1 \rightarrow v_4 \rightarrow v_3$. The comparison among different join orders is shown below. Obviously the join order found by the MVP algorithm is optimal.

<Table 3>Possible Join Sequence and Their Cost

Effective Spanning Tree	Total execution cost
$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$	6.44×10^9
$v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_1$	5.32×10^9
$v_3 \rightarrow v_4 \rightarrow v_1 \rightarrow v_2$	30.125×10^9
$v_4 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$	3.2×10^9
$v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$	17×10^9
$v_1 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2$	7.4×10^9
$v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_4$	3.77×10^9
$*v_2 \rightarrow v_1 \rightarrow v_4 \rightarrow v_3$	1.72×10^9

4. Performance evaluation

The evaluation environment and the comparison among our join algorithm (GRAPH), Lazy Multi-Way NLJ (LAZY) [10] algorithm and General Lazy Multi-Way NLJ (GEN) [10] algorithm are presented in this chapter. In LAZY algorithm, the join sequence is ordering the new arrived tuples first. The join order in GEN is specified to select the joining with minimal *JSF* first.

4.1 Evaluation environment

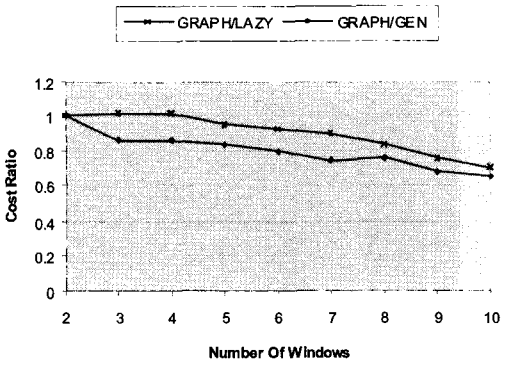
We implement our algorithm by using VC programming, which runs on Windows PC with a 1.8GHz Pentium(R) 4 processor and 1GB physical memory.

The data streams are produced by the following way. In each iteration of a loop stream i is selected with probability $\frac{\lambda_i}{\sum_{j=1}^{n}\lambda_j}$, while a tuple is generated for the selected stream. Each tuple includes three attributes: the system-assigned timestamp ts, the integer data d and the filled part. The integer data is selected form corresponding data set $\{1,\cdots,a_i\}$. The *JSF* of two windows is calculated as $\frac{1}{\max(a_i,a_j)}$. Towards *JCF*, the output tuple size is adjusted by increasing or reducing the filled part. For each case, certain tuples are generated and the processing time measured. Each case is run many times and the average result is reported.

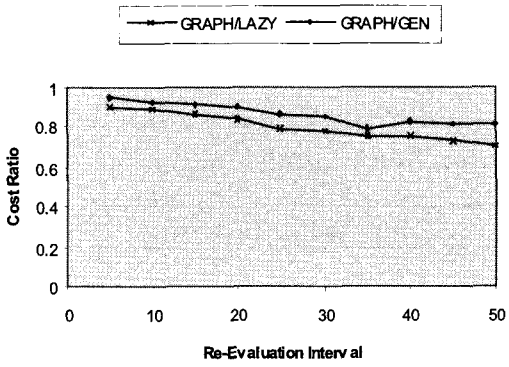
4.2 Evaluation results

In the first experiment, we specify the window size, value set and *JCF*. The relative rates of two streams are set certainly, while new streams are specified. Figure 3shows that LAZY is better than GRAPH at the beginning, but GRAPH outperforms as the number of window increases. The reason is- when the number of windows is low ordering newly arrived tuple first is always optimal so that GRAPH has to spend extra time finding query plan. As the number of windows increases the time saving from executing query plan is much more than the time to find the join order. GRAPH is much better than GEN. That is because *JSF* in this experiment is fixed so that GEN selects the join operator naively. Besides, GRAPH not only considers *JSF* but also *JCF*.

In the second case,the four sliding window has been joined with parameters in table 2 which is evaluated when varying re-evaluation intervals. From Figure 4, it can be observed that the LAZY and GENERAL algorithms are more expensive than GRAPH with optimal query plan.



<Figure 3> Varying Number of Streams



<Figure 4> Varying Re-execution Interval

5. Conclusion

In this paper, we proposed a new graph model, named SWWDJG, which can properly represent all the execution orders about sliding window multi-join in a standard way. Many significant parameters have been used in the graph to calculate the join cost and to gain optimal query plan. Then we present a new sliding window join algorithm by using the model to specify the sliding window join order. As the query plan selects the optimal join sequence, the performance can be improved in a great manner.

In future, we can find more efficient algorithms to provide query plan based on the graph model. As graph-based join algorithm assumes memory that is big enough to store all data in sliding window, we can also research load shedding within limited memory.

6. References

1. B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, "Models and Issues in Data Streams", In Proc. ACM Symp. on Principles of Database Systems, 2002, pp 1~16.
2. C. Cortes, K. Fisher, D. Pregibon, A. Rogers, F. Smith, "Hancock: A Language for Extracting Signatures from Data Streams", In Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp 9~17.
3. M. Sullivan, A. Heybey, "Tribeca: A System for Managing Large Databases of Network Traffic", In Proc USENIX Annual Technical Conf, 1998.
4. P. Bonnet, J. Gehrke, P. Seshadri, "Towards Sensor Databases Systems", In Proc. Int. Conf. on Mobile Data Management, 2001, pp 3~14.
5. S. Vigalas, J. Naughton, and J. Burger, "Maximizing the output rate of multi-way join queries over streaming information sources", Proc.29th Intl. Conf. on Very Large Databases, 2003.
6. Chiang Lee, Chi-Sheng Shih, and Yaw-Huei Chen, "Optimizing large join queries using a graph-based approach", IEEE Transactions on Knowledge and Data Engineering, 2001.13(2).
7. A. N. Wilschut, P. M. G Apers, "Dataflow

query execution in a parallel main-memory environment", Distributed and Parallel Databases, 1993.

8. A. N. Wilschut and P. M. G Apers, "Pipelining in query execution", Conference on Database, Parallel Architectures and their Applications, 1991.
9. T. Urhan, M. J. Franklin, and L.Amsaleg, "XJoin: A reactively-scheduled pipelined join operator", IEEE Data Engineering Bulletin, volume 23, 2000.
10. L.Golab and T.Ozsu, "Processing sliding window multi-joins in continuous queries over data streams", Proc. of the 2004 Intl. Conf. on Very Large Database, 2004.
11. V. Raman, A. Deshpande, J. M Hellerstein, "Using state modules for adaptive query processing", Proc. 19th Intl Conf. Data Engineering (ICDE), IEEE Computer Society Press, 2003, pp 353~364.

장 량 (Zhang Liang)

2004년 Department of Electronic Engineering, Nanjing University of Posts and Telecommunications, China (공학사)

2005~현재: Department of Computer Science, Chongqing University of Posts and Telecommunications, China (석사과정)

관심분야: stream data query processing, Spatial Database, Spatial Information & Data Management

유병섭 (ByeongSeob You)

2002년 인하대학교 전자 · 전기 · 컴퓨터공학부-컴퓨터공학(공학사)

2004년 인하대학교 대학원 전자계산공학과(공학석사)

2008년 인하대학교 대학원 정보공학과(공학박사 예정)

관심분야 : 공간 데이터베이스, 공간 데이터 웨어하우징, 센서 네트워크

거준위 (Junwei Ge)

1991년 geophysical technology, Chengdu University of Technology, China (이학박사)

1991년~현재 a professor in the Faculty of Software at Chongqing University of Posts and Telecommunications, China

관심분야 : software requirements engineering, software architectures, GIS

김경배 (GyoungBae Kim)

1992년 인하대학교 전자계산공학과(공학사)

1994년 인하대학교 대학원 전자계산공학과(공학석사)

2000년 인하대학교 대학원 전자계산공학과(공학박사)

2000년~2004년 한국전자통신연구원 선임연구원

2004년~현재 서원대학교 컴퓨터교육과 조교수

관심분야 : 이동실시간데이터베이스, 스토리지시스템, GIS, VOD

이순조 (SoonJo Lee)

1985년 인하대학교 전자계산학과(이학사)

1987년 인하대학교 대학원 전자계산학과(이학석사)

1995년 인하대학교 대학원 전자계산공학과(공학박사)

1997년~현재 서원대학교 컴퓨터정보통신공학부 부교수

관심분야 : 데이터베이스 시스템, 정보보안, GIS

배해영 (HaeYoung Bae)

1974년 인하대학교 응용물리학과(공학사)

1978년 연세대학교 전자계산공학과(공학석사)

1990년 숭실대학교 전자계산공학과(공학박사)

1992~1994년 인하대학교 전자계산소 소장.

1982년~현재 인하대학교 컴퓨터공학부 교수.

1999년~2007년 지능형GIS연구센터 센터장.

2000년~현재 중국 중경우전대학교 대학원 명예교수.

2004년~2006년 인하대학교 정보통신대학원 원장.

2006년~현재 인하대학교 대학원 원장.

관심분야 : 분산 데이터베이스, 공간 데이타베이스, 지리정보 시스템, 멀티미디어 데이타베이스