An Algorithm for Distributed Aggregation-join Query Processing in Data Grids

Hua Feng1[†] and Zhenhuan Zhang2[††],

[†]Daqing Oilfield Limited Company, China
[††]Daqing Oilfield Production Technology Institute China

Summary
Aggregation-join query plays an important role in query processing in data girds and has been applied to many fields, such as global climate simulation, high energy physics and molecular biology. Applying aggregation and join operations on remote relations in data grids is a unique and difficult issue due to the heterogeneous, unpredictable and volatile behaviors of the grid resources. To the best of our knowledge, little is done to date on aggregation-join query processing in data grids. An approach for efficiently processing aggregation-join query is proposed in this paper. And analytical and experimental results show the effectiveness and efficiency of the proposed approach.

Key words:
Data Grids, Aggregation-join Query, Partial Replica, Replica Maximum Cover, Minimum Maximum Edge Matching

1. Introduction

Data grids [1,2] are distributed data management architecture used as a coordinated and collaborative means for integrating data across high speed networks, and thus form a single virtual environment for data access and management [3,4,5] The employment of data grids has provided the scientific community with fast, reliable and transparent access to geographically distributed data resources.

Data grids utilize traditional replica technique to replicate one or several copies of each dataset and distribute them at different grid nodes. And thus the existence of replicas efficiently reduces data access cost and network transmission cost, as well as increases grid system fault tolerance, achieves load balancing and improves the security of data sets.

In such distributed grid environments, Data Grid Management System (DGMS) needs to continue monitoring grid global performance for capacity planning and system diagnosis, such as network bandwidth, available storage space, CPU power usage. Grid users need to monitor dynamic information about grid resource, thus discovery interested and appropriate resource. So Resource Monitoring and Discovery Mechanism play an important role in data grids.

Queries involving aggregates are very common used by Resource Monitoring and Discovery Mechanism for gird resource management, discovery and publication in data grids, which summarize a large set of records based on the designated grouping. The input set of records may be derived from multiple tables using a join operation. These queries are often used as a tool for strategic decision making and commonly used in a variety of applications including data integration services, decision support systems and scientific data analysis.

Although data grids offer a great deal of facilities for wide-area query processing, query processing in data grids is challenging due to the heterogeneous, unpredictable and volatile behavior of grid resources. As far as we know, there is little to date in the literature on distributed aggregation-join query in data grids exploring relation partial replicas and load balancing. The contribution of this paper is to have proposed an adaptive aggregation-join query processing algorithm that makes use of relation partial replicas and achieves load balancing.

The rest of the paper is organized as follows. In Section 2, the problem of grid aggregation-join query and the procedure for processing this query are presented. An algorithm is proposed to get efficient tuples by reducing the sizes of partial replicas. And, the concept of *Replica Maximum Cover* and its related algorithm are introduced in Section 3. Section 4 defines the concept of *Minimum Maximum Edge Matching* and develops its related algorithm for adaptively selecting computational nodes. Section 5 proposes the methods for parallel executing join and aggregation operations at selected nodes. Adaptive adjustment of query processing is given in Section 6. The experimental results are provided in Section 7, and the conclusion and future work are discussed in Section 8.

2. Problem Statement

Assume a user at any grid node issues a query to DGMS and the query is required to get aggregation-join results of relation *R* and *S* according to join attributed *T*, group-by attribute *R.GB* and aggregation attribute *S.A*. *R* and *S* have been split into numerous partial replicas and these replicas are present at different grid nodes, the partial replicas of

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

relation R, R_1, R_2,...,R_{m_1} locate at m_1 different grid nodes NR_1,NR_2,..., NR_{m_1} and the partial replicas of relation S, S_1,S_2,...,S_{m_2} locate at m_2 different grid nodes NS_1,NS_2,...,NS_{m_2}. The problem of aggregation-join query processing is formulated as follows.

INPUT: (1) an aggregation-join query Q
(2) partial replicas of relations R and S, R_1,R_2,..., R_{m1} and S_1,S_2,...,S_{m2} which locate different grid nodes.
(3) Join Attribute T, group-by attributed $R.GB$ and aggregation attribute $S.A$

OUTPUT: Aggregation-join results of $R \triangleright\triangleleft S$ according to R.GB and $S.A$

Considering the existence of multiple replicas of datasets and achieving system load balancing, the aggregation-join of R and S is computed in the following six steps in general.

Step 1. Reduce the sizes of partial replicas R_1,R_2,...,R_{m_1} and S_1,S_2,...,S_{m_2}, and get efficient tuple sets R'_1, R'_2,...,R'_{m_1} and S'_1,S'_2,...,S'_{m_2}.

Step 2. Respectively select n_1 ($n_1 \le m_1$) and n_2 ($n_2 \le m_2$) partial replicas from R'_1,R'_2,...,R'_{m_1} and S'_1,S'_2,..., S'_{m_2} as operand relations, satisfying $R=R'_1 \cup R'_2 \cup ... \cup R'_{n_1}$ and $S=S'_1 \cup S'_2 \cup ... \cup S'_{n_2}$.

Step 3. Remove duplication from operand relations R'_1,R'_2,...,R'_{n_1} and get R''_1,R''_2,...,R''_{n_1} satisfying $R''_i \cap R''_j = \varnothing$, ($i \ne j$,$1 \le i$, $j \le n_1$) and $R''_1 \cup R''_2 \cup ... \cup R''_{n_1} = R$. Similarly, remove duplication from operand relations S'_1,S'_2,...,S'_{n_2} and get S''_1,S''_2,..., S''_{n_2} satisfying $S''_i \cap S''_j = \varnothing$, ($i \ne j$,$1 \le i$, $j \le n_2$) and $S''_1 \cup S''_2 \cup ... \cup S''_{n_2} = S$.

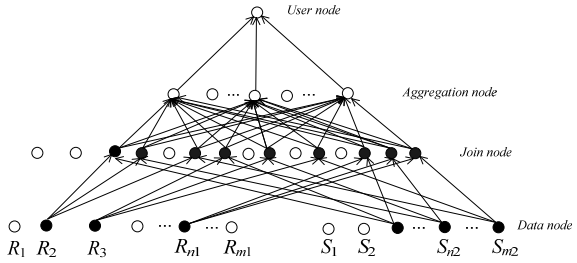


Fig.1 Illustration of processing an aggregation-join query

Step 4. Select at most $n_1 \times n_2$ grid nodes as join nodes (*JN*s) from m available ones, and transfer efficient tuple sets of R''_1,R''_2,...,R''_{n_1} and S''_1,S''_2,...,S''_{n_2} to join nodes for parallel performing join operations.

Step 5. According to the group-by attribute $R.GB$, select aggregation nodes (*AN*s) from m available grid nodes and transfer join results to *AN*s for completing aggregation operations.

Step 6. Parallel transfer aggregation-join results from *AN*s to user node in pipeline.

3. Pre-Processing of Relation Partial Replicas

This section first discusses how to get efficient tuple sets R'_1,R'_2,...,R'_{m_1} and S'_1,S'_2,...,S'_{m_2} from partial replicas R_1,R_2,...,R_{m_1} and S_1,S_2,...,S_{m_2}. Then the concept of *Replica Maximum Cover* is proposed and an algorithm for seeking optimal partial replicas as operand relations is present. Finally, a method for removing duplication from multiple selected partial replicas for each relation is described.

3.1 Getting Efficient Tuple Sets

In grid environments, grid services like Grid Information Service and Network Weather Service have been developed and used to dynamic resource discovery and network monitoring respectively. Our approach utilizes these services to locate partial replicas of the given relations R and S, i.e. R_1, R_2,...,R_{m_1} and S_1,S_2,...,S_{m_2}.

When processing aggregation-join queries in data grids, it is not necessary to transfer all tuples in each partial replica to computational nodes for performing join and aggregation operations. We only need to transfer the tuples that satisfy the join conditions, and thus the network transmission cost is reduced.

An algorithm for reducing the sizes of each partial replica, *Obtain-Effient-Tuples* (*OTE*), is proposed in this section. And the algorithm is processed in the following three steps.

Step 1. Respectively get the projection of each partial replica on the join attribute T, $R_1[T]$, $R_2[T]$, ..., $R_{m_1}[T]$ and $S_1[T]$, $S_2[T]$, ..., $S_{m_2}[T]$, at grid nodes NR_1, NR_2, ..., NR_{m_1} and NS_1, NS_2, ..., NS_{m_2}.

Step 2. Parallel execute (2.1) and (2.2)
(2.1) Parallel execute (2.1.1)-(2.1.m_2)
(2.1.1) Parallel transfer $R_1[T]$, $R_2[T]$, ..., $R_{m_1}[T]$ from nodes NR_1, NR_2,..., NR_{m_1} to node NS_1
(2.1.2) Parallel transfer $R_1[T]$, $R_2[T]$, ..., $R_{m_1}[T]$ from nodes NR_1, NR_2, ..., NR_{m_1} to node NS_2
...
(2.1.m_2) Parallel transfer $R_1[T]$,$R_2[T]$,...,$R_{m_1}[T]$ from nodes NR_1,NR_2,...,NR_{m_1} to node NS_{m_2}
(2.2) Paralle execute (2.2.1)-(2.2.m_1)
(2.2.1) Parallel transfer $S_1[T]$,$S_2[T]$,...,$S_{m_2}[T]$ from nodes NS_1,NS_2,...,NS_{m_2} to node NR_1
(2.2.2) Parallel transfer $S_1[T]$,$S_2[T]$,...,$S_{m_2}[T]$ from nodes NS_1,NS_2,...,NS_{m_2} to node NR_2
...

(2.2.m_2) Parallel transfer $S_1[T],S_2[T],...,S_{m_2}[T]$ from nodes $NS_1,NS_2,...,NS_{m_2}$ to node NR_{m_1}

Step 3 Parallel execute (3.1) and (3.2)

(3.1) Parallel execute (3.1.1)-(3.1.m_2)

(3.1.1) At node NS_1, compute

$S'_1(R_1)=R_1[T] \rhd\lhd S_1$,
$S'_1(R_2)=R_2[T] \rhd\lhd S_1$,
...
$S'_1(R_{m_1})=R_{m_1}[T] \rhd\lhd S_1$

(3.1.2) At nodeNS_2, compute

$S'_2(R_1)=R_1[T] \rhd\lhd S_2$,
$S'_2(R_2)=R_2[T] \rhd\lhd S_2$,
...
$S'_2(R_{m_1})=R_{m_1}[T] \rhd\lhd S_2$

...

(3.1.m_2) At node NS_{m_2}, compute

$S'_{m_2}(R_1)=R_1[T] \rhd\lhd S_{m_2}$,
$S'_{m_2}(R_2)=R_2[T] \rhd\lhd S_{m_2}$,
...
$S'_{m_2}(R_{m_1})=R_{m_1}[T] \rhd\lhd S_{m_2}$

(3.2) Parallel execute (3.2.1)-(3.2.m_1)

(3.2.1) At node NR_1, compute

$R'_1(S_1)=S_1[T] \rhd\lhd R_1$,
$R'_1(S_2)=S_2[T] \rhd\lhd R_1$,
...
$R'_1(S_{m_2})=S_{m_2}[T] \rhd\lhd R_1$

(3.2.2) At node NS_2, compute

$R'_2(S_1)=S_1[T] \rhd\lhd R_2$,
$R'_2(S_2)=S_2[T] \rhd\lhd R_2$,
...
$R'_2(S_{m_2})=S_{m_2}[T] \rhd\lhd R_2$

...

(3.2.m_1) At node NS_{m_2}, compute

$R'_{m_2}(S_1)=S_1[T] \rhd\lhd R_{m_1}$,
$R'_{m_2}(S_2)=S_2[T] \rhd\lhd R_{m_1}$,
...
$R'_{m_2}(S_{m_2})=S_{m_2}[T] \rhd\lhd R_{m_1}$

Thus at nodes NR_i ($1\le i\le m_1$), efficient tuple sets $R'_i(S_1)$, $R'_i(S_2),...,R'_i(S_{m_2})$, where $R'_i(S_1) \cup R'_i(S_2)\cup...\cup R'_i(S_{m_2})=R'_i$, are obtained. Similarly, at nodes $NS_j(1\le j\le m_2)$, efficient tuple sets $S'_j(R_1),S'_j(R_2),...,S'_j(R_{m_1})$, where $S'_j(R_1)\cup S'_j(R_2) \cup...\cup S'_j(R_{m_1})=S'_j$, are got

3.2 Selecting Optimal Partial Replicas

Definition 1 (Replica Cover): Given a relation R, two sets $\Phi=\{R_1,R_2,...,R_m\}$ and $F=\{R^1,R^2,...,R^n\}$, $R_i\subset R$, $R^j\subset R$, $1\le i\le m$, $1\le j\le n$, $n\le m$, $F\subseteq\Phi$. F covers relation R if the following conditions are satisfied:

(1) $\forall\ t\in R$, $\exists\ R^i\in F$ satisfying $t\in R^i$
(2) $R= R^1\cup R^2\cup...\cup R^n$
(3) $\forall R^i$ and R^j ($R^i\in F,R^j\in F,i\ne j$), satisfy $R^i\not\subset R^j$ and $R^j\not\subset R^i$

The number of partial replicas in F is the size of cover F, denoted by $|F|$.

Definition 2 (Replica Maximum Cover) Given a relation R, a set $\Phi=\{R_1,R_2,...,R_m\}$ where $R_i\subset R$, and a set $C_R=\{F_1, F_2,...,F_n\}$ where F_i is a replica cover of R, F_i is named *Replica Maximum Cover* of relation R if $|F_i|= Max\{|F_1|,|F_2|,...,|F_m|\}$.

Assume a relation R={(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)}, Φ={R_1, R_2, R_3, R_4, R_5},R_1, R_2, R_3, R_4 and R_5 are partial replicas of R, R_1={(0, 1), (1, 2)}, R_2={(2, 3), (3, 4), (4, 5), (5, 6)}, R_3={(2, 3), (4, 5)}, R_4={(3, 4), (5, 6)}, R_5={(1, 2), (3, 4), (4, 5)}. A cover set C_R={F_1, F_2}, F_1={R_1, R_2}, F_2={R_1, R_3, R_4} is get. Since $|F_1|$=2, $|F_2|$=3, F_2 is the replica maximum cover of R.

An algorithm for seeking the replica maximum cover of a relation, *Replica Maximum Cover Algorithm* (*RMCA*), is described as follows.

Algorithm 1. *Replica Maximum Cover Algorithm, RMCA*

INPUT: a relation R and a set Φ_R={R_1, R_2, ..., R_m}
OUTPUT: the *Replica Maximum Cover* of R, RMC_R

(1) Sort the elements in Φ_R according to the number of tuples in each partial replicas and get $R^1,R^2,...,R^m$ satisfying $|R^1|\ge|R^2|\ge...\ge|R^m|$
(2) Let $RMC_R=\varnothing$
(3) **FOR** i=1 **TO** m **DO**
(4) Delete R^i from Φ_R, let $\Phi_R=\Phi_R-\{R^i\}$
(5) **IF** ($R\ne(RMC_R\cup\Phi_R)$) **THEN**
(6) Add R^i into RMC_R, let $RMC_R=RMC_R\cup\{R^i\}$
(7) END IF
(8) END FOR
(9) **RETURN** RMC_R

Since in the algorithm *RMCA*, each element in Φ_R={R^1, R^2, ..., R^m} needs to be executed once from step (3) to step (8), the time complexity of algorithm *RMCA* is O(m).

When selecting operand relations for aggregation-join query, it is unnecessary take all partial replicas of R and S as operand relations. We only need to select some optimal partial replicas from $R'_1,R'_2,...,S'_{m1}$ and $S'_1,S'_2,...,S'_m$ as operand relations and transfer their tuples to computational nodes. But the selection of optimal partial replica of R and S depends on a lot of factors, such as sizes of selected partial replicas, network bandwidth, and computational capacities of m available computational nodes. Since the behaviors of grid resources are unpredictable and volatile, some factors can not be determined when we select optimal replicas. For maximizing parallelism of the aggregation-join query, we only take the sizes of partial replicas into consideration while selection optimal partial

replicas. And thus, the problem is transformed into Replica Maximum Cover problem. We use a method to solve this problem and get an acceptable and feasible solution within reasonable time, which may not be optimal.
The problem is formulated as follows.

INPUT: (1) relations R and S, as well as the sets $\Phi_R=\{R_1,R_2,...,R_{m1}\}$ and $\Phi_S=\{S_1,S_2,...,S_{m2}\}$
(2) C_R and C_S which involve all covers of R and S

OUTPUT: the optimal partial replicas of R and S, RMC_R and RMC_S, satisfying $C((RMC_R,RMC_S))=Min(C(C_R,C_S))$, C: $C_R\times C_S \rightarrow Q^+$, $\forall(RMC_R, RMC_S)\in C_R\times C_S$, $\exists$ minimum cost $C((RMC_R, RMC_S))$.

The *Replica Maximum Covers* of R and S are got by using algorithm *RMCA* respectively. For simplicity and without loss of generality, we assume $RMC_R=\{R'_1,R'_2,...,R'_{n1}\}$ and $RMC_S=\{S'_1,S'_2,...,S'_{n2}\}$.

3.3 Duplication Removals in Optimal Partial Replicas

Since two tuples t_R in R and t_S in S satisfying join condition $\wp(T)$ may exist in several partial replicas of R and S, i.e. $t_R\in R'_{a_1},R'_{a_2},...,R'_{a_c}(1\le a_i\le n_1,1\le i\le c)$, $t_S\in S'_{b_1},S'_{b_2},...,S'_{b_d}$ $(1\le b_j\le n_2,1\le j\le d)$, and each pair of partial replicas of R and S R'_{a_i} and S'_{b_j} is transferred to one join node to perform join operations, join result $t_R \rhd\lhd t_S$ may be generated many times at multiple different join nodes. For avoiding generating duplicate join results, it is necessary to remove duplicate tuples in $R'_1,R'_2,...,R'_{n_1}$. Such that for any t_R ($t_R\in R'$), it only exists one partial replica of R. Similarly, we need to remove duplication in partial replicas of S. And thus, each result of $R \rhd\lhd S$ is sure to be generated once at all join nodes.

We adopt the following method to remove duplication in selected optimal partial replicas. Assume the average cost for processing a tuple at node NR_a is T_{NR_a} and the average cost for processing a tuple at node NR_b is T_{NR_b}. If $R'_a\cap R'_b\neq\varnothing$ $(1\le a,b\le n_1, a\neq b)$ and $T_{NR_a}\le T_{NR_b}$, duplicate tuple t is removed from R'_b. Similarly, duplicate tuples in partial replicas of S are processed.

Proposition 1 $(R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (S_1\cup S_2\cup...\cup S_{n_2})=(R''_1\cup R''_2\cup...\cup R''_{n_1}) \rhd\lhd (S''_1\cup S''_2\cup...\cup S''_{n_2})$, where R''_1, R''_2, ..., R''_{n_1} and S''_1, S''_2, ..., S''_{n_2} are tuple sets after reduction and duplicate removals, satisfying $R''_a\cap R''_b=\varnothing$, $S''_i\cap S''_j=\varnothing$ $(1\le a, b\le n_1, a\neq b, 1\le i, j\le n_2, i\neq j)$.

Proof: According to above algorithm, we have

$$R'_a=R_a \rhd\lhd (S_1[T]\cup S_2[T]\cup...\cup S_{n_2}[T]) \ (1\le a\le n_1),$$

$$S'_b=S_b \rhd\lhd (R_1[T]\cup R_2[T]\cup...\cup R_{n_1}[T]) \ (1\le b\le n_2)$$

Thus, $R'_1\cup R'_2\cup...\cup R'_{n_1}$
$=(R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (S_1[T]\cup S_2[T]\cup...\cup S_{n_2}[T])$
Similarly, we have
$(S'_1\cup S'_2\cup...\cup S'_{n_2})=(S_1\cup S_2\cup...\cup S_{n_2}) \rhd\lhd (R_1[T]\cup R_2[T]\cup...\cup R_{n_1}[T])$
And thus, $(R'_1\cup R'_2\cup...\cup R'_{n_1}) \rhd\lhd (S'_1\cup S'_2\cup...\cup S'_{n_2})$
$=((R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (S_1[T]\cup S_2[T]\cup...\cup S_{n_2}[T])) \rhd\lhd ((S_1\cup S_2\cup...\cup S_{n_2}) \rhd\lhd (R_1[T]\cup R_2[T]\cup...\cup R_{n_1}[T]))$
By the join-associative property and the join-commutative property, we have
$(R'_1\cup R'_2\cup...\cup R'_{n_1}) \rhd\lhd (S'_1\cup S'_2\cup...\cup S'_{n_2})$
$=((R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (R_1[T]\cup R_2[T]\cup...\cup R_{n_1}[T])) \rhd\lhd ((S_1\cup S_2\cup...\cup S_{n_2}) \rhd\lhd (S_1[T]\cup S_2[T]\cup...\cup S_{n_2}[T]))$
Since $R_a[T]$ is the projection of R_a on T, we have $R_a[T] \rhd\lhd R_a=R_a$ and

$$R_a[T] \rhd\lhd R_b=\begin{cases} SR_b \quad SR_b\subseteq R_b & (R_a[T] \rhd\lhd R_b[T]\neq\varnothing) \\ \varnothing & (R_a[T] \rhd\lhd R_b[T]=\varnothing) \end{cases}$$

Similarly, we have $S_i[T] \rhd\lhd S_i=S_i$ and

$$S_i[T] \rhd\lhd S_j=\begin{cases} SS_j \quad SS_j\subseteq S_j & (S_i[T] \rhd\lhd S_j[T]\neq\varnothing) \\ \varnothing & (S_i[T] \rhd\lhd S_j[T]=\varnothing) \end{cases}$$

By the join-associative property and the join-commutative property, we have
$(R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (R_1[T]\cup R_2[T]\cup...\cup R_{n_1}[T])$
$=R_1\cup R_2\cup...\cup R_{n_1}$
$(S_1\cup S_2\cup...\cup S_{n_2}) \rhd\lhd (S_1[T]\cup S_2[T]\cup...\cup S_{n_2}[T])$
$=S_1\cup S_2\cup...\cup S_{n_2}$
Thus, we have
$(R'_1\cup R'_2\cup...\cup R'_{n_1}) \rhd\lhd (S'_1\cup S'_2\cup...\cup S'_{n_2})$
$=(R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (S_1\cup S_2\cup...\cup S_{n_2})$ (1)
Since for any $t_R\in R'_1\cup R'_2\cup...\cup R'_{n_1}$, R''_a exists and satisfies $t_R\in R''_a$, we have

$$R'_1\cup R'_2\cup...\cup R'_{n_1}=R''_1\cup R''_2\cup...\cup R''_{n_1}$$

Similarly, we have

$$S'_1\cup S'_2\cup...\cup S'_{n_2}=S''_1\cup S''_2\cup...\cup S''_{n_2}$$

And thus, we get
$(R'_1\cup R'_2\cup...\cup R'_{n_1}) \rhd\lhd (S'_1\cup S'_2\cup...\cup S'_{n_2})$
$=(R''_1\cup R''_2\cup...\cup R''_{n_1}) \rhd\lhd (S''_1\cup S''_2\cup...\cup S''_{n_2})$ (2)
According to equations (1) and (2), we have
$(R_1\cup R_2\cup...\cup R_{n_1}) \rhd\lhd (S_1\cup S_2\cup...\cup S_{n_2})$
$=(R''_1\cup R''_2\cup...\cup R''_{n_1}) \rhd\lhd (S''_1\cup S''_2\cup...\cup S''_{n_2})$

4. Selecting Join Nodes

This section first gives the cost model for performing join operations at join nodes. Then the method for selecting

join nodes based on the concept of *Minimum Maximum Edge Matching* (*MMEM*) is introduced.

4.1 Cost Model

Assume efficient tuple sets $R''_a(1\le a\le n_1)$ and $S''_b(1\le b\le n_2)$, which are a pair of partial replicsin R and S, are transferred to node JN_k to perform join operations. For R''_a and S''_b, only the tuples in $R''_a(S_b)$ and $S''_b(R_a)$ satisfies the join condition $\wp(T)$, $(R''_a(S_b)\subseteq R''_a, S''_b(R_a)\subseteq S''_b)$. So we only need to transfer $R''_a(S_b)$ and $S''_b(R_a)$ to join node JN_k to perform join operations instead of transferring all tuples in R''_a and S''_b to join node.

The load of join node JN_k is described by the vector $<CPU_{JN_k},Mem_{JN_k},I/O_{JN_k},TR_{N\to JN_k}>$, where CPU_{JN_k}, Mem_{JN_k} and I/O_{JN_k} are available CPU power, available memory and disk bandwidth, $TR_{N\to JN_k}$ is the average network bandwidth between node N and node JN_k.

The cost for performing join of $R''_a(S_b)$ and $S''_b(R_a)$ at JN_k consists of two parts: computational cost $T_{compu}(R''_a(S_b), S''_b(R_a),JN_k)$ and communication cost $T_{commu}(R''_a(S_b), S''_b(R_a),JN_k)$. Communication cost is the time for parallel transferring $R''_a(S_b)$ from node NR_a to node JN_k and transferring $S''_b(R_a)$ from node NS_b to node JN_k, i.e.

$$T_{commu}(R''_a(S_b),S''_b(R_a),JN_k) = Max\{C_0+|R''_a(S_b)|\times TR_{NR_a\to JN_k},C_0+|S''_b(R_a)|\times TR_{NR_b\to JN_k}\}$$

Computational cost is the time for completing the aggregation-join operations of $R''_a(S_b)$ and $S''_b(R_a)$ at node JN_k, i.e.

$$T_{compu}(R''_a(S_b),S''_b(R_a),JN_k) = (|R''_a(S_b)|/|T(R)|+|S''_b(R_a)|/|T(S)|)\times(T_{compa(JN_k)}+T_{move(JN_k)}),$$

Where $T(R)$ and $T(S)$ respectively denotes sizes of each tuple in R and S, $T_{compa(JN_k)}$ denotes the time for completing a comparison operation in memory of JN_k, $T_{move(JN_k)}$ denotes the time for moving a tuple in memory of JN_k. And thus, the total cost

$$TC(R''_a(S_b),S''_b(R_a),JN_k) = T_{commu}(R''_a(S_b),S''_b(R_a),JN_k)+T_{compu}(R''_a(S_b),S''_b(R_a),JN_k)$$

4.2 Selecting Join Nodes based on *MMEM*

Since the tuples in each pair of partial replicas $R''_a(S_b)$ and $S''_b(R_a)$ $(1\le a\le n_1,1\le b\le n_2)$ are transferred to a join node to perform join operations, at most $n_1\times n_2$ nodes need to be selected as join nodes from m available nodes. Each selected join node corresponds to a pair of partial replicas.

To minimize the cost for parallel performing join operations, the problem of selecting join nodes is formulated as the problem of seeking a *MMEM* in a weighted complete bipartite graph [8,9]. An weighted complete bipartite graph $G=(A,B,E,\varphi)$ shown in Fig.2 is constructed in the following four steps.

Step 1. $A=\{A_1, A_2, ..., A_{n_1\times n_2}\}$, where the pair of grid nodes, at which partial replicas $R''_a(S_b)$ and $S''_b(R_a)$ $(1\le a\le n_1,1\le b\le n_2)$ locate, is considered as a vertex in a vertex set A, i.e. $A_1=(NR_1, NS_1)$, $A_2=(NR_1, NS_2)$, ..., $A_{n_1\times n_2}=(NR_{n_1}, NS_{n_2})$.

Step 2. $B=\{JN_1, JN_2, ..., JN_m\}$, where each node in m available nodes is considered as a vertex in vertex set B.

Step 3. $E=\cup_{\forall x\in A}\{(x,y) \mid \forall y\in B\}$

Step 4. φ: $E\to\Re$, where $\Re$ is the set of real numbers, and $\varphi((A_i, JN_k))=t$ means the cost of computation and communication for performing aggregation join operations at node JN_k is t.

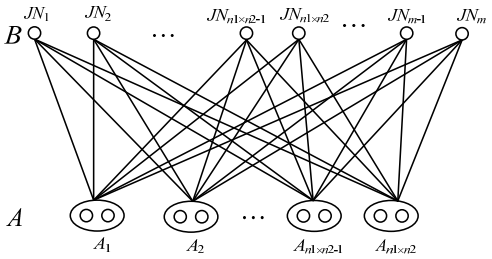


Fig.2 A constructed weighted complete bipartite graph G

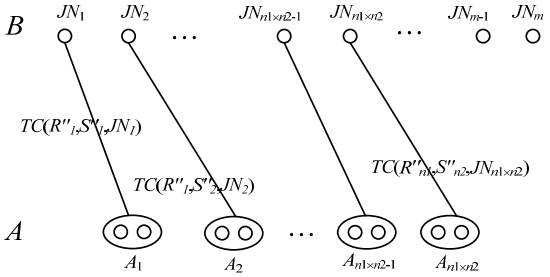


Fig.3 The desired *MMEM* in G

When the weighted complete bipartite graph G is constructed, we use the method proposed in [8,9] to seek a *MMEM* M in G. Once M is found, $n_1\times n_2$ vertexes in vertex set B correspond to the desired $n_1\times n_2$ join nodes. Assume $M=\{(A_1, JN_1), (A_2, JN_2),..., (A_{n_1\times n_2}, JN_{n_1\times n_2})\}$ is shown in Fig.3 and $JN_1,JN_2,...,JN_{n_1\times n_2}$ are the desired join nodes.

5. Executing Aggregation-join Operations

5.1 Executing Join Operations

For any $t_R\in R''$, t_R only exists in a partial replica R''_a $(1\le a\le n_1)$. But t_R may exists in several different efficient tuple sets $R''_a(S_1)$, $R''_a(S_2)$,..., $R''_a(S_{n2})$ at the same grid node N_a. Similarly, t_s may exists in several different efficient tuple sets $S''_b(R_1)$, $S''_b(R_2)$,...,$S''_b(R_{n1})$ at the same

grid node N_b. Since each pair of $R''_a(S_i)$ and $S''_b(R_j)$ is transferred to a join node to performing join operations, the join result $t_R \rhd\lhd t_s$ may be generated many times at different join nodes.

Assume $R''_a(S_b)$ and $S''_b(R_a)$ are transferred to JN_k to perform join operations. At JN_k, the join result of $R''_a(S_b)$ and $S''_b(R_a)$, $R''_a(S_b) \rhd\lhd S''_b(R_a)$, are get by using "block merge join" algorithm at join node JN_k [9].

According to the values on group-by attribute *R.GB* in join results, we partition an output buffer $OB(GB_i(JN_k))$ for each group in the memory of node JN_k. When the size of join results in $OB(GB_i(JN_k))$ arrives at the size of one block, the join results in $OB(GB_i(JN_k))$ are transferred to corresponding aggregation node AN_i. Assume there are L groups on the group-by attribute *R.GB* in $R'' \rhd\lhd S''$, we at most need to select L grid nodes as aggregation nodes.

5.2 Executing Aggregation Operations

Aggregation operations are processed at aggregation nodes in the following way.

The memory space at AN_j is partitioned into two parts, named $IB(AN_j)$ and $OB(AN_j)$. $IB(AN_j)$ is used for buffering arrived join results and $OB(AN_j)$ is used for storing join results which have finished being grouped and aggregated. For each aggregation node AN_j, its initial value equals to 0, i.e. $AggVal(AN_j))=0$.

Assume a block from $GB_i(JN_k)$ arrives at node AN_j, first, the arrived block is inserted into the input buffer $IB(AN_j)$. For any join result $t=(A[t],GB[t], ...)$, we check whether it exists in $OB(AN_j)$ or not. If not, insert $t=(A[t],GB[t], ...)$ into $OB(AN_j)$, compute $AggVal(AN_j)) = AggVal(AN_j)) + A[t]$ and remove $t=(A[t],GB[t], ...)$ from $IB(AN_j)$; otherwise, remove $t=(A[t],GB[t], ...)$ from $IB(AN_j)$ directly. Thus, even the same join results are generated many times at different join nodes, aggregation result is computed only once.

6. Adaptive Adjustment of Query Processing

6.1 Adaptively Adjusting the Selection of Join Nodes

Since data grids is an unpredictable and volatile computational environment, the loads of selected join nodes $JN_1,JN_2, ..., JN_{n_1 \times n_2}$ may vary a lot with time, i.e. available CPU power degrade and available memory space become less and so on. If the remained tuples in $R''_1,R''_2, ...,R''_{n1}$ and $S''_1, S''_2,...,S''_{n2}$ at $NR''_1,NR''_2,...,NR''_{n1}$ and $NS''_1, NS''_2,...,NS''_{n2}$ are still transferred to $JN_1,JN_2,...,JN_{n1\times n2}$ to continue performing join operations, the performance of the aggregation-join query may degrades. So we have to determine whether to select at most $n_1 \times n_2$ new join nodes $JN'_1,JN'_2,...,JN'_{n_1 \times n_2}$ and to transfer remained tuples to $JN'_1,JN'_2,...,JN'_{n_1 \times n_2}$ for continuing performing join operations.

Assuming that the remained tuple sets at $NR_1,NR_2,...,NR_{n1}$ and $NS_1,NS_2,...,NS_{n2}$ are $LR''_1,LR''_2,...,LR''_{n1}$ and $LS''_1,LS''_2,...,LS''_{n2}$ respectively. We have to re-construct a weighted complete bipartite graph $G'=(A,B,E,\varphi)$, where

(1) $A=\{A_1,A_2,...,A_{n_1 \times n_2}\}$ is a vertex set, where $A_1= (NR_1, NS_1),A_2=(NR_1,NS_2),..., A_{n_1 \times n_2}=(NR_{n_1},NS_{n_2})$

(2) $B=\{JN_1, JN_2, ..., JN_m\}$ is a vertex set, each vertex is an available join node

(3) $E=\cup_{\forall x \in A}\{(x,y) \mid \forall y \in B\}$, and

(4) $\varphi: E \rightarrow \Re$, where $\Re$ is the set of real numbers, and $\varphi((A_i,JN_k))=t$ means the cost of completing the join of LR''_a and LS''_b at JN_k is t.

If the *MMEM M'* of G' is same to the *MMEM M* of G, the remained tuples in $LR''_1, LR''_2,...,LR''_{n_1}$ and $LS''_1,LS''_2,..., LS''_{n_2}$ continue being transferred to $JN_1,JN_2,...,JN_{n_1 \times n_2}$ for performing join operations. Otherwise, the remained tuples are to be transferred to $JN'_1,JN'_2,...,JN'_{n_1 \times n_2}$ for performing join operations. This ensures the join operations are always parallel performed at most efficient join nodes and the time cost of performing join operations is decreased. While join results are being generated at join nodes, they still transferred to original selected aggregation nodes (*AN*s) for completing aggregation operations.

6.2 Dealing with Possible Lost Join Results

When the join node for performing join of $R''_a(S_b)$ and $S''_b(R_a)$ is changed from JN_k to JN_h, some join results in $R''_a(S_b) \rhd\lhd S''_b(R_a)$ may be lost. Assuming $HR''_a(S_b)$ and $HS''_b(R_a)$ are transferred tuples sets to JN_k, $R''_a(S_b)_{Val}$ represents the tuple sets having T value equal to Val in $R''_a(S_b)$, i.e. for any $t \in R''_a(S_b)_{Val}$, $t[T]=Val$. $||R''_a(S_b)_{Val}||$ represents the number of tuples in $R''_a(S_b)_{Val}$.

The method for avoiding losing some join results is composed of the following three phases.

Phase 1. For each value Val on the join attribute T of $R''_a(S_b) \rhd\lhd S''_b(R_a)$, we first record $||R''_a(S_b)_{Val}||$ and $||S''_b(R_a)_{Val}||$, as well as $||HR''_a(S_b)_{Val}||$ and $||HS''_b(R_a)_{Val}||$, where $HR''_a(S_b)_{Val}$ and $HS''_b(R_a)_{Val}$ are tuple sets having T value equal to Val which have been transferred to JN_k.

Phase 2. When determining whether to adjust join nodes, we compute the number of tuples having T value equal to Val in join results which have been generated at JN_k i.e. $GenTN_1=||HR''_a(S_b)_{Val}|| \times ||HS''_b(R_a)_{Val}||$, and compute the number of tuples having T value equal to Val in join results which will be generated at JN_h, i.e. $GenTN_2= (||R''_a(S_b)_{Val}||-$

$||HR''_a(S_b)_{Val}||)\times(||S''_b(R_a)_{Val}||-||HS''_b(R_a)_{Val}||)$

Phase 3. Compare $GenTN_1$ and $GenTN_2$.

(3.1) If $GenTN_1 \geq GenTN_2$, the tuples in $LR''_a(S_b)_{Val}$ continue being transferred to JN_k to perform join operations. When no tuples having T value equal to Val in $LR''_a(S_b)_{Val}$ at NR_a exist, i.e. $LR''_a(S_b)_{Val}=\varnothing$, we begin to transfer tuples in $LR''_a(S_b)$ from NR_a to JN_h for performing join operations.

(3.2) If $GenTN_1 < GenTN_2$, the tuples having T value equal to Val which have been transferred to JN_k must be re-transferred to JN_h to re-generate join results and delete generated join results $(HR''_a(S_b) \rhd\lhd HS''_b(R_a))_{Val}$ at join node JN_k

Phase 4. Similarly, the tuples in $LS''_b(R_a)_{Val}$ are processed.

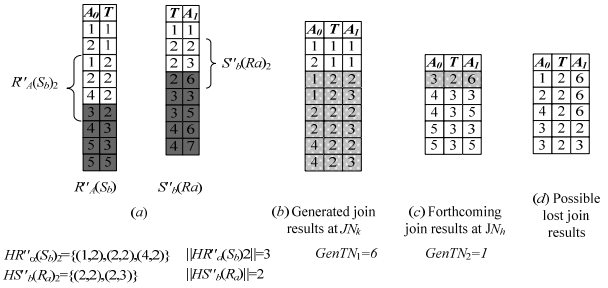


Fig.4 Dealing with possible lost join results

In the example shown in Fig.4, the tuples without background are transferred ones, and the tuples in gray are remained ones shown in Fig.4(a). Fig.4(b) shows the generated join results at JN_k, Fig.4(c) shows the forthcoming join results at JN_h and Fig.4(d) shows the possible lost join results. The tuples (3,2) in $LR''_a(S_b)$ and (2,6) in $LS''_b(R_a)$ continue being transferred to execution node JN_k to perform join operations. And, the tuples (4,3),(5,3) and (5,5) in $LR''_a(S_b)$, as well as the tuples (3,3),(3,5),(4,6) and (4,7) in $LS''_b(R_a)$ are transferred to new selected execution node JN_h to generate join results.

7. Experimental Results and Analysis

This section first describes our experimental setup, and then presents experimental results and analysis.

7.1 Experimental Setup

We built a simulation environment for conducting performance study. The system was implemented using Java, and is composed of 20 nodes which are interconnected with a 100M LAN. Configurations of the system are shown as Table 1.

Query. An aggregation-join query Q, being required to get aggregation value on $S.A$ after grouping $R \rhd\lhd_T S$ on $R.GB$.

Datasets. Four partial replicas of R, R_1,R_2,R_3,R_4 locate N_1,N_2,N_3,N_4 and each tuple of R comprises five integers(GB, A_1, A_2, A_3, T). Five partial replicas of S, S_1,S_2,S_3,S_4,S_5 locate N_5,N_6,N_7,N_8,N_9 and each tuple of S comprises four integers(T, A, B_1, B_2). Each partial replica is stored at one node as a textual file, the columns and rows of which correspond to the attributes and tuples.

Transfer of datasets. We use the *java.nio* package for transferring data sets across the network. The package provides efficient API's for unblocking network I/O. Different network transfer rates between computer nodes are devised through applying delay to data transmission.

Table 1 Configurations of the nodes in the aggregation-join query system

NAME	DESCRIPTION	CPU	MEMORY
N_0	User node, a user issues a query and gets the results at this node	1.8GHz	256M
N_1~N_4	Nodes where partial replicas of R locate	2.4GHz	512M
N_5~N_9	Nodes where partial replicas of S locate	2.4GHz	512M
N_{10}～N_{19}	Available join nodes and aggregation nodes	3.2GHz	2GB

7.2 The Impact of Replica Maximum Cover

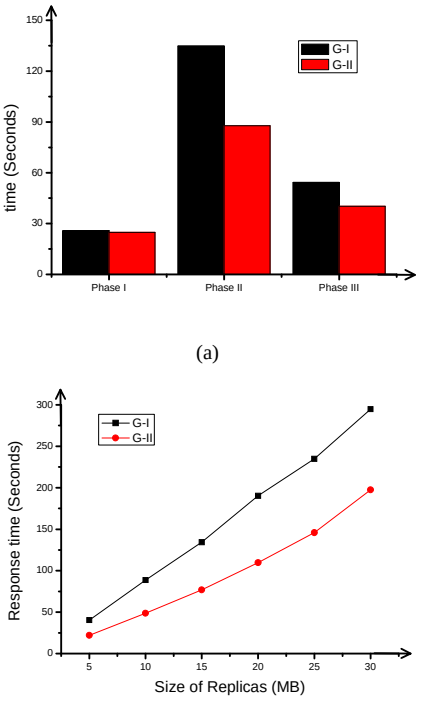


Fig.5 Effect of maximum replica cover algorithm

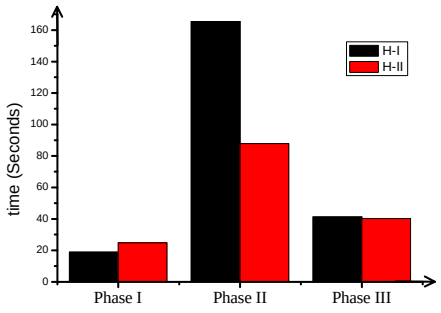
The experiment focuses on analyzing the effect on the query response time of the algorithm *RMCA* by two approaches. Commonly used *Set Cover* algorithm is

adopted in approach G-I, i.e. to seek a relation replica cover in which the number of partial replicas is minimal, and the algorithm *RMCA* is used in approach G-II.

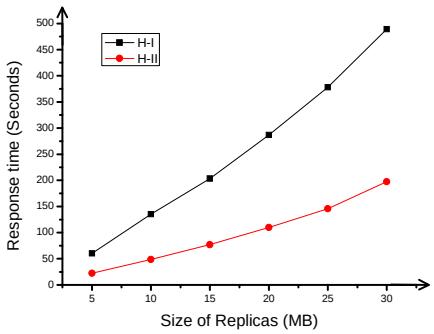
The process of aggregation-join query is decomposed into three phases: Phase I includes getting efficient tuple sets, selecting optimal replicas and removing duplication in selected partial replicas; Phase II includes parallel transferring tuples to join nodes and aggregation nodes, as well as parallel performing join operations and aggregation operations at selected computational nodes; transferring final aggregation-join results to user node is included in Phase III.

Fig.5 shows that, in Phase I the time costs between G-I and G-II are nearly same and in Phase II the time cost in G-I is higher than that in G-II. This is because the number of optimal replicas in G-I is less than that in G-II, thus the size of the maximal replica in G-I is much larger than that in G-II and the time cost for parallel transferring tuples and parallel performing aggregation-join operations in G-I is much higher than that in G-II. In Phase III, the time cost in G-I is still higher than that in G-II for the same amount of join results are parallel transferred to user node from less execution nodes in G-I than that in G-II. This shows our algorithm *RMCA* has good performance on query response time.

7.3 The Impact of Obtaining Efficient Tuple



(a)



(b)

Fig.6 Effect of efficient tuple sets

The experiment mainly analyzes the effect on the query response time of the algorithm *OET* by two approaches. In approach H-I, the algorithm *OET* is not used and the tuples are directly transferred to execution nodes to perform join operations, while in approach H-II, the tuples generated by the algorithm *OET* are transferred to join nodes to perform join operations. The process of aggregation-join query is also divided into three phases.

Fig.6 shows that, In Phase I, the time cost in H-I is less than that in H-II because the algorithm *OET* is used in H-II and it costs some time to get efficient tuples. In Phase II, the time cost in H-I is much higher than that in H-II. This is because all the tuples in partial replicas in H-I are transferred to join nodes whether they satisfy the join condition or not, and participant in the join operations at join nodes. Thus, the time costs for transferring tuples and performing join operations is much higher in H-I than that in H-II. In Phase III, since the same amount of final join results are sent to user node from the same amount of aggregation nodes in H-I and H-II, the time costs in H-I and H-II are nearly same. These experiments results show that our algorithm performs well and provides efficient support for minimizing the transfer cost.

7.4 The Impact of Adaptive Adjustment of Query Processing

Two approaches J-I and J-II are studied in this experiment to analyze the query performance related to the algorithm for adaptively adjustment of the query processing. In J-I, the join nodes are not adjusted without considering whether their loads are varied or not, i.e. once join nodes are selected, the join operations are performed at them all the time. We adaptively adjust the selection of join nodes according to their loads in J-II.

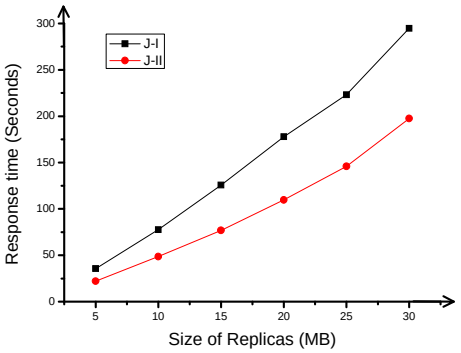


Fig.7 Effect of adjusting execution nodes

In the experiments, we adjust the selection of join nodes with intervals of 20 seconds from the beginning of transferring tuples to join nodes. As Fig.7 shows, the response time of the query increases as the sizes of

replicas become larger and the performance of J-II is well than that of J-I. Although some time is cost in J-II for adjusting the selection of join nodes and avoiding losing some join results, the time cost in J-II is much less than that in J-I. This is because load degradation of the join nodes in J-I causes much higher time cost.

8. Conclusion

The heterogeneous, unpredictable and volatile behavior of grid resources has posed new challenges to the query processing in data grids. This paper investigates the problem and proposes a novel approach for processing aggregation-join query exploring relation partial replicas and load balancing in data grids. Analytical and experimental results show that the approach has high performance. Nevertheless, there are still a number of aspects that require further investigation for the improvement of the aggregation-join query processing in data grids. For example, it is not well understood how to take relation replicas and load balancing into consideration in processing multi-join queries. In future work, we will address this issue.

References

[1] I. Foster, C. Kesselman. “The Grid 2: Blueprint for a New Computing Infrastructure”. San Francisco, CA: Morgan Kaufmann, 2003.

[2] A. Chervenak, I. Foste, C. Kesselman, et al. “The Data Grid: Towards an architecture for the Distributed Management and Analysis of Large Scientific Datasets”. Journal of Network and Computer Applications, 2001, 23: 187-200.

[3] J. Smith, P. Watson, A. Gounaris, N.W. Paton, A.A.A Fernandes, R. Sakellariou. “Distributed Query Processing on the Grid”. International Journal of High Performance Computing Applications, 2003, 179(4): 353-367.

[4] M.N. Alpdemir, A. Mukherjee, N.W. Paton, P. Watson, A.A.A. Fernandes, A. Gounaris, J. Smith. “Service-based Distributed Querying on the Grid”. Proceedings of the First International Conference on Service Oriented Computing. Heidelberg: Springer-Verlag, 2003. 467-482.

[5] A. Gounaris. “Resource Aware Query Processing on the Grid”. Ph.D Thesis.

[6] S. Vazhkudai, S. Tuecke, I. Foster. “Replica Selection in the Globus Data Grid”. Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid. Bangalore: IEEE Computer Society Press, 2001. 106~113.

[7] G. Anastasios, W.P. Norman, S. Rizos, A.A.A. Fernandes. “Adaptive Query Processing and the Grid: Opportunities and Challenges”. Proceedings of the 1st International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems. Bangalore: IEEE Computer Society, 2004. 506~510.

[8] D.H. Yang, J.Z. Li. “Distributed Multi-join Query Processing in Data Grids”. Information Sciences [J]. 177(1), 2007:3574–3591.

[9] D.H. Yang, J.Z. Li, Q. Rasool. “Join Algorithm Using Multiple Replicas in Data Grid”. Proceedings of the International Conference on Advances in Web-Age Information Management, Heidelberg: Springer-Verlag, 2005. 416-427.

Hua Feng received the B.S. and M.S. degrees in Mechanical Engineering from Jiamusi University in 1998 and Daqing Petroleum Institute in 2007, respectively. His research work focuses on query processing and data mining in wide-area network environments.

Zhenhuan Zhang received the B.S. degrees in Mechanical Engineering from Heilongjiang University in 1998. She now is an M.S candidate in Jilin University. Her research interest is in database and artificial intelligence.