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 grids 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 grid 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
Considering the existence of multiple replicas of datasets, 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 \( S \), \( R_{r1}, R_{r2}, \ldots \), \( R_{rn} \) and \( S_{s1}, S_{s2}, \ldots, S_{sm} \) which locate different grid nodes.

(3) Join Attribute Group-by attribute \( T \) attributed to relations \( R \) and \( S \).

**OUTPUT:** Aggregation-join results of \( R \bowtie S \) according to \( T \) and \( S \).

Step 1. Reduce the sizes of partial replicas \( R_{r1}, R_{r2}, \ldots, R_{rn} \) and \( S_{s1}, S_{s2}, \ldots, S_{sm} \) and get efficient tuple sets \( R_{1}', R_{2}', \ldots, R_{m}', S_{1}', S_{2}', \ldots, S_{m}' \). The problem of aggregation-join query is not necessary to transfer all tuples in each partial replica.

Step 2. Respectively get the projection of each partial replica \( R_{ri}' \) as operand relations, satisfying \( R'=R_{i1}' \cup R_{i2}' \cup \ldots \cup R_{in}' \) and \( S'=S_{i1}' \cup S_{i2}' \cup \ldots \cup S_{im}' \).

Step 3. Remove duplication from operand relations \( R_{i1}', R_{i2}', \ldots, R_{in}' \) and get \( R''_{i1}', R''_{i2}', \ldots, R''_{in}' \) satisfying \( R''=R_{i1}' \cup R_{i2}' \cup \ldots \cup R_{in}' \). Similarly, remove duplication from operand relations \( S''_{i1}', S''_{i2}', \ldots, S''_{in}' \) and get \( S'''_{i1}', S'''_{i2}', \ldots, S'''_{in}' \).

Step 4. Select at most \( n_1 \times n_2 \) grid nodes as join nodes (JNs) from \( m \) available ones, and transfer efficient tuple sets \( R''_{i1}', R''_{i2}', \ldots, R''_{in}' \) and \( S'''_{i1}', S'''_{i2}', \ldots, S'''_{in}' \) to join nodes for parallel performing join operations.

Step 5. According to the group-by attribute \( T \), select aggregation nodes (ANs) from \( m \) available grid nodes and transfer join results to ANs for completing aggregation operations.

Step 6. Parallel transfer aggregation-join results from ANs to user node in pipeline.

### 3. Pre-Processing of Relation Partial Replicas

This section first discusses how to get efficient tuple sets \( R_{r1}', R_{r2}', \ldots, R_{rn}' \) and \( S_{s1}', S_{s2}', \ldots, S_{sm}' \) from partial replicas \( R_{r1}, R_{r2}, \ldots, R_{rn} \) and \( S_{s1}, S_{s2}, \ldots, S_{sm} \). Then the concept of Replica Maximum Cover is proposed and an algorithm for seeking optimal partial replicas as operand relations is presented. 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., \( R_{r1}, R_{r2}, \ldots, R_{rn} \) and \( S_{s1}, S_{s2}, \ldots, S_{sm} \).

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-Efficient-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 \( T \), \( R_{i1}[T], R_{i2}[T], \ldots, R_{im}[T] \) and \( S_{i1}[T], S_{i2}[T], \ldots, S_{im}[T] \), at grid nodes \( NR_{r1}, NR_{r2}, \ldots, NR_{rn} \) and \( NS_{s1}, NS_{s2}, \ldots, NS_{sm} \).

**Step 2.** Parallel execute (2.1) and (2.2)

- Parallel execute (2.1.1)-(2.1.m)
  - Parallel transfer \( R_{i1}[T], R_{i2}[T], \ldots, R_{im}[T] \) from nodes \( NR_{r1}, NR_{r2}, \ldots, NR_{rn} \) to node \( NS_{s1} \)
  - Parallel transfer \( R_{i1}[T], R_{i2}[T], \ldots, R_{im}[T] \) from nodes \( NR_{r1}, NR_{r2}, \ldots, NR_{rn} \) to node \( NS_{s2} \)

- Parallel execute (2.1.m)-(2.1.m2)
  - Parallel transfer \( R_{i1}[T], R_{i2}[T], \ldots, R_{im}[T] \) from nodes \( NR_{r1}, NR_{r2}, \ldots, NR_{rn} \) to node \( NS_{s1} \)

**Step 2.** Parallel execute (2.2.1)-(2.2.m2)

- Parallel transfer \( S_{i1}[T], S_{i2}[T], \ldots, S_{im}[T] \) from nodes \( NS_{s1}, NS_{s2}, \ldots, NS_{sm} \) to node \( NR_{r1} \)
- Parallel transfer \( S_{i1}[T], S_{i2}[T], \ldots, S_{im}[T] \) from nodes \( NS_{s1}, NS_{s2}, \ldots, NS_{sm} \) to node \( NR_{r2} \)
Definition 1 (Replica Cover): Given a relation \( R \), two sets \( \Phi = \{R_1, R_2, ..., R_m\} \) and \( F = \{R^1, R^2, ..., R^n\} \), \( R \subseteq R \), \( R^i \subseteq R \), \( 1 \leq i \leq m \) and \( 1 \leq j \leq n \), \( F \cap \Phi = \emptyset \). \( F \) covers \( R \) if the following conditions are satisfied:

1. \( \forall t \in R, \exists R' \in F \) satisfying \( t \in R' \)
2. \( R = R^1 \cup R^2 \cup ... \cup R^n \)
3. \( \forall R' \) and \( R^i \) (\( R^i \in F, R^i \in F, i \neq j \)), satisfy \( R' \cap R^i \) and \( R' \cap R^j \)

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 \( R \subseteq R \), 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 \( |F_i| = \max (|F|) \quad (|F| = |F_1|, |F_2|, ..., |F_n|) \).

Assume a relation \( R = \{0, 1, (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)\} \), \( \Phi = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\} \). \( R_1 \) and \( R_2 \) are partial replicas of \( R \), \( R_1 = \{0, 1\}, R_2 = \{(2, 3), (3, 4), (4, 5), (5, 6)\} \). A cover set \( C_R = \{F_1, F_2\} \), \( F_1 = \{R_1, R_2\} \), \( F_2 = \{R_1, R_3\} \) 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 \( \Phi = \{R_1, R_2, ..., R_m\} \)
OUTPUT: the Replica Maximum Cover of \( R \), \( R_m \)

(1) Sort the elements in \( \Phi \) according to the number of tuples in each partial replica and get \( R^1, R^2, ..., R^n \) satisfying \( |R^1| \leq |R^2| \leq ... \leq |R^n| \)
(2) Let \( RMC = \emptyset \)
(3) FOR \( i = 1 \) TO \( m \) DO
(4) Delete \( R^i \) from \( \Phi \), let \( \Phi = \Phi \setminus \{R^i\} \)
(5) IF \( (R \cap \Phi = \emptyset) \) THEN
(6) Add \( R^i \) into \( RMC \), let \( RMC \cup \{R^i\} \)
(7) END IF
(8) END FOR
(9) RETURN \( RMC \)

Since in the algorithm RMCA, each element in \( \Phi \) covers \( R \), \( R^1, R^2, ..., R^n \) 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 \( R^1, R^2, ..., R^n \) 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.
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 \( \Phi_R = \{ \{ R_1, R_2, \ldots, R_{n1} \} \} \) and \( \Phi_S = \{ \{ S_1, S_2, \ldots, S_{n2} \} \} \)

(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 RMC_A respectively. For simplicity and without loss of generality, we assume \( RMC_R = \{ R'_1, R'_2, \ldots, R'_{n1} \} \) and \( RMC_S = \{ S'_1, S'_2, \ldots, S'_{n2} \} \).

### 3.3 Duplication Removals in Optimal Partial Replicas

Since two tuples \( t_R \) in R and \( t_S \) in S satisfying join condition \( \phi(T) \) may exist in several partial replicas of R and S, i.e. \( t_R \in R'_{a_1} \cup R'_{a_2} \cup \ldots \cup R'_{a_l} (1 \leq a_i \leq n_1, 1 \leq i \leq l) \), \( t_S \in S'_{b_1} \cup S'_{b_2} \cup \ldots \cup S'_{b_m} (1 \leq b_j \leq n_2, 1 \leq j \leq m) \), and each pair of partial replicas of R and S \( R'_{a_j} \) and \( S'_{b_j} \) is transferred to one join node to perform join operations, join result \( t_R \triangleright a \triangleright t_S \) may be generated many times at multiple different join nodes. For avoiding generating duplicate join results, it is necessary to remove duplicate tuples in \( R'_1, R'_2, \ldots, R'_{n1} \). Such that for any \( t_R (t_S \in R') \), it only exists one partial replica of R. Similarly, we need to remove duplication in partial replicas of S. And thus, each result of \( R \triangleright a \triangleright S \) 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 \( NR_R \) is \( T_{NR_R} \) and the average cost for processing a tuple at node \( NR_S \) is \( T_{NR_S} \). If \( R'_a \cap R'_b \not= \emptyset \) (1 \( \leq a, b \leq n_1, a \not= b \)) and \( T_{NR_R} \leq T_{NR_S} \), duplicate tuple \( t \) is removed from \( R'_a \). Similarly, duplicate tuples in partial replicas of S are processed.

**Proposition 1** \((R_1 \cup R_2 \cup \ldots \cup R_{n1}) \triangleright a \triangleright (S_1 \cup S_2 \cup \ldots \cup S_{n2}) = (R'_{a_1} \cup R'_{a_2} \cup \ldots \cup R'_{a_l}) \triangleright \phi(S'_{b_1} \cup S'_{b_2} \cup \ldots \cup S'_{b_m}) \), where \( R'_{a_1}, R'_{a_2}, \ldots, R'_{a_l} \) and \( S'_{b_1}, S'_{b_2}, \ldots, S'_{b_m} \) are tuple sets after reduction and duplicate removals, satisfying \( R'_a \cap R'_b = \emptyset \) (1 \( \leq a, b \leq n_1, a \not= b \), 1 \( \leq i \leq l, j \leq m, i \not= j) \).

**Proof:** According to above algorithm, we have \( R'_a \triangleright a \triangleright (S'_1 \cup S'_2 \cup \ldots \cup S'_{n2}) = (1 \leq a \leq n_1) \)

\( S'_a = S_{b_a} = \phi (R_1 \cup R_2 \cup \ldots \cup R_{n1}) \triangleright a \triangleright (S_1 \cup S_2 \cup \ldots \cup S_{n2}) \) \((1 \leq b \leq n_2) \)

Thus, \( R'_1 \cup R'_2 \cup \ldots \cup R'_{n1} = (R_1 \cup R_2 \cup \ldots \cup R_{n1}) \triangleright a \triangleright (S_1 \cup S_2 \cup \ldots \cup S_{n2}) \)

Similarly, we have \((S'_1 \cup S'_2 \cup \ldots \cup S'_{n2})\)\((S_1 \cup S_2 \cup \ldots \cup S_{n2}) \)

And thus, \( R'_1 \cup R'_2 \cup \ldots \cup R'_{n1} \triangleright a \triangleright (S'_1 \cup S'_2 \cup \ldots \cup S'_{n2}) \)\((S_1 \cup S_2 \cup \ldots \cup S_{n2}) \)

According to equations (1) and (2), we have \( R'_1 \cup R'_2 \cup \ldots \cup R'_{n1} \triangleright a \triangleright (S'_1 \cup S'_2 \cup \ldots \cup S'_{n2}) \)\((S_1 \cup S_2 \cup \ldots \cup S_{n2}) \)

**4. Selecting Join Nodes**

This section first gives the cost model for performing join operations at join nodes. Then the method for selecting
Computational cost is the time for completing the transferring $T_{comm}$ formulated as the problem of seeking a comparison operation in memory of $S_a$ and disk bandwidth, $\text{TR}_{e-a}$. The load of join node $JN_k$ is described by the vector $<\text{CPU}_{JN_k}, \text{Mem}_{JN_k}, \text{TR}_{JN_k}>=<C_0, S_{b}a, \text{TR}_{JN_k}>=<C_0, S_{b}a, (T_{comp}+T_{move})>$. 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.

$$T_{comm}(R'_a(S_b), S'_b(R_a), JN_k) = \max |C_0 + |R'_a(S_b)||TR_{NR_a-JN_k}|C_0 + |S'_b(R_a)||TR_{NS_b-JN_k}|$$

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.

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

Where $|T(R)|$ and $|T(S)|$ respectively denotes sizes of each tuple in $R$ and $S$. $T_{comp,JN_k}$ 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

$$T_{comm}(R'_a(S_b), S'_b(R_a), JN_k) = T_{comm}(R'_a(S_b), S'_b(R_a), JN_k) + T_{comp}(R'_a(S_b), S'_b(R_a), JN_k)$$

4.2 Selecting Join Nodes based on MMEM

Since the tuples in each pair of partial replicas $R'_a(S_b)$ and $S'_b(R_a)$ $(1 \leq a \leq m, 1 \leq b \leq n)$ 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 $G=(A,B,E,\phi)$ 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 $S'_b(R_a)$ $(1 \leq a \leq n_1, 1 \leq b \leq n_2)$ locate, is considered as a vertex in a vertex set $A$, i.e. $A_1=(NR_1, NS_1), A_2=(NR_2, NS_2), ..., A_{n_1 \times n_2}=(NR_{n_1}, NS_{n_2})$.

**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.** $E=\cup_{i \in A} \{x_i \mid \forall y \in B\}$

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

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 $t_k \in R'_a$, $t_k$ only exists in a partial replica $R'_a$ $(1 \leq a \leq m)$. But $t_k$ may exists in several different efficient tuple sets $R'_a(S_1), R'_a(S_2), ..., R'_a(S_{n_2})$ at the same grid node $N_a$. Similarly, $t_k$ may exists in several different efficient tuple sets $S'_b(R_1), S'_b(R_2), ..., S'_b(R_{n_1})$ at the same

[Fig.2 A constructed weighted complete bipartite graph $G$

[Fig.3 The desired MMEM in $G$]
grid node \(N_0\). Since each pair of \(R'_d(S_d)\) and \(S'_d(R_d)\) is transferred to a join node to performing join operations, the join result \(t \leq t_i\) may be generated many times at the different join nodes. Assume \(R'_d(S_d)\) and \(S'_d(R_d)\) are transferred to \(JN_k\) to perform join operations. At \(JN_k\), the join result of \(R'_d(S_d)\) and \(S'_d(R_d)\) 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 \(OB(GB(JN_i))\) for each group in the memory of node \(JN_k\). When the size of join results in \(OB(GB(JN_i))\) arrives at the size of one block, the join results in \(OB(GB(JN_i))\) are transferred to corresponding aggregation node \(AN_i\). Assume there are \(L\) groups on the group-by attribute \(R.GB\) in \(R''_d=S''_d\), 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 \(AN_i\) is partitioned into two parts, named \(IB(AN_i)\) and \(OB(AN_i)\). \(IB(AN_i)\) is used for buffering arrived join results and \(OB(AN_i)\) is used for storing join results which have been finished being grouped and aggregated. For each aggregation node \(AN_i\), its initial value equals to 0, i.e. \(AggVal(AN_i)=0\).

Assume a block from \(GB(JN_i)\) arrives at node \(AN_i\), first, the arrived block is inserted into the input buffer \(IB(AN_i)\). For any join result \(t=(A[t],GB[t],...\), we check whether it exists in \(OB(AN_i)\) or not. If not, insert \(t=(A[t],GB[t],...\) into \(OB(AN_i)\), compute \(AggVal(AN_i)=AggVal(AN_i)+A[t]\) and remove \(t=(A[t],GB[t],...\) from \(IB(AN_i)\); otherwise, remove \(t=(A[t],GB[t],...\) from \(IB(AN_i)\) 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 \(R'_d, S'_d\) at \(n_{1} \times n_{2}\) nodes at \(NR'_d, \{NR'_d, S'_d\}, NR'_d, S'_d, \{NR'_d, S'_d\}\) are still transferred to \(JN_{1}JN_{2}..., JN_{n_{1} \times n_{2}}\) 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 \(JN'_d, JN'_d, ..., JN'_d\) for continuing performing join operations.

Assuming that the remained tuple sets at \(NR'_d, \{NR'_d, S'_d\}, NR'_d, S'_d, \{NR'_d, S'_d\}\) are \(LR''_{d}, LR''_{d}, LR''_{d}, LR''_{d}\) and \(LS''_{d}, LS''_{d}, LS''_{d}, LS''_{d}\) respectively. We have to re-construct a weighted complete bipartite graph \(G=(A,E,\emptyset)\), where

\[
\begin{align*}
\text{(1)} & \ A=\{A_1, A_2, ..., A_{n_{1} \times n_{2}}\} \text{ is a vertex set, where } A_1=(NR_1, S_1), A_2=(NR_2, S_2), ..., A_{n_{1} \times n_{2}}=(NR_{n_{1} \times n_{2}}, S_{n_{1} \times n_{2}}) \\
\text{(2)} & \ B=\{JN_1, JN_2, ..., JN_m\} \text{ is a vertex set, each vertex is an available join node} \\
\text{(3)} & \ E=\cup_{v \in V_b} \{(x, y) | \forall x \in B\}, \text{ and} \\
\text{(4)} & \ \phi(A,JN_i)=t \text{ means the cost of completing the join in } \text{LR}'_{d} \text{ and LS}'_{b} \text{ at } JN_i \text{ is } t.
\end{align*}
\]

If the \(M'\) of \(G\) is same to the \(M'\) of \(G\), the remained tuples in \(LR''_{d}, LR''_{d}, LR''_{d}, LR''_{d}\) and \(LS''_{d}, LS''_{d}, LS''_{d}, LS''_{d}\) continue being transferred to \(JN_1JN_2..., JN_{n_{1} \times n_{2}}\) for performing join operations. Otherwise, the remained tuples are to be transferred to \(JN_1JN_2..., JN_{n_{1} \times n_{2}}\) 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 (ANs) for completing aggregation operations.

6.2 Dealing with Possible Lost Join Results

When the join node for performing join of \(R'_d(S_d)\) and \(S'_d(R_d)\) is changed from \(JN_i\) to \(JN_m\), some join results in \(R'_d(S_d)<\alpha S'_d(R_d)\) may be lost. Assuming \(HR''_{d}(S_d)\) and \(HS''_{d}(R_d)\) are transferred tuples sets to \(JN_i\), \(R'_d(S_d)<\alpha S'_d(R_d)\) represents the tuple sets having \(T\) value equal to \(Val\) in \(R'_d(S_d)<\alpha S'_d(R_d), i.e. for any \(t \in R'_d(S_d)<\alpha S'_d(R_d)\) value equal to \(Val\) which have been transferred to \(JN_i\).

Phase 1. For each value \(Val\) on the join attribute \(T\) in \(R'_d(S_d)<\alpha S'_d(R_d)\), we first record \(\|R'_d(S_d)<\alpha S'_d(R_d)\|\) and \(\|S'_d(R_d)<\alpha S'_d(R_d)\|\), as well as \(\|HR''_{d}(S_d)<\alpha S'_d(R_d)\|\) and \(\|HS''_{d}(R_d)<\alpha S'_d(R_d)\|\), where \(\|HR''_{d}(S_d)<\alpha S'_d(R_d)\|\) and \(\|HS''_{d}(R_d)<\alpha S'_d(R_d)\|\) are tuple sets having \(T\) value equal to \(Val\) which have been transferred to \(JN_i\).

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 \(JN_i\) i.e. \(GenTN_i=\|HR''_{d}(S_d)<\alpha S'_d(R_d)\|\times\|HS''_{d}(R_d)<\alpha S'_d(R_d)\|\), and compute the number of tuples having \(T\) value equal to \(Val\) in join results which will be generated at \(JN_i\), i.e. \(GenTN_i=\|R'_d(S_d)<\alpha S'_d(R_d)\|\times\|\)
Phase 3. Compare \(\text{GenTN}_i\) and \(\text{GenTN}_j\).

(3.1) If \(\text{GenTN}_i\geq\text{GenTN}_j\), the tuples in \(LR''(S_i)_{\text{val}}\) continue being transferred to \(JN_k\) to perform join operations. When no tuples having \(T\) value equal to \(\text{Val}\) in \(LR''(S_i)_{\text{val}}\) at \(NR_a\) exist, i.e. \(LR''(S_i)_{\text{val}}\setminus\emptyset\), we begin to transfer tuples in \(LR''(S_i)\) from \(NR_a\) to \(JN_k\) for performing join operations.

(3.2) If \(\text{GenTN}_i<\text{GenTN}_j\), the tuples having \(T\) value equal to \(\text{Val}\) which have been transferred to \(JN_k\) must be re-transferred to \(JN_k\) to re-generate join results and delete generated join results \((HR''(S_i) \gg HR''(S_i)_{\text{val}})\) at join node \(JN_k\).

Phase 4. Similarly, the tuples in \(LS''(S_k)_{\text{val}}\) are processed.

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(\(GB, A_1, A_2, A_3, A_4\), T). Five partial replicas of \(S, S_1, S_2, S_3, S_4\) 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>CPU</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_0)</td>
<td>User node, a user issues a query and gets the results at this node</td>
<td>1.8GHz</td>
<td>256M</td>
</tr>
<tr>
<td>(N_r\sim N_N)</td>
<td>Nodes where partial replicas of (R) locate</td>
<td>2.4GHz</td>
<td>512M</td>
</tr>
<tr>
<td>(N_r\sim N_S)</td>
<td>Nodes where partial replicas of (S) locate</td>
<td>2.4GHz</td>
<td>512M</td>
</tr>
<tr>
<td>(N_r\sim N_A)</td>
<td>Available join nodes and aggregation nodes</td>
<td>3.2GHz</td>
<td>2GB</td>
</tr>
</tbody>
</table>

7.2 The Impact of Replica Maximum Cover

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 \(JN_k\) and Fig.4(d) shows the possible lost join results. The tuples (3,2) in \(LR''(S_i)\) and (2,6) in \(LS''(S_k)\) continue being transferred to execution node \(JN_k\) to perform join operations. And, the tuples (4,3),(5,3) and (5,5) in \(LR''(S_i)\), as well as the tuples (3,3),(3,3),(4,6) and (4,7) in \(LS''(S_k)\) are transferred to new selected execution node \(JN_k\) 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 \(R\gg S\) on \(R.GB\).
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 \textit{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 transferred to user node from less execution nodes in G-I than that in G-II. This shows our algorithm \textit{RMCA} has good performance on query response time.

7.3 The Impact of Obtaining Efficient Tuple

The experiment mainly analyzes the effect on the query response time of the algorithm \textit{OET} by two approaches. In approach H-I, the algorithm \textit{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 \textit{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 \textit{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.

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

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.