Volume 2, Issue 9, September 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Query Decomposition and Data Localization Issues in Cloud Computing

**P.Ravinder Rao***
*MCA Dept,*
*Anurag Group of Institutions .*
*India*

**K.Harish Kumar**
*CSE Dept*
*St.Mary's Engg College*
*India*

**P.Raja Sekhar Reddy**
*CSE Dept*
*Anurag Group of Institutions*
*India*

***Abstract— The end of this decade is marked by a paradigm shift of the industrial information technology towards a pay-per-use service business model known as cloud computing. Cloud data storage is efficiently used to store the out-sourced data (data that is not stored/retrieved from the customers own servers). In this work we observed that, from a customer's point of view, the data need to be accessed with minimum time when the data is requested from the cloud servers. This can be achieved only when internal architecture of cloud supports for efficient Query processing mechanisms. This paper proposes the efficient available Query Decomposition and data localization techniques for efficient retrieval of data to satisfy the customer needs.***

***Keywords— Cloud Computing,Cloud Storage,Cloud Service Provider,Query Decomposition,HF,VF***

## I. INTRODUCTION

The industrial information technology towards a subscription based or pay-per-use service business model known as *cloud computing*. This paradigm provides users with a long list of advantages, such as provision computing capabilities; broad, heterogeneous network access; resource pooling and rapid elasticity with measured services .Huge amounts of data being retrieved from geographically distributed data sources, and non-localized data-handling requirements, creates such a change in technological as well as business model. One of the prominent services offered in *cloud computing* is the *cloud data storage*, in which, subscribers do not have to store their own data on their servers, where instead their data will be stored on the cloud service provider's servers. In cloud computing, subscribers have to pay the providers for this storage service. This service does not only provides flexibility and scalability data storage, it also provides customers with the benefit of paying only for the amount of data they needs to store for a particular period of time, without any concerns of efficient storage mechanisms and maintainability issues with large amounts of data storage[3]. In addition to these benefits, customers can easily access their data from any geographical region where the Cloud Service Provider's network or Internet can be accessed [1]. An example of the cloud computing is shown in Fig. 1. Since *cloud service providers* (*SP* ) are separate market entities, data integrity and privacy and retrieval are the most critical issues that need to be addressed in *cloud computing*[4]. Even though the cloud service providers have standard regulations and powerful infrastructure to ensure customer's data privacy, data retrieval and provide a better availability [5], the reports of privacy breach and service outage have been apparent in last few years


Fig. 1. Cloud computing architecture example

In this work we observed that, from a customer's point of view, relying upon data retrieving which he needs by performing an effective query Decomposition In addition of providing reliability, availability, Query Optimization and Decomposition is equally important. Query can be optimized by decomposing the Query for effective retrieval of data.

To address Decomposition issues in this paper, we proposed the techniques for decomposing the queries to provide customers with fast data retrieval [2].

## II. QUERY DECOMPOSITION

1. **Normalization**: Transform query to a normalized form
2. **Analysis**: Detect and reject ”incorrect” queries; possible only for a subset of relational calculus
3. **Elimination of redundancy**: Eliminate redundant predicates
4. **Rewriting**: Transform query to RA and optimize query

**Query decomposition:** Mapping of calculus query (SQL) to algebra operations (select, project, join, rename)

• Both input and output queries refer to global relations, without knowledge of the distribution of data.

• The output query is semantically correct and good in the sense that redundant work is avoided.

**Query Decomposition:**

**Normalization:** Transform the query to a normalized form to facilitate further processing.

Consists mainly of two steps:

1. **Lexical** and **syntactic** analysis
   - Check validity (similar to compilers)
   - Check for attributes and relations
   - Type checking on the qualification
2. Put into **normal form**

With SQL, the query qualification (WHERE clause) is the most difficult part as it might be an arbitrary complex predicate preceded by quantifiers

{

**}**

Conjunctive normal form:

($^{p11}$ v $^{p12}$v · … v p1n) ^…. ^ (pm1v pm2 v . . . v pmn)

Disjunctive normal form:

(p11 ^ p12 ^ · · · ^ p1n) v… v (pm1 ^ pm2 ^ . . . ^ pmn)

In the disjunctive normal form, the query can be processed as independent conjunctive sub queries linked by unions (corresponding to the disjunction)

**Example:** Consider the following query: *Find the names of employees who have been working on project P1 for 12 or 24 months?*

• The query in SQL:

SELECT ENAME FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO AND ASG.PNO = ‘‘P1’’ AND DUR = 12 OR DUR = 24

• The qualification in conjunctive normal form:

EMP.ENO = ASG.ENO ^ ASG.PNO = ”P1” ^ (DUR = 12 v DUR = 24)

• The qualification in disjunctive normal form:

(EMP.ENO = ASG.ENO ^ ASG.PNO = ”P1” ^ DUR = 12) v
(EMP.ENO = ASG.ENO ^ ASG.PNO = ”P1” ^ DUR = 24)

### *A. Query Decomposition – Analysis*

• **Analysis:** Identify and reject type incorrect or semantically incorrect queries

• Type incorrect:

Checks whether the attributes and relation names of a query are defined in the global schema

Checks whether the operations on attributes do not conflict with the types of the attributes, e.g., a comparison > operation with an attribute of type string

• Semantically incorrect:

Checks whether the components contribute in any way to the generation of the result

Only a subset of relational calculus queries can be tested for correctness, i.e., those that do not contain disjunction and negation. Typical data structures used to detect the semantically incorrect queries are:

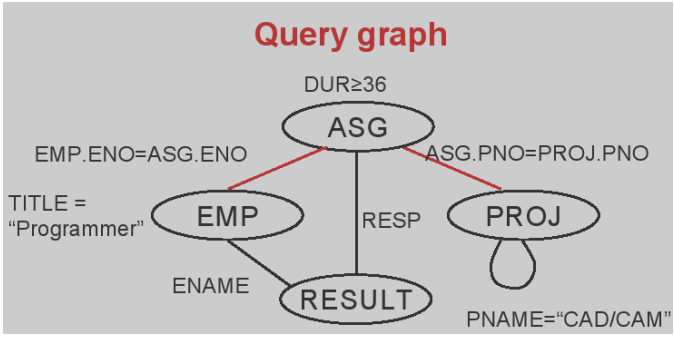
Connection graph (query graph)
Join graph

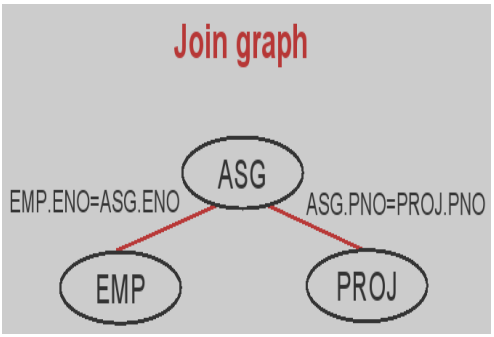
• **Example:** Consider a query:

SELECT ENAME,RESP FROM EMP, ASG, PROJ WHERE EMP.ENO = ASG.ENO AND ASG.PNO = PROJ.PNO AND PNAME = "CAD/CAM"
AND DUR ≥ 36 AND TITLE = "Programmer"

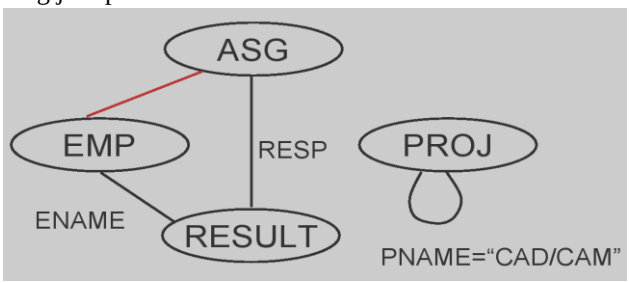
 

• Query/connection graph
– Nodes represent operand or result relation
– Edge represents a join if both connected
nodes represent an operand relation, otherwise it is a projection
• Join graph
– a subgraph of the query graph that considers
only the joins
• Since the query graph **is connected**, the query is semantically correct




• **Example:**
Consider the following query and its query graph:
SELECT ENAME,RESP FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO AND PNAME = "CAD/CAM" AND DUR ≥ 36 AND TITLE = "Programmer"
• Since the graph **is not connected**, the query is semantically incorrect.
• 3 possible solutions:
– Reject the query
– Assume an implicit Cartesian Product between ASG and PROJ
– Infer from the schema the missing join predicate ASG.PNO = PROJ.PNO




***B. Elimination of Redundancy***
• **Elimination of redundancy:** Simplify the query by eliminate redundancies, e.g.,
redundant predicates
– Redundancies are often due to semantic integrity constraints expressed in the query
language
– e.g., queries on views are expanded into queries on relations that satisfy certainintegrity and security constraints
• **Transformation rules are used, e.g.,**
– p ^ p ↔ p
– p _ p ↔ p
– p ^ true ↔ p
– p _ false ↔p
– p ^ false ↔ false
– p _ true ↔true
– p ^ ¬p ↔ false
– p _ ¬p ↔ true
– p1 ^ (p1 _ p2) ↔p1
– p1 _ (p1 ^ p2) ↔ p1
• **Example:**
Consider the following query:
SELECT TITLE FROM EMP WHERE EMP.ENAME = "J. Doe" OR (NOT(EMP.TITLE = "Programmer") AND ( EMP.TITLE = "Elect. Eng." OR EMP.TITLE = "Programmer" ) AND NOT(EMP.TITLE = "Elect. Eng."))
• Let p1 be ENAME = ”J. Doe”, p2 be TITLE = ”Programmer” and p3 be TITLE = ”Elect. Eng.”
• Then the qualification can be written as p1 v (¬ p2 ^ (p2 v p3) ^ ¬ p3) and then be transformed into p1
• Simplified query:

SELECT TITLE FROM EMP WHERE EMP.ENAME = "J. Doe"

***C .Rewriting***

• **Rewriting:** Convert relational calculus query to relational algebra query and find an **efficient** expression.

• **Example:**

Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.[2]

• SELECT ENAME FROM EMP, ASG, PROJ

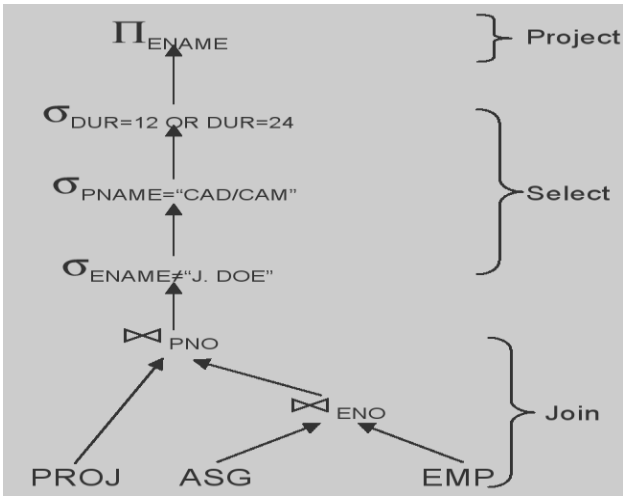
WHERE EMP.ENO = ASG.ENO AND ASG.PNO = PROJ.PNO AND ENAME 6= "J. Doe" AND PNAME = "CAD/CAM" AND (DUR = 12 OR DUR = 24)

• A **query tree** represents the RA-expression

– Relations are leaves (FROM clause)

– Result attributes are root (SELECT clause)

– Intermediate leaves should give a result from the leaves to the root




By applying transformation rules, many different trees/expressions may be found that are equivalent to the original tree/expression, but might be more efficient.

• In the following we assume relations R(A1, . . . ,An), S(B1, . . . ,Bn), and T which is union-compatible to R.

• **Commutativity** of binary operations

– R × S = S × R

– R ⋈ S = S ⋈ R

– R U S = S U R

• **Associativity** of binary operations

– (R × S) × T = R × (S × T)

– (R ⋈S) ⋈ T = R ⋈ (S ⋈T)

• **Idempotence** of unary operations

– $\Pi_A(\Pi_A(R)) = \Pi_A(R)$

$\sigma p1(A1)(\sigma p2(A2)(R)) = \sigma p1(A1) \wedge p2(A2)(R)$

• **Commuting selection with binary operations**

– $\sigma_{p(A)}(R \times S) \leftrightarrow \sigma_{p(A)}(R) \times S$

– $\sigma_{p(A1)}(R \bowtie_{p(A2,B2)} S) \leftrightarrow \sigma_{p(A1)}(R) \bowtie_{p(A2,B2)} S$

– $\sigma p(A)(R \cup T) \leftrightarrow \sigma_{p(A)}(R) \cup \sigma_{p(A)}(T)$

_ (A belongs to R and T)

• **Commuting projection with binary operations (assume C = A′ U B′)**

$\Pi_C(R \times S) \leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$

$\Pi_C(R \bowtie_{p(A',B')} S) \leftrightarrow \Pi_{A'}(R) \bowtie_{p(A',B')} \Pi_{B'}(S)$

$\Pi_C(R \cup S) \leftrightarrow \Pi_C(R) \cup \Pi_C(S)$

**Example:**

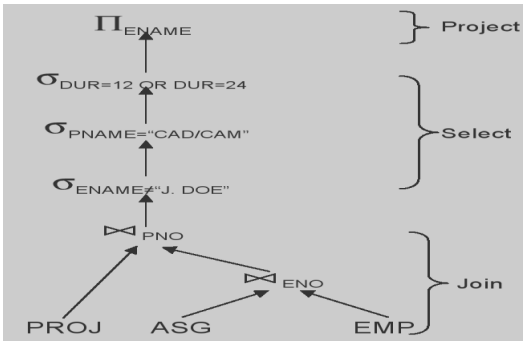
Two equivalent query trees for the previous example Recall the schemas:

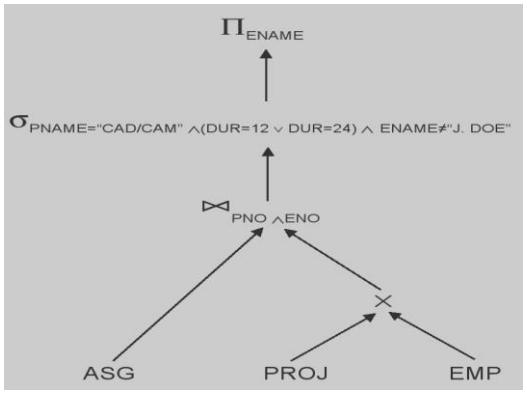
EMP(ENO, ENAME, TITLE)

PROJ(PNO, PNAME, BUDGET)

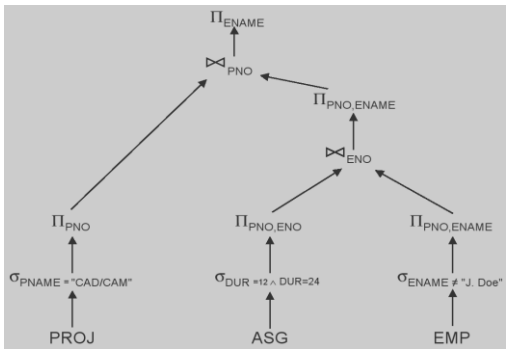
ASG(ENO, PNO, RESP, DUR)




Another equivalent query tree, which allows a more efficient query evaluation, since the most selective operations are applied first.




**III. Data localization**

**Input:** Algebraic query on global conceptual schema

**Purpose:**Apply data distribution information to the algebra operations and determine which fragments are involved Substitute global query with queries on fragments :Optimize the global query

**Example:**

Assume EMP is horizontally fragmented into EMP1, EMP2, EMP3 as follows:[8]

EMP1 = $\sigma_{ENO}$<=”E3”(EMP)

EMP2 = $\sigma_{\text{”E3”}}$<=ENO<=”E6”(EMP)

EMP3 =$\sigma_{ENO}$>”E6”(EMP)

ASG fragmented into ASG1 and ASG2 as follows:

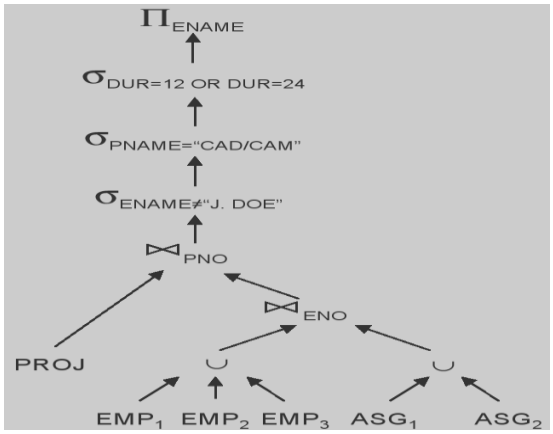
ASG1 = $\sigma_{ENO}$≤”E3”(ASG)

ASG2 = $\sigma_{ENO}$>”E3”(ASG)

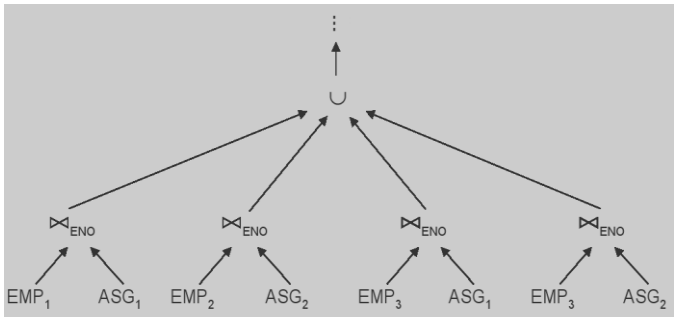
• Simple approach: Replace in all queries
EMP by (EMP1U EMP2 U EMP3) ASG by (ASG1 U ASG2) Result is also called generic query

• In general, the generic query is inefficient since important restructurings and simplifications can be done.



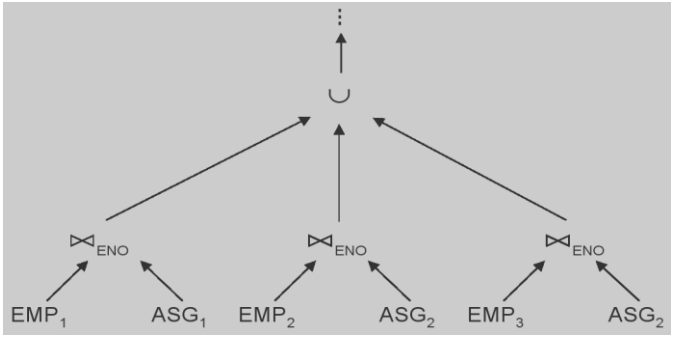

**Example (contd.)**: Parallelism in the evaluation is often possible
– Depending on the horizontal fragmentation, the fragments can be joined in parallel followed by the union of the intermediate results




**Example (contd.)**: Unnecessary work can be eliminated
– e.g., EMP3 ⋈ ASG1 gives an empty result

_ EMP3 = $\sigma_{ENO>"E6"}$(EMP)

_ ASG1 = $\sigma_{ENO\leq"E3"}$(ASG)




### *A. Data Localizations Issues*

• Various more advanced reduction techniques are possible to generate simpler and optimized queries.
• Reduction of horizontal fragmentation (HF)
  Reduction with selection
  Reduction with join
• Reduction of vertical fragmentation (VF)
  Find empty relations

***i) Reduction of HF:***

• **Reduction with selection for HF**

Consider relation R with horizontal fragmentation F = {R1,R2, . . . ,Rk}[2] , where

Ri = $\sigma_{pi}$(R)

Selections on fragments, $\sigma_{pj}$ (Ri), that have a qualification contradicting the qualification of the fragmentation generate empty relations, i.e.,

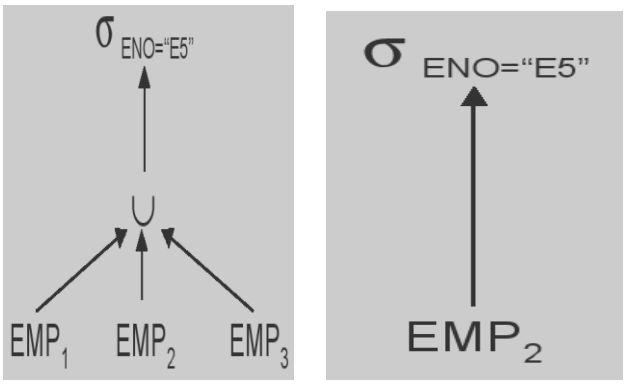
$\sigma_{pj}$ (Ri) = ø; for all x belongs to↔ R(pi(x) ^ pj(x) = false

Can be applied if fragmentation predicate is inconsistent with the query selection predicate.

• **Example:**

Consider the query: SELECT * FROM EMP WHERE ENO="E5"

After commuting the selection with the union operation, it is easy to detect that the selection predicate contradicts the predicates of EMP1 and EMP3.




**Example:**

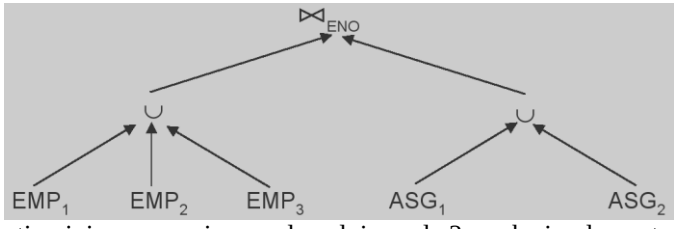
Consider the following query and fragmentation:

SELECT * FROM EMP, ASG WHERE EMP.ENO=ASG.ENO

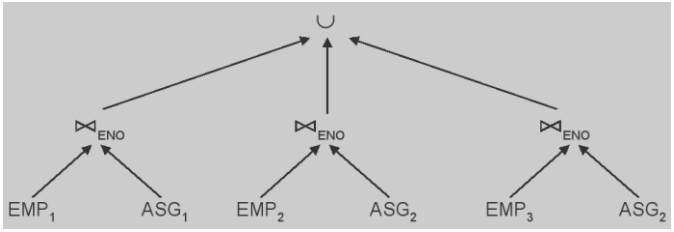
Horizontal fragmentation:

- EMP1 = $\sigma_{ENO\leq"E3"}$(EMP)
- EMP2 = $\sigma_{"E3"<ENO\leq"E6"}$(EMP)
- EMP3 = $\sigma_{ENO>"E6"}$(EMP)
- ASG1 = $\sigma_{ENO\leq"E3"}$(ASG)
- ASG2 = $\sigma_{ENO>"E3"}$(ASG)

**Generic query**




The query reduced by distributing joins over unions and applying rule 2 can be implemented as a union of three partial joins that can be done in parallel.




**Reduction with join for derived HF**

The horizontal fragmentation of one relation is **derived** from the horizontal fragmentation of another relation by using semijoins.

• If the fragmentation is not on the same predicate as the join (as in the previous example), derived horizontal fragmentation can be applied in order to make efficient join processing possible.

• **Example:**

Assume the following query and fragmentation of the EMP relation: **SELECT** * **FROM** EMP, ASG **WHERE** EMP.ENO=ASG.ENO[8]

Fragmentation (**not** on the join attribute):

EMP1 = $\sigma_{TITLE}$ ="Prgrammer"(EMP)

EMP2 = $\sigma_{TITLE}$ ≠"Prgrammer"(EMP)

To achieve efficient joins ASG can be fragmented as follows:

ASG1= ASG⋈ENOEMP1

ASG2= ASG⋈ENOEMP2

The fragmentation of ASG is derived from the fragmentation of EMP Queries on derived fragments can be reduced, e.g., ASG1 ⋈ EMP2 = ø.

***ii) Reduction for VF:***

**Reduction for Vertical Fragmentation**

– Recall, VF distributes a relation based on projection, and the reconstruction operator is the join.

– Similar to HF, it is possible to identify useless intermediate relations, i.e., fragments that do not contribute to the result.

– Assume a relation R(A) with A = {A1, . . . ,An}, which is vertically fragmented as

Ri = $\Pi_{Ai'}$(R) where Ai' is subset or equal to A

$\Pi_{D,K}$(Ri) is useless if the set of projection attributes D is not in $_{A I '}$ and K is the key attribute. Note that the result is not empty, but it is useless, as it contains only the key attribute

**Example:** Consider the following query and vertical fragmentation:

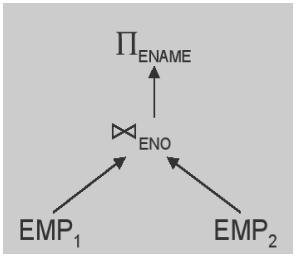
Query: **SELECT** ENAME **FROM** EMP

Fragmentation:

EMP1 = $\Pi_{ENO,ENAME}$(EMP)

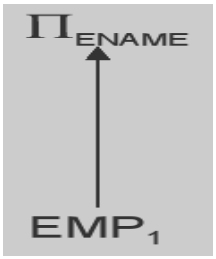
EMP2 = $\Pi_{ENO,TITLE}$(EMP)

• Generic query




• Reduced query

– By commuting the projection with the join (i.e., projecting on ENO, ENAME), we can see that the projection on EMP2 is useless because ENAME is not in EMP2




## IV. CONCLUSION

Query decomposition and data localization maps calculus query into algebra operations and applies data distribution information to the algebra operations. Query decomposition consists of normalization, analysis, elimination of redundancy, and rewriting. Data localization reduces horizontal fragmentation with join and selection, and vertical fragmentation with joins, and aims to find empty relations. In this paper, we proposed a various issues related to the Query Decomposition in cloud computing, which seeks to provide each customer with better data retrieval from cloud data storage.

## ACKNOWLEDGEMENT

We are very much thankful to Prof. G. Vishnu Murthy, Head of the CSE Dept who had given valuable suggestions in carrying out this paper.

## REFERENCES

[1] Amazon.com, "Amazon s3 availablity event: July 20, 2008", Online at http://status.aws.amazon.com/s3-20080720.html, 2008

[2] M. Tamer Oezsu, Patrick Valduriez ``Principles of Distributed Database Systems, Second Edition" Prentice Hall, ISBN 0-13-659707-6, 1999

[3] R. Gellman, "Privacy in the clouds: Risks to privacy and confidentiality from cloud computing", Prepared for *the World Privacy Forum*, online at http://www.worldprivacyforum.org/pdf/WPF Cloud Privacy Report.pdf,Feb 2009.
[4] W. Itani, A. Kayssi, A. Chehab, "Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures," *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, Dec 2009
[5] M. Jensen, J. Schwenk, N. Gruschka, L.L. Iacono, "On Technical Security Issues in Cloud Computing", *IEEE International Conference on Cloud Computing, (CLOUD II 2009)*, Banglore, India, September 2009, 109-116
[6] P. F. Oliveira, L. Lima, T. T. V. Vinhoza, J. Barros, M. M´edard,"Trusted storage over untrusted networks", *IEEE GLOBECOM 2010*, Miami, FL. USA.
[7] S. H. Shin, K. Kobara, "Towards secure cloud storage", *Demo forCloudCom2010*, Dec 2010.
[8] Stefano ceri Giuseppe pelagati "Distributed Databases-Principles and systems"Tata MC Graw Hill 2008
[9] J. Du, W. Wei, X. Gu, T. Yu, "RunTest: assuring integrity of dataflow processing in cloud computing infrastructures", In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS'10),ACM,NewYork,NY,USA,293-304