CB-PATTERN TREES : IDENTIFYING DISTRIBUTED NODES WHERE MATERIALIZATION IS AVAILABLE

K.Dhanasree[1] and C. Shoba Bindu[2]

[1]Associate professor, DRKIST, Hyderabad, Telangana, India.
[2]Associate professor, JNTUA, Anantapuramu, Andhra Pradesh, India.

ABSTRACT

The role of materialized views is becoming vital in today's distributed Data warehouses. Materialization is where parts of the data cube are pre-computed. Some of the real time distributed architectures are maintaining materialization transparencies in the sense the users are not known with the materialization at a node. Usually what all followed by them is a cache maintenance mechanism where the query results are cached. When a query requesting materialization arrives at a distributed node it checks in its cache and if the materialization is available answers the query. What if materialization is not available- the node communicates the query in the network until a node answering the requested materialization is available. This type of network communication increases the number of query forwarding's between nodes. The aim of this paper is to reduce the multiple redirects. In this paper we propose a new CB-pattern tree indexing to identify the exact distributed node where the needed materialization is available.

KEYWORDS

Distributed Data warehouse, Materialized views, vertical fragments, pattern trees, multiple redirects.

1. INTRODUCTION

Addressing high availability and huge data storage the distributed data warehouses have paved in [1]. The distributed architecture is where the primary data is fragmented and distributed across various sites. While fragmenting the architecture may follow distribution of horizontal fragments or vertical fragments [2,3]. Many of the distributed architectures follow location transparency [4] where the fragments located at various sites are not known to the user. For fast query processing many distributed architectures follow materialization [5] where parts of the views located at the distributed nodes are pre-calculated and presented. Many distributed architectures follow materialization transparencies where the materialization available at various nodes is not known to the user. With these transparencies what all followed by some of the architectures in addressing the user queries is communicate the queries in the network until a node that can answer the materialization is found. This type of network forwarding between the nodes increases the number of query redirects. With increased query redirects the communication cost also increases. This is the main drawback of many of the distributed scenarios following materialization.

To address this issue this paper presents a novel indexing mechanism to identify the distributed node that can answer the query. Once the node is identified the query requesting the materialization can be directly forwarded to the node, thereby reducing multiple query redirects. With reduced redirects the distributed communication obviously reduces.

We considered the distributed architecture where vertical fragments are distributed across various nodes. Though maintaining vertical fragments are difficult with huge lots of joins to be addressed we are inspired by the structural easiness of the vertical fragments.

We used a proxy: a stand by between the client and the distributed warehouses to identify the node where requested materialization is available. Our approach used two new concepts the control bounds (CB) and CB-pattern trees for identifying the needed DWs.

The main contributions of our work are:

- A novel pattern trees QPT and DPT.
- A pattern matching algorithm.

The rest of the paper is organized as follows: Section 2 presents background and preliminaries. Section 3 presents proposed method. The results are discussed in Section 4 and finally Section 5 concludes the paper.

2. BACKGROUND AND PRELIMINARIES

The central theme of any distributed architecture is to fragment the whole data and distribute across multiple nodes that are interconnected using a network. Addressing fragmentation architectures what is followed is either the distribution of horizontal or vertical fragments across various sites. The choice of what fragments to be distributed is an issue of the enterprise or a point of interest of the distributed node. Literature has shown wide varieties of distributed olap query processing methods. The main aspect of any distributed querying mechanism is to reduce the query processing cost.

Horizontal fragmentation divides the whole data cube across dimensional tuples and the fragments are distributed across nodes. Works discussed in [6] presented a model for distributing and querying horizontal fragments across various grids. Each fragment is given a unique identifier to locate the node where the fragment is available. Their approach used X-tree indexing to identify the materialized fragments across various nodes.

Vertical fragmentation is where data fragmentation is done along the dimensions and distributed across nodes. Distributing cubes vertical fragments discussed in [7]. The approach followed the multidimensional fragmentation of the fact table and the smaller sub fact tables are distributed across nodes. An encoded index is used to identify the materialized vertical fragments in the distributed scenario.

For fast query processing both the centralized and distributed DWs use materializations where parts of the cube are presented with pre-calculated values. Materialized view selection for multidimensional datasets is discussed in [8]. The approach used a fast heuristic algorithm PBS, (Pick by size) to select desired views for materialization, where PBS selects aggregating views in increasing order of their size. A cost model for materialized web views is discussed in [9].

Works presented in [10] explored the basic properties of the view selection problem under view configurations and workload over conjunctive queries. The works showed that the complexity of the view selection depends on how the query optimizer estimates the size of the view that is considered for materializing.

Works discussed in [11] presented a cache management mechanism to identify the materialized views across distributed nodes. The query is communicated in the network connecting various nodes until a node where materialization is available is found. The caches of the nodes are checked for materialized query, if found the query is answered and if not found the query is redirected in the network. This may increase number of query redirects.

Gnutella [12] a peer-to-peer network uses no indexing mechanism on materialized views. Gnutella receives the query and floods the query into the network. As such every peer receives the query and process the query. The main drawback of Gnutella network is a peer after processing the query identifies the materialization is not available with it and again broadcasts the query. This adds to unnecessary processing cost and increased query forwarding's (redirects).

2.1. Problem statement

The literature discussed above showed the location of the fragments and how the query is communicated in the distributed network. The approaches failed to control the increased number of query redirects in the distributed network and obliviously raised the communication cost. Also a peer after processing the query can only identify that the required materialization is not available with it and redirects. This is overhead of unnecessary query processing cost. This problem is raised because though the approaches followed indexing of nodes they didn't follow the indexing: whether materialization is available at a node.

To address the above issues we proposed a proxy management scheme that indexes the extent of materialization available at a distributed node. Using this indexing technique the query can be directed to the node which can answer the materialization requested. We observed that this type of indexing reduced multiple query redirects and communication cost.

2.2. Preliminaries

Here in this section we discuss some basic preliminaries needed to move our proposed work: Consider a primary data warehouse with d_1, d_2,… d_N base dimensions. For an easy move let d_1=A, d_2=B, d_3=C, d_4=D, d_5=E and with dimensional hierarchies $<a_1,a_2,a_3,a_4>$, $<b_1,b_2,b_3,b_4>$, $<c_1,c_2,c_3,c_4>$ etc such that the partial ordering between these hierarchies $a_1<a_2<a_3<a_4$ etc holds and a fact table with measure sales (S).

2.2.1. Block Queries

Processing block queries increases the performance of distributed query processing mechanisms. A block query is the union of two or more single fragment queries. A single fragment query is a request for materialized single view fragment. Here we present few example block queries:

Q_1:
Select b_2, c_2, D, Sum(S)
from S,B, C,D
where S. B _id = B. B _id and S. C _id = C. C _id and b_2='x' and c_2 =r and D=s
group by B, C, D;
UNION
Select b_1, c_4, Sum(S)
from S, B, C
where S. B _id = B.B _id and S.C _id =C.C _id and b_1='x' and c_4=r
group by B,C;

Q_2:

```
Select   B, Sum(S)
from S, B,D
where S. B _id = B. B _id and b3='x' and d4=y
group by A,B,D;
UNION
Select  C, Sum(S)
from  S, B,C
where S. B _id = B.B _id and A=s and c2=y
group by A,C;
```

Q_3:

```
Select   Sum(S)
from S, B, A,C, D,E
group by A,B,C,D,E;
```

Here Q_1 is requesting two fragment views BCD, BC where materialization (Sum(s)) is expected on the sub fragment cells b_2, c_2, D, b_1, c_4 (from where conditions). D shows materialization on whole base dimension D. Q_2 is requesting fragment views ABD, AC where materialization (sum(s)) is expected on the sub fragment cells b_3, d_4, A, c_2. Q_3 is requesting the materialization of the whole fragment ABCDE.

2.2.2. Vertical fragments

Vertical fragments are the distributions of cross sections along base dimensions of the primary DW. If a DW is constructed on base dimensions A, B, C, D then the possible vertical fragments are: ABCD, ABC,ABD, AB etc. These vertical fragments are various views of the whole view ABCD. These views are also called as user requested group-bys. Hence vertical fragments are nothing but the user requested group-bys. Clearly observing the block queries discussed in section 2.2.1 are requesting materialized vertical fragments at various levels of hierarchies. Our proposed work is to address various block queries requesting materializations at various levels.

3. PROPOSED CB-PATTERN TREE INDEXING

Indexing the extent of materialization available at a distributed node enhances the query performance by directly forwarding the query to the node where the query needed materialization is available and hence reduces number of un-necessary query redirects. With reduced redirects the communication cost is also reduced.

This section presents a novel proxy management mechanism to identify the materialization available at a distributed node. To implement the indexing the proxy makes use of two new concepts: the control bounds (CB) and CB-pattern trees.

3.1. CB-Pattern Trees

Here we present few definitions which can escalate our work.

Definition 1: Control Bounds (CB): The amount of granularity materialized at any DW is measured by ordered bounds called as Control bounds. For a base dimension H with hierarchies h_1,h_2 the CBs are ordered bounds given by $h_1<h_2$.

Inclusion of hierarchy label in the bound shows the fragment corresponding to the hierarchy is completely materialized. Inclusion of * in the bound shows the fragment corresponding to the hierarchy is not materialized. The CBs are used to frame the extent of materialization available at any distributed node.

If suppose h_1, h_2, h_3 and h_4 are the dimensional hierarchy's of the dimension H then the control bounds presents the following cases:

- $h_1<h_2<h_3<h_4$ shows the fragment view H is completely materialized.
- $h_1<h_2<*<*$ shows the fragment view H is partially materialized up to level 2 hierarchy h_2 of H. A full materialized fragment view H can be obtained by calculating the aggregation along the hierarchy's h_3, h_4 and this can incur additional aggregation cost.

Definition 2: CB-pattern tree: A CB-pattern tree is constructed with CBs representing the branches. Our approach uses two types of CB pattern trees:

- CB pattern tree for the DW (DPT).
- CB pattern tree for the query (QPT).

The CB- pattern tree includes the following nodes:

- Level (-1): has root node representing the query (Q) or distributed DW.
- Level (0): has branching nodes representing base dimensions.
- Level (2 to n): includes the (n) hierarchies of the base dimensions of the primary site and are labeled with (hierarchy, level).
- Star node: a star node is a node with label (*, level).
- Non star node: a non star node is a node with label (hierarchy, level).

Definition 3: DW pattern tree (DPT): The DPT shows the amount of materialization available at a distributed node.

Every DDW materializes the hierarchies of their interest and constructs a CB-pattern tree which is the DPT of the respective DW. The branching nodes of the DPT are the base dimensions distributed at that DW. Based on the hierarchies materialized each branching node is extended to include hierarchy label or (*). If a hierarchy label is included then the fragment corresponding to the hierarchy is materialized. If (*) is included then the fragment corresponding to the hierarchy is not materialized.

Let with interest site1 is maintaining the vertical fragments <ACD, BD>. Now the base dimensions available at site1 are A, B, C, D. Now suppose site1 is maintaining the control bounds: $a_1<*<a_3$, $b_1<*<*$, $c_1<c_2<c_3$, $*<d_2<d_3$.

The materialization available at site1 is shown in DPT of Figure 1:

Definition 4: Query pattern tree (QPT): A user query may or may not request materialized hierarchies. Based on the hierarchies requested by the query a QPT is constructed. A missing hierarchy of the query is labeled as * in the QPT.

Definition 5: Equivalence of QPT and DPT: QPT and DPT are equivalent if:

- Both have same number of levels in each branch.
- Both have similar labels in each level.
- If one can be derived from other.
- A QPT is equivalent to DPT if all the non star nodes of QPT match with non star nodes of DPT and the star nodes of QPT need not match.

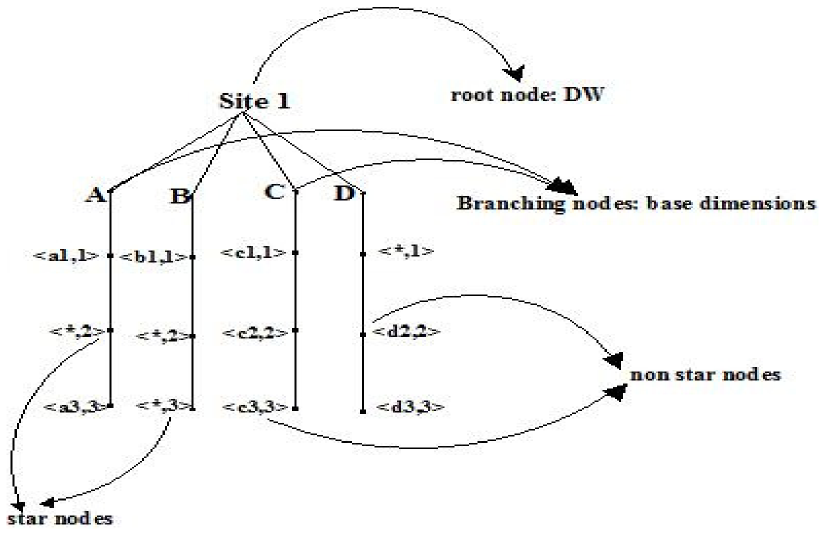

Figure 1: DPT of site1.

Definition 6: Non equivalent pattern trees: Non equivalent pattern trees cannot be completely derived from one pattern tree and possibly be derived from two or more pattern trees.

3.2 CB-tree pattern matching: Mapping star nodes and non star nodes: For QPT and DPT pattern tree matching we use the following rules on star nodes and non star nodes: the star nodes are labeled as <*, level> and the non star nodes are labeled as <hierarchy, level>.

Rule-1: <*, 1> of QPT $\rightarrow$ <h_1, 1> of DPT: The rule shows a mapping of star node of the QPT to a non star node of DPT. Here the query which is not requesting any materialization of the particular leveled fragment is mapped to a materialized fragment at the same level at a DW. This depicts a complete match as the query is not requesting any materialization. Hence the pattern matching infers requested materialization of the hierarchy is available at the DW.

Rule-2: <h_1, 1> of QPT $\longrightarrow$ <*, 1> of DPT: The rule shows a mapping of non star node of the QPT to a star node of DPT. Here the query requesting materialization at level 1 fragment is mapped to a star node of the DPT where materialization is not available. This mapping is not a complete match. Shows query requested materialization of the hierarchy is not available at the DW.

Rule-3: <h_1, 1> of QPT $\longrightarrow$ <h1, 1> of DPT: The rule shows a mapping of non star node of the QPT to a non star node of DPT. Here the query which is requesting materialization at level 1 fragment is mapped to materialized fragment at the same level of the DPT. This depicts a complete match as the query needed materialization is available at the node.

Rule-4: <*, 1> of QPT $\longrightarrow$ <*, 1> of DPT: The rule shows a mapping of star node of the QPT to a star node of DPT. Here the query is not requesting materialization at level 1 fragment and the materialization is not available at DW. This mapping is a complete match.

3.2.1. CB-Pattern Tree Matching Algorithm

Based on the so defined rules a QPT, DPT pattern matching algorithm is presented. Algorithm 1 presents the pattern matching between QPT and DPTs.

Algorithm 1: CB-pattern tree matching algorithm

```
Input: QPT
Output: DW
Step1: for each DPT in proxy file compare QPT branching nodes with DPT branching nodes.
Step 2: If branching nodes match collect the respective DPTs in M, go to step 3
Step 3: for each DPT in M
        Compare QPT with DPT along each branch.
        Identify the DPTs where Rule-1,3, 4 are satisfied for complete match.
        Collect the DWs in S.
        Forward the query to a DW in S that incurs less communication cost.
Step 4: if branching nodes does not completely match then identify the DWs where QPT
        branching nodes  can be derived.
        Apply rules for complete or partial match.
        Split the query and forward to respective DWs where the fragment branching nodes
        match.
Step 5: if branching nodes does not match nor can be derived forward the query to primary DW.
```

3.3. Architecture

Addressing high availability and usability a distributed architecture is followed. We modeled the primary DW to include the base dimensions A,B,C,D,E and with hierarchies <a_1,a_2,a_3,a_4>, <b_1,b_2,b_3,b_4>, <c_1,c_2,c_3,c_4>, <d_1,d_2,d_3,d_4> and <e_1,e_2,e_3,e_4>. Our approach uses distribution of vertical fragments across 3 sites. ABCDE, ABC, ABD, BCD, BD etc are vertical fragment across base dimensions A,B,C,D,E.The primary site includes the whole data, given by the primary fragment ABCDE. Other distributed sites can have the fragments of the interest that are allocated using standard allocation methods.

The fragment views located and materialized (may or not) at respective DWs are shown in Table 1. On materializing the needed hierarchies each DW constructs the DPTs and updates the proxy. The proxy file maintains the DPTs of various distributed DWs. The branching nodes needed for the DPTs are identified from the base dimensions in the fragments at various DWs. The DPT of various DWs are constructed using the control bounds and are shown in figure 2.

Table 1: Views and materialization at various DWs

DW	Fragment views	Base dimensions at DWs	CBs
Site1	BCD,BC	B, C,D	$b_1<b_2<b_3<*$,$*<c_2<c_3<c_4$, $d_1<d_2<d_3<d_4$
Site2	ABD, AC	A,B,C,D	$a_1<a_2<a_3<a_4$, $*<b_2<b_3<b_4$, $*<*<*<*$, $d_1<d_2<d_3<d_4$
Site3	AC	A,C	$a_1<a_2<a_3<a_4$, *<c2<c3<*,
Primary site	ABCDE	All	Un materialized.

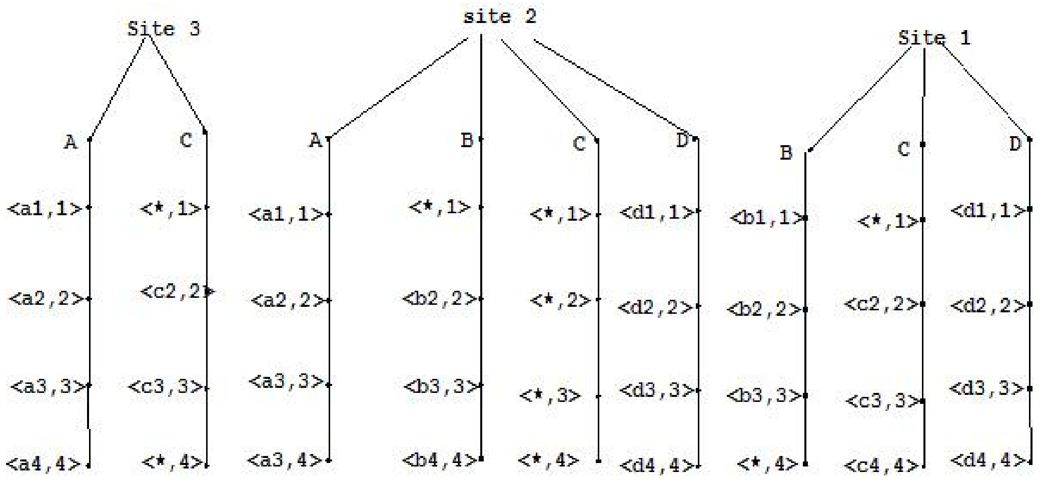

Figure 2: DPTs of all DWs.

3.4. CB-Pattern Tree Indexing

Every distributed site updates the proxy with their DPTs. All the incoming queries are first directed to the proxy. The proxy constructs QPT for the query. These QPTs are then compared with DPTs at proxy. Using CB- pattern tree matching algorithm 1 the DPTs for which the QPTs match are identified. If only one DPT is identified then the query is forwarded to that DW. If more than two DPTs are identified then the query is directed to the DW incurring less communication cost. If none of the DWs are identified then the query is forwarded to the primary site.

3.4.1. Query processing

Here in this section we present the processing of the block queries discussed in section 2.2.1.

On receiving the block queries Q_1, Q_2, Q_3 the proxy first constructs the QPTs. The QPTs for various queries is shown in figure 3. For this the proxy first generated the control bounds on each fragment of block and then combines these control bounds. The query CBs are generated based on the materialization requested by the query at various levels of hierarchies (as requested in the where conditions of the block query).

For Q_1: on fragment BCD:

CBs: B: $*<b_2<*<*$
C: $*<c_2<*<*$
D: $d_1<d_2<d_3<d_4$

on fragment BC:

CBs: B: $b_1<*<*<*$
C: $*<*<*<c_4$

Combining the CBs of the fragments the total CBs of Q_1 are: $b_1<b_2<*<*$; $*<c_2<*<c_4$; $d_1<d_2<d_3<d_4$.

For Q_2: on fragment ABD:
CBs: A: $*<* <*<*$
B: $*<*<b_3<*$
D: $*<*<*<d_4$
on fragment AC:
CBs: A: $a_1<a_2<a_3<a_4$
C: $*<c_2<*<*$

Combining the CBs of the fragments the total CBs of Q_2 are: $a_1<a_2<a_3<a_4$; $*<*<b_3<*$; $*<c_2<*<*$; $*<*<*<d_4$.

For Q_3: on fragment ABCDE (as there are no where conditions)
Total materialization is requested: CBs are: $a_1<a_2<a_3<a_4$; $b_1<b_2<b_3<b_4$; $c_1<c_2<c_3<c_4$; $d_1<d_2<d_3<d_4$.

Now proxy constructs the QPTs as shown in figure 3.

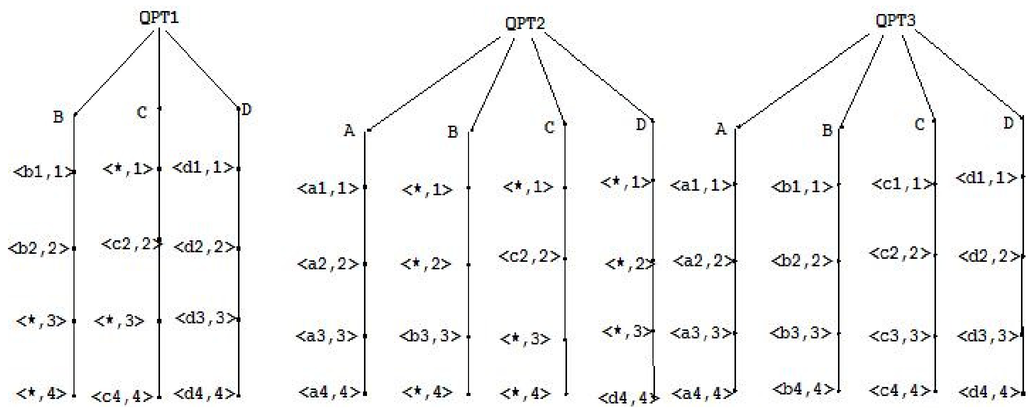

Figure 3: QPTs of block queries Q_1, Q_2, Q_3.

We discuss the processing of queries under three cases:

Case 1: Processing Q_1:

On receiving Q_1 the proxy constructs QPT and compares with each DPT for the match of branching nodes. The branching nodes of QPT of Q_1 matches with branching nodes of DPTs at site 1and site 2. Now the proxy compares along each branching node of QPT and identified DPTs for a match of level nodes. Clearly the QPT of Q_1 is a complete match with DPT of Site 1 using various rules of pattern matching. But QPT is not a complete match with DPT at site 2 as of rule 2. Hence the pattern matching algorithm 1 identifies site 1. Now the proxy forwards Q_1 to site 1.

Case 2: Processing Q_2:

The branching nodes of QPT of Q_2 matched with branching nodes of DPT of site 2.Using CB-pattern matching algorithm the fragment view ABD as requested by the query is a complete match but the materialization on base dimension C as requested by the query is not available at site 2 i.e the materialization on fragment view AC as requested by the query cannot be answered at site 2. Hence a partial match at Site 2. Now the proxy compares unmatched branching nodes AC with other DPTs. The branching nodes AC matched with the branching nodes of DPT at site 3. Using CB-pattern matching algorithm the materialization as requested by the query Q_2 on fragment AC is available at site 3. Now the proxy splits the block query into two fragment query blocks and forwards query block ABD to site 2 and query block AC to site 3.

Case 3: Processing Q_3:

The QPT branching nodes match with branching nodes of DPT of site 2 and didn't match with any other DPTs. But the rules of CB-pattern matching algorithm are not satisfied at site 2, showing materialization as requested by the query Q_3 is not available at site 3 as well as other sites. The proxy forwards the query to primary site.

4. IMPLEMENTATION AND RESULTS

We implemented the CB-pattern tree indexing on two scenarios of architecture:

- Without SPS module.
- Using SPS module.

We implemented two different sets of experiments with varied distributed nodes. All the algorithms are written in C and we used Sql server supporting 500 fact and warehouse tuples. We initially modeled the warehouses to support five dimensional tables and four dimensional hierarchies and then varied the base dimensions up to 40. We are limiting our method only to few dimensions not more than 50.

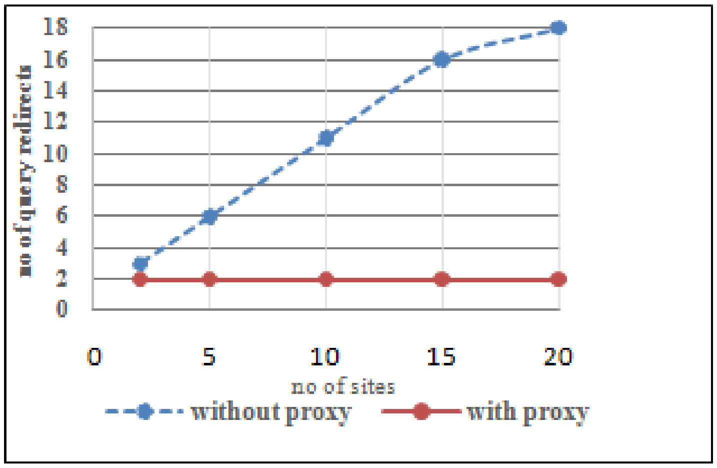

Figure 4: Number of query redirects.

In the first set of experiments we implemented our model without using the SPS proxy. We executed the 3 query blocks Q_1, Q_2, Q_3 . We designed the architecture to study the worst case scenario, where the required DW is the last node of the network. As the client is transparent to where the requested views are located and materialized, the queries are forwarded to the nearest neighbor DWs. We observed that neighbor forwarding increased number of redirects and communication cost.

We implemented the indexing using SPS proxy architecture. We observed that every query has taken only two redirects one to proxy and other to identified DW. Figure 4 shows the number of redirects with and without proxy which is implementing the indexing.

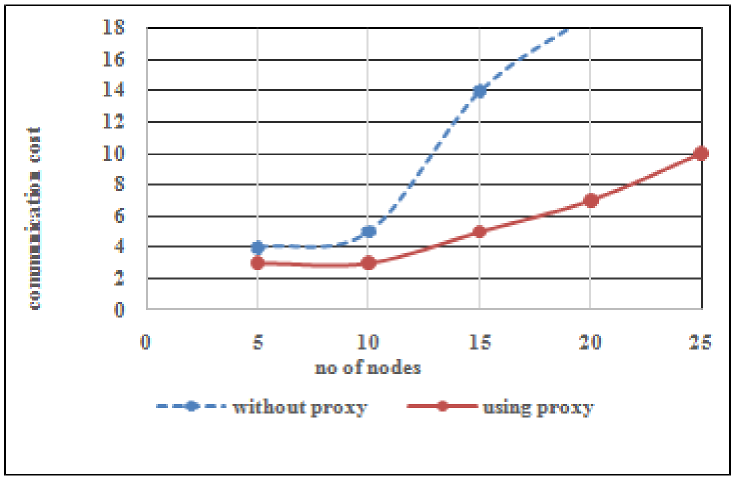

Figure 5: Communication cost

With increased redirects the communication cost increased drastically. Figure 5 shows the increased communication cost without proxy.

5. CONCLUSIONS

Many of the distributed architectures lacked a mechanism to identify where the query requested materialization is available. A common approach followed by many distributed Data warehouses is forward the query in the network until the node that can answer the query is found. This kind of neighbor forwarding increased number of query redirects from node to node and increased communication cost. In this paper we proposed a CB-pattern tree indexing to identify where exactly the query needed materialization is available. For this we used the concept of control bounds and pattern trees. We used a proxy between the client queries and the distributed data warehouses to implement our indexing. We derived two types of pattern trees QPT for the query requesting the needed materialization, DPT for the DW showing the amount of materialization available at the DW. All the distributed DWs materialize the needed fragments and construct the DPTs showing the available materialization with them. They update the DPTs to the proxy. When a query arrives the client forwards the query to the proxy. Proxy constructs the QPT and compares QPT with each DPT. The proxy forwards the query to the DW whose DPT matches with QPT. This type of indexing has incurred only two query redirects one to proxy and other to identified DW, thus reducing communication cost.

REFERENCES

[1] Albrecht T,Lehner W. "Online analytical processing in distributed data warehouses". Proceedings of Database engineering and applications.(1998).p.78-85.

[2] Noaman AY, Barker K. "A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse". Proceedings of the 8th international conference on information and knowledge management.(1999).p.154-161.

[3] Bellatrechr L, Kariapalem K, Mohania M. "What can partitioning do for your data warehouses and data marts".Database engineering and applications Symposium.(2000).p.437-445.

[4] V Raman, I Narang, C Crone, L Haas. "Data access and management services on grid".grid-db,2002.

[5] Vijay kumar TV, Devi K. "Materialized view construction in data warehouses for decision making". International journal of business information systems.Vol.11.issue.4.(2012).p.379-396.

[6] M Golfarelli,D Maio, S Rizzi. "Vertical fragmentation of views in relational data warehouses". SEBD, p.19-33, (1997).

[7] L W F Chaves, E Buchmann, F Hueske, K Bohm. "Towards materialized view selection for distributed databases". Proceedings of the 12th international conference on extending Database Technology.p.1088-1099. (2009).

[8] A Shukla, P Deshpande, J F Naughton." Materialized view selection for multidimensional Datasets".Proceedings of the 24th VLDB Conference. P.488-499.(1998).

[9] A B Ammar, A Abdellatif. " Cost model for the recommendation of materialized webviews". IJDKP. Vol.2.Issue.1. (2012).

[10] R Chirkova, A Y Halevy, D Suciu. "A formal perspective on the view selection problems".The international journal on very large databases.Vol.11, Issue.3.p.216-237.(2002).

[11] R Zhang, YC Hu." Assisted peer-to-peer Search with partial indexing". IEEE transactions on parallel and distributed systems. Vol.18, Issue.8,p.1146- 1158,(2007).

[12] D.Huges, G Coulson, J. Walkerdine. "Free riding on gnutella revisited: the bell tolls".Distributed Systems online. IEEE vol.6, Issue.6.(2005).

AUTHORS

K. Dhanasree is currently an Associate Professor in Department of Computer Science and Engineering at Dasari Rama Kotaiah Institute of Science and Technology, Hyderabad, India. She is pursuing her Ph.D from JNTUA, Anantapuramu, Andhra Pradesh, India. Her research interests include Data Mining, Database security, and Network Security.

Dr. C .Shoba Bindu, received her Ph.D degree in Computer Science and Engineering from JNTUA, Anantapuramu, Andhra Pradesh. She is currently extending her services as Associate Professor in department of Computer Science and Engineering, JNTUA. She has guided many external and internal projects and has good contributions in many reputed journals. Her research interests are in the areas of Mobile and Adhoc Networks, Network security, Data Mining and Cloud Computing. She has delivered many guest lectures' in renowned institutions.