Intelligent Cache Management for Data Grid

Mobin Uddin Ahmed, Raja Asad Zaheer, M.Abdul Qadir
Muhammad Ali Jinnah University
Jinnah Avenue, Islamabad, Pakistan
mobin@jinnah.edu.pk, rajaasad@jinnah.edu.pk, aqadir@jinnah.edu.pk

Abstract

Data grid is a composition of different services for accessing and manipulating data in a distributed environment. One of the major problems is high access time of remote queries in the data grid. In this paper we present a scheme for data cache management system for data intensive applications on Data Grid, which provides a faster access to remote data by intelligently managing the copies in the local cache. In our architecture, we emphasize on data cache management for a multi DB heterogenous database environment using basic Grid services. We give a detailed view for manipulating intelligent cache for structured queries. We also cover, to some extent, intelligent query distribution concept in our homogenous intra-organization data grid with example scenarios.

Keywords: Intelligent Cache Management, Data Grid, Intelligent Query Builder, Query Router, Grid Cache, Distributed Database Management System, Cache for Distributed Databases, Cache for Grid.

1 Introduction

The amount of data used to support the operations of an organization is growing exponentially, so the storing and accessing of this huge data set is becoming a great challenge. Data Grid (Foster, Kesselman and Tuecke 2001 ; Chervenak, Foster, Kesselman, Salisbury, Charles and Tuecke 2001; Foster, Kesselman, Nick and Tuecke 2002) offers an economical solution to handle huge amounts of data by sharing distributed disk space in a virtual database environment. Although Data Grid is an economical way of resource sharing but we may have to compromise performance in the form of access time because of high network latency (Qin, and Jiang 2003) from remote accesses. This issue can be addressed up to some extent by replicating (Allcock, Bester, Bresnahan, Chervenak, Foster, Kesselman, Meder, Nefedova, Quesnel and Tuecke 2001) data on multiple nodes in a Data Grid. However, due to storage cost and less frequent access of information from replicated data, this is not a good solution. In this situation a persistent data cache (Tierney, Johnston and Lee 2003) if managed intelligently may offer a faster access to remote data in an economical way.

Our paper presents, architecture for ***Intelligent Cache Management (ICM).*** The architecture is explained with the help of a population database (intra organization grid) as shown in Figure 1.The data is distributed on regional basis on geographically distributed nodes in a hierarchical way.

Although the scenario we discuss is of hierarchical nature but our architecture works in non-hierarchical systems as well.

In an intra organization data grid for population database replicas can be managed for nearby localities, as there are chances of accessing nearby data more than the remote data. Of course, it would not be feasible to keep replicas of remote localities on each node due to large overheads and less frequent access.

When data is accessed from a remote location, the issue described above i.e. availability of data and network latency can affect the user query. Our proposed system of ***Intelligent Cache Management (ICM)*** preserves data fetched from these remote queries with the assumption that there is a likelihood of accessing the same data again and again. So in case of the same query being issued again, users don't have to fetch data from the remote location, rather our system will fetch records from local cache.

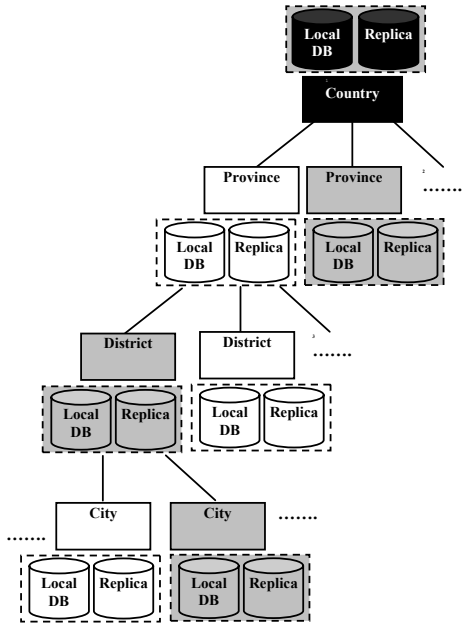

Figure-1: Replication Hierarchy

The issue of maintaining such a cache is not an easy task, as the query may not be exactly the same next time. The query could be fulfilled partially by the local cache and remote data access. There may be a need to combine multiple queries in order to execute them on local cache. In order to gain maximum benefit from this cache, we included the concept of knowledge base (Cannataro and

Tali 2003; Moore 2001; Cortés, Martínezb, Comasb, Sànchez-Marrè a, Pochb and Rodríguez-Roda 2004) that will be discussed in the coming sections. In the context of a national population database we can safely assume following points.

1. Distribution of data is on regional basis; each region has its own local database as well as cache and replicated data. All regions are managing their data by using the same distributed DBMS.
2. A single schema, which is known at every operating region, dealing with structured data.
3. Chances of access frequency for local or nearby data will be higher, but data from remote geographical location will not be accessed as frequently. Therefore, the system may maintain a replica of nearby datasets in order to avoid cache management overheads.
4. Data access is much more frequent as compared to data updation.
5. As Data Grid belongs to intra-organization setup, so a unified policy can be imposed for issues like security, data consistency etc.

If we consider the classifications of integrated database systems, as given in (Aberer 2003), i.e. multi database, federated database and mediator, then to explain the working of the architecture, we are taking into account the multi DB heterogeneous database scenario, where data model is same but databases could be of any vendor though strictly relational in nature. In this paper we are not taking into account the flat files or other forms of data, those will be discussed in another paper.

2 Architecture of Intelligent Data Cache

The distributed **database** cache management technique is significantly different from the traditional caching techniques (Castro, Adya, Liskov and Meyers 1997) for distributed systems. The database cache should be managed intelligently, so it may deal with different scenarios like splitting and merging of user queries into sub-queries for available data sets in cache, in order to reduce the load of remote query. In the case of complex queries, we may find partial results from local cache and the remaining portion of the data set may be fetched from remote locations. After the complete execution of all the sub-queries, user will get the resultant data by union of above multiple data sets.

If one can properly manage the storage of both the parsed query and relevant data in data cache, definitely the incoming queries can be checked for local or remote execution. In the example scenario, we have three levels of data sets for each region, one is local DB, second is the replication of nearby regions i.e. local DB and the third is cache DB, as shown in figure 2. By keeping in view these three levels we may have following options to execute a query.

a. Request data set can be extracted from Local domain i.e. Local DB, Replication of nearby database, and Cache DB with some of the scenarios as follows:

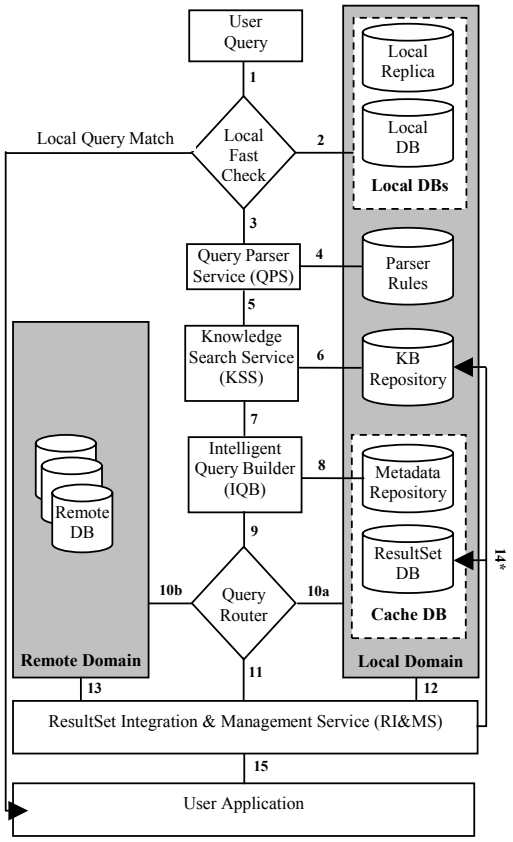

Figure-2: Process Architecture of Intelligent Cache

1 Complete user query is fulfilled from local database.
2 The complete resultant dataset against user query is extracted from local replica of nearby databases.
3 Data set against user query is available partially from Local, and Replicated database.

b. Data set against user query is not fully available with local domain, and it require remote query. So the user query must be parsed and executed partially from local domain and partially from remote location.

The purpose of the system for Intelligent Data Cache management is to manage the remote data set locally; therefore, the remote queries can be entertained partially or fully from local cache ultimately reducing the overheads for a complete remote query.

The architecture of the intelligent cache management system is shown in Figure-2. In our approach, firstly the user query is executed on ***Local Domain*** of Figure-2. There is a chance that complete data set of requested query is available and the result is immediately returned to the user. Otherwise, to comprehend the user query, parsing will be performed for constructing the local and remote query according to keywords available in the *Knowledge Base Repository (KBR)* of database cache. After the execution of both the queries the resultant data set is combined as directed by the splitting logic, and the result is presented to the user. The dataset and keywords

are stored in cache DB and repository against the remote query for future use.

Our proposed architecture has many components bounded together to make Intelligent Data Cache System. The above-mentioned procedure can be implemented by using these bounded components; each performing different functions as explained in coming sections.

In maintaining a cache there may be issue, like data coherency (data consistency) and cache replacement. Data coherency (Agarwal, Simoni, Hennessy and Horowitz 1988) and cache replacement (Otoo, Olken and Shoshani 2002), both can be taken care of by using proposed technique as in (Agarwal, Simoni, Hennessy and Horowitz 1988; Otoo, Olken and Shoshani 2002; Park, Lee, Lim and Yu 2001). We are considering the suitability of new technique for this architecture in another research activity.

2.1 Local Fast Check

This component is responsible for checking whether the query is for the local DBs i.e. local database and replica. The user query is handed over to the local distributed DBMS in operation. If the complete user query is fulfilled by the local DBs then no further processing will take place and result would be handed over immediately to the user application. Non-availability of resultant dataset in local DBs will cause the whole user query to be handed over to the Query Parse Service (QPS). But if the local DBs contain partial results, then we can use them in order to avoid future query processing overheads. These partial dataset will be forwarded to RI&MS, meanwhile the whole user query will be forwarded to Query Parse Service (QPS). In such case it is the responsibility of *Intelligent Query Builder* (IQB) to trim off the local DBs portion of the user query. The trimmed portion of the user query will be sent to RI&MS to properly tackle the integration of datasets

2.2 Query Parse Service (QPS)

This service is responsible for parsing the user query. The parser may consult the *Parser Rules Repository* maintained in the system. The parsing includes, breaking user query into collection of database names, columns, attributes and conditions. Then the query parse service makes the sets of columns, attributes and conditions contained in a particular database for subsequent processing by knowledge search service.

2.3 Knowledge Search Service (KSS)

The task of this service is to get the keywords from the Query Parse Service and make sets of these keywords based on the Schema or databases. After making the sets, this service has to perform a search on the *Knowledge Base Repository (KBR)*, which contains the information about the previous queries and the related data contained in the Cache DB. History of executed queries along with mapping in the Cache DB is stored in the KBR. By matching the current sets, we are able to extract the information out of the KBR such as, which data regarding a particular set is present in our cache. The information regarding data and mapping of the original cache location is handed over to the *Intelligent Query Builder* for further processing. This service uses the Knowledge-Based Repository that has the structure as described in Figure-3. User query can be satisfied fully or partially, and there may be a chance that no match is found in the repository. In either case, some flags and parameters must be set to make other components of the system aware of those conditions, so they may act accordingly.

Table/ Schema Name	Col Name	Search Condition	Insertion ID	Time Stamp	ResultSet DB

Figure-3: Structure of KSR

2.4 Intelligent Query Builder (IQB)

This component is capable of modifying, splitting and merging the query. IQB receives the information from *KSS* that contains results regarding the formed query sets. These results contain information regarding the presence of data in the Cache DB for all sets. The job of this component is to make queries based on the information provided by *KSS*. If distributed DBMS in operation is not exposing metadata for remote locations to our cache management system, then there is a requirement to fetch metadata from Metadata Repository according to the queries for remote domains. Two types of queries are formed by *IQB*. First type of queries formed is the one that would be carried out on the *Cache DB*. Second types of the queries are, those that are performed on the remote locations.

The Intelligent Query Builder is able to intelligently build queries on the basis of user queries. Suppose a user wants the records of those persons, having salary less than 5000/month. For the first time, records are fetched from remote location and stored into ResultSet DB. After that, the user wants to execute a query and find out how many employees are there having salary less than 8000/month.

Intelligent Query Builder (IQB) will construct a new query on the basis of the information provided by the *KSS.* As the Knowledge-Based Repository (KBR), contains dataset of those records having salary less than 5000/month. So, the new composed query by the IQB must fetch dataset of those employees having salaries between 5000 and 8000 per month. This will definitely reduce the query processing time as the data for less than 5000/month will be available locally and only the data for more than 5000/month and less than 8000/month is accessed from remote nodes. The Query Router is handed over all the queries and related metadata, which are composed by this component.

2.5 Query Router

After getting the query along with its destination, this component routes the query to its appropriate destination i.e. local or remote domain. If distributed DBMS in operation provides the facility for making customized query plans as per the directions provided by the *Intelligent Query Builder* (IQB), then the system will get the benefits from these existing capabilities of DBMS. In the previous case the responsibilities of *Query Router* is

performed by the distributed DBMS in operation. Those active query plans are provided to the ResultSet Integration and Management Service (RI&MS) in order to tackle the resultant datasets from both local and remote domains.

2.6 ResultSet Management & Integration Services (RI&MS)

This component is responsible for tackling with the resultant datasets from local or remote domains, as per the active query plans provided by IQB. In case of timeout or failure of any active query plan for local or remote domains, it is liable to direct the *Query Router* to re-execute that plan or some other plan against that failure. It is an important task in order to get the complete resultant datasets. In case where a domain is not accessible there should be an option to display the partial data for user against the query. Query Parser Service (QPS) is responsible to generate the keywords for the complete set of user query. In the case of remote query, these keywords also need to be stored by the RI&MS. Another task that is performed by this component is to store the resultset from remote domain into the Cache DB (ResultSet DB) against the InsertionID generated by the KBR.

2.7 Cache DB

It has two major sub-components i.e. *Metadata Repository* and *ResultSet DB*. The necessary metadata (Singh, Bharathi, Chervenak, Deelman, Kesselman, Manohar, Patil and Pearlman 2003) (i.e. remote destination address, schema name etc.) against each remote location is managed in Metadata Repository. The ResultSet DB will contain those records that are fetched from remote domains on the basis of user queries. These records are saved against InsertionID of KBR. If we have such powerful distributed DBMS in operation, that is capable of exposing metadata about remote locations and

```
SELECT    A.empno, A.ename, A.job, A.desig, A.sal
          B.dname, B.loc
FROM      CityA_DB.emp A, CityA_DB.dept B
WHERE     A.deptno = B.deptno
AND       B.dname = 'IT Dept'
AND       A.job = 'Manager';
```

Figure-4: Input query for scenario one

their available schemas, then there is no need to maintain the Metadata Cache. That information can be obtained from the DBMS.

3. SCENARIOS

In this section we demonstrate the working of the system proposed, with some example scenarios. All the scenarios are based on certain assumptions, which are mentioned before the description of each example.

3.1 Scenario One (Local Query)

Assumptions: User wants to query records of Managers in IT Department from local database, with the field's i.e. employee No., name, job, designation and salary. This query in Figure-4 is being performed from City-A. The local database is maintained at City-A with the name CityA_DB.

3.1.1 Local Fast Check (Step-1)

At the first step of this scenario, the query in Figure-4 will be given as an input. The same set of query will be handed over to the distributed DBMS operating at the site. It is the responsibility of the same DBMS to parse and execute full set of query and if complete result set is found then return the results to user application. If local DBMS does not satisfy complete query, the same set of user query is forwarded to *Query Parser Service* of *Intelligent Cache Management*. In this particular scenario, the entire query is satisfied locally because only

```
SELECT    empno, ename, job, desig, sal
FROM      CityB_DB.emp
WHERE     job = 'Manager';
```

Figure-5: Input query for scenario two

two tables of local schema are used for desired result set. No further processing is required by the system and the DBMS in operation locally returns results to the user application.

3.2 Scenario Two

Assumptions: User wants to query remote databases for all Managers with the fields i.e. employee No, name, job designation, and salary. This query in Figure-5 is being performed from the City-A, and City-B doesn't come under its hierarchy, so there is no dataset related to CityB_DB at local domain. This is not the first time that any data is required from City-B because our user frequently requires data from City-B. Because of that our

```
Schema Name   = {CityB_DB}
Tables Name   = {emp}
ColumnName    = {empno, ename, job, design, sal}
SearchCondition = {CityB_DB.job = 'Manager'}
JoinCondition  = {Null}
```

Figure-6: Output of QPS in scenario two

Cache DB contains complete resultant dataset against the user query and there is no need to send query towards remote domain.

3.2.1 Local Fast Check (Step-1/2)

As the first step of this scenario, the query in Figure-5 is given as an input. This complete user query is simply passed towards local distributed DBMS. The DBMS in operation parses and executes the query on local domain. After the execution of the query, DBMS fails to get the desired result set. After that, the query is forwarded to another component of *Intelligent Cache Management,* that is the *Query Parser Service*.

3.2.2 Query Parser Service (Step-3/4)

The query in Figure-5 will be the input of this step. A parser is a set of rules that describe the structure, or syntax, of a user query. User query is broken down into set of keywords, based on *Parser Rules Repository*. This

Table/ Schema Name	Col Name	Search Condition	Insertion ID	Time Stamp	ResultSet DB
CityB_DB.emp	empno, ename, job, desig, sal	Job= ”Manager”	CityB_DB_1	00-00-00: 27-06-04	CityB_DB1_tab

Figure-7: Record Structure of KSS for Scenario Two

process is described in Figure-6. For further processing of query, these keywords would be used.

3.2.3 Knowledge Search Service (Step5)

The keywords in Figure-6 will be the input of this step. Using the keywords and *Knowledge-Based Repository,* this particular service will decide how much data will be available in *CacheDB*.

According to our assumption our Cache (*ResultSet DB*) contains complete record set related to City-B.

The *Knowledge-Based Repository (KBR)* has the complete record of previously executed query as shown in Figure-7.

The *KSS* searches Insertion IDs from its repository on the basis of keywords of Figure-6 and then sends all concluded results to the next component i.e. Intelligent Query Builder. In this scenario it returns CityB_DB1_tab as the table of CacheDB (ResultSet DB) and CityB_DB_1 referring the Insertion ID of matched records. In case of partial results found in KBR, there is a need to set some flags and parameters that indicate how much of data is there in CacheDB, in order to tell other components of the system to treat the user query accordingly.

3.2.4 Intelligent Query Builder (Step-6)

On the basis of the information set/extracted by *KSS* and then passed to IQB (Intelligent Query Builder) for further query processing, IQB will be responsible for building queries.

In the above scenario, *KSS* passed the Insertion ID of existing records in cache DB and also looked for that. If complete match is found in CacheDB, IQB will only have to make single query plan of local domain for *Query Router* (Step-9 of Figure-2). *Query Router* is responsible for directing each and every query composed by IQB towards its destination. As there is no remote location involved for resultant dataset in this example, so, *Query Router* (Step-9 of Figure-2) does not perform the remote query.

3.2.5 ResultSet Management & Integration Service

In this scenario, the Q*uery Router* will route the queries to the ResultSet DB (Step-10a of Figure-2). The resultant dataset extracted by the query is directly forwarded to the ResultSet Management and Integration Service (Step-12). This service is also gets information from *Query Router* about the actual query plans and their destinations, in order to entertain the datasets that are expected from different domains. Local domain is responsible for providing complete dataset, which will be forwarded to the user application. Because of the complete involvement of local domain, no updation is required in the ResultSet DB and KB Repository (Step-14*).

```
SELECT    empno, ename, job, desig, sal , age
FROM      CityB_DB.emp,CityA_DB.emp, CityC_DB.emp
WHERE     job = 'Manager'
AND       age <35
```

Figure-8: Input query for scenario three

3.3 Scenario Three

Assumptions: User want records of Manager with age less than 35 years, from remote databases with fields i.e. employee No., name, job, designation, salary and age. This particular query as shown in Figure-8, is executed from City-A. As City-B and City-C do not come under the hierarchy of City-A, there is no replica of CityB_DB and CityC_DB at City-A. But on the other hand as our user at City-A frequently performs the same query on the database at City-B, so the Cache DB will contain exact matching record sets of employees with required columns and conditions.

```
Schema Name   = {CityB_DB, CityA_DB , CityC_DB}
Tables Name   = {emp}
Column Name   = {empno, ename, job, design, sal}
Search Condition = {CityB_DB.job = 'Manager', age<35}
JoinCondition  = {!Null}
```

Figure-9: Breakdown into keywords in QPS for scenario three

```
{CityB_DB, emp, empno, ename, job, design, sal,
 CityB_DB.job = 'Manager' , age<35}

{CityA_DB,emp, empno, ename, job, design, sal,
 CityA_DB.job = 'Manager' , age<35}

{CityC_DB, emp, empno, ename, job, design, sal,
 CityC_DB.job = 'Manager' , age<35}
```

Figure-10: Output of QPS for scenario three

3.3.1 Local Fast Check (Step-1/2)

As described in above assumption that City-B and City-C do not belong to the hierarchy of City-A, so in this step nothing is found when completes user query is executed on local domain by the distributed DBMS in operation. Soon after this step, complete query is handed over to the Query Parser Service.

3.3.2 Query Parser Service (Step-3/4)

When user query comes to the Query Parse Service it breaks-down it into a set of keywords as shown in the Figure-9, based on the rules mentioned and stored in Parser Rules Repository, for further searching from repository.

The query parser will then make three sets out of the user query as described in Figure-10.

3.3.3 Knowledge Search Service (Step5)

Now the parsed query is at the *KSS* (*Knowledge Search Service*), which on the basis of *KBR* (*Knowledge-Based Repository*) decides how much data is available in Cache DB. The Knowledge-Based Repository (KBR) has following structure and only one tuple exists, which is shown in Figure-11. According to the data available with KBR, only record set of City-B is available with Cache DB.

The *KSS* extracts Insertion IDs and the name of ResultSet DB from the KBR. It also sets proper flags and parameters, which reflect that the record set of City-A and City-B is available with local domain, and for City-C a remote query is required. *Intelligent Query Builder* (IQB) receives all of the above flag and parameters

3.3.4 Intelligent Query Builder (Step-6)

At this point, the IQB analyzes the information coming from *KSS* and builds the queries.

In this scenario IQB have received information as:

1. There is no result for CityC_DB contained in the local domain, so it has to build a remote query for it. For this purpose it requires to consult from Metadata Repository (Step-8) for the physical location of CityC_DB in the Data Grid.
2. The result relating to CityA_DB is contained in the LocalDB, so, it has to build a query for the LocalDB.
3. Complete result set for CityB_DB is maintained by Cache DB (ResultSet DB), so there is no need for any remote query, but we require a full fledge query from Cache DB (ResultSet DB) based on the Insertion ID and table name provided by the KSS.

It is now the responsibility of the *Query Router* to make query plans and send appropriate query towards its deserving domain for the resultant dataset of user query. These query plans are also forwarded to ResultSet Integration and Management Service, so that, it is able to accept the resultant dataset from respective domains.

Table/ Schema Name	Col Name	Search Condition	Insertion ID	Time Stamp	ResultSet DB
CityB_DB.emp	empno, ename, job, desig, sal	Null	CityB_DB_1	00-00-00: 27-06-04	CityB_DB1_tab

Figure-11: Record structure of KBR for scenario three

3.3.5 ResultSet Management & Integration Service

The resultant data sets, extracted by those queries will be passed to this component of the system. It is responsible for merging all the given result sets according to the distribution scheme provided by IQB.

```
SELECT    empno, ename, job, desig, sal , age
FROM      CityB_DB.emp
WHERE     age <50;
```

Figure-12: Input query for scenario four

```
Schema Name   = {CityB_DB}
Tables Name   = {emp}
Column Name   = {empno, ename, job, design, sal}
Search Condition = {CityB_DB.job = age<50}
JoinCondition  = {Null}
```

Figure-13: Breakdown into keywords by QPS in scenario three

```
{CityB_DB, emp, empno, ename, job, design, sal,
CityB_DB.job = 'Manager' , age<50}
```

Figure-14: Output of QPS for scenario three

Another important task, that needs to be accomplished in order to complete this cycle is to maintain the Cache DB and KB-Repository, so that the keywords generated by the Query parser against remote query are saved in KB-Repository and related data result set saved into the Cache DB (ResultSet DB) as shown in Step-14* of Figure-2.

After the completion of above functions, the system is able to make the resultant data set available to the user.

3.4 Scenario Four

Assumptions: User want records of Manager with age less than 50 years, from remote databases with fields i.e. employee No., name, job, designation, salary and age. In this particular scenario our source is City-A and the user wants the records again from City-B. In the absence of City-B replica at City-A, the system starts checking the Cache DB and found out that the two different Insertion IDs in Knowledge-Based Repository (which if merged together) can give us the resultant data set as required by the user.

3.4.1 Local Fast Check (Step-1/2)

According to the situation described above, when user query of Figure-12 is executed by the distributed DBMS in operation, no exact match is found from local DBs and the complete user query is handed over to Query Parser Service.

3.4.2 Query Parser Service (Step-3/4)

First of all the Query Parser Service breaks down the user query in Figure-12 into set of keywords as in Figure-13 based on the Parser Rules Repository. After this step the process of further searching from Knowledge-Based Repository can be continued.

The query parser then makes sets out of the user query as shown in Figure-14.

3.4.3 Knowledge Search Service (Step5)

At this step, the keywords and sets are used to search matching records from KBR (Knowledge-Based Repository) and it is decide that how much data is available in Cache DB. The Knowledge-Based Repository (KBR) has the structure as shown in Figure-15. It is found that the two tuples exist in KBR, those needs to be merged together in order to find the query.

When query is performed according to the set explained in Figure-14 by *KSS,* both Insertion IDs are extracted with the name of ResultSet DB from the KBR. After setting proper flags and parameters, the preceeding components can be made well aware of what data is available with Cache DB. In this scenario complete data set is available with Cache DB and there is no need to perform remote query. This information is forwarded to the *Intelligent Query Builder* (IQB), that is responsible for modifying the query by splitting or merging based on the user query or parameters provided by *KSS*.

3.4.4 Intelligent Query Builder (Step-6)

By intelligently analyzing the information provided by *KSS* to this component, it is decided that by merging both datasets available with CacheDB, gives the resultant dataset of user query. So the query is composed for Cache DB (ResultSet DB) on the basis of Insertion IDs and table name provided as given by *KSS*.

Query Router now makes query plans and sends the appropriate query towards CacheDB and ResultSet Integration and Management Service so that the resultant dataset can be handled properly from CacheDB.

3.4.5 ResultSet Management & Integration Service

Because data is extracted from CacheDB, so there is no need to perform Step-14* as in Figure-2. After the arrival of datasets from CacheDB and checking the timestamp, they are simply forwarded to the user application.

We have discussed some of the possible scenarios in our paper, there can be many more. Also there is a possibility of occurrence of very complex scenario arising from complex queries, in that case the system will check the search conditions and in case of no exact match it will intelligently forward the query to the remote domain instead of solving the complexities itself. So, in one-way or another our system will work in almost all situations resiliently, but we are assuming that the occurrence of these complex situations is very rare.

4 Conclusion

This paper provides architecture of intelligent cache management system and environment where efficient cache management can help us to reduce the access time of remote accesses in a data grid. We looked at some common scenarios where replication and cache can provide us better and efficient retrieval of record set. We are implementing a prototype for intelligent cache for data grid and will be monitoring its impact after integration with existing OGSA services. We believe that this Intelligent Cache Management, and its integration with existing systems like Globus, will provide new dimensions for next generation of homogeneous, distributed data and computational intensive, high-performance systems. The Intelligent Cache Management component can be an important component for such data grid environment.

Table/ Schema Name	Col Name	Search Condition	Insertion ID	Time Stamp	ResultSet DB
CityB_DB.emp	empno, ename, job, desig, sal	age<=35	CityB_DB_1	00-00-00: 27-06-04	CityB_DB1_tab
CityB_DB.emp	empno, ename, job, desig, sal	age>=36 AND age<=50	CityB_DB_5	00-00-00: 28-06-04	CityB_DB1_tab

Figure-15: Record structure in KBR for scenario four

5 Acknowledgements

We are thankful to the members of "High Performance Computing/Grid Computing Research Group" at Muhammad Ali Jinnah University, Islamabad, especially Mr.Mohib-ur-Rehman and Mr.Imran Ihsan for their valuable comments.

6 References

Aberer, K. (2003): Conception of Information Systems Part 2: Integration of Heterogeneous Databases. *EPFL-SSC, Laboratoire de systèmes d'informations rèpartis*.http://lsirwww.epfl.ch/courses/cis/2003ss/part2Heterogeneity.pdf, Accessed 01 Nov 2004.

Agarwal, A., Simoni, R., Hennessy, J. and Horowitz, M. (1988): An evaluation of directory schemes for cache coherence. *Proceedings of the 15th Annual International Symposium on Computer architecture,* Honolulu, Hawaii, United States, **16**: 280 –298, IEEE Computer Society Press.

Allcock, B., Bester, J, Bresnahan, J., Chervenak, AL., Foster, I., Kesselman, C., Meder, S, Nefedova, V, Quesnel, D and Tuecke, S.(2001): Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. *Eighteenth IEEE Symposium on Mass Storage Systems and Technologies (MSS'01),* Hyatt Regency Islandia, San Diego, USA, P. **13,** IEEE Computer Society Press.

Cannataro, M. and Tali, D (2003): The Knowledge Grid. *Communications of the ACM,* **46**:89-93, ACM Press.

Castro, M, Adya, A., Liskov, B and Meyers, AC. (1997): HAC: hybrid adaptive caching for distributed storage systems. *Proc. ACM symposium on Operating systems principles,* Saint Malo, France, **31:**102 - 115, ACM Press.

Chervenak, A, Foster, I., Kesselman, C., Salisbury, C and Tuecke, S. (2001): The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications,* 23:187-200.

Cortés, U., Martínezb, M., Comasb, J., Sànchez-Marrè a, M., Pochb, M. and Rodríguez-Roda, I. (2004): A conceptual model to facilitate knowledge sharing for bulking solving in wastewater treatment plants. *Proc. AI Communications*. The European Journal on Artificial Intelligence, **16**(4):279-289.

Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002): The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22, 2002.

Foster, I. , Kesselman, C. and Tuecke, S. (2001): The Anatomy of the Grid Enabling Scalable Virtual Organizations. *Intl. Journal of High Performance Computing Applications,* **15**(3):200-222.

Moore, RW. (2001): Knowledge-Based Grids. *Eighteenth IEEE Symposium on Mass Storage Systems and Technologies (MSS'01),* Hyatt Regency Islandia, San Diego, USA, P. **29,** IEEE Computer Society Press.

Otoo, E., Olken, F. and Shoshani, A. (2002): Disk cache replacement algorithm for storage resource managers in data grids. *Proceedings of the 2002 ACM/IEEE conference on Supercomputing,* Baltimore, Maryland, 1-15, IEEE Computer Society Press.

Park, S., Lee, D., Lim, M. and Yu, C (2001): Scalable data management using user-based caching and prefetching in distributed virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology,* Baniff, Alberta, Canada, 121-126, ACM Press.

Qin, X and Jiang, H (2003): Data Grid: Supporting Data-Intensive applications in Wide-Area Networks. *Technical Report No. TR-03-05-01,* Department of Computer Science and Engineering, University of Nebraska-Lincoln.

Singh, G, Bharathi, S, Chervenak, A, Deelman, E, Kesselman, C., Manohar, M, Patil, S and Pearlman, L (2003): A Metadata Catalog Service for Data Intensive Applications. *Proceedings of the ACM/IEEE SC2003 Conference*, Phoenix, Arizona, P. **33**, IEEE Computer Society Press.

Tierney, B., Johnston, W. and Lee, J. (2003): A Cache-Based Data Intensive Distributed Computing Architecture for "Grid" Applications. *CERN School of Computing*, Sept, 2000.