Future Generation Computer Systems 26 (2010) 1418–1425

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

VDB-MR: MapReduce-based distributed data integration using virtual database

Yulai Yuan, Yongwei Wu *, Xiao Feng, Jing Li, Guangwen Yang, Weimin Zheng

Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, 100084, China

ARTICLE INFO

Article history:
Received 30 November 2009
Received in revised form
30 March 2010
Accepted 16 April 2010
Available online 24 April 2010

Keywords:
MapReduce
Virtual database
Data integration
Heterogeneous resources
Query execution

ABSTRACT

Data Integration is becoming very important in many commercial applications and scientific research. A lot of algorithms and systems have been proposed and developed to address related issues from different aspects. Virtual database systems are well-recognized as one of the effective solutions of data integration. The existing execution modules in virtual database systems are very ineffective. MapReduce (MR) is a new computing model for parallel processing and has a good performance on large-scale data execution. In this paper, we propose a new distributed data integration system, called VDB-MR, which is based on the MapReduce technology, to efficiently integrate data from heterogeneous data sources. With VDB-MR, a unified view (i.e., a single virtual database) of multiple databases can be provided to users. We also conducted a series of experiments to evaluate VDB-MR by comparing it with an open source data integration system OGSA-DAI and two DBMSs in parallel. Experiment results show that VDB-MR significantly outperforms OGSA-DAI and the DBMSs in parallel.

1. Introduction

Nowadays, data integration is becoming important in many commercial applications and scientific research. It integrates multiple databases and heterogeneous resources on the Internet so that a unified view of these databases and resources can be provided to users. Many approaches and systems have been proposed and developed to achieve data integration [1–11], having the same or a similar objective: providing flexible and efficient access to information residing in a collection of distributed, heterogeneous and overlapping resources such as databases, plain texts, file systems, distributed file systems, XML files, World Wide Web, and many other information sources.

As one of the existing data integration approaches, virtual database is, however, composed of a set of models, which specify the structural characteristics of data sources, views and web services. Many systems based on these models have been developed to deal with various real problems of data integration. On the Internet, virtual databases are used to collect information from different sites in order to provide complete information about books and other commodities [3]. In the environment research field, they are also a useful approach to integrate spatially-related environmental data so that advanced Web-based retrieval, analysis and visualization of these data can be facilitated [2]. In E-commerce systems, virtual databases act as a "broker" to aggregate information from multiple enterprise systems for the purpose of reducing costs associated with inter-business transactions and improving information accuracy between co-operating businesses [4].

A typical structure of a virtual database, as shown in Fig. 1, is composed of four components: *Mapper*, *Publisher*, *Executor*, and *Wrapper*. *Mapper* usually specifies a global schema according to the information contained in the entire collection of resources. *Publisher* provides a query language for users to access the system and operate data shown by the global schema. Algorithms are designed to interpret this query language and decompose each global query into sub-queries on physical resources. *Executor* is in charge of executing these sub-queries in their corresponding resources, merging query results, and dealing with possible conflict and inconsistency of the query results. *Wrapper* (or *Adapter*) directly connects to the resources, translates the sub-queries into the form that the resources can comprehend, and standardizes the answers.

Much research work has been done to address issues as to how a global schema is extracted from autonomous resources, how an effective query language is identified, and/or how decomposing queries are optimized [12,13]. However, research on query execution has not been given sufficient attention. One may suggest that query execution in virtual databases can be considered as a problem of query execution in distributed databases. However, these two query mechanisms have vast differences in many aspects, especially in terms of the high-level autonomy and heterogeneity between member resources systems, for the following reasons. First, communication autonomy in multi-resource systems means that member resources independently determine what to share, and when and how to participate in a global system. Second, design autonomy makes it impossible to

* Corresponding author.
E-mail address: wuyw@tsinghua.edu.cn (Y. Wu).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.04.001

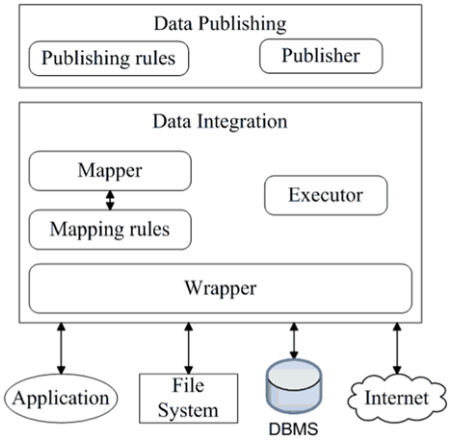


Fig. 1. Typical structure of a virtual database.

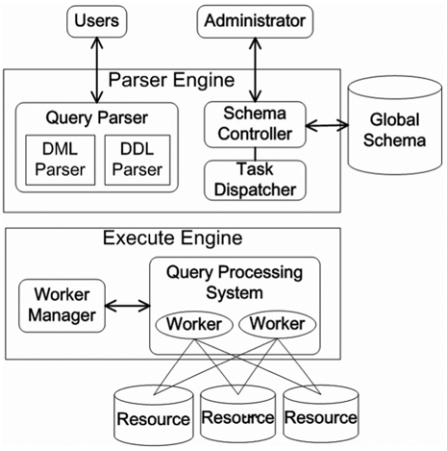


Fig. 2. Architecture of VDB-MR.

optimize resources of each member in terms of the needs of the whole system, as members do not indeed have that obligation. Third, execute autonomy leads to the result that the whole system is not able to decide how the processing is carried out in each member [14]. In this paper, more attention will be paid to the query execution for higher efficiency, compared with traditional data integration approaches.

MapReduce [15,16] is a software framework introduced by Google to facilitate processing and generating huge datasets on a large number of computers/nodes. It is simple since it only contains two independent operations: Map and Reduce. MapReduce allows for distributed processing of these two operations. MapReduce is best suited for executing large-scale datasets in parallel and has well proved success in Google [15] and Hadoop communities [17]. MapReduce has been successfully applied in the database field, for example Amazon Dynamo [18] and HadoopDb [19]. But these works mainly focus on isomorphic data resources stored in the form of key-value. To the best of our knowledge, no work has been done to apply MapReduce for data integration of heterogeneous data resources such as database or file systems.

The objective of our research is to provide a solution for distributed data integration of virtual databases by utilizing the MapReduce technology. In this paper, we propose a distributed data integration system, named as VDB-MR, based on the MapReduce technology, to efficiently integrate data from heterogeneous data sources. In order to provide uniform access to users; a SQL-like query language is also particularly designed for VDB-MR. Considering the relativity with lower data management technology, as well as execution efficiency, we developed a flexible implementation of MapReduce independently for our VDB-MR. With VDB-MR, a unified view (i.e., a single virtual database) of multiple databases can be provided to users. A series of experiments are conducted to evaluate VDB-MR by comparing it with an open source data integration system OGSA-DAI and two DBMSs in parallel. Experiment results show that VDB-MR significantly outperforms OGSA-DAI and the DBMSs in parallel.

The rest of the paper is organized as follows. In Section 2, we present the architecture of VDB-MR. Section 3 discusses the MapReduce-based implementation of VDB-MR. In Section 4, the experiments are discussed and the experiment results are presented and analyzed. Related work is given in Section 5 and we draw the conclusion in Section 6.

2. Architecture of VDB-MR

In this section, we discuss the overall architecture of VDB-MR (Section 2.1) and a SQL-based query language particularly designed for VDB-MR (Section 2.2).

2.1. Architecture

The overall architecture of the distributed data integration system we propose in this paper, VDB-MR, is presented in Fig. 2, which is based on the virtual database system described in [20]. The architecture of VDB-MR is similar to the typical structure of virtual database presented in Fig. 1. The objective of VDB-MR is to integrate data in disparate databases and file systems and provide a uniform access to users. A SQL-like query language is particularly designed for VDB-MR and will be described in Section 2.2 in detail. As the brain of the system, the *parser engine* manages the global schema which specifies the way users perform mapping and queries. The *execute engine* is however the body of the system. It collects metadata from member resources, optimizes and executes query processing. Because the parse engine and *execute engine* of VDB-MR are dispatched in a cluster, how to improve parallel execution efficiency is an important research issue, which will be discussed in Section 3.

2.2. SQL-based query language

Though we mainly focus on data management, the set of query syntaxes specified in our query language also includes syntaxes for users and resources management. Virtual database provides interfaces like a standard database management system (DBMS), so our query language is naturally based on SQL, including basic operations such as *Select*, *Delete*, *Update*. Additionally, some other operations are newly proposed for the purpose of efficient management of distributed systems. In the rest of the section, we describe four important operations (i.e., *Map*, *Select*, *Join*, and *Update*) of our SQL-based query language with the emphasis on query execution.

2.2.1. Map

Map is a very import operation, which specifies how a virtual database or a virtual table links to real resources. Its basic regulation expression is provided below.

```
$map  = Map Vtable $vtable_name[($attribute_name
        [, attribute_name])]
    From $resource [ And $source];
$source = $resource_name:$table_name($map_attribute
        [, $map_attribute]);
$map_attribute = $null | $resource_table_attribute_name
```

As mentioned previously, the wrapper of VDB-MR provides uniform interfaces like a DBMS for difference resources. The Map operation is used to map a virtual table into several tables in different resources, so that the execution of the virtual table can be

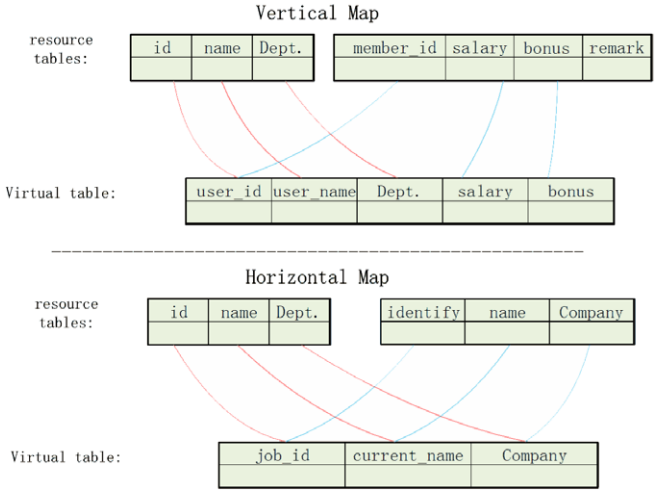


Fig. 3. Two kinds of map operations.

transferred to the real table. There are two kinds of Map: *vertical map* and *horizontal map*, as shown in Fig. 3. In *vertical map*, the attribute number of resources tables does not necessarily equal the attribute number of virtual tables, while in *horizontal map*, they must be equal.

The map operations have the following characteristics. First, it is valid to have *vertical* and *horizontal* maps in the same map query. Second, each primary key of a virtual table must be mapped into an attribute of at least one mapped resource table. For example, the primary key of the virtual table shown in the vertical map of Fig. 3 is *user_id*, and it must be mapped into an attribute for each real table, as *id* and *member_id* in this example. Third, Key is an important feature of a database table; it guarantees the data validity and prevents data from confliction. Fourth, only if every resource table provides all correlated attributes of the key attributes defined in a virtual database, can we then consider this map as a legal operation.

After the map operation, the mapping result is stored in the *Parser Engine*, so that the query for the virtual table can be translated to execution for the real table according to the corresponding relation of the attribute.

2.2.2. Select

As opposed to the complex syntax of the select operation of SQL, the select operation of our query language is however very concise; it does not support nested queries. This is because such a simple select operation can satisfy most query requirements: complex queries can be achieved by combining map, simple select and join operations, and a well-designed schema of virtual database systems rarely requires complicated select operations like the one provided in normal SQL. The basic regulation expression of the select operation of VDB-MR query language is given as follows.

$select = Select $select_expr [Into $new_vtable_anme]
From $vtable_name [Where $where_condition]
[Order By $attribute_name [, $attribute_name];
$select_expr = * | ($attribute_name [, $attribute_name])

2.2.3. Join

The join of the VDB-MR query language is also a simplified version of the join operation of SQL. Its regulation expression is provided as follows.

$join = $table_reference Join $table_name
[On $condition_expr]
[Order By $attribute_name [, $attribute_name]];
$table_factor = $table_name | $select

The join also just supports equivalent expressions. In other words, we only consider the execution of equivalent joins.

2.2.4. Update

The update operation is used to update data in different tables and it is an integration of a series of operations such as update, delete, and insert. Their syntaxes are mainly similar to those in SQL, see Reference [21] for more information.

3. Implementation of VDB-MR execute engine

Although the MapReduce technology is applied in VDB-MR, we do not utilize any available open source implementations of the MapReduce technology because of the following three reasons. First, VDB-MR requires an open environment for a variety of experiments in terms of query decomposition and execution. Many existing MapReduce systems cannot satisfy this requirement. Second, some existing implementations of MapReduce are not flexible to be configured for the purpose of query execution optimization. Third, third-party implementations may contain undesired components which may degrade query execution efficiency. With all these concerns, we developed a flexible implementation of MapReduce for VDB-MR, which is suited to execute query processing and is more convenient for our experiments. In the rest of the section, the structure of our MapReduce-based *Execute Engine* is discussed first, followed by the query execution strategy we propose in the paper.

3.1. Structure of our mapreduce-based execute engine

The structure of our MapReduce-based *Execute Engine* is presented in Fig. 4. Execution Engine is a distributed parallel system, of which the smallest unit is the process. We call each process of the execution engine a worker. Compared with the MapReduce structure described in [15], our system is more complicated since we want to provide more interfaces for the experiments and function extension. Our experiment results (Section 4) show that it is valid.

Besides the three commonly contained processes (map-only process, MapReduce process, and reduce-only process), there are two other important processes: manage process and collect process in our system. As shown in Fig. 4, the *execute engine controller* sends a list of queries to a worker during the manage process and the worker assigns MapReduce tasks to the master worker, which is responsible of directing all the slaves (all the other workers) in doing their work. Notice that a worker is an atomic element executing tasks in a parallel execution framework. In the rest of the subsection, we describe them one by one.

3.1.1. MapReduce process

The MapReduce process is the most common process of the MapReduce model. First, the map process pre-analyzes input data and maps these data to different reduce workers, these reduce workers obtain intermediate data from the master worker and reduce them using pre-defined algorithms. In our query execution system, most of the input data of the MapReduce process are the lists of queries decomposed by the parse engine shown in Fig. 2. Map workers need to execute these queries first, translate the query result into collections of ⟨Key, Value⟩ pairs, and partition these collections for the upcoming reduce process. Reduce workers obtain the input data prepared by the map process and merge them according to the regulations we specified for our system (Section 2.2).

3.1.2. Map-only process

Some basic operations (e.g., the update operation) can be finished in only a single round of the map process. Most of these operations just update data in member resources and return a message to indicate whether the operations are successful or not. This process maps operations to different workers and then executes them concurrently.

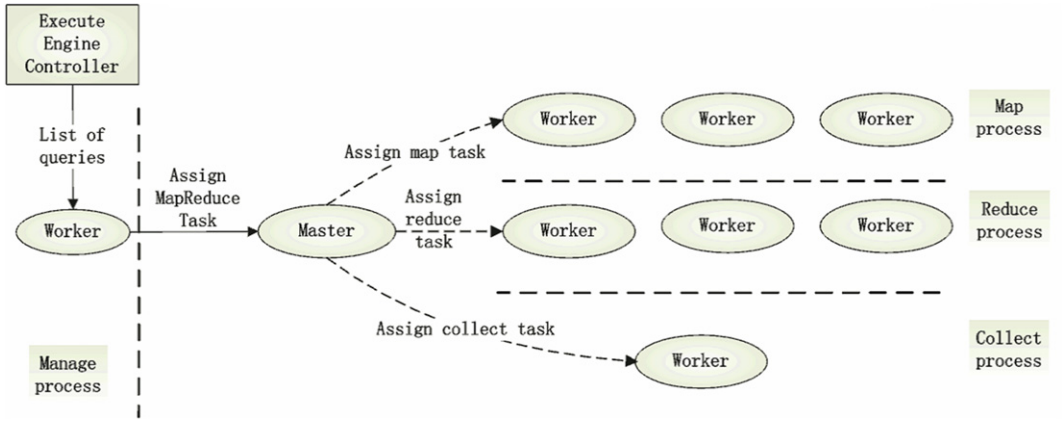


Fig. 4. Structure of our MapReduce-based *Execute Engine*.

3.1.3. Reduce-only process

For some complicated operations a single round of the MapReduce process is not sufficient, in which case a sequence of reduce processes are required to further process the results generated by the first round of the MapReduce process. The sequence of reduce processes following a first round of the MapReduce process is called the Reduce-only process.

3.1.4. Manage process

In most existing MapReduce systems, the strategy of map process and reduce process is predefined manually, because it is hard for these systems to obtain an appropriate schema from the mass data (e.g. mass data processed by search engines of World Wide Web). Though one of the important goals of our system is to deal with mass data, the scale of data we manipulate is far smaller than the scale of data processed by search engines of the World Wide Web. Therefore we assign a manage process to make a plan of each query execution and control the execution of each map reduce round. The manage process can be considered as the core of our query execution system. It translates query lists into tasks which can then be executed. It is also responsible for sending back the query results to the *Parser Engine*, which are then returned to the user.

The responsibility of the manage process is different from the master worker's responsibility. The master picks idle workers, assigns them tasks, monitors the status of these tasks, and guarantees the normal operation of the process. The manage process, however, defines the strategy of a query execution, decides the number of map workers and reduce workers and the rounds of MapReduce processes, generates tasks for each MapReduce round, and sends them to the master of the round for execution.

3.1.5. Collect process

Results generated by the reduce process are mostly distributed in different machines and a collector is required to collect and merge these results together. As compared to a regular reduce worker of the reduce process, a collect worker of the collect process needs to translate the results into a format that can be recognized by other modules of the virtual database systems, and then send the results with this certain format back to *Parser Engine*.

3.2. Strategy of our MapReduce-based VDB-MR system

As previously discussed in Section 2, our SQL-based query language contains four main query operations: *Map*, *Select*, *Join* and *Update*. The *Map* operation has nothing to do with the data, which only defines the relationships between the virtual table and the real table from data resources, and is executed in the *Parser Engine*.

So, in the rest of the section, we discuss the *Select*, *Join* and *Update* operations one by one.

3.2.1. Select operation

Select operation is the most frequently-used operation in a virtual database system. When the virtual database system receives a select query, it decomposes the query into a set of operations (a set of sub-queries) according to different member resources, which can then be executed on the member resources, according to the map regulations of the virtual table pre-specified by users. Since the *select* operation only supports a single virtual table, as discussed in Section 2, the decomposition is not complex. When the sub-query list is generated it is sent to the query execution system. The manage process in the query execution system defines the procedure of map and reduce processes, decides the number of map workers and reduce workers, and then executes the sub-queries. Generally, a *select* operation can be finished by one round of map and reduce operations. In the rest of the section, we discuss the main strategies of the map, reduce, manage, and collect processes when a *select* operation is executed.

Map: Each map worker needs to complete two tasks. First, a map worker executes the sub queries sent to it in the related database and collects the query result which must be a form of table. Second, the map worker translates the query result into a collection of ⟨Key, Value⟩ pairs and partitions the collection for the reduce workers. Each row of the result table is translated into a ⟨Key, Value⟩ pair. The key of the pair is a collection of values of the attributes that have been mapped to the key attributes of the virtual table. The value of the pair is the data of the corresponding row.

Reduce: As a traditional reduce process, a reduce worker collects ⟨Key, Value⟩ pairs with equal keys and forms a collection. The elements of this collection are merged into a ⟨Key, Value⟩ pair by the reduce worker. As the keys of these pairs are equal, this merge action is actually the merge of rows in different resources of tables that have same values of key attributes. So the resulted ⟨Key, Value⟩ pair is translated into a row of the resulting table.

Manage: During the manage process, sub-queries are partitioned into a number of sub lists according to the number of related resources. Each of these lists contains sub-queries which need to execute in the same resource. The number of map workers is the same as the number of sub lists. The number of reduce workers is half the number of map workers. When the reduce worker merges the ⟨Key, Value⟩ collection, values of different attributes defined in the virtual table are connected together. If there are several different values of an attribute, the first value occurring will be selected.

Collect: The collect process is the last step of a select query. After reduce workers have finished all the reduce tasks, a collect worker collects the reduce results from different locations, merges them together in order, and translates these data into a table.

An example is provided as follows to illustrate *Select* operation.

Create vtable vt1(col_a Int, key, col_b String key, col_c int, col_d int);
Map vtable vt1 into resource1:table1(t1a, t1b, other1,)
And resource2:table2(t2a, t2b, ,other2);
*Select * from vt1 where col_a = col_b and col_c = 5;*

The sub-query list of this query is given as follows:

Select t1a as col_a, t1b as col_b, other1 as col_c in table1 where col_a = col_b and col_c = 5;
Select t2a as col_a, t2b as col_b, other2 as col_d in table2 where col_a = col_b;

In this example, two map workers are requested to execute the two queries in *table 1* and *table 2* separately. The key of ⟨Key, Value⟩ pair is {value of *col_a*, value of *col_b*}. The merge operation of the reduce process will merge the key pairs with the same key (e.g., the values of *col_a* equal the values of *col_b*).

3.2.2. Join operation

As discussed in Section 2.2.3, our system only supports equivalent joins; therefore the algorithm of hash join first introduced in [22] is suitable for the implementation of our *join* operation. The MapReduce technology is also based on the hash algorithm, so it is easy to deal with the *join* operation defined in this virtual database system.

The *join* operation needs two rounds of the MapReduce process. The first round of the MapReduce process selects the necessary data from the two related virtual tables separately, assuming that these data have been prepared and distributed in different locations. Then the second round of the MapReduce process starts, with the input of this round being the resulting data of the first round. Map workers of the second round collect the ⟨Key, Value⟩ pairs and reform the key of each pair. The new key of the ⟨Key, Value⟩ pair is a collection of join attributes in the join condition, and the value of the pair remains unchanged. An input ⟨Key, Value⟩ pair can generate several outputs with different keys. For example, if the join condition is

(t1.a = t2. b and t1.c = t2.d) or (t1. e = t2.f and t1.g = t2.h)

Then the new Key of ⟨Key, Value⟩ pairs (i.e., the result of $t1$) should be decomposed into two pairs:

⟨*{value of attribute t1.a, value of attribute t1.c}, row value*⟩ *and* ⟨{*valueofattributet1.e*, *valueofattributet1.g*}, *rowvalue*⟩.

Fortunately, every complex condition expression can be translated into the form of

($expr [and $expr]) [or ($exp [and $expr])].

So we can use this MapReduce strategy to solve all join problems.

After the map process, the reduce workers of the second round merge the ⟨Key, Value⟩ pairs with same keys. Besides checking whether the keys are equal or not, the attributes names in the key also need to be matched as described in the join condition. Then the collect process collects the result and sends it back to the *Parser Engine*, which then returns it to the user.

In our implementation, we start two MapReduce rounds concurrently to finish the select query of two virtual tables, then a MapReduce round for the *Join* operation. The number of map workers for the second MapReduce round is determined by the scale of data of the first MapReduce round, which is automatically calculated at the end of the first MapReduce round. Furthermore, the number of reduce workers is half of the number of map workers.

3.2.3. Update operation

Actually, the *update* operation includes three operations: insert, delete and update. They have the same execution process. A map-only process is enough to execute the *update* operation. The map task is similar to that described in Section 3.1.2. A collect process is also necessary at the end to calculate the number of influenced rows.

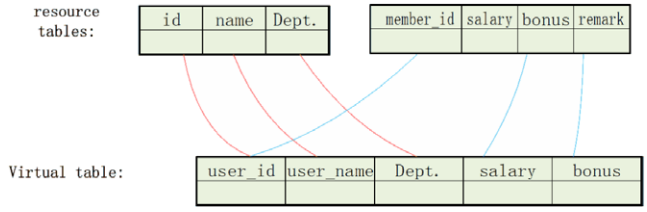


Fig. 5. Virtual table and the resource tables.

4. Experiment and evaluation

A series of experiments has been conducted to evaluate our VDB-MR system by comparing it with other two systems: OGSA-DAI [23] and two DBMSs in parallel, with an emphasis on the performance of the execution engines measured by the execution time of each system. In the rest of the section, we discuss the experimental environment in Section 4.1 followed by the design of the experiments (Section 4.2). The experiment results and analysis are presented in Section 4.3.

4.1. Experimental environment

The experimental environment is composed of a cluster with eight nodes, a front-end machine and a user machine. Each of these computers was running on the Linux 2.4.21-20.EL operating system. Their principal components are two Intel Itanium 2 IA-64 CPUs with a clock speeds of 1.3 GHz, 4 GBytes of DDR2 RAM, and two 36.4 GByte SCSI hard drives. We installed VDB-MR in this cluster, OGSA-DAI in a machine the same as the cluster node, and Mysql installed in one cluster node acts as the DBMSs in parallel.

4.2. Experiment design

We constructed two tables in two different databases as the resource tables shown in Fig. 5. The goal of the experiment is to merge the content of these two tables into one according to the links shown in shown in Fig. 5.

For the OGSA-DAI database, the process of execution is to query the content of these two tables in parallel and then join the query results together. For the two DBMSs in parallel, we need to store content of these two tables in the DBMSs first, and then join and return query results. For our VDB-MR system, we first need to customize the virtual table and the map rule and then execute the *select* operation.

4.3. Experiment results and analysis

In this section, we describe the series of experiments we conducted to compare the execution time of our system with the other two systems when data size is both below and above 20,000 (large-scale data size). We also compare the performance difference of our system when large-scale data is executed on a single multi-core node vs. a set of multiple nodes.

4.3.1. Execution time of data with size under 20,000

The execution time of different data sizes across different systems is shown in Fig. 6. The abscissa axis is the number of rows (indicating the size of data) and its maximum is 20,000. The y-axis is the execution time of different sizes of data. As indicated in the figure, the execution time of VDB-MR is around 1.5 s and less than 2 s for 20,000 rows of data; while the other two systems require as much as 60 s (DBMS system) and 70 s (OGSA-DAI).

For more clarity, the execution time of VDB-MR dealing with data within 20,000 rows is again presented in Fig. 7. The execution time increases along with the size of data. 20,000 rows of data are relatively small for our system and the experiment result is vulnerable to scheduling, the network environment and response speed of data resources; therefore the experiment results are

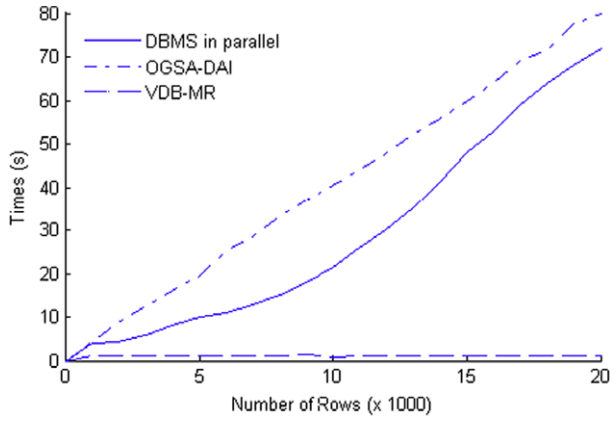


Fig. 6. Execution time of data with size under 20,000 across different systems.

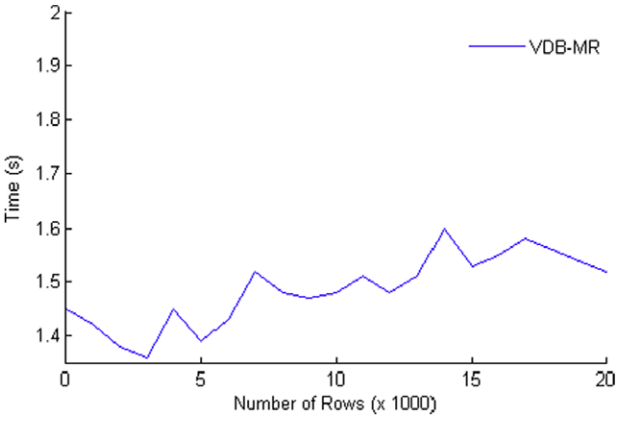


Fig. 7. Execution time of data within 20 000 rows of VDB-MR.

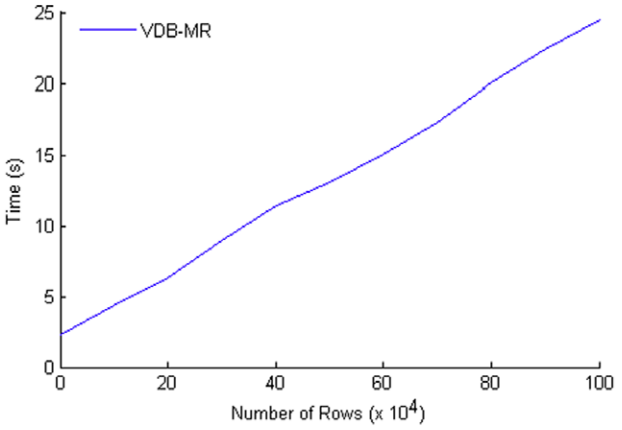


Fig. 8. Execution time of large-scale data of VDB-MR.

not strictly monotonically increasing. But in fact, the standard deviation of the execution time is less than 0.1 s.

4.3.2. Execution time of large-scale data

For more than 20,000 rows of data, the execution time of OGSA-DAI and the DBMS-based system reached an intolerable level of execution time: more than 600 s. Furthermore, OGSA-DAI met problems such as memory overflow because of its implementation mechanism. Therefore, we only present the experiment results from our system in Figs. 8 and 9.

From Fig. 8, we can see that the execution time of VDB-MR is monotonically linearly increasing with the increase of the data size. Our system took about 24 s to complete the execution of 1,000,000

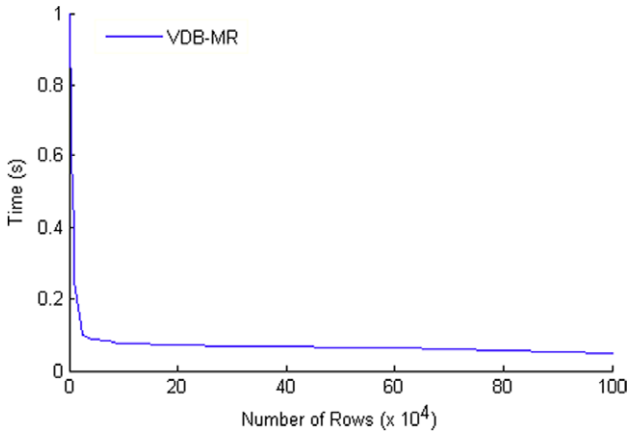


Fig. 9. Execution time of unit data of VDB-MR.

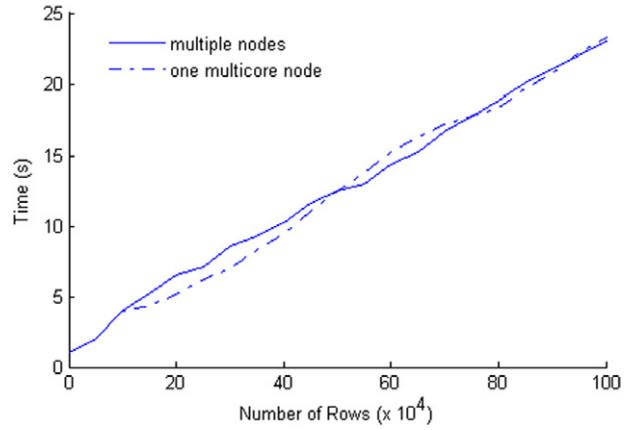


Fig. 10. Execution time of large-scale data on a multi-core node vs. multiple nodes.

rows of data. This result demonstrates that VDB-MR is not only scalable, but also has a very good performance in handling large-scale data.

The execution time of unit data of VDB-MR is presented in Fig. 9. The figure indicates that the execution time of unit data is high when the data size is relatively small, but is basically stable at 0.03 s when data is more than 20,000 rows. So for larger data size (up to 10^6 rows of data), VDB-MR has very stable performance.

4.3.3. Execution time of large-scale data on one multicore node and multiple nodes

We conducted this experiment to compare the execution time of large-scale data on one multicore node vs. a set of multiple nodes.

Fig. 10 shows the execution time on one multicore node and a set of multiple nodes (called multi-node in the rest of the paper). The efficiency of execution on a single node is slightly higher for a data size within 500,000 rows; while multi-node performs better for a data size more than 500,000 rows. This is because different nodes need to interact through the network IO; while the single node interacts simply by a simple file operation. When the data size is less than 500,000 rows, network IO between different nodes takes relatively more time and is therefore less efficient. However, the single node performs relatively poorly on parallel processing when the data size is more than 500,000 rows. But the overall performances of one multicore node and multi-node are similar, which demonstrates that our system has a very good performance on multicore systems.

5. Related work

Many approaches and systems have been proposed and developed to achieve data integration [1–9], having the same or a similar objective: providing flexible and efficient access to information residing in a collection of distributed, heterogeneous and overlapping resources such as databases, plain texts, file systems, distributed file systems, XML files, World Wild Web, and many other information sources.

Virtual database is a type of data integration technique. Much research work on virtual databases has been done to address issues of how a global schema is extracted from autonomous resources, how an effective query language is identified, and how decomposing queries are optimized [12,13]. However, research on query execution has not been given sufficient attention. Our VDB-MR is an approach to address specifically query execution.

In [2,4,24], DBMS and MMDB are imported to work as the query execution module. They store the query result from heterogeneous member resources, and the operations of query, join, merge, and select of the global virtual database and tables are all done by the interfaces provided by the DBMS and MMDB. It is very hard for this system to be tuned to improve query execution performance. Furthermore, this system is not capable of processing large-scale data, and querying from remote resources limits the efficiency of the query execution.

In [14], a query execution plan is generated by decomposing a query to a virtual database or tables. The plan specifies the procedure of the query execution. If two intermediate tables need to be joined together, they need to be migrated to a worker first and then executed with the help of a DBMS. With this solution, parallel execution is possible; some select, join or merge operations can be done concurrently. However, because the table is the minimum unit for these operations, if large tables are involved during the execution of these operations then the execution of these operations will be the bottleneck of the whole process. Compared with this architecture, the minimum unit of our query execution system is a row of a table.

Yale University developed a hybrid system that combines DBMS and MapReduce technologies, called HadoopDb [19]. HadoopDb is comprised of PostgreSQL on each node, Hadoop as a communication layer that coordinates the multiple nodes, and Hive as the translation layer. The system works as a parallel database, where users can interact with the system using a SQL-like language. However the system is based on Hadoop and the data must be stored in the form of key-value; therefore the system cannot handle heterogeneous data resources.

6. Conclusion

The main concept of data integration is to combine data from different resources and provide users with a unified view of these data. Many data-integration solutions have been proposed, and virtual database is one such solution. A virtual database system is basically composed of a set of models that specify the structural characteristics of data sources, views and web services. Many systems based on these models have been developed to deal with various real problems of data integration, such as collecting information from different web sites, integrating spatially-related environmental data, and aggregating information from multiple enterprise systems. However, most of these works address the issues of how a global schema is extracted from autonomous resources, how an effective query language is identified, and how decomposing queries are optimized. Query execution has received little attention.

MapReduce is a software framework introduced by Google that has well-proven success in Google and Hadoop communities. MapReduce has been also successfully applied in the database field, such as Amazon Dynamo and HadoopDb. However, these works mainly focus on isomorphic data resources stored in the form of key-value. To the best of our knowledge, no work has been done to apply MapReduce for data integration of heterogeneous data resources such as database or file systems. This is the objective of this work.

In this paper, we propose a new data integration approach of a virtual database by utilizing the MapReduce technology, named as VDB-MR. VDB-MR translates a query into sub queries by the global schema, executes them and merges the results concurrently using the MapReduce technology. VDB-MR provides many interfaces and therefore can be very flexibly customized to accommodate different experiments with different algorithms. A series of experiments were conducted to evaluate VDB-MR against an open source data integration system OGSA-DAI and two DBMSs in parallel. Experiment results show that VDB-MR significantly outperforms OGSA-DAI and the DBMSs in parallel.

In the future, the emphasis of our research is to optimize the algorithm we used in VDB-MR and further improve the efficiency of the query execution. We are also looking for an effective fault tolerant mechanism in order to improve the robustness of the system.

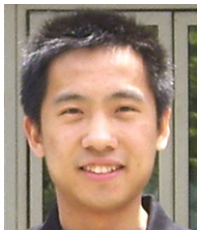
Acknowledgements

This work is supported by Natural Science Foundation of China (60803121, 60773145, 60911130371, 90812001, 60963005), National High-Tech R&D (863) Program of China (2009AA01A130, 2006AA01A101, 2006AA01A108, 2006AA01A111, 2006AA01A117).

References

[1] Ashish Gupta, Venky Harinarayan, Anand Rajaraman, Virtual database technology, ACM SIGMOD Record 26 (4) (1997) 57–61.
[2] Marcel Frehner, Martin Brandli, Virtual database : spatial analysis in a web-based data management system for distributed ecological data, Environment Modelling & Software 21 (July) (2006) 1544–1554.
[3] STS Prasad, Anand Rajaraman, Virtual database technology, XML, and the evolution of the web, IEEE Data Engineering Bulletin 21 (June) (1998) 27–29.
[4] Join Blackham, Peter Grundeman, John Grundy, John Hosking, Rick Mugridge, Supporting pervasive business via virtual database aggregation in: Proceedings of Evovle'2001: Pervasive Busines, May 2001.
[5] Jacob Berlin, Amihai Motro, Autoplex: automated discovery of content for virtual database, in: Proceedings of the 9th International Conference on Cooperative Information Systems, May 27–31, 2002, pp. 108–122.
[6] James S. Fine, Aaron K. Ching, Joe B. Schneider, Darcy Pollum, Michael L. Astion, Biblecard: network-based virtual database for laboratory information, Clinical Chemistry 41 (9) (1995) 1349–1353.
[7] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, Jennifer Widom, The TSIMMIS project: integration of heterogeneous information sources, in: Proceeding of IPSJ Conference, 1994, pp. 7–18.
[8] Walter Sujansky, Heterogeneous database integration in biomedicine, Computers and Biomedical Research 34 (2001) 285–298.
[9] Norman W. Paton, Malcolm P. AtKinson, Vijay Dialani, Dave Pearson, Tony Storey, Paul Watson, Database access and integration services on the grid, UK e-Science Technology Report Series, 2002.
[10] Li Keqiu, Hong Shen, Francis Y.L. Chin, Weishi Zhang, Multimedia object placement for transparent data replication, IEEE Transactions on Parallel and Distributed Systems 18 (2) (2007) 212–224.
[11] Carmela Comito, Gounaris Anastasios, Sakellariou Rizos, Talia Domenico, A service-oriented system for distributed data querying and integration on Grids, Future Generation Computer Systems 25 (5) (2009) 511–524.
[12] Y. Breitbart, Multidatabase interoperability, SIGMOD RECORD 19 (3) (1990).
[13] Witold Litwin, Leo Mark, Nick Roussopoulos, Interoperability of multiple autonomous databases, ACM Computing Surveys 22 (3) (1990).
[14] Hongjun Lu, Beng-Chin Ooi, Cheng-Hian Goh, Multidatabase query optimization: issues and solutions, in: Proceeding RIDE-IMS'93, 1993, pp. 137–143.
[15] Jeffrey Dean, Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, December 06–08, 2004.
[16] Ralf Lammel, Google's MapReduce programming model-revisited, Science of Computer Programming 70 (1) (2008) 1–30.
[17] Apache Hadoop, http://hadoop.apache.org.

[18] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, Werner, VogelsDynamo: Amazon's Highly Available Key-value Store, SOSP'07, October 14–17, 2007.
[19] Azza Abouzeid, Kamil BajdaPawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin, HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, VLDB'09, August 24–28.
[20] Wenhao Xu, Jing Li, Yongwei Wu, Xiaomeng Huang, Guangwen Yang, VDM: virtual database management for distributed databases and file systems, Grid and Cooperative Computing (2008).
[21] Jim Melton, SQL language summary, ACM Computing Survey 28 (1) (1996).
[22] David J. Dewitt, Randy H. Katz, Frank Olken, Leonard D. Shapiro, Michael R. Stonebraker, David Wood, Implementation techniques for main memory database systems, in: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, 1984, pp. 1–8.
[23] Open grid services architecture — data access and integration, http://www.ogsadai.org.uk.
[24] Ulla Merz, Roger King, DIRECT: a query facility for multiple databases, ACM Transaction on Information Systems 12 (4) (1994) 339–359.

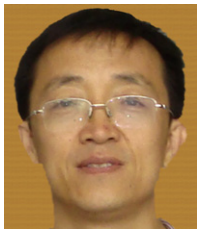


Yulai Yuan received his B.S. degree in Computer Science and Engineering from the Beijing Information Technology Institute, Beijing in 2005. He is currently pursuing his Ph.D. degree in Computer Science at Tsinghua University. His research interests include distributed systems, performance modeling and prediction.

Yongwei Wu received his Ph.D. degree in Applied Mathematics from the Chinese Academy of Sciences in 2002. Since then, he has worked at the Department of Computer Science and Technology, Tsinghua University, as research Assistant Professor from 2002 to 2005, and Associate Professor since 2005. His research interests include grid and cloud computing, distributed processing, and parallel computing.

Xiao Feng is a postgraduate student in Computer Science and Technology Department, Tsinghua University. His research interests include data integration, distributed database and distributed computing systems.

Jing Li received his master's degree from the Computer Science and Technology Department, Tsinghua University in 2009. His research interests include grid computing, distributed processing, and parallel computing.



Guangwen Yang is Professor of Computer Science and Technology, Tsinghua University, China. He is also an expert on "high-performance computer and its core software", the National "863" Program of China. He is mainly engaged in the research of grid computing, parallel and distributed processing and algorithm design and analysis.

Weimin Zheng is a Professor of Computer Science and technology, Tsinghua University, China. He got his B.S. and M.S. degrees from the Department of Automatics, Tsinghua University in 1970 and 1982 respectively. He is currently the research director of the Institute of High Performance Computing, Tsinghua University, and managing director of the Chinese Computer Society. His research interests include computer architecture, operating systems, storage networks, and distributed computing.