A Distributed Query Execution Engine in a Grid Environment

Gustavo G. Trevisol, Cristiano Biancardi, Alvaro C. P. Barbosa,
José G. Pereira Filho, Ramon G. Costa, Evellin S. Cardoso
Federal University of Espirito Santo
Computer Science Department
Av. Fernando Ferrari, 514 - 29060-970 Vitoria, ES - Brazil
(gtrevisol, cbiancardi, alvaro, zegonc, ramoncosta, evellinc)@inf.ufes.br

Abstract

Grid is a computational environment in which applications can use multiple distributed computational resources in a safe, coordinated, efficient and transparent way. Data Integration Middleware Systems (DIMS) are originally distributed systems that can make use of Grid environments to obtain a better performance and a rational use of available resources. This work describes a Distributed Query Execution Engine (DQEE) inserted in a Grid environment for executing sub-queries and operators in a distributed and parallel way. We present an approach to obtain an efficient distributed query execution with a reduced response time by the use of the DQEE. [1]

1. Introduction

A common problem faced by many organizations today is the existence of multiple, large, distributed, heterogeneous and autonomous data sources [24]. In this scenario, Data Integration Middleware Systems (DIMS) became important tools to provide integrated and transparent access to these data. Despite their importance, developing DIMS is not a trivial task due to the complexity of supporting diverse data models, to the different software and hardware environments and to the complex query processing strategies [18, 19].

Query Query processing is a major challenge in DIMS development. The query processing service in a DIMS is achieved by the Query Execution Engine (QEE) [4]. When a query is submitted to a DIMS, a Query Execution Plan (QEP) is generated to be executed by the QEE. A QEP is represented by a tree data structure where the leaf nodes contain the subqueries to access specific data sources and the internal nodes contain the operators (e.g. natural join) [24]. In order to improve the query processing service the use of a distributed environment presents itself as a promising alternative. Despite the available infrastructure, there is no definitive and complete solution for the distributed query processing problem [20]. This scenario demands efficient mechanisms and technologies for QEE development, which must be redesigned to incorporate facilities in order to provide new capabilities to distributed computation.

Nowadays, Grid computing technology has being disseminated and used in all sorts of applications domains. Grid computing provides transparent access to distributed computing resources such as processing capabilities and storage capacity. Grid users essentially see a single, large virtual computer despite the fact that the pool of resources can be geographically distributed and connected to worldwide networks [15]. Therefore, a Grid computing environment presents a natural option to be incorporated in DIMSs QEE, considering: *i)* the distributed nature of data sources; *ii)* the amount of data involved in some applications; *iii)* the existence of an accentuated demand for computational resources both for translating data models and for the process of integration of the data in these data sources.

We are currently researching on Grid computing solutions for developing a Distributed Query Execution Engine (DQEE) [28] for a DIMS. In this paper we show the use of a Grid environment for executing the QEPs using the DQEE in order to execute sub-queries and operators in a distributed and parallel way. We present an approach to obtain an efficient QEP execution with reduced time by the use of DQEE.

The remainder of this paper is structured as follows: section 2 presents background and related works; section 3 overviews the architecture of CoDIMS, our middleware for data integration; section 4 presents the stages involved in DQEE; section 5 describes the implementation environment used in our tests and evaluate some results. Section 6 concludes the work presenting future research directions.

[1]This work has been partially funded by the FAPES (Fundacao de Apoio a Ciencia e Tecnologia do Espirito Santo) - Espirito Santo's Scientific and Technological Research Funding Agency.

2. Background and related works

Our research in data integration systems date back to 1998 when we started investigating techniques for building flexible systems in the Ecohood project [23]. That experience was the start point to later develop the CoDIMS, a Configurable Data Integration Middleware System [6]. The need to improve query processing strategies led us to incorporate Grid technology to our DIMS. As a result we have proposed an instance of CoDIMS named CoDIMS-G [13, 14]. CoDIMS-G uses a Grid environment for parallel programs execution over an unique and large database. The database is divided into parts and each sub-query accesses one of these parts. The sub-queries are executed in a Grid environment, but the final composition is done in a centralized way.

The problem of integrating distributed query processing technology with Grid services has been addressed in the OGSA-DQP project [1]. In this project, the query processor uses a set of orchestrated services that manages distributed data access, resources metadata information and service instances creation and management. The DQP service is built on top of a OGSA-DAI [3] implementation that provides services interfaces for data source access. Our ongoing project differs from the OGSA-DQP in the following aspects: *i)* In our case, Grid node selection is based on historical of application execution, adaptive query execution strategy, and real time node allocation using a dynamic distributed query optimization strategy, [14]; *ii)* Moreover, our internal metadata is defined by means of ontologies; *iii)* We allow different global data models and user-query languages; *iv)* We enable the generation of lightweight DIMS tailored for a specific application; *v)* We make possible to dynamically send operators and wrappers for executing in Grid nodes.

In comparison to other systems that use distributed query processing, the ones which are most related to our work are ParGRES [22] and MOCHA [21].

ParGRES proposes a solution for executing parallel joins in a cluster using PostgreSQL database. However, it does not implement schema integration since a single database is used. Additonally, in this system, no query optimization is used. When a SQL query is submitted, it is divided into sub-queries and sent to the Grid nodes. The result of each sub-query execution is returned to a specific component that executes join operations to compose the final result.

In MOCHA, the sub-queries are executed by the DAPs (Data Access Providers). DAPs run at the data source site or in close proximity to it. MOCHA does not support DAP reallocation in the execution time. The results generated by the sub-queries are joined in an unique node by the Query Processing Coordinator.

Our work differs from these two projects aforementioned by allowing parallel execution of all QEP: sub-queries and join operations are executed in a distributed way using Grid.

3. Architecture

3.1 CoDIMS overview

CoDIMS (Configurable Data Integration Middleware System) [6] is a flexible and configurable environment which allows the generation of lightweight [7] systems for integration of heterogeneous and distributed data, configured for specific applications. The architecture is based on frameworks [5] and components techniques. The components are based on Database Management Systems services [12] with plug and play features, supplying a high level of interoperability, composition and abstraction. The systems therein produced are considered to be instances of CoDIMS.

The CoDIMS metadata is defined using ontologies [26] with benefits to: improve the mapping between data sources; translate data sources models to a single global data model; facilitate the process of integration of the data in the data sources. When a global query is submitted to CoDIMS, a global-to-local mapping is done, defining the sub-queries to be sent to each data source according to an optimized query execution plan (QEP). The query execution engine (QEE) [25] manages the execution of QEP sending the sub-queries to Data Access Component (DAC) to access the data sources through the wrappers. The sub-results returned from each data source are converted by the wrappers [11] to the global model and sent to the QEE. The QEE executes the necessary operations on sub-results to compose the final result (Figure 1).

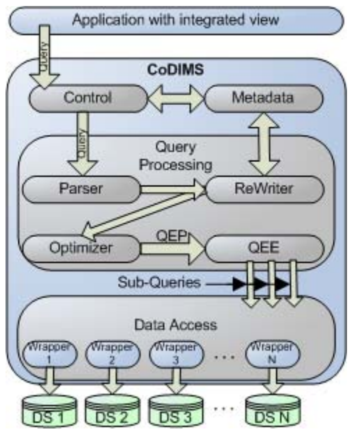


Figure 1. The CoDIMS and its main components

3.2 Distributed query execution engine

The current QEE inserted into CoDIMS [25, 10] is based on the framework QEEF [4] and was implemented as a framework component. The stable parts (frozen spots) must exist in all instances of the QEE while some specific parts (hot spots) are instantiated according to the applications requirements.

The stable parts are the interface and the processor of the QEE, the basic structure of the operators and the result set. The parts to be instantiated are composed by the algebraic operators and by the result set of each data model.

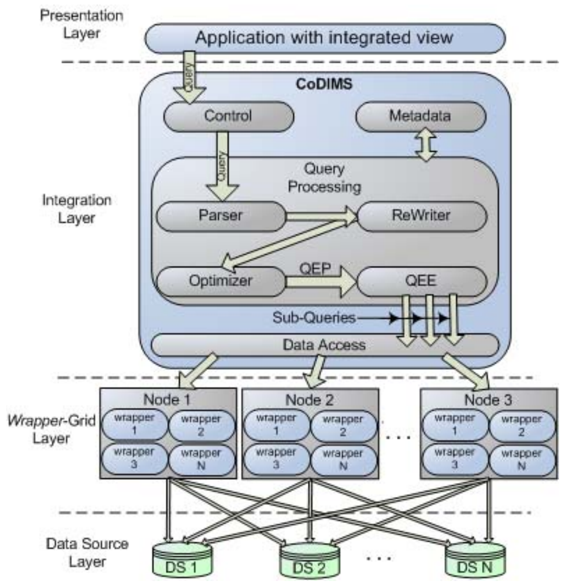


Figure 2. The Wrapper-Grid layer [8]

In cases in that there are many data sources and large ones to be integrated, the use of a Grid environment presents itself as a promising alternative. The works presented in [8, 10, 9] propose a Wrapper-Grid layer to allow the execution of sub-queries in a distributed and parallel way. However, despite the sub-results available in the Grid nodes, the composition is done in a centralized way (Figure 2). As solution for this limitation we developed a Distributed Query Execution Engine (DQEE) (Figure 3). Using DQEE it is possible: *i)* to distribute the processing of all QEP, executing sub-queries and operators in a distributed and parallel way, by using the algebraic operators that are executed remotely in the Grid nodes; *ii)* to make available the result produced by the execution of one operator to be used by other operators, accordingly to the tree hierarchy described by the QEP; *iii)* to (re)allocate instances of the operators in Grid nodes, thus, optimizing the use of computational resources available and decreasing the query processing time.

The Wrapper-Grid layer was extended to support the needs of DQEE. The new layer, named QEE-Grid (QEEG), supplies some Grid services through its QEE-G component (QEE-GC), in order to: *i)* allow communication among the Grid nodes; *ii)* transfer the sub-results among the Grid nodes; *iii)* execute operations in a distributed and parallel way. The QEE-GC is installed in each Grid node and contains the wrappers and algebraic operators. In addition, a new component, named Access Grid (AGC), was incorporated to CoDIMS architecture. Its main purpose is make the requirements needed for execution of Grid services transparent to the integration layer. Figure 3 presents the new architecture of CoDIMS.

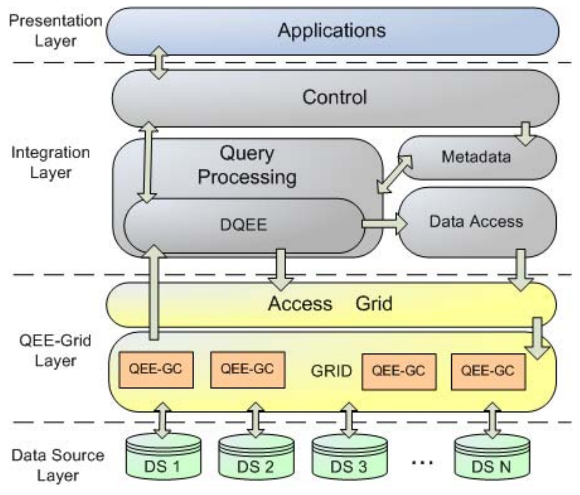


Figure 3. The CoDIMS new architecture.

4 Distributed query processing

4.1 Plan query execution

A QEP is represented by a tree in which the nodes are operators, the leaves are the data sources and the arcs are the relationship between the producer and the consumer operators. Therefore, each operator of a QEP plays the role of a data producer and/or data consumer. Figure 4 shows a QEP containing operators (algebraic and control) related in a producer/consumer form where the arrows indicate the data flow direction between the producer and consumer.

4.2 Paralleling sub-queries

In CoDIMS, the QEP generated by the Optimizer [17] is implemented as a query tree. Each leaf is directed to a specific data source. The ability of CoDIMS for integrating heterogeneous and distributed data is partly due to the

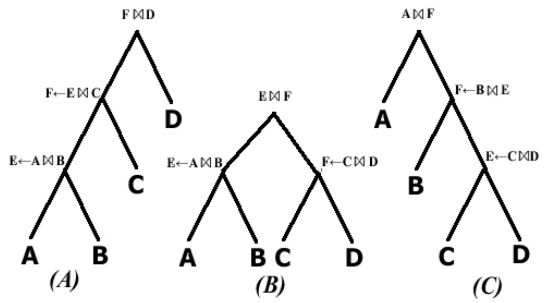


Figure 4. Topology of the QEP: linear at left, ramified, linear at right [4]

behavior of the Data Access Component (DAC). It accesses different local data sources through the wrappers. Therefore, each sub-query must be sent to the DAC by DQEE.

Each leaf of the QEP calls asynchronously the DAC, sending information referred to the sub-queries to be executed. The information are: query identifier, sub-query identifier and the sub-query itself (coded in XML and represented according to a standard previously established in [25]).

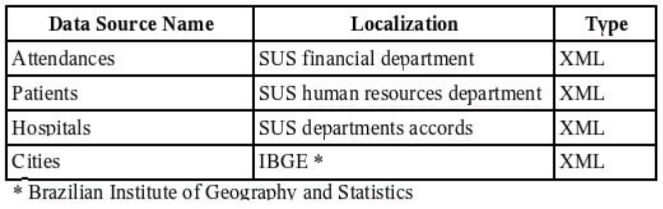
To proceed with the execution, the DAC receives a request, identifies the responsible data source for the sub-query and sends a solicitation to AGC. The AGC determines the most suitable node for the sub-query execution. After that, the AGC sends the information to the QEE-GC located in the selected node. The QEE-GC starts the execution of the wrapper responsible for the sub-query. The wrapper is responsible for the following tasks: *i)* translates the sub-query into the native data source model; *ii)* set up a communication with the data source; *iii)* receives a sub-result and translates it into the CoDIMS global data model; *iv)* finally, the wrappers store the result set in the Grid node and return asynchronously to the DQEE the information about the result set generated (name file, location and size).

4.3 Parallel Processing of Sub-results

When the DQEE is notified about the production of the sub-results to be joined it sends to AGC the following information (based on the QEP): query identifier, operator identifier, operator name, sub-results name and location. Then, the AGC determines the most appropriate node for the operator execution and sends the information received to the QEE-GC of the selected node. The QEE-GC starts the operator execution that uses the information about location of the sub-results to be joined. A new sub-result is generated and it is stored in the Grid node. The QEE-GC returns to DQEE the information about the result set produced (file name, location and size). The QEP is processed until the QEP root is achieved and the information of the final result is returned to the respective CoDIMS user application.

5 Evaluation of the Results

To evaluate the results of the DQEE we created a case study based on a telemedicine research project being carried out in our Labs, named TeleCardio [2]. In this project, a telehomecare system for remote monitoring of patients with cardiological syndromes is being developed. In cases in which patients are in locations far from the hospitals, the TeleCardio system can monitor their health state information, by periodically sending this information to an hospital or another health center. In this application it is necessary to know about medical attendances in patients who live in cities distant from the hospital. To be able to process this query, it is necessary to access distributed data sources. Figure 5 depicts the location and types of data sources involved in typical query in this system.

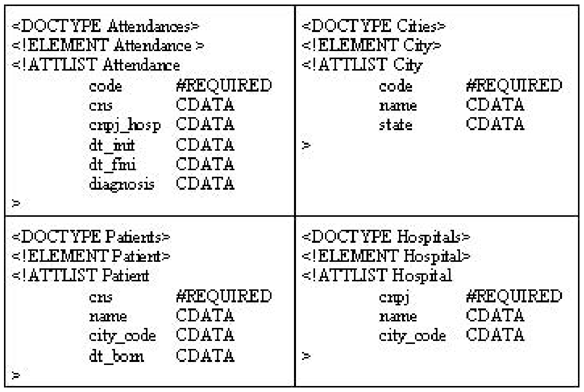


Data Source Name	Localization	Type
Attendances	SUS financial department	XML
Patients	SUS human resources department	XML
Hospitals	SUS departments accords	XML
Cities	IBGE *	XML

* Brazilian Institute of Geography and Statistics

Figure 5. Data sources involved in the query.

Figure 6 depicts the DTDs of data sources Attendance, Cities, Patients and Hospitals.



```
<DOCTYPE Attendances>
<!ELEMENT Attendance >
<!ATTLIST Attendance
        code        #REQUIRED
        cns         CDATA
        cnpj_hosp   CDATA
        dt_init     CDATA
        dt_fini     CDATA
        diagnosis   CDATA
>
```

```
<DOCTYPE Cities>
<!ELEMENT City>
<!ATTLIST City
        code        #REQUIRED
        name        CDATA
        state       CDATA
>
```

```
<DOCTYPE Patients>
<!ELEMENT Patient>
<!ATTLIST Patient
        cns         #REQUIRED
        name        CDATA
        city_code   CDATA
        dt_born     CDATA
>
```

```
<DOCTYPE Hospitals>
<!ELEMENT Hospital>
<!ATTLIST Hospital
        cnpj        #REQUIRED
        name        CDATA
        city_code   CDATA
>
```

Figure 6. DTDs of the data sources Attendance, Patients, Hospitals and Cities.

5.1 Configuration of the Test Environment

In order to execute the queries and verify the results, three instances of CoDIMS have been generated as described in section 5.2 We have used the relational data model as an integrator (global) model. For this reason, it has been necessary to translate data source native models (XML) into to the global model (Relational), defined accordingly to the Figures 7, 8, 9 and 10.

Attendance		
Attribute	**Type**	**Modifier**
code	Varchar(10)	Primary Key
cns	Number(20)	Foreign Key (Patient)
cnpj_hosp	Number(20)	Foreign Key (Hospital)
dt_init	Date	Null
dt_fini	Date	Null
diagnosis	Varchar(500)	Null

Figure 7. Global Schema. Table of Attendance.

Cities		
Attribute	**Type**	**Modifier**
code	Varchar(10)	Primary Key
name	Varchar(100)	Not null
state	Varchar(2)	Not null

Figure 8. Global Schema. Table of Cities.

Patients		
Attribute	**Type**	**Modifier**
cns	Number(20)	Primary Key
nome	Varchar(50)	Not null
city_code	Varchar(10)	Foreign Key (City)
dt_born	Date	Null

Figure 9. Global Schema. Table of Patients.

After configuring the global metadata, the SQL query depicted in Figure 11 has been submitted to CoDIMS producing the QEP shown in Figure 12. The index in operators identifies its execution order in the QEP. The QEE then interprets the received QEP and generates a sub-query to each data source involved in the query (Figure 13). Since the data model is relational the sub-queries have been generated in the relational format, and encapsulated in XML according to a standard defined in [25].

Hospitals		
Attribute	**Type**	**Modifier**
cnpj	Number(20)	Primary Key
name	Varchar(100)	Not null
city_code	Varchar(10)	Foreign Key (City)

Figure 10. Global Schema. Table of Hospitals.

```
select patient.name, patient.cns, attendance.diagnosis,
        attendance.dt_init, attendance.dt_fini,
        hospital.name, city.name, city.state
from patient, attendance, hospital, city
where hospital.cnpj = attendance.cnpj_hosp and
       patient.city_code = city.code and
       patient.cns = attendance.cns and
       patient.city_code <> hospital.city_code
```

Figure 11. Query sent to the CoDIMS in SQL language.

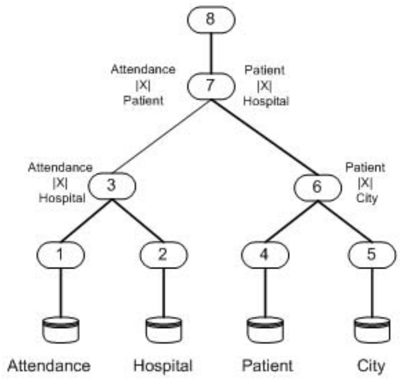


Figure 12. QEP of a query in a tree form.

```
<local-plan>
   <column order="0" name="cns" table="Attendance" />
   <column order="1" name="cnpj_hosp" table="Attendance" />
   <column order="2" name="dt_init" table="Attendance" />
   <column order="3" name="dt_fini" table="Attendance" />
   <column order="4" name="diagnosis" table="Attendance" />
   <table name="Attendance" />
</local-plan>
```

Figure 13. Example of a sub-query to be sent to the Attendance data source.

5.2 Test Scenarios

In order to compare the possible forms of query execution, three test scenarios have been created:

Scenario 1 - Distributed sub-queries processing: The sub-queries are sent and executed sequentially in a synchronous way and the sub-results composition is achieved in a centralized way by the QEE, implemented in [25]. The QEE receives the QEP and starts sub-queries processing sequentially. At the end of all sub-queries processing, the QEE transfers each sub-result file to its server and combines them in a centralized way.

Scenario 2 - Using the Wrapper-Grid layer [8]: The sub-queries processing is achieved in a distributed and parallel way, using the Grid environment. However, like in Scenario 1, the sub-results composition is achieved in a centralized way.

The sub-queries are sent by QEE to the Grid environment asynchronously. One thread is created for each sub-query to be executed in a Grid environment. After the sub-queries are executed, the QEE transfers each sub-result to its server and combines them in a centralized way.

Scenario 3 - Using DQEE, Access Grid Component, QEE-Grid layer and the QEE-GC. The process of sending and executing sub-queries is the same as the Scenario 2. To compose the final result, the DQEE, using the QEE-Grid layer, combines the sub-results in a distributed and parallel way in a Grid environment.

The DQEE starts the processing of sub-queries sending them asynchronously to the AGC. It sends each sub-query to be processed in a Grid node. According to the QEP, when two sub-results to be combined are available, the DQEE starts a join operation execution in a Grid node. The DQEE submits a request to the AGC which determines a proper node for operation execution based on the location and size of the sub-results. For executing the operation 3 (Attendance |X| Hospital), since the size of Hospital sub-result is smaller, the AGC selects the node that has the Attendance sub-result to execute the join operation. This choice aims at decreasing the sub-results transfer time from one node to another. The operation execution order 6 (Patient |X| City) is executed in the Patients sub-result node because the Cities sub-result is smaller.

The DQEE, after starting the processing of operations 3 and 6, continues the QEP execution waiting for the generation of the two sub-results by the algebraic operators in the Grid nodes. When the operations 3 and 6 finish, the DQEE is notified about it and the DQEE tells the AGC to start the operation 7 (Attendance |X| Patient and Patient |X| Hospital) in a Grid node. Finally, the DQEE returns to the client application the location of the final result.

5.3 Results

The approach for the tests execution was the following: using data sources with the same quantity of registers, the query was executed many times for each scenario and the average time was calculated. Then, the amount of registers in each data source was increased and the tests were repeated in the same way.

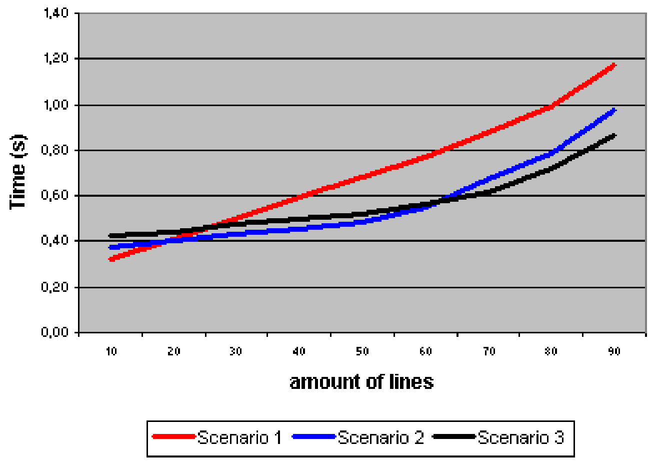


Figure 14. Graphic for comparing the three approaches with a few quantity of registers. ($<$ 100).

Figure 14 shows that for a small data set (up to 20 registers) to be processed, the approach used in Scenario 1 is more efficient; When the number of registers is between 21 and up to 65 registers, Scenario 2 presents better results; when the number of registers is bigger than 65, Scenario 3 using the DQEE is much more efficient. The reason for this is that with a few number of registers, the centralized composition makes up the overhead caused by the creation of threads and communication with Grid environment in other scenarios. In Scenario 3, in which there are many threads to be managed (sub-queries execution plus sub-result composition) as well and many communications with the Grid nodes, the benefits of using a Grid environment become evident with more than 65 registers. Moreover, it is possible to notice that from 30 registers on, the execution time of the Scenario 1 is higher than Scenario 3.

On the graphic of Figure 15, as we can notice by increasing the amount of registers, the average time for the execution of the queries is increased in an exponential way, despite the different factors in each scenario. However, as the amount of registers increases, the execution time in Scenario 1 becomes bigger than in the others two scenarios. In scenario 3, in contrast, due to the use of the DQEE, the time

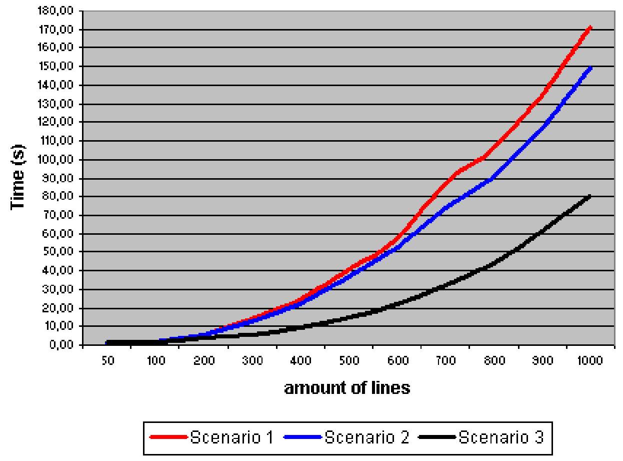


Figure 15. Graphic for comparing the three approaches with a large number of registers. (> 100).

increase is not so accentuated, being more advantageous its utilization when it is necessary to achieve the integration of a high quantity of registers.

6 Conclusion

Data Integration Middleware Systems became an important tool to provide integrated and transparent access to information stored in multiple, large, distributed, heterogeneous and autonomous data sources. In order to investigate new technologies and processes to improve data integration solutions, we have proposed CoDIMS. CoDIMS is an ongoing project at the Multimedia Networking Research Laboratory at Federal University of Espirito Santo (Brazil). CoDIMS is a Web Services-based [27] middleware running in a Linux operating system and implemented using Java. Recently we have started investigating how to improve query processing strategies incorporating Grid technology. As a result, we have proposed an instance of our DIMS named CoDIMS-G.

In this paper, we present an approach for distributed query processing to be incorporated to CoDIMS through the implementation of a Distributed Query Execution Engine (DQEE). The DQEE allows its operators to be executed in a parallel and distributed way in a Grid environment. The use of grid computing, instead of other distributed computing paradigm, let us to benefit from its inherent technologies ands features, such as fault tolerance, optimization and sharing of available resources, dynamic service allocation etc. The Grid environment considered in this work is the Globus Toolkit 3.2 [16].

Partial results showed that: for applications with small data sources, the use of centralized scenario is more efficient; for a small increasing in the data source size, a partially distributed scenario is more efficient; and for larger data sources, according to the results of our tests, we have that the new approach of query execution presented in this work, i.e., a fully distributed scenario, becomes much more efficient. In conclusion, the larger the data sources, the more significant are the difference between the approach proposed here and other scenarios, concerning query execution time. As a result of our work, we conclude that the performance of the data integration systems when distributed query processing is considered can be highly improved by the use of a Grid environment.

The main contributions of this work are: *i)* the development of a DQEE to be incorporated in a DIMS (in particular, in the CoDIMS) using a Grid Environment; *ii)* the proposal of an approach for distributed and parallel sub-queries execution and operators composition; *iii)* the exemplification of the use of Grid a environment to access, in a secure, trusty and transparent way the idle resources in distributed machines used in the execution of distributed queries; *iv)* the application of this work in the Telecardio project.

At the moment we are working on: *i)* the definition and implementation of new scenarios for testing the join operator in different data models of data sources with different sizes, in order to evaluate the performance of CoDIMS in these situations; *ii)* the development of a dynamic optimizer for distributed queries; *iii)* the development of a dynamic scheduler to assist the process of determining the most suitable node for sub-queries and operations executions.

As future work, we intend to elaborate a mechanism for security access to grid Nodes, using the digital certificates contained in the Globus Toolkit distribution package.

7 Acknowledgment

The authors are thankful to Giancarlo Guizzardi for his assistance in proofreading the final version of this article.

References

[1] M. N. Alpdemir, A. Gounaris, A. Mukherjee, D. Fitzgerald, N. W. Paton, P. Watson, R. Sakellariou, A. A. A. Fernandes, and J. Smith. Experience on performance evaluation with ogsa-dqp. *International Summer School on Grid Computing 2006. Ischia, Italy*, July 2006.

[2] R. Andreao, J. G. P. Filho, and C. Z. Calvi. Telecardio: Telecardiologia a servico de pacientes hospitalizados em domicilio. in portuguese. In *XX Congresso da Sociedade Brasileira de Informatica em Saude*, Florianopolis, SC, Brazil, 2006.

[3] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. C. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. W. Paton, D. Pearson, T. Sugde, P. Watson, and M. Westhead. The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience*, 17:357–376, 2005.

[4] F. V. M. Ayres. *QEEF: Uma Maquina de Execucao de Consultas Extensivel. In Portuguese*. PhD thesis, PUC-Rio, Rio de Janeiro, RJ, Brazil, Dec. 2003.

[5] A. C. P. Barbosa. *Middleware para Integracao de Dados Heterogeneos Baseado em Composicao de Frameworks. In Portuguese*. PhD thesis, PUC-Rio, Rio de Janeiro, RJ, Brazil, May 2001. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[6] A. C. P. Barbosa, F. A. M. Porto, and R. N. Melo. Configurable data integration middleware system. *Brazilian Computing Society. Campinas, SP, Brazil*, 8:12–19, 2002.

[7] D. S. Batory and J. Thomas. P2: A lightweight dbms generator. Technical report, University of Texas, Aug. 1995.

[8] C. Biancardi. Distribuicao e execucao de wrappers em ambiente de grid para o codims. in portuguese. Master's thesis, UFES, Vitoria, ES, Brazil, 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[9] C. Biancardi, L. J. Silvestre, and A. C. P. Barbosa. Uma abordagem de grid para integracao de dados. in portuguese. *XXVI Iberian Latin-American Congress on Computational Methods in Engineering. Vitoria, ES, Brazil*, 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[10] C. Biancardi, L. J. Silvestre, and A. C. P. Barbosa. Uma proposta para distribuicao e execucao de wrappers em um ambiente de grid para o codims. in portuguese. *3rd Workshop on Computational Grids and Applications. Petropolis, RJ, Brazil*, pages 183–236, June 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[11] T. Coco. Implementando wrappers xml e relacional para o codims. in portuguese. Monograph - UFES, 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[12] C. J. Date. *Introduction to Database Systems*. Addison-Wesley, 8th edition, 2004.

[13] V. Fontes, M. Dutra, F. A. M. Porto, B. Schulze, and A. C. P. Barbosa. Codims-g: a data and program integration service for the grid. *2nd Workshop on Middleware for Grid Computing. Toronto, Ontario, Canada*, 76:29–34, 2004.

[14] V. Fontes, M. Dutra, F. A. M. Porto, B. Schulze, and A. C. P. Barbosa. An adaptive parallel query processing middleware for the grid. *Concurrency and Computation: Practice and Experience. Chichester, UK*, 18:621–634, May 2006.

[15] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed system integration. *Open Grid Service Infrastructure WG, Global Grid Forum. Argonne, IL, USA*, pages 183–236, June 2002.

[16] Globus. The globus toolkit. Technical report, University of Chicago, 2004. Available in: http://www.globus.org.

[17] F. R. Gomes. Um otimizador de consulta para o codims. in portuguese. Monograph - UFES, 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[18] F. Hakimpour and A. Geppert. Resolving semantic heterogeneity in schema integration: an ontology based approach. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 297–308, Maine, USA, 2001.

[19] A. Y. Halevy. Data integration: A status report. In *Proceedings of 10th Conference on Database Systems for Business Technology and the Web*, Heppenhiem, Germany, 2003.

[20] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32:422–469, 2000.

[21] M. R. Martinez and N. Roussopoulos. Mocha: A self-extensible database middleware system for distributed data sources. In *ACM SIGMOD International Conference on Management of Data*, pages 213–224, Texas, USA, 2000. ACM Press.

[22] M. Mattoso, G. Zimbro, A. A. B. Lima, F. Baio, V. P. Braganholo, A. A. Aveleda, B. Miranda, B. K. Almentero, and M. N. Costa. Middleware para processamento paralelo de consultas olap em clusters de banco de dados. in portuguese. In *Simposio Brasileiro de Banco de Dados*, Uberlandia, MG, Brazil, 2005. ACM Press.

[23] R. Melo, F. Porto, F. Lima, and A. C. P. Barbosa. Ecohood: Constructing configured dbmss based on frameworks. *XIII Simposio Brasileiro de Banco de Dados. Maringa, PR, Brazil*, 1998.

[24] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Inc., New Jersey, USA, 2nd edition, 1999.

[25] F. S. Pinheiro. Incorporando uma maquina de execucao de consultas ao codims. in portuguese. Monograph - UFES, 2004. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[26] L. J. Silvestre. Uma abordagem baseada em ontologias para a gerencia de metadados do codims. in portuguese. Master's thesis, UFES, Vitoria, ES, Brazil, 2005. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[27] G. G. Trevisol. Codims: Incorporando nova abordagem na comunicacao entre seus componentes. in portuguese. Monograph - UFES, 2004. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.

[28] G. G. Trevisol and A. C. P. Barbosa. Uma proposta para execucao distribuida de consultas em um ambiente de grid para o codims. in portuguese. *IV Workshop on Computational Grids and Applications. Curitiba, PR, Brazil*, 2006. Available in: http://codims.lprm.inf.ufes.br/publicacoes.html.