Grid Computing and Scientific Visualization

Gilson [a] Giraldi[1], Fabio Porto[2], Vinicius Fontes[13], M L Dutra[1], Bruno Schulze[1]
[1] Laboratório Nacional de Conputação Científica – LNCC
Zip Code 25651-075 – Petrópolis – RJ – Brazil
[2] Swiss Federal Institute of Technology Lausanne
Database Lab, Switzerland
[3] Pontifícia Universidade Católica do Rio de Janeiro
Computer Science Departament

{gilson,vfontes,emldutra,schulze}@lncc.br, fabio.porto@epfl.ch

Abstract. *In this paper we focus on distributed scientific visualization using a grid environment. More specifically, we are interested on basic requirements for distributed graphics applications on such environments. We propose a middleware infrastructure for supporting scientific visualization applications that meets these requirements. We claim that we should consider scientific visualization in grids from an integrated global view of data and programs published by heterogeneous and distributed data sources. This idea can be implemented by CoDIMS which is an environment for the generation of Configurable Data integration Middleware Systems.*

Resumo. *Neste trabalho enfocamos a visualização científica distribuída em ambiente de grade; mais especificamente, em alguns requisitos para aplicações gráficas distribuídas neste ambiente. Propomos uma infraestrutura de middleware para suportar aplicações de visualizalização científica que satisfazem a estes requisitos. Consideramos que aplicações visualizalização científica devem ser tratadas a partir de uma visão global integrada de dados e programas disponibilizads por fontes de dados distribuídas e heterogêneas. Esta idéia pode ser implementada pelo CoDIMS, que é um ambiente para a criação de sistemas de middlewares configuráveis para a integração de dados.*

1. Introduction

Grid Computing provides transparent access to distributed computing resources such as processing, network bandwidth and storage capacity. Grid user essentially sees a single, large virtual computer despite of the fact that the pool of resources can be geographically-distributed and connected over world-wide networks [Foster 2002]. At its core, Grid Computing is based on an open set of standards and protocols (i.e., Open Grid Services Architecture: OGSA [Foster 2002]) that enable communication across heterogeneous, geographically dispersed environments. The grid environment is open in the sense that not only the systems are open but the services are open as well. To support resource and services encapsulation, openness and privacy management, it is necessary a layer between the operating system and the applications that provides open distributed processing support, allowing applications development, usage and maintenance. Such layer, called Middleware, is a shell over the operating system, adding new functionalities to support the distribution of applications [Berman et al. 2002].

In this paper we are interested on grid computing solutions for scientific visualization applications [Shalf and Bethel 2003]. Traditional solution found in the literature can be cast in MPI-Based, Client-Server architectures and Solutions for Bandwidth Limitations. More recently, researchers have realized that distributed visualization in grid needs a middleware infrastructure [Giraldi et al. 2004]. The Grid Visualization Kernel GVK [Kranzlmuller et al. 2003], which is a grid middleware extension built on top of the Globus Toolkit, is such a proposal. The main goal of the GVK is to provide a middleware layer extension for visualization in grid applications. This identifies GVK as an extension to the existing grid middleware approaches instead of another stand-alone distributed visualization system. The basic philosophy is to provide GVK interfaces as extensions of the commonly available visualization toolkits (OpenDX, VTK, etc). Thus, existing visualization functionalities may be reused. While features such as encryption and authentication are sufficiently covered by existing middleware approaches, specific requirements of visualization, such as multiplexing, buffering, and synchronization, must be provided within GVK.

Despite of its capabilities, GVK does not addresses important issues for visualization applications distributed in grid environments. For instance, the performance of a grid resource (nodes, network links, etc.) may change due to internal problems or extra works. Even more basic, we need to predict performance before launch components. Also, out-of-core capabilities may be added sometimes.

In this paper we focus on to provide an infrastructure for the development of scientific visualization applications in grid that addresses these requirements. We claim that we should consider scientific visualization in grid from an integrated global view of data and programs published by heterogeneous and distributed data sources. We propose a solution based on CoDIMS (COnfigured Data Integration Middleware System) [Barbosa and Porto 2002] which is a middleware environment for the generation of adaptable and configurable middleware systems. With CoDIMS we aim at providing an architecture that can be adapted to new data integration application requirements, so that light weight middleware systems can be generated that conform to the requirements of the applications. We exemplify our proposal with the CoDIMS-G [Silva et al. 2004], a middleware system generated for the grid environment that supports the execution of scientific visualization applications.

2. CoDIMS-G Architecture

In this Section, we present the CoDIMS-G architecture, depicted in Figure 1. CoDIMS-G is a configured data and program integration system generated by the CoDIMS configuration.

The Control orchestrates the communication among the components. Users access the service through a Web application running in a public accessible gatekeeper host. User's requests are forwarded to the Control component, which sends them to the Parser Component. The Parser transforms the user's requests in a query graph representation.

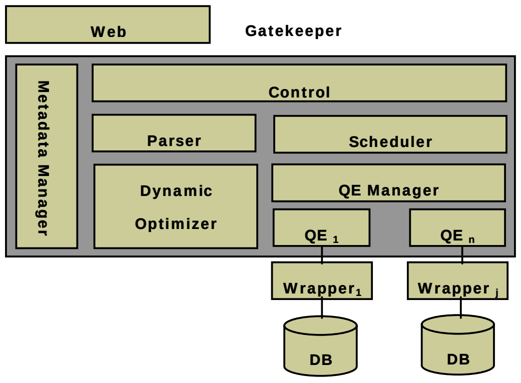

Figure 1. A CoDIMS-G Architecture

The Query Optimizer (QO) receives the graph and generates a physical distributed query execution plan (DQEP), using a cost model based on data and programs statistics stored in the Metadata Manager (MM). For DQEPs that include operations for the evaluation of parallelizable user programs (i.e. programs that evaluate on a tuple by tuple basis), the optimizer calls the Scheduler (SC) component. The latter accesses the MM for capturing: grid nodes execution performance history and user programs' evaluation costs; in addition to online statistics of intermediate results. Based on the collected statistics, the SC applies a grid node scheduling algorithm (see section 3.1 bellow) to devise a subset of the grid nodes to be allocated by the query engine manager (QEM) for executing in the user programs.

The QEM is responsible for deploying the query execution engine (QE) services at the nodes specified in the DQEP and managing their life-cycle during the query execution. The QEM manages the QEs real-time performance by querying the QEs throughput statistics and deciding on an online rescheduling of the QEs with a plan re-optimization. Each QE receives a fragment of the complete DQEP and is responsible for the instantiation of the operators and the execution control, including the communication with other fragments for retrieving the tuples. As part of the DQEP, the scan operator accesses the data sources with wrappers that prepare the data according to a common data format. We are interested in emphasizing the Query Processing functionality which is presented next.

3. DISTRIBUTED QUERY PROCESSING

The CoDIMS-G query processing model follows the adaptive execution strategy named Eddies [Avnur and Hellerstein 2000]. The Eddies strategy provides for an adaptive scheduling of the query operators according to their on-line production. Since the statistics on the (query) machine performance and the data characteristics may vary during the execution or may even not be available, the idea is to produce a first DQEP (Distributed Query Execution Plan) without spending too much time looking for an optimal query plan, and to define the tuples routing trough the DQEP operators in real time. During execution, initial node allocation is reevaluated, according to real-time nodes throughput statistics, and a new allocation strategy may be devised by the dynamic optimizer.

This is certainly a fair assumption regarding the grid environment, where nodes´ performance may be changing as a consequence of new tasks allocations. In addition, data statistics may be imprecise or inexistent, as is the case where data comes from data sources on the web.

As a result of the above mentioned adaptive execution strategy selection, we use a simple but efficient query optimization strategy, which is explained as follows.

We express a query as a query graph QG, defined as a partial ordered set of operators QG = $\{\Omega, \lambda\}$, where Ω is a set of algebraic operators and λ is a set of dependencies relations, where if ($\omega_1\ \lambda_1\ \omega_2$), with $\omega_1, \omega_2 \in \Omega$ and $\lambda_1 \in \lambda$, then ω_2 succeeds ω_1 in a bottom-up navigation of the DQEP and not ($\omega_2\ \lambda_1\ \omega_1$). In addition, we define Ω_1, Ω_2 as subsets of Ω, where the former comprises join predicates, and Ω_2 comprises a set of user programs. A DQEP is generated by: (1) pushing down simple predicates (restrictions) close to the corresponding relations, (2) permuting join orders in Ω_1[†] and choosing the one providing for a minimal cost, obeying the precedence in λ; (3) analyzing the parallelization of user programs in Ω_2 using a node scheduling policy, which completes the DQEP with a list of nodes to be allocated for the user program instances evaluation. This strategy guarantees that costly programs only get invoked when all predicates have been evaluated, eventually reducing the number of tuples to be processed by them.

In the next subsections, the scheduling algorithm and adaptive query execution strategy are presented.

3.1. The Grid Greedy Node Scheduling Algorithm

In this section we present the grid greedy node (G^2N) scheduling algorithm. The main idea behind G^2N can be stated as: “an optimal allocation strategy for an independent user program of the tuple by tuple type is the one where the elapsed-time is as close as possible to the total cost on each node evaluating an instance of the user program”.

The problem can be formalized as follows: given a set of nodes’ throughputs N, and a set of equally costly independent tasks P, define a subset N_1 of N, which minimizes the elapsed-time for evaluating all the tasks in P.

The algorithm receives a set of available nodes with corresponding average throughputs ($tp_1, tp_2, \ldots, tp_n$) measured in seconds per particle. This includes the average cost involved in transferring one particle to the evaluating node and processing it. The total estimated number of tasks (*T)* to be evaluated, corresponds to the number of virtual particles (*NP*) multiplied by the number of iterations (*NI*) for each particle. The output of the G^2N comprises a set of selected grid nodes.

We now present a brief description of the G^2N. Initially, the algorithm classifies the list of available grid nodes according to a decreasing order of their corresponding throughput values. Then, it allocates all *T* tasks to the fastest node. After this, the program tries to reallocate at least one particle from the already allocated nodes to a new grid node. If this succeeds, with a new query elapsed-time less than the previous one, it continues reallocating particles to the new allocated grid node, until the overall

[†] We consider that the number of joins to be small, not greater than 3 in supported applications

elapsed-time becomes higher than the last computed one. Conversely, if the reallocation of a single particle produces an execution with higher elapsed-time than the one without the new grid node, the algorithm stops and outputs the grid nodes accepted so far.

In addition to the (pre-execution) static allocation of grid nodes, a dynamic strategy validates and corrects possible differences observed during the execution, when the *QEM* compares the estimates for the nodes throughput against the real-time values. Whenever the estimates fall bellow a certain threshold the *QEM* calls the Optimizer to reevaluate the node assignment. A new grid node allocation is produced considering: the number of tasks yet to be evaluated and the up-to-date grid nodes throughput.

3.2. Query Execution

Query execution in the CoDIMS-G is implemented by an instance of the QEEF framework [Ayres and Porto 2004], which is a software framework for the development of query execution engines that can be extended to support new query characteristics, including new operators, different algebras and different execution models among others.

In the CoDIMS-G, the QEEF framework has been extended to support the following functionalities: user's function execution, distribution and parallelism. The strategy for introducing user programs into a DQEP is to implement the Apply operator [Porto 2001] as a new algebraic operator that encapsulates users' program invocation and parameters passing by values extracted from input tuples. The operator implements the standard iterator interface and, for each tuple, invokes the user program with the input parameters captured from the input tuple. User program's result is concatenated to the input tuple to compose an output tuple.

In order to support distribution and parallelism, we conceived two new operator modules: remote and parallel. The remote module implements the communication mechanisms among distributed fragments of a DQEP. It comprehends two dataflow operators Sender and Receiver [Kossmann 2000]. The Sender and Receiver operators are responsible for providing transparency in distribution aspects such as communication technology and data (de)serialization. In our current implementation, the Sender operator buffers the results produced by a fragment and sends them to the demanding Receiver. A Sender operator is coupled to one and only one Receiver operator. In order to reduce the communication overhead, the results are sent in blocks in an attachment of the reply message (i.e. getnext block message).

The parallel module supports the parallel evaluation of logical operations in a DQEP. It is implemented by the control operators: merge and split. The split operator distributes the dataflow through local queues that cache data to be sent to each individual instance of the parallelized operator, whereas the merge operator captures results coming from parallelized operators.

4. CoDIMS-G Particle Tracing

In this section we sketch the extension of the CoDIMS environment to support the particles trajectory computing problem (TCP) in grid architecture. Examples of scientific visualization applications problems that consider particles tracing problems can be found in [Rosenblum et al. 1994]. Such problem can be mathematically defined by an initial value problem:

$dx/dt = F(x,t),\ x(0)=P_0,$ (1)

where $F{:}R^3 \times R_+ \rightarrow R$, is a time-dependent vector field (velocity, for example). The solution of problem (1) for a set of initial conditions gives a set of integral curves which can be interpreted as the trajectory of massless particles upon the flow defined by the field $F(x,t)$. Other particle tracing methods can be used (streamlines, streaklines, among others) with slight modifications of the above equation [Rosenblum et al. 1994]. Usually, scientific data are represented in a cell decomposition of the domain and the velocity field *F* is only known at the nodes of the cells (tetrahedrons in what follows).

Information regarding particles initial position, domain decomposition and fluid velocity vectors are modelled in our work as relations:

Particles (part-id,time-instant,point): particles in their initial position in a time instant;

- Geometry (id, tetrahedron): cell domain decomposition, modelled as tetrahedrons;

- Velocity (point,time-instant,velocity): fluid velocity in cell vertices;

In addition, we consider a program trajectory (particle-id, point, velocity), that computes the next position of a given particle, producing a tuple t (part-id,point). The computation of particles path consists of estimating particles positions in a path for a certain time interval. Each record corresponds to an instant in time and is represented by data according to the above schema. The TCP aims at interpolating particles position through the fluid path between record intervals.

The computation of particles position in a time instant can be expresses as a SQL-like query embedded in a TCP procedure, such as:

```
Begin TCP procedure
   for i= 1 to number-iterations  do
       select  part-id,  trajectory(p.part-id, p.poin t,  v.velocity)
       from Particles p, Geometry g, Velocity v
       where  contains (p.poin, g.tetrahedron)  and
               matches(g.tetrahedron, v.point, p.time-instant)
end TCP procedure
```

The TCP procedure computes particles subsequent positions up to the number of defined iterations. The computation is encapsulated in an SQL-like query. The user-function contains finds a tetrahedron in Geometry whose region contains a virtual particle, whereas matches captures the velocity vectors according to tetrahedron vertexes positions. The trajectory program computes the resulting velocity vector and particle´s next position in the path. The TCP problem is an instance of a class of visualization application problems that will benefit from a middleware infra-structure capable of providing an efficient evaluation within a grid environment. Many challenges are brought by this application. Initially, we are interested in integrating the middleware within the Grid architecture and its standard software [GT3] platform. In respect to the TCP problem itself we will be focusing on issues related to the parallel computation of particles trajectory using available grid resources. In this initial prototype we will be concerned with three main issues: load balancing the parallel computation, managing memory resources and minimizing message exchange between nodes.

5. References

[Avnur and Hellerstein 2000] Avnur, R. and Hellerstein, J., "Eddies: Continuously Adaptive Query Processing", ACM SIGMOD Record 29(2): 261--272 (2000).

[Ayres and Porto 2004] Ayres, F., Porto, F. Melo, R.,"An Extensible Query Execution Engine for Supporting New Query Execution Models", to appear in LNCS 2004.

[Barbosa and Porto 2002] Barbosa, A., Porto, F., Melo, R. N., "Configurable Data Integration Middleware System", Journal of the Brazilian Computer Society, 8(2): 12-19 (2002).

[Berman et al. 2002] Berman, F., Fox, G. C., Hey, A.J.G., "Grid Computing: Making the Global Infrastructure a Reality". John Wiley & Sons Inc., 2002.

[Boulic and Renault (2001)] Boulic, R. and Renault, O. (1991) "3D Hierarchies for Animation", In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

[Dyer et al. 1995] Dyer, S., Martin, J. and Zulauf, J. (1995) "Motion Capture White Paper", http://reality.sgi.com/employees/jam_sb/mocap/MoCapWP_v2.0.html, December.

[Foster 2002] Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed System integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[Giraldi et al. 2004] Giraldi, G. A. ; de Oliveira, J.C.; Schulze, B.R.; Silva, R.L.S.; Porto, F.A.M., CoDIMS - An Adaptable Middleware System for Scientific Visualization in Grids, *Concurrency and Computation: Practice and Experience*, 16(5):515--522, (2004).

[GT3] The Globus toolkit, www.globus.org, 2004.

[Holton and Alexander 1995] Holton, M. and Alexander, S. (1995) "Soft Cellular Modeling: A Technique for the Simulation of Non-rigid Materials", Computer Graphics: Developments in Virtual Environments, R. A. Earnshaw and J. A. Vince, England, Academic Press Ltd., p. 449-460.

[Kossmann 2000] Kossmann, D., "The State of the Art in Distributed Query Processing". ACM Computing Survey, 32(4): 422--469, (2000).

[Kranzlmuller et al. 2003] Kranzlmuller, D., Heinzlreiter, P., and Volkert, J., "Grid-Enabled Visualization with GVK", In Proc. of the 2003 Annual Crossgrid Project Workshop & 1st European Across Grids Conference, 2003.

[Porto 2001] Porto, F., "Strategies for the Parallel execution of user programs in scientific applications", PhD Thesis, PUC-Rio, April 2001, In Portuguese.

[Rosenblum et al. 1994] Rosenblum, L. et al., "Scientific Visualization: Advances and Challenges", Academic Press, 1994.

[Shalf and Bethel 2003] Shalf, J., and Bethel, E. Wes, "The Grid and Future Visualization System Architectures", IEEE Comput. Graph. Appl. (Special Issue), 23(2):6--9, (2003).

[Silva et al. 2004] Silva, V. F. V., Dutra, M. L., Porto, F., Schulze, B., Barbosa, A. C., Oliveira, J. C.. An Adaptive Parallel Query Processing Middleware for the Grid. To appear in *Concurrency and Computation: Practice and Experience (Special Issue)*, 16(5), (2004).