Data Integration Middleware System for Scientific Visualization

Gilson A. Giraldi(1) Fabio Porto(2) Bruno Schulze(1) Vinicius Fontes(1,3)
M. L. Dutra(1,4)

(1)Department of Computer Science
National Laboratory for Scientific Computing
Petrópolis, RJ, ZIP 25651-070, Brazil

(2)Database Lab.
Swiss Federal Institute of Technology Lausanne
Switzerland

(3)Computer Science Departament
Pontifical Catholic University Of Rio de Janeiro
Rio de Janeiro, RJ, Brazil

(4)System Engineering Department Military Institute of Engineering
Rio de Janeiro, RJ, Brazil

gilson,schulze,vfontes,mldutra@lncc.br
fabio.porto@epfl.ch

Abstract

In this paper we focus on distributed scientific visualization on the grid. Specifically, we firstly find out basic requirements for distributing graphics applications over a grid environment. Then, we propose a middleware infrastructure adapted for supporting scientific visualization applications that meets 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. This idea can be implemented by CoDIMS which is an environment for the generation of Configurable Data integration Middleware Systems. CoDIMS adaptive architecture is based on the integration of special components managed by a control module that executes users workflows. We exemplify our proposal with the CoDIMS-G, which is a middleware that follows CoDIMS architecture and was designed to provide data and program integration service for the grid. An specific configuration of CoDIMS-G for distributed particle tracing within a grid environment is also presented.

1 Introduction

The growth in the memory and data storage capabilities of the largest supercomputing installations in the world combined with the huge amount of high resolution data that feeds from remotely operated network-connected observatories and experimental equipment [DeFanti and Brown, 2001] has lead to significant data

management, data integration and data analysis problems. Such scenario brings the problem of visualizing large data sets that may be geographically distributed. Henceforth, scientific visualization researchers need computational power and transparent access to network resource. Such requirements can be fit by grid environments [Berman et al., 2002, Foster et al., 2002].

Grid Computing provides transparent access to distributed computing resources such as processing, network bandwidth and storage capacity. A single system image is created, offering open distributed processing support, allowing applications development, usage and maintenance. Grid users essentially see 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 et al., 2002]. At its core, Grid Computing is based on an open set of standards and protocols (i.e., Open Grid Services Architecture: OGSA [Foster et al., 2002]) that enable communication across heterogeneous, geographically dispersed environments. To support resource and services encapsulation it is necessary a layer between the operating system and the applications, called Middleware, which is a shell over the operating system, adding new functionalities to support the distribution of applications [Berman et al., 2002, Foster et al., 2002].

The grid environment is open in the sense that not only the systems are open but the services are open as well. Exporters offer services and importers search for services, independently of the identity and location of the exporter. An open service environment is also characterized by the unlimited access in the sense that the environment is always able to accept a new user. On the other hand, in this environment, an exporter is autonomous to decide on how to perform the service. The Middleware deals with this dichotomy: openness and privacy.

In this paper we are interested on grid computing solutions for scientific visualization applications [Hansen and Johnson, 2003, Porto et al., 2004, Giraldi et al., 2003b]. We offer a survey of this field trying to identify basic requirements for distributing visualization applications over a grid environment. A fundametal point which emerges from this analysis is that we should consider scientific visualization in grid from an integrated global view of data and programs published by heterogeneous and distributed data sources. Henceforth, we propose a solution based on CoDIMS (Configurable Data Integration Middleware System) [A.C.P. Barbosa, 2002] which is a middleware environment for the generation of adaptable and configurable middleware systems. Data integration systems, like CoDIMS, allow applications to benefit from transparently accessing published resources independently of their localization, data model and original data structure [Tomasic et al., 1998, SEL, 1999, Martinez and Roussopoulos, 2000]. 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 to conform to the requirements of the applications.

We focus on the configuration of the CoDIMS environment to generate middleware systems to support the execution of scientific visualization applications in a grid environment. We exemplify our proposal with the CoDIMS-G, which is a configurable data and program integration system that follows CoDIMS architecture and was designed to support scientific applications. An specialization of CoDIMS-G for particle tracing within a grid environment is also discussed.

This paper is organized as follows. Section 2 offers a discussion of middleware from the viewpoint of services in the grid. Next, section 3 provides a review of scientific visualization in grid. Then, in section 4 we summarizes the specific services that a middleware should provide for performing scientific visualization in grid. We emphasize that CoDIMS is suitable to generate such infrastructure. CoDIMS and CoDIMS-G are described in sections 5 and 6, respectively. The configuration of CoDIMS-G for supporting particle tracing in a grid is discussed on section 7. We are implementing this architecture and some experimental results will be available soon. Finally, we show final considerations about this work.

2 Middleware

Two Middleware approaches can be applied to grid environments: based on object orientation and on service containers. The former approach supports infrastructure and application objects that use object-oriented mechanisms, like inheritance, polymorphism, encapsulation, and so on. Examples of infrastructure objects are transactional, concurrency and naming service. CORBA (Common Object Request Broker Architecture) [Group, 2002, Group, 2004] is an example of this kind of Middleware. The last approach supports containers with infrastructure and application objects coexisting together. The Web Services standardization [Domain, 2003] is an example of this class of Middleware.

Although both kinds of Middleware provide access and location transparencies, the service containers approach is more suitable for our purpose because we must identify common requirements for scientific visualization applications in grid. In general, Grid computing applications (simulation, scientific visualization applications, among others) needs the following basic services from the Middleware [Schulze and Madeira, 2004]:

* Naming;

* Creation of transient services;

* Service invocation;

* Service discovery;

* Multiple protocol bindings;

* Life-time management;

* Change management to allow different versions of the same service;

* Event notification;

* Instrumentation and Monitoring;

* Allocation management to allow task distribution;

* Security;

* Concurrency control.

Moreover, **data management** is another fundamental service for applications that involve heterogeneous data types. For instance, in the e-Science field [of Liverpool,], experimentalists observe and measure phenomena in the natural world and theories are elaborated to explain these natural phenomena. Then, computational scientists develop numerical models and simulations to predict how nature would behave, if the theories were correct. In the end, real (observational) data is compared with simulation data. Besides, for some applications, these heterogeneous datasets can be combined to augment the reality through graphical representations created from the numerical datasets or even superimpose a simulated system on a real scene to get more insights about a phenomenon [Silva et al., 2004]. These research facilities involve the issue of integrating heterogeneous data sources which has received a great attention from the database community. This can be noted by the number of systems proposed, each one having a special type of application in view. One may want to classify these initiatives according to the type of application they were designed for supporting. A first category includes those systems specifically designed to integrate data from different sources that are the basis for the creation and maintenance of Web sites, like Araneus [Mecca et al., 1998] and Strudel [Fernandez et al., 1997]. A second category groups those systems based

on mediation technique, like: TSIMMIS [Molina et al., 1997] and DISCO [Tomasic et al., 1998]. A third category are database oriented: MIRO-Web [Fankhauser et al., 1998], Garlic [23, 2000]. Finally, MOCHA [Martinez and Roussopoulos, 2000] and LeSelect [SEL, 1999] are considered data integration middlewares, in the sense that they extend traditional data integration services with mechanisms for user defined function execution. In this scenario, user queries can be seen as simplified workflows over distributed and heterogeneous data and programs. As suggested by the above classification, these systems are designed having a specific application in mind. Supporting new application requirements may entail quite a strong developing effort. Systems directed towards supporting a wide spectrum of applications are commonly large and heavy which translates into execution inefficiency and complexity in use. Instead, a better option is a to use a flexible and configurable environment to generate data integration middleware systems.

A data integration middleware system is responsible for providing access to data that is distributed and stored over heterogeneous data sources. Given this general definition, the CoDIMS approach for the development of data integration systems specifies some pre-defined interfaces corresponding to data integration middleware services (DIMS) commonly presented in this type of systems, which include: Metadata Manager, Query Processing, Transaction Manager, Concurrency Control, Rule Manager and Communication. Also, the possibility of combining data integration services with mechanisms for user defined function execution is useful to integrate processing routines, data access, and visualization (post-processing) methods. This is the point we aim to explore for scientific visualization in grid. We will propose a Middleware System for data and user's program integration. However, before the system specification we must review visualization in grid environment in order to identify specific requirements (services) to be offered by such middleware.

3 Review of Visualization in Grid

Scientific Visualization is a relatively new computer-based field concerned with techniques that allow scientists to create graphical representations from the results of their computations, as well as to visualize features of interest in a data set obtained through imaging instruments (see [Rosenblum et al., 1994] for a review). Despite been computational expensive, visualization techniques have become essential in interpreting data. To address the computational cost, specially for large datasets, researchers have explored high performance computers and, more recently, grid environments [Giraldi et al., 2003a, Hansen and Johnson, 2003].

The hallmark of grid-based applications is focused on remote visualization. Typical past approaches to remote visualization focused on one of two broad methods. The first approach performs all visualizations on the server and sends image data (or X-protocol streams) to the client. This approach works best with a large data set. The alternative approach transfers subsets of the large scientific data from the server to the client workstation, where visualization occurs completely on the desktop machine. This approach works best when interactivity is most important. Such solutions must be re-designed for grid environments. Next, we offer a review of important proposals in this field [Singh, 2003, Kranzlmller et al., 2002]. We start with application specific approaches to end with a middleware proposal.

3.1 Visapult

It is a pipelined-parallel image-based system for volume rendering which implements a high-performance extension to the IBRAVR visualization algorithm [Bethel and Shalf, 2002, Bethel and Shalf, 2003] (see also [Max, 1995] for an introduction on volume rendering). Visapult has three components:a raw data source, a viewer,and an MPI-based back end (Figure 1).

During the partitioning process, data is read into each processor in parallel. Each processor then performs software volume rendering upon its subset of the data. The resulting image from each processor is transmit-

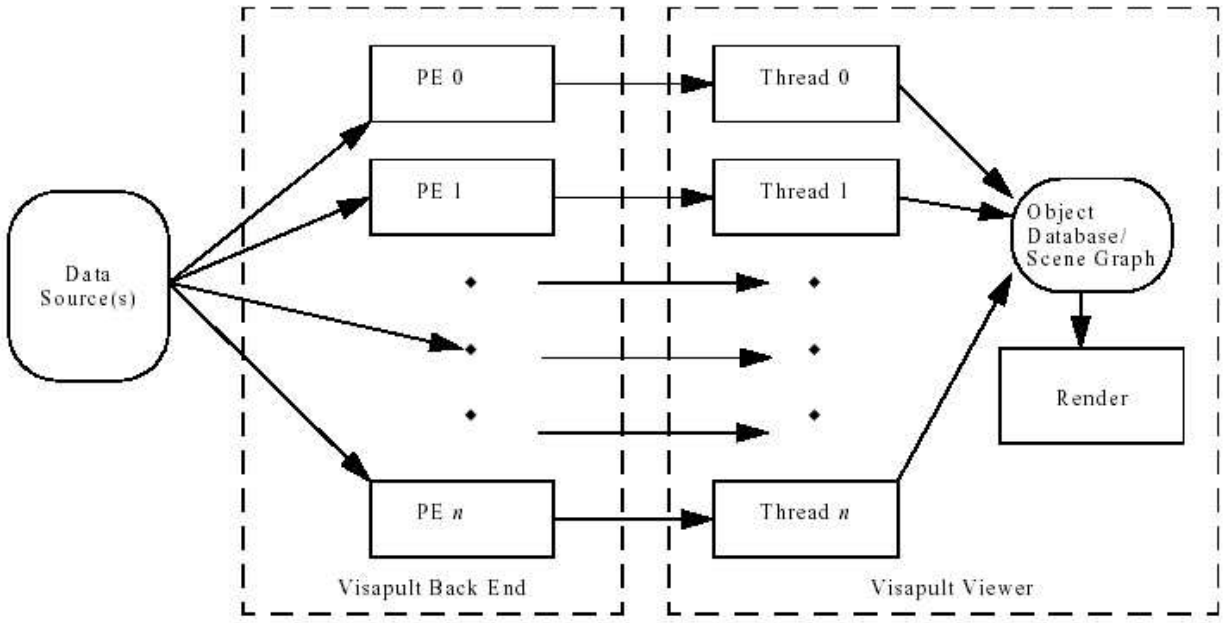

Figure 1: Visapult architecture.

ted over the network to a peer receiver in the Visapult viewer, where it is inserted into the scene graph as a 2D texture. This process takes advantage of both the computing horsepower of MPPs for partial rendering as well as the benefits of hardware-accelerated rendering available on many low-cost client workstations. It also decouples interactivity on the desktop from the latencies involved in moving data over the network.

3.2 Patras/ITBL

Patras/ITBL visualization system [Suzuki et al., 2003] follows an architecture based on a client-server model. It provides capabilities for simultaneous execution of a simulation program and visualization of results, as well as visualization of extremely large datasets.

The Patras server-client remote visualization system carries out the visualization process on each processor of a supercomputer (server) and displays images onto a user terminal (client), as shown in Figure 2.

The Patras library is designed to run in data-parallel. The visualization process is based on the domain decomposition method. Each processor generates a fraction of a total image data with a Z-buffer value, and then a specified processor composes an integrated image. Using this Z-buffer value, Patras merges partial images into a single image at each simulation time step and then implements Message Passing Interface (MPI) communications to gather the image data.

The user can track and steer simulation parameters and control the simulation procedure. To actually use the Patras visualization library, users must insert function call routines into the simulation program.

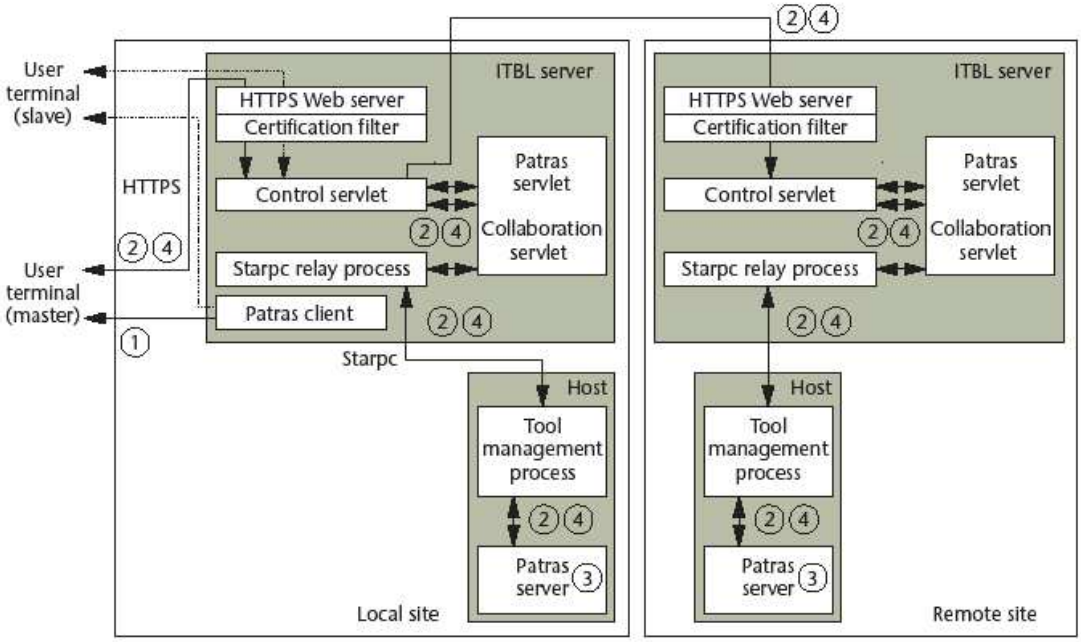

Figure 2: Patras/ITBL visualization system.

3.3 Solutions for Bandwidth Limitations

In [Norton and Rockwood, 2003] authors address the problem of communication bandwidth limitations for interactive volume visualization. They propose a new mechanism for visibility detection, a specification of a new client/server division of effort, and a new mechanism for filtering wavelet coefficients through octree prioritization.

The client/server architecture can visualize the results of grid computations. The supplier of data installs volumetric data resulting from a computation at a server site on the grid that's suitable for distributing data to visualization clients. The rendering occurs on a client machine with sufficient memory and rendering performance to support interactive volume rendering of local data (see Figure 3).

The grid computation results in a volumetric data set. To set up the data server, the wavelet transform is applied to the full volume data set, resulting in a repository of wavelet-encoded data that is maintained at the data server. The client workstation progressively builds and maintains its own 3D array of volumetric data. The client volume data array (initially only a coarse approximation to the original volume data) is progressively populated by reconstructing data from wavelet-filtered coefficients transmitted from the data server. The method was applied using Haar and Daubechies ($D4$ and $D6$) wavelets due to their relatively small compact support, which make easier the control over selective filtering of visible portions of the volume and to explore the correspondence that exists between octrees and wavelet decomposition of a volume (see [Norton and Rockwood, 2003] and references therein). Progressive transmission of volumetric datasets through wavelets was also explored in [Trott et al., 1996].

In [Norton and Rockwood, 2003] authors distinguish between interactive and responsive volume visualization. A visualization session is interactive if users can navigate through the scene, rendering at 10 frames per second or faster. A visualization is responsive if the users experience a delay of 2 seconds or less between

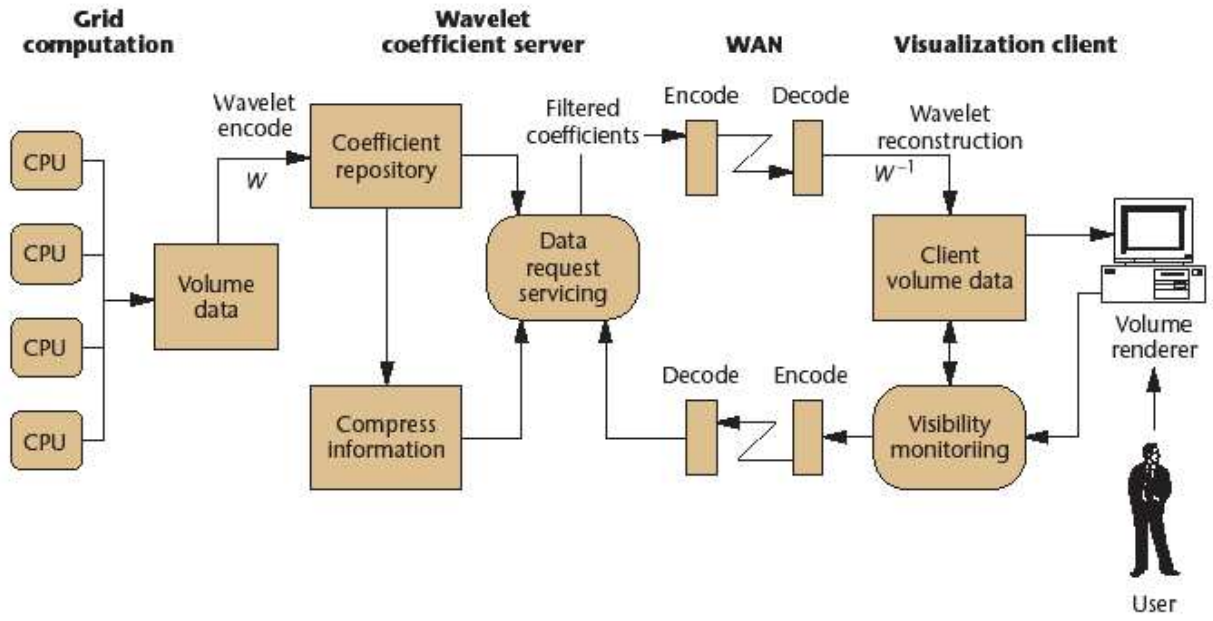

Figure 3: Wavelet-Based progressive transmission architecture.

when a volume is selected and when they can access a detailed rendered image. Interactive visualization requires adequate rendering hardware; responsive visualization requires sufficient data communication (or local storage) of volume data. The architecture that was proposed in [Norton and Rockwood, 2003] provides responsive visualization of data on the grid, assuming the users already have access to an interactive volume visualization capability.

Network bandwidth limitations is addressed in [Bethel and Shalf, 2003] by means of aggressive network tuning and network protocol modifications. The Visapult's effectiveness have been improved through this work. In particular, it is used a new connectionless user datagram protocol (UDP) that improves network efficiency from a 25 to 88 percent line rate increase for multigigabit networks. This connectionless protocol also dramatically reduces the latency of network event delivery, improving the responsiveness of wide area distributed interactive graphics applications as compared to transmission control protocol (TCP) streams.

The initial implementation of Visapult reader (section 3.1) used a TCP protocol and requested data from the network in a specific order; TCP guarantees correct delivery of data. In contrast, UDP makes no such guarantees, so each UDP packet must contain information indicating the location in the domain-decomposed array where the Visapult back end must place the data payload (see Figure 4).

This ensures that the system can treat each packet independently so that packet ordering and loss have minimal effect on destination processing. The Visapult back end was modified to render continuously rather than waiting for all packets in a given frame to arrive. The visual effect of this choice is an immediate response involving a coarser representation with progressive refinement of the image over time. Data from the previous frame was used to prime the receive buffer and fill in gaps where packets were lost. Authors claim in favor of such approach by saying that possibly packages loss due to the unreliable connection would be tolerated because it (visually) works as a kind of level-of-details approach.

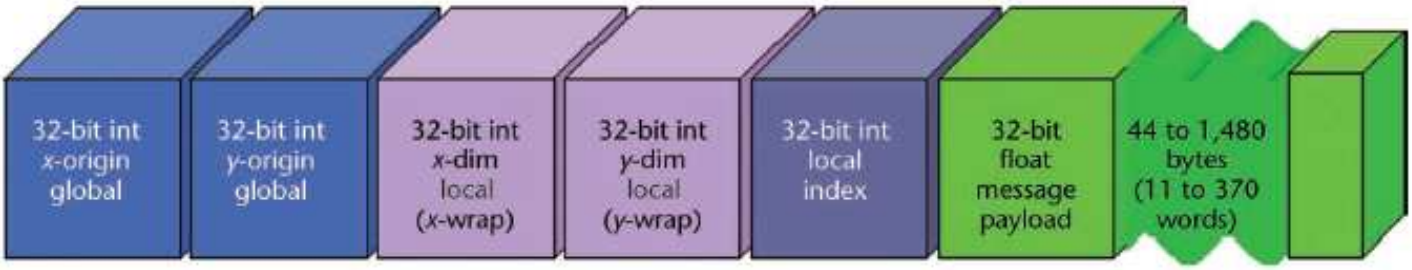

Figure 4: UDP packet format diagram.

3.4 Controlling Visualization

End-users need to control visualization applications. Research on controlling visualizations using the grid typically falls into two areas: Web-based visualization systems and distributed visualization in grid-like environments, like Portals [Suzuki et al., 2003].

To use grid resources effectively, researchers need a central access point to manage the resources, provide a visual means to explore the data, and record these explorations for further investigation and dissemination. The article [Jankun-Kelly et al., 2003] describes such a system that is being developed jointly by the University of California, Davis, and the Lawrence Berkeley National Laboratory (LBNL).

This centralized system acts as a portal into grid enabled visualization systems. Scientists using the portal can focus on the important task of extracting insights from their data through visualization instead of having to worry about process management. The system consists of three major components:

. a Web-based user interface to grid-enabled visualization services,

. a visualization Web application that tracks the exploration of visualization results, and

. the portal application server that manages and coordinates the authentication for and use of grid resources.

The Web interface (WebSheet) implements an entirely Web-based version of the visualization exploration sheet-like interface discussed in [Jankun-Kelly and Ma, 2001].

Figure 5 summarizes the VisPortal/AMRWebSheet architecture, which encompasses a volume renderer for volumetric datasets represented by Adaptive Mesh Refinement (AMR) methods. Such multiresolution data representation allows a broad range of scales, represented by different mesh resolutions. Thus, its volume rendering should be properly performed to avoid artifacts in the final image (see [Jankun-Kelly et al., 2003] and references therein for details). The AMRWebSheet interface and the Web application were implemented in a flexible visualization exploration and encapsulation framework. The framework, implemented in Python, consists of a series of objects that manage visualization sessions and the interface's interactions with the sessions. Visualization session, transform, parameter, result, and derivation objects exist within the framework to capture the information described in the visualization exploration process model.

It was implemented the Web application servlets that manage visualization sessions in Python using the Webware application environment (http://webware.sourceforge.net/). Apache running under Linux was used to serve Web pages. A group of servlets creates, processes, and stores sessions. When a client connects, the application creates a new session identified by a temporary cookie in addition to servlet-persistent

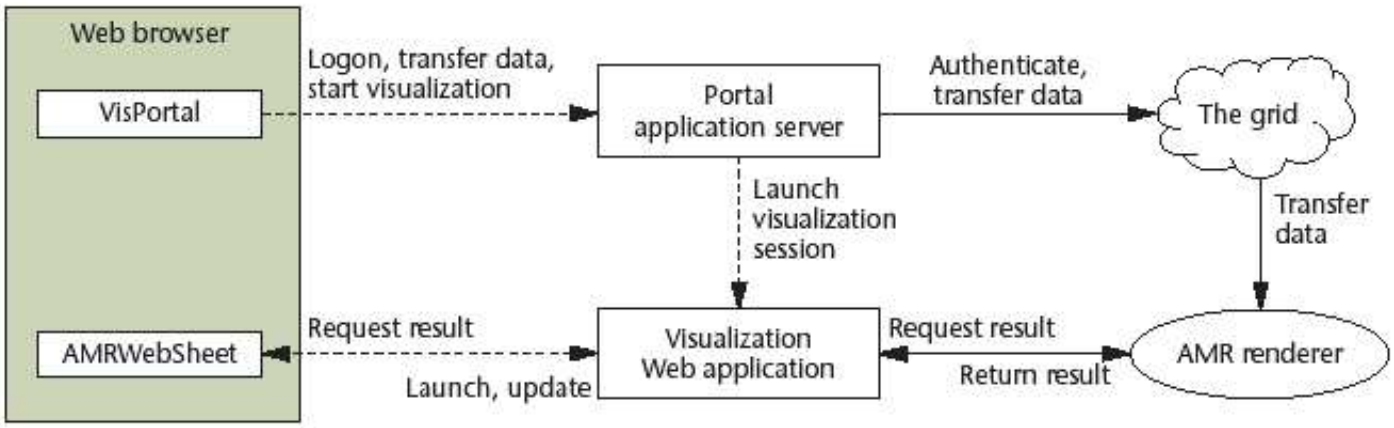

Figure 5: The VisPortal/AMRWebSheet architecture.

objects. Whenever a user interacts with the generated HTML interface, the application communicates the AMRWebSheet HTTP requests to the interface servlet. This request in turn modifies the visualization session state. When the client needs to update, the server performs a refresh to display the new information. When a session terminates or expires due to inactivity, the application encodes the session results as an XML document on the application server for later retrieval.

The application server handles all communication between the AMR volume renderer and the AMRWebSheet. When the AMRWebSheet requests a result, a visualization transformation class instance on the Web application requests the corresponding result from the volume-rendering server. When the volume renderer returns the corresponding result, the Web application stores a copy with the visualization session results. The application then displays the result by forcing a refresh on the client's Web browser.

3.5 Middleware Solution

Recently, researchers have realized that distributed visualization in grid needs a middleware infrastructure. This layer would provide basic services and resource allocation for visualization applications. The Grid Visualization Kernel GVK [Kranzlmller et al., 2002, Kranzlmuller et al., 2003], which is a grid middleware extension built on top of the Globus Toolkit, is such a proposal.

GVK offers efficient and flexible transportation mechanisms for visualization purposes, while using standard interfaces for connecting to scientific applications and output devices. With the available interfaces of GVK, grid-enabled applications may use traditional visualization toolkits as well as sophisticated virtual reality devices to present the results to the scientific user.

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. In fact, a main issue of GVK is its seamless integration into existing grid infrastructures, which is only possible if GVK utilizes the available infrastructure services. With this in mind, the main characteristics of GVK are: transparent access, ease of use, arbitrary interconnections, optimal performance.

The basic philosophy is to provide GVK interfaces as extensions of the commonly available visualization

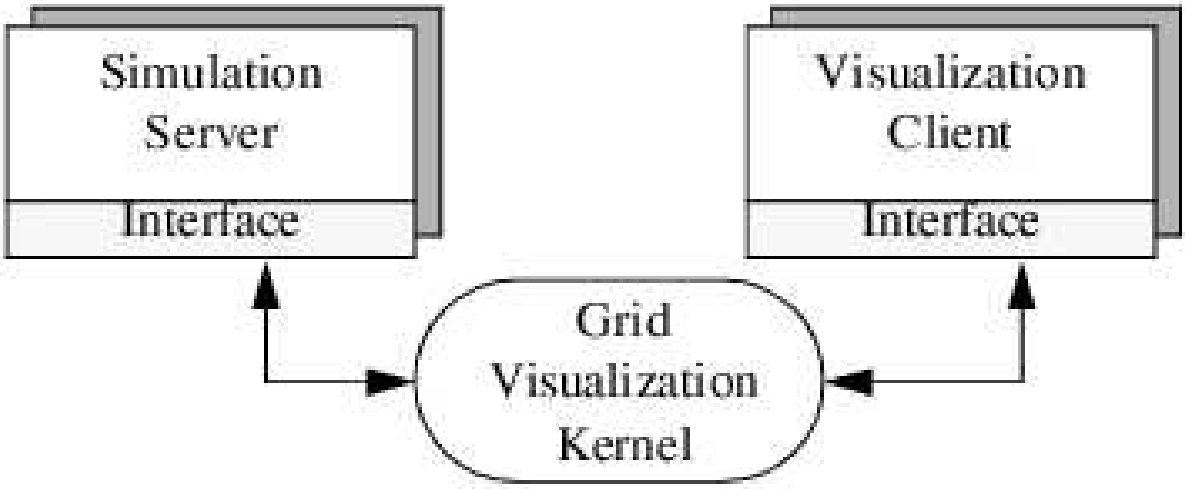

Figure 6: Basic elements of GVK architecture.

toolkits (OpenDX, VTK, etc.). Thus, existing visualization functionalities may be reused. This accelerates the integration of visualization in grid applications, since users do not need to learn "another" visualization toolkit.

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.

For that reason, GVK contains standard compression techniques as utilized in most networked environments. Furthermore, certain optimizations for visualization data and graphical representations may be applied. Some examples in this area include multiple levels-of-detail, occlusion-culling, and image-warping. These optimization techniques utilize the grid itself for their tasks, which makes the GVK a grid application itself.

The visualization process with GVK involves three components as shown in Figure 6: the original simulation server running on the grid and generating the data, a visualization client intended for displaying the output, and the kernel itself for the interconnection between server and client.

Figure 7 gives details of these components and shows the information flow between them, which encompasses the following phases:

(1) During the initialization phase, the simulation sends a visualization request to the GVK portal server. The contents of this request are the minimal requirements to possible visualization clients.

(2) The portal server authorizes the simulation and forwards the visualization request to the GVK visualization planner.

(3) The visualization planner sends a resource information request to Globus MDS in order to inquire about the available resources for the visualization pipeline.

(4) Globus MDS returns information about available resources to the GVK visualization planner.

(5) The visualization planner sends a resource allocation request to the Globus Resource Allocation Manager

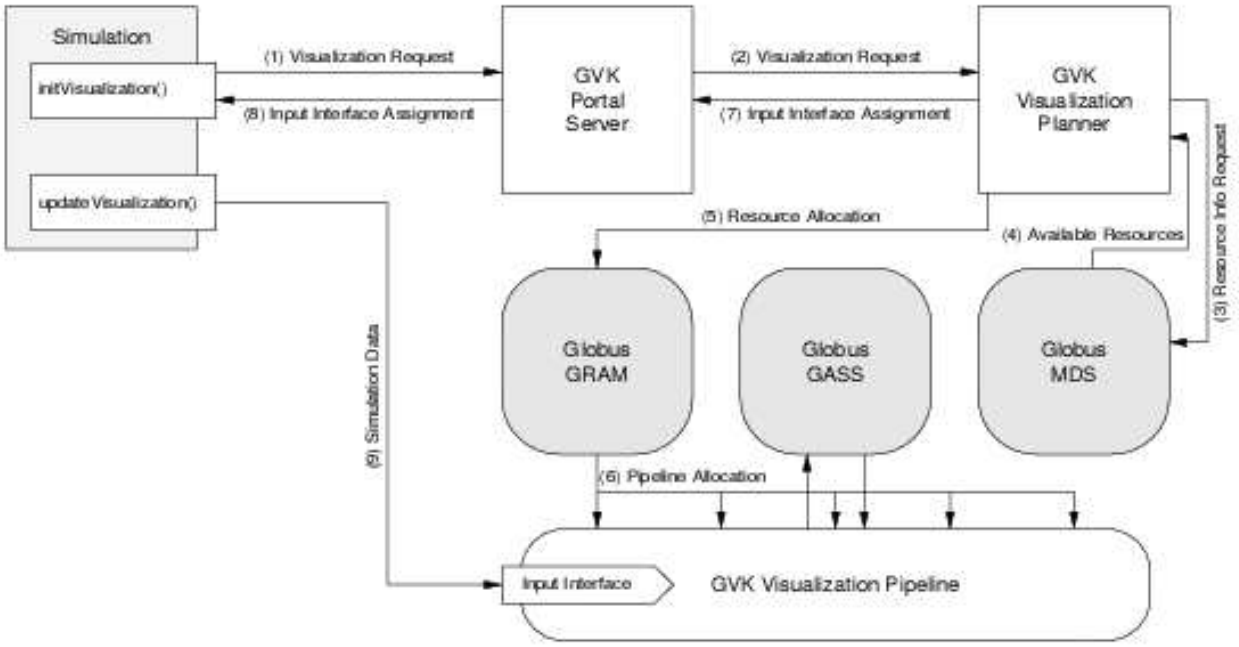

Figure 7: GVK architecture in details.

(GRAM).

(6) Globus GRAM performs the visualization pipeline allocation and initializes all transformation modules as requested by the visualization planner.

(7) The visualization planner returns the input interface assignment to the GVK portal server about the newly initiated input interface of the GVK visualization pipeline.

(8) The portal server stores the new visualization service for future requests and forwards the input interface assignment to the simulation source.

(9) The simulation continues processing the data and updates the visualization (when appropriate) by sending simulation data to the input interface of the GVK visualization pipeline.

Next, we point out GVK limitations and describe our middleware solution.

4 Our Proposal

From the viewpoint of visualization techniques, volume rendering is suitable for distributed memory machines because it scales well for these architectures. This point is explored by Visapult architecture described in section 3.1. Client-server solution are elegant models for component integration. A remote server (supercomputer) carries out the visualization process and then uses some communication interface to send image data to the client for display (section 3.2). End-users can control application sections through Web-based or Portals engines (section 3.4).

The transport services and GVK middleware reported on sections 3.3 and 3.5, respectively change the viewpoint of the problem. Instead of center on specific techniques or architectures they focus on services to be provided to allow distributed visualization in grid. In section 2 we already pointed out common requirements for grid application. The GVK developers cited multiplexing, buffering, and synchronization as specific needs for visualization in grid. Now, we claim that the next requirements come from the need for interactivity. This is a fundamental issue for visualization applications. In order to address this requirement in grid architectures we must provide:

(a) Dynamic scheduling: We need to predict performance before launch components as well as to re-schedule tasks during an application execution. Grid environments are heterogeneous computing resources which nodes architectures may be of any kind. Thus, we should predict performance which implies directory services with statistics about grid nodes [Fontes et al., 2004].

(b) Performance management: We should monitor the performance of grid nodes during an application section. In grid environments resources can change dynamically at runtime. Allocated nodes may undergo performance problems, traffic in network links is variable, etc. Henceforth, performance management becomes an important issue to maintain interactivity.

(c) Transport Services (Connectionless UDP, Transport encoding, Low-latency transport): When reading data from a remote raw data source, the bandwidth of the network connection may be a bottleneck. The solutions presented on section 3.3 (transport encoding process protocols with progressive transmission, data compression services or more suitable transport protocols) must be available in the transport layer.

Moreover, as already mentioned out on section 2, when visualizing large and distributed heterogeneous data sets, data management is another fundamental service. Henceforth, we also need:

(e) Data-driven architecture: It should provide resources for data distribution, data integration and data management.

These requirements can be addressed through a middleware solution more general than GVK (section 3.5). In GVK the scheduling is static, in the sense that it is performed just before that the components are launched.. There is no performance management at run-time. Besides its client-server architecture basically aims the initialization of the visualization pipeline (client) and its link with the data generator (server). Data distribution-integration-management are not provided by GVK itself.

In this paper we describe a data-driven middleware architecture that can encompasses dynamic scheduling, performance management and transport services for distributing scientific visualization tasks in a grid. This architecture was recently proposed by some of us [Fontes et al., 2004] and we are in charge to complete its implementation.. We consider scientific visualization in grid from the viewpoint of an integrated global view of data and programs published by heterogeneous and distributed data sources. Such viewpoint can be meet by CoDIMS as we shall see next.

5 CoDIMS

In this section we describe the basic components of CoDIMS and show how CoDIMS features meet the above requirements. CoDIMS is a flexible and configurable environment to generate middleware systems. The configuration is obtained through a Control component that exports the interface of integrated components and maps user request to an executable workflow of services. In order to flexibilize services implementation, they are implemented using the software engineering framework technique with hot-spots [Nierstrasz and Dami, 1995] to be instantiated. Through component reuse or adaptation of framework components the environment offers an answer for the generation of heterogeneous data integration systems

tailored to specific application requirements.

A data integration middleware system is responsible for providing users transparent access to distributed and heterogeneous data sources. Given this general definition, the CoDIMS approach for the development of data integration systems specifies some pre-defined interfaces corresponding to data integration middleware services (DIMS) commonly presented in this type of systems, which include: Metadata Manager, Query Processing, Transaction Manager, Concurrency Control, Rule Manager and Communication (see Figure 9). For each of these interfaces we provide different components that may be selected into a configured system. In addition, the environment offers a Control component that takes part in any configuration.

New interfaces, corresponding to DIMS not initially previewed, can be added to the environment through their publication in the Control Component (see section 5.2) and by providing its implementation. The added service is included in a workflow sequence to be evaluated by the Control in response to a corresponding user request. The flexibility obtained with these techniques can be summarized as follows:

- DIMS components: provides data integration services fucntionality.

- Framework modules: provides DIMS behavior flexibility.

-Control component: enables DIMS integration into a configurable system.

DIMS component are described in the following section.

5.1 System Components Description

The CoDIMS environment includes some basic data integration functionality implemented by the so called predefined DIMS as well as some interfaces modeled to support DIMS integration and the Control module for managing such integration. In this section, we briefly describe the functionality of predefined DIMS:

- The Metadata Manager component is responsible for managing data source and global schema metadata. In a grid environment, this component transparently maps users data views to distributed data sources. User data views are expressed according to a canonical data model adequate to the characteristics of supported applications. The set of user data views is called integrated global view. Users formulate queries over objects in the global view. The mapping of global views to physical data sources is also expressed through the Metadata Manager component.

- The Query Processing component (QPC) is the most important one in a CoDIMS middleware system. It provides for the efficient evaluation of queries over distributed resources within a grid environment. Within the grid, QPC is extended to support data distribution and query parallelism. It interfaces with Metadata Manager to obtain: mapping information, data sources schema, statistics and localization information.

- The Communication component is responsible for the communication between the middleware system and each data source. A wrapper translates local sub-queries into specific data source access method calls, as in others mediator systems.

- The Transaction Manager component is responsible for providing transaction ACID properties (Atomicity, Consistency, Isolation, and Durability) to applications, in cases where updates of data sources are allowed. This component will not take part into the grid middleware instance.

- The Concurrency Control component provides mechanisms for implementing the isolation of concurrent transactions. This component will not take part into the grid middleware instance.

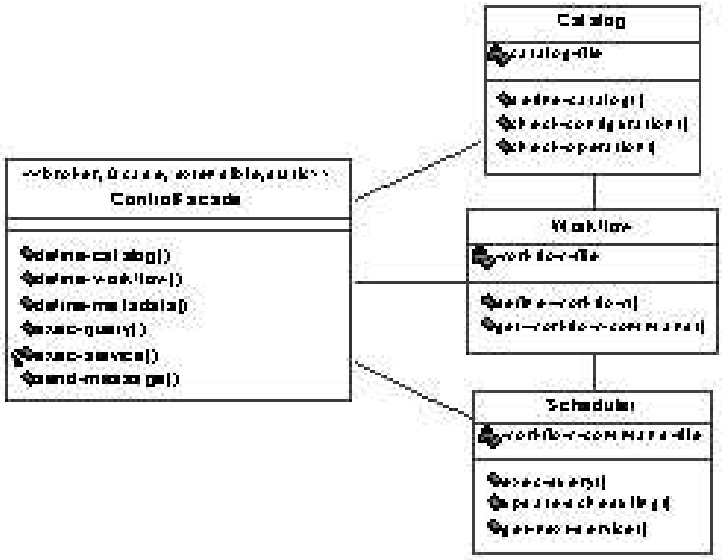

Figure 8: Control Component.

- The Rule Manager component enriches the system with active behavior as proposed in [Fankhauser et al., 1998]. This component will not take part into the grid middleware instance. DIMS components are integrated via the Control component that is described in the following subsection.

5.2 The Control Component

The Control component is the essence of the CoDIMS environment. The Control stores, manages, validates, and verifies both Physical and Logical Configuration. Physical configuration corresponds to the selection of DIMS components, their customization according to application requirements, and registration in the catalog. The selection of DIMS components is subject to the set of offered and required operations that each DIMS specify. By matching the requirements of all selected DIMS in a configuration, the Control component validates it.

Logical configuration refers to the execution logic that integrates DIMS components during the evaluation of a user request. The CoDIMS approach achieves a complete adaptability in terms of services to be executed in a configured system, by associating to an user request a workflow program [Gamma et al., 1995] of DIMS invocation. New configurations require the specification of a workflow program to each of its user requests.

During execution, the Control component automatically responds to client requests by scheduling DIMS services, based on its workflow, and proceeding with DIMS invocation. Figure 8 presents the Control component class model, its façade and its three main modules. In the diagram, some specific notation expresses the façade class as extensible, symbolizing its implementation as an OO framework and static, meaning that the module is not reconfigurable during execution.

The Catalog specifies the physical configuration, which registers each DIMS component present in a configuration, including its name and offered and requested operations. The Workflow module stores workflow programs associated to user requests. Finally, the Scheduler module evaluates workflow programs by following its execution logic and invoking DIMS services, accordingly. The user of the CoDIMS environment

generates a configuration through a process that we present in the next subsection.

5.3 The Configuration Process

The Control, Query Processing, and Metadata Manager components must exist in all configurations. According to the application requirements, other components may be included. The range of possible configurations of CoDIMS can vary from a middleware with the functionality of a simple wrapper to a complex HDBMS (*Hierarchical DataBase Management System*). In cases where a full HDBMS functionality is required the middleware system could incorporate the Transaction Manager and Concurrency Control components.

The process of generating a specific configured system comprehends the following phases:

- Design: the system designer selects the necessary DIMS components. In this phase, new DIMS components may be projected or existent ones may need adaptation;

- Configuration: the system configurator registers the physical and logical configuration in the Control component. For this, two script files are executed by the operations: define-configuration and define-workflow;

- Load Metadata: the application designer (database administrator) defines local, external, and global metadata through three scripts files to be processed by the define-metadata function.

During the Configuration step, the Catalog module check-configuration method verifies if all the required services are being offered by the components interfaces that participate in the configuration. In a similar way, the check-operation method verifies if all operations defined in the workflow are being offered by the specific component. After all these phases, the system is configured and ready for client requests.

6 CoDIMS-G Architecture

In this Section, we present the CoDIMS-G architecture, depicted in Figure 9. CoDIMS-G is a configurable data and program integration system based on the CoDIMS configuration environment architecture to support scientific grid applications [Fontes et al., 2004] .

The focus of CoDIMS-G design is to provide a data service layer on top of grid middleware offering transparent and efficient query processing capabilities for queries comprising non-traditional data formats and user programs. Our initial implementation considers spatial and temporal data types and scientific user programs that together form a scientific application scenario [Fontes et al., 2004].

In order to support such a scenario, we identified two main aspects that drove CoDIMS-G design: adaptivity and parallelism. The former is a requirement to variations in grid resource performance that may affect query response time whereas the latter takes advantage of available grid nodes to reduce query elapsed time. As a matter of fact, CoDIMS-G is, to the best of our knowledge, the first system to combine both adaptivity and parallelism into an integrated query processing system.

The system architecture concentrates around the Control component, having as DIMS services: a query processor and a metadata manager, in addition to a client access point. 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

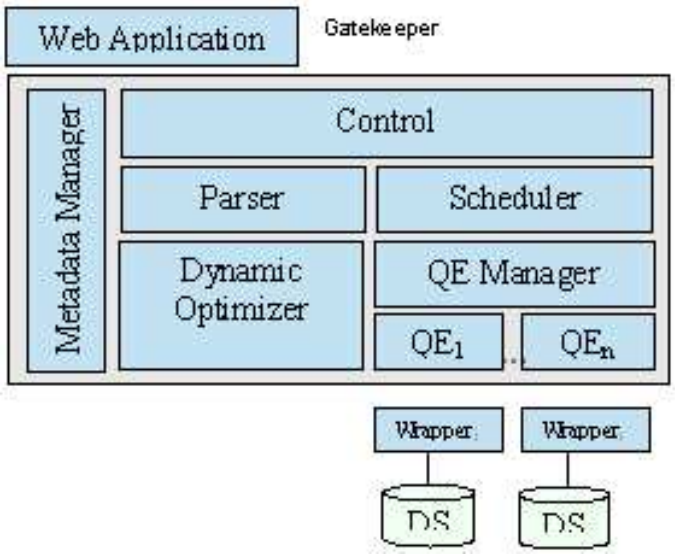

Figure 9: CoDIMS-G Architecture.

them to the Parser Component. The Parser transforms the user's requests in a query graph representation. The Query Optimizer (QO) receives the graph and generates a physical distributed query execution plan (DQEP), using a parallel cost model based on data and programs statistics stored in the Metadata Manager (MM). For DQEPs including parallelizable operators, that is, operators that evaluate on a **tuple** by its basis (*tuple* is a database term refering to a data unity and its attributes), the optimizer uses its Scheduler (SC) module. The latter accesses the MM for capturing: grid nodes execution performance history and estimations for operators unitary 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 6.2) to devise a subset of the grid nodes to be allocated by the query engine manager (QEM) for parallel execution of operations.

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 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. An important issue of the CoDIMS-G architecture is the Query Processing functionality which is presented next.

6.1 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 distributed query execution plan (DQEP) 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 re-evaluated, according to real-time nodes throughput statistics, and a new allocation strategy may be devised by the dynamic optimizer.

The emphasis on a dynamic approach to query processing is justified by the grid environment, in which nodes performance and network throughput estimates change as a consequence of task allocation and peak communication periods. In addition, data statistics are imprecise or inexistent, as is the case where data comes from data sources on the web. As a result of the above, we use a simple but efficient query optimization strategy, which is explained as follows. A distributed query execution plan is chosen from a search space of equivalent such plans derived from the initial query graph by means of commutativity and associativity transformations. We support *select,project, join* type of queries with the addition of user functions. We also assume that in the scientific application scenarios, queries usually involve not more than four relations. In this context, we produce an initial distributed query execution plan by computing a search space formed by permuting the orders of joins and user programs in a deep left type of query operator tree [Elmasri and Navathe, 1994]. In addition to define query operator's order, this phase also analyzes possible parallel strategies for each operator. In this respect, two main decisions should be taken: the choice of operators to be parallelized and the respective nodes where the parallel evaluation should take place. Our option was to adapt to the parallel problem the well known SystemR [et al., 1981] strategy for distributed operation placement. Thus, for each operator in a query operator tree we take one of the following three different parallel decisions: (1) evaluate according to the parallel strategy defined for the previous operator; (2) define a new parallel strategy based on the scheduling algorithm; (3) decide on a strategy according to the next operator in the query operator tree. Finally, for each plan a cost is annotated and the one with the least cost is chosen.

Thus, for each operator in a query operator tree we take one parallelization approach among the following three different ones: (1) evaluate the operator according to the parallel strategy applied to its previous operator, in the query operator tree;(2)define a new parallel strategy based on the result of our scheduling algorithm;(3) decide on a strategy according to the next operator in the query operator tree. Finally, for each plan a cost is annotated and the one with the least cost is chosen.

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 an adaptive query execution strategy are presented.

6.2 The Grid Greedy Node Scheduling Algorithm

In this section we present the grid greedy node (G2N) scheduling algorithm. The main idea behind G2N can be stated as: an optimal parallel strategy for a query operator is the one where its elapsed-time is as close as possible to the sequential cost on each node evaluating an instance of the operator.

The problem can be formalized as follows: given a set of node throughputs N, and a set of equally costly independent tasks P, define a subset N1 of N, which minimizes the elapsed-time for evaluating all the tasks in P. The algorithm is presented in Figure 10. It receives a set of available nodes with corresponding average throughputs $(tp1, tp2, ..., tpn)$ measured in seconds per tuple. This includes the average cost involved in transferring one tuple to the evaluating node and processing it. The output of the G2N comprises a set of selected grid nodes.

We now present a brief description of the G2N. Initially, the algorithm classifies the list of available grid nodes according to a decreasing order of throughput values. It then allocates all T tasks to the fastest node. The main program loop tries to reallocate, at least one task from the already allocated nodes to a new grid node (less performing next in line). If this succeeds, with a new query elapsed-time less than the previous one, it continues reallocating tasks to the new allocated grid node, until the overall elapsed-time becomes higher than the last computed one. Conversely, if the reallocation of a single task produces an execution

```
Scheduler (throughput(tp ,tp ,…, tp ),number-
tasks):result
nodelist:= descending order(throughput);
result:= result ∪ {nodelist(1)};
cost(1):= number-tasks * nodelist(1);
current-cost:=cost(1);
While (nodes in the list and add-new-node)
  total-cost:= current-cost;
  new-node:= next-node in nodelist;
  While (current-cost <= total-cost)
   move tuples from lowest node in result to new-node;
   Update costs of nodes and total-cost;
   If current-cost > total-cost
     If we could move at least 1 tuple to the new-node
           result:= result ∪ {new-node}
     else
           add-new-node:=false;
     Endif
     Stop loop;
   Endif
  endwhile
enwhile
output result;
```

Figure 10: The $G2N$ algorithm.

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 real-time values. Whenever the estimates fall bellow a certain threshold the QEM calls the Optimizer to re-evaluate 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.

Thus, we have a mechanism that allows initial allocation of a set of more efficient nodes for the execution of the total number of tasks and the re-evaluation of this allocation during the execution. Such mechanism allows a simple adaptation of the system to a dynamic and heterogeneous Grid environment which is one requirement for scientific visualization tasks in grid, according to the section 4.

6.3 Query Execution

Query execution in CoDIMS-G is implemented by an instance of the QEEF (*Query Engine Execution Framework*), 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 CoDIMS-G, the QEEF framework has been extended to support the following functionalities: user's functions execution, distribution and parallelism. The strategy for introducing user programs into a DQEP is to implement the Apply operator [Bouganim et al., 2001] as a new algebraic operator that encapsulates user's program invocation and parameters passing by value extracted from the 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. The user program result is concatenated to the input tuple to build an output tuple.

In order to support data distribution and operator parallelism, changes were necessary in the structure of QEEF. We added some communication mechanism among the different distributed fragments of a DQEP. In order to satisfy this constraint, control operators that manage the communication among logical operators were added to the interface of the execution machine and a unique identifier was associated to each operator. By doing this, an operator of a fragment may communicate to another, as long as it knows the service instance where the other fragment is being evaluated and the identifier of the remote operator.

The communication between nodes is implemented through the control operators: Sender, Receiver and Union. The Sender and Receiver [Kossmann, 2000] operators are responsible for providing transparency in distribution aspects such as communication technology and data (de)serialization. In the proposed 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. In case of any difference among the fragments' execution characteristics such as: data unicity and used control flow, other control operators have to be used. The Union operator is responsible for grouping the execution flow, which could be parallel or distributed. Threads are responsible for consuming the results of a producer and let them available in a buffer.

Figure 11 presents a CODIMS-G adaptive query execution plan generated for the Query 1 described by:

Select TP(p.part-id, p.point,Resulting-vector(p.point,g.polyhedron,g.velocity))

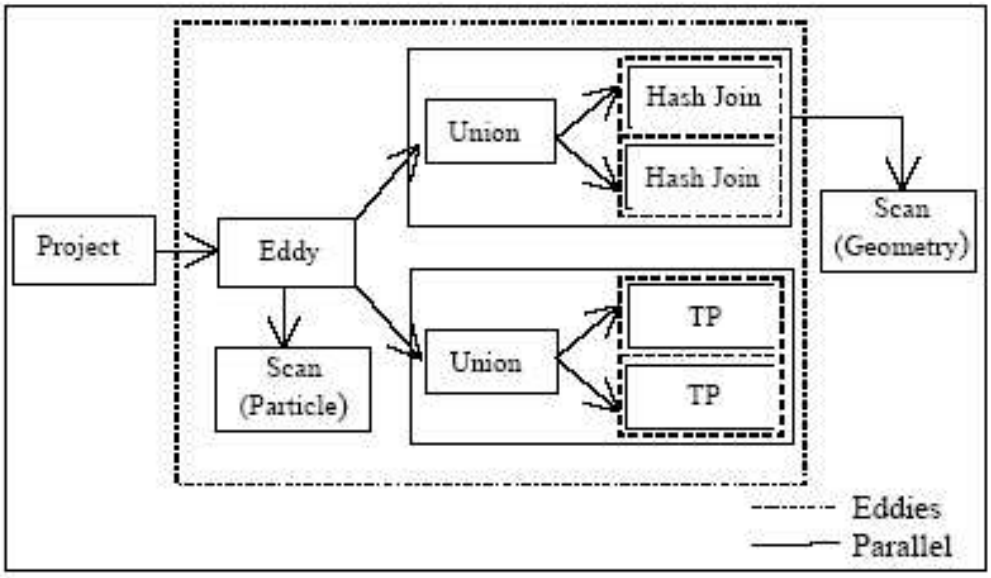

Figure 11: CODIMS-G Distributed query execution model.

From geometry g, particle p

Where p.point in g.plyhedron

The DQEP is described according to the syntax defined in [Ayres et al., pear], in which special execution strategies are defined by the two modules: Eddies and Parallel, in Figure 11. Eddies module controls the routing of tuples among DQEP physical algebraic operators. It controls the number of iterations that each tuple has gone through by storing an iteration value in the tuple. The Parallel module defines the query execution structure necessary for supporting the parallelism of the encapsulated algebraic operator. For example, in the upper box parallel module, the Hash Join is parallelized in two independent instances.

The results generated by the evaluation of a query may be consumed in two ways: first-tuple-first or last-tuple-first. In the former, the result is consumed as data is produced. In the later, the result is consumed only after terminating the query execution. In this mode the result is stored in a temporary area and the user will be noticed at the end of the processing. By submitting a query to the CoDIMS-G, the user will receive an identifier to the process and a handle to an instance of the service where to consume from.

7 CoDIMS-G Particle Tracing

In this section we sketch the extension of the CoDIMS environment to support the particles trajectory computing problem (TCP) in a grid architecture. It is a scientific visualization technique, called particle tracing, that can be thought as the trajectory of massless *virtual* particles upon a simulated flow.

The techniques in scientific visualization can be classified according to the data type they manage: Scalar fields ($F : D \subset \Re^3 \rightarrow \Re$), vector fields ($F(x)$ is a vector, $x \in D \subset \Re^3$) and tensor fields compose the usual range of data types in this field.

Henceforth, we have methods for scalar fields visualization (isosurface generation and volume rendering, colormap, etc.), vector fields visualization (field lines generation, particle tracing, topology of vector fields, LIC, among others) and techniques for tensor fields (topology and hyperstreamlines) [Rosenblum et al., 1994].

Once in computational grids the resources may be geographically-distributed, we must consider in this discussion the visualization techniques that can be implemented efficiently in distributed memory environments. As already pointed out in section 7, volume rendering is one of them. Another one is isosurfaces extraction [Chiang et al., 2001].

In volume rendering, the visualization model is based on the concept of extracting the essential content of a 3D data field by *virtual* rays passing the field [Rosenblum et al., 1994]. These rays can interact with the data according to artificial physical laws designed to enhance structures of interest inside. These laws can be summarized in a transport equation of the form:

$$\frac{dI}{ds} = -\sigma(s) I(s) + g(s), \tag{1}$$

where s is a distance over the ray, I is the scalar field to be visualized, σ is the extinction coefficient, and $g(s)$ represents generalized sources [Rosenblum et al., 1994]. Figure 12 pictures the basic idea behind the model.

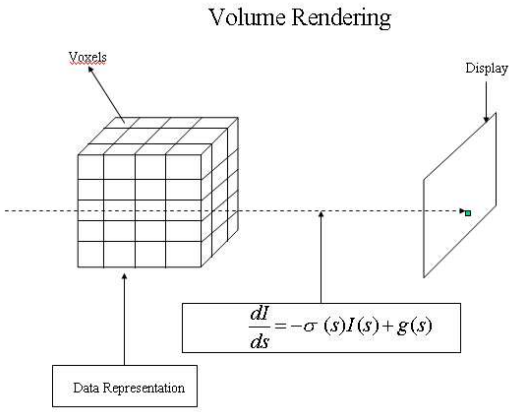

Figure 12: Volume Rendering: virtual rays passing the field, interacting with data and give the final image.

Isosurface extraction methods work differently [Rosenblum et al., 1994, Chiang et al., 2001]. Given a value q and a scalar field F, the isosurface q is defined as the set: $C(q) = \left\{x \in \Re^3, F(x) = q\right\}$.

As well as volume rendering, the result is a bi-dimensional image out of the three-dimensional data. However, in this case all data cells are first visited to identify cells that intersect the isosurface $C(q)$. Then, the necessary polygon(s) to represent the portion of the isosurface within the cell is generated and stored. Up to the end of this process, the obtained set of polygons gives a piecewise linear representation of the isosurface.

In volume rendering, each ray contribution can be computed independently (see Figure 12). The same is true for each portion of an isosurface. That is why both these methods can be efficiently implemented in distributed memory machines [Ma, 1995, Lombeyda et al., 2000, Rosenblum et al., 1994].

The implementation of the other methods cited, for distributed memory machines, depends on special considerations. In this paper we are interested on particle tracing methods [Lane, 1994, Rosenblum et al., 1994, Hamann et al., 1995, Stolk and van Wijk, 1992]. They can be mathematically defined by an initial value

problem [Rosenblum et al., 1994, van Wijk, 2002]:

$$\frac{dx}{dt} = F(x,t), \quad x(0) = P_0, \tag{2}$$

where $F : \Re^3 \times \Re_+ \rightarrow \Re$, is a time-dependent vector field (velocity, for example).

The solution of the above problem 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, etc.) through slight modification of the equation (2) [Rosenblum et al., 1994].

The problem (2) in general does not have analytical solution. Thus numerical methods must be used [van Wijk, 2002, Barnard et al., 1999]. These methods basically generate the particle trajectory step-by-step following some scheme for field interpolation inside cells [Hamann et al., 1995].

7.1 The Particle Trajectory Computing Problem

In this section we discuss a middleware architecture, based on CoDIMS-G for the TCP problem. In what follows, we compute particles trajectories by interpolating their positions through a path based on a velocity field associated to a Domain Decomposition Model. Information regarding particles initial position, domain decomposition (tetrahedron decomposition) and fluid velocity vectors are given to the problem [Stolk and van Wijk, 1992, Rosenblum et al., 1994]. These information are modelled 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 expressed as a *SQL-like* query embedded in a TCP procedure, such as:

```
Begin TCP procedure
for i=1 to number-iterations do
    Select trajectory(p.part-id, p.point, v.velocity)
        From Particles p, Geometry g, Velocity v
        Where  p.point matches i.point. and
                i.point = v.point and p. time-istant= 'ti'
endfor
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.

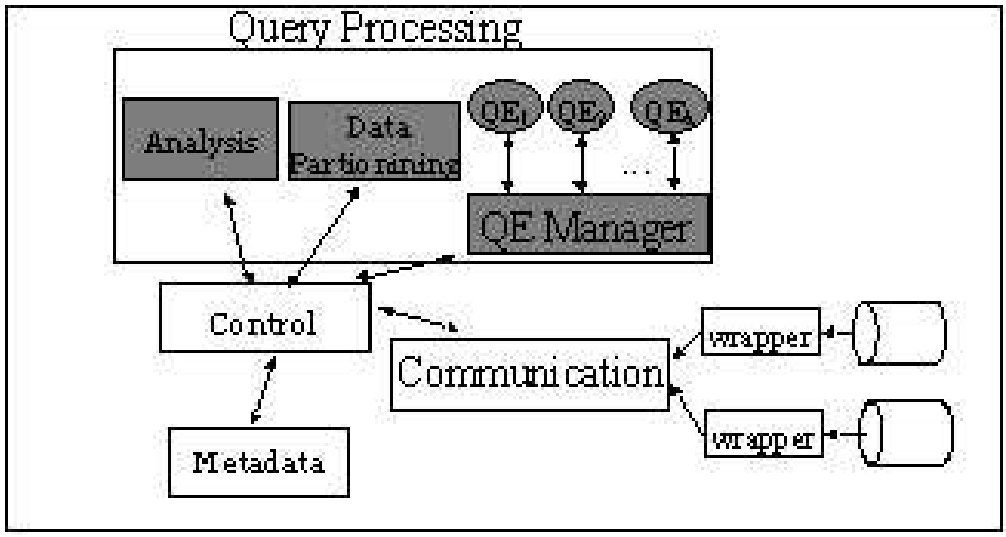

Figure 13: The TCP CoDIMS architecture.

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 (Globus toolkit 3) 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.

7.2 CoDIMS-G Configuration for Particle Tracing

In order to deal with the issues raised by the TCP class of applications within a grid environment we propose a CoDIMS-G configuration. The CoDIMS-G instance will comprehend the following components: Control, Metadata Manager, Query Processing, and Communication, see Figure 13. The Control, Metadata Manager and Communication components implement their traditional functionality, whereas the Query Processing component is adapted for supporting the TCP on a grid environment. It basically provides for parallel evaluation of particles trajectory within grid nodes. The middleware completely encapsulates the grid environment in respect to the visualization application. It provides: a set of adapted components, implemented as web services (http://www.w3.org/2002/ws/); flexible scheduling of services coordinated by the Control component and transparent access to data sources. The main functionality is implemented in the Query Processing component that is responsible for the efficient computation of particles trajectory. In the next section we detail the Query Processing component.

7.3 The Query Processing Component (QPC)

The TC problem once modelled as an SQL-type query and corresponding relations allows us to use query processing techniques in order to cope with issues raised in section 7.1. The grid environment can be seen as a loosely coupled query processing system for which parallel and distributed strategies have been developed. In particular, one may try to use parallel query processing techniques to deal with the issues

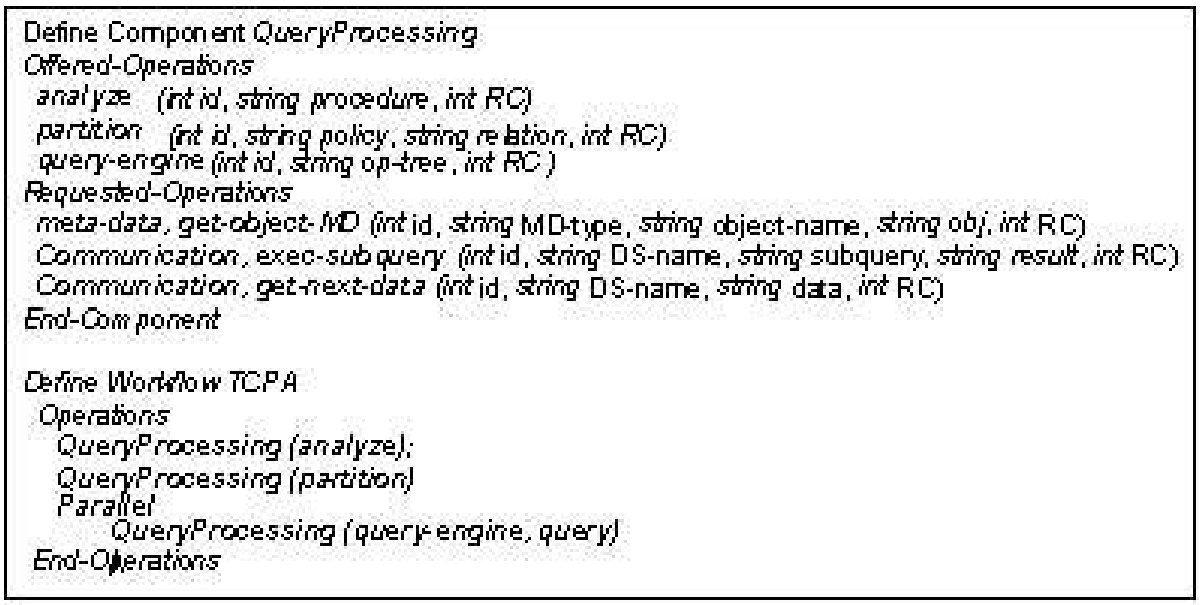

Figure 14: Physical and Logical Configuration.

presented in Section 7.1. Before describing the techniques proposed to parallel the TCP, we will describe the modules designed to integrate the Query Processing component for this CoDIMS instantion. The Query analysis module evaluates the syntactic and semantic correctness of user requests. Data partitioning receives a partitioning policy; retrieves the corresponding relation fragments and stores them in the designated node. Finally, the Query Processing module process a query execution plan (QEP). Figure 13 presents the configured system components: Control, Metadata Manager, Query Processing, and Communication.

On the top of Figure 14 we present the definition of the Query Processing component with its offered and requested operations, including their parameters. In the logical configuration we provide a workflow that defines services schedule. A workflow instance is loaded by the Control component according to user requests. Next it starts its execution following the order established by the workflow. On the bottom of Figure 14 we present a definition for the TCP workflow. Having given a general overview of the QPC component we present in more detail, in the next section, the Query Engine module.

7.4 The Query Engine Module

The process of computing particles trajectory is modelled through a parallel query execution plan, similar to the ones in traditional database systems [Silberschatz and Zdonic, 1997]. A query execution plan is evaluated by the query execution module. Query evaluation will include treatment for: data partitioning, data transfer and management of local memory. In respect to data distribution, a main concern is to efficiently partition and transfer the domain geometry dataset between grid nodes. On this matter, two main aspects are considered: the data volume initially transferred between the database and the processing nodes and the volume of message exchange during query evaluation. Secondly, regarding memory management we need also to consider out-of-core problems, i.e techniques to deal with data volumes that do not fit completely in main memory. This first prototype solution adopts the Spatial Hash-Join(SHJ) algorithm [Lo and Ravishankar, 1996] for supporting the evaluation of the contains user-defined function. The intuition is that we can replicate the whole domain geometry relation (Geometry) on each available node of the grid. The first step of the SHJ will spatially split the domain geometry in b buckets, each

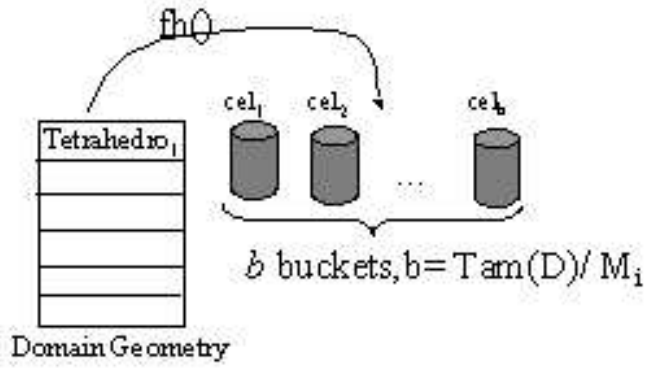

Figure 15: First step in the Spatial Hash-Join.

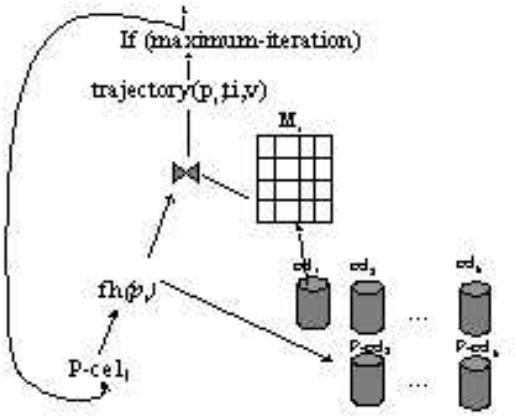

Figure 16: Spatial Hash-Join step 2.

measuring size(D)/Mi, where D is the size of the domain geometry representation in bytes and Mi is the size of available memory in node i (see Figure 15) minus a constant corresponding to the memory needed for query operators and processing tuples.

Once the domain has been entirely processed by step1, the first bucket is copied to memory and step 2 begins. In the second step, sets of particle tuple are read and placed into an input buffer. Each particle is then processed by the spatial hashing function according to its current position. If the produced bucket-id is in memory, the tuple is directly joined with the tetrahedrons in the corresponding cell. Otherwise, it is stored in a particle bucket for later evaluation, (see Figure 8). In Figure 16, we observe that the trajectory user function is treated as an operation in the query operator tree. An interesting aspect of this SHJ implementation is that a tuple produced by the join is returned to the input queue until it has passed through a specified number of iterations. This is not standard, considering traditional hash-join algorithm [Garcia-Molina et al., 2000], and may cause extra swapping of buckets to and from memory.

Our main motivation for adopting this strategy is to guarantee that a complete particle trajectory can be computed in a single node, eliminating inter-node communication. Furthermore, as a consequence, we can devise a partition policy regarding uniquely the number of particles to be processed by each grid node,

which is the basis for a balanced execution.

8 Conclusions

The fundamental point of this work is that we should consider scientific visualization in grid from an integrated global view of data and programs published by heterogeneous and distributed data sources. This can be addressed by CoDIMS-G , which is a middleware, based on grid services and distributed database technology, designed for the efficient evaluation of scientific queries over the Grid.

Our work is motivated by a particle path computation problem within the scientific visualization domain. The intuition is to build upon the experience on parallel and distributed query processing to offer efficient computation of scientific queries in the Grid environment. Particles trajectory path computation is modeled as a special case of the bag of tasks parallel type of problems, where for each time instant we need to compute a particle next position based only on its current position and a velocity vector in the flow. In particular, we are interested in how to parallel the computation of particles trajectory in grid nodes subject to: restrictions on the memory size of each node, in respect to the geometry data, and the high cost of inter-node message interchange during computation. A CoDIMS-G configuration for supporting the particle tracing in grid is being implemented at the National Laboratory for Scientific Computing.

References

[SEL, 1999] (1999). Le select: a middleware system for publishing autonomous and heterogeneous information sources. Technical report, http://www-caravel.inria.fr/ xhumari/LeSelect/.

[23, 2000] (2000). The garlic project. Technical report, http://www.almaden.ibm.com/cs/garlicl.

[A.C.P. Barbosa, 2002] A.C.P. Barbosa, F. Porto, R. N. M. (2002). Configurable data integration middleware system. In *Journal of the Brazilian Computer Society*, volume 8, pages 12–19.

[Avnur and Hellerstein, 2000] Avnur, R. and Hellerstein, J. (2000). Eddies: continuously adaptive query processing. *SIGMOD Record*, 29(2):261–272.

[Ayres et al., pear] Ayres, F., Porto, F., and Melo, R. (2005 (To Appear). An extensible query execution engine for supporting new query execution models. In *LNCS*.

[Barnard et al., 1999] Barnard, S., R.Biswas, Saini, S., der Wijngaart, R. V., Yarrow, M., and Zechtzer, L. (1999). Large-scale distributed computational fluid dynamics on the information power grid using globus. In *The 7th Symposium on the Frontiers of Massively Parallel Computation*, http://www.ipg.nasa.gov.

[Berman et al., 2002] Berman, F., Fox, G. C., and Hey, A. J. G. (2002). John Wiley & Sons Inc.

[Bethel and Shalf, 2002] Bethel, E. W. and Shalf, J. (2002). Cactus and visapult: An ultra-high performance grid-distributed visualization architecture using connectionless protocols. Technical report, http://www-vis.lbl.gov/Publications/2002/LBNL-50237-CactusVisapult.pdf.

[Bethel and Shalf, 2003] Bethel, E. W. and Shalf, J. (2003). Grid-distributed visualizations using connectionless protocols. *IEEE Comput. Graph. Appl.*, 23(2):51–59.

[Bouganim et al., 2001] Bouganim, L., Fabret, F., Porto, F., and Valduriez, P. (2001). Processing queries with expensive functions and large objects in distributed mediator systems. In *ICDE*, pages 91–98, http://computer.org/proceedings/icde/1001/10010091abs.htm.

[Chiang et al., 2001] Chiang, Y.-J., Farias, R., Silva, C., and Wei, B. (2001). A unified infrastructure for parallel out-of-core isosurface and volume rendering of unstructured grids. In *IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*.

[DeFanti and Brown, 2001] DeFanti, T. and Brown, M. (2001). Report to the national science foundation directorate for computer and information science and engineering (cise) advanced networks infrastructure & research division. Technical report, http://www.evl.uic.edu/activity/NSF/final.html.

[Domain, 2003] Domain, W. A. (2003). Web services. http://www.w3.org/Architecture/#ws.

[Elmasri and Navathe, 1994] Elmasri, R. and Navathe, S. B. (1994). *Fundamentals of Database Systems, 2nd Edition.* Benjamin/Cummings, DBLP, http://dblp.uni-trier.de.

[et al., 1981] et al., D. D. C. (1981). A history and evaluation of system r. *Communications of the ACM*, 24(10).

[Fankhauser et al., 1998] Fankhauser, P., Gardarin, G., and Lopez, M. e. a. (1998). Experiences in federated databases: From iro-db to miro-web. In *Proceedings of the 24th VLDB Conference*.

[Fernandez et al., 1997] Fernandez, M., Florescu, D., Kang, and et al., J. (1997). Strudel: A web-site management system. In *ACM SIGMOD International Conference on Management of Data*.

[Fontes et al., 2004] Fontes, V., Schulze, B., Dutra, M., Porto, F., and Barbosa, A. (2004). Codims-g: a data and program integration service for the grid. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 29–34, http://doi.acm.org/10.1145/1028493.1028498. ACM Press.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed system integration. Technical report, www.globus.org/research/papers/ogsa.pdf.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Addison-Wesley.

[Garcia-Molina et al., 2000] Garcia-Molina, H., Ullman, J., and Widom, J. (2000). *Database System Implementation*. Prentice-Hall.

[Giraldi et al., 2003a] Giraldi, G., Oliveira, J., Schulze, B., and Silva, R. (2003a). Distributed visualization of fluids using grid. In *Proc. Of the International Middleware Conference (MGC).*

[Giraldi et al., 2003b] Giraldi, G. A., Oliveira, J., Schulze, B., and Silva, R. (2003b). Distributed visualization of fluids using grid. In *In Proc. of the First International Workshop on Middleware for Grid*.

[Group, 2002] Group, O. M. (2002). Omg common request broker: Architecture and specification. http://www.omg.org.

[Group, 2004] Group, T. O. M. (2004). The common object request broker architecture and specification. http://www.corba.org.

[Hamann et al., 1995] Hamann, B., Wu, D., and II, R. J. M. (1995). On particle path generation based on quadrilinear interpolation and bernstein-bzier polynomials. *IEEE Trans. on Visualization and Comp. Graph.*, 1(3).

[Hansen and Johnson, 2003] Hansen, C. and Johnson, C. (2003). Guest editors' introduction: Graphics applications for grid computing. *IEEE Comput. Graph. Applications. Special Issue.*, 23(2):20–21.

[Jankun-Kelly et al., 2003] Jankun-Kelly, T. J., Kreylos, O., Ma, K.-L., Hamann, B., Joy, K. I., Shalf, J., and Bethel, E. W. (2003). Deploying web-based visual exploration tools on the grid. *IEEE Comput. Graph. Appl.*, 23(2):40–50.

[Jankun-Kelly and Ma, 2001] Jankun-Kelly, T. J. and Ma, K.-L. (2001). Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287.

[Kossmann, 2000] Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Survey*, 32(4):422–469.

[Kranzlmuller et al., 2003] Kranzlmuller, D., Heinzlreiter, P., and Volkert, J. (2003). Grid-enabled visualization with gvk. In *In Proc. of the 2003 Annual Crossgrid Project Workshop & 1st European Across Grids Conference*.

[Kranzlmller et al., 2002] Kranzlmller, D., Kurka, G., Heinzlreiter, P., Volkert, J., and Arabnia, H. (2002). A grid middleware extension for scientific visualization. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, http://www.gup.uni-linz.ac.at/ gk/docs/pdpta02b.pdf.

[Lane, 1994] Lane, D. (1994). Parallelizing a particle tracer for flow visualization. Technical report, http://www.nas.nasa.gov.

[Lo and Ravishankar, 1996] Lo, M. and Ravishankar, V. (1996). Spatial hash-joins. In *ACM-SIGMOD Intl. Conf. On Management of Data*, Montreal, Canada.

[Lombeyda et al., 2000] Lombeyda, S., Aivazis, M., and Rajan, M. (2000). Making remote tools available for visualization of large data sets: Parallel isosurface calculation and rendering of large data sets in iris explorer. In *Visualization Development Environments 2000 Proceedings*.

[Ma, 1995] Ma, K. L. (1995). Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. In *IEEE Parallel Rendering Symposium*, pages 23–30.

[Martinez and Roussopoulos, 2000] Martinez, M. and Roussopoulos, N. (2000). Mocha: A self-extensible database middleware system for distributed data sources. In *ACM SIGMOD International Conference on Management of Data*.

[Max, 1995] Max, N. (1995). Optical models for direct volume rendering. *IEEE Trans. on Visualization and Comp. Graph.*, 1(2):99–108.

[Mecca et al., 1998] Mecca, G., Atzeni, P., Masci, A., Merialdo, P., and Sindoni, G. (1998). The araneus web-base management system. In *ACM SIGMOD International Conference on Management of Data*.

[Molina et al., 1997] Molina, H. G., Hammer, J., and Ireland, K. e. a. (1997). Integrating and accessing heterogeneous information sources in tsimmis. *Journal of Intelligent Information Systems*, 8(2).

[Nierstrasz and Dami, 1995] Nierstrasz, O. and Dami, L. (1995). Prentice-Hall Inc.

[Norton and Rockwood, 2003] Norton, A. and Rockwood, A. (2003). Enabling view-dependent progressive volume visualization on the grid. *IEEE Comput. Graph. Appl.*, 23(2):22–31.

[of Liverpool,] of Liverpool, U. E-science home page. Technical report, http://www.liv.ac.uk/HPC/.

[Porto et al., 2004] Porto, F., Giraldi, G., Oliveira, J., Silva, R., and Schulze, B. (2004). Codims - an adaptable middleware system for scientific visualization in grids. *Concurrency and Computation: Practice and Experience*, 16(5).

[Rosenblum et al., 1994] Rosenblum, L., Earnshaw, R., Encarnacao, J., Hagen, H., Kaufman, A., Klimenko, S., Nielson, G., Post, F., and Thalmann, D. (1994). *Scientific Visualization: Advances and Challenges*. Academic Press.

[Schulze and Madeira, 2004] Schulze, B. and Madeira, E. R. M. (2004). Grid computing and active services. *Concurrency and Computation: Practice and Experience*, 16(5).

[Silberschatz and Zdonic, 1997] Silberschatz, A. and Zdonic, S. (1997). Database systems - breaking out the box. *SIGMOD Record*, 26(3).

[Silva et al., 2004] Silva, R., Rodrigues, P., Oliveria, J., and Giraldi, G. (2004). Augmented reality for scientific visualization: Bringing datasets inside real world. In *In Proc. of the Summer Computer Simulation Conference (SCSC)*, http://www.lncc.br/ jauvane/papers/SCSC2004.pdf.

[Singh, 2003] Singh, G. (2003). Unlocking the grid. *IEEE Comput. Graph. Appl.*, 23(2):4–5.

[Stolk and van Wijk, 1992] Stolk, J. and van Wijk, J. J. (1992). Surface-particles for 3d flow visualization. In Post, F. H. and Hin, A. J. S., editors, *Advances in Scientific Visualization*, pages 119–130. Springer, Berlin, Heidelberg.

[Suzuki et al., 2003] Suzuki, Y., Sai, K., Matsumoto, N., and Hazama, O. (2003). Visualization systems on the information-technology-based laboratory. *IEEE Comput. Graph. Appl.*, 23(2):32–39.

[Tomasic et al., 1998] Tomasic, A., Raschid, L., and Valduriez, P. (1998). Scaling access to heterogeneous data source with disco. *IEEE Transactions on Knowledge and Data Engineering*, 10(5).

[Trott et al., 1996] Trott, A., Moorhead, R., and McGinley, J. (1996). Wavelets applied to lossless compression and progressive transmission of floating point data in 3-D curvilinear grids. In Yagel, R. and Nielson, G. M., editors, *IEEE Visualization '96*, pages 385–388.

[van Wijk, 2002] van Wijk, J. J. (2002). Image based flow visualization. *ACM Transactions on Graphics (TOG)*, 21(3):745–754.