Data transport between visualization web services for medical image analysis

Koulouzis, S.; Zudilova-Seinstra, E.; Belloum, A.

DOI
10.1016/j.procs.2010.04.194

Publication date
2010

Document Version
Final published version

Published in
Procedia Computer Science

Citation for published version (APA):
Abstract

With the rapid development of IT data driven experiments and simulations have become very advanced and complicated. The amount and complexity of scientific data increase exponentially. Nowadays the challenge is to efficiently support scientists who generate more data than they can possibly look at and understand. This requires not only high-performance visualization and feature extraction techniques but also efficient and intuitive management of available data and underlying computational resources. Consequently, Service Oriented Architectures and workflow management systems are becoming popular solutions for the deployment of e-Science Infrastructures aimed to assist in exploration and analysis of large scientific data. In this paper, we compare two transport models of workflow execution for data-intensive medical visualizations that rely on web services. The image-based analysis of vascular disorders served as a case study for this project. We applied a service oriented approach to construct distributed visualization pipelines, which allow visualization experts to develop visualizations to view and interact with large medical data sets. Moreover, end-users (i.e., medical specialists) can explore these visualizations irrespective of their geographical location and available computing resources. The paper reports on the current implementation status and presents our main findings.

Keywords: Service-oriented visualization, Web services, Data Transfers, Image-based medical analysis

1. Introduction

With the fast development of information technologies, the amount of data at one’s disposal is enormous. Data is produced at different scales, arrives from various sources (sensors, instruments, simulations, databases, direct manipulations, etc.) and has different structures. The data explosion has led to very large detailed data sets. Sheer sizes, large-scale multi-regional and multi-institution collaborations have resulted in the distribution of data around the world. The greatest challenge today is to efficiently support the community of researchers who generate more data than they can possibly look at and understand [1].

Clinical data-driven experiments and simulations have become very advanced and complicated. For instance, manual tracing of structures such as the vascular system, which may appear in more than 1000 images in a single...
Magnetic Resonance Angiography (MRA) or Computed Tomography (CT) imaging study, requires both considerable expertise and a great deal of time. Hence, processing, visualization and integration of large medical data sets are the major cornerstones of the modern Healthcare [2]. e-Science can assist in coping with the challenges presented by the creation, management and analysis of these large data sets. Thus, e-Science has been introduced as a paradigm that promotes collaborative and interdisciplinary research [1, 3]. With the use of Service Oriented Architectures (SOA), concepts of distributed processing, and data access, have scaled to new levels.

Adopting SOA provides new challenges and opportunities for data delivery and information management. Connectivity between distant locations, interoperability between different kinds of systems and resources and high levels of computational performance are some of the most promising characteristics of this architecture. Taking advantage of these benefits, we have developed a visualization framework based on the service oriented model introduced in [4]. This model suggests that any intermediate visualization sub-process is potentially transformable into an independent web service. The granularity offered by this model, enables users to compose and execute distributed visualization pipelines, including those running on heterogeneous distributed resources. The image-based analysis of vascular disorders served as the case study for this project. We experimented with scanned and simulated medical data.

2. Background

Vascular diseases are among the leading causes of death and disability all over the world, especially in developed countries. In general, vascular diseases fall into two main categories [5]: aneurisms and stenosis. An aneurysmal disease is a balloon like swelling of the artery. Stenosis is a narrowing of the artery. To redirect the blood flow or to repair the weakened artery, a vascular reconstruction procedure can be applied. A criterion for the success of a vascular procedure is the normalization of the blood flow in the affected area. However the best treatment is not always obvious because of the complexity of the human vascular system and because of other diseases that a patient may have.

Numerical simulation of the blood flow in human vascular structures helps to obtain an extensive knowledge about its behavior and to develop solutions for the treatment of vascular disorders [6]. Sometimes simulation data sets are so big and complex that they cannot be easily interpreted. Instead, visualizations of such data can reveal relevant information without loosing the overall perspective. However, the location of these data sets is often distributed. This with the combination of limited local resources (hardware, software, etc.) creates difficulties to researchers for easy development, sharing and execution of their visualizations.

e-Science focuses on creating a collaborative research environment where resources can be shared at a global scale. This enables research to cross and combine many disciplines, in order to address and solve more complex problems. Grid computing as the (e)investiture of e-Science provides excellent opportunities in this respect [7, 8], especially in recent years when this paradigm has shifted towards SOA [9]. This has motivated several attempts to build Grid enabled visualization frameworks. For instance, the RealityGrid [10] project used visualization as part of a bespoke application to visualize the output of simulations running on the Grid. In the gViz project [11], an extension to the NAG IRIS Explorer [12] was developed. Thanks to this extension, individual IRIS Explorer visualization modules can be prepared and executed using remote Grid resources. The Grid Visualization Kernel (GVK) developed as part of the CrossGrid project is a middleware extension built on top of the Globus Toolkit. GVK allows remote interactive visualization of both original medical data and acquired simulation results [13, 14].

Unfortunately, existing frameworks have drawbacks caused by the visualization models they support. None of the traditional visualization models (e.g., visualization cycle model [15], visualization pipeline model [16], data-pull models [17], reference visualization models [18, 19], etc.) fully comply with the needs and requirements of distributed computing as they are based on the major linkage of visualization sub-processes. In the Grid environment, this linkage often results in underutilization of the used computing resources, which are often requested for a period of time which is longer than the actual usage [20, 21]. Also, to allow interactive user steering over the intermediate visualization stages, external monitoring tools need to be developed. When frequently invoked, these tools can significantly slowdown the visualization process [11, 22].
3. A Service-Oriented Approach to Data Visualization

To better utilize the available distributed resources and provide users with an easier and more flexible way of visualizing medical data sets, we have opted for a SOA approach. SOA refers to a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services [23]. This approach provides some benefits, namely: i) **Scalability**: Services can be added and removed in order to meet the demands of the application. ii) **Reliability**: Provided that services maintain a standard interface new or updated implementations can be introduced, without disrupting the functionality of any workflow or application. iii) **Fault tolerance**: If a service becomes unavailable for any reason, clients can still query for alternate services that offer the same functionality. iv) **Reusability**: A service can be part of more than one workflow or application, as it can interact with multiple clients. Moreover, because web services provide a descriptive interface –called Web Service Description Language (WSDL)–, this makes them ideal for usage in workflow management systems. Thanks to these workflow management systems, end-users can easily create visualizations that use distributed data sets and resources.

Considering these benefits, we applied a service-oriented visualization model presented in [4]. This model originates from the traditional visualization pipeline model of Haber and McNabb [16] and is based on the principle that any visualization sub-process (i.e., reading, filtering, mapping and rendering) can be transformed into an independent service [24]. In the architecture shown in Figure 1, each visualization service performs a specific role and can be executed on a different logical machine at a different geographical location. Therefore, thanks to the granularity of SOA, services can be assigned to different resources based on their complexity and computational demands. For instance, the rendering service\(^2\) can be provided with a more significant computational power than the mapping service responsible only for the conversion of the filtered data into geometrical primitives.

Interactive execution of workflows plays an important role in service oriented visualization. This can be achieved by means of workflow orchestration, where control over each visualization sub-process is centralized in the workflow engine\(^3\) to allow a better configuration and control over the visualization pipeline, as well as error handling. In addition, the amount of data that needs to be analyzed by visualization services, and thus to be transferred from one service to another, might cause significant time loss with respect to the overall workflow execution.

4. A Visualization Case Study

In this project, we focused on the visualization of two different types of medical data: experimental and simulated. The experimental data about the patient’s vascular condition was be obtained via Magnetic Resonance Imaging (MRI), and then converted to the Visualization Toolkit (VTK) format. To simulate the microscopic properties of the blood flow, while exploring its macroscopic properties, the lattice-Boltzmann method (LBM) was applied. The main idea behind LBM is mapping the average motion of the fluid/blood particles on the lattice [5].

In this project, we chose the iso-surface extraction\(^4\) technique to visualize data of the patient’s vascular condition. To represent the simulated blood flow in a selected region of interest, we used streamline-based flow visualization (Figure 2-c-d). Streamlines represent a set of paths traced out by massless particles as they move within a flow [25, 26].

\(^2\)It takes the geometry generated by the mapping stage of the visualization pipeline and renders it into the final image.

\(^3\)A set of higher level clients, able to coordinate the invocation of web services

\(^4\)Iso-surface extraction allows the representation of human vascular structures as surfaces, which are generally constructed using polygons as primitives (see Figure 2-a)
Using streamlines, the user (e.g., medical specialist) can investigate how the flow curves and whether it diverges or converges at specific points. To make it easier for users to analyze the behavior of the blood flow, streamlines can be displayed in combination with the patient’s data where streamlines are positioned inside the arterial structures as it is shown in Figure 2-e-f.

5. Visualization Services

Based on our previous work [4] we developed a set of web services using the VTK. These web services allow the construction of three visualization pipelines suitable for our case study described in Section 4. The first pipeline is the iso-surface extraction pipeline shown in Figure 3(a). As can be seen, the iso-surface extraction pipeline starts with the data reading sub-process handled by the `vtkStructuredPointsReader` class. Further on, the pipeline contains the `vtkPolyDataMapper` class that maps polygonal data (passed by `vtkContourFilter`) to graphics primitives, and finally the `vtkRenderer` class responsible for rendering graphics primitives into the final image (3D arterial structures). The second pipeline is a streamline-based flow visualization pipeline (Figure 3(a)). It also starts with the reading sub-process handled by the `vtkStructuredPointsReader` class. The derived data object is then passed to `vtkStreamLine` filter to generate streamlines. The position of each streamline has to be properly defined via the `vtkPointSource` class which generates random points in a sphere space with a specified center, radius and number of seed points. The `vtkTubeFilter` is a filter that can be applied additionally to generate tubes wrapped around streamlines. The `vtkLookupTable` class is used to apply the color palette to streamtubes. Finally, the pipeline contains `vtkPolyDataMapper` and `vtkRenderer`, which serve the same purpose as explained for the iso-surface extraction pipeline. In this project, we experimented with the blood flow velocity values.

In total, three visualization services have been developed in this project: (i) the `ReadVTK` service responsible for reading and filtering sub-processes in the iso-surface extraction pipeline as well as in the streamline-based flow visualization pipeline; (ii) The `StreamVTK` service responsible for the filtering sub-process in the streamline-based flow.
Figure 4: Service-oriented visualization pipelines: (a) iso-surface extraction pipeline; (b) streamline-based visualization pipeline; (c) combined visualization pipeline.

Figure 5: Data transport models: (a) centralized — web services exchange data through a central repository; (b) distributed — web services exchange directly with each other.

In addition to the iso-surface extraction pipeline (Figure 4(a)) and the streamline-based flow visualization pipeline (Figure 4(b)) explained earlier, the aforementioned visualization web services can also be used to construct the combined visualization pipeline (Figure 4(c)).

6. Data Transport of Medical Data

As mentioned earlier, distributed visualization pipelines can potentially analyze and generate large amounts of data. It is therefore required to efficiently transport data between visualization services. In this project, in order to increase the efficiency of distributed visualization pipelines, we identified two models for delivering data to each service. Each model uses SOAP to transfer references, instead of the actual data, because SOAP is inefficient for transporting large data sets [27]. Additionally, using references in an orchestration workflow allows the workflow engine to maintain full control over the data flow, while controlling the application logic. This approach enables users to employ conditional elements in their workflow execution. In this way, visualization parameters can be passed to web services allowing a flexible configuration of a visualization pipeline. For the first data transport model shown in Figure 5(a), we assumed that medical data are located in a central data repository, and all services transfer their results to this location. This model should be reliable and able to support large medical data sets. The second model shown in Figure 5(b), adopts a distributed approach. Intermediate temporary data are delivered directly from the producing to the consuming services. This approach can speed up data driven workflows, while at the same time it relieves data resources from unnecessary storage requirements.

Both models were implemented the use of the ProxyWS. The ProxyWS is a data-aware web service able to coordinate the transportation of large data volumes from a variety of remote locations (GridFTP [28], LFC [29], SRM [30], etc.), connect web services via direct streaming and also provide support for large data transfers to existing or “legacy” web services.
The ProxyWS is designed to be a non-intrusive, portable web service that can transport large data sets between web services, while taking care that results produced are not returned via SOAP. The main components that enable the design to facilitate large data transfers are, the Virtual Resource System (VRS), the WSDL interface and the VRSServer.

More specifically, VRS is a Java API used by the VBrowser [31] that enables homogeneous access to multiple resources (services, data streams, file systems, etc.). It adopts a layered architecture, where at the top level the VRS defines a resource abstraction seen as a handler for referencing resources like files and services. The next levels, the API defines resources such as file systems (local file systems, GridFTP, LFC, SRM, etc.), HTTP(S) streams, OGSA services [8], and web services. Thus this layered implementation adopted by the VRS enables a uniform access to multiple data resources just by providing their URI. These features enable the ProxyWS to access different data resources in a uniform fashion, thereby offering web services (legacy or new) access to those resources.

Next, the WSDL used by the ProxyWS exposes a set of methods that support data transports for legacy web services (Figure 6(a)). The most important are: i) callService: This method invokes a web services located in the same container as the ProxyWS, and returns to the caller a URI referring to the data produced. ii) getUploadURI: To be able to bring data to the consuming web services, the ProxyWS can provide multiple URIs that are used to transfer data (i.e., HTTP or GridFTP). This gives to a client the chance to decide which is more suitable. iii) getResourceURI: This method can reference any resource, local or remote, in any of the available protocols (i.e., a local file or a data stream to an HTTP URL, or a remote GridFTP file to an HTTP URL).

Besides using the ProxyWS as a service for making proxy calls, the ProxyWS can be used as an API for developing web services capable of exchanging large data sets. The effort required for such a task is minimal as it only involves the implementation of a method to obtain URI for data transfer. Figure 6(b) shows the architecture of such a web service.

Finally the VRSServer is a simple interface that enables web services to stream data to other resources independently from a specific infrastructure (GridFTP, etc.). An implementation of that is the HTTPTransport component, which from the web service’s point of view only provides secure I/O streams to data resources, or other web services. More specifically the HTTPTransport is able to directly stream data produced by the ProxyWS or any file, irrespective of whether it is local or remote.

---

A web service container is a SOAP engine i.e. Axis

In this case this is achieved by HTTPS, which provides reasonable protection from eavesdroppers and man-in-the-middle attacks.
8. Experimental Setup

In order to compare advantages and disadvantages of the data transport models described in Section 6, we implemented two workflows. The first workflow shown in Figure 7(a) adopts the centralized data flow model, while the second shown in Figure 7(b) uses the distributed model. More specifically the centralized data workflow supports the following scenario:

1) The workflow engine invokes the first ReadVTK service to read the artery data set from a remote location (a GridFTP server). 2) The first ReadVTK service downloads and reads the artery data set, the location (memory or local file system) of which is returned back to the workflow engine. 3) The workflow engine provides the first ReadVTK service with the remote URI (the GridFTP server) to upload the results. 4) The same procedure takes place with the second ReadVTK service, but with the blood flow field data instead. 5) The workflow engine calls the StreamVTK web service to calculate streamlines for the flow field data, by providing the URI of the remote server, where the simulated data are located. 6) The result of the previous step is uploaded to the remote server. 7) The workflow engine calls the rendering service by providing the URIs of the artery and flow field (that now is mapped to streamlines) polydata. As soon as these polydata have been rendered, the resulting image is uploaded again to the remote server. 8) Finally, the user can inspect the generated image from the remote server.

Alternatively, the distributed data workflow shown in Figure 7(b), is executed as follows: 1) The workflow engine invokes the first ReadVTK service to read the experimental data set from a remote location (a GridFTP server), which returns a URI (local file system) where the result is saved. 2) The same procedure is followed for the second ReadVTK service. 3) In order to calculate streamlines for the flow field data set, the workflow engine invokes the StreamVTK service to obtain the data from the second ReadVTK service. 5) As soon as both data sets have been read, filtered, and mapped, the workflow engine calls the RenderVTK service to render the final image. 6) After the RenderVTK has generated the final image, the user can inspect it directly from the location of the RenderVTK service.

These two workflows where compared in terms of total execution time, using different configurations and data sets. Thus for each workflow three different data sets were considered, i.e. 1.9 MB, 30.1 MB and 66.7 MB. For each of these data sets, several ratio values were defined to generate different number of seed points for the streamlines to be rendered: 300, 150, 110, 50 and 30. So for both workflows, it was expected that the longest execution time would be for the 66.7 MB data set with the ratio of 30 (i.e., every 30th point in a data set will be used as a seed point). Firstly, because larger files take longer to process (read, calculate streamlines and render) and secondly because the number of streamline increases with the low ratio value resulting in the bigger output file generated by the StreamVTK service. The last affects the execution time of the RenderVTK service, since it will have to render a much more detailed representation of the blood flow field.

Additionally, all services used in both workflows were hosted on different machines. These machines have an Intel Dual-Core E2180 CPU at 2GHz and 2 GB of RAM, while the GridFTP server has an Intel Quad-Core E5450 CPU at 3GHz and 8GB of RAM. Furthermore, all the machines are part of the same network. This however did not affect the nature of measurements, as both workflows were compared under the same circumstances, and the main of focus was...
Figure 8: Rendered artery and flow field data. Starting from the top left, the streamlines ratios used are 300, 150, 110, 50, and 30, while the last image (bottom right), shows the artery with a ratio of 2.

Figure 9: Workflow execution times for the GridFTP and ProxyWS workflows. The differences in workflow implementation, as well as the data flow models used.

9. Results and Discussion

Figure 8 shows the different rendered images obtained for ratios specified for the 66.7 MB flow field data set. These results show that as the ratio value decreases the number of visualized streamlines increases, which also implies the increase in computation and storage demands.

Figure 9 shows the results obtained for the centralized and distributed data workflows when executed with the data sets and configuration mentioned in Section 8. From these results, we conclude that by transferring data directly to the consuming service execution time can be shortened. This is especially true, as data and computations become more intensive. So in the case when the flow field data set is 66.7 MB and the streamlines ratio is 30, the time difference between the two methods (GridFTP and ProxyWS) is about 35.14 sec.

Providing a more detailed look at the previous results, Figure 10 shows a breakdown of the total execution time for both workflows for the 66.7 MB data set (the left part represents the time break down for the ProxyWS workflow, while the right part for the GridFTP workflow). To obtain this graph, we measured the individual steps of the two workflows. The total execution time is broken down as follows: i) Read time, representing the time each workflow needs to access the data repository and to read the raw data (experimental and/or simulated). ii) Process time, which
Figure 10: Breakdown of execution time in read, process, render and postage time, for both workflows while visualizing the 66.7 MB data set. The left part of the graph shows the time breakdown for the ProxyWS workflow while the right part shows the GridFTP.

is the time it takes to get the data (flow field) from the previous step and calculate streamlines for it. iii) Render time, representing the time it takes for each data set to be converted into an image, and finally iv) Postage time, which is the time it takes for the image to be transferred to the user.

Apart from the read time, which is almost the same for both workflows, the two other metrics (process and postage times), show a clear advantage when considering the ProxyWS workflow. This is because for the GridFTP workflow all services need to upload their results back to the GridFTP server, before the workflow can proceed to the next call. Another interesting finding shown in Figure 10 is the tendency of rendering time to increase non-linearly depending on processing intensity (i.e. more streamlines). This observation reveals potential scaling problems, but the nature of SOA should make it easy to cope with this problem. More specifically, we can assign more resources to rendering services allow the architecture to scale better. So, for example, the rendering service may be the front-end of a multiprocessor system. This shift will not disrupt the workflow itself, since SOA separates the implementation of a service from its access interface (WSDL).

10. Conclusions and Future Work

Traditional visualization models do not fully exploit features and benefits of distributed computing oriented mostly towards SOA. Having this in mind, we took advantage of the benefits that the SOA model provides to construct distributed visualization pipelines.

The image-based analysis of vascular disorders served as a case study for our research. We experimented with two different types of medical data: the experimental data of the patient’s vascular condition and the simulated blood flow data, which can be used to predict the outcome of a vascular reconstruction procedure.

We used service orchestration for the flow control, which allows interactive user steering over distributed visualization services via workflow management systems. We compared two data flow models: a centralized, where data are located in a centralized data repository and a distributed where data are delivered directly from producing to consuming services. For distributed data, the flow model provided by the ProxyWS was able to speed up the workflow execution, while visualization services could remain independent from a specific data infrastructure, which increases their portability. In the case when data had to be accessed and/or archived from remote data repositories, the ProxyWS provides a uniform and simple way to read data from such locations via the VRS.
To be able to even further speed up the execution of visualization pipelines, it would be desirable to be able to directly connect the visualization services with data streams. This could be achieved with the use of the VtkSocketCommunicator. This approach however might be difficult to implement due to firewall restrictions. Additionally having identified a potential scaling problem for rendering, it would be desirable to parallelize this process. The last will not affect the current workflow implementation, as the service interface (WSDL) will remain unchanged.

References

URL http://dx.doi.org/10.1109/38.314622