Image Processing for the Grid: A Toolkit for Building Grid-enabled Image Processing Applications.*

Shannon Hastings, Tahsin Kurc, Stephen Langella, Umit Catalyurek, Tony Pan, and Joel Saltz
Department of Biomedical Informatics
The Ohio State University
Columbus, OH, 43210

Abstract

Analyzing large and distributed image datasets is a crucial step in understanding the structural and functional characteristics of biological systems. In this paper, we present the design and implementation of a toolkit that allows rapid and efficient development of biomedical image analysis applications in a distributed environment. This toolkit employs the Insight Segmentation and Registration Toolkit (ITK) and Visualization Toolkit (VTK) layered on a component-based framework. We present experimental results on a cluster of workstations

1 Introduction

Rapidly advancing imaging sensor technologies have made it possible for researchers in the physical and biological sciences to observe biological systems and measure their structural and functional characteristics at great resolutions. The anatomic structures described by the imagery vary in scale from macroscopic (e.g., radiology studies) to microscopic (virtual slides from biopsies). The imagery can be two or three-dimensional; the images can be static or time dependent (e.g., functional MR, follow-up imaging studies for treatment assessment). There is an increasing recognition of the need to support the information service needs posed by overlapping collections of work groups and organizations. Multi-institutional collaborative biomedical research studies involve the need to pool and support distributed analysis of imaging information, in addition to epidemiological, clinical, and laboratory information. While a passive viewing of information can be achieved by the traditional imaging studies, the capability to reconstruct and analyze the data is needed for effective analysis of biological structures. In order to realize this capability, software tools are needed for acquisition, storage, and interactive manipulation in distributed environments of large datasets generated by measurements or computational models.

In this paper we present a framework for developing image analysis applications in a Grid environment. Within this framework, we describe the design and implementation of the Image Processing for the Grid (IP4G) middleware. The goal of this middleware is to enable rapid and efficient implementation of image analysis methods in a distributed storage and computational environment. We address this goal by layering two widely used image analysis and visualization toolkits, namely the Insight Segmentation and Registration Toolkit (ITK) from National Library of Medicine [11] and Visualization Toolkit (VTK) [16], on a component-based infrastructure called DataCutter [3].

2 Motivating Applications

In this section we present two application scenarios and discuss their requirements. These scenarios are based on the applications we are developing in collaboration with Radiology at OSU and the Telescience and BIRN group at National Center for Microscopy and Imaging Research (NCMIR).

Dynamic Contrast Enhanced MRI Studies. State-of-the-art research studies in dynamic contrast enhanced magnetic resonance imaging (DCE-MRI) [12] involve use of large datasets, which consist of time dependent, multidimensional, heterogeneous collections of data from multiple imaging sessions. For image analysis, we can identify several use cases: 1) A scientist retrieves a subset of images stored on a remote image server from a single DCE-MRI session and performs viewing and analysis of image data on their local workstation. In this simplest scenario, the scientist can perform some standard set of image processing techniques on an individual data set. 2) She can extend the analysis to longitudinal studies and apply several

*This research was supported in part by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497).

different statistical analysis methods on data from multiple different studies to view the effects of a treatment method across a patient group. 3) She wants to iteratively execute image-processing algorithms on the datasets in order to determine which parameters maximize the effectiveness of a particular image operation (e.g., registration, segmentation, statistical analysis). This scenario corresponds to a parameter study. Allowing for parameter studies enables the researcher to quickly quantify the results and to determine an algorithm's efficacy for a target study.

Telemicroscopy: Remote Query of Large Digitized Microscopy Slides. The DCE-MRI studies involve processing across several hundreds or thousands of relatively small (a few megabytes in size) images. However, datasets in telemicroscopy applications (such as those targeted by Telescience and BIRN [4] and telepathology) consist of images that are very large in size. An image acquired by high-power light or electron microscopes is composed of many smaller tiles that are montaged together to form the whole image. Advanced microscopes can produce images with resolutions of 40,000 by 40,000 pixels or higher and sizes ranging from several hundred megabytes to several tens of gigabytes (compressed). A challenging issue is to be able to remotely query a subset of a microscopy image and generate an image at desired resolution by subsampling. For instance, miscroscopy images in the Telescience project[1] are stored on storage systems managed by Storage Resource Broker[2]. A query specifies an image dataset (which may consist of a single TIFF image file containing stacked images of tiles or many TIFF files each corresponding to a tile in the whole image), a rectangular region in the image, and a subsampling factor for the desired resolution. Evaluating the query can involve retrieving the tiles that intersect with the query window, processing these tiles to generate a clipped and subsampled image, and storing the resulting image back in SRB for further analysis.

As these two application scenarios depict, the amount of data per dataset can be very large in some imaging studies, thus making it difficult to transfer and process the data of interest in a distributed setting. In other imaging studies, the volume of data per data set may be small, but a large ensemble of datasets should be queried and processed. It can also entail carrying out computational analysis, such as a complete parameter study using a particular image analysis algorithm or a set of algorithms, on the data in order to derive conclusions.

3 System Architecture

In order to realize the scenarios in Section 2, a framework is needed 1) to have the ability to integrate images from multiple studies and modalities in one or more databases, 2) to implement the functionality to submit one or more queries against such image databases, and 3) to be able to quickly process very large images and image datasets with thousands of images. The overall architecture of the framework proposed in this paper to address these issues are shown in Figure 1.

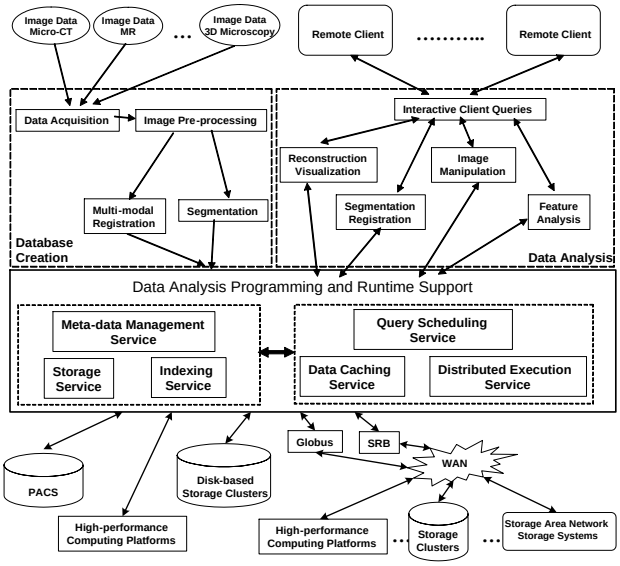

Figure 1. System Architecture.

In this framework, the *storage service* encapsulates efficient storage methods for image datasets. Given the significant disparity between the bandwidths of processor, memory, and disk, a runtime system must provide low-latency retrieval of large volumes of data from the storage system to efficiently support user queries. This service supports data declustering and clustering methods for effective organization and placement of datasets on a storage system. The *indexing service* provides support for efficient indexing of input image datasets and features extracted from the image datasets. Biomedical image datasets are multi-dimensional and multi-resolution, thus an instance of this service can implement spatial indexing methods (e.g., R-trees and their variants) for accessing multi-dimensional datasets. The *meta-data management service* provides a unified mechanism for efficient handling of meta-data associated with datasets (e.g., image types, resolution of image, functional information, and the distribution of datasets across a storage system). The *data caching service* supports methods for detection of common regions of interest and processing requirements among queries, and use of intermediate results cached on disk and in memory. The *query scheduling service* performs scheduling of multiple queries for execution in the system. It also constructs sub-queries for requests that cannot be answered from data cached in the

[1] https://gridport.npaci.edu/Telescience/

[2] http://www.npaci.edu/DICE/SRB/index.html

data caching service. Its task is to schedule the sub-queries to data servers in the system so as to minimize the overall response time. Application dependent processing of large volumes of data is a critical component in analysis and visualization of image data. The *distributed execution service* should make it possible to apply user-defined processing on data as it progresses from data sources to clients. The design and implementation of storage, indexing, caching, and query scheduling services have been discussed in earlier work [3, 2, 17]. In the next section, we focus on an implementation of the distributed execution service to support quick and efficient development of image analysis methods in a distributed environment.

4 Distributed Execution Service

We have developed a prototype implementation of the distributed execution service as an integration of four pieces: 1) an image processing toolkit, 2) a visualization toolkit, 3) a runtime system for execution in a distributed environment, and 4) an XML based schema for process and workflow description. In the current implementation, we have used VTK and ITK frameworks for image processing and visualization and a component-based framework, called DataCutter [3], for runtime support.

4.1 DataCutter

DataCutter supports a filter-stream programming model for developing data-intensive applications. In this model, the application processing structure is implemented as a set of components (referred to as filters) that exchange data through a stream abstraction. The interface for a DataCutter filter consists of three functions: (1) an initialization function (init), in which any required resources such as memory for data structures are allocated and initialized, (2) a processing function (process), in which user-defined operations are applied on data elements, and (3) a finalization function (finalize), in which the resources allocated in init are released.

Filters are connected via logical streams, which denote a uni-directional data flow from one filter (i.e., the producer) to another (i.e., the consumer). A filter is required to read data from its input streams and write data to its output streams only. We define a data buffer as an array of data elements transferred from one filter to another. The current implementation of the logical stream delivers data in data buffers, whose size is specified by the application. The current runtime implementation provides a multi-threaded execution environment and uses TCP for point-to-point stream communication between two filters placed on different machines.

The overall processing structure of an application is realized by a filter group, which is a set of filters connected through logical streams. An application query is handled as a unit of work (UOW) by the filter group. Filters in a filter group can be placed onto computational resources to minimize communication and computation overheads. Processing of data through a filter group can be pipelined. Multiple filter groups can be instantiated simultaneously and executed concurrently. Work can be assigned to any filter group. While UOWs assigned to a single filter group are processed in first-in-first-out order, there is no ordering between different filter groups. A transparent filter copy is a copy of a filter in a filter group. The filter copy is transparent in the sense that it shares the same logical input and output streams of the original filter. If copies of a filter that maintains state (e.g., accumulator) are to be created, an application-specific combine filter is needed to make sure that the output of a unit of work should be the same, regardless of the number of transparent copies.

The filter runtime system maintains the illusion of a single logical point-to-point stream for communication between a logical producer filter and a logical consumer filter. It is responsible for scheduling buffers in a data stream among the transparent copies. For distribution between transparent copies, the runtime system implements a Demand Driven (DD) mechanism based on buffer consumption rate. DD policy aims to send buffers to the filter that will process them fastest. When a consumer filter starts processing of a buffer received from a producer filter, it sends an acknowledgement message to the producer filter to indicate that the buffer is being processed. A producer filter chooses the consumer filter with the minimum number of unacknowledged buffers to send a data buffer.

4.2 Visualization Toolkit (VTK) and Insight Segmentation and Registration Toolkit (ITK)

VTK [16] provides a community standard toolkit for visualizing images and geometry. This toolkit has well defined underlying data structures which pre-written processing methods can operate on. There are basic data structures and methods for representing and accessing structured and unstructured image data, and various types of geometry. Although VTK does not provide mechanisms for serializing its internal objects, it can read and write all of its basic data representation types. The toolkit is also designed to support an object oriented paradigm which allows base functions to be chained together in a program.

ITK from National Library of Medicine [11] provides libraries to enable image processing algorithms focusing in the areas of segmentation and registration. The focus of the toolkit is to create a common code base for containing well published textbook algorithms as well as for researchers to

work on new algorithms. ITK is similar to VTK in many ways. It is also an object-oriented framework and has basic data structures and basic function types for image processing. However, ITK lacks visualization capabilities as its focus is on segmentation and registration. The framework has a layer that allows it to integrate VTK processing with it. This enables using VTK for particular pieces of the processing pipeline, visualization for example, and ITK for other pieces, segmentation and registration for example. ITK does not currently provide any mechanisms for parallel and distributed computing.

4.3 Layering VTK and ITK on DataCutter

In our infrastructure, an image analysis application is represented as a filter group consisting of filters that process the data in a pipelined fashion. That is, VTK and ITK functions constituting the image processing and visualization structure of the application are implemented as application components using DataCutter filter support. These application components operate in a dataflow style, where they repeatedly read buffers from their inputs, perform application-defined processing on the buffer, and then write it to the output stream. The corresponding dataflow (i.e., the layout of the application components and their interaction) is expressed using an XML schema.

We have developed a simple abstraction layer that provides isolation between DataCutter and ITK/VTK. This enables the integration between different toolkits to be independent from any future changes to their underlying implementation. We here present the current underlying hierarchy to write filters that contain ITK/VTK code to be executed in a distributed environment. DataCutter provides a base filter class, which encapsulates the init, process, and finalize methods. A ip4gBasicFilter class extends the DataCutter base filter class, and provides the *processFilter* and delete functions that should be customized by the application developer using VTK/ITK functions. It also implements methods for processing of XML parameter blocks. The parameter block enables each filter to request variables which can be provided in the workflow model description. Another layer of filters is provided to provide default implementations of init, process, and finalize methods for various VTK and ITK basic object types. The processFilter function is called in the default implementation of the process method, and the delete function is called in the finalize method. These filters provide for the basic data marshalling and demarshalling of most generic VTK/ITK data filter types.

As an example, a user would like to create a filter in the IP4G system which could take in image data, subsample it, and pass it to the next filter (as shown in Figure 2). The user would need to create a class which extends the base

```
#include “dcvtkImageToImageFilter.h”
#include “vtkImageShrink3D.h”

class myShrinkF : public dcvtkImageToImageFilter {
    vtkImageShrink3D  *iShrink;
 public:
    myShrinkF() { iShrink = vtkImageShrink3D::New(); }
    ~myShrinkF() { iShrink->Delete(); }
    virtual void processFilter();
};

void myShrinkF::processFilter() {
    int sFactorX = getIntParamValue(“shrinkFactorX”);
    int sFactorY = getIntParamValue(“shrinkFactorY”);
    int sFactorZ = getIntParamValue(“shrinkFactorZ”);
    // put that data into the shrink filter to be downsampled)
    iShrink->SetShrinkFactors(sFactorX,sFactorY,sFactorZ);
    iShrink->SetInput(input);
    iShrink->Update();
    output = imageShrink->GetOutput();
};
```

Figure 2. An example filter.

class. The next step would be to implement the processFilter method. In this particular case, the VTK method called vtkImageShrink3D is used as the subsampling code. The user simply needs to grab the input data, call the appropriate VTK functions, and set the output data. *input* and *output* are defined in the base class and correspond to the input and output streams to and from the user-defined processFilter method. When a data buffer is received from a producer filter, the deserialization method in the base dcvtkImageToImageFilter class is called to create the VTK data structure which is pointed to by input. When the processFilter method returns, the data structure pointed to by output is serialized into a data buffer and sent to the consumer filter of this filter.

4.4 Dataflow Description

We have adopted the process modeling schema developed for the Distributed Process Management System (DPM) [9]. The DPM model allows for a workflow to be described in XML. We have chosen XML as it is widely accepted as a standard way of describing semantic information by Web and Grid services communities, and a range of tools are freely available for parsing and validating XML documents. In DPM, the execution of an application in a distributed environment is modeled by a directed acyclic task graph of processes. This task graph is represented as an XML document. For our implementation, we have extended

```
<job id="analyze 1">
  <process_group>
    <process type = "A" placement = "host1" id = 0>
      <parameter_block>
        <parameter name = "isovalue" val = 0.7/>
        <parameter name = "smooth" val = true/>
      </parameter_block>
      <process type = "B">
        <parameter_block> ... </parameter_block>
        <transparent_copies count=2>
          <placement="host2" count=1/>
          <placement="host3" count=1/>
        </transparent_copies>
        <process type="D" placement="host4" id=1>
          <parameter_block> ... </parameter_block>
        </process>
      </process>
      <process type="C" placement="host2">
        <parameter_block> ... </parameter_block>
        <process_include id="1"/>
      </process>
    </process>
  </process_group>
</job>
```

Figure 3. The job description of a filter group in XML.

the DPM schema to allow for transparent copies of filters and specification of filter placement. A client request in XML to the runtime infrastructure is defined as a job, whose description is encoded between <job> and </job> tags. A job may contain multiple filter groups. This enables a client to submit and instantiate multiple filter groups in a single message. In this paper, we used the same naming convention for tags as provided by DPM. The <process_group> body defines a filter group within a job. Each filter in the filter group is described by a <process> element. The name of the filter is given in the type attribute. The element has two variables. The placement attribute holds the name of the host machine the filter is placed on. The placement for a filter can be determined by the client or by a scheduling algorithm. The optional Id attribute denotes the unique id assigned to the filter by the client for that job and used for reference purposes.

The DPM model recursively represents the data flow dependencies between processes in the XML job definition. In the example shown in Figure 3, filter A depends on the output of filters B and C. Hence, the descriptions of those two filters are given in the <process> block of filter A – if there is no dependency between two filters, they are represented at the same level. Filter D is a producer for both filter B and filter C. There needs to be some way of linking the filter streams together for the filters that are waiting on the referenced input from filter D. To accomplish this with XML, the Id attribute is used for a filter that needs to be referenced. To create a filter reference we simply add the tag <process_include> with the attribute Id set equal to the process that it is being referenced. Filter D is described in the block of filter B and is referenced by its Id in the filter C description. The input parameters for a filter are defined in the <parameter_block> block. The <transparent_copies> block defines the transparent copies of a filter and their placement in the system. The number of copies can be determined by the client or a scheduling algorithm.

5 Interaction with Other Grid Services

In a Grid-wide deployment, the services presented in this paper should be augmented by support for security and authentication, access to files in a distributed environment, resource allocation, and resource monitoring. There is an array of middleware toolkits that can be used for such support and IP4G can leverage those existing toolkits. Globus toolkit[3] provides support for resource discovery and allocation, authentication/authorization, and file transfer (via GridFTP) across multiple administrative domains. The Metacomputing Directory Service (MDS) of Globus can be used for resource discovery and allocation. The meta-data service can make use of the MCAT infrastructure of Storage Resource Broker (SRB) for storing meta-data for datasets. SRB provides unix-like I/O interfaces for distributed collections of files across a wide-area network and our framework can take advantage of SRB functions for remote file access. The Network Weather Service [19] can be used to gather on-the-fly information about resource availability so as to determine where to instantiate the data processing components of an image analysis application.

We have chosen DataCutter as the underlying runtime system of IP4G for two reasons. First, it supports a framework for executing application-specific processing as a set of components in a distributed environment. Processing, network and data copying overheads are minimized by the ability to place filters on different platforms. This capability easily enables execution of various services and user-defined filters in a distributed environment. Second, in the DataCutter project, we are developing interfaces to the Globus, SRB, and NWS toolkits. This will allow us to readily use the security, authentication, remote file access, resource monitoring and allocation services provided by these toolkits.

Other high performance computing research projects such as Condor have a need for process modeling and data flow description [7]. Condor contains a tool called

[3] http://www.globus.org

DAGMan, which like DPM uses a Directed Acyclic Graph (DAG) to manage processes whose input and output is dependent on the execution of other processes. Both the Web Services and Grid Service communities are defining standard schemas whose respective services will provide interfaces such that they can defined, described, registered, discovered, and executed [6, 8]. As these standards evolve within these communities, we plan to evolve the IP4G and its dataflow description schema such that IP4G applications can become or interact with both web and grid services.

6 Related Work

The growth of biomedical imaging data, has led to the development of Picture Archiving and Communication Systems (PACS) that store images in the DICOM standard format [10]. PACS supports storage, management, and retrieval of image datasets and meta-data associated with the image data. A client using those systems usually has to retrieve the data of interest, often specified through a GUI, to the local workstation for application-specific processing and analysis. The effectiveness of imaging studies with such configurations is severely limited by the capabilities of the client's workstation. Advanced analysis techniques and exploration of collections of datasets can easily overwhelm the capacity of most advanced desktop workstations.

Manolakos and Funk [13] describe a Java-based tool for rapid prototyping of image processing operations. This tool uses a component-based framework, called JavaPorts, and implements a master-worker mechanism. Oberhuber [15] presents an infrastructure for remote execution of image processing applications using SGI ImageVision library, which is developed to run on SGI machines, and NetSolve [5]. Dv [1] is a framework for developing applications for distributed visualization of large scientific datasets on a computational grid. It is based on the notion of active frames, which are application level mobile objects. An active frame contains application data, called frame data, and a frame program that processes the data. Active frames are executed by active frame servers running on the machines at the client and remote sites. Like these projects, our work targets distributed execution of image analysis and visualization applications. However, it differs from them in that we are developing a services-oriented framework, and the image analysis toolkit presented in this paper builds on a component-based middleware that provides combined task- and data-parallelism through pipelining and transparent component copies. VXL and TargetJr [18] are two toolkits that are mainly used by the image understanding community. The main focus here is usually temporal sequences of 2D images.

There are a number of projects that target development of infrastructures for shared access to data and computation in various areas of medicine, science, and engineering. Biomedical Informatics Research Network (BIRN) [4] initiative focuses on support for collaborative access to and analysis of datasets generated by neuroimaging studies. It aims to integrate and enable use of heterogeneous and grid-based resources for application-specific data storage and analysis. MEDIGRID [14] is another project recently initiated to investigate application of Grid technologies for manipulating large medical image databases.

7 Experimental Results

We present a preliminary evaluation of the toolkit using two separate test applications. We used a 22-node Linux Cluster (referred to as osumed) of Pentium 933 MHz with 512MB main memory and a 6-node Linux Cluster (referred to here as DC) of Pentium 2.0 GHz with 512MB main memory. The nodes in each cluster are connected through Switched Fast Ethernet, while the two clusters are connected to each other over a shared 100MBit Ethernet.

The first set of experiments was performed with an application that reads in a 3D time-dependent volume dataset, generated from an oil reservoir simulation. The application then extracts an isosurface for each time step of the dataset and sends it to the client. The pipeline of operations is as follows: read a time step of data (R), scale the data values by an integer (S), cast the data values from float to short (C), generate an isosurface in the volume (I). Each of these steps is implemented as a separate filter. In the experiments, this process repeated for all 20 time steps of the example dataset. In a comparison of the VTK standalone performance to that of the IP4G implementation, we observed about 25% overhead when all the filters run on a single machine. This overhead is the result of data serialization and data copying due to splitting of processing into separate filters.

Figure 4 shows the performance gain by creating transparent copies of the isosurface extraction filter. In this experiment, the number of transparent copies was varied from 2 to 5. Each transparent copy was executed on a separate node of the machine. The graph shows that the baseline run with no IP4G was about 14 seconds. With transparent copies, we were able to get the runtime down around 7 seconds. This execution time reduction of about half is not substantial, but we were able to achieve this quickly and without having to write any specific distributed execution code. We also expect the constant time overhead incurred by using the distributed execution service will be shadowed as datasets are scaled in size and complexity.

Figure 5 shows the performance numbers using two clusters connected across a wide-area network over a shared 100MBit Fast Ethernet. In this experiment, the dataset was stored on the DC cluster. The first bar shows the total execution time when the data set is copied from the DC cluster to

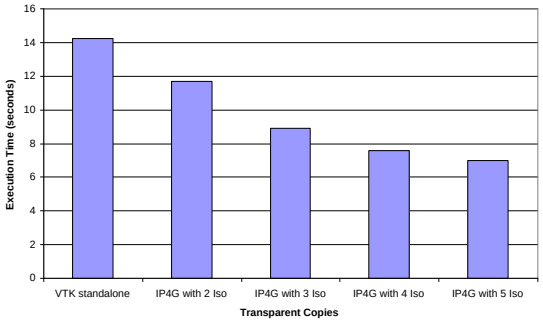

Figure 4. Execution of the application with transparent copies of the isosurface extraction filter. The number of transparent copies is varied from 2 to 5 and each transparent copy is placed on a different node of the cluster.

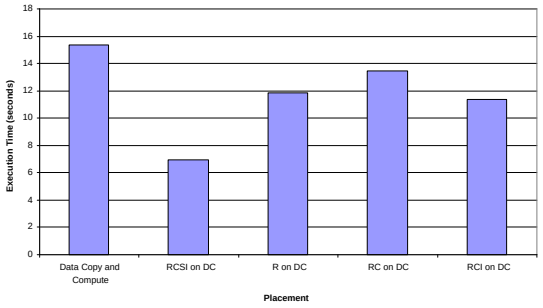

Figure 5. Data access and processing in a wide-area environment. One cluster (osumed) is located at the Ohio Supercomputer Center, the other one (DC) at Biomedical Informatics.

osumed cluster, where the client is located, and processed on that cluster. The execution time is the sum of the data copying and processing time. If all of the processing can be run on the DC cluster using multiple transparent copies, the execution time goes down, as expected. The last three bars in the figure show the execution times when the filters are placed on both clusters. As seen in the figure, executing some of the filters on the DC cluster and some of them on the osumed cluster achieves better performance than copying the data and performing the operation locally. Being able to place the filters across two clusters is especially beneficial when the data server becomes overloaded.

The next set of experiments uses an image processing pipeline for processing DCE-MR images [12]. The dataset consists of 14,400 images, each of which is 100KB, for a total of 1.4GB. The processing pipeline contains stages, implemented as filters, to read the image (R), supersample it by a factor of 2 (S), gaussian smooth (G), clip out a subimage (C), and threshold between a predetermined set of isovalues (T). We ran this pipeline in two different ways: a completely data parallel model and a hybrid model that is a combination of task and data parallelism. The data parallel model processes all the data for one processing stage and caches the result data from that stage on disk. Copies of the corresponding filter are created on all the processors. After all the data has been processed in a stage, the next stage reads the cached data and processes it for the next filter stage and so on. The hybrid case, on the other hand, creates transparent copies of each filter on all nodes. All filters are run at the same time and data is processed in a pipelined fashion. The data travel from one filter to the next by a demand driven scheduling process, as described in Section 4.1.

Figure 6 shows the results of the data parallel vs. hybrid model for a single cluster (osumed) and multi-cluster (osumed and DC) setup where data is partitioned uniformly across the nodes. It also demonstrates how the pipeline scales when the data and computation are partitioned with varying number of nodes. The amount of time it takes to process a single image is between 2 to 2.7 seconds for the data parallel model, and is relatively constant with respect to the number of nodes. The graph shows that the hybrid approach is much faster than the purely data parallel approach. We attribute this to extra cache I/O needed to create a completely data parallel process when compared to that of a hybrid pipe-and-filter approach.

Figure 7 shows the performance of data parallel model vs hybrid approach when a second cluster (DC) is introduced as an additional compute only resource. In this experiment, the data is stored on 4, 8, and 16 nodes of osumed and 4 nodes from DC are added as compute only resources. As seen from the figure, hybrid approach performs better than completely data parallel approach in this case as well. We should note that since the second cluster depends on the first as data source, slow data transfer over network can decrease the speedup. In the case where the bandwidth of the shared network is very low, data partitioning and processing of data locally can be more effective.

8 Conclusions

We presented the design of a framework for supporting image analysis applications in the Grid environment. We described the implementation of a toolkit, which builds on a component-based framework. This toolkit aims to provide support for rapid development and efficient execution of image analysis applications using ITK and VTK in a distributed environment. Our results show that the toolkit can achieve good performance by enabling combined task and data parallelism in a chain of processing operations.

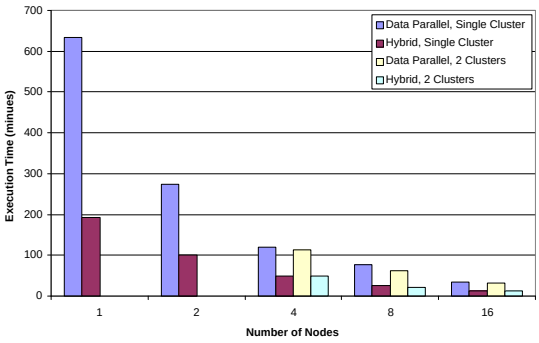

Figure 6. Data parallel and hybrid approaches for a single cluster and multi-cluster setup. In the two-cluster case, the ratio of the nodes on osumed and DC is 3 to 1.

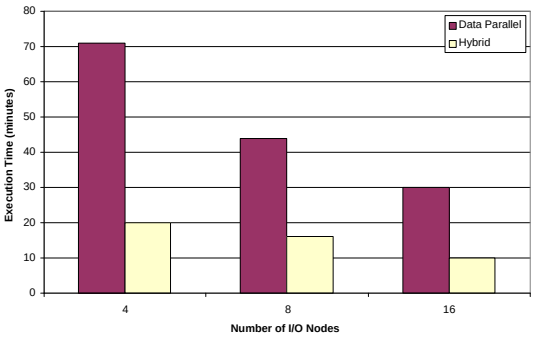

Figure 7. Performance of data parallel and hybrid approaches, when 4 nodes from DC are added as compute only nodes. In this case, the total number of compute nodes is the number of I/O nodes on osumed plus 4.

References

[1] Martin Aeschlimann, Peter Dinda, Julio Lopez, Bruce Lowekamp, Loukas Kallivokas, and David O'Hallaron. Preliminary report on the design of a framework for distributed visualization. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1833–1839, Las Vegas, NV, June 1999.

[2] Henrique Andrade, Tahsin Kurc, Alan Sussman, and Joel Saltz. Active Proxy-G: Optimizing the query execution process in the Grid. In *Proceedings of the 2002 ACM/IEEE SC02 Conference*. ACM Press, November 2002.

[3] Michael Beynon, Chialin Chang, Umit Catalyurek, Tahsin Kurc, Alan Sussman, Henrique Andrade, Renato Ferreira, and Joel Saltz. Processing large-scale multidimensional data in parallel and distributed environments. *Parallel Computing*, 28(5):827–859, May 2002. Special issue on Data Intensive Computing.

[4] Biomedical Informatics Research Network (BIRN). http://www.nbirn.net.

[5] Henri Casanova and Jack Dongarra. NetSolve: a network enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[6] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. *http://www.globus.org/ogsa*, 2002.

[7] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. IEEE Press, Aug 2001.

[8] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. SAMS Publishing, 2002.

[9] S. Hastings. Distributed architectures: A java-based process management system. Master's thesis, Computer Science Department, Rensselear Polytechnic Institute, 2002.

[10] H.K.K. Huang, G. Witte, O. Ratib, and A.R. Bakker. *Picture Archiving and Communication Systems (PACS) in Medicine*. Springer-Verlag, 1991.

[11] National Library of Medicine. Insight Segmentation and Registration Toolkit (ITK). *http://www.itk.org/*.

[12] M.V. Knopp, F.L. Giesel, H. Marcos, H. von Tengg-Kobligk, and P. Choyke. Dynamic contrast-enhanced magnetic resonance imaging in oncology. *Top. Magn Reson. Imaging*, 12(4):301–308, 2001.

[13] E.S. Manolakos and A. Funk. Rapid prototyping of component-based distributed image processing applications using javaports. In *Workshop on Computer-Aided Medical Image Analysis, CenSSIS Research and Industrial Collaboration Conference*, 2002.

[14] MEDIGRID. http://creatis-www.insa-lyon.fr/MEDIGRID/home.html.

[15] M. Oberhuber. Distributed high-performance image processing on the internet. Master's thesis, Technische Universitat Graz, 2002.

[16] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 2nd edition, 1997.

[17] Matthew Spencer, Renato Ferreira, Michael Beynon, Tahsin Kurc, Umit Catalyurek, Alan Sussman, and Joel Saltz. Executing multiple pipelined data analysis operations in the Grid. In *Proceedings of the 2002 ACM/IEEE SC02 Conference*. ACM Press, November 2002.

[18] TargetJr and VXL. *http://www.esat.kuleuven.ac.be/ targetjr*.

[19] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.