Managing Data and Image Processing Workflows for Large Scale Image Analysis*

Shannon Hastings, Stephen Langella, Tony Pan, Scott Oster, Umit Catalyurek, Joel Saltz, Tahsin Kurc

Department of Biomedical Informatics, The Ohio State University
Columbus, OH, 43210
{hastings,langella,tpan,oster,umit,jsaltz,kurc}@bmi.osu.edu

Abstract

Biomedical image data can provide rich information about morphological and functional characteristics of biological systems. Major challenges to more effective use of biomedical imaging in basic and clinical research are the efficient management of large image datasets and image processing workflows and the effective sharing of data and workflows in a collaborative environment. In this paper, we present a suite of services that work collectively to address issues associated with metadata management, data storage and management, and management and execution of image processing workflows. We demonstrate the use of a generic XML-based metadata and data management system in tandem with a distributed execution system. We perform a preliminary performance evaluation of the system on a cluster system.

Keywords: Workflow management, Metadata, Distributed Computing, Image Analysis, XML.

1 Introduction

Analysis of image data is increasingly becoming a key component of biomedical research. Recognizing this, several large and collaborative initiatives have targeted development of systems to support access to distributed repositories of image datasets [7, 24, 30]. Advanced image acquisition devices make it possible for the biomedical researcher to obtain structural and functional information at very high resolution. High-end scanners can generate images up to 20 Gigabytes in size. Storage requirements can be enormous as a single study can generate hundreds of such images. There are also difficulties in achieving rapid response time for various types of data analysis queries into the slide image database.

On the hardware side, PC clusters built from low-cost, commodity items are increasingly becoming widely used, causing a change in the way hardware can be scaled and leveraged in a cost-effective manner. A remaining challenge is to develop techniques and middleware support for 1) management of large datasets (from gigabytes to multiple terabytes in size), 2) execution of simple and complex image analysis workflows, and 3) management of metadata associated with input/output datasets, intermediate results, and workflows (the number of images and data files can go up to millions of files distributed across multiple systems) on distributed collections of commodity clusters. In addition to allowing users to traverse images, software systems should efficiently support the synthesis and analysis of thousands of images and facilitate the sharing of results obtained from image analysis.

In [20], we developed a system to support rapid implementation and distributed execution of image analysis applications, implemented using the Insight Segmentation and Registration Toolkit (ITK) [21] and Visualization Toolkit (VTK) [27]. The system provided support for the combined use of task- and data-parallelism and pipelined execution in a distributed environment. However, it implemented a rather rudimentary mechanism for management of storage, data, and metadata, which is important for collaborative studies.

In this work, we extend our previous work to develop and evaluate a suite of services that work collectively to address challenging issues in metadata management, data storage and retrieval, and image workflow management and execution. We demonstrate the use of a generic, XML-based metadata and data management system, called Mobius [25], in conjuction with the distributed execution environment developed in [20]. We perform a preliminary performance evaluation of the system on a cluster of machines with varying number of images.

*This research was supported in part by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ANI-0330612, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), NIH NIBIB BISTI #P20EB000591, Ohio Board of Regents BRTTC #BRTT02-0003.

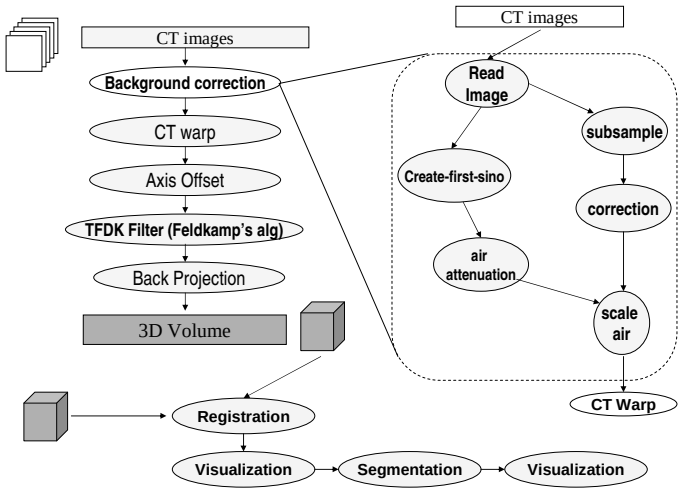

Figure 1. A sample image analysis workflow for micro-CT images. A series of methods are applied on the CT images to reconstruct a 3D volume. In this workflow, the background correction method consists of another series of operations on data. The result 3D volume can then be registered with a 3D volume from another imaging session, registered volumes can be visualized and segmented, and segmentation results can be visualized by the researcher.

2 Motivation and Overview

2.1 Motivation

Although data values and types of data acquired by different imaging modalities may differ, a common characteristic is that the amounts of data captured by advanced imaging devices can surpass the storage capacity of a single machine, requiring use of distributed storage systems. Another challenge is to provide support for on-demand image analysis and collaboration. In order to address these needs, a system should support:

Management and Evolution of image processing workflows. Researchers should be able to compose, register, and version workflows. The definitions and instances of workflows should be managed in a standard and efficient way so that researchers can share workflows and reference others' workflows in theirs. The system should also allow a researcher to version workflows (e.g., to add new analysis components or modify the order of data processing steps) and manage the updated instances of workflows.

On-demand creation and management of distributed image databases. In addition to sharing of the output of data analysis, datasets generated by the intermediate steps of a workflow could be stored and shared. By examining these datasets, a collaborator can better understand how the researcher arrived at her results and to potentially modify the workflow to see if better results or different conclusions can be obtained. Furthermore, any of these intermediate datasets could be utilized as input to a completely new workflow that builds on previous steps of the original workflow. This could potentially provide a significant runtime performance improvement as redundant workflow steps do not need to be recomputed if they are shared between multiple workflows [31]. A researcher should be able to create a database of images and make it available for remote access. From a performance point of view, the database should be able to scale to very large datasets and to large number of clients by taking advantage of heterogeneous collections of clusters and multiprocessor machines. In order to minimize I/O overheads, techniques should be implemented for data distribution and data staging.

Efficient execution of image analysis workflows. An image data processing flow can consist of several stages of 1) sequences of simple and complex operations, 2) an array of parameters applied to a group of operations (e.g., registration, segmentation), and 3) interactive inspection and visualization of images. As an example, Figure 1 shows an image analysis workflow for micro-CT images. In order to speed up complex operations and parameter studies, the system should support execution of image analysis workflows in a heterogeneous and distributed environment. It should also allow check-pointing of intermediate results so that the user can carry out interactive inspection on the data.

2.2 System Overview

In this work, we propose a system that consists of three core services to address these issues. These services collectively provide support for metadata management, data storage and management, and distributed execution.

- The first service builds on a distributed metadata management component that keeps track of metadata associated with workflows, input image datasets, intermediate results check-pointed during execution of a workflow, and output image datasets. This service allows registration, management, and versioning of workflows and image information as XML schemas. Schemas defined by other researchers can be included in a schema for the workflow and image information for a study. This service also manages the metadata associated with image datasets generated by an execution of a workflow.
- The second service is a storage service that efficiently manages distributed collections of disks in the system for storage and staging of collections of input images, images generated by intermediate stages of the image analysis workflow, and output datasets. This service supports distributed storage of large images and index generation and lookup for efficient data retrieval. It enables an XML virtualization of relational databases, as defined by XML Schemas. It allows user-defined data types to automatically manifest custom databases at runtime, and data adhering to these data types to be stored in these databases.
- The third service builds on the distributed execution middleware developed in [20]. It supports efficient execution of image analysis workflow as a network of image processing components in a distribute environment. The network of components can consist of multiple stages of pipelined operations, parameter studies, and interactive visualization.

We build the three services using Mobius [25] Global Model Exchange (GME) and Federated Data Exchange (Mako) services and DataCutter [5]. Mobius is a middleware framework designed for efficient metadata and data management in dynamic, distributed environments. Its design is motivated by the Grid (in particular by the activities of the Data Access and Integration Services group at Global Grid Forum [11, 17]), by earlier work done at General Electric's Global Research Center [22], and by the requirements of biomedical research studies that involve integration of data, including proteomic, molecular, genomic, and image data, in a multi-institutional environment. Mobius provides a set of generic services and protocols to support distributed creation, versioning, management of database schemas, on-demand creation of databases, federation of existing databases, and querying of data in a distributed environment. Its services employ XML schemas to represent metadata definitions and XML documents to represent and exchange metadata instances.

In our system, image analysis is described as a network of data processing components (an image processing workflow). Input images in a dataset are processed through this network to produce sets of output images. Image processing workflows and image datasets are modeled by XML schemas. These schemas are managed by the metadata management services of Mobius. Instances of schemas are stored, retrieved, and managed by the data management services of Mobius. The image schema may define attributes associated with an image, such as the type of the image, study id for which this image was acquired, date and time of image acquisition, etc. An instance of the schema corresponds to an image dataset with images and associated image attributes stored across multiple storage systems running data management servers. For a workflow, the schema defines the skeleton of the data processing network. An instance specifies the function names and locations of individual components, the number of copies and placement of copies for a component, persistent check-pointing locations in the workflow (which tells the execution environment that output from check-pointed components should be stored as intermediate image datasets in the system), input and output datasets (conforming to some schema registered for image data), and data selection criteria (which specifies the subset of images from input datasets to be processed). The workflow instance can be stored in the system so that clients can search for it and execute it using the distributed execution environment. The distributed execution service carries out instantiation of components on distributed platforms, and management of data flow between components, and data retrieval from and storage to distributed collections of data management servers. An example use of the system is shown in Figure 2.

In the rest of the paper, we describe each service in more detail and present performance results. Our preliminary experimental evaluation of the proposed system focuses on execution of workflows with and without check-pointing, and scalability with respect to storage and retrieval of images on parallel storage and compute platforms. In a future work, we plan to investigate the efficacy of the system for collaborative studies and studies in which database and workflow definitions change over time.

3 Related Work

Several projects have focused on problem solving environments and runtime support for application execution in distributed environments [23, 26, 9, 1, 3, 15, 28, 6, 10, 2].

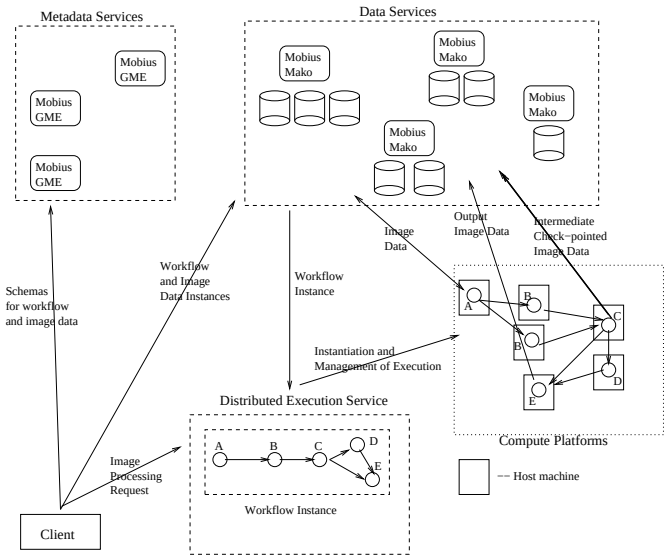

Figure 2. The overall system architecture. A client can create new schemas for workflow and image data or use existing ones in the metadata service. The client creates an instance of the workflow schema, stores it in the data service, and sends an image processing request to the distributed execution service. The distributed execution service retrieves the workflow instance, instantiates it on a distributed platform, and manages its execution. The execution of the workflow reads input data from data services and stores the check-pointed intermediate images and output images in the system.

The Chimera [3, 15] system implements support for establishing virtual catalogs that can be used to describe how a data product in an application has been derived from other data. The system allows storing and management of information about the data transformations that have generated the data product. This information can be queried and data transformation operations can be executed to regenerate the data product. The AppLeS Parameter Sweep Template [10] is a middleware system for expressing and executing parameter sweep applications in the Grid. The Pegasus project develops systems to support mapping and execution of complex workflows in a Grid environment [13, 12]. The Pegasus framework uses the Chimera system for abstract workflow description and Condor DAGMan and schedulers [16] for workflow execution. It allows construction of abstract workflows and mappings from abstract workflows to concrete workflows that are executed in the Grid.

These systems, including our system, are conceptually similar; they target execution of applications in distributed environments. They differ in their target applications and the underlying systems and techniques they use. Our system targets image analysis applications in which the processing structure can be exposed as a network of operations. We address the metadata and data management issues using a generic, XML-based metadata and data management system. This system allows modeling and storage of image data and workflows as XML schemas and XML documents, enabling use of common protocols for storing and querying data. The system is designed as a set of loosely coupled services with well-defined protocols to interact with them. For example, the metadata and data services of our system could be used by other systems in order to store workflow definitions and instance data. Similarly, our system could use Condor [16] and Pegasus [13, 12] for scheduling of computations in a Grid environment.

Linda [2] is a coordination language designed to simplify the construction of complex parallel systems. It relies on asynchronous and associative communication to coordinate computation in its global Tuple Space. A key to the success of Linda is its simplicity. However, this simplicity also makes it difficult to express richer data types and computional requirements, such as maintaining data locality in the global tuple space. Our work addresses these limitations in serveral ways. Our data storage system provides the ability to make intelligent storage decisions such as maintaining workflow locality. Our data types and querying capabilities are exteremely expressive as we utilize the rich structural descriptions of XML Schemas and query using advanced XML query languages.

A large body of research has been devoted to develop-

ing software infrastructure to enable application execution in the Grid [14, 17]. Globus [18] provides a set of services that support authentication, authorization, replica management, resource allocation, and job execution. The Storage Resource Broker (SRB) [29] and the Network Weather Service (NWS) [32] are other Grid systems that provide support for remote file access and resource monitoring, respectively. While our work focuses on issues associated with metadata/data management and workflow exection, our system can leverage those systems in several ways. For example, the data service can utilize Globus replica management services for creating and managing replicas of the datasets. SRB can be leveraged for remote file access, if images are stored on remote file systems. NWS can be used for run time load management of our distributed metadata/data storage services.

Image Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="projectmobius.org/1/image"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:img="projectmobius.org/1/image" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="imageType">
        <xs:sequence>
            <xs:choice>
                <xs:element name="imageDataRef" type="xs:string"/>
                <xs:element name="imageData" type="xs:base64Binary"/>
            </xs:choice>
            <xs:element name="imageMetaData" type="img:detailSetType" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="encoding" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="detailSetType">
        <xs:sequence>
            <xs:element name="detail" type="img:detailType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="image" type="img:imageType"/>
    <xs:complexType name="detailType">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="descriptor" type="xs:string"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

Figure 3. Schema for image data.

4 Metadata Service: Management of Models for Workflow and Data

For any image processing application it will be essential to have a model and the metadata for the image data. In order to support processing of image datasets, we should also be able to model the workflow, which represents the image processing structure of an application. In this work, image data and workflows are modeled using XML schemas and managed by the metadata service. The metadata service is implemented using the Global Model Exchange (GME) service of Mobius. We first give a brief description of the GME and then present how image data and image processing workflows are modeled in the system.

4.1 Mobius GME: A Global Model Exchange

In order to be able to share data type definitions among data services, there must be a service that allows data models or schemas to be published and retrieved. Such a service would also need to be able to handle model versioning, naming conflicts, and model to model referencing. The GME is a distributed service that provides a protocol for publishing, versioning, and discovering XML schemas. Since the GME is a global service, it needs to be scalable and replicable. To handle these issues, the GME is implemented as an architecture similar to Domain Name Server (DNS), in which there are multiple GMEs each of which is an authority for a set of namespaces.

The GME provides the ability for inserted schemas that reference entities already existing in other schemas and in the global schema defined by a researcher. When processing an entity reference, the local GME contacts the authoritative GME to make sure that the entity being referenced exists and that the schema referencing it has the right to do so. The GME enforces a policy for maintaining the integrity of the global schema when deleting schemas. For example, deleting a schema which contains entities that other schemas reference will destroy the integrity of the global schema. The GME schema deletion policy will have to handle such special cases.

Other services such as storage services can use the GME to match instance data with their data type definitions. Since we are using Mobius Mako to store our instance data, all of our schemas will be published to a GME. These schemas can be versioned (e.g., a researcher can extend the base image data schema to include annotations and definitions of features) and referenced in other schemas.

4.2 Image Data Model

Our image model, defined in the XML Schema shown in Figure 3, is made up of two components: the image metadata and the image data. The image metadata component contains the encoding type of the image (JPEG, TIFF, etc.) and a detail set entity which allows a collection of user-defined key value pair metadata. This allows application specific metadata to be attached to an image, for application-specific purposes, while maintaining a common generic model for component reuse. The image data component contains either a Base64 encoded version of the image data or an application-interpreted protocol/file system reference to the location of the image data. The image model is generic enough to allow the image data to be embedded with in an instance of the model or it provides enough information to retrieve the image data from the file system or other storage system.

4.3 Image Processing Workflow Model

We have adapted the process modeling schema developed for the Distributed Process Management System (DPM) [19]. 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 in an XML schema. The workflow schema (Figure 4) in our system defines a hierarchical model starting with a <workflow> entity which contains several jobs, each represented by the <job> tag. Each job represents a distibuted application which is composed of a set of local/distributed processes. Each component (or process), represented by the <process> tag, has attributes that describe its execution to the distributed execution middleware. The name attribute allows the component to be named such that it can be uniquely identified by the distributed execution framework. The componentType attribute describes which component type the process is.

We currently support internal and external component definitions. An internal component represents a process or function that adheres to an interface, DataCutter in this case, and is implemented as a application-specific component in the system. An external component represents a self-contained executable that exists outside the system and communicates with other components through files[1].

The component attribute identifies the path or name of the execution binary. For an external component, this may be a path to an executable, whereas for an internal component it can be a method or interface. Each component entity may also contain three optional sub entities, a parameter set list, a placement, and set of processes. The *parameter set list* consists of a list of set of key-value pair parameters that are passed as input into the process. A single set defines the input to a component, while the list allows for parameter studies, in which the same data is processed by the same component, but with different parameters. The *placement* entity specifies to the distribute execution service which nodes to run the process on and how many copies of the process should be run on each node. The placement entity also allows for more dynamic scheduling, instead of specifying specific hostnames wild cards can be use. In this case, a scheduling software should generate the placement of components. Finally, the *processes set* is a set of processes that the current process is dependent on. The output from all the processes in the process set will be sent as input to the current process, thus creating a distributed process pipeline.

[1] Although the difference between internal and external components may seem subtle, this distinction is important because it allows the execution environment to make assumptions on internal components that it could not make on external components such as number of input and output streams for example.

Workflow Model

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="datacutter.org/1/DatacutterProcessModel"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:dc="datacutter.org/1/DatacutterProcessModel" elementFormDefault="qualified">
    <xs:element name="workflow">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="job" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="processes" type="dc:processesType"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                        <xs:attribute name="name" type="xs:string"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="description" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="processType">
        <xs:sequence>
            <xs:element name="params" type="dc:paramsType"/>
            <xs:element name="placement" minOccurs="0">
                <xs:complexType>
                    <xs:attribute name="host" type="xs:string" use="required"/>
                    <xs:attribute name="nCopies" type="xs:integer" use="prohibited"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="processes" type="dc:processesType" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="componentType" type="dc:componentType" use="required"/>
        <xs:attribute name="component" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="processesType">
        <xs:choice maxOccurs="unbounded">
            <xs:element name="process" type="dc:processType" minOccurs="0"/>
            <xs:element name="process_ref" type="dc:processRefType" minOccurs="0"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="paramType">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="paramsType">
        <xs:sequence>
            <xs:element name="param" type="dc:paramType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="processRefType">
        <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:simpleType name="componentType">
        <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="internal"/>
            <xs:enumeration value="external"/>
            <xs:enumeration value="parallel"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Figure 4. Workflow Schema

As discussed earlier, the GME allows a schema to reference in its definition other schemas registered in the system. This allows contruction of complex database schemas from multiple schemas. We use this functionality to support nested workflows. That is, workflows consisting of components and other workflows. A process in a workflow schema can be a reference to another job definition enabling one researcher's process model to be comprised of references to other small process models.

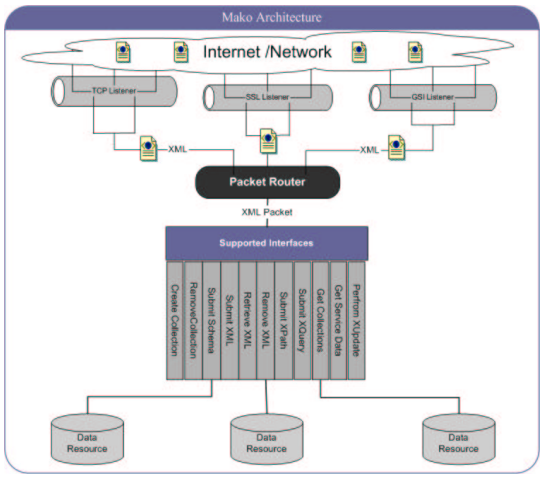

Figure 5. Mako Architecture

5 Data Service: Data Storage and Management

In order to actually process data, a number of XML instances must be created and stored that adhere to the models defined by users. Instances of workflows must be created and stored in the data storage service, so that the execution service can retrieve them and execute them when appropriate. Additionaly, the input data sets must be ingested and indexed by the data service, so that they can be referenced by the workflows. The data service component of the system uses the Mako service of Mobius as the backbone for storage and management of metadata and data instances. We have augmented the Mako service with a data distribution and staging component to efficiently utilize distributed storage and computing power in the environment.

5.1 Mako

Mako (Figure 5) is a service for storing, updating, retrieving, and querying XML documents. Mako exposes data sources as XML data services through a set of well defined interfaces based on the Mako protocol. Data services can be relational databases, native XML databases, an internal storage database, or any other database or file system. Mako defines both a protocol for communication and a service implementation that utilizes this protocol. Below are descriptions of the operations that are relevant to our image processing infrastructure, and how they are implemented in Mako.

Storage, Update, and Retrieval. All XML documents received by a Mako server are validated against an XML schema as retrieved from the GME server. A Mako instance can be configured to accept only a specific set of XML schemas. When each element in the XML document is inserted it is indexed and assigned a unique identifier within the respective Mako. Mako allows XML documents that are stored in it to be updated. Given proper authorization, a user may retrieve the XML at any level of the document by specifying the element's identifier. This ability to uniquely identify and retrieve subsections of a particular document enables efficient retrieval processing time for queries that only require partial documents.

Querying. Clients and other services may query XML documents within a Mako by sending it either an XQuery [8] statement[2] or an XPath [4] statement. Upon receiving an XQuery or XPath, if the underlying database is a relational database, the Mako converts it to SQL and executes it on the relevant database.

XML Referencing. Mako supports the concepts of virtual inclusion and virtual copying. This means that Mako allows XML documents to be created that may contain references or copies of existing XML documents or elements, both local and remote. In this way, an XML document can be distributed and stored across multiple Mako servers by storing subsections of the document remotely and integrating them with references. This ability is critical in enabling large data documents to be partitioned across a cluster while still maintaining the single document semantics of the model.

For our infrastructure we utilize a collection of Makos distributed across storage nodes to provide a distributed data management service. Both workflow instances, metadata instances describing the image datasets, and image data are stored in Makos. Currently, when an image is to be stored in Mako, it is encoded in Base64; more efficient binary data encoding schemes will be incorporated into the system in future.

5.2 Data Distribution and Staging

To maximize the efficiency of parallel data accesses for image data, the data storage service utilizes declustering techniques. The goal of declustering is to distribute the data across as many storage units as possible so that data elements that satisfy a query can be retrieved from many sources in parallel.

This component consists of two subcomponents. The distribution computation subcomponent implements multiple algorithms for distribution of datasets across the storage system. The user or the workflow execution environment can specify a preferred declustering mechanism, such as round-robin or demand driven, when it requests a distribution plan. The data mover subcomponent is responsible for staging the data according to a distribution plan. This subcomponent can be manifested as either a centralized service,

[2]Our current implementation provides limited support for XQueries.

through which all data passes, or as a decentralized component, utilized by execution nodes. Traditionally, the centralized component is only utilized for initial data staging for the input data sets. The decentralized component is utilized during workflow execution to distribute computed datasets (intermediate or output datasets). Presently, the data distribution and staging component is implemented as a set of DataCutter read and write filters that can read data from and write data to multiple Mako servers.

The system currently provides round-robin and demand-driven strategies for data distribution. In round-robin, data is distributed across multiple Makos in a round-robin fashion, while the demand driven schema distributes data proportional to the on-the-fly data ingestion bandwidth of individual Makos. When data sets are ingested, they can be associated with a user-defined identifier that can later be used to retrieve the data set from the data storage service.

6 Distributed Execution Service

The distributed execution service uses the runtime system developed in [20]. We provide a brief description here and present the extensions we have implemented. This service integrates two components: 1) the ITK and VTK image processing and visualization libraries [21, 27] and 2) a component-based runtime system, called DataCutter [5]. A DataCutter application consists of application-specific components, called *filters*, and one or more *filter groups*. Image data processing is represented as a filter group, composed of filters that collectively process the data in a sequence of operations. These application filters can operate in a dataflow style, where they repeatedly read buffers from their input ports, perform application-defined processing on the buffer, and then write it to their output ports. We developed a simple abstraction layer that provides isolation between DataCutter and the ITK and VTK libraries. This enables the integration between different toolkits to be independent from any future changes to their underlying implementation. The runtime system allows for combined use of task-parallelism, data-parallelism, and pipelining for reducing execution time of data processing and analysis applications.

In this paper, we have extended the runtime system with two system level filters, *MakoReader* and *MakoWriter*. These filters provide interface between Mako servers and other filters in the workflow. The *MakoReader* filter retrieves images from Mako servers, converts them into VTK/ITK data structures, and passes them to the first stage filters in the workflow. The *MakoWriter* filter can be placed between two application filters, which are connected to each other in the workflow graph, if the output of a filter needs to be check-pointed. The last stage filters in the workflow also connect to *MakoWriter* filters to store output in Mako servers. The execution of a workflow and check-pointing can be done stage-by-stage (i.e., all the data is processed by one stage and stored on Mako servers before the next stage is executed) or pipelined (i.e., all stages execute concurrently; the *MakoWriter* filters interspersed between stages both send data to Mako servers and pass it to the next filter in the workflow).

7 Experimental Results

We investigated the performance of the distributed execution service in [20]. In this paper, we look at three different aspects of the system: 1) scalability of I/O when the number of Mako servers is varied, 2) performance impact of demand-driven and round-robin data distribution strategies, and 3) effect of checkpointing.

7.1 Varying Number of Mako Servers

The first set of experiments investigate the performance of the system as the number of Mako servers that can store data is varied. A cluster with 7 nodes was used for this experiment. Each node of the cluster consists of two AMD Opteron processors with 8 GB of memory, 1.5 TB of disk storage in RAID 5 SATA disk arrays. The nodes are connected to each other via a gigabit switch. In this experiment, a simple pipeline of *MakoReader->Inverter->MakoWriter* filter group is used with 7500 images as input – each image was a 256x256-pixel gray scale image. The Inverter filter inverts the color of each pixel in an image. Two sets of experiments were executed. In the first set of experiments, the number of MakoReader and Inverter filter copies was fixed at 7 and each copy was executed on one of the nodes. The number of MakoWriter filters was varied from 1 to 7. Each reader filter reads from a single Mako server, sends the data to Invert filter copies using demand-driven buffer scheduling [5, 20]. Each writer filter writes to the Mako servers using demand-driven distribution (see Section 5.2). The number of Mako servers that can store data is equal to the number of MakoWriter filters. The images were distributed evenly among 7 Mako servers. In the second set of experiments, the number of Inverter and MakoWriter filter copies was fixed at 7. The number of MakoReader filters was varied from 1 to 7. In this experiment, each reader filter read from a single Mako server. The images were distributed evenly across an increasing number of Mako servers as the number of MakoReader filters increased (i.e., if there are N MakoReader filters, the images were distributed across N Mako servers). The MakoWriter filters stored data in Mako servers using demand-driven distribution.

The results of these experiments are shown in Figure 6. The numbers in the graphs are the total execution time for processing 7500 images. The bars labeled as “Reader” and

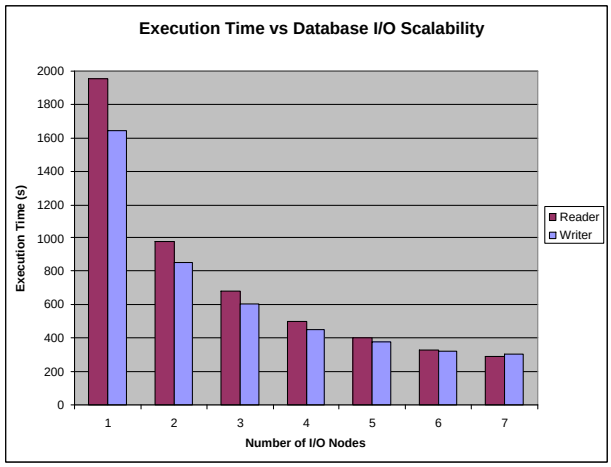

Figure 6. Data I/O Scalability Experiments. "Reader" and "Writer" bars denote the experiments, in which the number of MakoReader and MakoWriter filters is varied, respectively.

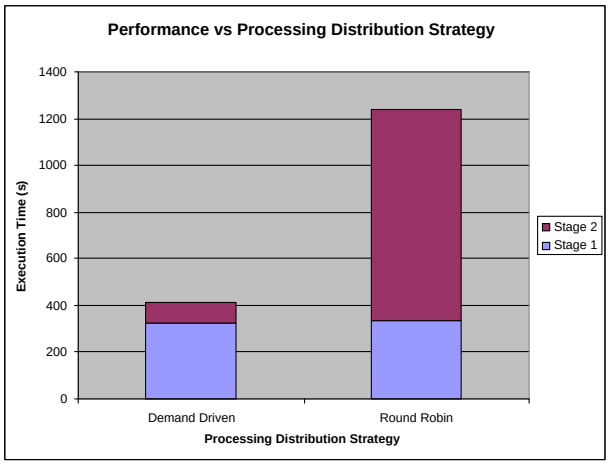

Figure 7. Comparison of demand-driven vs round-robin data distribution strategy.

"Writer" show the execution times when the number of MakoReader and MakoWriter filters is varied, respectively. In the figure, we see that the execution time of the image processing pipeline decreases as the number of reader and writer filters is increased. The graph shows almost linear decrease in execution time and good scalability of the system when the number of Mako servers that can store and serve data is increased.

7.2 Effect of Data Distribution Strategies

These experiments investigate the effect of data distribution strategy used by the data service. The cluster setup is the same as the previous section. The cluster is divided into two subsets. Subset 1 contains 4 nodes (1, 2, 3, and 4), each with one MakoReader filter and one MakoWriter filter. Subset 2 contains 3 nodes (5, 6, and 7), each with one MakoReader filter, one MakoWriter filter, and one copy of each of the processing filters. A Mako server is run on each node. This setup corresponds to a configuration in which some of the machines are used by storage only, while others can be used for both storage and computation. The workflow contains 2 stages. Stage 1 performs Invert, Gaussian smooth, and Clip functions, while Stage 2 carries out Threshold operation on the images. All of these functions are implemented as filters and form a chain of operations on data. Stage 1 is executed first; filters in this stage retrieve the input images from Mako servers. Output from this stage is stored in Mako servers. Stage 2 filters are executed after Stage 1; they retrieve data stored at the end of Stage 1 and process it.

The execution time in figure 7 shows that demand driven strategy outperforms round robin strategy by a factor of 10 in Stage 2. In Stage 2, the imbalance in amount of data read by each read filter is now exacerbated by the need to transfer 2/3 of all images read on each node to other nodes in Subset 2 when round-robin data distribution is used. This additional network load increases the network traffic dramatically, and therefore increases the total execution time in the round-robin case. On the other hand, the demand-driven strategy stores more data on local Mako servers in Subset 2, hence reducing the network and remote data access overhead. This effect is shown in Figure 8. Although images initially are evenly distributed across all Mako servers, more images are stored on local Mako servers after Stage 1 and Stage 2. This experiment demonstrates demand-driven strategy can increase performance dramatically compared to round robin strategy, when data and processing are heterogeneously distributed.

7.3 Effect of Checkpointing

We also examined the effect of checkpointing. This experiment used a cluster of 5 nodes. The nodes were dual 2.4GHz Xeon's with 1GB memory, 320GB storage, and connected via a gigabit switch. The goal of this experiment was to show the cost of capturing intermediate processing results. The ability to capture these results enables process failure recovery, data provenance, and enhanced process model tuning. We varied the number of images processed between 2250 and 9000. The basic image analysis pipeline, as shown in Figure 9, reads the data from a data storage

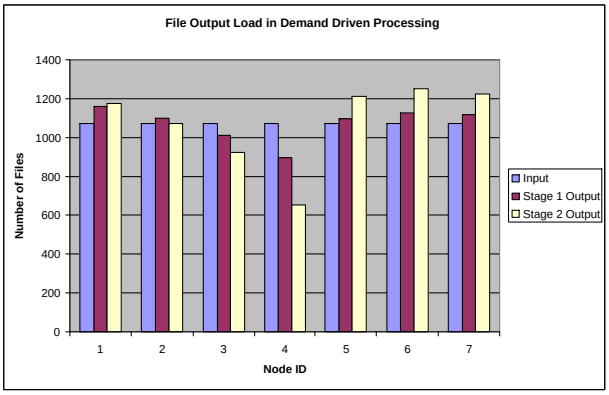

Figure 8. Distribution of image data files across nodes using demand-driven strategy after Stage 1 and Stage 2.

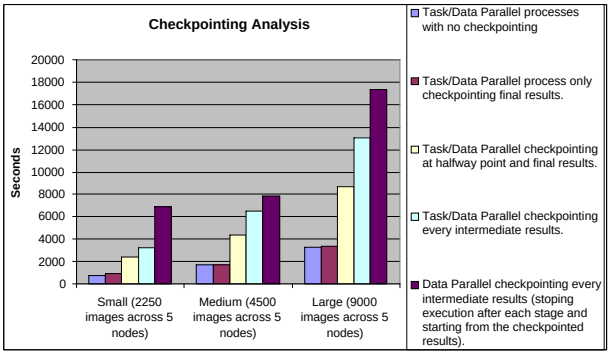

Figure 10. Intermediate Process Checkpointing Experiments.

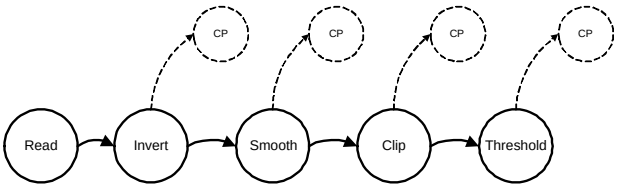

Figure 9. Checkpointing Experiments Model

service, inverts the image, Gaussian smooths it, clips out a smaller region, and finally a scalar value based threshold is applied. Figure 10 shows the execution time of the pipeline with different checkpointing configurations (corresponding to the legends from top to bottom in the graph): 1) read and process all 4 stages of the image analysis, 2) do the same as in 1 and also write the output data to Mako servers, 3) do the same as in 2 and also checkpoint at the halfway point of the image analysis after the *smoothing* stage, 4) is the same as case 3 and checkpoint the results of the *inverting* and *clipping*, and 5) is the case in which each stage in the pipeline was run one at a time, running a copy of the filter on each node. The data generated by each stage was stored in Mako servers. The next stage was initiated when the previous stage finished processing. Each stage read the data that had been stored by the previous stage in Mako servers. As is seen from the figure, case 5 performs worst. In case 5, although filters in a stage does not share the CPU power with filters in other stages, the overhead of checkpoint and restart reduces the performance. This situation is alleviated by running all filters at the same time and checkpointing while filters execute (case 4). We also observe that checkpointing increases the execution time, as expected. This experiment shows that careful consideration must be taken when looking into checkpointing processes. There may be locations in the model where it is cheaper to re-process than it is to checkpoint unless one wants to checkpoint purely for data sharing purposes and not just for process failure/recovery.

8 Conclusions

In this work, we presented a suite of services to address challenging issues in metadata management, data storage and retrieval, and image workflow management and execution. Our work demonstrated the use of an XML-based metadata/data management system in tandem with a distributed execution service. Our preliminary results show that the system is capable of scaling its I/O performance as more data server nodes are added. We also observe that demand-driven data distribution across storage nodes is an effective approach when data is to be processed and stored in a heterogeneous setting.

References

[1] M. Aeschlimann, P. Dinda, J. Lopez, B. Lowekamp, L. Kallivokas, and D. 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] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *Computer*, 19(8):26–34, 1986.

[3] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Proceedings of Supercomputing 2002 (SC2002)*, 2002.

[4] A. Berglund, S. B. X. WG), D. Chamberlin, M. F. Ferndez, M. Kay, J. Robie, and J. Simon. *XML Path Language*

(XPath). World Wide Web Consortium (W3C), 1st edition, August 2003.

[5] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, Oct. 2001.

[6] BioPSE: Problem Solving Environment for Modeling, Simulation, and Visualization of Bioelectric Fields. Scientific Computing and Imaging Institute (SCI), http://software.sci.utah.edu/biopse.html.

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

[8] S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simon. *XQuery 1.0: An XML Query Language*. World Wide Web Consortium (W3C), 1st edition, August 2003.

[9] H. Casanova and J. 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.

[10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *Proceedings of Supercomputing 2000 (SC2000)*, 2000.

[11] Data Access and Integration Services. //http://www.cs.man.ac.uk/grid-db/documents.html.

[12] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Pegasus: Planning for execution in grids. *GriPhyN Technical Report 2002-20*, 2002.

[13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1), 2003.

[14] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[15] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, 2002.

[16] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. 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.

[17] Global Grid Forum. *http://www.gridforum.org/*.

[18] The Globus Project. http://www.globus.org.

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

[20] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz. Image processing for the grid: A toolkit for building grid-enabled image processing applications. In *CCGrid: IEEE International Symposium on Cluster Computing and the Grid*. IEEE Press, May 2003.

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

[22] S. Langella. Intellegent data management system. Master's thesis, Computer Science Department, Rensselear Polytechnic Institute, 2002.

[23] E. 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.

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

[25] The Mobius Project. http://www.projectmobius.org.

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

[27] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 2nd edition, 1997.

[28] SCIRun:A Scientific Computing Problem Solving Environment. Scientific Computing and Imaging Institute (SCI), http://software.sci.utah.edu/scirun.html.

[29] SRB: The Storage Resource Broker. *http://www.npaci.edu/DICE/SRB/index.html*.

[30] The Telescience Portal. https://gridport.npaci.edu/Telescience/.

[31] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. In *Proceedings of High-Performance Distributed Computing (HPDC-12)*, pages 152–161, Seattle, Washington, June 2003.

[32] 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.