Data Synchronization and Replication Tool

Pradeeban Kathiravelu
INESC-ID Lisboa
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
Email: pradeeban.kathiravelu@tecnico.ulisboa.pt

Ashish Sharma
Department of Biomedical Informatics
Emory University
Atlanta, Georgia, USA
Email: ashish.sharma@emory.edu

***Abstract*—Consumers download the data by searching the image repository using the browser. The information that the consumer is interested in, gets updated whenever the data producers update or add patient information. The current download tool lacks the ability to track the relevant updates to the consumer. A data replication and synchronization tool will assist automated downloads to the consumers. Users can create replica sets as a sub set of their search queries, and share their replica sets with other users, and update their replica sets periodically. Replica sets can be used as a way of tracking and sharing information. This project exploits this model to create a data replication and synchronization tool for data sources. As a proof of concept, a data replication and synchronization tool has been implemented for The Cancer Imaging Archive (TCIA).**

I. Introduction

Data sources contain data of different granularity. Data is organized in a hierarchical structure, where different levels are used to present data in specific formats. Folders and documents make a good example of this. A unique identifier is assigned to each of the data units. A search across the data source would present the user with the list of matching criteria. Interesting sub set of the matching criteria may be bookmarked by the user and shared with others.

The sub set which is commonly known as a replica set, can be updated, duplicated, shared with other users, and deleted later. A search query may contain different parameters that can define the scope of the search and the outcomes, and often return the outputs in a finer granularity. When a sub set of such information is stored as a replica set, it is sufficient to store the unique identifiers of the matching data units of finer granularity than the original search query, as it would be sufficient to reproduce the data that is represented by the replica set.

II. Background

A. Representation of Medical Images in TCIA

Medical images are represented in multiple granularity. Figure 1 represents how the images are structured hierarchically in TCIA.

III. Design and Implementation

Having multiple instances running over different nodes provide fault-tolerance, as when one node terminates, the other nodes have the backup replica of the partitions stored in the terminated node. Figure 2 shows the higher level deployment view of the solution.

A. Generic Design

Two distributed cache instances exist in InfDataAccessIntegration.

```
protected static Cache userReplicasMap;
protected static Cache replicaSetsMap;
```

userReplicasMap is a mapping of userId → Array of replicaSetIDs. UserID could be the logged in user name. (for now, testing with random strings). replicaSetsMap is a mapping of replicaSetID → replicaSet.

Though this could be replaced with a single cache instance with the mapping of userID → replicaSets, having two cache instances will be more efficient during searches, duplicates, and push changes. Hence, two cache instances design was chosen.

$InfDataAccessIntegration$ implements the PubConsAPI for publisher/consumer, where $InterfaceManager$ implements the $InterfaceAPI$ for the interface between the data source and the data replication and synchronization tool. Invoker classes extending the abstract class InterfaceManager, implement the respective data source integration to invoke these

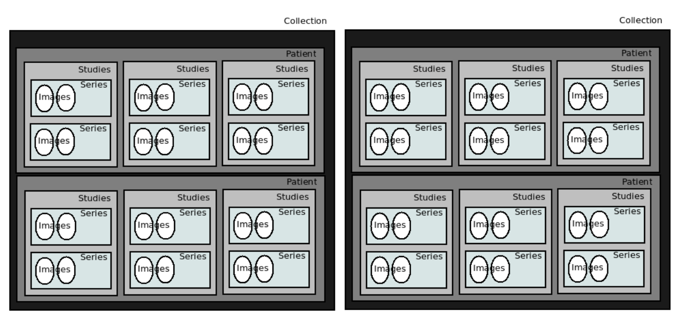

Fig. 1. Medical Images Granularity

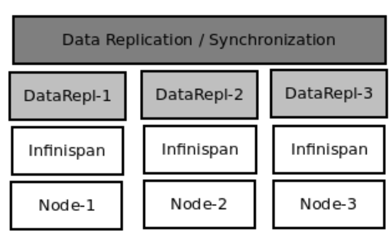

Fig. 2. Deployment

methods. The execution flow is depicted by Figure 3. When the user logs in, $logIn()$ checks whether the user has already stored replicaSets from the Infinispan distributed Cache. If so, execute them all again. This would be changed later as we do not have to execute all. Rather, we need to execute for the diffs. When the user performs new searches, for the images, series, collections, and the other meta data, the results will be returned to the user, and the user can chose ta subset of the returned results to create a replicaSet. The replicaSet for the

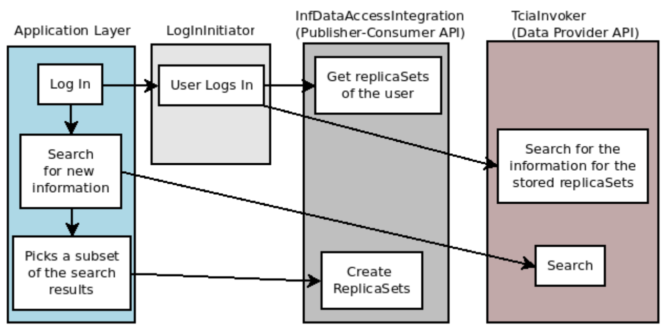

Fig. 3. Execution Flow

image will be as,

```
TCIAConstants.IMAGE_TAG + "getImage?SeriesInstanceUID=" +
     seriesInstanceUID
```

For other information (meta data), such as collections and seies,

```
TCIAConstants.META_TAG + query;
```

Here, query takes the below format.

```
"getSeries?format=" + format +
                 "&Collection=" + collection +
                 "&PatientID=" + patientID +
                 "&StudyInstanceUID=" + studyInstanceUID +
                 "&Modality=" + modality;
```

When a new instance starts now, and invokes the log in action for the same user, it will execute the queries for the stored replicaSets again, and reproduce the same results.

B. Design and Implementation for TCIA

Different complex data sources require custom development extending the generic framework. As creating and customizing the replicaSet require a more specific data structure, further implementations are done, extending the core class hierarchy. An extension based on the base design was developed for TCIA, as shown by Figure 4, which provides a core class hierarchy of the system.

$TciaInvoker$ extends $InterfaceManager$ to implement the interfacing layer between the TCIA data source and the data replication and synchronization tool. Meta data such as collections, patients, studies, and series are retrieved at different levels, though the default download manager of TCIA downloads the data in series level, composed of the images of the series in a single zip archive. While having the default userReplicasMap to contain the IDs of the replica sets for each user, the replica set itself is stored in multiple maps instead of a single replicaSets map, to provide an efficient storage and access.

$DataProSpecs$ extends the $InfDataAccessIntegration$ class. 5 maps are created as below to represent the replica sets.

```
protected static Cache<Long, Boolean[]> tciaMetaMap;
protected static Cache<Long, String[]> collectionsMap;
protected static Cache<Long, String[]> patientsMap;
protected static Cache<Long, String[]> studiesMap;
protected static Cache<Long, String[]> seriesMap;
```

$tciaMetaMap$ contains a boolean array, which reflects which of the granularity of meta data is selected as a whole. For TCIA, if a few collections are selected, the first element of the array is set to true, and similarly, the other meta data are marked to true or false as shown by the below code segment.

```
Boolean[] metaMap = new Boolean[4];
metaMap[0] = collection != null;
metaMap[1] = patientID != null;
metaMap[2] = studyInstanceUID != null;
metaMap[3] = seriesInstanceUID != null;

putReplicaSet(replicaSetId, metaMap);
```

The name of the collections, patientID, studyInstanceUID, and seriesInstanceUID are stored against the respective replicaSetID in collectionsMap, patientsMap, studiesMap, and seriesMap respectively. Hence changes are done at the respective maps. Duplicating the replicaSets duplicate the contents of the entire row to a new replicaSetID. Similarly, deleting a replicaSet deletes the respective information from all the maps.

TCIA public API provides methods to retrieve the images and meta data of different granularity. Figure 5 depict the methods that retrieve image and metadata at different granularity from TCIA. These methods are invoked by the replication manager tool to retrieve the images. As shown by Figure 5 an initial search on TCIA may contain parameters such as modality, in addition to collection name, patient ID, study instance ID, and series instance UID. However, each of these search returns the output in a finer granularity, and when a sub set of this finer granularity is selected, each of the selected elements are always identified by their respective identifier. Hence, storing an array of patient ID would be sufficient to identify the selected sub set of the collection including the array of patients. Similary an array of study instance UID is sufficient to represent the selected sub set of any patient, and an array of series instance UID is sufficient to represent the selected sub set of any study, as it will contain an array of series.

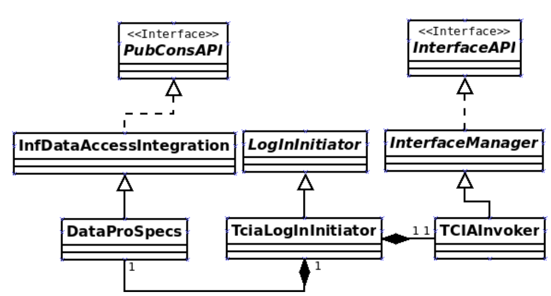

Fig. 4. Core Class Hierarchy

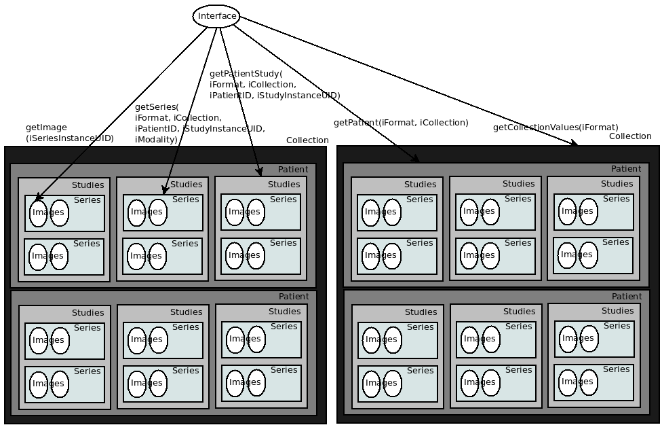

Fig. 5. Retrieving images and meta data

User Interface: Apache Velocity is used to generate the web pages for the replication tool. $DataRetriever$ class is invoked to retrieve the meta data from TCIA and to create and retrieve replica sets. $UIGenerator$ class invokes the Apache Velocity templates to create the web pages, presenting the data to the users. $DataRetriever$ has instances of $TciaInvoker$ and $TciaLogInInitiator$ to invoke the TCIA queries and replication tool create and retrieve replicaSets mechanisms. $TciaLogInInitiator$ has instances of $TciaInvoker$ and $DataProSpecs$, to invoke the public APIs provided by these classes for the data provider specification and publisher/consumer APIs. Figure 6 depicts the class hierarchy involved in the user interface.

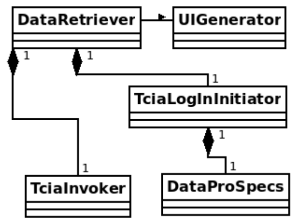

Fig. 6. Class Hierarchy of the User Interface

IV. Evaluation

V. Conclusion