University of Moratuwa

Department of Computer Science and Engineering

CS 4202 - Research and Development Project

Final Year Project Report

Schema-Independent Scientific Data Cataloging Framework

Project Group - DataMedici

S. W. S. Dhanushka (100097C)

A. L. H. S. Jayawardena (100225U)

M. K. D. S. Kumarasiri (100285C)

S. C. Nakandala (100352F)

Internal Supervisor

Dr. H. M. N. Dilum Bandara

External Supervisors

Dr. Srinath Perera

Mr. Suresh Marru

Dr. Sudhakar Pamidighantam

Coordinated By

Dr. Malaka Walpola

THIS REPORT IS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE OF ENGINEERING AT UNIVERSITY OF MORATUWA, SRI LANKA.

15th of May, 2015

Declaration

We, the project group DataMedici (S.W.S. Withana, A.L.H.S. Jayawardena, M.K.D.S. Kumarasiri and S.C. Nakandala under the supervision of Dr. H.M.N. Dilum Bandara, Dr. Srinath Perera, Mr. Suresh Marru and Dr. Sudhakar Pamidighantam) hereby declare that except where specified reference is made to the work of others, the project "Schema-Independent Scientific Data Cataloging Framework" is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgement.

Signatures of the candidates:

1. …….S.W.S. Dhanushka (100097C)
2. …….A.L.H.S. Jayawardena (100225U)
3. …….M.K.D.S. Kumarasiri (100285C)
4. …….S.C. Nakandala (100352F)

Supervisor:

……..

(Signature and Date)

Dr. H.M.N. Dilum Bandara

Project Coordinator:

……..

(Signature and Date)

Dr. Malaka Walpola

Abstract

Project Title: Schema-Independent Scientific Data Cataloging Framework

Authors: S.W.S. Dhanushka (100097C)

A.L.H.S. Jayawardena (100225U)

M.K.D.S. Kumarasiri (100285C)

S.C. Nakandala (100352F)

Internal Supervisor: Dr. H. M. N. Dilum Bandara

External Supervisors: Dr. Srinath Perera

Mr. Suresh Marru

Dr. Sudhakar Pamidighantam

Modern scientific experiments generate vast volumes of data which are hard to keep track of. Consequently, scientists find it difficult to search, reuse, and share these data sets. We address this problem by developing a schema-independent data cataloging framework for efficient management of scientific data. The proposed solution consists of an agent which automatically identifies new data products and extracts metadata from them. Agents can be used to extract any form of data by providing a data-specific plugin. Extracted metadata is then forwarded to a server, which indexes metadata using a NoSQL database. The server also provides a REST API for querying, sharing, and reusing data sets through a website, workflow, or science gateway. Moreover, the proposed solution enables fast indexing and querying of large number of metadata and attributes, dynamic metadata fields, as well as outperforming solutions based on relational databases. For example, our Apache Solr based implementation can resolve full text, sub-string, prefix, and suffix queries 91% - 99% faster than a MySQL-based implementation. Apart from developing a standalone system, we have integrated this into Apache Airavata scientific gateway middleware framework.

Acknowledgement

First and foremost we would like to express our sincere gratitude to our project supervisor, Dr. H.M.N. Dilum Bandara for the valuable guidance and dedicated involvement at every step throughout the process.

We would also like to thank our external supervisor Dr. Srinath Perera for the valuable advice and the direction given to us regarding the project.

In addition, we would like to thank Mr. Suresh Marru and his Airavata team, for the continuous support, encouragement and insightful comments. We would also like to thank Dr. Sudhakar Pamidighantam for guidance and assistance given to us.

We would like to express our warm gratitude to Dr. Malaka Walpola for coordinating the final year projects.

Last but not least, we would like to express our greatest gratitude to the Department of Computer Science and Engineering, University of Moratuwa for providing the support for us to successfully finish the project.

Table of Contents

List of Figures vii
List of Tables viii
List of Abbreviations ix
1. Introduction 1
1.1 Motivation 1
1.2 Contribution 3
1.3 Outline 4
2 Literature Review 5
2.1 Metadata 5
2.2 Workflow 6
2.3 Scientific Gateway 8
2.4 Apache Airavata 10
2.5 Scientific Data Output Formats and Parsers 13
2.6 Scientific Data Management Solutions 13
2.6.1 MCAT 14
2.6.2 Metadata Cataloging Service 16
2.6.3 MyLEAD 23
2.6.4 iRODS 27
2.6.5 Comparison 28
2.7 Database Technologies 28
2.7.1 Solr 29
2.7.2 Cassandra 30
2.7.3 Comparison of RDBMS, Solr and Cassandra 31
2.8 Agent Server Communication 32
2.8.1 SOAP 33
2.8.2 REST 33
2.8.3 Comparison of SOAP and REST 34
3 Problem Statement 35
3.1 GridChem Use Case 36
3.2 Airavata Use case 36
4 Design 37

4.1 Agent 38
4.2 Server 38
4.3 Web Portal 39
5 Implementation 40
5.1 Languages, Tools, and Technologies 40
5.2 Metadata Cataloging System 40
5.2.1 Agent 41
5.2.2 Server 43
5.2.3 Web Portal 45
5.3 Testing 46
5.4 Security Controls 47
6 Airavata Integration 48
6.1.1 RabbitMQ [39] 48
7 Performance Analysis 52
7.1 Data Insert Performance 53
7.2 Query Performance 54
7.2.1 Exact match queries 55
7.2.2 Range Queries 55
7.2.3 Full text queries 56
7.2.4 Prefix match queries 57
7.2.5 Suffix match queries 58
7.2.6 Wild card queries 58
7.2.7 Substring Queries 59
7.3 Space Utilization 60
7.4 Conclusion 61
8 Summary 62
8.1 Problems and Challenges 63
8.2 Future Work 63
9 References 65
Appendix I – Server Configuration Details 69
Appendix II – API Specification for DataCat Server 70
Appendix III – Agent Configuration Details 78

Appendix IV – Performance Test Results 79

List of Figures

Figure 1.1 Reuse for new studies . 2
Figure 1.2 Publish and Forget. 2
Figure 2.1 Example scientific workflow . 7
Figure 2.2 Science Gateway. 9
Figure 2.3 Airavata Functionality. 10
Figure 2.4 Airavata Architecture. 11
Figure 2.5 Airavata Stakeholders 12
Figure 2.6 MCAT Architecture 14
Figure 2.7 Attribute based discovery and accesing metadata services 18
Figure 2.8 Data model . 19
Figure 2.9 MCS Architecture 21
Figure 2.10 MyLEAD Architecture . 25
Figure 2.11 Service interactions during and experiment execution . 27
Figure 4.1 System architechture. 37
Figure 5.1 Web Portal. 46
Figure 6.1 Airavata datacat workflow. 49
Figure 6.2 Datacat deployment diagram 50
Figure 6.3 Basic search user interface 51
Figure 6.4 Advanced search user interface 51
Figure 7.1 Data Insertion Time. 54
Figure 7.2 Query execution time for exact match queries. 55
Figure 7.3 Query execution time for range queries. 56
Figure 7.4 Query execution time for full text search queries. 57
Figure 7.5 Query execution time for prefix match queries. 57
Figure 7.6 Query execution time for suffix match queries. 58
Figure 7.7 Query execution time for wild card queries. 59
Figure 7.8 Query execution time for substring queries. 60
Figure 7.9 Storage utilization. 61

List of Tables

Table 2.1 - MCS attributes and their respective categories. 20
Table 2.2 - System level requirements vs. Data model requirements. 24
Table 2.3 - Comparison of metadata cataloging solutions. 28
Table 2.4 General comparison between Solr and RDBMS 31
Table 2.5 - Comparison between Solr and RDBMS considering the search capability 31
Table 2.6 Comparison of Solr and Cassendra 32
Table 2.7 - Comparison of SOAP and REST 34
Table 7.1 - Schema for performance test. 53

List of Abbreviations

ACL	Access Control List
API	Application Programming Interface
CESM	Community Earth System Model
CIPRES	Cyber Infrastructure for Phylogenetic Research
ESG	Earth System Grid
Java EE	Java Enterprise Edition
FTP	File Transfer Protocol
GSI	Grid Security Infrastructure
HPC	High Performance Computing
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
iRODS	Integrated Rule Oriented Data System
LEAD	Linked Environments for Atmospheric Discovery
LIGO	Laser Interferometer Gravitational-Wave Observatory
JDBC	Java Database Connectivity
MAPS	Metadata Attribute Presentation Structure
MCS	Metadata Cataloging Service
OGSA-DAI	Open Grid Services Architecture Data Access and Integration
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

SQL	Structured Query Language
SRB	Storage Resource Broker
RDBMS	Relational Database Management System
REST	Representational State Transfer
WSDL	Web Service Description Language

1. Introduction

1.1 Motivation

We live in an age where vast volumes of data are getting created at every second by various means. These data are generated from various domains, and each domain usually has its own way of managing such vast volumes of data. It is said that data volumes from scientific experiments and simulations are doubling every year [1]. Consequently, with the increasing data types and volumes, scientists are finding it difficult to manage these data products. After executing scientific experiments, the generated data products are pushed into a data archive for long-term preservation. These data can be of different formats, e.g., binary, text, or XML. The hierarchical folder structure is often used to organize these data.

Typically, the scientific community use simple mechanisms to organize and store these datasets. Often they use hierarchical folder structures to organize the data. Because of this, when the data volume increases, the scientific community faces challenges such as [2]:

- Limited file and directory naming schemes.
- Scientists retrieve entire files to ascertain relevance.
- Inability to reuse and share the data products which have a high academic and research value with the broader scientific community.
- Important information and metadata relating to data products are usually in scientists notebooks and heads; hence, not accessible to others.
- Un-owned data after a large project.

If there is a mechanism to provide efficient searching of these data archives, then the stored data can be reused by the scientific community. For example, when a scientist wants to carry out an experiment, he/she can first search these data archives to see whether the same experiment is already done using the same inputs. If so, he/she can obtain the relevant output file(s), saving time and computing resources. This will change the lifecycle of scientific data from using once to reusing many times. This change is illustrated in Fig. 1.1 and 1.2.

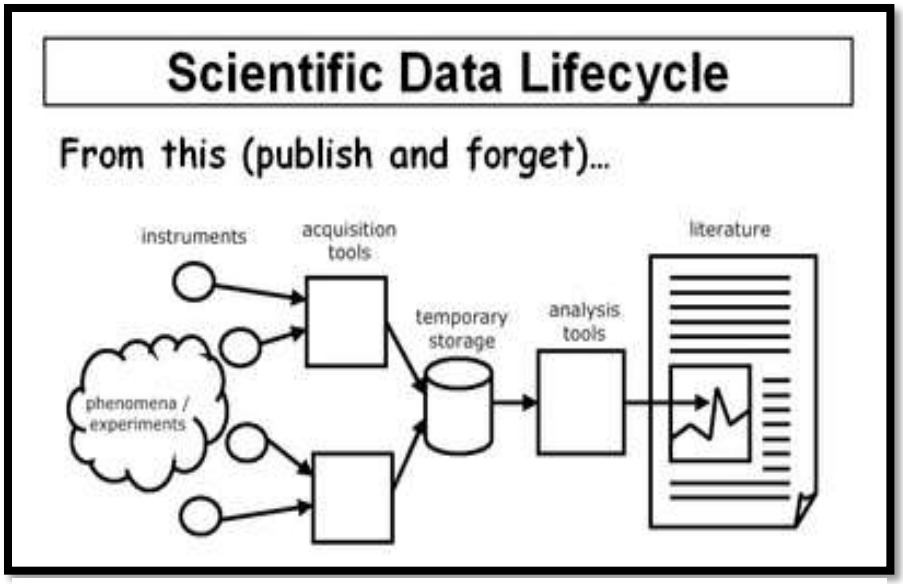

Figure 1.1 Publish and Forget [2].

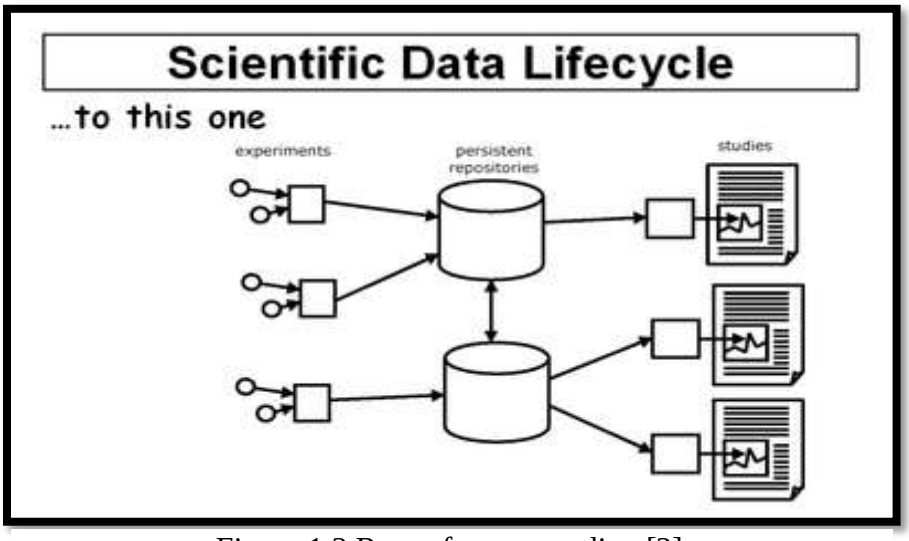

Figure 1.2 Reuse for new studies [2].

MCAT [6], MCS [7], and MyLEAD [8] are widely used scientific data management systems based on a metadata catalog. All three systems are tightly coupled to a specific computing infrastructure or use case. Also, as these solutions are based on relational database technology, they have a high response time when responding to more complex wildcard, substring, fulltext, and range queries. These solutions use static schemas for metadata types. Therefore, they are unable to support dynamic metadata fields (i.e., metadata fields whose existence were not known at the beginning, but found later) and cannot handle the problem of having metadata fields which are common only to some of the data products, efficiently. These limitations suggest that there is a need for improving the existing systems or revisiting the problem again.

1.2 Contribution

One of our use cases is based on "Gaussian 9" [3], which is a computational chemistry simulation application. Gaussian generates a vast volume of output data consisting of the output text file (*.out) and a binary checkpoint file (*.chk). The output file contains the details of the experiment results, whereas the checkpoint file contains the execution states and details of the experiment. Computational chemists parse these files to extract the information they need from the experiment and archive it.

We propose a scientific data catalog to easily and efficiently search large volumes of scientific data. Our approach is to extract and index the metadata in a metadata catalog, where scientists can search and find the results of scientific experiments. By keeping a detailed metadata catalog, search time and the overhead involved in searching through the entire dataset again and again can be reduced [4]. When a new data product is generated, the agent component of our system detects it and sends the file(s) to the parser system. The parser system, then extracts the metadata. This metadata is then indexed in the NoSQL database based on Apache Solr [5]. Moreover, the system provides a REST API for querying, sharing, and reusing datasets through a website, workflow, or science gateway. The novelty of our solution lies in the pluggable metadata extraction logic, extensible data product generation monitors, use of a NoSQL database, and the ability to dynamically add new metadata fields. Our system is also generalizable and can be used in other domains by replacing the metadata extraction logic.

The response time of our NoSQL-based implementation is much better compared to a traditional relational database implementation based on MySQL. We compared the performance for different query types such as exact, range, prefix, suffix, full text, and wild card. These tests were carried out for databases consisting of 100,000 records. For exact and range queries, MySQL-based implementation outperformed the Solr-based implementation. But in wildcard queries response time of the Solr-based implementation was 99.2% lower than the MySQL-based implementation. Similarly, response time for substring, suffix, and prefix queries were 97.7%, 99.2%, and 97.8% lower than the MySQL-based implementation, respectively.

While MCAT [6], MCS [7], and MyLEAD [8] are widely used scientific data management systems, they are tightly coupled to their specific use cases and utilize relational databases to index metadata. Compared to these application-specific systems, our solution can be applied to any

domain by replacing the metadata extraction logic. Moreover, the given REST API enables querying, sharing, and reusing data sets through many systems such as websites, workflows, or science gateways. Furthermore, its NoSQL implementation based on Solr enables fast resolution of natural language queries issued by scientists.

In our second use case, we integrate our system with Apache Airavata [9]. Apache Airavata is a software framework for executing and managing computational jobs and workflows on distributed computing resources including local clusters, supercomputers, national grids, and academic and commercial clouds. It acts as a middleware layer between scientific gateways [10] and computational resources. Scientific communities use Airavata to implement their own scientific gateway. By integrating our system to Airavata we integrate the searching and indexing capabilities to the scientific gateways which are built using Airavata.

1.3 Outline

This report is organized as follows. Chapter 2 discusses the existing literature which is relevant to the project. Chapter 3 presents the problem statement with our two use cases. Design of the system and its architecture are presented in Chapter 4. Chapter 5 presents the implementation details of the project including the tools and technologies, system components, testing, and security controls. Airavata integration is presented in Chapter 6. Experimental setup and performance analysis are presented in Chapter 7. Chapter 8 will conclude the report with problems encountered, challenges, and future work.

2 Literature Review

In this chapter we analyze the related work. The proposed solution for the problem is a metadata catalog. Therefore, understanding metadata has a significant value in our project. Section 2.1 presents a detailed description about metadata and their classifications. Section 2.2 will describe workflows, workflow management systems and their problems which lead to the development of scientific data management solutions. Science gateways manage scientific workflows on behalf of scientists. Section 2.3 presents about scientific gateways followed with Section 2.4 which describes Airavata, which is a scientific gateway middleware solution. Integrating our solution to Apache Airavata is an integral part of the project. For this we should have a good understanding on the main functionality of Airavata and its architecture. In Section 2.5 we have discussed the output formats and parsers scientists use to extract the relevant information from outputs. Several scientific data management solutions are presented by various research groups for specific domains. In Section 2.6 we describe popular and widely used scientific data management services; namely, MCAT, Metadata Cataloging Service, myLEAD and iRODS. Section 2.7 describes the database technology we chose to store metadata in our proposed system. To store metadata we had several options. We could use a RDBMS or a NoSQL database like Solr or Cassandra. Section 2.7 also describes RDBMS, Solr and Cassandra with their advantages and disadvantages with respect to our requirements. Section 2.8 presents the reasoning behind the selection of a REST architecture to implement the APIs of the proposed system. At the end of the chapter there is a summary of similar solutions and technologies which we use for our system.

2.1 Metadata

Metadata plays an important role in discovering the datasets for a given specific set of attributes. Keeping a detailed metadata catalogue can reduce the search time and overhead in searching through the entire dataset [4]. Most common metadata attributes are user defined and customized to fit specific environments.

Federal Geographical Data Committee [11] and Maryland State Geographic Information Committee [12] have defined a set of common attributes which make a generic metadata record:

- Metadata Record Information – Contains information about the metadata record such as the language it is written in, unique file identifier, metadata standard and the date that the metadata record is written.
- Content Information – Contains information about the actual datasets and attributes.
- Identification Information – Contains citation-level information like title, abstract, purpose for creation, status and keywords.
- Data Quality Information – Contains the information about the processes and sources used to develop data and accuracy assessments provided.
- Maintenance Information – Contains information about the updates like scope and frequency to data updates.
- Distribution Information – Contains information about the data distributors and methods for observing the data.
- Application Schema Information – Contains information about the schema or data models used to structure the data.

Another classification of metadata is as follows:

- Structural Metadata – Description of physical and logical framework of the data sets.
- Descriptive Metadata – Description about data objects such as title, author, subject, keywords and publisher.
- Technical Metadata – Description of mode of creation and storage such as indexes, data types and distribution lists.
- Administrative Metadata – Description of storage format, copyrights, licensing and preservation of datasets.
- Process Metadata – Description of information generated from scientific experiments.

Even though there exist very comprehensive standards for metadata, in practice most systems use a subset of the above metadata types to suit their specific requirements of modeling and managing.

2.2 Workflow

Scientific experiments and simulations can have one to thousands of steps. A step in a scientific workflow can be considered as a single activity or computation. A scientific workflow is a series

of structured activities and computations that arise in scientific problem solving [13]. Workflows can also be defined as a paradigm for representing and managing complex, distributed computations [14]. Scientific workflows are typically time consuming and produce large volumes of data. Therefore, managing them is a difficult task. Figure 2.1 represents an example workflow for functional genomics from the Taverna workflow system [15].

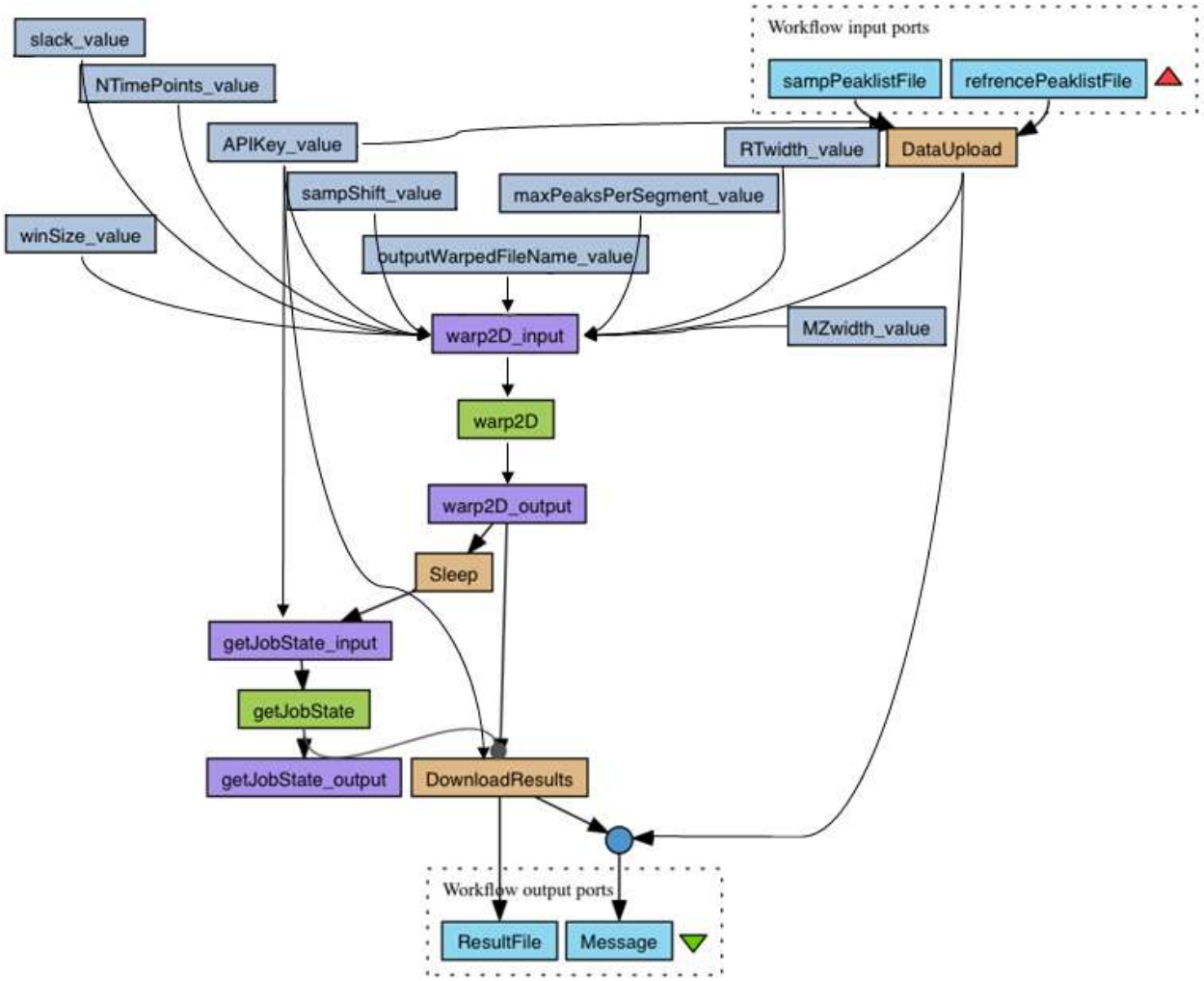

Figure 2.1 An example of a scientific workflow [15].

A workflow management system models and controls the execution of workflows in dynamic and heterogeneous environments [16]. It faces the following major issues when controlling workflows [14]:

- Reproducibility of scientific analyses and processes – Reproducibility is important in scientific workflows to prove the validity of a hypothesis. It also provides the basis for

establishing the known truths. Reproducibility requires rich provenance information. This requires provenance records to be indexed and made available for referencing. Therefore, a workflow management system must capture and generate the provenance information as a critical part of the workflow generated data.

- Scalability, reliability and security – The scientific data should be secured from malicious manipulations and access. They should be only allowed to be accessed by an approved community by the preference of the scientists. The workflow management system should be scalable since the workflows can generate large amounts of data in a single run. Moreover, it should accommodate the increasing numbers of workflows too. The system should always be available and reliable.
- Accommodate the inconsistencies and heterogeneities coming from different communities – When managing workflows, there can be various inconsistencies and heterogeneities because the information can come from different societies and sources. Therefore, there must be some mechanism to curate, validate, translate and integrate the data needed for sharing scientific information.

To overcome these issues scientists have to use a proper scientific data management system. Our proposed solution will address these issues.

2.3 Scientific Gateway

Scientific workflows use a variety of computational resources like super computers, HPC servers, work stations, etc. In a typical setup scientists have to go to the resource, add their experiment to the queue for the resource and wait till it gets the chance to get executed. This wastes scientists' valuable time which they can spend on their other scientific goals. Moreover, as we discussed in the previous section, there are various difficulties in managing workflows. To overcome these problems science gateways were invented. A science gateway is a community developed set of tools, applications and data collections that are integrated through a portal or a suite of applications [10]. Using a science gateway, scientists can handover the problem of assembling cyber infrastructure to run their experiment to the gateway middleware and focus on achieving more important scientific goals. Science gateways usually provide a graphical interface to the scientists to submit their experiments. Science gateways will handle all the other details, whereas the

scientists only have to log into the portal to execute experiments, track the status of the experiments and to get the results of those experiments.

These gateways are configured in a way that they will provide optimal usage of computational resources by enabling the entire scientific community through a common interface. Science gateways solve security, scalability and reliability issues in workflow management systems. A high-level architecture of a science gateway is shown in Figure 2.2 [17].

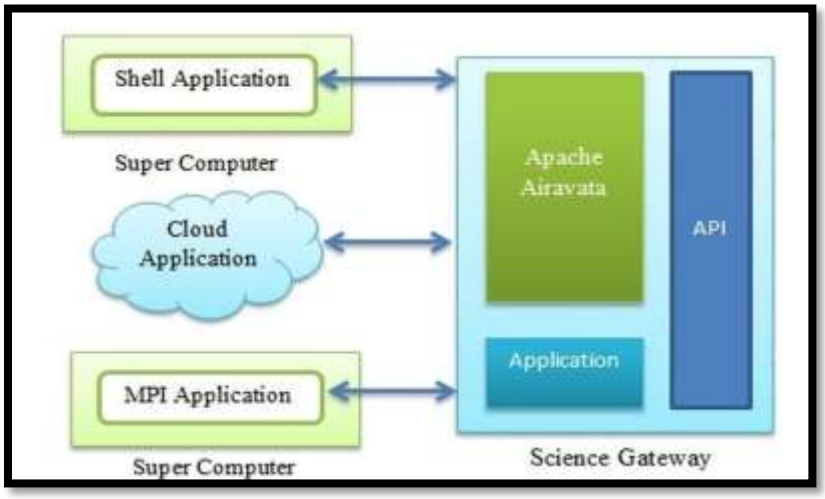

Figure 2.2 Science Gateway.

There are many science gateways operating around the world. A few major ones are:

1. CIPRES [18] – CIPRES is one of the most heavily used science gateways in the United States. It enables scientists to do phylogenetic reconstructions on a large scale which supports analysis of huge data sets containing hundreds of thousands of bio-molecular sequences. CIPRES stands for Cyber Infrastructure for Phylogenetic Research.
2. UltraScan [19] – The UltraScan gateway is used as a user-friendly web interface for evaluation of experimental analytical ultracentrifuge data using the UltraScan modeling software.
3. CESM [20] –The CESM gateway makes a world-leading, fully coupled climate model. It is easy to use and available to a wide audience.

A science gateway has to maintain communication with every computational resource in order to execute workflows. Since there are various types of resources including supercomputers, academic clouds, national grids, local clusters and commercial clouds, the protocols used to communicate

with these resources also vary. Therefore, this communication will give an unnecessary burden to the gateway. As a solution, a middleware layer dedicated to communicate with resources can be used. Then the gateway does not have to maintain the communication protocols or other properties about the resources.

2.4 Apache Airavata

Apache Airavata is one such solution which acts as the middleware layer between the science gateways and the resources and takes care of the experiments submitted by science gateways [9]. Its functionality is illustrated in Figure 2.3. Airavata composes, manages, executes and monitors distributed applications and workflows on computational resources ranging from local resources to computational grids [21]. It has been built on the general concepts of service oriented computing, distributed messaging, workflow composition and orchestration. All communications are secured using https in the transport layer.

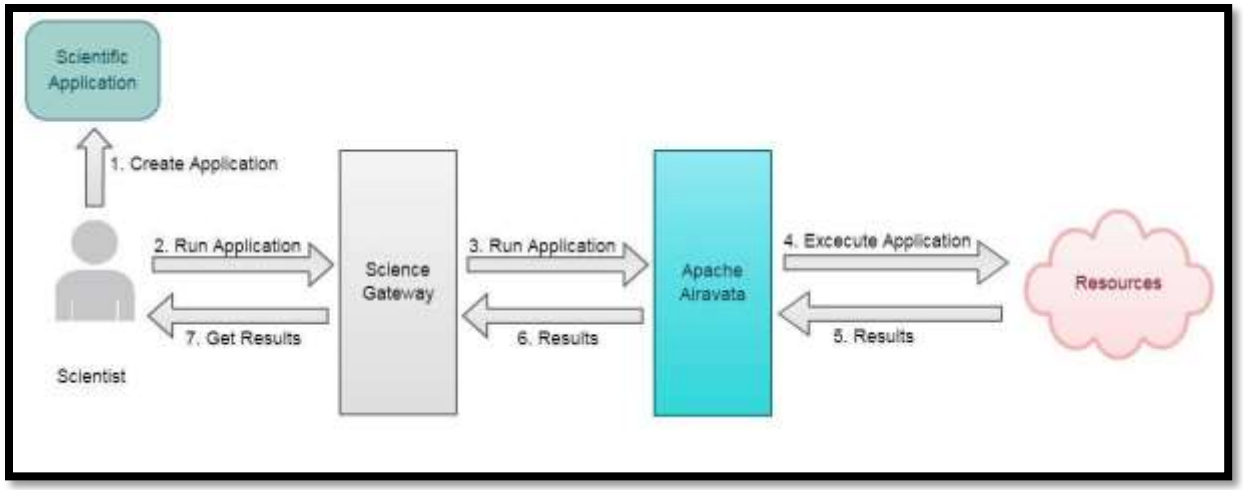

Figure 2.3 Airavata Functionality.

The main goal of Airavata is to support long running applications and workflows on distributed computational resources. The underlying architecture of Airavata is given in Fig. 2.4. Airavata provides the following features to the users:

- Desktop tools and browser-based web interface components for managing applications, workflows and generated data.

- Sophisticated server-side tools for registering and managing scientific applications on computational resources.
- Graphical user interfaces to construct, execute, control, manage and reuse scientific workflows.
- Interfacing and interoperability with various external (third party) data, workflow and provenance management tools.

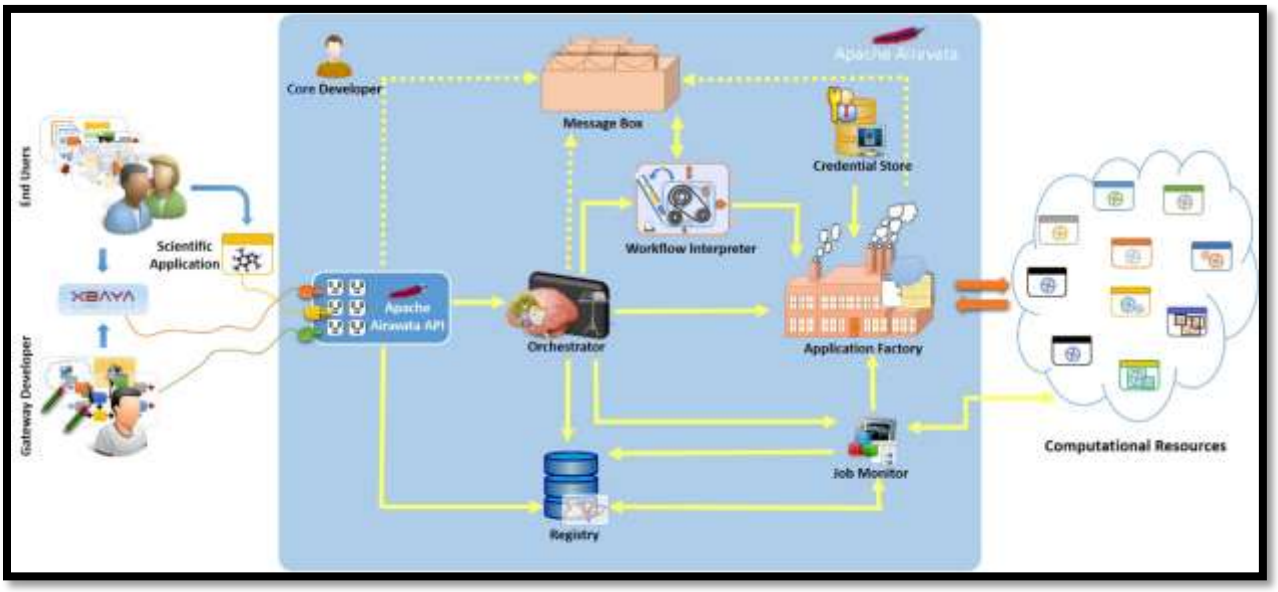

Figure 2.4 Airavata Architecture [9].

XBAYA workflow suite includes the GUI for workflow composition and monitoring. The workflows can be interpreted at each step, providing dynamic interactive capabilities. The composed workflow can be exported to various workflow languages like BPEL, SCUFL, Condor DAG, Jython and Java. Airavata has an application wrapper service called GFac, which is used to wrap command line driven science applications and make them into robust, network assessable services. It is built on Axis2 [22] web service stack and constructed workflows. There is a WS-Messenger component which is a publish subscribe based message broker implemented on top of Axis2. It facilitates the communication with clients behind firewall and solves network issues. It will handle the workflow execution engine notifications such as start, end, failure or successful invocation of the workflow execution. To put and get documents there is a thick client registry API.

Airavata interacts with various stake holders. These interactions are shown in Fig. 2.5.

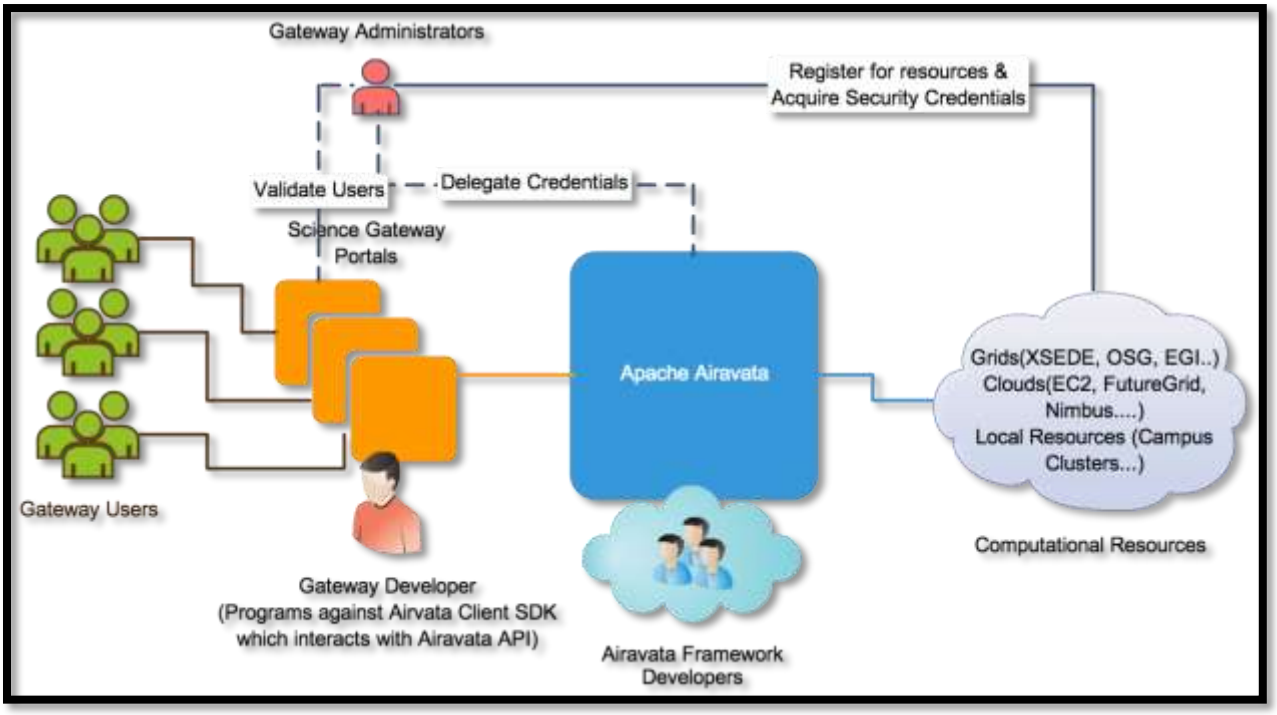

Figure 2.5 Airavata stakeholders [23].

- Gateway End Users – These users have a model code to do some scientific application. Sometimes this End User can be a Research Scientist. He/she writes scripts to wrap up the applications and by executing those scripts, they run the scientific workflows in super computers. These users interact with the portal and desktop user interfaces and are agnostic to the existence of Airavata.
- Gateway Developers - The job of the Gateway Developer is to use Airavata and wrap the above mentioned model code and scripts together. Then, scientific workflows are created out of them. The primary target for Airavata Client SDK's is gateway developers. They program against Airavata API (eased through client SDK's in Java, or PHP or C++ or JS) and build science centric gateway interfaces.
- Gateway Administrators - These users are responsible for operating a developed gateway and are primary targets for Airavata Administrative Tools which facilitate system level monitoring, security and user management and so forth.

- Airavata Framework Developers - These group of developers understand the internals of Airavata and develop, enhance, maintain the core software.

2.5 Scientific Data Output Formats and Parsers

The output files from scientific workflows can be in different formats. They can be in XML, binary or text format. In some situations metadata can be easily extracted from headers of these files. If not output files have to be parsed or mined in order to extract the metadata. If the file format is XML, the parsing becomes easy as we can extract the interesting properties from node values itself.

For example, in the Gaussian simulation experiments [3] in computational chemistry, there are two types of files which will be generated when an execution is finished. Those are the output file which is a text file and the checkpoint file which is a binary file. The checkpoint file gives all the execution states and details of the simulation experiment. This file is important to re-create or reinvestigate the experiment. The output file can be parsed to extract important information. Lexical parsers such as Cup [24] and JFlex [25] can be used to parse these text files. Domain knowledge is required to create these parsers. CUP stands for construction of useful parsers. It is a LALR (Look Ahead Left-to-Right) parser generator which generates parsers from a given specification [24]. JFlex is a scanner generator which is used with CUP files [25].

Typically scientists in a particular domain have the parsers to extract their required information from output files. A single output file can be parsed using different parsers to get different information extracted.

2.6 Scientific Data Management Solutions

There are several scientific data management solutions presented by various research groups. Here we will describe the most successful and widely used efficient scientific data cataloging systems.

2.6.1 MCAT

MCAT [6] is a metadata catalog implemented by San Diego Supercomputer Center (SDSC) as a part of the larger Storage Resource Broker (SRB) System. It has been designed to serve both core level and domain-independent metadata. There are four main elements in MCAT:

1. Resource – Computing platforms, communication networks, storage systems, peripherals
2. Methods – Access methods for APIs, system and user defined functions
3. Data Objects – Individual data sets and collection of data sets
4. Users and Groups – Who can create, update and access the resources, methods and data objects

MCAT architecture is illustrated in Figure 2.6.

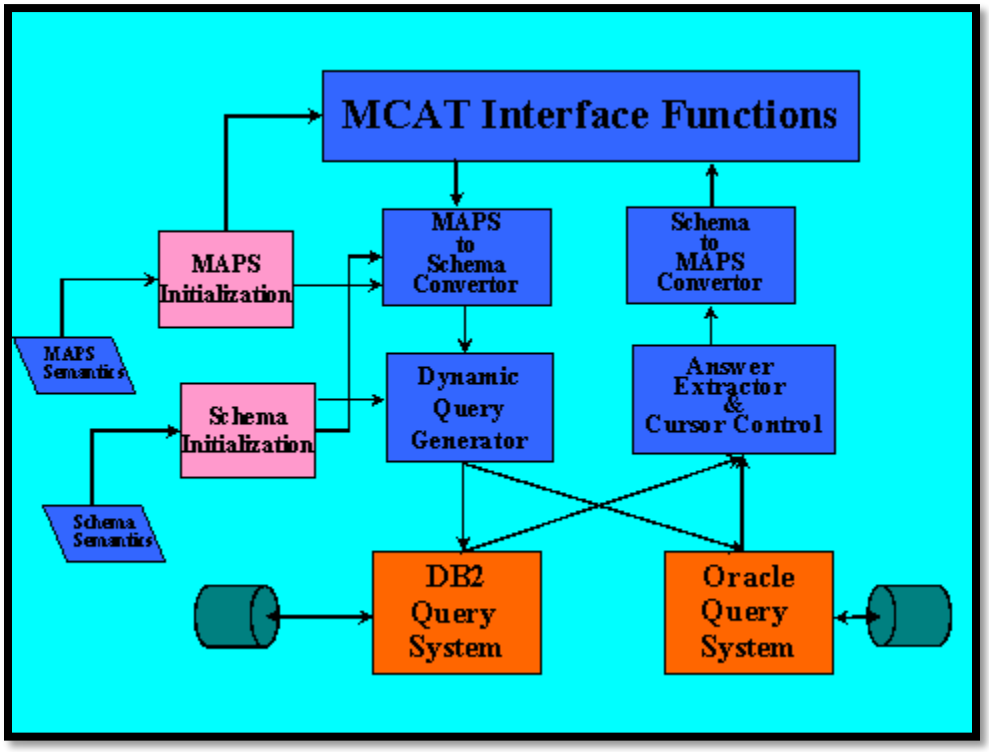

Figure 2.6 MCAT Architecture [6].

MCAT provides an interface protocol for the application to interact with the system. The protocol uses MAPS (Metadata Attribute Presentation Structure) data structure to represent metadata. The schema for storing metadata is different from MAPS; hence, it is essential to map the internal format with MAPS. Therefore, MCAT has used a MAPS to schema converter and a schema to MAPS converter. A dynamic query generator is used for internal query definitions, relationships

and semantics to develop an appropriate query. It is able to deal with multiple and heterogeneous catalogs internal and external to MCAT.

Data objects are first class objects in MCAT. Internal metadata are mostly extracted, derived or mined whereas external metadata are annotated or logged from external resources. Metadata attributes visible to MCAT user are given below:

- Name
- Type and formats
- Size
- Comments
- Liveliness
- Replica number
- Creation timestamp
- Creation owner

There is also an internal identifier metadata attribute which is not visible to the user. Every data object in MCAT is associated with a collection. A collection in MCAT is defined as a set of data objects or other collections. It is hierarchical, which allows navigation and searching recursively. Hence, a data object has another property called collection name. A data object cannot belong to more than one collection in MCAT. Since the data object resides in storage, metadata should also contain a link to the physical resource and the location inside the resource.

To control access there is an Access Control List (ACL), which contains data object id, user id and permission id. It also provides an abstract representation of resources where one can gather a set of physical resources to form a single logical resource with abstract properties. Each action on a data object is audited with its success or failure state. The audit trail record contains object id, user id, action id, timestamp and comments. Apart from the system oriented metadata mentioned above, there can be application oriented, domain oriented and user defined metadata.

In addition to the data centric view, MCAT also supports other views as well. In resource centric view the following metadata about the resources are stored:

- Name

- Type
- Access address
- Default location
- Replica number
- Comments

From a user centered view the following attributes are stored:

- Name
- Type
- Address
- Email
- Phone
- Pass phrase
- Domain
- User group

A user is given a type like privileged, normal, project, etc., which is used to define different capabilities. Moreover, a user is associated with at least one domain. Domain is a concept to distinguish users at a higher level. To apply access control user groups are used.

2.6.2 Metadata Cataloging Service

MCS [7] can be considered as a successor to the MCAT system. It has incorporated many features from MCAT and built on top of them. MCS has focused more on different types of metadata that has to be stored with relation to data products. The main types of metadata that get stored in MCS are as follows:

There are several types of metadata. One classification of metadata can be described as follows:

- User metadata – Metadata attributes such as annotations with data items or collections which individual users may want to associate with.
- Virtual organization of metadata – Additional set of metadata conventions for characterizing the data set defined by multiple scientific or corporate institutions in a virtual organization.

- Domain specific metadata – Domain specific metadata attributes are often defined by ontologies that are developed by application communities.
- Domain independent metadata – Domain independent metadata is general metadata attributes that apply to data items regardless of the application domain or virtual organization in which the data sets are created and shared.
- Physical metadata – Physical metadata relates to the physical characteristics of the objects such as their size, access permissions, owners and modification information.

In MCS metadata services are defined as services that maintain mappings between logical name attributes for data items and other descriptive metadata attributes and respond to queries about these mappings. They deal with all the other metadata except with physical metadata such as location replica services.

There are several processes in MCS. Publication is the process by which the data sets and associated attributes are stored and made accessible to a user community. Discovery is the process of identifying data items which are interesting to the user. Metadata services allow the users to discover datasets based on the value of descriptive attributes. Figure 2.7 shows a simple scenario of attribute based discovery and access using metadata services.

A sample execution of the system is given below based on the diagram:

1. Client application query to metadata service to find datasets with particular attribute values.
2. Metadata service responds with a set of logical name attributes for data items with matching attributes.
3. Client queries the replica location service.
4. Replica location service responds with the set of locations where the data items are resided.
5. Client selects and contacts the storage system where the actual data is.
6. Physical storage system returns desired data sets using GridFTP protocol.

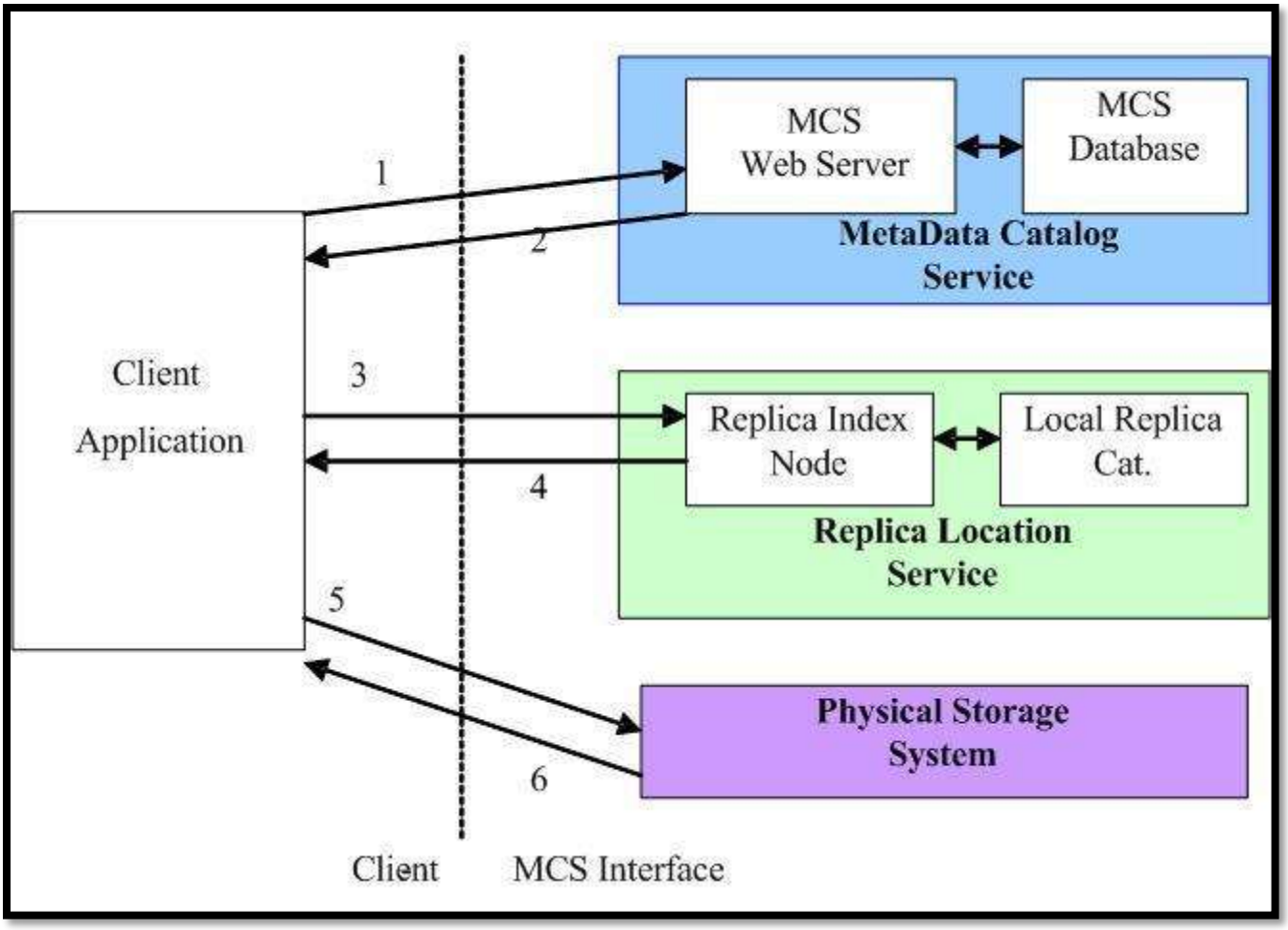

Figure 2.7 Attribute based discovery and accessing metadata services [6],

Various features that have been implemented by MCS to handle metadata efficiently are as follows:

- Providing a mechanism to associate logical name attributes with domain independent metadata attributes according to a predefined schema as well as user defined attributes that extend this schema to store domain independent, virtual organization and user metadata attributes.
- Supporting queries on its content.
- Implementing the policies regarding the consistency guarantees, authentication, authorization, and auditing capabilities provided by the service.
- Supporting the ability to aggregate metadata mappings into collections or views by associating aggregation attributes with logical name attributes.

- Providing good performance and scalability.

The creators of MCS argue that a metadata service is a simple extension of a database service which stores metadata attributes. It is a specialized service with the following components:

- A data model that includes mechanisms for aggregation of metadata mappings.
- A standard schema for domain independent metadata attributes with extensibility for additional user defined attributes.
- A set of standard service behaviors.
- Query mechanisms for accessing the database.
- A set of standard interfaces and APIs for storing and accessing the metadata.
- A set of policies for consistency access control, authorization and auditing.

MCS is designed based on those components. Its data model is a file-based data model. Figure 2.8 gives an illustration of the data model.

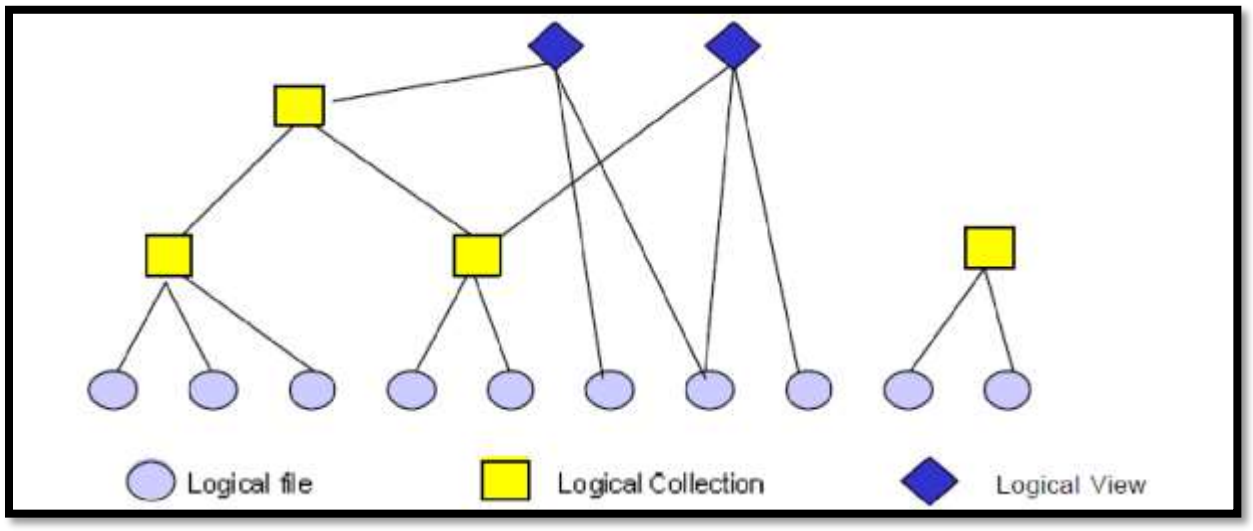

Figure 2.8 Data model [6].

A logical file is uniquely identified by a logical name. It is the basic item of the data model. Logical collections are user defined aggregations of logical files and collections. A logical view is another type of aggregation which contains logical files, logical collections and other logical views. For a consistent authorization logical files must be included in at least one logical collection. MCS supports a tree hierarchy of collections and simple queries that maps the logical files into logical collections. Authorization is given to logical collections. Logical views make users more flexible

in creating groups of logical collections and logical files. Logical views do not deal with authorization. Attributes in MCS schema is divided into logical categories. Table 2.1 will describe the attributes and their categories.

Table 2.1 - MCS attributes and their respective categories.

Categories	Attributes
Logical file metadata	Logical file name
	Data type
	Version
	Validity
	Container identifier and container service
	Creator
	Last modifier
	Creation time
	Last modification time
	Master copy location
	Audit information
Logical collection metadata	Collection name
	Description
	Composition
	Creator
	Modifier
	Audit information
	Parent logical collection
Logical view metadata	View name
	Description
	Composition
	Modifier
	Creator
	Audit information
Authorization metadata	Access privileges
User metadata	Writer
	Contact information
Audit metadata	Object identifier
	Audited action
	User
	Timestamp
User-defined metadata	
Annotation	Object identifier
	Object type
	User
	Timestamp
Creation and transformation history	

External catalog metadata	

.

MCS uses a web service model for its implementation. Figure 2.9 shows its components and their relationships.

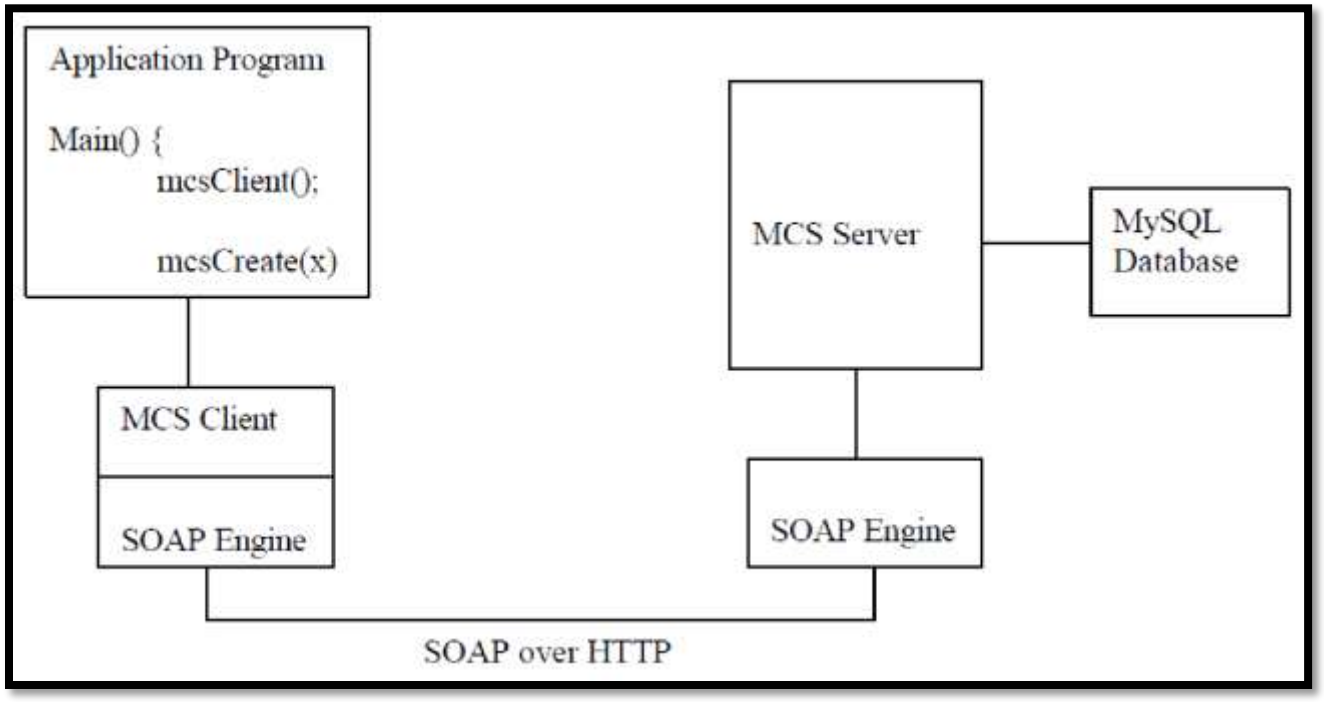

Figure 2.9 MCS Architecture [7].

The application program issues queries using an API. Client sends these queries to server using SOAP. MCS server interacts with the MySQL backend and returns the results back to the user. MCS provides a synchronous Java API which handles following operations:

- Querying for logical objects based on their attributes.
- Querying static attributes of a logical object.
- Querying the user defined attributes of logical object.
- Querying on logical collections and views.
- Creating a logical file, collection or view.
- Modifying the attributes of a logical object.
- Deleting logical file, view or a collection.
- Annotating a logical object.
- Adding logical objects in to views.

MCS is integrated to two applications; Pegasus system and Earth System Grid project.

2.6.2.1 Pegasus Application

Pegasus (Planning for Execution in Grids) is a planning component developed within GriPhyN project. It is used to map complex workflows into available resources. LIGO (Laser Interferometer Gravitational-Wave Observatory) uses Pegasus. Pegasus uses MCS to discover existing application data products. When a user requests logical files that includes a particular frequency band, MCS returns a list of results to the Pegasus planner. Since MCS only saves logical file names Pegasus uses an additional component for data cataloguing and discovery (Replication Location Service). For usage in LIGO, a predefined MCS schema and user defined attributes were adequate to provide the needed functionality.

2.6.2.2 Earth System Grid Application

ESG includes MCS with replica and storage management services. To include the metadata in MCS, domain specific attributes as well as user specified attributes parsed from the data has been used. MCS has proved that its functionality of querying and accessing metadata is adequate to ESG. But for the ESG, MCS capabilities were not enough. It needed a more flexible and simple mapping between data and metadata. Most of the general attributes which were predefined in MCS were not used in ESG. It also needed more query type flexibility.

The performance study by the MCS team has proved that MCS scales well with simple queries, but have limited performance with complex queries. The performance degradation is largely due to the MySQL penalty.

As mentioned earlier, MCS is an extension of MCAT metadata catalogue of the Storage Resource Broker from San Diego Super Computing Center. Some of the similarities between the two systems are as follows:

- Supports logical name space which is independent from physical namespace.
- Provides GSI authentication.
- Allows specification of a logical collection hierarchy.
- Support the notion of containers to aggregate small files.

Some of the differences between the two systems are as follows:

- Different architecture models
- MCAT is tightly coupled with other components of SRB and cannot be used as a standalone component whereas MCS is an orthogonal component from other services
- MCAT stores both logical metadata and physical metadata whereas MCS stores only logical metadata.

2.6.3 MyLEAD

MyLEAD is a personalized information management tool which extends the general Globus Metadata Catalogue Service into a general and extensible schema [8]. It plays a major role in large-scale distributed computational environments by characterizing grid and web services. MyLEAD handles the data management aspects of the LEAD (Linked Environments for Atmospheric Discovery) project which uses meteorological research into forecast and prediction of severe weather conditions.

The main problem with the existing scientific data management that is handled by myLEAD project is the inadequate tool support. The general tools which are used in the Internet cannot be adopted to use in most of the scientific data management scenarios. The main problems of the system can be listed as follows:

- The general sharing model in the Internet is "written by a single source and read-only by many". This model allows you to tag the data with bookmarks for later access. In scientific data handling, users need the ability to save the history and origin of the data.
- The Internet's default availability model for data objects is group or worldwide which complements the scientific data access default model which is user only. Although the availability has been set to user only in scientific data, the researchers or owners of the data should have the ability to make the data available to a broader audience.
- Information guard in general search engines is modifiable by scientific users. For example scientific users must be able to control the number, type and meaning of the attributes describing any given data objects.

To overcome these problems myLEAD team have come up with a personalized scientific metadata catalogue including search facilities, content storage, data object cataloguing, and active engagement of users.

MyLEAD is a metadata catalogue at its core. It stores metadata associated with data products generated and used in the course of scientific investigation. The data products can reside with the metadata in the database or in a separate storage. It will organize and keep track of all relevant information for experimental runs, associated documentation, generated data products and their provenance and run status.

Since this is designed for meteorological data analysis the input data for the prediction models comes from various sensors and have different types of formats. The system must be able to search the datasets, convert between formats and use the new products as the inputs to the prediction model. The requirements can be divided into two categories; system-level requirements and data model requirements. Table 2.2 represents the two categories of requirements.

Table 2.2 - System level requirements vs. Data model requirements.

System-Level Requirements	Data Model Requirements
Driven by system's Service Oriented Architecture (SOA)	Derived from the need to provide search, storage and browsing capabilities
Support concurrency	Rich search capabilities over personal collections
Employ a data replication mechanism to ensure fault tolerance	Rich metadata descriptions that includes application-level characteristics, usage and provenance
Secure user access	Ability to add attributes seamlessly to existing metadata descriptions
Real time catalogue product generation	

MyLEAD used MCS as its base because MCS provides dynamically expandable descriptions on metadata. The architecture of myLEAD tool is given in Figure 2.10. It has three major components; User interface, client-side services and server-side services.

- User Interface – User interface provides a portal to clients to interact with myLEAD via LEAD portal. The single portal may contain portlets to provide management, browsing and searching capabilities.
- Client side services – Client side services layer include the myLEAD agent and MCS client components. MyLEAD agent is a transient, short-lived service that serves a single user for a single session. Therefore, multiple agents can be existing at the same time and represent

users who are doing experimental runs. MCS client is embedded in that agent. It is a java interface to the myLEAD service.

- Server side services – MyLEAD server service is a persistent grid service which is built on top of a relational database management system (RDMS). MCS is built on top of the Open Grid Services Architecture Data Access and Integration web service (OGSA-DAI). It provides a set of generic web services to access any database management system. It connects to the database with Java Database Connectivity (JDBC). MCS extends OGSA-DAI by providing the ability to access particular tables in the database schema. MyLEAD extends this schema further by supporting spatial and temporal attributes.

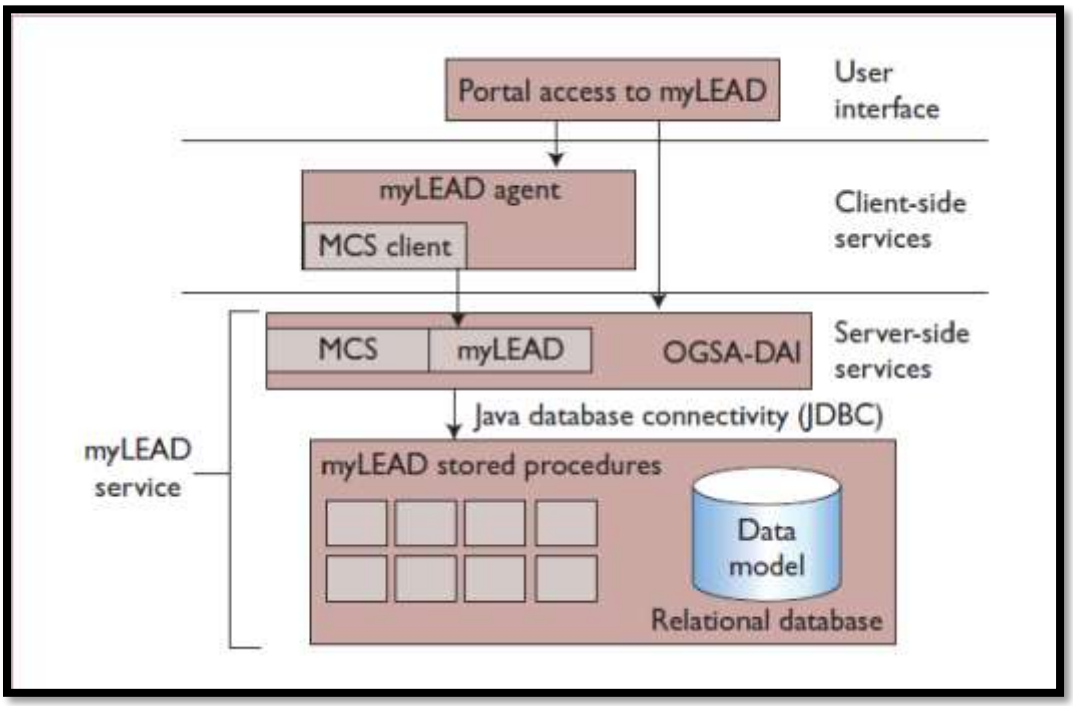

Figure 2.10 MyLEAD Architecture [8].

MyLEAD data catalog consists of investigations and collections comprising one or more logical files, views or groups of logical files that belongs to multiple collections, user defined complex attribute types, pre-defined spatial and temporal attribute types and abstraction hierarchy from low level vocabulary to high level concepts.

To offer more intuitive querying than MCS, myLEAD extends OGSA-DAI modular design allowing additional activities. It also allows bulk loading capabilities that MCS does not provide.

MyLEAD agent is a transitory grid service that works on the user's behalf to negotiate with other services in generating metadata and storing, recording and accessing data products generated and used during investigations. It represents the user negotiations with other web services, initiates file movement to the storage repository, ensures consistency between storage and myLEAD, and manages local space available to the user.

One of the unique characteristics which make myLEAD distinct from previous metadata catalogues is that its agent can take over more responsibilities like decision making while in other systems it is only limited to database access.

Figure 2.11 shows the service interactions that take places during an experiment execution. The step by step process is as follows:

1. User creates a new experiment at the portal.
2. Catalog creates a myLEAD agent to act on user's behalf and it makes a connection with the resource
3. User configures the workflow script and launches it at the designated site.
4. Sequence execution of the script with forecasting model
5. Consult the myLEAD agent to find space to store generated data.
6. Save the generated data on the proper location
7. Script executes the data mining task on the results
8. Record the analyzed results in persistent storage
9. Add the metadata to metadata catalog

As myLEAD is based on a relational database system it has a static schema. But evolving scientific data has shown the need of a dynamic schema for the metadata catalogue. With the development of NoSQL languages this has been feasible.

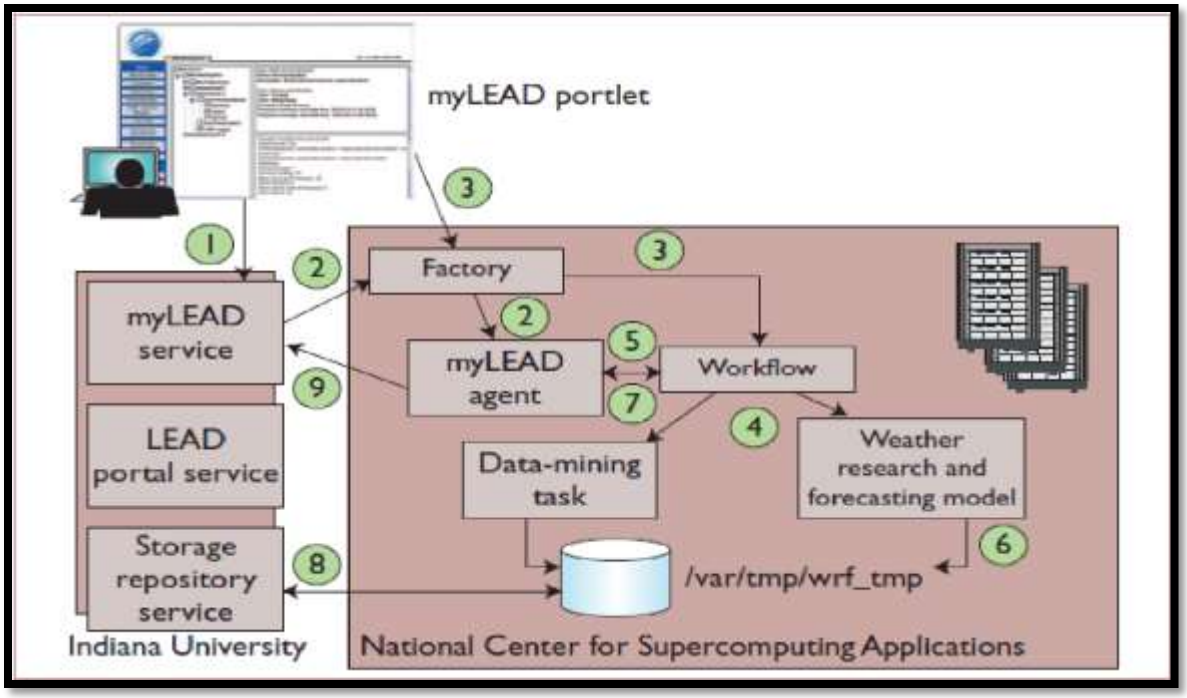

Figure 2.11 Service interactions during and experiment execution [8].

2.6.4 iRODS

iRODS (integrated Rule-Oriented Data System) is an open source data management software in use at research organizations and government agencies worldwide [26]. Advantages of iRODS include:

- iRODS enables data discovery using a metadata catalog that describes every file, every directory and every storage resource in the data grid.
- iRODS automates data workflows, with a rule engine that permits any action to be initiated by any trigger on any server or client in the grid.
- iRODS enables secure collaboration, so users only need to log into their home grid to access data hosted on a remote grid.
- iRODS implements data virtualization, allowing access to distributed storage assets under a unified namespace and freeing organizations from getting locked into single-vendor storage solutions.

2.6.5 Comparison

Table 2.3 summarizes the comparison between the above mentioned metadata cataloging services.

Table 2.3 - Comparison of metadata cataloging solutions.

	MCAT	MCS	MyLEAD	iRODS
Ability to use independently	An integrated component of Storage Resource Broker. Cannot be used independently.	A component in a larger grid software infrastructure.	Tightly coupled to the LEAD project. Cannot be used in a different scenario.	Comes as a complete grid middleware.
Database technology	Combination of IBM DB2 & Oracle Database.	MySQL Database.	MySQL Database.	MySQL Database.
Support Access controlling	Yes	Yes	Yes	Yes
Supporting logical collections	Yes	Yes		
Search operations	Attribute-value based search.	Attribute value-based search.	Attribute value-based search.	Attribute value-based search.
Handling provenance	No explicit notion of provenance information.	No explicit notion of provenance information.	Stores & manages provenance information relating to data products.	Stores & manages provenance information relating to data products.
Data mining	No notion of data mining.	No notion of data mining.	Supports data mining tasks on data on data products.	No notion of data mining.
Communication protocols/ architectures	Service- oriented architecture	Service-oriented architecture	Client server architecture	

These existing systems are tightly coupled with their grid infrastructures. They use relational database management systems where the querying is not flexible. Our proposed solution should have various query type support such as wild card queries, substring queries and fielded queries. These systems do not handle dynamic metadata fields. Our requirement is a system with a generalizable framework which will support flexible querying. Therefore, we cannot use these existing metadata catalogs for our requirement.

2.7 Database Technologies

For the implementation of the proposed scientific data cataloging system we need to choose the best technologies out from vast amount of existing ones. Previous metadata catalog systems used Relational Database Management Systems (RBDMS) to store and search the metadata. Another

alternative is to use No-SQL database solutions. In the preceding section we elaborate on the potential technologies that we considered, the technologies that we ultimately selected and the rationale for choosing them.

2.7.1 Solr

Solr is an open source Apache subproject which provides mature Java-based indexing and search technology [27]. It is built on Apache Lucene [28]. Solr is a standalone enterprise search server with a REST-like API. You put documents in it (called "indexing") via JSON, XML, CSV or binary over HTTP. You query it via HTTP GET and receive JSON, XML, CSV or binary results [29]. Solr features will be given as follows:

- Advanced full-text search capabilities- since it is powered by Lucene, Solr enables powerful matching capabilities including phrases, wildcards, joins, grouping and much more across any data type.
- Optimized for high volume traffic
- Standards Based Open Interfaces - XML, JSON and HTTP
- Comprehensive administration interfaces
- Easy monitoring- Solr publish metric data via JMX.
- Highly scalable and fault tolerant- since it is built on Apache Zookeeper, Solr makes it easy to scale up and down. Solr bakes in replication, distribution, re-balancing and fault tolerance out of the box.
- Flexible and adaptable with easy configuration
- Near real time indexing
- Extensible plugin architecture
- Data driven schema less mode
- Rich document parsing
- Multiple search indices
- Dynamic Fields enable on-the-fly addition of new fields that auto-map to field types based on the field name
- Text analysis components including word splitting, regex, stemming

Table 4 summarizes the comparison between Solr and RDBMS.

2.7.2 Cassandra

Another possible alternative for data storage is Cassandra NoSQL database. It is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing a highly available service with no single point of failure [30]. Advantages of using Cassandra as a database solution can be given as follows [31]:

- Fault tolerance- Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.
- High performance- Cassandra consistently outperforms popular NoSQL alternatives in benchmarks.
- Decentralization- There are no single points of failure. There are no network bottlenecks. Every node in the cluster is identical.
- Durability- Cassandra is suitable for applications that can't afford to lose data, even when an entire data center goes down.
- Elasticity- Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

When using as a database choice we should consider following limitations of Cassandra as well [32].

- No join or subquery support, and limited support for aggregation. This is by design, to force you to de-normalize into partitions that can be efficiently queried from a single replica, instead of having to gather data from across the entire cluster.
- Ordering is done per-partition, and is specified at table creation time. Again, this is to enforce good application design; sorting thousands or millions of rows can be fast in development, but sorting billions in production is a bad idea.

Table 5 gives a summarized comparison of Cassandra and Solr.

2.7.3 Comparison of RDBMS, Solr and Cassandra

In Table 2.4 Solr and RDBMS is compared according to general characteristics of databases. Table 2.5 compares the two databases with their search options.

Table 2.4 General comparison between Solr and RDBMS [33].

	Solr	Relational DB
Text Search	Fast and sophisticated	Minimal and slow
Features	Few, targeted to text search	Many
Deployment Complexity	Medium	Medium
Administration Tools	Minimal open source projects	Many open source & commercial
Monitoring Tools	Weak	Very Strong
Scaling Tools	Automated, medium scale	Large scale
Support Availability	Weak	Strong
Schema Flexibility	Must in general rebuild	Changes immediately visible
Indexing Speed	Slow	Faster and adjustable
Query Speed	Text search is fast & predictable	Very dependent on design & use case
Row Addition/Extraction Speed	Slow	Fast
Partial Record Modification	No	Yes
Time to Visibility After Addition	Slow	Immediate
Access to Internal Data Structures	High	None
Technical Knowledge Required	Java (minimal), web server deployment, IT	SQL, DB-specific factors, IT

Table 2.5 - Comparison between Solr and RDBMS considering the search capability [27].

Solr	RDBMS
Adept in full text search	Adept in data modeling and transaction.
Field type is only keyword	Field types char, varchar, datetime, int, etc., can exist
Query time is independent of the data amount for exact queries. Less query time than RDBMS. Solr uses an inverted index.	Query time of unindexed RDBMS has a monotonic relationship with the amount of data for exact queries. When it is indexed using a clustered index the query time is similar to Solr.
For wildcard prefix queries Solr performs well than unindexed RDBMS.	Indexed RDB performs well on prefix queries. Sometimes it outperforms Solr.
Query time of full wildcard queries in Solr has the same monotonic relationship with the data amount as RDBMS. However, full wild card queries are often performed on one field in Solr, if type of the field is set to text; it will be tokenized by the analyzer and support full text search.	Query time of full wildcard queries in RDBMS has the same monotonic relationship with the data volume as Solr.
Since Solr is implemented in Java it results in lower efficiency in program executing, memory usage and file I/O	The most commonly used SQL server databases are encoded in binary.
For the combination queries, Lucene query time increases slowly with the number of elements in the combination.	Query time depends on whether the query fields are indexed or not.

Query time increases with record complexity	Unindexed RDBMS's performance is same as Solr. Query time of the indexed RDBMS is independent of record complexity.

Our metadata catalog should have several search options like full text search, wildcard queries, substring queries and exact match queries. When going through the above comparisons we can say that Solr will be much suitable for the implementation of our metadata storage than RDBMS. Table 2.6 gives a summarized comparison between Solr and Cassandra.

Table 2.6 Comparison of Solr and Cassendra [34].

	Solr	**Cassandra**
Database model	Search Engine	Column Store
Data scheme	Exists a schema	Schema free
APIs and other access methods	Java API, RESTful HTTP API	Proprietary Protocol
Replication methods	Cloud/distributed (via Zookeeper) Master-slave application	Selectable replication factor
Map reduce	no	yes
Consistency concepts	Eventual consistency	Eventual consistency, Immediate consistency
Secondary indexes	yes	restricted

For the implementation of the metadata storage we considered using Cassandra because of its performance. But since our system needs flexible query support, search functionality and since Cassandra has a larger footprint in terms of resource utilization we chose Solr as our database solution.

2.8 Agent Server Communication

For the communication between agent and server, we considered two options. They are SOAP and REST based web services. We consider REST or SOAP over basic TCP-based solutions because they provide better support for implementation.

2.8.1 SOAP

SOAP is a well-developed protocol used in web industry to exchange the structured information in the implementation of web services in a network. WSDL (Web service description Language) is the standard used with SOAP to make the messages available over the web via web services. This protocol encodes messages so they can be delivered over the network using a transport protocol such as HTTP, SMTP, FTP or others.

2.8.2 REST

REST is a set of architectural principles designing web applications which is gaining popularity because of its simplicity, scalability and architectural dependence on the World Wide Web [35]. REST principles focus on how resource states are addressed and transferred over HTTP, by a wide range of clients written in different languages [36]. The main idea behind REST is to use well developed HTTP for transferring data between machines, rather than using a protocol that works on top of the HTTP layer for message transfers such as SOAP [35].

The key features of REST can be given as follows [37]:

- Use HTTP methods explicitly- This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping, POST will create a resource in the server, GET will retrieve a resource, PUT will update a resource and DELETE will remove a resource. These verbs are already defined by the protocol.
- Stateless- REST Web services need to scale to meet increasingly high performance demands. Clusters of servers with load-balancing and failover capabilities, proxies, and gateways are typically arranged so that they form a service topology, which allows requests to be forwarded from one server to the other as needed to decrease the overall response time of a Web service call. Using intermediary servers to improve scale requires REST Web service clients to send complete, independent requests; that is, to send requests that include all data needed to be fulfilled so that the components in the intermediary servers may forward, route, and load-balance without any state being held locally in between requests.

- Expose directory structures like URIs- From the standpoint of client applications addressing resources, the URIs determine how intuitive the REST Web service is going to be and whether the service is going to be used in ways that the designers can anticipate. URIs should also be static so that when the resource changes or the implementation of the service changes, the link stays the same.
- Transfer XML. JSON or both- This is allowed using MIME types and HTTP accept headers.

2.8.3 Comparison of SOAP and REST

Table 2.7 - Comparison of SOAP and REST [38].

SOAP	REST
A protocol	An architecture
Tightly coupled system	Loosely coupled system
Assumes point to point communication model	Designed for distributed computing environments
Less verbose	More verbose
Built in error handling	No error handling
Not reliable	Reliable

For proposed scientific metadata catalog we need a reliable, lightweight solution which can be easily integrated to web applications. SOAP gives a huge overhead in terms of metadata and latency in communication. Therefore we chose REST for our implementation.

3 Problem Statement

When a scientist needs some data from an experiment, he/she has to execute the experimental workflow by submitting it to a science gateway. After the execution is completed, the scientist extracts the data he/she wants from the output and discards the output or saves it in a data archive. When the same scientist or some other person wants to run the same experiment with the same inputs, he/she has two options; (1) run the experimental workflow from the beginning or (2) search the results from the output data archive to see whether there exists any reusable data product that matches his/her requirement. Typically, these experimental workflows are long running and need machines with high computing power like super computers, clusters, or workstations. If the workflow is to execute from the beginning, a lot of time and resources are required. If the scientist decides to search for the achieved data for experimental or simulation results, it is time consuming as the output data archives typically contain terabytes of data. Therefore, an efficient data management system is needed to reduce the time spent on generating new data (by promoting data reuse) or searching existing data, while reducing the resource consumption.

Our research problem is to develop a framework to enable efficient and effective retrieval and sharing of scientific data. This will allow scientists to easily reuse the scientific data outputs. The proposed framework should have the following features:

- Easily searchable – The output data should be easily searchable through the framework so that scientists are able to search the output of previously executed workflows. This should allow search queries of various types including full text search, wildcard search, substring search, fielded search, etc.
- Allow sharing with access control mechanisms – Scientist belong to different scientific sub-communities. They should be able to share their experimental outputs with their relevant sub-communities as well as with the general public. To allow this we have to provide access control at multiple levels.
- Automated metadata generation – To make the data searchable, we have to generate meta-data from the workflow outputs and build a catalogue using those metadata. So every time a workflow is executed and outputs are generated, metadata for the data products should also be generated and should be added to the catalogue so that it is searchable.

Our project deals with two separate use cases. They are the Gridchem use cases and Airavata use case.

3.1 GridChem Use Case

The Computational Chemistry Science Gateway (GridChem) includes desktop software and middleware services to integrate molecular science applications and tools for community use [39]. It provides a comprehensive end-to-end solution for managing jobs, workflows, allocations, and data, as well as processing simulation results through a graphical user interface.

Gridchem use case deals with computational chemistry simulation data of "Gaussian 9" [3] experiments executed via the GridChem gateway. The output files have many important attributes like RMS force, number of iterations, atomic number. Typically, scientists use parsers to extract the data that they need from the output files and put the file into the archive. The scientists in the Gridchem scientific community sees this as a waste of data and resources. Hence, they need some mechanism to efficiently extract the important experimental data without spending time for search them from the data archive. Another requirement they specifically mentioned is that, right now they don't have a proper way of sharing the experimental data with the community of a group of communities. The experiments are visible only to the scientist who ran it. Therefore the solution should also contains an access control mechanism to allow sharing of the output data.

3.2 Airavata Use case

Currently, Apache Airavata middleware is used by many scientific communities to implement their own scientific gateway. A large number of scientists use Airavata daily to submit and execute their computational tasks, workflows on the grids or supercomputers. This results in the generation of vast volumes of scientific data which eventually get abandoned in the computational resources. Currently, Airavata does not provide any mechanism to support search, browse or reuse this data. In this project we hope to tackle this problem and enable the scientist who is using Airavata to have better control of the data products that are generated by him/her or any data products that are shared with him/her.

4 Design

This chapter describes the design of the metadata cataloging system we propose, to easily and efficiently search large volumes of scientific data. The system architecture consists of agent, server, and the web portal. In our use cases we assume that all the data products will be pushed to a data archive after their generation. The agent continuously monitors to identify the generation of new data products. Once it discovers such a data product(s), it generates a notification to the metadata extractor. Metadata extractor then extracts or mines the metadata and important attributes, and publishes those to the server. Server uses Solr, an open-source search platform as its backend data store. It provides a query API (Application Programming Interface) for the web portal users to query on metadata and extracted attributes. Figure 4.1 will give the high level view of the system architecture.

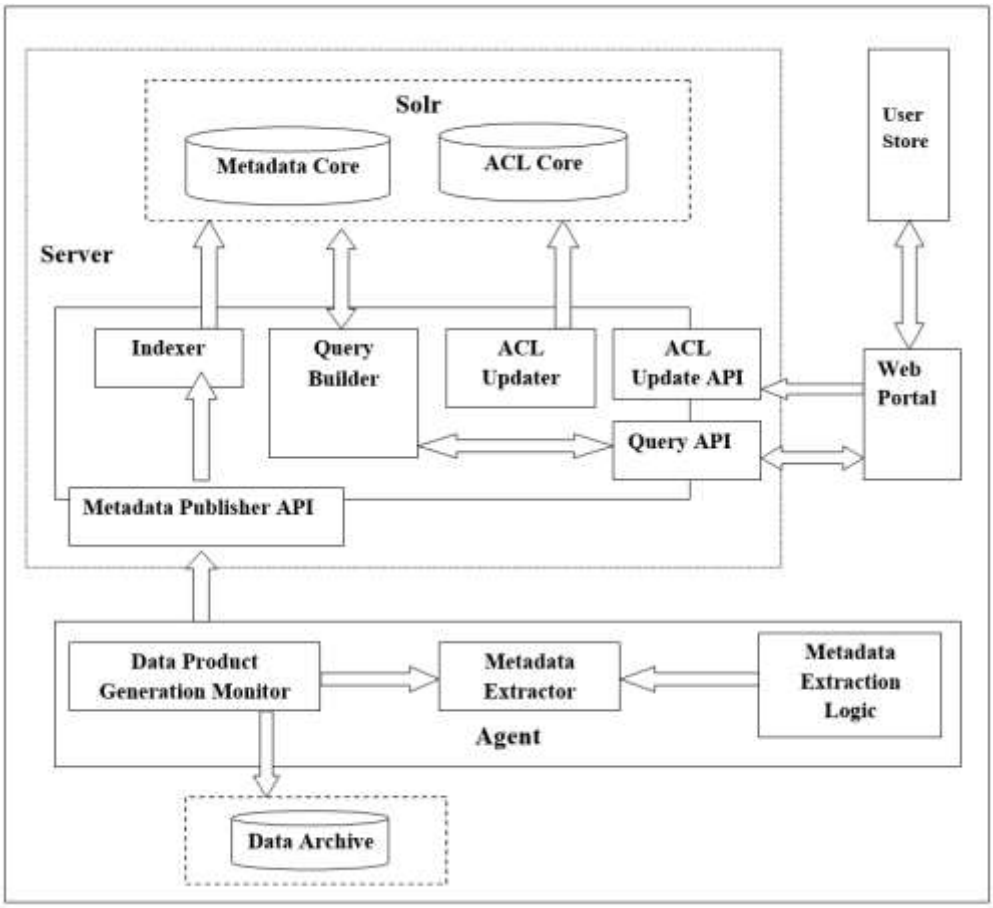

Figure 4.1 System architecture.

4.1 Agent

Agent sub system consists of three components; data product generation monitor, metadata extractor and metadata extraction logic. Data product generation monitor is the component responsible for detecting the generation of a new data product. This monitoring can happen in various ways depending on the system. It can be implemented as a file system monitor or message broker. The system is designed to be able to replace this component to extend the system to use with any external system.

When data product generation monitor identifies a new data product it will send a notification including the physical location of the data product to metadata extractor component. If the file system already contained data products before the agent was started, the data product generation monitor will create notifications for every existing data product. This is essential to make sure all existing data products are indexed. It is also possible to have more than one data archiving locations. In such a scenario multiple agent instances have to be run in each data archiving node.

When this notification is received the metadata extraction component extracts the metadata using metadata extraction logic. Metadata extraction logic contains a set of domain specific parsers to extract metadata and other important attributes.

4.2 Server

The server is the central component in our system where all the metadata and access-control information is maintained. The server provides three service APIs. They are metadata publisher API, query API, and access control list update API. The metadata publisher API is used by the agents to publish the generated metadata objects. The query API is used by the web portal to make queries on behalf of the users. The web portal can also use access control list update API to change the access control lists maintained for the particular metadata object. All these APIs are implemented as REST (Representational State Transfer) based web services.

The server uses Apache Solr, an open-source search platform, as its backend data store. It provides mature Java based indexing and search technology [5]. We selected Solr over a relational database management system like MySQL, as the backend because of its flexible query support and search functionality. Metadata and ACL information will be stored in Solr database.

4.3 Web Portal

The web portal is the primary entry point for users to access the query service provided by the server. Users can use the web portal to generate queries using the graphical controls. Experienced users should be able to directly submit the query in the form of a string. Before submitting a search query, users need to login to the system using their credentials. If not, users can retrieve only the publicly shared information. It is also possible to integrate the web portal with an existing user store, used by the scientific community.

The main functions of the web portal can be summarized as follows:

- Users should be able to perform a full text search.
- Users should be able to use graphical constructs to generate a more advanced query.
- Users should be able to select the type of query they want to perform.
- Users can issue a query based on the data product creation date.

5 Implementation

This chapter gives a detailed description about implementation of our system. Section 5.1 describes the languages and technologies used. Section 5.2 describes the implementation details of each component in the system. Finally, Section 5.3 discusses the testing process. This section will conclude after discussing the security controls used to implement the solution.

5.1 Languages, Tools, and Technologies

Our system is developed using Java EE 7 [40]. To simplify the development process we used IntelliJ IDEA [41] as our IDE. As discussed in Chapter 2, we used Apache Solr 4.10 for the database which is used to store metadata and REST for communication between the client and server. The web portal is implemented using HTML, CSS, JQuery and for the look and feel Bootstrap [42] web development framework.

For version control we used Git. Our project is hosted at Git hub [43]. It is easier to maintain the versions and do collaborative work using Git.

5.2 Metadata Cataloging System

The metadata cataloging system was developed according to the system architecture which was discussed in Chapter 4. In our system implementation we assume that all data products will be pushed to a data archive after they are generated, where the data is organized in a hierarchical folder structure. In our use case the data is located in a directory called *data_root*, where the top-level directories inside it correspond to the users of the system. All the data corresponding to a particular user will be pushed into his/her corresponding subdirectory inside the *data_root* directory. The data products are also organized into directories, where they are given names corresponding to their experiment and an execution date for the ease of identification. There are mainly two types of files inside each data directory. Those are the output file (*.out), which is a text file and the checkpoint file (*.chk), which is a binary file. The checkpoint file gives all the execution states and details of the simulation experiment. This file is important to recreate or re-

investigate an experiment. The output file contains a summary of the execution of the experiment in textual format which is human readable.

5.2.1 Agent

Agent consists of Data Product Generation Monitor, Metadata Extractor and Metadata Extraction Logic Component. It will communicate with Data Archive to detect new data product generation. When the metadata and important attributes are extracted, the Agent will send them to sever for indexing and storing.

The Data Product Generation Monitor component in the Agent sub-system contains a local file system monitor. It scans the file system starting from *data_root* recursively at regular intervals to identify new data products that have been generated. If it identifies a new data product, it sends a notification to the metadata extractor component. This notification contains information regarding the location of the new data products. If the file system already contained data products before the agent was started, data product generation monitor will create notifications for every existing data product. This is essential to make sure all existing data products are indexed. It is also possible to have more than one data archiving location. In such a scenario multiple agent instances have to be run in each data archiving node.

Data product generation monitor can be implemented in many ways depending on the application. Here we implemented it using a local file system scanner for our standalone application. For the system which gets integrated to Airavata we used a message broker. Local file system scanner was implemented using java.nio.file [44] package in JDK 1.7. It gives the capability to watch a directory for changes. The java.nio.file package provides a file change notification API, called the Watch Service API. This API enables you to register a directory (or directories) with the watch service [45]. When registering you have to tell the service that you are interested in tracking file creation. After registering when a service detects a file creation it will be forwarded to the registered process.

When a data product generation is detected by the data product generation monitor notification is sent to the Metadata extractor with the location of the data product. Upon receiving a notification message, the metadata extraction component extracts metadata and important attributes from the

data product and publishes those information to the server via the metadata publisher API. In our use case the output file (*.out) is parsed to extract the metadata and important attributes. These output files are in human readable text format. But the content is complex so that more advanced mechanisms like parsers are needed rather than extracting the important attributes using regular expressions. Moreover, these output files contain information which do not have parsers in the system. Therefore, system should be extendable so that new parsers can be added to the metadata extraction logic component. Since these parsers are domain specific it is hard to generalize the metadata extraction logic.

In our system we offer the ability to parse the data using regular expressions as well. If the scientist does not have a parser and wants to extract some trivial information which can be extracted from regular expressions, he/she can explicitly give the regular expression and the attribute he/she wants to extract. We maintain a regex parser which can be dynamically changed to add regular expressions and attributes so that when the parser is run next time, the respective attributes will be extracted.

Metadata extraction logic component contains domain logic in the form of parsers. In our use case we obtained CUP [24] and JFlex [25] files of the parsers in computational chemistry simulation experiments from the GridChem scientists. They use these parsers to extract information when an output file is generated. From these lexical parsers we generate Java parsers which will suit our purpose.

These parsers extract metadata and other important attributes from output files. Extracted metadata comprises of domain independent metadata such as file name, file path, generated application name, data archive node, created date, and owner name. It also contains domain dependent metadata and attributes such as Universal Chemical Identifier (InChI) string, energy value, and number of iterations for convergence.

For example, let us take a parser which extracts InChI string from an output file. The IUPAC (International Union of Pure and Applied Chemistry) International Chemical Identifier (InChI™) is a non-proprietary identifier for chemical substances that can be used in printed and electronic data sources thus enabling easier linking of diverse data compilation [46]. Typically, chemical compounds are defined by their chemical structures, connection tables or 2D diagrams. InChI will give a unique identifier for each compound in the string format [47]. To run InChI parser you need

Open Babel installed. It is a chemical toolbox designed to speak the many languages of chemical data. It is an open, collaborative project allowing anyone to search, convert, analyze, or store data from molecular modeling, chemistry, solid-state materials, biochemistry, or related areas [48]. This parser with its dependencies is already installed in the system. When the scientist has a parser for a specific attribute which is not already installed in the system, we provide the ability to add it to the list of parsers.

The metadata extraction logic is given as a plugin which can dynamically change by adding parsers or replacing the whole set of parsers with a new set of a completely different domain.

Configuration details for the Agent is given in Appendix I.

5.2.2 Server

Metadata and access control information are maintained in the server. The server provides three service APIs. They are metadata publisher API, query API, and access control list update API. All these APIs are implemented as REST (Representational State Transfer) based web services.

Server uses Apache Solr, an open-source search platform, as its backend data store. It provides mature Java based indexing and search technology [27] . Putting data into Solr is called indexing in Solr. Basically when you add data to an index, they become searchable by Solr. A Solr index can accept data from many different sources. In our system we have a separate Java API to insert data. Regardless of the method used to ingest data, there is a common basic data structure for data being fed into a Solr index: a document containing multiple fields, each with a name and containing content, which may be empty. One of the fields is usually designated as a unique ID field (analogous to a primary key in a database), although the use of a unique ID field is not strictly required by Solr [49].

Solr can have multiple cores in an installation. Solr core is a single index with associated transaction log, configuration files, and schema [50]. Core allows indexing data in different structures in the same server. We have created two cores in Solr. They are metadata core and ACL core.

Metadata and important attributes are stored in the metadata core. Although there is a well specified schema, it allows additional fields which are not specified in the schema at the beginning to add to the database dynamically as a string field. Entire content is indexed using a full text index. N-gram indexing is also performed with remove capitalization filter to get better query support. All the fields except data fields are stored as "String" fields. Date fields are stored in "Date" format.

Every file has its own unique ID which is used to identify the file. In ACL core this document IDs are stored with their respective access control lists.

Indexer component in the server is responsible for indexing the metadata received from the agents. It creates documents based on the received metadata information and indexes them in Solr. The index needs to be recreated with every insertion of a new metadata object. Therefore, to amortize the cost of recreating the index, we opted for batch inserts. Hence, the indexer was designed in such a way so that it accumulates metadata objects up to a configurable threshold or up to a configurable time interval, and then indexes them as a batch.

The query builder component generates queries based on the query parameters received from the query API and executes them against Solr index. The generated queries requires a join between the two cores metadata and ACL to check whether the user issuing the query has permission to access a particular metadata product.

Even though the system maintains only metadata and some important attributes, enforcing a proper access controlling mechanism is very important. This is because the indexed data sets may contain high academic and research value. Therefore, our system explicitly maintains the access control information. It is also possible for the owner of a metadata object to share the information with intended collaborators. ACL updater is used to access and update the ACL core in the database.

The metadata publisher API is used by the agents to publish the generated metadata objects. When metadata and important attributes are extracted by the agent, they are sent to the server through metadata publisher API.

The query API is used by the web portal to make queries on behalf of the users. Users can access the data using various query types such as wildcard queries, suffix queries, prefix queries and full text queries apart from typically used exact match and range queries. These queries are sent to Query Builder to form the actual queries joining with ACL information.

If the user wants to update the access control information on a certain document he/she has to use ACL Updater API from the web portal. Then this will communicate with ACL Updater component to update ACL information in ACL core.

A detailed API specification is given in Appendix II. Configuration details is given in Appendix III.

5.2.3 Web Portal

Figure 5.1 shows a screenshot of the web portal, the primary entry point for users to access the query service provided by the server. To handle user management we have used a third-party user store called WSO2 Identity Server [51]. It is also possible to integrate the web portal with an existing user store, used by the scientific community. WSO2 Identity Server provides sophisticated security and identity management of enterprise web applications, services, and APIs, and makes life easier for developers and architects with its hassle-free, minimal monitoring and maintenance requirements [51].

Web portal is implemented using JQuery to simplify the native JavaScript functionality. It will enable the developer to manipulate the component easily. When exchanging data with the server JQuery AJAX calls are used. AJAX is the art of exchanging data with a server, and updating parts of a web page without reloading the whole page [52]. Bootstrap framework is used along with JQuery to increase the look and feel of the web portal. The Bootstrap responsive grid system is used to maintain the responsiveness of the app.

The main functions of the web portal shown in Fig. 5.1 can be summarized as follows:

1. Users can perform a full text search.
2. Users can also use graphical constructs to generate a more advanced query. These constructs will be joined by AND operators.
3. Users can also select the type of query they want to perform. Currently supported query types include exact match, substring, wildcard, range, and full text search.
4. Users can issue a query based on the data product creation date.

Because of the feature rich REST API, another website, workflow, or scientific gateway can access the same functionalities given by the web portal.

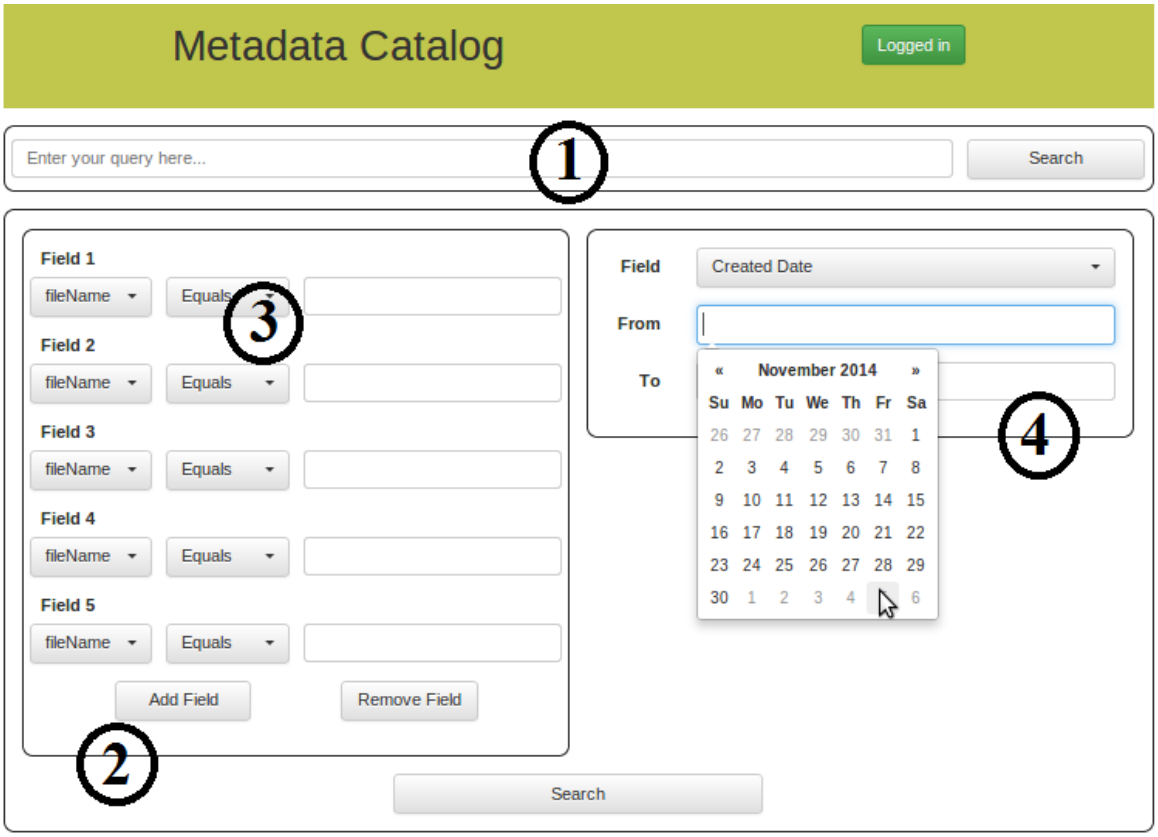

Figure 5.1 Web portal.

5.3 Testing

The system is tested at various stages in the process. In the development stage, to ensure the coding quality and standards, Junit unit tests are used. We have implemented a suite if integration tests to ensure that the functionalities of our project are working properly.

After the system is implemented we carried out a performance test to ensure that the performance of our system is better than the traditional relational database based solutions. To do so, we implemented a similar solution replacing the database. A detailed description of the performance testing will be given in Chapter 7.

5.4 Security Controls

These output data and their metadata contains high research value. The scientific data passed through the system should not be accessed by an unwanted person. Apart from managing the ACL information security steps are taken to limit unwanted access to those data. Therefore, the security plays major role in the system.

The data is only shared among the given scientific community. The owner of the data has the ability to add and remove scientific communities which have access to that particular file. This is handled by the user store.

For the communication via REST APIs, https requests and responses are used with public key cryptography. Each component has a public key and a private key. The public key is published whereas the private key is secret to the component. When a message is sent it is encrypted using the public key. This message cannot be decrypted by anyone who does not have the matching private key.

6 Airavata Integration

According to our second use case we integrated our system into Apache Airavata. The communication among components in Airavata happens mainly through a message broker. Therefore, the file generation monitor will take inputs from the message broker client. All the services subscribe to the required queues and consume the messages corresponding to those queues. The message service which Airavata use is RabbitMQ [53].

6.1.1 RabbitMQ [39]

RabbitMQ is a commercially supported open source message broker software that implements the Advanced Message Queueing Protocol (AMQP). AMQP is a networking protocol that enables conforming client applications to communicate with conforming messaging middleware brokers [54]. Messaging brokers receive messages from *publishers* (applications that publish them, also known as producers) and route them to *consumers* (applications that process them). For the message brokering between Airavata and our system, a separate exchange called the *datacat* exchange is used. It will inform the system that a new output is generated instead of having a local file system monitor.

When an experiment is completed and outputs are generated, Airavata copies those outputs into a data store that is managed by an agent from our system. Airavata then triggers an *ExperimentOutputCreatedEvent* for each and every output file excluding the standard error and standard out files. The *ExperimentOutputCreatedEvent* contains meta-information that is relevant for the processing of the file and to allow proper access control.

The workflow is shown in the Fig. 6.1. In this use case our system is named as *Datacat*.

The *ExperimentOutputCreatedEvent* contains the following fields:

- Experiment ID
- Experiment Name
- Output URI
- Owner Name

- Gateway Name
- Application Name
- Computational Resource

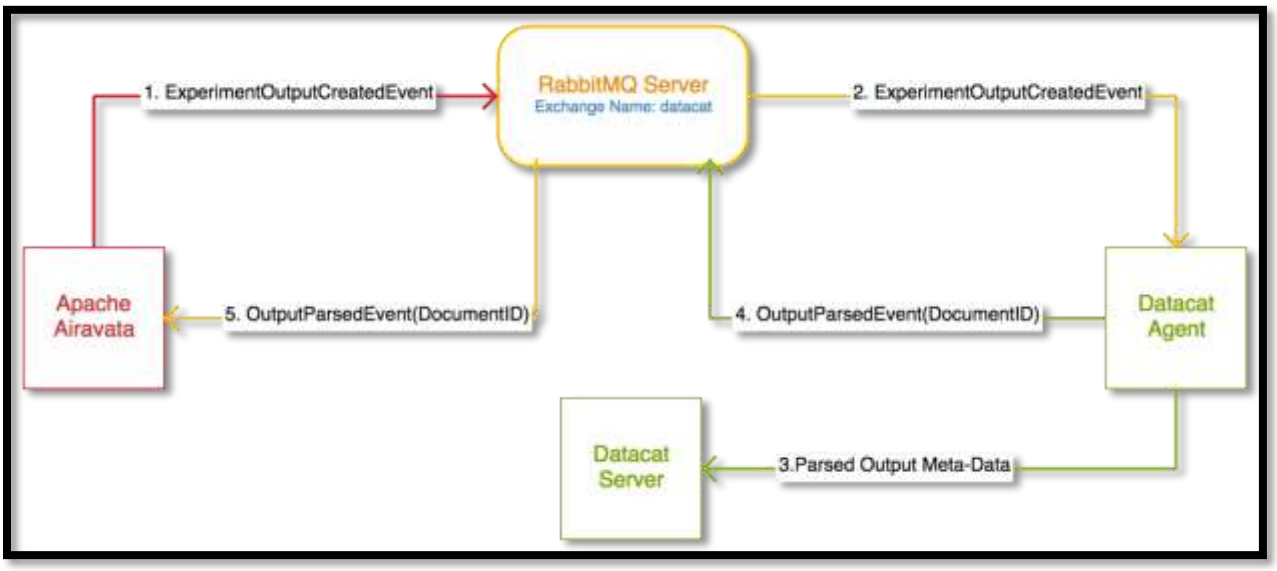

Figure 6.1 Airavata datacat workflow.

This event is processed by the *FileUpdateListener* (a RabbitMQ consumer) in the Datacat Agent which creates the file meta-data object and adds it to the Monitor Dispatcher Queue to be parsed and sent to the Datacat Server.

As soon as the extracted meta-data of the output files are sent to the server, the Agent sends a new message of the type *ExperimentOutputParsedEvent* which will provide Airavata with an ID which is of the indexed output file so that Airavata can call the Datacat Server using that ID to get the extracted meta-data.

The deployment diagram of Datacat system is given in Fig. 6.2.

To demonstrate the functionalities of our system integrated to Airavata we have used SciGap PHP Reference Gateway [55]. PHP Reference Gateway is a user interface developed by Airavata team for demonstration purposes.

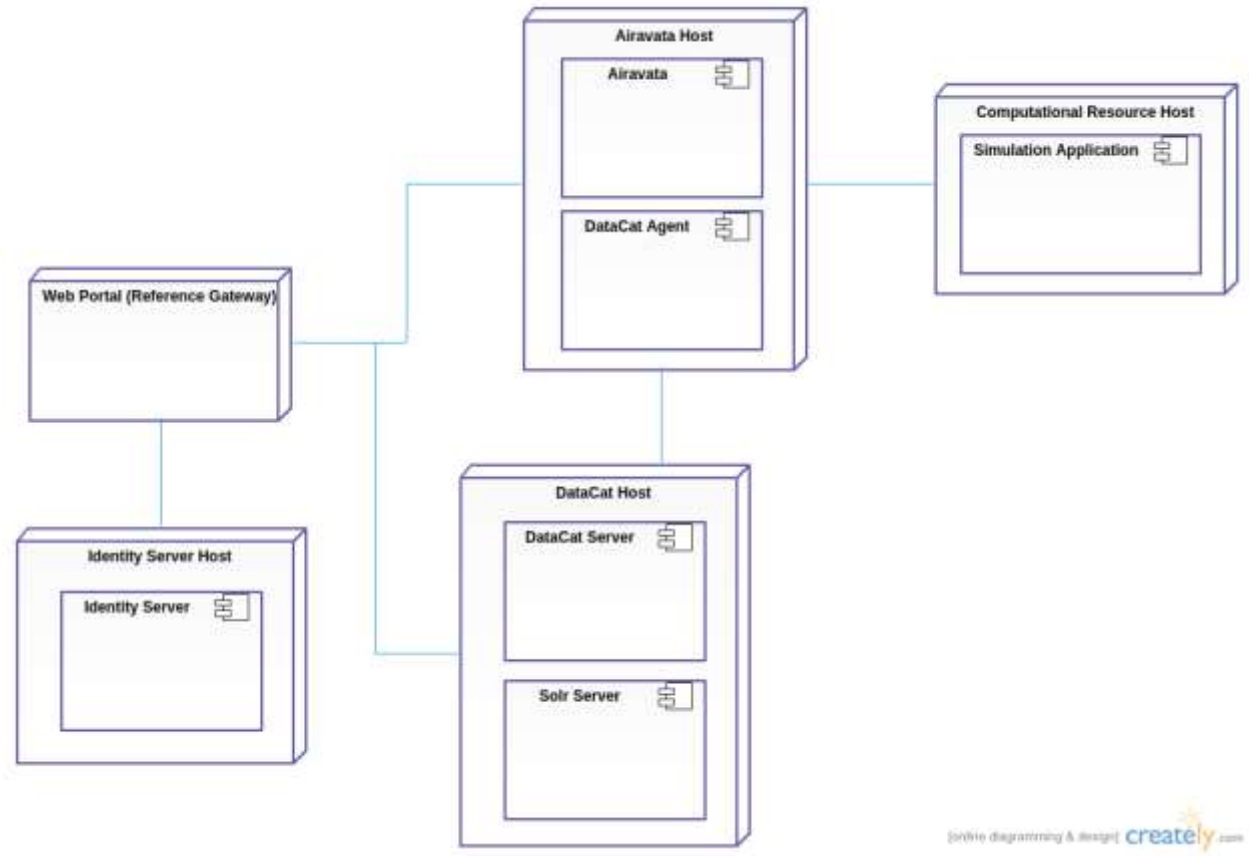

Figure 6.2 Datacat deployment diagram.

Figure 6.3 and 6.4 shows how we integrated our system to the PHP Reference Gateway. Figure 6.3 shows the basic search screen where a user can type what he wants right away. The search terms will be considered as a full text search. If the user has some more information about the search he wants to perform then he can go to the advanced search tab which is shown in Fig. 6.4. In advanced search you can search the metadata fields or/and search type.

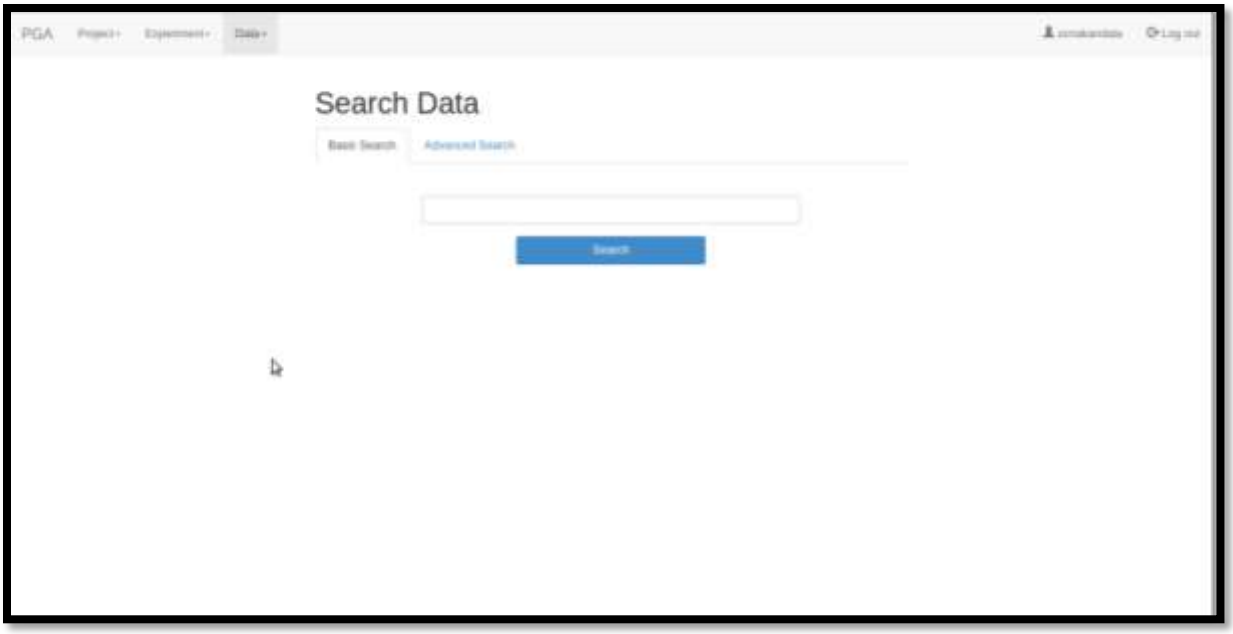

Figure 6.3 Basic search user interface

Figure 6.4 Advanced search user interface

7 Performance Analysis

Even though Solr has been used to index and search large amounts of data, it has rarely been used in a use case as ours to index and search metadata information. Relational Database Management Systems (RDBMS) and Solr have different strengths and weaknesses. RDBMS provide greater support for modelling complex data and maintaining consistency. Similar to Solr, RDBMS can also provide search operations like exact attribute match queries, range queries, and wildcard queries. In Solr all searchable words are stored in an *inverted index*, enabling orders of magnitude faster searching compared to a RDBMS. But this speed comes at the cost of high disk space utilization. Therefore, we developed two versions of our system one using Solr (version 4.10) and the other using MySQL (version 5.6) as the backend data store. We then designed a series of tests to verify which backend data store performs better and under what cases. Both MySQL and Solr had the same schema. In MySQL-based design, all the text fields were indexed using a full text index and other fields were indexed using B-Tree indexes.

There are several widely adopted performance metrics that are used to compare the performance of data stores. Some of them are index size, index creation time, and query time for a mixture of queries. In both MySQL and Solr data stores, the indexes can be created in advance, and hence we will not address the indexing time metric in our analysis. In the next parts of this section data insertion time, index size, and query performance will be considered. For clarity we use *Sys-Solr* to denote the system that uses Solr as the backend data store and *Sys-MySQL* to denote the system that uses MySQL as the backend data store.

The experimental server had the following configuration. Dual core Intel Core i5 480M processor running at 2.67GHz processor. 4GB of RAM. The I/O subsystem was a spin type hard disk drive with an RPM of 5,600. Results are based on ten samples, which were sufficient to attain insertion time and query time within ±5% accuracy and 95% confidence level.

The following schema was used for testing and records were added up to a maximum of 100,000. The content of the records were randomly generated. For both the systems the same test dataset was used. The result of performance analysis is given in Appendix II.

Table 7.1 - Schema for performance test.

Field Name	Sys-MySQL Field Type	Sys-Solr Field Type
id	varchar(255) (B-tree index)	string
fileName	varchar(255) (B-tree index)	string
filePath	varchar(255) (B-tree index)	string
generatedApplicationName	varchar(255) (B-tree index)	string
dataArchiveNode	varchar(255) (B-tree index)	string
ownerName	varchar(255) (B-tree index)	string
inChi	varchar(255) (B-tree index)	string
inChiKey	varchar(255) (B-tree index)	string
text	text (full-text index)	text

7.1 Data Insert Performance

It is very important to test the data insert performance of the metadata catalog as scientific applications and experiments can generate large volumes of data especially in situations where instrumental observations are recorded. The test was performed on both systems Sys-Solr and Sys-MySQL, assuming that the indexes have already been created. Data were inserted in batches of 1,000 records up to a total of 100,000 records. The results are shown in Fig. 7.1. It can be seen that the metadata insertion time of Sys-Solr grows much more slowly compared to the Sys-MySQL implementation.

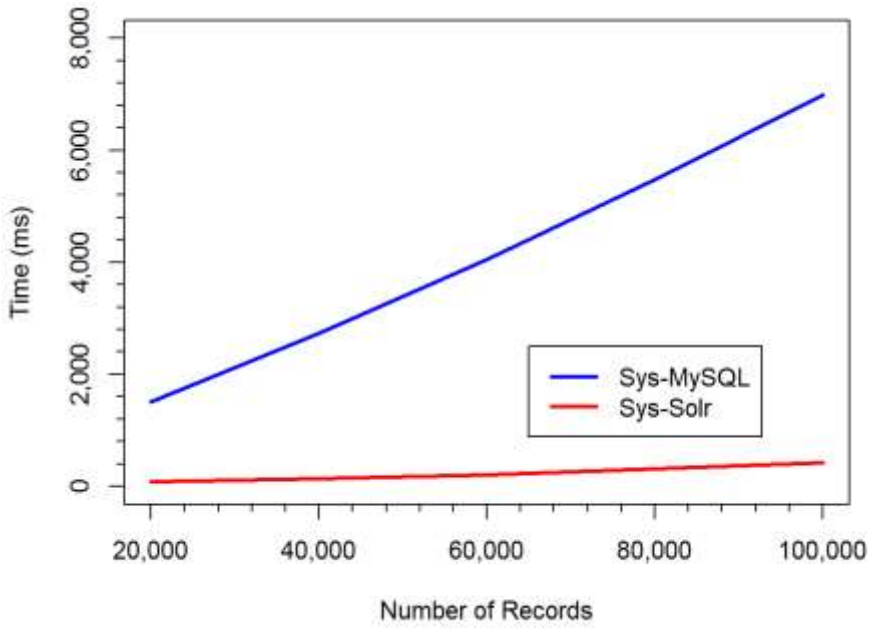

Figure 7.1 Data Insertion Time.

7.2 Query Performance

For this performance test we analyzed seven types of queries which are listed below:

- Exact match queries
- Range queries
- Full text queries
- Prefix match queries
- Suffix match queries
- Wildcard queries
- Substring queries

To maintain the timing accuracy, tests were carried out as 100 test cases per query type, and average query response time was considered based on time per each batch. Same test queries were used in both the systems.

7.2.1 Exact match queries

In exact match queries the server will return only the results which exactly match with the query parameters. Figure 7.2 shows the graph between the time taken for the execution for exact match query in Sys-Solr and Sys-MySQL. According to this we can see that Sys-MySQL has outperformed Sys-Solr. But the gradient of Sys-MySQL performance is relatively higher than Sys-Solr. Therefore, as the number of records increases Sys-Solr performance will be comparable to Sys-MySQL.

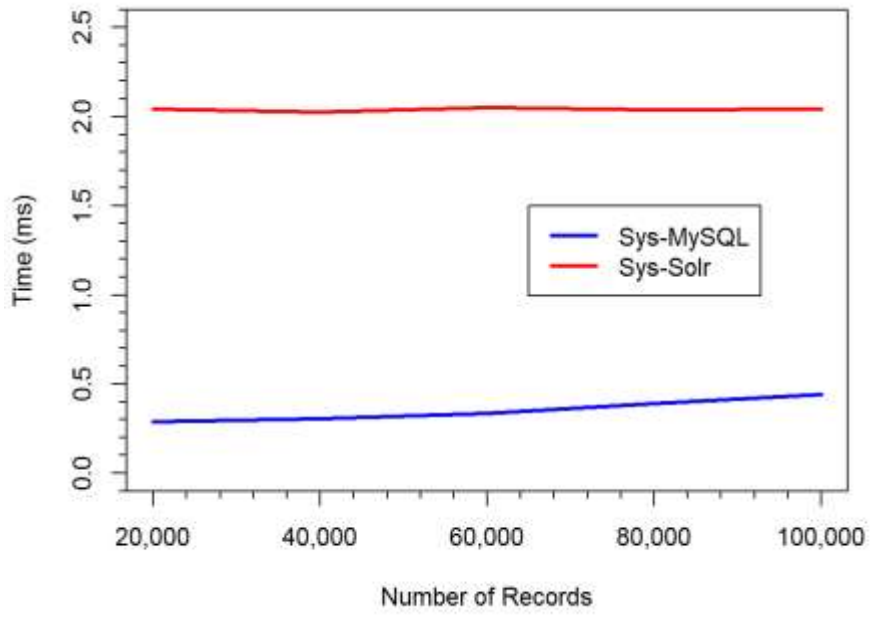

Figure 7.2 Query execution time for exact match queries.

7.2.2 Range Queries

Range queries are queries which return the values which are inside a value range defined by the query. Figure 7.3 represents the response time for range queries for Sys-Solr and Sys-MySQL. Similar to exact match queries, Sys-MySQL has better performance in range queries.

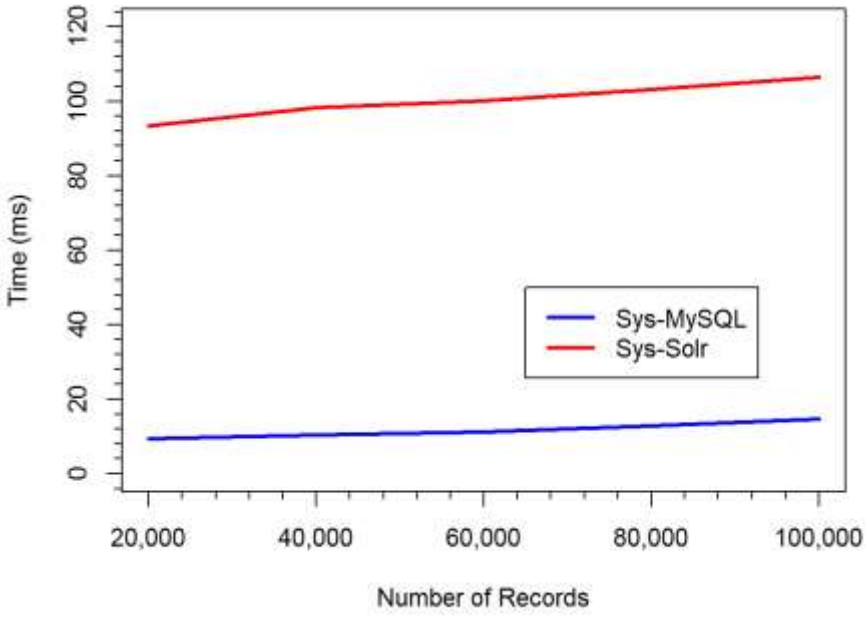

Figure 7.3 Query execution time for range queries.

7.2.3 Full text queries

In full text search, the search engine tries to match all of the words in every stored document with the search criteria given by the user. In MySQL this is handled using ranking with vector spaces [56]. Rank is a relevance measure which shows how good a match is. Solr is powered by Lucene [28]. Lucene is a powerful full text search library implemented in Java.

Figure7.4 shows the response time for query execution in full text queries. For a database consist of 100,000 records, Sys-Solr gives 97.45% better performance than Sys-MySQL in full text queries.

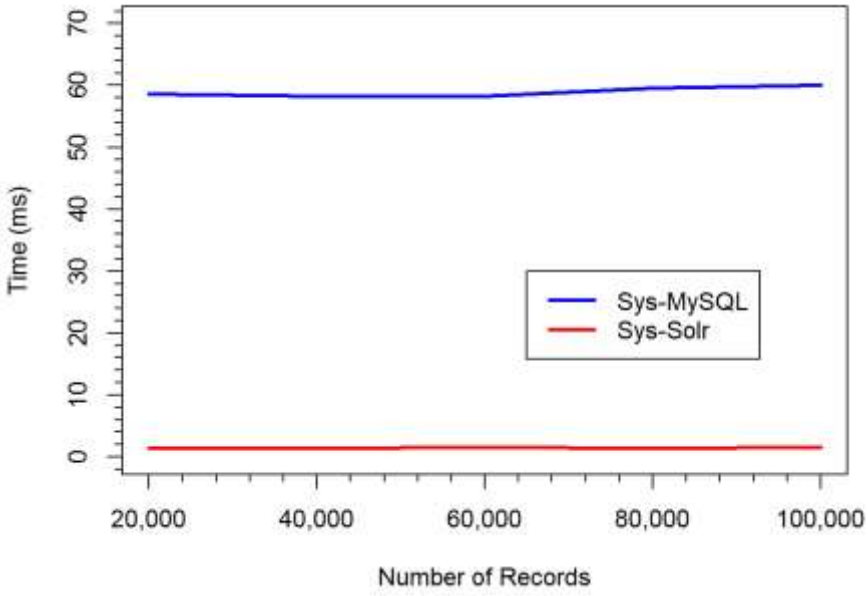

Figure 7.4 Query execution time for full text search queries.

7.2.4 Prefix match queries

Prefix match query returns results which starts with the value given by the user. Figure 7.5 shows the response time for query execution in prefix match queries. For a database consisting of 100,000 records, Sys-Solr gives 99.2% better performance than Sys-MySQL prefix match queries.

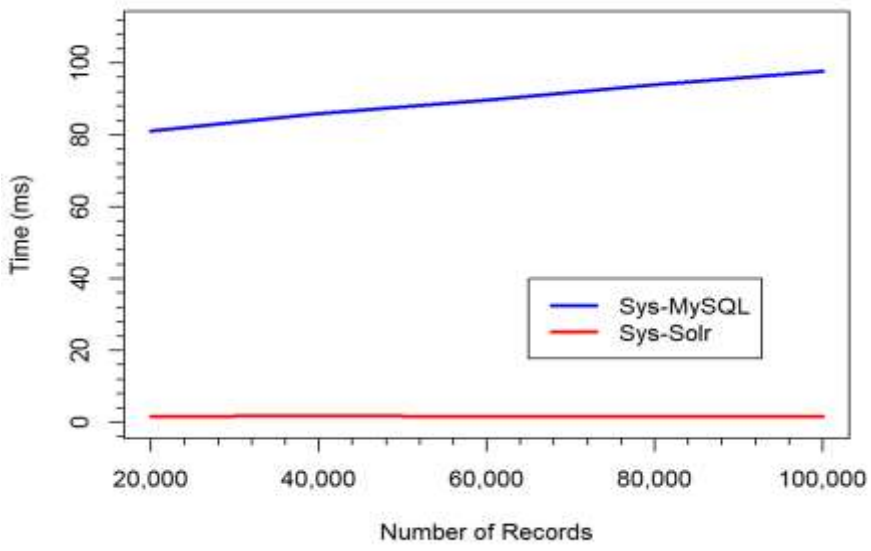

Figure 7.5 Query execution time for prefix match queries.

7.2.5 Suffix match queries

Suffix match query returns results which ends with the value given by the user. Figure 7.6 shows the response time for query execution in suffix match queries. For a database consisting of 100,000 records, Sys-Solr gives 97.8% better performance than Sys-MySQL in suffix match queries.

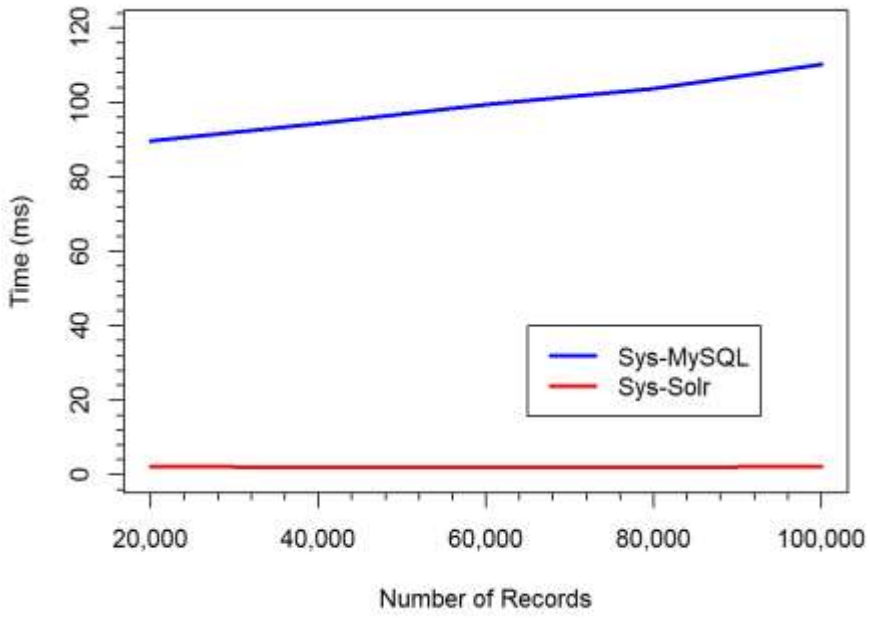

Figure 7.6 Query execution time for suffix match queries.

7.2.6 Wild card queries

In wild card queries the server returns the terms which have the given value anyplace in the term. Figure 7.7 shows the response time for query execution in wild card queries. For a database consisting of 100,000 records, Sys-Solr gives a 99.16% better performance than Sys-MySQL in wild card queries.

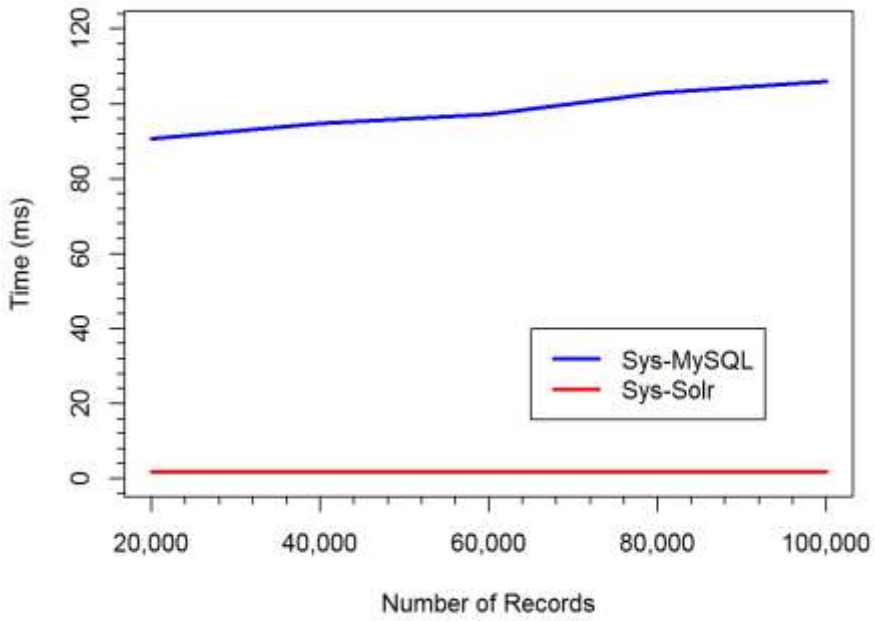

Figure 7.7 Query execution time for wild card queries.

7.2.7 Substring Queries

A substring query returns only a part of a string which is defined by the user indicating the start and end indexes. Figure 7.8 shows the response time for query execution in substring queries. For a database consisting of 100,000 records, Sys-Solr gives 97.72% better performance than Sys-MySQL in substring queries.

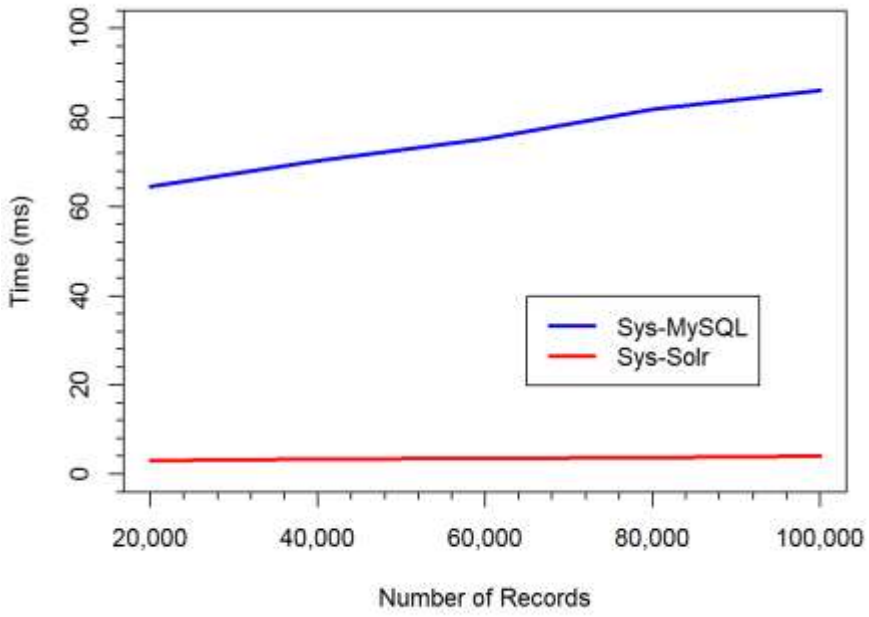

Figure 7.8 Query execution time for substring queries.

7.3 Space Utilization

Even though disk space is getting cheaper it is good to have an understanding of the disk space utilization of the two alternatives. In this test, we analyzed the storage utilization of each system. Fig. 25 shows the disk utilization at different numbers of records. Sys-Solr consumes more disk space and has a relatively higher gradient than Sys-MySQL.

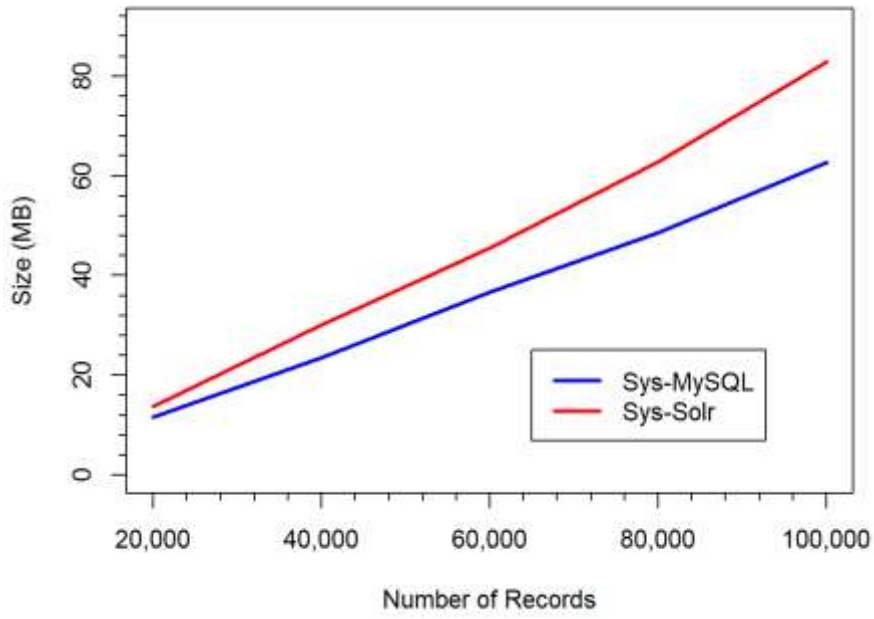

Figure 7.9 Storage utilization.

7.4 Conclusion

In conclusion, Solr-based implementation provides a much better query performance at the expense of relatively high storage utilization. It resolves more complex queries 91% - 99% faster than a MySQL-based implementation. As our system indexes only the metadata (not the actual data generated by the scientific experiment or simulation), we believe that this increase in storage utilization can be justified given the lower cost of storage devices and end user convenience it gives in terms of better querying, sharing, and reusing of scientific datasets.

8 Summary

Scientists have long running experimental workflows which use computational resources like super computers, clusters, and workstations. When they run an experimental workflow and get the output file, they parse it to get the data they want and dump the file into an archive. If they want to run exactly the same experiment with the same inputs, they run it again. This wastes valuable resources and time. If there is a mechanism to search the data archives effectively, then the scientific community can reuse the previous run experimental outputs rather than running the experiment again. We address this problem by developing a scientific data catalog which stores the important information in an experiment output in the form of a metadata catalog.

Our solution consists of an agent which detects the output data as they get generated and parse them to extract metadata, a server which stores the metadata and handles queries, and a web portal through which scientists can access the data shared with them.

While there are many similar solutions which uses metadata catalogs to enable efficient searching of scientific data, they are tightly coupled to their problem domain. The uniqueness of our system lies in the pluggable metadata extraction logic, which enables our system to be used across many scientific domains for data management.

All the previously implemented metadata catalogs use relational database management systems as their backend database. For our system we used Solr as our backend because it supports flexible querying and has a better performance compared to a relational database management system. We carried out a performance analysis replacing our backend with similar MySQL based implementation and measured query execution times for various queries, data insertion time and space utilization to compare with Solr based implementation. We found out that even though MySQL-based implementation has better performance in exact match and range queries, Solr-based implementation outperforms MySQL based implementation in full text search, prefix match search, suffix match search, and wild card search. Solr based implementation resolves these complex queries 91% - 99% faster than a MySQL-based implementation.

8.1 Problems and Challenges

When designing and implementing the system the main challenge we faced was that we were lacking domain knowledge to implement the parsers. GridChem use case deals with "Gaussian 9" computational chemistry experiments. The parsers are generated from CUP and JFLEX lexical parsers. To integrate these parsers to our system we had to have an understanding on the metadata attributes and how they are used. For example, attributes like InChI string of a chemical compound is new to all the team members because we do not have a chemistry background. Therefore, before understanding the parsers we had to understand the attributes.

Another challenge that we encountered is using Solr database. We were accustomed to use relational database management systems like MySQL. Since we have not used Solr before, we had to learn it from scratch and research about it to see whether it will be suitable for our system and feasible to implement in our metadata store.

It was difficult to obtain large datasets through the campus network, and we also had to face with several issues while remotely accessing the resources at Indianan University. We solved the difficulty in obtaining the large dataset issue by taking only a sample set of output files without taking the large dataset. Problems with accessing resources at Indiana University is resolved using constant communication through mailing lists.

8.2 Future Work

One possible extension points on this project is to let the system auto generate parsers which are defined by the user. We try to address this problem within our scope and found out because the output data is in a complex format, it will need a significant time to figure out how it is done and how do it. Although we were not successful in auto generating parsers given the attribute, we added the functionality to extract metadata when the regular expression of an attribute which can be extracted directly is given.

Another interesting area to extend the project is allowing provenance-aware workflow execution. When executing a workflow it can have several intermediate states which generate intermediate outputs. These intermediate states can be reused by some other experiments which are similar and have the same states up to a certain point. In that case the solution we implemented now will be of

no use because it only tracks the output and its important attribute. If we take this intermediate provenance information and enable the searching capability to search from them, it will make the scientist's life easier. Instead of running the whole experiment in the computational resource he/she can run a part of it if the provenance information of the other part can be taken from another experiment.

9 References

[1] J. Gary, D. T. Liu, M. Nieto-Santisteban, A. Szaley, D. Dewitt, and G. Heber, "Scientific Data Management in the Coming Decade," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 34-41, Dec. 2005.

[2] M. Valle, "Scientific Data Management," [Online]. Available: http://mariovalle.name/sdm/scientific-data-management.html.

[3] "Gaussian," High Performance Computing Virtual Laboratory, [Online]. Available: http://www.hpcvl.org/faqs/application-software/gaussian.

[4] O. A. Adeleke, *A Metadata Service for an Infrastructure of Large Scale Distributed Scientific Datasets*, M.S. Thesis,Fac. of Science, Univ. Witwatersrand, Johannesberg, 2014.

[5] "Apache Solr," Lucene, [Online]. Available: http://lucene.apache.org/solr/.

[6] "MCAT," [Online]. Available: http://www.sdsc.edu/srb/index.php/MCAT.

[7] G. Singh et al., "A Metadata Catalog Service for Data Intensive Applications," in *SC'03*, Phoenix, Arizona, USA, 2003.

[8] B. Plale, D. Gannon, J. Alameda, B. Wilhelmson and A. R. K. D. S. Hampton, "Active Management of Scientific Data," *IEEE Internet Computing*, pp. 27-34, Jan. 2005.

[9] "Airavata," Apache Airavata, [Online]. Available: https://airavata.apache.org/architecture/overview.html.

[10] "What is a Science Gateway," [Online]. Available: http://sciencegateways.org/what-is-a-science-gateway..

[11] "Geospatial Metadata," [Online]. Available: http://www.fgdc.gov/metadata/index_html.

[12] "Maryland State Geographic Information Committee," [Online]. Available: http://www.msgic.state.md.us.

[13] M. Singh and A. Vouk, "Scientific computing meets transactional workflows," in *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, Athens, 1996.

[14] G. Yolanda et al., "Examining the Challenges of Scientific Workflows," *IEEE Computer*, vol. 40, no. 12, pp. 24-32, 2007.

[15] "Common Motifs in Scientific Workflows," [Online]. Available: http://mayor2.dia.fi.upm.es/oeg-upm/files/dgarijo/motifAnalysisSite/.

[16] M. Weske, *Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects,* Habilitation Thesis, University of M¨unste, 2000.

[17] "Apache Airavata's Architecture," Apache Airavata, [Online]. Available: https://airavata.apache.org/architecture/user4.png.

[18] "The CIPRES Science Gateway V. 3.3," [Online]. Available: http://www.phylo.org./.

[19] "Welcome to the XSEDE Science Gateway for UltraScan!," UltraScan Project, [Online]. Available: http://uslims.uthscsa.edu/.

[20] "Welcome to the Community Climate System Modeling Portal!," Purdue University, [Online]. Available: https://gridsphere.rcac.purdue.edu:8453/gridsphere/gridsphere.

[21] S. Marru et al., "Apache Airavata: A framework for Distributed Applications," in *Proceedings of the 2011 ACM workshop on Gateway computing environments,* New York, 2011.

[22] "Welcome to Apache Axis2/Java," Apache Software Foundation, [Online]. Available: http://axis.apache.org/axis2/java/core/.

[23] "Airavata Stakeholders," Apache Airavata, [Online]. Available: https://airavata.apache.org/architecture/airavata-stakeholders.html.

[24] F. Flannery, C. Ananian, D. Wang and M.Petter, "CUP User's Manual," [Online]. Available: http://www2.cs.tum.edu/projects/cup/docs.php.

[25] "JFlex - The fastest scanner generator for Java," [Online]. Available: http://www.jflex.de/.

[26] "iRods Overview," [Online]. Available: http://irods.org/about/overview.

[27] Y. Ging, C. Zhang and X. Wang, "An Empirical Study on Performance Comparison of Lucene and Relational Database," in *International conference of Communication Software and Networks,* Macau, 2009.

[28] "Apache Lucene," Lucene, [Online]. Available: http://lucene.apache.org/.

[29] "Solr features," Lucene, [Online]. Available: http://lucene.apache.org/solr/features.html.

[30] A. Lakshman and P. Malik, "Cassandra- a decentralized structured storage system," *ACM SIGOP Operating System Review Newsletter,* pp. 35-40, April 2010.

[31] "Welcome to Apache Cassandra," Apache Software Foundation, [Online]. Available: http://cassandra.apache.org/.

[32] 'Limitations," [Online]. Available: http://wiki.apache.org/cassandra/CassandraLimitations.

[33] "Why Use Solr," [Online]. Available: http://wiki.apache.org/solr/WhyUseSolr.

[34] "Cassendra Vs. Solr," [Online]. Available: http://db-engines.com/en/system/Cassandra%3BSolr.

[35] P. Potti, *On the Design of Web Services: SOAP vs. REST,* M.S. thesis, School of Computing, Univ. Florida, 2011.

[36] "RESTful Web Services: The Basics," [Online]. Available: https://www.ibm.com/developerworks/webservices/library/ws-restful.

[37] A. Rodriguez, "RESTful Web services: The basics," [Online]. Available: http://www.ibm.com/developerworks/library/ws-restful/.

[38] "Compare RESTful vs. SOAP Web Services," [Online]. Available: http://java.dzone.com/articles/j2ee-compare-restful-vs-soap.

[39] "GridChem," [Online]. Available: http://www.sciencegatewaysecurity.org/case-study/gridchem.

[40] "Java EE at a glance," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[41] "IntelliJIDEA," JetBrains, [Online]. Available: https://www.jetbrains.com/idea/.

[42] "Bootstrap," [Online]. Available: http://getbootstrap.com/.

[43] "GitHub," [Online]. Available: https://github.com/.

[44] "File I/O (Featuring NIO.2)," Oracle, [Online]. Available: http://docs.oracle.com/javase/tutorial/essential/io/fileio.html.

[45] "Watching a directory for changes," Oracle, [Online]. Available: http://docs.oracle.com/javase/tutorial/essential/io/notification.html.

[46] "The IUPAC International Chemical Identifier (InChI)," International Union of Pure and Applied Chemistry, [Online]. Available: http://www.iupac.org/home/publications/e-resources/inchi.html.

[47] S. Stein, S. R. Heller and D. Tchekhovski, "An Open Standard for Chemical Structure Representation - The IUPAC Chemical Identifier".

[48] "Open Babel: The Open Source Chemistry Toolbox," [Online]. Available: http://openbabel.org/wiki/Main_Page.

[49] “Introduction to Solr Indexing,” Apache Software Foundation, [Online]. Available: https://cwiki.apache.org/confluence/display/solr/Introduction+to+Solr+Indexing.

[50] “Solr Cores and solr.xml,” Apache Software Foundation, [Online]. Available: https://cwiki.apache.org/confluence/display/solr/Solr+Cores+and+solr.xml.

[51] “WSO2 Identity Server,” WSO2 Inc., [Online]. Available: http://wso2.com/products/identity-server/.

[52] “JQuery AJAX Methods,” [Online]. Available: http://www.w3schools.com/jquery/jquery_ref_ajax.asp.

[53] “RabbitMQ,” RabbitMQ, [Online]. Available: http://www.rabbitmq.com/.

[54] “AMPQ model explained,” RabbitMQ, [Online]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts.html.

[55] “PHP Reference Gateway,” Apache Software Foundation, [Online]. Available: https://cwiki.apache.org/confluence/display/AIRAVATA/PHP+Reference+Gateway+User+Guide.

[56] “Full Text Search,” MySQL, [Online]. Available: http://dev.mysql.com/doc/internals/en/full-text-search.html.

[57] S.Marru et al., “Apache Airavata: A framework for Distributed Applications,” in *GCE'11*, Seattle, Washinton, USA, 2011.

Appendix I – Server Configuration Details

```
# The end point of the datacat service
DATACAT_URI=https://localhost:8887/

# The end point of the publisher URL
PUBLISHER_URI=https://localhost:8888/

# The end point of the (optional) userstore URI
USERSTORE_URI=https://localhost:8889/

# Solr metadata core access URL
SOLR_METADATA_URL=http://localhost:8983/solr/metadata

# Solr ACL core access point URL
SOLR_ACL_URL=http://localhost:8983/solr/acl

# Username of the servlet container containing Solr
SOLR USERNAME=datacat

# Password of the servlet container containing Solr
SOLR_PASSWORD=datacat

# WSO2 Identity server URL
IS_URL=https://localhost:9443

# Identity server administrative username
IS_USERNAME=admin

# Identity server administrative password
IS_PASSWORD=admin

#When empty server adds a randomly generated GUID
METADATA_PRIMARY_INDEX=


# keystore and truststore specific communication
KEYSTORE_FILE=keystore.jks
KEYSTORE_PWD=keystore_password
TRUSTSTORE_FILE=client-truststore.jks
TRUSTSTORE_PWD=truststore_password
```

Appendix II – API Specification for DataCat Server

DataCat Service

This API is used by the web portal to delegate user tasks such as searching over data and updating the access control information.

1. Get API Version

Method to get the API Version of the DataCat Service.

Request

Method	GET
URL	http://<host>/datacat/getAPIVersion

Response

Status	Response
200 (ok)	<api_version>

2. Get Metadata by id

Method to get the metadata information given the metadata document id. This method returns a JSON representation of the metadata information (Map of key value pairs).

Request

Method	GET
URL	http://<host>/datacat/getMetadataById?id=<id>

Response

Status	Response (example)
200 (ok)	{ “id” = “dkss923-n232nD-23sd2d”, “inChi” = “InChi=C2H20”, .

	. . “key_n” = “val_n” }

3. Get Results

Method to get matching metadata documents given a query object. This method returns a JSON representation of the metadata information (Map of key value pairs).

Request

Method	GET
URL	http://<host>/datacat/getResults
Body (example)	{ "username": "sudhakar", "userGroups": ["gridchem"], "queryStringSet": false, "queryString": "*:*", "startRow": 0, "numberOfRows": 25, "primaryQueryParameterList": [{ "primaryQueryType": "EQUALS", "firstParameter": "InChI=1S/C5H9O4", "secondParameter": null, "field": "InChi_s" }, { "primaryQueryType": "SUBSTRING", "firstParameter": "C5H9O4", "secondParameter": null, "field": "InChi_s" }, { "primaryQueryType": "RANGE", "firstParameter": "*", "secondParameter": "2015-09-08",

	"field": "createdDate"
	},
	{
	"primaryQueryType": "PHRASE",
	"firstParameter": "InChI=1S/C5H9O4",
	"secondParameter": null,
	"field": "InChi_s"
	}
	]
	}

Response

Status	Response (example)

200 (ok)	{
	[
	“id” : “dkss923-n232nD-23sd2d”,
	“inChi” : “InChi=C2H20”,
	.
	.
	.
	“key_n” : “val_n”
	],
	[
	“id” : “dkss923-n232nD-23sd2d”,
	“inChi” : “InChi=C2H20”,
	.
	.
	.
	“key_n” : “val_n”
	]
	}

4. Get Access Control List

Method to get the Access Control List information given the metadata document id. This method returns a JSON representation of an array of group names of the allowed groups.

Request

Method	GET
URL	http://<host>/datacat/getAclList?id=<id>

Response

Status	Response (example)

200 (ok)	{
	[
	“gridchem”,
	“parachem”
	“public”
	]
	}

5. Update Access Control List

Method to update the Access Control List information given the metadata document id and the new Access Control List.

Request

Method	POST
URL	http://<host>/datacat/updateAclList

Body	{
	id : “8dhgrx-anaqm2-sd2m22”,
	acl :
	[
	“gridchem”,
	“parachem”
	“public”
	]
	}

Response

Status	Response (example)

200 (ok)	success

Publisher Service

Publisher Service is used by agents to publish the generated metadata to the central server.

1. **Add Metadata Document**

Method to insert the generated metadata to the central server. Metadata is inserted one document at a time.

Request

Method	POST
URL	http://<host>/publisher/addFileMetadata

Body	{
	"id" = "dkss923-n232nD-23sd2d",
	"inChi" = "InChi=C2H20",
	.
	.
	.
	"key_n" = "val_n"
	}

Response

Status	Response (example)
200 (ok)	success

2. **Add Batch Metadata**

Method to insert multiple generated metadata files to the central server.

Request

Method	POST

URL	http://<host>/publisher/addFileMetadata

Body	{
	[
	“id” : “dkss923-n232nD-23sd2d”,
	“inChi” : “InChi=C2H20”,
	.
	.
	.
	“key_n” : “val_n”
	],
	[
	“id” : “dkss923-n232nD-23sd2d”,
	“inChi” : “InChi=C2H20”,
	.
	.
	.
	“key_n” : “val_n”
	]
	}

Response

Status	Response (example)
200 (ok)	success

User store Service

This is an optional service which can be used by the web portals if they don’t have their user management system. This service is backed by WSO2 Identity server and most of user management functionalities can be done from the WSO2 IS. Only the functionalities that are relevant for the DataCat application is exposed in this service.

1. **Authenticate user**

Method to authenticate a user giving username and password

Request

Method	POST
URL	http://<host>/userstore/authenticate

Body	{
	“username” : ”abc”,
	“password” : “pwd”
	}

Response

Status	Response (example)
200 (ok)	true : login success false : login fails

2. Get All Groups

This method returns all the available groups in the system.

Request

Method	GET
URL	http://<host>/userstore/getAllGroups

Response

Status	Response (example)

200 (ok)	{
	[
	“gridchem”,
	“parachem”
	]
	}

3. Get Group List of User

This method returns all the groups of the given user

Request

Method	POST
URL	https://<host>/userstore/getGroupsOfUser?username=<username>

Response

Status	Response (example)

200 (ok)	{
	[
	“group1”,
	“group2”,
	“group3”
	]
	}

Appendix III – Agent Configuration Details

```
# number of maximum parser threads
MAX_PARSER_THREADS=20

# batch based waiting time for monitor scan
BATCH_MONITOR_WAIT_TIME=2000

# time delay for file to get finished updating
FILE UPDATE MESSAGE DELAY=5

# end point of the datacat publisher post
PUBLISHER_ADD_ENDPOINT=https://localhost:8888/publisher/addFileMetadata/

#Monitor type is FILE_SYSTEM or RABBITMQ
MONITOR_TYPE=RABBITMQ
# data archive root directory location
DATA_ROOT=/home/supun/datacat/data_root

# RabbitMQ  related configuration
RABBITMQ_HOST=localhost
EXCHANGE_NAME=datacat
PARSER_CLASS=org.apache.airavata.datacat.parsers.gridchem.GridChemDemoParser

# keystore and truststore related configuration
KEYSTORE_FILE=keystore.jks
KEYSTORE_PWD=keystore_password
TRUSTSTORE_FILE=client-truststore.jks
TRUSTSTORE_PWD=truststore_password
```

Appendix IV – Performance Test Results

Table1 Data insertion time

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	1509.185	78.781
40,000	2737.777	139.204
60,000	4047.541	209.743
80,000	5471.707	315.017
100,000	6991.735	421.454

Table 2 Query execution time for exact match queries

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	0.289	2.043
40,000	0.306	2.028
60,000	0.337	2.053
80,000	0.392	2.037
100,000	0.441	2.045

Table 3 Query execution time for range queries

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	9.426	93.298
40,000	10.365	98.237
60,000	11.225	100.095
80,000	12.8	103.116
100,000	14.606	106.485

Table 3 Average query execution time for prefix match queries.

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	81.052	1.733
40,000	85.975	1.792
60,000	89.796	1.747
80,000	94.082	1.763
100,000	97.684	1.74

Table 4 Average query execution time for suffix match queries.

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	89.629	2.145
40,000	94.483	2.117
60,000	99.501	2.118
80,000	103.797	2.133
100,000	110.321	2.151

Table 5 Average query execution time for full text queries.

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	58.661	1.457
40,000	58.219	1.459
60,000	58.219	1.494
80,000	59.564	1.483
100,000	60.044	1.529

Table 6 Average query execution time for substring queries

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	64.576	3.064
40,000	70.265	3.315
60,000	75.227	3.642
80,000	81.865	3.802
100,000	86.216	4.136

Avarage query execution time for wildcard queries

Number of Records	Sys-MySQL Response Time (ms)	Sys-Solr Response Time (ms)
20,000	90.783	1.772
40,000	94.745	1.775
60,000	97.231	1.779
80,000	103.035	1.774
100,000	106.116	1.772