Schema-Independent Scientific Data Cataloging Framework

Supun Nakandala,
Sachith Dhanushka Withana,
Dinu Kumarasiri,
Hirantha Jayawardena
and H.M.N. Dilum Bandara
Dept. of Computer Science and Engineering
University of Moratuwa
Katubedda, Sri Lanka

Srinath Perera
WSO2 Inc.
Colombo, Sri Lanka

Suresh Marru
and Sudhakar Pamidighantam
Indiana University
Bloomington, USA

***Abstract*—Modern scientific experiments generate vast volumes of data which are hard to keep track of. Consequently, scientists find it difficult to 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 extract metadata from them, as well as a server which indexes the metadata using a NoSQL database and provides a REST API for querying, sharing, and reusing the data sets. 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. The use of Apache Solr as the backend database enables the proposed solution to index and search data products much fatser than a solution 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.**

***Index Terms*—indexing; metadata catalog; scientific data management**

I. Introduction

Various scientific experiments and simulations create vast volumes of data every minute. It is said that the 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. However, when the data volume increases, scientific community faces several challenges such as the following [2]:

- Limited file and directory naming schemes.
- Scientists have to 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.
- Unowned data after a large project.

978-1-4799-1740-2/15/$31.00 ©2015 IEEE

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 do an experiment, he/she can first go and 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.

MCAT [6], MCS [7], and MyLEAD [8] are widely used scientific data management systems based on a metadata catalog. Table I presents a comparative summary of these three metadata cataloging systems. It can be seen that 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, full-text, 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.

We consider a use case based on "Gaussian 9" [3], which is a computational chemistry simulator. 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 needed from the experiment and then archive it.

TABLE I: Comparison of existing metadata catalogs.

	MCAT	MCS	MyLEAD
Ability to use an independent component	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.
Database technology	Combination of IBM DB2 and Oracle database.	MySQL database.	MySQL database.
Access control	Yes	Yes	Yes
Supporting logical collections	Yes	Yes	No explicit notion of logical collections.
Search operations	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.	Explicitly stores and manages provenance information.
Data mining	No notion of data mining on data products.	No notion of data mining on data products.	Supports data mining tasks on the collected data products.
Communication protocols / architectures	Service oriented architecture.	Service oriented architecture.	Client server architecture.

In this paper, we propose a schema-independent scientific data cataloging framework 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 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 framework detects it and sends the file(s) to the parser system. The parser system, then extracts the metadata. This metadata is then indexed in a NoSQL database based on Apache Solr [5]. Moreover, the framework also provides a REST API for querying, sharing, and reusing data sets through a website, workflow, or science gateway while enforcing necessary access control rules.

Compared to application-specific systems such a MCAT, MCS, and MyLEAD, our solution is generalizable and can be used in other domains by replacing the metadata extraction logic. Moreover, its NoSQL implementation based on Solr enables fast resolution of natural language queries issued by scientists. Solr also enables the dynamic addition of metadata fields. Hence, our solution is schema independent comapred to prior solutions. Furthermore, the given REST API enables querying, sharing, and reusing data sets through many systems such as websites, workflows, or science gateways.

The response time of our NoSQL-based implementation is much lower 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.

This paper is organized as follows. The architecture of the proposed solution is described in Section II. Section III presents the performance analysis. Concluding remarks are given in Section IV.

II. System Architecture

The architecture of the framework consists of an agent, server, and the web portal (see Fig. 1). In our use case, we assume that all the data products will be pushed to a data archive after their generation, where the data is organised in a hierarchical folder structure. The agent continuously monitors the folder structure 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 mine the metadata and important attributes, and publish those to the server. The 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. Next, we discuss the details of major components of the framework.

A. Agent

In our use case the data was 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 organised into directories, where they are given names corresponding to their experiment and 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.

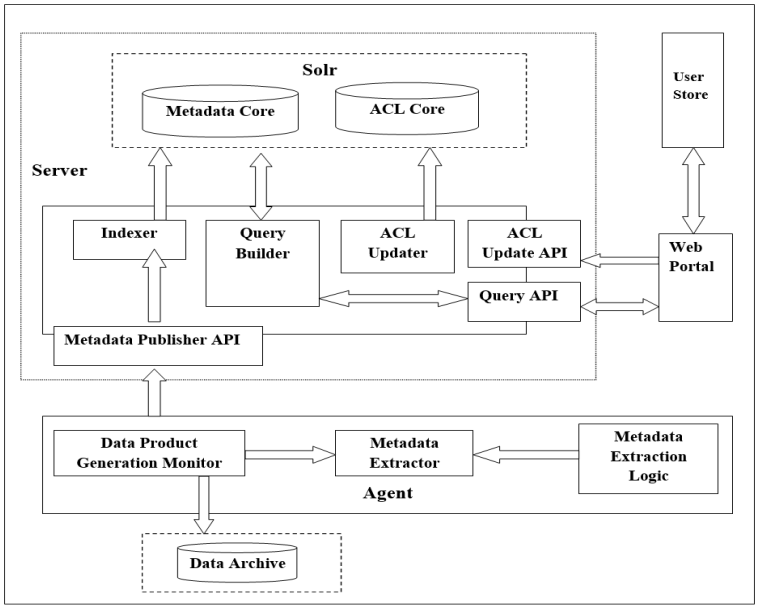

Fig. 1: System architecture.

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 has 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, 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.

Upon receiving a notification message, the *metadata extractor* component extracts metadata and important attributes of the data product. It then publish this 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. *Metadata extraction logic* component contains domain logic in the form of lexical parsers such as Cup [9] and JFlex [10], which is used to extract metadata and important attributes. Extracted metadata comprises of both domain dependent and independent metadata. Domain independent metadata includes 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. It is possible that some data products may have certain metadata fields that other data products do not have. This is one reason that motivated us to use Apache Solr, a document oriented data store rather than a static, schema-based data store such as MySQL. Solr supports dynamic fields and gives the flexibility of having a common basic schema for all data products and extending the schema dynamically using dynamic fields to match the requirements of different data products. Even though the agent was designed specifically to cater the requirements of our use case, it can be modified to suit any type of generic scientific data management requirement by changing the *data product generation monitor* and the *metadata extraction logic*, which can be given as a plugin.

B. Server

The server is the central component in our framework 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 (ACL) 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. Web portal can also use ACL update API to modify the ACLs 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 a mature Java-based full text indexing and search solution [11]. We selected Solr over a relational database management system as the backend metadata store because of its flexible query support and search functionality. Solr *core* is a single index with associated transaction log, configuration files, and schema [11]. Solr supports having multiple cores in the same server where each core may index the data under different structures. We have created two cores in Solr. The first core is named *metadata* and it is used to store the metadata information and important attributes.

Second core is called *acl* and it stores the access control information. *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 index 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 that it accumulates metadata objects up to a configurable time interval, and then index them as a batch. The *query builder* component generates queries based on the query parameters received from the query API and executes them against the Solr index. The generated queries require 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 framework maintains only the metadata and some important attributes, enforcing a proper access controlling mechanism is very important. This is because the indexed data sets may contains high academic and research value. Therefore, our framework explicitly maintains the access control information. It is also possible for the owner of a metadata object to share the information with intended collaborators. Therefore, we have provided an access control update API to update the access control lists corresponding to a particular metadata object.

C. Web Portal

Fig. 2 shows a screenshot of the web portal, 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 may 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. To handle user management we have used a third-party user store called WSO2 Identity Server [12]. 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 shown in the Fig. 2 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 web portal.

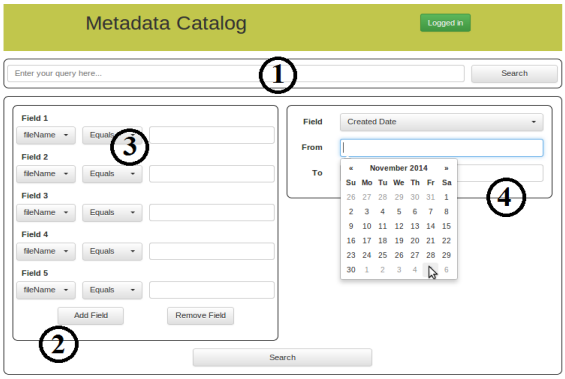

Fig. 2: Web portal.

III. Performance and Scalability 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 provides 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, wildcard queries, etc. 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 solution 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 datastore 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 fulltext 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 solution that uses Solr as the backend data store and *Sys-MySQL* to denote the solution 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. The content of the records was randomly generated. For both the solutions same test dataset was used. Results are based on ten samples, which were sufficient to attain insertion time and query time within $\pm 5\%$ accuracy and 95% confidence level.

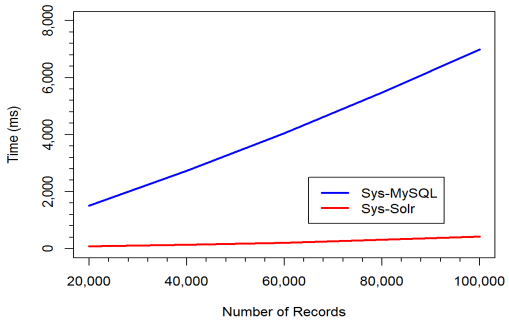

Fig. 3: Data insertion time.

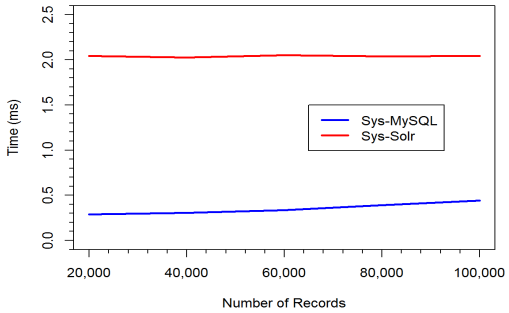

Fig. 4: Query execution time for exact match queries.

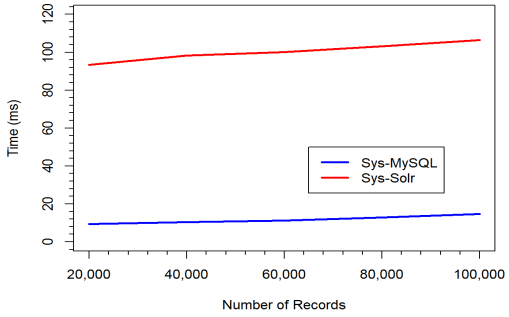

Fig. 5: Query execution time for range match queries.

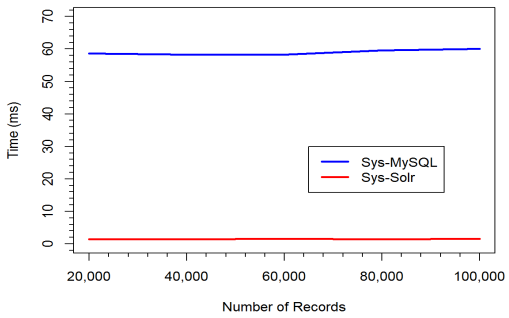

Fig. 6: Query execution time for full text search queries.

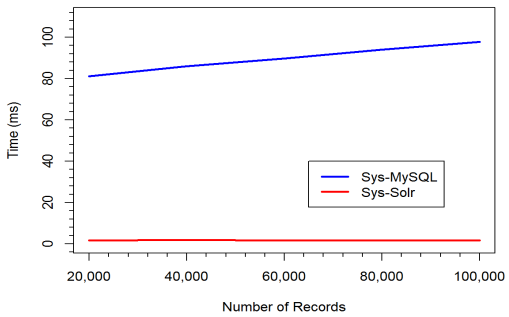

Fig. 7: Query execution time for prefix match queries.

A. Data Insert Performance

It is 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 solutions *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. 3. It can be seen that the metadata insertion time of *Sys-Solr* grows much slowly compared to the *Sys-MySQL* implementation.

B. Query Performance

We analyzed the query performance under six different types of queries. They are exact match, range, suffix match, prefix match, full text search, and wildcard 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 solutions. As seen in Fig. 4 and 5, for exact match and range queries *Sys-MySQL* performs better than *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*. Fig. 6 to 10 show the query execution time for full text, prefix match, suffix match, substring, and wildcard queries, respectively. For a database consisting of 100,000 records, wild card query response time of *Sys-Solr* is 99.2% lower than *Sys-MySQL*. Similarly response time for substring queries of *Sys-Solr* is 97.7% lower than *Sys-MySQL*. In prefix and suffix queries it is 99.2% and 97.8%, respectively. In full text search queries *Sys-Solr* response time is lower than *Sys-MySQL* by 97.5%. These performance gains are due to the inverted indexes in Solr.

C. Space Utilization

Even though the disk space is getting cheaper it is important to have an understanding of the disk space utilization of the two alternatives. In this test, we analyzed the storage utilization of each solution. Fig. 11 shows the disk utilization at different number of records. *Sys-Solr* consumes more disk space and has a relatively higher gradient than *Sys-MySQL*.

In conclusion, Solr-based implementation provides much bet-

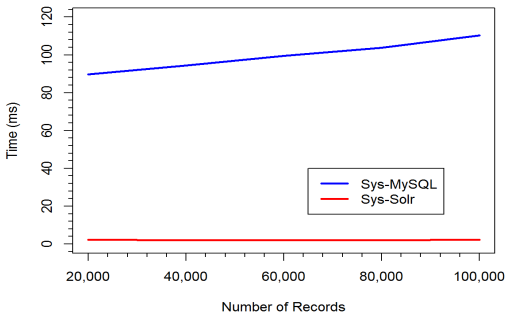

Fig. 8: Query execution time for suffix match queries.

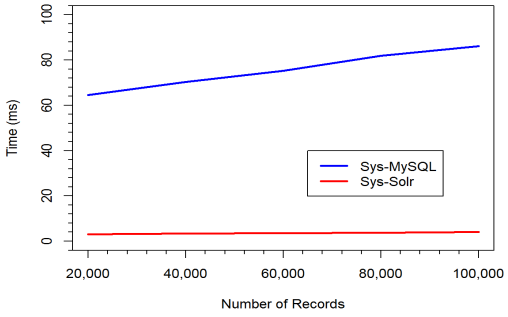

Fig. 9: Query execution time for substring search queries.

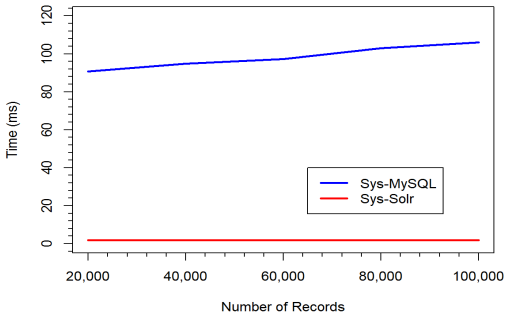

Fig. 10: Query execution time for wild card search queries.

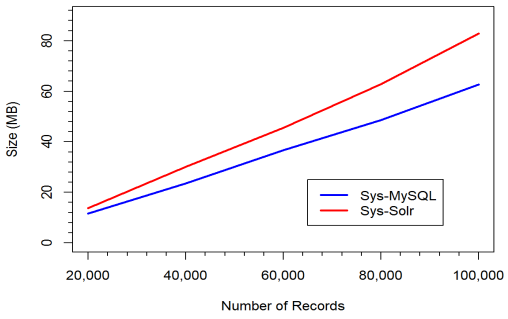

Fig. 11: Storage utilization.

ter 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 framework indexes only the metadata (not the actual data generated by the scientific experiment or simulation), we believe that the 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 data sets. Therefore, we decided to use Solr as the backend data store. We are currently in the process of integrating our data cataloging framework with Apache Airavata [13].

IV. Summary

A data cataloging framework for efficient management of scientific data was presented. A proof of concept solution was developed based on the use cases related to large volumes of Gaussian 9 computational chemistry simulation data. The proposed solution enables fast indexing and querying of a large number of static and dynamic metadata and attributes. For example, it resolves complex queries 91% - 99% faster than traditional RDBMS solution based on MySQL. The data cataloging framework can be adopted for other kinds of scientific data management by simply changing the metadata extraction logic. Compared to prior work, this enables our framework to be used as an independent component in a larger computing infrastructure by integrating it with web portals, workflows, or science gateways.

References

[1] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. 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," [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," [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 for data intensive applications," in Proc. ACM/IEEE conference on Supercomputing, Phoenix, Arizona, 2003.

[8] B. Plale et al., "Active management of scientific data," IEEE Internet Computing Special Issue on Internet Access to Scientific Data, vol. 9, no. 1, pp. 27-34, Jan 2005.

[9] "Cup users manual," [Online]. Available:http://www2.cs.tum.edu/projects/cup/docs.php.

[10] "JFlex," [Online]. Available:http://www.jflex.de/

[11] "Solr cores and Solr.xml," [online]. Available:https://cwiki.apache.org/confluence/display/solr/Solr+Cores+and+solr.xml

[12] "WSO2 identity server," [online]. Available:http://wso2.com/products/identity-server/

[13] S. Marru, et al., "Apache airavata: a framework for distributed applications and computational workflows," in Proc. ACM workshop on Gateway computing environments, pp. 21-28. Nov. 2011.