# Managing Groups of Files in a Rule Oriented Data Management System (iRODS)

Andrea Weise[1], Mike Wan[2], Wayne Schroeder[2], and Adil Hasan[3]

[1] Centre for Advanced Computing and Emerging Technologies (ACET), University of Reading, UK
a.weise@reading.ac.uk
[2] San Diego Supercomputing Center (SDSC), University of California, San Diego, USA
[3] Science and Technology Facilities Council, Rutherford Appleton Laboratory, UK

**Abstract.** The iRODS system, created by the San Diego Supercomputing Centre, is a rule oriented data management system that allows the user to create sets of rules to define how the data is to be managed. Each rule corresponds to a particular action or operation (such as checksumming a file) and the system is flexible enough to allow the user to create new rules for new types of operations. The iRODS system can interface to any storage system (provided an iRODS driver is built for that system) and relies on its' metadata catalogue to provide a virtual file-system that can handle files of any size and type.

However, some storage systems (such as tape systems) do not handle small files efficiently and prefer small files to be packaged up (or "bundled") into larger units. We have developed a system that can bundle small data files of any type into larger units - mounted collections. The system can create collection families and contains its' own extensible metadata, including metadata on which family the collection belongs to. The mounted collection system can work standalone and is being incorporated into the iRODS system to enhance the systems flexibility to handle small files.

In this paper we describe the motivation for creating a mounted collection system, its' architecture and how it has been incorporated into the iRODS system. We describe different technologies used to create the mounted collection system and provide some performance numbers.

## 1 Introduction

The SDSC Storage Resource Broker (SRB) is a data grid management system, which is able to unite and manage storage media of many kinds on heterogeneous systems across the network and, as a result, to make the storage infrastructure appear transparent for the end user. The software supports data grids, digital libraries, and persistent archives [1]. The success of the Storage Resource Broker has shown the demand for such data management systems. Currently petabytes of data are stored in various SRB systems, such as the Biomedical Information

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 321–330, 2008.

Research Network, UK eScience (e-minerals/e-materials project) or Taiwan National Digital Archives Program. The SRB is a complex system with built-in abilities which are not easily changeable. Therefore, one of the main goals for developing a new data management system is to provide the user with a system which is easier to administrate and is flexible enough to change with different existing environments.

The Rule Oriented Data System (iRODS), also developed by the San Diego Supercomputing Center (SDSC), is a state of the art open source client-server middleware to manage huge amount of data in a grid environment. It supports in addition to the SRB features real-time sensor data collection and large scale data analysis [2].

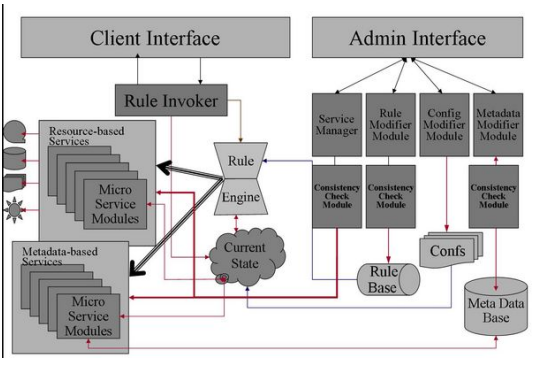


**Fig. 1.** iRODS Architecture

One of the main difference between iRODS and SRB is that within iRODS the data is managed automatically by rules [3]. Fig. 1 [2] displays the iRODS architecture with its' main modules. The client connects to the server and the client request will always go through the rule engine. Another subject which distinguish the iRODS from SRB are the levels of virtualisation which are workflow, management policy, service and rule virtualisation [2]. This is also provided through the administration interface with its' modules.

With the introduction of rules, the user is able the manage their own data in almost any way. Additionally, the implementation of services to manage or process the data, such as data conversion can be easily archived by the end user. The above mentioned rules are operations which are executed on the server side [2]. Each iRODS server has its' own rule engine. Fig. 2 [2] shows a flow diagram, which gives a basic overview about the way the rule engine works. The rule engine acts as interpreter of the rules [2] and is one of the core modules of iRODS. Rules are composed of the actual event, conditions, action sets and recovery sets [2]. The rule format can be seen as follows.

```
actionDef|condition|workflow-chain|recovery-chain
```

Rules consist of micro services. Micro services are C-functions [2], which can be provided by anyone to organise and structure the data. This above format put in action could look like the following example.

```
HAAW(*A,*B)||HAAW-Bundle(*A,*B)|nop
```

The name of the rule is `HAAW`. For this rule no conditions are attached. The micro service which is executed is called `HAAW-Bundle`. Two parameters (`A` and `B`) are passed to the function. Further, there is no recovery function provided. The sign "`|`" servers as a separator [2]. There are different types of rules namely, rules which can be applied immediately, rules where the application can be deferred and rules that could be applied periodically.

Due to the rule and micro service structure the iRODS becomes a highly configurable and flexible data management system, a system which is able to respond to many kinds of different requirements. Those requirements can be located in the data preprocessing or postprocessing. In terms of managing files within iRODS rules enable the user to automatically archive or organise files. To achieve that desired automatism a certain framework is needed, which will allow to plug-in any archive system.

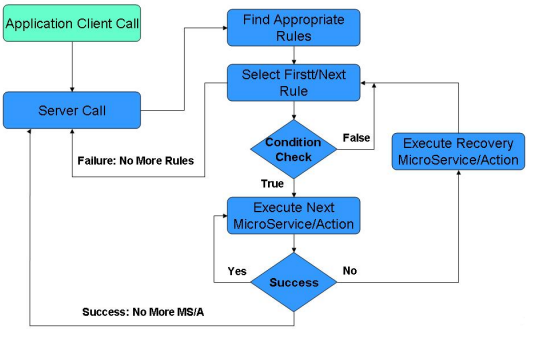


**Fig. 2.** Rule Engine Flow Diagram

## 2 Mounted Collection - Concept

### 2.1 Motivation

Nowadays almost every system has the desire to perform as fast as possible. While working with SRB a common problem was faced when archiving a large number of small files [4]. Small files typically make very inefficient use of mass storage capabilities which can have deleterious effects on the system performance [4]. This problem becomes very clear when tape systems are involved. The operations required to successfully write a file to a tape are time consuming, e.g. seek operation to find the right position [4].

To overcome the above listed performance problems within SRB the container concept was introduced. However, this concept of containers has certain shortcomings. The way containers are handled within SRB are not very efficient. Once the data is deleted from the container the space is not released and can not be reused. Therefore, the size of the container does not decrease. Further, the container concept does not leave much space for different archiving systems and are unfortunately tied to SRB. It is not possible to extract and therefore handle a container outside of the SRB system. To extract individual files from a container the metadata held within SRB is necessary. Consequently, the need emerged to develop a flexible system, which will be able to accept different kinds of mounted collections or archiving systems and which will overcome the weak points of the container concept.

### 2.2 Requirements

Out of the aforementioned reasons the following requirements for the mounted collections are stated. The mounted collection should

- be able to store all types of files and be a single file only
- be able to add, replace (update) and delete files to the mounted collection
- be able to pre-size the mounted collection
- be able to create mounted collection families
- hold checksums of files inside the mounted collection
- be as compact as possible (or compressible)
- be accessible/ usable inside and outside of iRODS
- be searchable to gain information about the files in the mounted collection without actually unpacking the mounted collection
- provide the ability to add limited metadata
- reflect the original folder structure
- should hold sufficient metadata to manage the mounted collection files in different environments

The possibility to take a whole mounted collection out of the iRODS system and still be able to access the information which are held in the mounted collection is a useful feature. This will allow to move huge amount of data efficiently between different systems.

In addition of the basic functionality a few further issues were taken into consideration during the development. The operations in connection with mounted collection have to be reasonable fast. Further, the overhead of the mounted collection structure itself should be as small as possible concerning the needed memory. To be able to use the mounted collections outside of the iRODS environment efficiently, flexible metadata capabilities are very useful. The possibility to compress individual files within the mounted collection will make the system more desirable. More, the mounted collection should support hierarchies and should be platform independent with a long term support.

Before integrating the mounted collection into iRODS a standalone mounted collection application was developed. By doing so, the performance of the

mounted collection operation could be determined. Furthermore, through the application the possibility is provided to handle the collection outside of iRODS.

There are many concepts on the market to organise data and information in a structured way. A very efficient way is offered by databases. During the development databases were examined regarding speed, interface possibilities, database file structure, database operation possibilities and open source based code.

Finally, three database engines were chosen - SQLite version 3.3.8., Berkeley Database version 4.4.20 and Apache Derby version 10.2.2.0.

SQLite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine [5]. This and the fact that the database file only consist of one single file made this engine an ideal candidate. As for the Berkeley DB, the developer promise a high performance and embeddable engine [6]. Derby is a light database engine using the SQL standard [7]. To be able to give an appropriate statement concerning the mentioned criteria all database engines were tested.

## 3 Finding the Right System

First of all, a standalone application was developed, which is currently a command line tool that performs the required mounted collection operations. On one hand, this application allows the user to handle the collections outside iRODS. On the other hand, the application is used for performance tests of the individual collections operations to determine the most suitable database engine.

The mounted collection can hold any kind of data (files). With each data certain metadata are associated, e.g. folder structure or file size, which will enable other potential applications to manage the collections with its' content successfully. The metadata can be extended according to the users request. For each file a checksum is computed using the Message Digest 5 (MD5) hash algorithm. This will detect any tampering with or corruption of the mounted collection. Files can be added, extracted, updated. The collection content can be listed in two different ways, short and long versions.

The test environment consists of functions, which are later embedded into the iRODS system. Each mounted collection operation is presented by a function. These procedures perform basic operation, e.g. open a collection, add file to a collection or delete file from a collection. For each function a certain version for each database engine has been implemented under the restriction to keep the code as similar as possible.

For testing purposes 1,000 to 100,000 files were created with different sizes (1KB - 10 MB). Then the time was measured for the mounted collection operations. Fig. 3 depicts the time measurement for all three database engine, executing the insert and extract function.

The Berkeley database is faster than the SQLite engine. The results for the Derby engine are not acceptable for the needed purpose. Currently Java programs are slower in performance than C programs. First of all, the Java code is

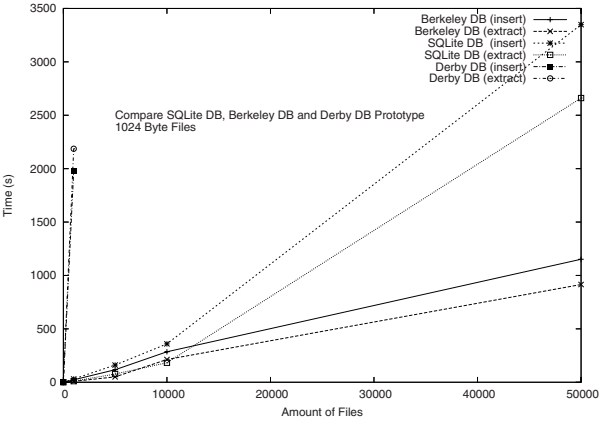


**Fig. 3.** Time to Store 1 KByte Files

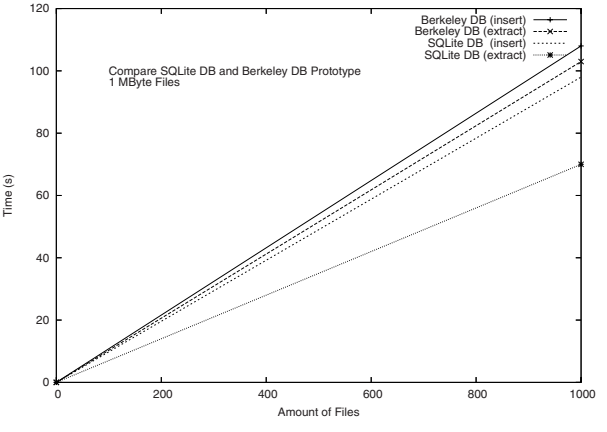


**Fig. 4.** Time to Store 1 MByte Files

run in a virtual machine. This causes a significant delay in the start up progress, since everything is being loaded [8]. By increasing the size of the files the SQLite database engine becomes faster compared to the Berkeley engine (Fig. 4)

Concerning the needed memory, the SQLite and Berkeley engines are very similar. The SQLite database consist only of one database file (Berkeley creates one file for each table). Due to the fact, that the SQLite database engine perfoms faster with growing file size and the SQLite API is easier to incorporate into C code, the SQLite database is the most suitable for a mounted collection prototype.

# 4 Mounted Collections within iRODS

By integrating the mounted collection into iRODS two things need to be achieved. Firstly, the collection forming process has to be executable manually and secondly, through rules. To accomplish that it is important to understand the concept of the iRODS framework.

## 4.1 iRODS Framework

The client-server architecture of the iRODS separates the software in two major parts. The icommands is one possibility to present the client side. Each basic file operations such as list or make a directory, are presented by an individual commands, e.g. `ils`, `imkdir` or `iput`, to submit a file to the iRODS. According to the given task (command), on the client side a certain amount of preprocessing is executed, e.g. mapping file name to the iRODS environment.

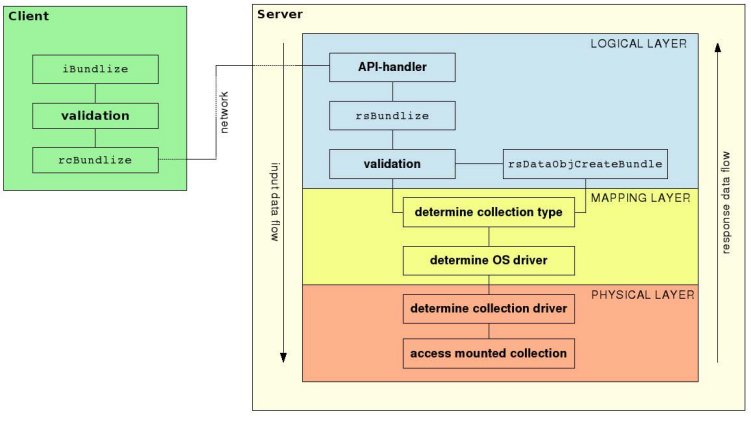


**Fig. 5.** iRODS Mounted Collection Framework

The iRODS can be divided in three basic layers on the server side, as shown in Fig. 5. The user requests are evaluated and processed in the logical layer. The logical layer queries the iRODS catalogue (ICAT). The ICAT (a database) holds all the metadata and information needed to manage the data within iRODS. To be able to map the logical name to the right physical location the mapping layer serves as glue layer. The actual access to the physical medium is done in the physical layer. On the basis of this layer frame work the iRODS becomes highly scalable. By providing the correct driver in the physical layer the system can adjust to many different environments. Based on that concept, the mounted collection framework was developed.

### 4.2 Mounted Collection Framework

**General Concept.** Mounted collections are only managed on the server side. That means, a file can not be transfered directly from the client into a collection. The user has to submit a file first to the iRODS and then the file can be transfered from there to a collection. This goes conform with rule oriented idea of iRODS. Once a file is relocated to the mounted collection, the actual file and its' metadata will be deleted from the iRODS environment. The collection itself is able to hold certain metadata, which are sufficient to reintegrate the files back into the iRODS.

A new icommand is introduced - `ibundle`. This command accepts a directory. This directory will be bundlized on the server side. It is not possible to submit a single file directly to a collection using an icommand. Further, the user has the possibility to specify the name of the new mounted collection. The iRODS itself contains "special" files. Those files are not visible to the user and can not be accessed directly by the user. The location of those files are determined by the iRODS system. Mounted collections are defined as special files they will be stored under their own collection hierarchy, which will be determined by the management system. This way, all the collection files will be hold in one location and the user can not accidentally tamper with the collection files. Mounted Collection files are registered with the ICAT, indicating that this file is a mounted collection/archive file. The user will also has a choice of different collection types. If the corresponding drivers are provided any archive or collection type is possible.

Retrieving a file from a mounted collection directly to the user (client side) is also not possible. First, the user has to extract the file within the iRODS. From there the file can then be transfered to the user.

Furthermore, the user will be able to segment the collection file. This is called mounted collection families. The user can specify a maximum size of a single collection member. If this size is exceeded a new member of this family is created. The family management is done within the mounted collection itself.

**Framework.** A basic overview of the mounted collection framework embedded into the iRODS framework can be seen in Fig. 5. On the client side the icommand `ibundle` is provided. The submitted user information, e.g. collection name or collection type, are evaluated on the client side. Those formated information are put into a structure, which will be transfered to the server side. From the iRODS framework an application interface (API) handler is provided, which will connect the client API call with its' opponent call on the server side. On the server side the new API `rsBundlize` was developed. The client counterpart is `rcBundlize`. In the logical layer the information provided by the the client are processed (validation), e.g. is the collection really existing? If the mounted collection is not available it will be created. In order to do that, another API was developed - `rsDataObjCreateBundle`. This will create the mounted collection and register this as a special file at the ICAT. To meet the conditions of the existing iRODS framework the new API's follow the concept of dividing the processing into 3 basic layers. That means, procedures for the logical layer, which

involves all ICAT handling, are provided. Further, glue functions are developed to be able to map the information to the physical layer. The physical layer has also mapping routines. First of all, the mapping to the correct collection type driver has to be done (mapping layer). After this, the mapping to corresponding driver for the current operating system in use is needed (physical layer). Each mounted collection/archive system may use a different descriptor to access their own collection/archive. This creates the need to allow any descriptor needed. The demanded flexibility is offered through the mapping system.

The glue function and the function to access the media can also be seen as API calls. Through this level of abstraction any other function from the level above is able to use the API calls in the lower level. This leads to the possibility to extended existing functionality to increase the functionality of the system. For example the icommand `ils`, which is used to list directory contents, is extended to list the mounted collection content as well.

The new mounted collections framework structure is modular. That means, each of the new functions can be accessed easily from the core functions of the iRODS system. This makes the framework accessible for the rule engine. With the provided functions microservices can easily be created and therefore, rules to automatically make use of the mounted collection can be established.

## 5 Future Work

The current prototype fulfils the requested conditions. But the SQLite database has a deflating limitation. Currently this engine supports terabyte-sized databases and gigabyte-sized strings and blobs [5]. To avoid any of those limitations a collection will be developed, which will be a hybrid out of the SQLite engine and 7-Zip. 7-Zip is a open source file archiver with a high compression ratio [9]. The combination of a collection with a high compression rate and certain amount of useable metadata, which will be fast searchable, will make the mounted collections even more interesting. With the incorporation of 7-Zip the limitation in size will only depend on the hardware.

Security is an important, but complicated issue. First of all, the mounted collections, since handled outside of iRODS needs more security features such as password protection and encryption possibilities. The user also only wants to authenticate once at the system. There are different systems available on the market. Kerberos is such a service [10]. Another identity management possibility is provided by Shibboleth [11]. In future both authentication management system could be embedded within iRODS.

## 6 Conclusion

In this paper the iRODS framework was briefly described and its' difference to the Storage Resource Broker are highlighted. A certain emphasis was placed on rules, which define the rule oriented data system. The need for mounted collections or archives within such systems where pointed out, which lead to the

presentation of the philosophy behind the new mounted collection concept. To reach a high flexibility in handling the collection a standalone application was developed. The new collection is based on a database engine. The standalone application was used to determine the best database engine through performance tests. Some results are presented within this paper. Further, major development steps which led to the mounted collection framework are outlined. The paper shows how the iRODS framework the mounted collection framework defines and how the collection was successfully incorporated into the rule oriented data management system. With the collection framework embedded into the iRODS this management system becomes even more flexible and powerful compared to other existing systems. Furthermore, the made enhancement of iRODS by the described collection framework makes the system more attractive for the public. iRODSs' rule oriented nature is supported in any way by the here presented contribution.

## Acknowledgements

The authors would like to thank The National Science Foundation (NSF) [USA], The National Archives and Records Administration (NARA) [USA] and The Science and Technology Facility Council [UK] who support this research financially. Further, the authors would like to thank Prof. V.Alexandrov (ACET) for his support while developing this software.

## References

1. Rajasekar, A., Wan, M., Moore, R., Schroeder, W., Kremenek, G., Jagatheesan, A., Cowart, C., Zhu, B., Chen, S.-Y., Olschanowsky, R.: Storage resource broker - managing distributed data in a grid. Technical report, San Diego Supercomputer Center (SDSC), University of California
2. About irods (September 19, 2007), `http://irods.sdsc.edu/index.php/Main_Page`
3. Rajasekar, A., Wan, M., Moore, R., Schroeder, W.: A prototype rule-base distributed data management system. Technical report, Paris, France (May 2006)
4. Strong, B., Corney, D., Berrisford, P., Folkes, T., Moreton-Smith, C., Kleese-Van-Dam, K.: Key lessons in the efficient archive of small files to the cclrc mss using srb. Technical report, IEEE (2005)
5. About sqlite (September 13, 2007), `http://www.sqlite.org`
6. Why oracle berkeley db? (September 13, 2007), `http://www.oracle.com/database/berkeley-db/index.html`
7. What is apache derby? (September 13, 2007), `http://db.apache.org/derby/`
8. Lewis, J.P., Neumann, U.: Performance of java versus c++ (accessed on December 3, 2007), `http://www.idiom.com/~zilla/Computer/javaCbenchmark.html`
9. 7-zip (September 17, 2007), `http://www.7-zip.org/`
10. Clifford Neumann, B., Tso, T.: Kerberos: An authentication service for computer networks. Technical report, Institute of Electricaland Electronics Engineers (September 1994)
11. Shibboleth, `http://shibboleth.internet2.edu/`