# First iRODS Experience in a Neutrino Experiment

*F. Di Lodovico*[a], *A. Hasan*[b], *Y. Iida*[c], *T. Sasaki*[c]

[a]Queen Mary University of London, London, UK
[b]University of Liverpool, Liverpool, UK
[c]High Energy Accelerator Research Organisation (KEK), Tsukuba, Japan

## Abstract

The T2K experiment is a long-baseline neutrino experiment with the primary aim of observing muon into electron neutrino appearance. An important task is the determination of the quality of the collected data. In this paper we describe the first experience gained in managing the T2K data quality data for the near detector such that it is stored and accessible to collaborators world-wide in a timely fashion. We also describe a set of rules developed for the project that have much more general applicability. This is the first use of iRODS in a neutrino experiment.

***Index Keyword Terms—*** T2K, neutrino, iRODS

## 1. Introduction

The Tokai–to–Kamioka (T2K) experiment [1] is a long baseline neutrino experiment whose primary goal is to measure the electron into muon neutrino oscillation. T2K consists of the world highest intensity muon neutrino beam produced at the 50 GeV Proton-Synchrotron at J-PARC (Tokai) being directed at a far detector, Super-Kamiokande (SK), 295 km away from J-PARC (see Fig. 1).

The initial proton energy is limited to 30 GeV and will then ramp up to 50 GeV. A near detector system, ND280, is located 280m away from the target and consists of several sub-detectors. The main purpose of the ND280 detector system is to measure the neutrino flux and spectrum before the oscillation. T2K started to take data in January 2010.

The assessment of the quality of the data before any analysis is performed is crucial to achieve reliable results. The corresponding data–quality files needs to be available to world–wide collaborators to be checked as soon as produced at J–PARC. This requires a data–distribution system and iRODS [2] provides the best system to achieve that.

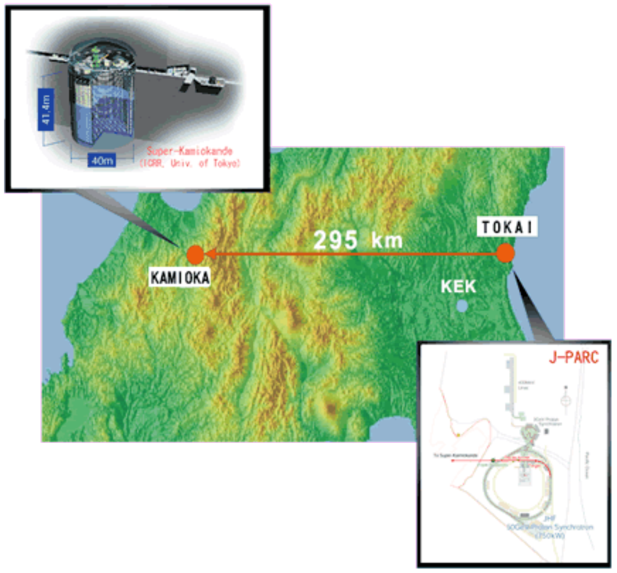


**Figure 1: Schematic overview of the T2K detector.**

## 2. Data Quality at T2K

The data quality assessment for T2K has a crucial role, as it allows an assessment of whether the data can be used for analysis as soon as the data are collected. The data quality files that we need to store and access remotely were a total of about 60000 during 2010 for the near detector. This spans a fraction of the year due to shutdown periods. The number of files is expected to increase significantly in the near future with the increase in the number of protons on target. Not all the files have the same size. We can approximately divide the sample into three categories: very-small-size files (less than 1 kB), small-size files (2 MB), large files (above 10 MB), that correspond to about 30%, 30% and 40%, respectively of the total. The total size of all the files is about 360GB. From the data–quality point of view, the requests to the data distribution system that we need to use are several, and iRODS reveled itself as the best match to our needs. The summary of our requests is shown below, whilst the technical solution offered by iRODS is described in Sects. 3 and 4:

- storage of files as small as a few kB;
- Easy access to the files for approx 500 collaborators predominantly to download files. The easiest way for a casual user to

access the files is through a web interface, where a read–only account is provided to the T2K collaborators.

Moreover, the line command is also available for users who want to have more tailored access to the data. Search tools are also needed for both the web–interface and the command–line user.

Fig. 2 shows the current flow diagram for the data quality. The data are collected at the T2K near detector ND280, and then processed at J-PARC and data quality files produced. Those files are then copied into iRODS, from where the user can access them either through the graphical interface or the command line.

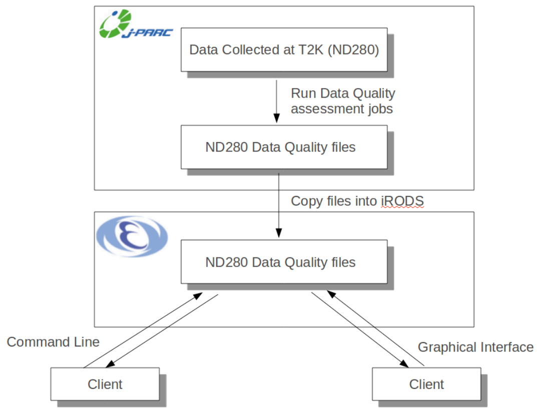


**Figure 2: T2K ND280 data quality flow diagram.**

## 3. iRODS

The integrated Rule Oriented Data System (iRODS) is developed by the Data Intensive Cyber Environments (DICE) group at the University of North Carolina. The system was built on the experience of the successful Storage Resource Broker (SRB) [3] also developed by the DICE group. The iRODS is an open-source, policy-based data management system that abstracts the underlying storage systems. The abstraction makes it easier to replace storage or change the type of storage, or even add new storage without affecting access to the system. This is achieved by creating logical-to-physical mappings for the storage resource and file location. Access to the system uses the logical names and iRODS maintains the logical-to-physical mappings in an SQL database (currently PostgreSQL, ORACLE or MySQL). Physical access to a storage system is handled by an iRODS driver that presents a POSIX standard interface to the system. New types of storage system may only require a new storage driver to be written.

The iRODS also provides controlled access through a number of different authentication mechanisms: Kerberos, secure username/password and Grid Security Infrastructure (GSI). User groups can be created making it easier to manage permissions for a set of collections (users only need to be added or removed from the group in order to inherit/disinherit the access permissions). Access to collections of data can be more finely tuned by using the iRODS rules to, for example, grant access temporarily to a collection of data.

The iRODS allow policies to be imposed on any iRODS object (that is users, storage resources or data) through the creation of iRODS rules which are server-side workflows that are executed by the iRODS rule engine. The iRODS rules follow an event-condition-action approach with the rule engine checking each rule in the rule file from top to bottom. The first rule that satisfies the condition is executed. This schema allows rules to be overridden by placing the overriding rule above the overridden rule in the rule file. A rule is executed sequentially from left to right.

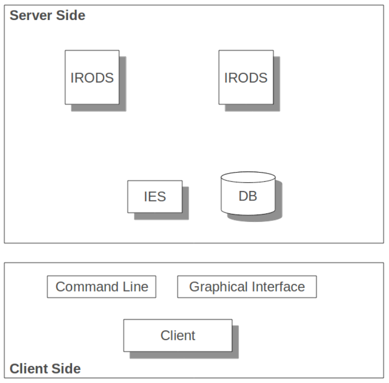


**Figure 3: Diagram of a typical iRODS system.**

An iRODS system consists of iRODS server application running on each storage resource plus an iRODS server application that interfaces to the iRODS metadata catalogue that holds the logical-to-physical mappings and access control information as shown in Fig. 3. An iRODS rule engine runs on each resource including the iRODS enabled catalogue server (IES). Clients communicate with iRODS through client-side applications (either command-line or GUI based) by creating an iRODS session and then interacting with iRODS. Each interaction with iRODS triggers the rule engine to check if the event satisfies any conditions that require a rule to be invoked. By default, all events trigger the rule-engine with the exception of the command to list the contents of a collection (the *ils* command) for performance reasons.

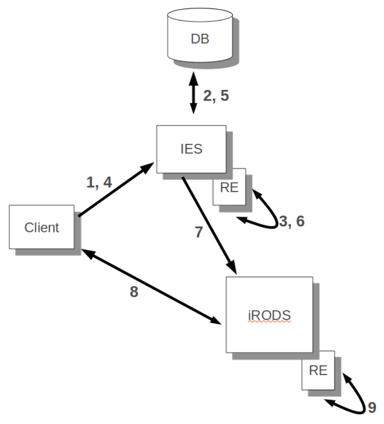


**Figure 4: An example iRODS workflow.**

An example iRODS workflow is shown in Fig 4 where an iRODS session is created by the client first connecting to an iRODS server (1) which in turn queries the iRODS metadata catalogue database to verify the client's credentials (2) before and after the server queries the catalogue the rule-engine (RE) is triggered (3) which checks for any rules that the event's condition satisfies and executes the first that is satisfied. Once the client is verified an iRODS session is created. The client can then, in this example, store data in an iRODS controlled storage resource. The client sends the store request to the server (4) which causes the server to lookup the physical location of the storage server in the database (5) and triggers the rule-engine to search for any rules to execute that match the events condition (6).

The server then passes the request to the destination storage server (7) and the client sends the data directly to the destination server (8). After the data has been transferred to the server the rule-engine fires executing the first rule that successfully satisfy the event condition (9). Once the data have been successfully stored the iRODS metadata catalogue database is updated with information on the data name and location.

The iRODS provides support for the IBM High Performance Storage System (HPSS) [4] that provides the KEK hierarchical storage system.

## 4. Architecture

The data quality requires the storage to be visible to collaborators around the world and to be able to manage large numbers of small files efficiently. At the time of the project the KEK HPSS system was not tuned to handle small files and access was restricted to KEK. So, iRODS described in Section 3 was used to manage the data stored in HPSS. The iRODS-based architecture shown in Fig. 5 took care of managing the small files and provided remote access to the content. Specific ports are opened between JPARC and KEK that allow iRODS communication. Data and metadata are sent from JPARC to the iRODS cache storage at KEK (1). Successful storage of the data triggers the rule engine which executes a rule that both tars and stores the data in HPSS if the total data size is greater than 1 GB (2) or backs up the data to a backup storage resource if the total size of the data is below 1GB (3). The rule executes every six hours checking the total size of the data on the cache, once it reaches 1GB the data are tarred and stored in HPSS and the cache and backup copies are deleted.

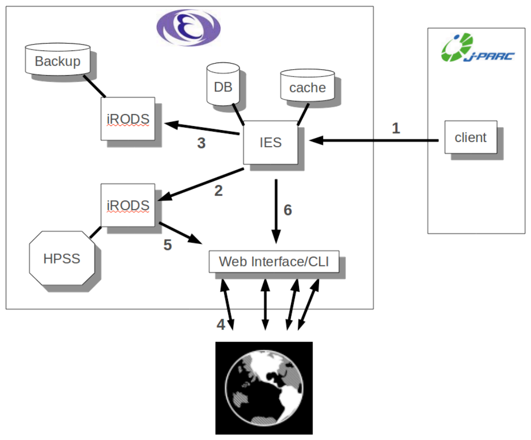
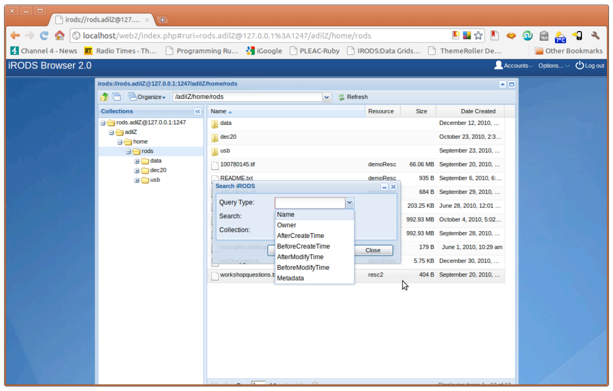


**Figure 5: Architecture for the T2K iRODS-based system.**

Remote T2K users (4) access the data either through a modified version of the iRODS PHP browser or through the client-side irods icommands. In both cases a read-only iRODS account is used to access the data to prevent accidental modifications to the data.



| Name | Resource | Size | Date Created |
|---|---|---|---|
| data | | | December 12, 2010, ... |
| dec20 | | | October 23, 2010, 2:3... |
| usb | | | September 23, 2010, ... |
| 100780145.tif | demoResc | 66.06 MB | September 20, 2010, ... |
| README.txt | demoResc | 935 B | September 6, 2010, 6:... |
| add-clients.sh | demoResc | 684 B | September 29, 2010, ... |
| [illegible] | [illegible] | 203.25 KB | June 28, 2010, 12:01 ... |
| [illegible] | [illegible] | 992.93 MB | October 4, 2010, 5:02... |
| [illegible] | [illegible] | 992.93 MB | September 28, 2010, ... |
| examples.desktop | [illegible] | 179 B | June 1, 2010, 10:29 am |
| runDoxygen.rb | demoResc | 5.75 KB | December 30, 2010, ... |
| workshopquestions.t | resc2 | 404 B | September 20, 2010, ... |

**Figure 6: Screenshot of the modified iRODS browser.**

The PHP iRODS browser provided practically all the functionality that was required with the exception of two important features: a read-only version and metadata search. The upload and delete functions were removed from the browser and a search menu was added allowing basic search of the iRODS attribute-value-unit metadata and iRODS file metadata (see Fig. 6).

### 4.1. iRODS Rules

The advantage of the iRODS system over a standard data management system such as the SRB is the ability to implement policies on users and data. For the T2K experiment several rules were written to prevent users from changing the read account password and to manage data. All rules were placed in the default rule file.

The rule to prevent changes to an iRODS account password required implementing a pre-processing rule that would be triggered for a modify user account event:

```
acPreProcForModifyUser(*UserName,*Option,*NewV
alue) || ifExec(
         *UserName == rods,
           ifExec(
                   *Option == password,
                      msiWriteRodsLog(”Alert:
                        attempt to change password
                        for *UserName”,
                            *status)
                      ##fail, nop##nop, nop, nop
                 ),
           nop, nop, nop
      ) || nop
```

The rule is only executed if the user account is ‘rods’ in which case the rule writes out a message to the iRODS log file and then forces the rule to fail resulting in a failure of the client side application (*ipasswd*).

The management of T2K data required a rule that would (a) backup the data on the cache disk to a backup resource if the total size of data on the cache was below 1GB, (b) tar and copy the data to the HPSS system if the data were greater than or equal to 1GB. The rule *acBundleOrReplicate* is given by:

```
acBundleOrReplicate(*collPath, *cacheRes, *backRes,
                       *archive, *threshold)||
msiCheckCollSize(*collPath, *cacheRes, *threshold,
                       *aboveThreshold, *status)##
ifExec(
  *aboveThreshold == 1,
  msiWriteRodsLog(”Creating bundle”, *status)##
  msiPhyBundleColl( collPath, archive, *status)##
  msiWriteRodsLog(”Finished bundling”, *status),
  nop##nop##nop,
  msiWriteRodsLog(”Starting to backup files”,
    *status)##
  acGetIcatResults(list, COLLNAME LIKE
      ‘*collPath’, *List)##
  forEachExec(
     *List, msiGetValByKey(*List, DATA_NAME,
        *Data)##
          msiGetValByKey(List, COLL_NAME,
            *Coll)##
          msiGetValByKey(List,
                              DATA_RESC_NAME,
                              *dataRes)##
           ifExec(
             *dataRes == *cacheRes,
             msiWriteRodsLog(”Replicating
                 file *Coll/*Data”, *status)##
                 msiDataObjRepl(*Coll/*Data,
                   verifyChksum++++backupRes
                   cName = *backRes, *status)##
               msiWriteRodsLog(”Completed
                 replicating file *Coll/*Data”;
                 *status),
                 nop##nop##nop,
                 nop,
                 nop
           ),
            nop##nop##nop
  ),
nop##nop##nop
)|| nop##nop
```

The rule first finds if the total collection size is greater than a threshold value (1GB) using the *msiCheckCollSize* microservice and returns 1 if it is. In that case the rule involves the *msiPhyBundleColl* microservice to tar-up the collection and store the contents on the HPSS resource. If the total size is below the threshold the list of files in the collection is obtained with the *acGetIcatResults* rule and each file in the collection is backed up to the back resource using the *msiDataObjRepl* microservice. The *acBundleOrReplicate* rule runs periodically checking the newly input files. A rule to delete data from the backup resource that is stored in HPSS also runs periodically.

These rules were developed for the T2K iRODS project, but are of general applicability.

## 5. Performance

Treating the large number of small files individually introduced an unavoidable overhead when storing or retrieving (see Table 1). This was due to the finite time needed to establish the connection between the client and server.

| Mode | Bulk Mode | Individual Files (JPARC) | Tar File (JPARC) | Individual Files (KEK) |
|---|---|---|---|---|
| *iput* | 250s | 630s | 210s | 213s |
| *iget* | N/A | 1396s | 456s | 410s |

**Table 1: iput and iget comparisons for 1.2GB collection of files.**

Table 1 shows the overhead for handling large numbers of small files compared with a treating the collection of files as a single unit. The iRODS provides the possibility to treat individual files as a collection with the *iput* commands through the 'bulk' option which minimizes the number of connections between the client and server and provides a performance much closer to that for an individual file.

Unfortunately, the data quality data cannot currently take advantage of the bulk mode as data must be archived as soon as it is produced and not all files in a directory must be copied. If the ability to filter the input collection were implemented this would be highly desirable.

The poorer performance for the *iget* command for small files compared with the tar file is due to a number of factors:

- The overhead of establishing connections between client and server,
- The overhead in translating the logical name to a physical file in order to fetch the data.
- The time to write to the target disk being different to the time to read from the disk which can become significant if the disk is fragmented.

The last column shows the total time for importing and exporting individual files within KEK. The numbers show performance comparable with the transfer of an individual file. This suggests that a majority of the latency is actually not due to the iRODS infrastructure, but due to the network connectivity and also the storage resource at JPARC.

These are early days for the iRODS setup for T2K and efforts are being made to improve the performance for small files in conjunction with the iRODS developers that will be of benefit to the project and the community in general.

## 6. Conclusions

The T2K experiment in conjunction with the KEK computer centre has setup an iRODS system to manage the storage and access to the T2K data quality data for the near detector. This is the first neutrino experiment (and possibly particle physics experiment) to adopt the iRODS for data management. The setup has simplified the management of the small files in HPSS and has provided easy access to the content to collaborators world-wide. More importantly, the effort to manage and maintain the system is proving to be extremely light compared to other alternatives. Another important result of the adoption was the development of a set of rules that greatly help in the management of small files that we hope will be of wider use by the community.

## 7. Acknowledgements

This work has been supported in part by the EU project T2KQMUL - grant agreement no. 207282-T2KQM and by the KEK foreign researcher grant. We are also grateful to Jean-Yves Nief (ccin2p3), Mike Wan (DICE group), Wayne Schroeder (DICE group) and Arcot Rajasekar (DICE group) for technical help and advice.

## 8. References

[1] Y. Itow et al., The JHF-Kamioka neutrino project, hep-ex/0106019, 2001.

[2] The DICE group, IRODS https://www.irods.org/index.php/IRODS:Data_Grids,_Digital_Libraries,_Persistent_Archives,_and_Real-time_Data_Systems.

[3] The DICE group, SRB http://www.sdsc.edu/srb/index.php/Main_Page.

[4] High Performance Storage System, http://www.hpss-collaboration.org/