Event Processing in Policy Oriented Data Grids

Arcot Rajasekar[1], Reagan Moore[1], Mike Wan[2]

Data Intensive Cyberinfrastructure Environments
[1]University of North Carolina, Chapel Hill, NC
[2]University of California at San Diego, La Jolla, CA
{sekar,moore,mwan}@diceresearch.org

Abstract

Data Grids are used for managing massive amounts of data (Peta scale) that are distributed across heterogeneous storage systems. As such they are complex in nature and deal with multiple operations in the life-cycle of a data set from creation to usage to preservation to final disposition. Administering a data grid can be very challenging (not only for system administrators, but also for data providers and user communities). Data grids are reactive systems that handle events based on contextual information. They also maintain transactional capabilities in order to ensure consistency across distributed storage systems. We are developing a data grid system called integrated Rule Oriented Data Systems (iRODS) manage the phases of the data life-cycle using ECA-type rules. Such a system not only captures the complex operational policies of a data grid but also provides a declarative semantics for describing event processing based on a side effects ontology and context information stored in the data grid. In this paper we describe the event management and processing being implemented in iRODS and how a distributed rule engine is used to handle actions in a data grid. The iRODs data grid can be viewed as a complex, distributed event processing system providing data life-cycle management capabilities using a rule-oriented architecture.

Introduction

Grid technologies provide software mechanisms to support distributed computation and access to distributed data. Data Grids [Chervenak et.al., Rajasekar 2003, Moore 2006] provide access to large collections of data whose sizes are measured in Peta Bytes and hundreds of millions of files. A data grid provides access to geographically distributed heterogeneous data resources assembled from file systems, tape archives, relational databases, semi-structured data systems, video streaming systems, and sensor data streams. Data grids support operations for end-to-end data life-cycle management and are used to share data across inter and intra-disciplinary groups without having to aggregate the data into a centralized location. Data grids provide a virtualized interface to applications, hiding the idiosyncrasies of the underlying infrastructure from users and applications. They also provide semantic indexing using keyword-value-unit triplets as associative metadata describing the content and context of the data. Search on the metadata supports discovery of desired files in a collection. Data grids also support long-term preservation of data collections through management of technology evolution (hardware, software and data formats), support for contextual information drift, and virtualization of administrative functions.

Data grids support the multiple operations required by the data life-cycle. In addition to being fault-tolerant and network latency aware, they provide facilities for multiple-levels of authentication and authorization, facilities for auditing and accounting, support strategies for data placement, replication, backup, archiving and versioning, interfaces for rapid data ingestion and access, visualization and third-party data movement, support for server-side services for data assimilation, analysis, transformations, fusion and domain-specific server-side applications. The data grid operations are quite complex in nature and can be viewed as chains of functions. Moreover, for the same operation, depending upon the context (defined by the user profile, resource attributes, contextual information of the data collection that is being accessed and administrative policies) the same operation can lead to a different chain of actions to be performed. The contextual information in a data grid is captured in a persistent (relational) database holding information about data sets, collections of data, users, resources, meta data and system ontologies, services, operations and policies.

Integrated Rule Oriented Data Systems (iRODS)

Data grids can be viewed as large distributed event processing systems handling concurrent events from multiple users, administrators and temporal cues. In the Data Intensive Cyber Environments [DICE] group, based at UNC and UCSD, we have been conducting research and prototype development on policy-governed data grids

called the integrated Rule-Oriented Data Systems [Rajasekar 2006, iRODS, Moore 2007]. IRODS is adaptive middleware that provides a flexible, extensible and customizable data management architecture by encoding operations as workflow functions that are defined by sequences of micro-services. The work flow functions can be interpreted as policies and are encoded as user-defined and/or administrator-defined ECA-type rules [Dayal 1996]. The "semantics" (or side effects) of a functionality can be decided by users or data providers within limits imposed by the system administrators. Hence, changes to a particular process or policy can be easily constructed by the user and tested and deployed without the aid of system and application developers. The user can also define conditions when these rules get triggered thus controlling application of different rules and rule sets based on current events and operating conditions.

The programming of rules in iRODS can be viewed as lego-block type programming. The building blocks for the iRODS are *micro-services* - small, well-defined procedures/functions that perform a certain task. Users and administrators "chain" micro-services to implement a larger macro-level functionality (as rules) to be executed on triggered events. The rules can be published for use by others. For example, one may encode a rule that when accessing a data object from a collection named "B", additional authorization checks need to be made. These authorization checks can be encoded as a set of additional rules with different triggers that can fire based on current operating conditions. In this way, one can control access to sensitive data based on rules and can escalate or reduce authorization levels dynamically as the situation warrants. The iRODS rule engine design builds upon the application of theories and concepts from a wide range of well-known paradigms from fields such as active databases, transactional systems, logic programming, business rule systems, constraint-management systems, workflows, service-oriented architecture and program verification.

We have implemented a prototype of the iRODS system. The system consists of the following components:

1. A three-layer peer-to-peer server system that interfaces to a remote client on one side and the storage and data providers at the other side. The middle layer implements the intelligence of the system through explicit rules and micro-services.
2. An iCAT catalog system based on a relational database. Mechanisms are provided to access the database using its native API while exposing higher-level functions that can be used to interact with the iCAT catalog.
3. A rule engine that interprets rules from a rule base and executes micro-services.
4. A rich set of Client-side APIs and utilities.
5. An administration utility.
6. A Messaging Server that can be used for communications between micro-services running in parallel or at different times.
7. A Rule Scheduler which can schedule rule invocations. The scheduler can also be used for delaying operations or for asynchronous execution.

Figure 1 provides an architectural diagram for the iRODS system. Figure 2 provides a servic-level diagram of the iRODS system.

iRODS rules are invoked either by client operations (events such as ingestion of a new file, access to a copy of a file, etc) or by temporal cues or by events that are triggered by changes in the state of the iCAT persistent database. The distributed rule engine fires the rules at the storage location of the data and executes the relevant micro-services. The micro-services can check for conditions, change the metadata state or perform an operation such as replicating a file, or broadcasting an email, etc. The Administrative interface (under development) will provide a means for defining rules (a rich rule language supports atomic, deferred, and periodic execution of hierarchical rule sets), checking for rule base consistency, and managing other aspects of the data management system.

The iRODS prototype has been developed based on our experience in building and deploying a first generation data grid system called the Storage Resource Broker [SRB]. The SRB manages collections in a wide variety of domains from bioinformatics, to astronomy sky surveys, to earthquake simulations, to sensor data streams. It has been shown to scale to petabytes of data with 100s of millions of files. The main drawback of the SRB was the use of hard-coded data management policies. If a user wanted to perform complex sets of operations on datasets, they had to program at the client level and move the data from the server to perform the operation. IRODS obviates the need for this through its flexible rules and micro-services allowing operations to be performed at the data server.

Event-driven Data Management

Distributed data management is normally viewed as a set of operations (such as get, put, remove, etc) that are performed on datasets residing at remote sites. In iRODS, we took a different approach and viewed the operations as occurring within a distributed event-based system. Our view takes the following principles in defining the operations that are performed by a data grid:

1. an operation is a triggering event that can result in different actions based on the context in which the event occurs.
2. An event triggers a sequence of actions and can trigger additonal events.
3. Events are coded as ECA-type rules (called *iRODS rules*) that use a guard condition "C" to check the context for application. . The actions "A" are called *micro-services* (in the current proto-type they are C-language functions) that have well-defined semantics attributed to them. In particular, they provide an external semantics based on the side-effects they cause in the outside world as well as in state information changes within the iCAT persistent database.
4. Departing from normal ECA semantics, only one iRODS rule per event can succeed. The rules are prioritized and applied in a specified order and the rule invocation completes when an ECA rule succeeds. (if needed, there is syntax sugar in the language which all allows application of all rules)
5. Extending ECA semantics, an iRODS rule is transactional in nature: it leaves the distributed system in the samestate as before if the action sequence fails at any part. This is necessary to make sure that the data grid is not left in an unstable state. To enable this transactional nature, every micro-service used in a rule is associated with *a recovery micro-service*. Most of the side-effects can be rolled back except for a few external effects that will be discussed later. With this, we have an extended ECAR type of rule (R standing for recovery). In relational databases transactional properties are handled with commits and rollbacks, even extending them into triggers. But in the case of a data grid, because of the complexity of operations involved, and the diversity of the side effects, explicit recovery is needed.
6. An event can trigger sub-events (also encoded as iRODS rules) and hence one can look at the operational semantics as a hierarchical event tree.
7. Operationally the sub-events and actions can happen serially, in parallel, in a distributed execution environment and/or in a time-delayed mode.
8. Micro-services can communicate with each other in multiple ways:
    a. Using parameter passing
    b. Using a white-board which holds the local execution context. The white board can be ported or checkpointed for execution by a remote micro-service or for delayed execution.
    c. Using the global status and contextual information stored in the persistent iCAT database.
    d. Using a messaging system to transmit serialized messages. This way dataflow and workflow operational characteristics can be applied.
9. The set of iRODS rules that are applied is normally fixed for each session for a user connection to the iRODS system. But flexibility is built into the system that can allow different sets of rules to be used for different users and groups. Hence a data grid can apply one set of rules for the seismic community and another set for the astronomy community.

With the above principles in mind, we defined a rule system with rules of the form:

$A :- C \mid M_1, \ldots, M_n \mid R_1, \ldots, R_n$

where A is the name triggering event,

C is the guard condition for the rule to fire

M_i is a micro-service or a rule, and

R_i is a recovery micro-service.

The checking of the guard condition may not change the state of the system, but only evaluate the context as defined by the white-board, parameters and results of iCAT database queries. When more than one rule is defined for an event, the rules are tried in a pre-defined priority order. The first rule whose guard condition succeeds is applied and its sequence of micro-services/sub-events is executed. If any of the micro-services fail, the recovery micro-services are executed. Then, the rest of the rules are checked for applicability repeating the process.

As an example, consider an ingestion event, where a new dataset is being uploaded into the iRODS data grid. There is a sequence of actions that need to be performed for the data to be store in the grid. First, the user's permission for performing the action in the data collection needs to be checked, then the users permission to store the file in a particular physical location is checked, then depending upon the type and size of the file, a data movement protocol is selected (to do the move in paralllel, in multiple hops by staging , etc). and then finally the dataset is stored into the system. Then the information about this new addition is entered into the iCAT database with information about its physical location, its logical name, ownership, size, type, etc. Moreover, if other events are triggered by this ingestion (such as extraction of metadata, replication of the data into other distributed resources, or performance of some transform or integrity check) these events can lead to more rules being fired and actions taken therewith. We show below a simpler (example rule) that can be used for such an ingestion event where contextual information plays a role. In this case, the context is based on either a user group or because of the type of data being ingested. [Many of the details of the parameters are not shown for clarity purposes. In each of the rules, the first defines the event and the guard condition, the second line shows the action sequence and the third the recovery action sequence.]

(a) OnIngest :- userGroup == astro
| findResource, storeFile, registerInIcat, replicateFile
| nop, removeFile, rollback, unReplicate.
(b) OnIngest :- userGroup == seismic && size > 1GB
| findTapeResource, storeFile, registerInIcat, seisEv1
| nop, removeFile, rollback.
(c) OnIngest :- userGroup == seismic && size <= 1GB
| findTinyResource, storeFile, registerInIcat, seisEv2
| nop, removeFile, rollback.

Rule (a) is applied for users within user group "astro". When a file is put into the data grid, a storage resource is selected, the file is stored, the state information is registered into the iCAT catalog, and the file is replicated. Note that this can be a sequence of images from a telescope that is being ingested.

Rule (b) is applied for users within user group "seismic" for file sizes greater than 1 Gigabyte. When a file is put into the data grid, a tape storage system is selected, the file is stored, the state information is registered into the iCAT catalog, and an additional micro-service called "seisEv1" is executed. This might trigger a delayed or asynchronous action on the newly ingested file such as extraction of metadata, searching for some seismic markers, etc. The input may consist of a stream of packets from a sensor network.

Rule (c) is also applied for users within user group "seismic", but for files smaller than 1 Gigabyte. When a file is put into the data grid, a "small" storage system is selected, the file is stored, the state information is registered into the iCAT catalog, and an additional micro-service called "seisEv2" is executed.

One might define another event as shown below which is triggered after ingestion of an image to extract astronomy metadata using a template called "astroTemplate". The descriptive metadata is loaded as attributes for the image and stored in the iCAT catalog for subsequent query and discovery operations.

(d) OnNewFile :- Collection == astroColl
| extractMetadata (astroTemplate,M), ingestICat(M).
| nop, rollback.
(e) OnNewFile :- Collection == astroColl
| sendEmail(astroCollOwner)

In the above case, when a new file is registered into the iCat, the OnNewFile event triggers application of the rules (d). If (d) fails for some reasons, then (e) fires and an email is sent to the collection owner about the failure.

The triggering events in iRODS can be either user generated (ingest a single file or a collection of files), or dynamically generated from the iCat database (periodically perform replication checks or integrity checks), or queued events in the messaging system (workflow operations on a file based on flagged status messages). The iRODS system has many rules coded for firing based on different types of events. These rules range from events of the data life-cycle such as ingest, remove, copy, move, replicate, trim, archive, synchronize, modify, change access permission, relocate, stage, purge, etc. They can also involve events that are associated with a single dataset or a collection such as extract metadata, resize or change resolution of an image, transform a document (eg., Word to pdf), snapshot a table, etc. They can also involve events in data grid administration such as adding, and removing users, modifying user profile, creating a new resource, removing an existing resource, migrating files from one resource to another, backing up a resource, etc. . In a full-fledged data grid system such as the SRB there were more than one hundred such operational events that were identified. They were initiated by specific commands that a user or administrator could execute in the SRB system.

Micro-Services and iRODS Rules

Micro-services are the building blocks of the iRODS system. They can be viewed as defining the Action part in an ECA-type rule. As described above, micro-services communicate with each other and with the process environment using four different communication models. The semantics of a micro-service can be captured based on the change-semantics that it encodes. The semantics of micro-services are based on three types of properties that hold before and after the execution of a micro-service:

1) A micro-service can modify the value of variables and structures in the white board it shares in the execution environment with other micro-services.
2) A micro-services can insert, delete or update the values in the tables in the iCAT database.
3) A micro-service can have side-effects outside the iRODS system (eg. creation of a new file in a file system).

The white board is shared only during that particular rule execution. Note that a rule can fire other rules in a hierarchical fashion. The white board can be shared across two sequential rule invocations provided the data grid "uses" the same white-board for the second rule. The white board can be packed and passed to remote servers for use in their micro-service execution. Serializability of the hierarchical execution of micro-services is an important criteria needed for establishing the semantics of the distributed execution environment. We denote the status of the white board using the symbol $ (dollar).

The iCAT is a persistent database, hence any changes made to it are seen not only by other micro-services in the current rule execution, but also by other rules excuted locally or remotely, and by rules and micro-services executed at a later time period. We rely on the relational

database transactional properties for maintaining the stability and recoverability of the iRODS operations. We denote the status of the iCAT database using the symbol % (percent).

The side-effects that result from a micro-service execution can be recoverable for the most part. A newly created file can be deleted, a change made to an external database can be deleted or rolled back, a change made to a file can be recovered when replicas of the file are available. A message sent through the iRODS messaging system can be rolled back by sending an appropriate counter message that rolls back the operation initiated by the original message. But there are some side-effects that are not recoverable. For example, if a micro-service sends an email to a user, even though it is possible to send another email to ignore the previous email, the recoverability is not as clean as one would want. Other side-effects such as remove the only copy of a file may not be recoverable at all. We denote the side effects using the symbol # (hash sign).

The side effects are defined by an ontology which we are developing to succinctly describe the changes that are made. Some of the elements of this ontology is shown below:

create_new_file(filename, pathname, resource).
send_email (from, to, subject, body).
replicate_file(filename, newpath, newresource).
compute_checksum(filename, checksum).

The side effects elements are similar to the atoms in first-order logic. In order to provide transactional capabilities, we have added a few system-centric elements:

begin_side_effect_transaction
end_side_effect_transaction
commit_side_effect_transaction
rollback_side_effect_transaction

These are similar to that used in relational databases to help in defining concurrent transactions and their semantics.

The semantics of a micro-service is defined by a change semantics:

MS: {$, %, #} micro-service {$',%', #'},

where $, % and # denote the semantic status before the execution of the micro-service and $', %' and #' denote the status after the execution. Note that by capturing the change semantics as such, we define the semantics of the micro-service independent of the communications that happens through the micro-service. The semantics of a rule can be easily established as a composition of the semantics of the micro-services. But, as mentioned earlier, the white-board is pertinent only for actions within a rule invocation and hence the semantics of a rule is defined only in terms of the iCAT database changes and the side effects ignoring the change made to the white board.

RS: {%, #} rule {%', #'}

The Event semantics of an event is defined by the rule semantics of the rules. Since only one rule is applicable as per the policy of the iRODS system, one can define the event semantics as follows:

ES: {%, #} Event {%', #'}, where

$ES = RS_1 + RS_2 + \ldots + RS_n$

RSi is the semantic of the ith rule which is defined for the Event. Notationally "+" denotes the exclusive-OR of the semantics but operationally it is applied left-to-right. The disjunctive nature of the above equation makes it quite complex to study the consistency of such a system and is an area of research. Applying logic programming methods [Lloyd 19877] and stable model semantics [Gelfond 1988] for disjunctive logic programs one can study such a semantics. We also build upon some of the work done in reasoning with policy description language [Lobo 1999, Son 2001]. We do not explore this further in this paper.

Conclusion

Traditionally data management systems are not viewed as an event processing system. In our experience with the implementation and deployment of the Storage Resource Broker (SRB) and its use in large-scale production environment for more than 10 years in multiple disciplines (neuroscience, astronomy, seismology, cosmology, real-time sensor systems, and high-energy physics), we have started to characterize the complex event handling that is being done during the life-cycle management of the data. This has led us to the design and development of the integrated Rule-Oriented Data Systems (iRODS) that captures the policies needed for handling such events. Moreover, our diverse user community has their own requirements for data management. This led us to realize that a declarative rule-oriented environment that not only allows explicit declaration of policies for actions to be performed but also provides "programmability" for the user communities is a necessity.

In this paper, we have put forth the idea that data life-cycle management can be viewed as complex and concurrent event processing. With iRODS, we have shown that such a system is viable and useful.

This work is supported by the National Science Foundation SDCI grant (OCI-0721400) and a grant from the National

Archives and Records Administration / National Science Foundation (OCI-0848296).

References

Chervenak, A., I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets**,** in Journal of Network and Computer Applications, 23:187-200, 2001.

Dayal U., Active Database Systems: Triggers and Rules for Advanced Database Processing, Morgan Kaufmann Publishers Inc. 1994.

DICE: Data Intensive Cyber Environments Group. http://www.diceresearch.org/DICE_Site/Home/Home.html

Gelfond, M., and V. Lifschitz, The stable model semantics for logic programming. In: Proceedings of the Fifth International Conference on Logic Programming.

IRODS: integrated Rule Oriented Data System, https://www.irods.org/index.php.

Lloyd, J.W., Foundations of Logic Programming (2nd edition). Springer-Verlag 1987.

Lobo, J., R. Bhatia, and S. Naqvi, A PolicyDescription Language, Proc. AAAI, 1999.

Moore,R.,S-Y. Chen, W. Schroeder, A. Rajasekar, M. Wan, and A. Jagatheesan, Production Storage Resource Broker Data Grids, 2nd International Conf. e-Science and Grid Computing, Amsterdam, Netherlands, 2006.

Moore, R.W., A. Rajasekar, Rule-BasedDistributed Data Management, Grid 2007: IEEE/ACM International Conference on Grid Computing, 2007.

Rajasekar A., et.al., Storage Resource Broker - Managing Distributed Data in a Grid. Computer Society of India Journal, Special Issue on SAN, Vol. 33, No. 4, pp. 42-54 Oct 2003.

Rajasekar, A., M. Wan, R. Moore, W. Schroeder, A Prototype Rule-based Distributed Data Management System, HPDC workshop on "Next Generation Distributed Data Management," Paris, France, 2006.

Son, T.C., J. Lobo, Reasoning about Policies using Logic Programs, Proc. AAAI 2001 Spring Symposium on Answer Set Programming, 2001.

SRB: Storage Resource Broker, http://srb.diceresearch.org

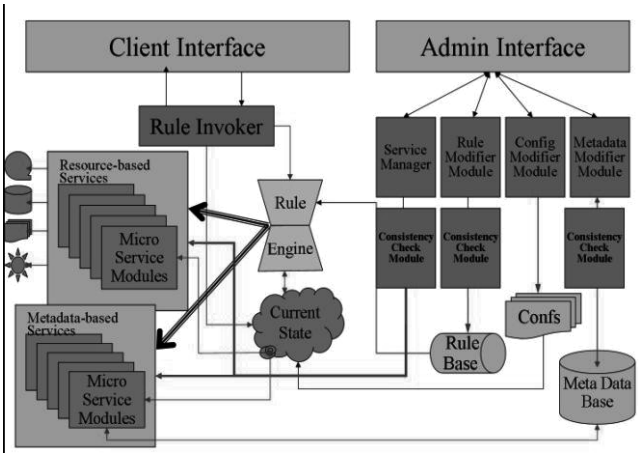

Figure 1: iRODS Architecture

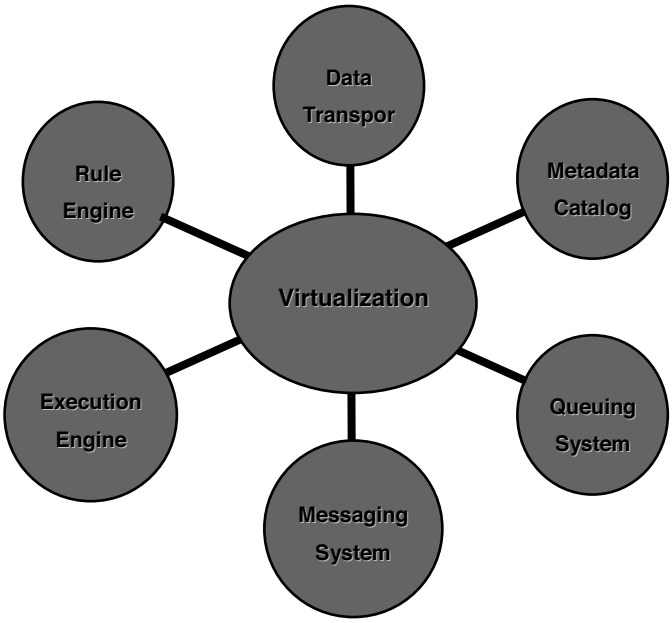

Figure 2: iRODS Services