Framework for Querying Distributed Objects Managed by a Grid Infrastructure*

Ruslan Fomkin and Tore Risch

Department of Information Technology, Uppsala University,
P.O. Box 337, SE-751 05 Uppsala, Sweden
{Ruslan.Fomkin, Tore.Risch}@it.uu.se

Abstract. Queries over scientific data often imply expensive analyses of data requiring a lot of computational resources available in Grids. We are developing a customizable query processor built on top of an established Grid infrastructure, the NorduGrid middleware, and have implemented a framework for managing long running queries in Grid environment. With the framework the user does not specify the detailed job and parallelization descriptions required by NorduGrid. Instead s/he specifies queries in terms of an application-oriented schema describing contents of files managed by the Grid and accessed through wrappers. When a query is received by the system it generates NorduGrid job descriptions submitted to NorduGrid for execution. The framework considers limitations of NorduGrid. It includes a submission mechanism, a job babysitter, and a generic data exchange mechanism. The submission mechanism generates a number of jobs for parallel execution of a user query over wrapped data files. The task of the babysitter is to submit generated jobs to NorduGrid for the execution, to monitor their execution status, and to download results from the execution. The generic exchange mechanism provides a way to exchange objects through files between Grid execution nodes and user applications.

1 Introduction

Nowadays a lot of scientific data are stored in Grids. Scientists need to access and analyze them. Their analyses often imply expensive computations that need to process a lot of data. Thus scientists need to use external computational resources to process their analyses, and storage resources to store and share huge amounts of data. For this many Grids are developed to provide computational resources and storage facilities.

For example, the ATLAS collaboration [1] motivates many Grid projects such as LCG [2], EGEE [3], and NorduGrid [4]. These projects provide storage facilities to store and share data produced by ATLAS [1] and to be produced by the Large Hadron Collider (LHC) [5], along with computational resources to analyze the data.

* This work is funded by The Swedish Research Council (VR) under contract 343-2003-955.

J.-M. Pierson (Ed.): VLDB DMG 2005, LNCS 3836, pp. 58–70, 2005.

A typical analysis of data for the LHC projects is selections of subsets of the input data. The selections, called *cuts*, consist of not only simple logical predicates but also numerical computations. We show that such analyses can be expressed in a declarative way using an extensible query language.

We are developing *POQSEC* [6] (*Parallel Object Query System for Expensive Computations*) that processes scientific analyses specified as declarative SQL-like queries over data distributed in the Grid. It utilizes computational resources of Swegrid [7] and storage resources of Nordic countries through the middleware Grid infrastructure NorduGrid [4]. The goal of the POQSEC project is to provide a transparent and scalable way to specify and execute scientific queries. A user should be able to specify his/her query transparently in a client database without respect to where it will be executed and how data will be accessed.

Currently we have implemented a framework for submitting user queries for execution in the Grid. The system then creates jobs executing the queries, submits the jobs to NorduGrid, monitors execution of the jobs by NorduGrid, downloads results of the jobs, and delivers results of the queries to the user. The user states queries to POQSEC in terms of a database schema available in the client database. The schema contains both an application-oriented part and Grid meta-data. The application schema describes data stored inside files in Grid storage resources, for example events produced by ATLAS. Wrappers are defined for accessing the contents of these files, e.g. in our application we use a wrapper of the ROOT library [8]. The Grid meta-data contains information about the files. Thus user queries can restrict data both in terms of application data contents and meta-data about files. The latter is very important since there is a huge amount of Grid data files and queries are normally over a small percentage of them. User queries are parallelized to a number of jobs for execution. The parallelization is done by partitioning data between jobs. Our preliminary results show that the parallelization gives significant performance improvements.

The rest of the paper is organized as follows. Related work is discussed in Sect. 2. Section 3 describes the POQSEC architecture. It is followed by a description of an application from High Energy Physics, which is our test case. The implementation of the framework is discussed in Sect. 5, and Sect. 6 concludes the paper.

2 Related Work

Another system that utilizes a Grid infrastructure and provides high-level declarative query language for data access and analysis is Distributed Query Processing system (DQP) [9] or its web service version OGSA-DQP [10]. The DQP is part of the Grid infrastructure myGrid [11], which fully controls resources and where resources can be allocated dynamically. The resources for the query execution are allocated and provided by a user. Any of them can be utilized by DQP dynamically. It is different from our system where NorduGrid is a middleware above autonomous local batch systems that control computational resources. Unlike the DQP, we need to consider the NorduGrid limitation that jobs are not

guaranteed to start immediately. Furthermore, as part of a job description NorduGrid requires to specify descriptions of resources in advance. This includes, for example, estimating execution time and number of computational nodes for jobs.

STORM [12] is a distributed query processing environment for processing selections over distributed large scientific datasets and transferring the selected data to its clients. STORM does not leverage an existing Grid infrastructure for data transportation, job scheduling, and batch query processing as POQSEC.

In [13,14] a batch database system is developed to support scientific queries. It is there applied on astronomical data. The data are stored in back-end SQL servers managed by a front-end batch query system. In POQSEC we use a middleware approach to access wrapped data stored in native format rather than storing the data in SQL databases.

ATLAS Distributed Analysis (ADA) [15] project has goal to provide high-level interface for scientists who analyze data produced by ATLAS and LHC. The users specify jobs containing datasets for processing in terms of meta-data and their analyses as snippets of programming code, for example in C++ or Fortran. A typical analysis performs selections that include computations over input datasets and aggregations over results of the selections. The jobs are submitted to Grid resources and their execution is monitored. In contrast, POQSEC uses a declarative high-level query language to specify analyses and the goal of POQSEC is transparent execution without considering whether the query will be executed on Grid resources or locally.

3 POQSEC Architecture

The architecture presented in Fig. 1 illustrates the current implementation of POQSEC. The POQSEC architecture considers limitations of the NorduGrid (NG) middleware. NorduGrid and its limitations are briefly described in Sect. 3.1. Section 3.2 describes POQSEC components and its interaction with NorduGrid.

3.1 NorduGrid Middleware

NorduGrid (also called Advance Resource Connector) [4,16] is a middleware between Grid users and computational resources that are managed by local batch systems. Thus NorduGrid does not control computational resources; instead it submits user tasks to local batch systems on clusters and each local batch system allocates cluster nodes according to its policy and current load of the cluster.

The *Computing Elements* (CE) are clusters where Grid jobs are executed while *Storage Elements* (SE) are file servers where the data to be queried are stored. The CEs and SEs are managed by NorduGrid and are accessible by submitting Grid jobs to an *NG Client*. The NG client is a set of command line tools to submit, monitor, and manage jobs on the Grid. It also has commands to move data between storage elements and clients, and to query Grid resource information such as loads on different CEs and job statistics. Users of NorduGrid always first initiate communication with the NG client.

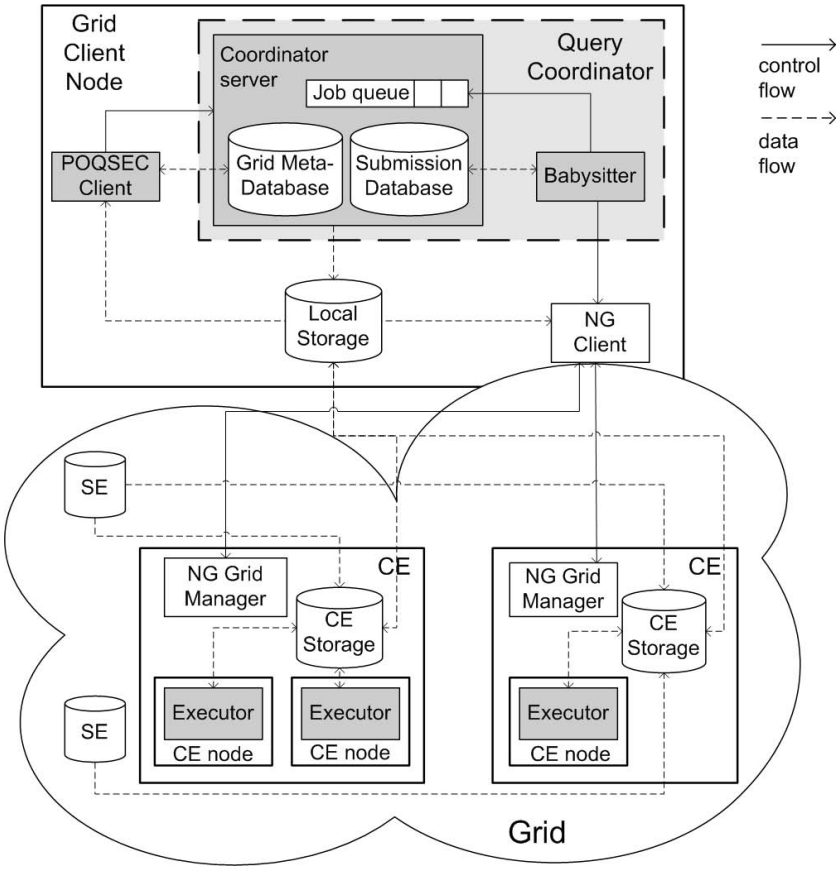

Fig. 1. Architecture of current implementation of POQSEC

The NG client includes a resource brokering service [17] to find suitable resources for jobs. Jobs are described in a resource specification language, *xRSL* [18], which includes specification of, e.g.:

- A user executable and its arguments to be run on some suitable computing element.
- Files to be transported to and from the chosen computing element before and after the execution.
- Maximal CPU time for the execution.
- *Runtime environments* for the execution. A runtime environment is an additional software package required, e.g. an application library such as ROOT [8].
- Standard input, output, and error files for the execution.
- Optional names of the computing elements where the executable can run.
- The number of parallel sub-jobs to be run on the computing element.

In summary NorduGrid requires a lot of user specifications to fully describe computation tasks as xRSL scripts. An example of the script is shown in Fig. 4. POQSEC simplifies this considerably by automatically generating NorduGrid interactions and job scripts to execute a task specified as a declarative query of contents of data. To manage jobs generated by POQSEC, to track their executions, and to download results we provide a *babysitter* integrated with the POQSEC framework.

3.2 POQSEC Components and Their Interaction with NorduGrid

The *Query Coordinator* of POQSEC (Fig. 1) manages user queries submitted to POQSEC for execution on the Grid. It communicates with an NG client directly through a command line interface. Both the query coordinator and the NG client are running on the same node, the *Grid Client Node*, which is a user accessible computer node. On it the user must first initialize his/her Grid credentials required for using NG client services according to the *Grid Secure Infrastructure* (GSI) [19] mechanism.

The *POQSEC Client* component is a personal POQSEC database running on the Grid client node and communicating with the query coordinator. It could also run on a separate node from the Grid client node, e.g. on a user's desktop computer, if GSI is used for the communication with the query coordinator. Queries are submitted through the POQSEC client to the query coordinator for further execution on Grid resources.

The components of the query coordinator are the *Coordinator Server* and the *Babysitter*. The coordinator server contains a *Grid Meta-Database*, a *Submission Database*, and a *Job Queue*. The Grid meta-database stores information about data files and computational elements accessible trough POQSEC. It is needed since Grid resources are heterogeneous and require Grid users to know the computational elements that are able to execute their jobs and properties of the computational elements required for job executions, e.g. runtime environments. POQSEC users need not specify this information when submitting queries since it is stored in the Grid meta-database.

The submission database contains descriptions of queries submitted from the POQSEC client and job descriptions generated by POQSEC to execute the queries. The job queue contains jobs that are created but not yet submitted to NorduGrid for execution.

The process of submitting and evaluating a query is presented in Fig. 2. When a query is received (1) from the POQSEC client the coordinator server first registers the query in the submission database and stores there a number of job descriptions to parallelize the query execution. The number of jobs to create is currently provided by the user as part of the query submission[1]. Information about computational resources and data files from the Grid meta-database is used to generate these job descriptions. xRSL scripts are generated from the job descriptions and are stored (2) in the local storage. Then the jobs are registered

[1] We are working on automating this.

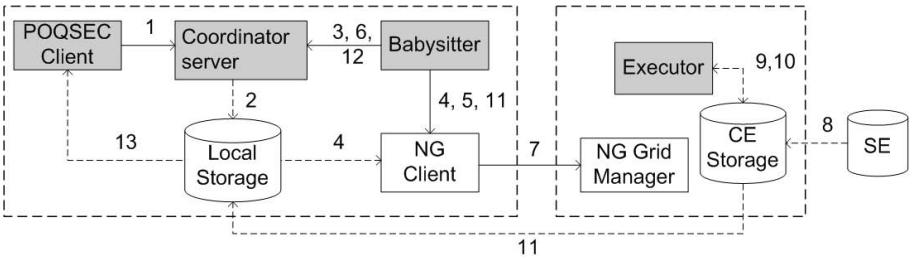

Fig. 2. Interactions between POQSEC components and NorduGrid

in the job queue. The babysitter picks (3) jobs from the job queue and submits (4) them as xRSL scripts to the NG client for execution on Grid resources. Once a job has been submitted the babysitter regularly polls (5) the NG client for its job status and reports (6) the status to the coordinator server to update the submission database. When a job is finished the babysitter downloads (11) the result to the *Local Storage*, which is the file system of the Grid client node, and notifies (12) the coordinator server. The result can be retrieved (13) to the POQSEC client after the query is finished.

On each CE NorduGrid maintains an *NG Grid Manager*. It receives (7) job descriptions from NG clients. In our case these jobs are executing POQSEC subqueries. The NG Grid manager uploads (8) input files from SEs to the local *CE Storage* before each job is submitted to the local batch system. The local batch system allocates *CE nodes* for each job according its policies and current load, and then starts the job executions. For POQSEC these jobs contain *Executors* that evaluate (9) subqueries over uploaded data and store (10) the results in local CE storage files. The babysitter polls (5) the NG client regularly for finished executions. After a job has finished the babysitter requests (11) the NG client to download (11) the result to the local storage of the Grid client node and notifies (12) the coordinator server that the job is ready. Since a given POQSEC query often generates many jobs a query is ready only when all its jobs are finished. However, partial results can be obtained once some jobs are finished.

4 User Application

Our current test application is an application for analyzing data produced by LHC projects for containing charged Higgs bosons [20].

Input data for the analyses are *events*, which describe collision events between elementary particles. Each event comprises sets of particles of various types such as *electrons*, *muons*, sets of other particles called *jets*, and sets of event parameters such as missing momentum in x and y directions (*PxMiss* and *PyMiss*). Each particle is described by its own set of parameters, e.g., the ID-number of the type of a particle (*Kf*), momentum in x, y, and z directions (*Px*, *Py*, and *Pz*), and amount of energy (*Ee*). The data are stored in files managed by an object-oriented data analysis framework, *ROOT* [8].

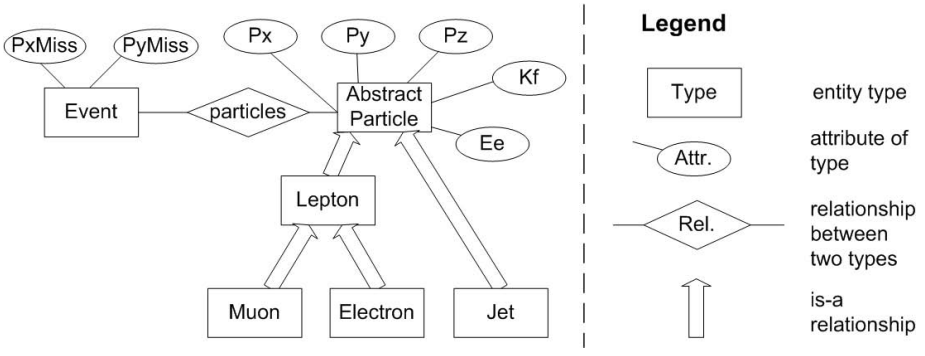

Fig. 3. The schema of the application data

Analysis of events consists of selecting those events that can potentially contain the charged Higgs bosons. A number of predicates, called *cuts*, are applied to each event and if the event satisfies all of them it is selected. A cut is a selection of events of interest for further analysis according to a scientist's theory.

The application is implemented as an extension of a functional and object-oriented mediator system Amos II [21]. It is called *ALEH* (Analysis LHC Events for containing charged Higgs bosons). ALEH has a ROOT wrapper to access data from files managed by ROOT. An object-relational schema of the event data is defined in Fig. 3. It is a view of relevant parts of ROOT files.

A number of analysis queries implementing cuts are defined as *derived functions* expressed in a query language, *AmosQL* [22]. Often a researcher selects events satisfying several cuts. For example, such query in AmosQL is:

```
SELECT ev
FROM Event ev
WHERE jetvetocut(ev) AND zvetocut(ev) AND
    topcut(ev) AND misseecuts(ev) AND
    leptoncuts(ev) AND threeleptoncut(ev);
```

The query is expressed in terms of derived functions, which define the cuts. The definition of one of the cuts in AmosQL is:

```
CREATE FUNCTION zvetocut (Event ev) -> Event AS
SELECT ev
WHERE NOTANY(oppositeleptons(ev)) OR
    (abs(invMass(oppositeLeptons(ev)) - zMass) >= minZMass)
```

where `invMass` calculates the invariant mass of a pair of two given leptons, `zMass` is the mass of a Z particle, `minZMass` is range of closeness, and `oppositeLeptons` is a derived function defined as another query:

```
CREATE FUNCTION oppositeLeptons (Event ev) -> <Lepton, Lepton> AS
SELECT l1, l2
FROM Lepton l1, Lepton l2
```

```
WHERE l1 = particles(ev) AND l2 = particles(ev) AND
      Kf(l1) = -Kf(l2);
```

5 Implementation

A POQSEC client running our test application ALEH has an interface to a coordinator server through which a user can submit queries for execution in the Grid. It can monitor the status of submitted queries, and can retrieve results of finished queries. To submit a query the user invokes a system interface function named `submit` and specifies there the query defined in terms of the application schema, set of file names which should be processed by the query, number of jobs for parallelization the query, CPU time required for processing one job, and optionally a computing element where the query's jobs should be executed. If no computing element is specified the jobs will be submitted to an NG client along with a list of possible computing elements for execution. The result of the submit function is an object used to monitor the status and to retrieve the result.

The test data, which are events, are produced by ATLAS simulation software and stored on storage recourses accessible through NorduGrid. Paths to the data files are stored in the Grid meta-database of the coordinator server in a format according to xRSL specification [18]. Thus the user provides file names without paths during submission.

For example, the user wants to execute the general analyzing query presented in Sect. 4 over eight specific files containing equal number of events, with parallelization in four jobs that each job will process two files, where the CPU time of executing the query over the two files is 20 minutes, on any of available computational resources of Swegrid. The user submits the query and assigns the result of the submission to a variable `:s`:

```
SET :s = submit("SELECT ev FROM Event ev WHERE jetvetocut(ev) AND
zvetocut(ev) AND topcut(ev) AND misseecuts(ev) AND leptoncuts(ev)
AND threeleptoncut(ev)",{"bkg2Events_ruslan_000.root",
"bkg2Events_ruslan_001.root","bkg2Events_ruslan_002.root",
"bkg2Events_ruslan_003.root","bkg2Events_ruslan_004.root",
"bkg2Events_ruslan_005.root","bkg2Events_ruslan_006.root",
"bkg2Events_ruslan_007.root"},4,20);
```

The submission is then translated into four xRSL scripts, which are submitted to a NG client for execution. One of the scripts is presented in Fig. 4. The executable there is the ALEH application, which contains the wrapper of ROOT files.

It is necessary for the user to specify which files to analyze to restrict amount of data for processing. In the example the user specifies file names explicitly. Alternatively the user can define a query over the meta-database of the coordinator server to retrieve the file names. The local batch systems of all computational elements available through NorduGrid require specification of CPU time and thus the user needs to provide this[2].

[2] We are working to estimate this automatically.

```
& (executable=aleh)
(arguments="aleh.dmp")
(inputfiles= (aleh "/home/udbl/ruslan/Amox/bin/aleh")
  (aleh.dmp "/home/udbl/ruslan/Amox/bin/aleh.dmp")
  (query2005420103329443.osql "query2005420103329443.osql")
  (bkg2Events_ruslan_001.root "gsiftp://se1.hpc2n.umu.se:2811/
se3/ruslan_poqsec/bkg2Events_ruslan_001.root")
  (bkg2Events_ruslan_000.root "gsiftp://se1.hpc2n.umu.se:2811/
se3/ruslan_poqsec/bkg2Events_ruslan_000.root"))
(outputfiles=(result.out ""))
(cputime=20)
(| (runtimeenvironment=ROOT-3.10.02)
  (runtimeenvironment=APPS/HEP/ATLAS-8.0.8)
  (runtimeenvironment=APPS/PHYSICS/HEP/ROOT-3.10.02)
  (runtimeenvironment=ATLAS-8.0.8)
  (runtimeenvironment=APPS/HEP/ATLAS-9.0.3))
(stdin="query2005420103329443.osql")
(stdout="outGen.out")
(stderr="errGen.err")
(gmlog="grid.debug")
(middleware>="nordugrid")
(| (cluster=sg-access.pdc.kth.se) (cluster=bluesmoke.nsc.liu.se)
  (cluster=hagrid.it.uu.se) (cluster=hive.unicc.chalmers.se)
  (cluster=ingrid.hpc2n.umu.se) (cluster=sigrid.lunarc.lu.se))
(jobName="POQSEC: swegrid2005420103329444.xrsl")
```

Fig. 4. Example of the xRSL file with name swegrid2005420103329444.xrsl

The performance of many queries can be significantly improved by parallelization into several jobs. Our experience shows that parallelization of executing a query gives dramatic improvements. For example, the above submission took 24 minutes. The time was calculated as the elapsed time between when the query was submitted until all job results were downloaded from the Grid. A submission of the same query without parallelization as one job took 3 hours and 45 minutes, where 3 hours and 10 minutes were spent for the query evaluation. It is much longer response time compared with the parallelized Grid execution.

During execution of a query submitted to POQSEC the user can monitor its status by calling `status(:s)`. The status of the query is computed from its batch jobs statuses. The status "DOWNLOADED" will be returned only if results of all jobs of the query were downloaded. Then the user can retrieve the result data by executing `retrieve(:s)`. The result of the query can be retrieved also by using the function `wait(:s)`. The difference is that if `wait` is invoked before the result of the jobs is available the system waits until the coordinator server notifies it that all jobs are downloaded. Then it retrieves the result while `retrieve` will just print a message if the query is not finished[3]. The user can cancel his/her query submission by executing `cancel(:s)`.

[3] We are implementing functions to retrieve partial results.

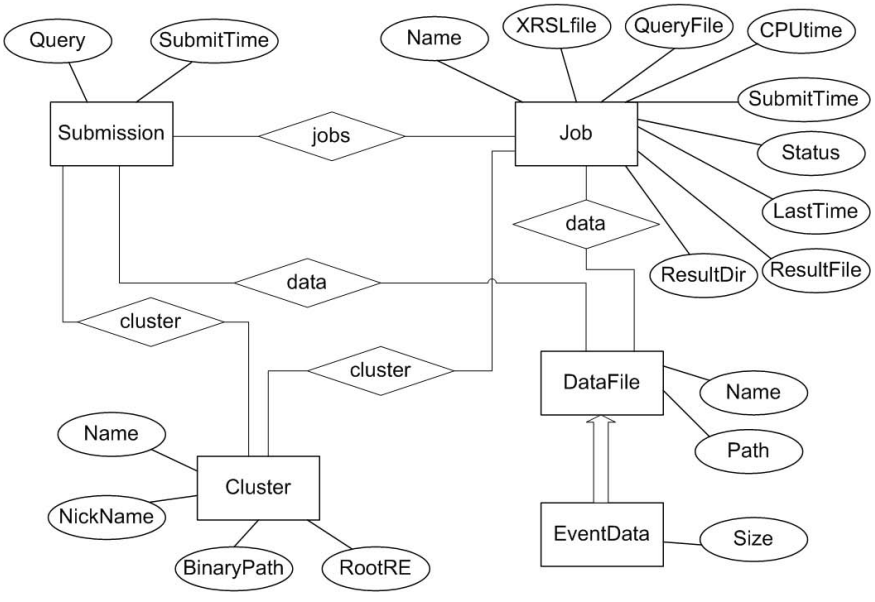

Fig. 5. Schema of the Grid meta-database and the submission database

The coordinator server, the babysitter, and the NG client are running on the same Grid client node as the POQSEC client. The coordinator server contains the Grid meta-database and the submission database. The user is able to query the coordinator server for data from the Grid meta-database and to request updates of the Grid meta-database through the POQSEC client. The babysitter polls the coordinator server to pick up jobs from the job queue and to request updates of the submission database.

A schema of the Grid meta-database and the submission database is presented in Fig. 5. The types `Cluster` and `DataFile` and its subtype `EventData` are parts of the Grid meta-database. The submission database is presented by the types `Submission` and `Job`.

When the coordinator server receives query submissions from the POQSEC client it generates job descriptions and creates xRSL files for NorduGrid and script files for POQSEC executors. For example, for the submission given above the coordinator server generates four xRSL files and four script files. Example of one of the xRSL file is given in Fig. 4. The POQSEC script files contain commands for executors to load the input data from the data files through the ROOT wrapper and to execute the user query. In our example one of the script files contains:

```
load_root_file("bkg2Events_ruslan_001.root");
load_root_file("bkg2Events_ruslan_000.root");
save("result.out",SELECT ev FROM Event ev WHERE jetvetocut(ev)
  AND zvetocut(ev) AND topcut(ev) AND misseecuts(ev) AND
  leptoncuts(ev) AND threeleptoncut(ev));
```

The results of the query executions are saved by the executors in files (here in `result.out`) in a way that they can be read by the POQSEC client. Objects, in our case events, which originally were the same, will be treated by the POQSEC client as the same object regardless of that they came from different sources.

The other three xRSL files and three script files are similar except that they have different input data files. Automatic generation of the files by POQSEC exempts the user from manually creating such files for each job.

The main purposes of the babysitter is to interact with the NG client to submit jobs, to monitor status of executing jobs, and to download finished jobs. Each interaction with the NG client can take from several seconds to a minute; thus the coordinator server does not contact the babysitter immediately when a job is created. Instead the babysitter polls the coordinator server regularly when it is not interacting with the NG client.

6 Conclusion and Future Work

We have implemented a framework which provides basic tools for executing long running batch queries on Grid resources over wrapped scientific data distributed in the Grid. The framework is a part of our development of POQSEC (Parallel Object Query System for Expensive Computations), the goal of which is to provide a fully transparent query execution system for scientific applications.

Our on-going work is to automate estimates of maximal CPU time required for the execution of an arbitrary query on a partition of input data. The estimates will be based on probing the query on a number of small samples. We also investigate strategies for suspending those jobs for which the maximal CPU time were underestimated, and then resuming them on other resources.

Another on-going work considers automatic parallelization of a user query submitted for execution on the Grid resources. To decide automatically how many jobs to parallelize the query into depends on the current load of computational resources of the Grid and which computational elements would be chosen for the execution of the generated jobs. We combine this together with development of our own resource brokering algorithm. The resource brokering algorithm will not just decide where to execute the query but also to how many jobs parallelize the query. It should take in account that different computing elements of the Grid have different policies. We will base our resource broker algorithm on job statistics available from the NorduGrid middleware [23].

Job execution on computational resources accessible through NorduGrid can fail and users of NorduGrid need to deal with failures. We are investigating various strategies to deal with failures of job executions.

References

1. ATLAS collaboration. http://atlas.web.cern.ch/Atlas/internal/Welcome.html
2. LHC Computing Grid. http://lcg.web.cern.ch/lcg/
3. EGEE: Enabling Grids for E-sciencE.
   http://egee-intranet.web.cern.ch/egee-intranet/gateway.html

4. Eerola, P., Ekelöf, T., Ellert, M., Hansen, J.R., Konstantinov, A., Kónya, B., Nielsen, J.L., Ould-Saada, F., Smirnova, O., Wäänänen, A.: Science on NorduGrid. In Neittaanmäki, P., Rossi, T., Korotov, S., Oñate, E., Périaux, J., Knörzer, D., eds.: ECCOMAS 2004. (2004) See also http://www.nordugrid.org.
5. LHC - the Large Hadron Collider. http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/
6. Fomkin, R., Risch, T.: Managing long running queries in Grid environment. In Meersman, R., Tari, Z., Corsaro, A., eds.: OTM Workshops. LNCS 3292, Springer (2004) 99–110
7. Swegrid. http://www.swegrid.se
8. Brun, R., Rademakers, F.: ROOT - an object oriented data analysis framework. In: AIHENP'96 Workshop. Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81–86 See also http://root.cern.ch.
9. Smith, J., Gounaris, A., Watson, P., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Distributed query processing on the Grid. In Parashar, M., ed.: GRID. LNCS 2536, Springer (2002) 279–290
10. Alpdemir, M.N., Mukherjee, A., Gounaris, A., Paton, N.W., Watson, P., Fernandes, A.A.A., Fitzgerald, D.J.: OGSA-DQP: A service for distributed querying on the Grid. In Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E., eds.: EDBT. LNCS 2992, Springer (2004) 858–861
11. myGrid. http://www.mygrid.org.uk
12. Narayanan, S., Kurç, T.M., Çatalyürek, Ü.V., Saltz, J.H.: Database support for data-driven scientific applications in the grid. Parallel Processing Letters **13** (2003) 245–271. See also http://storm.bmi.ohio-state.edu.
13. Nieto-Santisteban, M.A., Gray, J., Szalay, A.S., Annis, J., Thakar, A.R., O'Mullane, W.: When database systems meet the Grid. In: CIDR. (2005) 154–161
14. O'Mullane, W., Li, N., Nieto-Santisteban, M.A., Szalay, A.S., Thakar, A.R., Gray, J.: Batch is back: CasJobs, serving multi-TB data on the Web. Technical Report MSR-TR-2005-19, Microsoft Research (2005)
15. Adams, D., Deng, W., Chetan, N., Kannan, C., Sambamurthy, V., Harrison, K., Tan, C., Soroko, A., Liko, D., Orellana, F., Branco, M., Haeberli, C., Albrand, S., Fulachier, J., Lozano, J., Fassi, F., Rybkine, G.: ATLAS distributed analysis. In: CHEP04. (2004)
16. The NorduGrid/ARC User Guide. (2005) Available at http://www.nordugrid.org/documents/userguide.pdf.
17. Ellert, M.: The NorduGrid brokering algorithm (2004) Available at http://www.nordugrid.org/documents/brokering.pdf.
18. Smirnova, O.: Extended Resource Specification Language Reference Manual. (2005) Available at http://www.nordugrid.org/documents/xrsl.pdf.
19. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S.: Security for Grid services. In: HPDC'03, IEEE Computer Society (2003) 48–57. See also http://www-unix.globus.org/toolkit/docs/3.2/gsi/.
20. Hansen, C., Gollub, N., Assmagan, K., Ekelöf, T.: Discovery potential for a charged Higgs boson decaying in the chargino-neutralino channel of the ATLAS detector at the LHC. SN-ATLAS-2005-050 (2005)
21. Risch, T., Josifovski, V., Katchaounov, T.: Functional data integration in a distributed mediator system. In: The Functional Approach to Data Management: Modeling, Analyzing, and Integrating Heterogeneous Data. SpringerVerlag (2003)

22. Flodin, S., Hansson, M., Josifovski, V., Katchaounov, T., Risch, T., Skold, M.: Amos II Release 7 User's Manual. Uppsala Database Laboratory. (2005) Available at http://user.it.uu.se/~udbl/amos/doc/amos_users_guide.html.
23. Konstantinov, A.: The Logger Service, Functionality Description and Installation Manual. (2005) Available at http://www.nordugrid.org/documents/Logger.pdf.