Published in *Proc. 1st Intl. Workshop on GRID Computing and its Applications (GADA'04)*, Lacarna, Cyprus, Oct. 2004.

Managing Long Running Queries in Grid Environment

Ruslan Fomkin and Tore Risch

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

Abstract. Exceptionally large amounts of both distributed data and computational resources are becoming available through the Grid. This will enable efficient exchange and processing of very large amounts of data combined with CPU intensive computations, as required by many scientific applications. We propose a customizable Grid-based query processor built on top of an established Grid infrastructure, NorduGrid. It allows users to submit queries wrapping user-defined long running operations that filter and transform distributed customized data. Limitations imposed by the used Grid infrastructure influence the architecture. For example, resource requirements have to be specified before Grid jobs are started and delays may occur based on the availability of required resources for a job. We are developing a fully functional prototype to investigate the viability of the approach and its applicability. Our first application area is Particle Physics where scientists analyze huge amount of data produced by a collider or simulators to identify particles.

1 Introduction

The Grid initiative provides an infrastructure for transparent distributed computations among widely distributed computer clusters. Exceptionally large amounts of both distributed data and computational resources are becoming available through Grid infrastructures. Many scientific applications within, e.g. bioinformatics, neuroscience, and space physics require large, scalable, high-performing, CPU demanding, and memory-intensive computations over large amounts of distributed data. They also require representation of not only the traditional tabular data managed by relational database management systems but also representation, modeling, and querying of numerical data, e.g. for vector and matrix algebra.

We are developing a new kind of data manager and query processor, Parallel Object Query System for Expensive Computations (POQSEC) that utilizes the operational Grid infrastructure NorduGrid [41] and leverages the Globus I/O library [17] from the Globus Toolkit [14] for processing declarative queries that access data from storage elements and wrapped external systems on the Grid. POQSEC has support for customizable data representations, allows user-defined long-running distributed computations in queries, and can access conventional relational databases. It processes application specific code on both local data and data distributed through the Grid.

The aim of this system is to achieve high query performance by utilizing large main memories, disk, CPU power and other resources on many distributed nodes accessed through the NorduGrid, which is a distributed peer oriented Grid middleware system that does not rely on a central broker. Computer clusters accessed through NorduGrid have certain restriction with respect to resource allocation, communication, and process management that the POQSEC architecture must cope with and this influences its architecture.

The POQSEC data manager and query processor scales up by utilizing the Grid to transparently and dynamically incorporate new nodes and clusters for the combined processing of data and computations as the database and application demands grow. Conventional databases and file-based Grid storage elements are used as back-ends for data repositories. Extensible and object-oriented query processing and rewrite techniques are used to efficiently combine distributed data and computations in this environment.

Each node in POQSEC has DBMS facilities like storage management, customizable query processing, along with the possibility to call CPU demanding and long running user computations from queries. POQSEC uses an extensible and distributed query engine allowing user-defined data representations and query optimization algorithms that consider calling user-defined CPU intensive query functions. Query transformation rules transparently distribute customized data and their representations to nodes in the clusters. A given user query is rewritten into many distributed execution subplans given knowledge about available resources and observed

execution costs. POQSEC nodes are automatically started or closed when needed based on resource demands for the current query workload.

The POQSEC prototype is being developed using as test case data and queries from Particle Physics where large amounts of data describing particles of events are produced by proton-proton collisions. The data is currently produced by simulations at NorduGrid [11] and will be produced by detectors of the Large Hadron Collider (LHC) [26] from 2006. The amount of data produced is huge (10 million events = 20-30 TB) and produced at a very high rate (10 million events over 2-3 weeks) [1]. The data therefore needs to be stored as distributed files in storage elements accessible though the Grid. Many scientists are querying these huge distributed data sets to identify interesting particles. The queries involve regular data comparisons and aggregation operators along with user defined filter operations in terms of C++ based libraries such as the ROOT library [7]. A single such analysis of a single dataset of size 1 million events often takes more than 1 hour to execute on a single machine. Thus, the processing needs to scale up to cover all distributed data produced by LHC [3]. Many scientists will simultaneously submit large amounts of queries and the system, therefore, must be able to manage queries off-line. Better performance can be achieved by identifying when results from running and old queries can be reused.

The remainder of this paper is organized as follows. Related works are discussed in Section 2. Section 3 describes the application from Particle Physics. Section 4 introduces the NorduGrid middleware and its restrictions. The architecture of our system is presented in Section 5. Section 6 gives the query execution strategy used in the system for the application specific queries. Section 7 concludes.

2 Related work

Relational DBMSs allow the representation of very large and reliable disk-based databases with good performance for business application that store large amounts of structured tabular data. They are furthermore optimized for many short transactions over a few relatively small data items, while the new applications require also the incorporation of long running, expensive, and distributed computations using and producing many large objects and files.

Object-relational DBMSs [40] allow the definition of abstract data types in relational databases having *user defined functions* that are executed in the database server. Object-relational extensions are now part of the SQL-99 standard. However, computationally intensive applications still do not use DBMS technology since they require customized main-memory data structures for realistic performance. They also rely on program libraries written in conventional programming languages and operating on conventional files. Thus a query system utilizing scientific data needs also to be able to incorporate existing scientific libraries and data files into queries. Regular query processors assume cheap primitive operations while the scenarios considered here require optimization of queries with very expensive user-define functions.

In *distributed databases* [33] data distribution is transparent only for queries while data itself is manually distributed using placement conditions. *Parallel databases* are similar, but aim at letting the DBMS transparently and automatically distribute data. Parallel databases normally run on clusters of co-located computers with very fast intercommunication links. Query processing techniques for distributed and parallel databases are used by POQSEC for transparent access to distributed clusters.

Another modern development in the database area is to use large main memories to represent the database [20]. DBMS vendors are developing lightweight main memory relational databases [23] [32] [38] often interoperating with their DBMSs. For high performance, we are utilizing an embeddable main-memory DBMS [35] which includes an extensible and object-oriented query optimizer, i.e. it is an object-relational main-memory DBMS.

Mediators (multi-databases, federated databases) (e.g. [4] [19] [21] [27] [34] [43] [44]) are a modern database approach aiming at transparently integrating (combining) data from many different external data sources. In POQSEC we extend our previous work on integration of heterogeneous data sources [15] [22] [28] for access to Grid-managed external data repositories, relational databases, and to wrap CPU demanding computations.

POQSEC runs on top of existing Grid infrastructures, in particular the Swegrid computational infrastructure [37] and the NorduGrid resource management system [31] [41]. This puts certain restrictions on availability of computational resources and possibility to obtain them compared to systems such as distributed query processors Polar* [42] and LeSelect [5] that assume full control over all resources including computational resources, network management, etc.

One example of utilizing operational Grid infrastructures for transparent execution of jobs is the computational management agent Condor-G [8] [16]. It has its own resource broker to find computational Grid resources. It submits the jobs to a local *gate keeper* of the resource, e.g. the Globus Gatekeeper [14], using the Grid Resource Allocation and Management protocol (GRAM) [9]. In contrast, POQSEC reuses the resource broker from the NorduGrid middleware [41]. It receives declarative queries unlike Condor-G, which receives traditional batch jobs. Based on the queries POQSEC then generates and submits jobs to NorduGrid for execution.

Rather than manually organizing long-running activities as in workflow systems (e.g. [10]), the long running computations are specified in POQSEC as declarative queries from which the query optimizer automatically generates execution plans consisting of several dependent tasks executed on the Grid and managed by POQSEC.

3 Application

In order to evaluate POQSEC we have chosen an application from Particle Physics. The aim of the application is to analyze data produced by the ATLAS Data Challenges [1] for containing charged Higgs bosons [6]. The analysis is long running because of the huge amount of input data.

The input data are called *events,* which describe collision events between elementary particles. Each event comprises sets of particles of various types such as *electrons*, *muons*, and 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*, *Pz*), and amount of energy (*Ee*). For example, an event might have one electron, two muons, four jets, and values of PxMiss and PyMiss for the event are -198.237 and 181.605, correspondingly. Examples of values for some particles of the event are presented in Table 1.

Table 1. Values of the particles from the example event

Particle name	Kf	Px	Py	Pz	Ee
Electron	-11	-96.3295	55.0114	-336.974	354.764
Muon	-13	-24.6514	-18.9128	-91.4735	96.6065
Muon	13	-6.23232	-10.2039	-38.1292	39.9601
Jet	2	197.085	8.94369	-165.14	257.281
Jet	2	141.008	-86.3656	-205.205	263.536
Jet	5	-35.3334	-45.2087	29.3406	64.4449
Jet	5	31.1251	-84.9691	-342.232	353.993

These event data are generated by the ATLAS software and stored in files managed by the ROOT library [36]. The size of generated files is from one to several GBytes [2].

The analysis of each event can be divided into several steps. Each step is a condition containing logical, arithmetic, and vector operations over event data. Events satisfying all conditions constitute the final analysis result. Partial results are also produced based on events that satisfy partial conditions.

An example of one of the steps is a *Z-veto cut*. An event satisfies this step if it does not have a pair of opposite charged *leptons*, where each lepton is either a muon or electron, or if it has a pair of opposite charged leptons the invariant mass of the pair is not close to the mass of a Z particle.

This step can easily be expressed as a declarative query, for example, in the declarative language AmosQL [13] that is used in POQSEC. AmosQL has a functional object-relational data model and SQL-like syntax. We developed an object-oriented schema to present the application data in the system. Fig. 1 illustrates the main part of the schema in Enhanced Entity-Relationship notation [12]. Events are represented by type *Event*, which has three attributes, values *PxMiss*, and *PyMiss*, along with a relationship to particles that the event comprises. The particles are represented by three basic types *Electron*, *Muon*, and *Jet*, and two general types *Lepton* and *AbstractParticle*. The types *Electron*, *Muon,* and *Jet* correspond to electrons, muons, and jets, respectively. Electrons and muons often participate in analysis queries as one kind of particles, leptons, thus the types *Electron* and *Muon* are generalized by type Lepton. The types *Electron*, *Muon,* and *Jet* have the same attributes, *Kf*, *Px*, *Py*, *Pz*, and *Ee*, therefore, a general type *AbstractParticle* is introduced.

The Z-veto cut step is expressed as the following AmosQL query:

```
SELECT ev
FROM Event ev
WHERE NOTANY(oppositeLeptons(ev)) OR
      abs(invMass(oppositeLeptons(ev)) - zMass) >= minZMass);
```

The function `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 a query:

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

```
Ee(l1) = -Ee(l2);
```

The function `leptons` returns a set of lepton pairs belonging to a given event, and `Ee` is a function which returns the value of the energy of a given lepton.

All steps of the analysis were expressed declaratively in AmosQL.

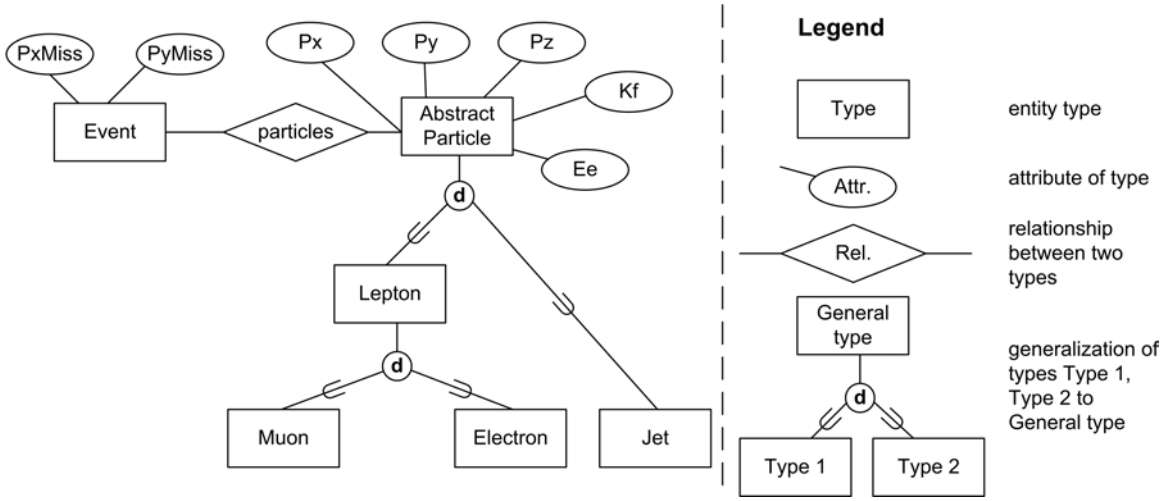

Fig. 1. Basic schema of the application data

4 The NorduGrid infrastructure

The first prototype of POQSEC is running on the Swedish computational Grid SweGrid [37] on top of the NorduGrid resource management system [31] [41]. The NorduGrid (NG) middleware (also called the Advanced Resource Connector, ARC) provides a uniform job submission mechanism so that a user can submit jobs to any or particular cluster(s) of the Grid. The Nordugrid middleware can be divided in three parts: the *NG client* which contains the *user interface* and the *resource broker*, the *Grid manager,* and the *information system*. The task of the user interface is to process job submissions, job status requests, and data transfer requests by a user. The user specifies jobs and their requirements, such as maximum CPU time, requirements for a cluster where the job is executed, requests to move output data from and to specific storage elements, number of sub-jobs for parallel tasks, requests for cluster nodes with IP connectivity, etc. The jobs are specified using the Extended Resource Specification Language (xRSL) [39]. The user interface includes resource broker functionality. It finds a suitable cluster for executing a submitted job using the information system that provides monitoring and discovery service [25]. On each cluster accessible through the NorduGrid a Grid manager [24] is installed. The task of the Grid manager is to submit a job received from a user interface to the *local batch system* of the cluster and to download input data and upload output data. The NorduGrid guarantees that the cluster fulfills the job requirements and that all input are available on the cluster when the job is started by the local batch system.

There are certain limitations on the usage of computational resources available throughout the NorduGrid, for example:

- The NorduGrid can notify about a job status only by email. However the job status can be obtained by querying the NG client [30].
- A job submitted to NorduGrid has a limit on maximum execution time, which must be set by the user for submission.
- Data produced by a job will be kept on the cluster only for limited amount of time, usually 24 hours.
- A job submitted for parallel execution to NorduGrid has to use the Message Passing Interface (MPI) [29] to be started in parallel.
- When a job is submitted for parallel execution to NorduGrid the number of needed cluster nodes must be specified in the job script.
- The input and output data are handled only at the beginning and at the end of a job.

Other limitations arise from the fact that the NorduGrid middleware does not control clusters directly. Therefore, it cannot guarantee the time when the execution of a submitted job starts.

5 System Architecture

Fig. 2 illustrates the architecture of the POQSEC system running on top of the NorduGrid middleware (NG). The architecture contains four main POQSEC components: *query coordinators*, *start-up drivers*, *executors*, and *supervisors*.

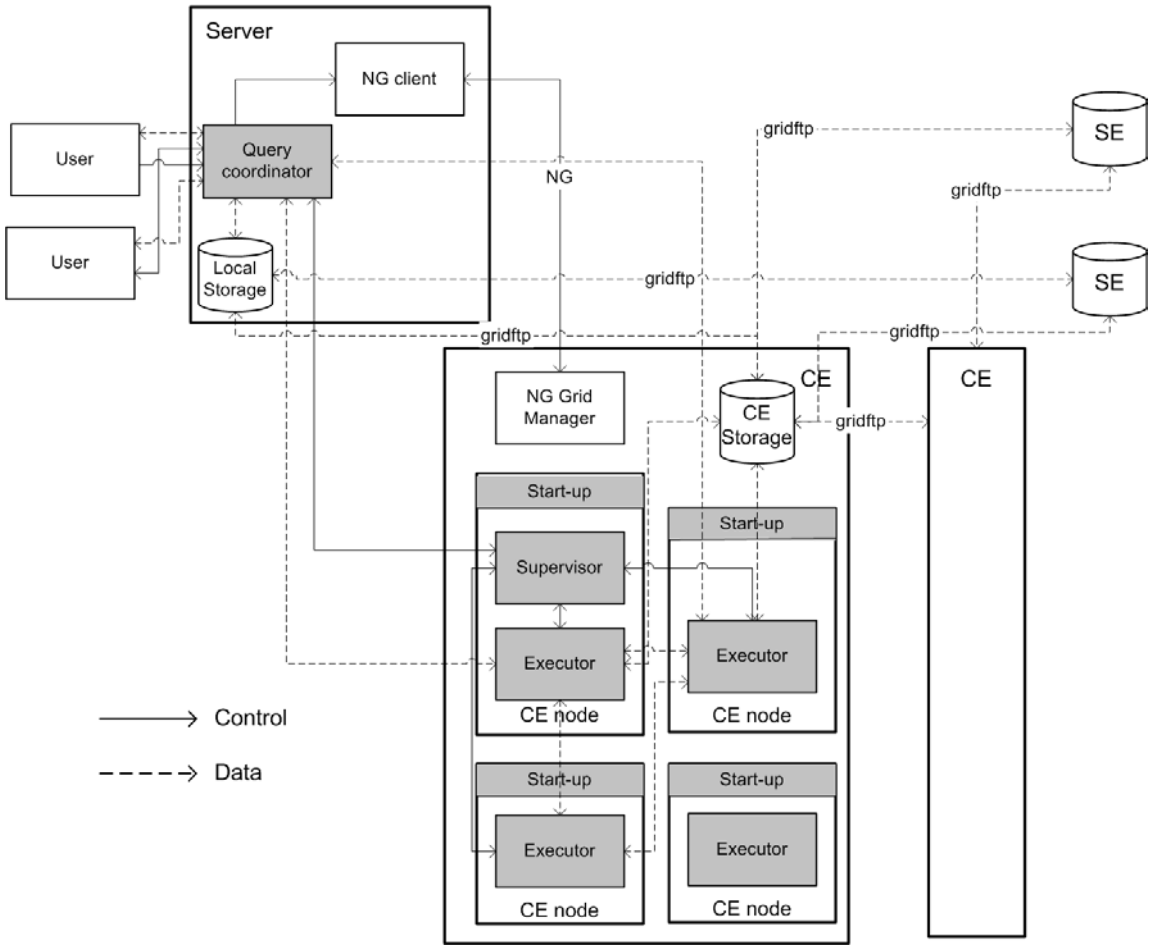

Fig. 2. POQSEC Architecture

NorduGrid components that play important roles in the architecture are the *NG client* and the *NG Grid manager,* which manage jobs submitted to NorduGrid. The *query coordinator* is a POQSEC node that runs on the same server as the NG client. It processes user queries and generates jobs submitted to the NG client. The queries by nature contain database operations such as data selection, projection, and join in addition to application specific operations. The application specific operators are executed as user defined functions calling application libraries such as ROOT [7].

The *computing elements* (*CE*s) and *storage elements* (*SE*s) are external resources available through the NorduGrid infrastructure. SEs are nodes managing permanent data repositories. CEs are computational clusters and each of them has working nodes (*CE node*s), providing local computation, and storage (*CE storage*). The CE storage provides temporary data storage shared between working nodes.

A *user* is an application, running on a client or server machine that interacts with a POQSEC query coordinator. It submits queries to the query coordinator for data processing. The queries often contain computations over large data sets and may therefore be running for long time periods.

Each query coordinator has its own database manager and access to a *local storage,* which is used as temporal file storage. A query coordinator processes queries submitted by users. Some of the queries that do not require computational resources for execution can be executed directly by the query coordinator. Queries expected to be long running are decomposed into subqueries and sent to the NorduGrid for parallel execution. Long running queries require generating xRSL job scripts describing the requirements for computational resources, the procedures for starting POQSEC modules on CE nodes, and the input and output data for each subquery to be

submitted to the NorduGrid client. The query coordinator keeps track of all submitted subqueries by communicating with *supervisors* that monitor and control execution of (sub)queries and run on CE nodes. The query coordinator notifies users about the query status on request. It also sends the query result on a user request when the query execution is finished.

The query coordinator must also deal with job failures. It tracks status of jobs submitted to NorduGrid by polling NG client and when the status is "failed" it resubmits the job.

The NG client software manages the submitted jobs by first extracting from the xRSL scripts requirements for computational resources, such as estimated CPU time, number of required cluster nodes, possibility to connect cluster nodes from an outside cluster, required memory and disk space, version of installed middleware, etc. [39]. The NG client then submits the scripts for execution on clusters fulfilling the requirements. NorduGrid takes care to transfer input data before execution starts, thus guaranteeing that all input data is available when the execution starts, and to transfer result data to a destination specified in the script when execution is finished.

POQSEQ modules are started on CE nodes of the selected cluster when the required number of nodes is available. Each POQSEC module starts by executing a *start-up driver*, which is an MPI program, to initialize the execution. MPI functionality of the start-up driver provides an easy way to choose one node to run a supervisor and to notify start-up drivers of the other nodes about the hostname of the node where the supervisor is running. After the supervisor is started the start-up drivers start *executors* on all the nodes and pass them the hostname of the supervisor. Executors start by registering themselves in the supervisor and then retrieving from the supervisor the information about those other executors they need to communicate with. The executors optimize and execute the queries over the data that is available on the CE storage. Data is sent between executors if it is required for the execution. Produced result is saved in local CE storage for transfer to a SE, another CE, or local storage of the node where the query coordinator is located.

The main task of a supervisor is to monitor and control executions. It communicates with executors to obtain the status and with query coordinators to notify them about the current status.

The architecture assumes the possibility to connect CE nodes with nodes outside CE using IP communication. All communication among POQSEC modules is secure and Globus I/O library [17] is used for this.

6 Query execution strategy

The query coordinator decomposes each user query in two levels:

1. *Cluster subqueries* are generated to be executed on clusters as separate NorduGrid jobs.
2. Each cluster subquery is further decomposed into *local subqueries* for parallel execution on the cluster to which it is submitted.

The query optimization has to be done by the query coordinator since required computational resources must be specified in the NorduGrid job scripts when they are submitted.

The first query execution strategy of the system is being developed and evaluated for the application in Section 3.1.

We assume that the data is stored in files of size about 1-2 GB on SEs [2] and accessible through NorduGrid. Execution of a query over one file of such size by one executor on one node takes more than 1 hour. Thus it is reasonable to parallelize execution of the query over one or several files between several executors that run on CE nodes in parallel. Query parallelization is done by partitioning data between executors and executing the same query by different executors over different data partitions. Each executor first accesses a chosen input data file to extract its own portion of data to be processed. Different strategies for file partitioning are being investigated.

Often a user is interested in analyzing many files. Then the query coordinator creates several parallel jobs, one per group of files of the total size 5-10 GB, and for each job it specifies the number of required CE nodes. The query coordinator takes into account that one CE node will be used to merge results produced by other nodes. Each created job is submitted to NorduGrid and the execution is started as described earlier in the architecture section. The executor collocated with the supervisor is used to merge the result; other executors process analysis queries. Each executor optimizes the query and executes a created optimal plan over the data. The executors stream their result data to the executor chosen to do the result merge. The merged result is first saved in the local CE storage and then uploaded by NorduGrid to the local storage of the node where the query coordinator is running. The query coordinator finally merges all the results produced by the jobs and retrieves them to the user.

7 Summary and continued work

An architecture has been developed for utilizing the operational NorduGrid infrastructure for managing long running and expensive scientific queries. The architecture is being implemented and evaluated for a Particle

Physics analysis application [6]. It is running on top of the NorduGrid infrastructure [41] so special attention is needed to adhere to limitations of this architecture, e.g. that the exact starting time or place of a job cannot be guaranteed and that the number of nodes to use on a cluster must be explicitly given in the job specification. Globus I/O [17] is used for secure communication between distributed POQSEC nodes.

POQSEC utilizes the AMOS II database management system [35] that provides object-relational DBMS functionality, peer to peer communication, declarative query language AmosQL [13], and interfaces to C++ and Java. The kernel is being extended in order to implement the architecture.

The analysis of event data in our implementation is specified declaratively using AmosQL query language. The queries are expressed in terms of logic and arithmetic functions over numbers and vectors. This requires optimization of scientific queries and aggregation functions over numerical data and operations.

In order to be able to read data produced by ATLAS software an object-oriented data analysis framework, the ROOT library [36], was linked to the system. The ROOT library is also going to be used for computations in the application analysis.

The POQSEC system runs under Linux on Swegrid clusters managed by NorduGrid. Parallel query plans are currently created manually by splitting data and queries to investigate trade-offs with the goal to develop automatic query partitioning methods next. Based on this analysis, automatic query decomposition strategies for distributed execution are being investigated.

Another problem that will require special attention and investigation is tracking CPU time so that when the job execution time is approaching deadline the job should be suspended and intermediate results and states should be saved. POQSEC would then resume execution later by restoring the saved state.

Since availability of resources cannot be precisely predicted in advance, adaptive query optimization technique [18] can be used to improve the performance.

Acknowledgments

This work is funded by the Swedish Research Council (www.vr.se/english). We would like also to thank Christian Hansson and professor Tord Ekelöf from the department of Radiation Science, Uppsala University, for providing the particle physics analysis application and data as system test case. The secure communication is implemented by Mehran Ahsant from Center for Parallel Computers, Royal Institute of Technology, Stockholm.

References

[1] *ATLAS Data Challenges*, 2001, [Online], http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/doc/AtlasDCs.pdf

[2] *ATLAS Data Challenge 1, Draft 3.0*, 2003, [Online], http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/DC1/DC1-Report-V3[1].0-22092003.pdf

[3] S. Bethke, M. Calvetti, H.F. Hoffmann, D. Jacobs, M. Kasemann, D. Linglin: *Report of the Steering Group of the LHC Computing Review*, CERN/LHCC/2001--004, 2001. Available at http://lhcb-comp.web.cern.ch/lhcb-comp/Reviews/LHCComputing2000/Report_final.pdf

[4] O. Bukhres, A. Elmagarmid (eds.): *Object-Oriented Multidatabase Systems: a Solution for Advanced Applications*, Prentice Hall, 1996.

[5] L. Bouganim, F. Fabret, F. Porto, P. Valduriez: Processing Queries with Expensive Functions and Large Objects in Distributed Mediator Systems, *Proc. of Int. Conf. on Data Engineering (ICDE)*, 2001

[6] M. Bisset, F. Moortgat, S. Moretti: Trilepton+top signal from chargino-neutralino decays of MSSM charged Higgs bosons at the LHC, *Eur.Phys.J.C30*, 419-434,2003

[7] R. Brun, F. Rademakers: ROOT – An Object Oriented Data Analysis Framework, *Proceedings AIHENP'96 Workshop*, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86

[8] Condor-G System, [Online], http://www.cs.wisc.edu/condor/condorg/

[9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke: A Resource Management Architecture for Metacomputing Systems, *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62-82, 1998

[10] Directed Acyclic Graph Manager, [Online], http://www.cs.wisc.edu/condor/dagman/

[11] P. Eerola, T. Ekelof, M. Ellert, J.R. Hansen, S. Hellman, A. Konstantinov, B. Kónya, T. Myklebust, J.L. Nielsen, F. Ould-Saada, O. Smirnova, A. Wäänänen: Atlas Data-Challenge 1 on NorduGrid, *ECONF C0303241:MOCT011*, 2003

[12] R. Elmasri, S.B. Navathe: Enhanced Entity-Relationship and UML Modeling, *Fundamentals of Database Systems*, 4th ed., Addison-Wesley, 85-121, 2004

[13] S. Flodin, M. Hansson, V. Josifovski, T. Katchaounov, T. Risch, M. Sköld: *Amos II Release 6 User's Manual*, 2004, [Online], http://user.it.uu.se/~udbl/amos/doc/amos_users_guide.html

[14] I. Foster, C. Kesselman: Globus: A Toolkit-Based Grid Architecture, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 259-278, 1999

[15] G. Fahl, T. Risch: Query Processing over Object Views of Relational Data*, The VLDB Journal*, Vol. 6, No. 4, 261-281, 1997. Available at http://www.dis.uu.se/~udbl/publ/vldbj97.pdf

[16] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke: Condor-G: A Computation Management Agent for Multi-Institutional Grids, *Proceedings of the Tenth {IEEE} Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, California, pp. 7-9, 2001

[17] *Globus I/O API*, [Online], http://www-unix.globus.org/api/c-globus-3.2/globus_io/html/

[18] A. Gounaris, N.W. Paton, A.A.A. Fernances, R. Sakellariou: Adaptive Query Processing: A Survey, *Proceedings of the 19th British National Conference on Databases: Advances in Databases*, 2002

[19] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y.Sagiv, J. Ullman, V. Vassalos, J. Widom: The TSIMMIS Approach to Mediation: Data Models and Languages, *Journal of Intelligent Information Systems (JIIS),* Kluwer, 8(2), 117-132, 1997

[20] H. Garcia-Molina, K. Salem: Main Memory Database Systems: An Overview, *IEEE Transactions on Knowledge and Data Engineering*, 4(6), 509-516, 1992.

[21] L. Haas, D. Kossmann, E.L. Wimmers, J. Yang: Optimizing Queries across Diverse Data Sources, *23rd Intl. Conf. on Very Large Databases (VLDB'97),* 276-285, 1997

[22] V.Josifovski, T.Risch: Query Decomposition for a Distributed Object-Oriented Mediator System, *Distributed and Parallel Databases*, J. Kluwer, 307-336, 2002. Available at http://user.it.uu.se/~torer/publ/dpdb.pdf

[23] J.S.Karlsson, A.Lal, C.Leung, T.Pham: IBM DB2 Everyplace: A Small Footprint Relational Database System, *17th International Conference on Data Engineering*, 2001.

[24] A. Konstantinov: *The NorduGrid Grid Manager and GridFTP Server, Description and Administrator's Manual*, 2004, [Online], http://www.nordugrid.org/papers.html

[25] B. Kónya: *The NorduGrid Information System*, 2002, [Online], http://www.nordugrid.org/documents/ng-infosys.pdf

[26] The Large Hadron Collider, [Online], http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/

[27] L.Liu, C.Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources, *Distributed and Parallel Databases,* Kluwer, 5(2), 167-205, 1997.

[28] H.Lin, T.Risch, T.Katchanounov: Adaptive data mediation over XML data. In special issue on "Web Information Systems Applications" *Journal of Applied System Studies (JASS)*, Cambridge International Science Publishing, 3(2), 2002. Available at http://user.it.uu.se/~torer/publ/jass01.pdf

[29] The Message Passing Interface (MPI) Standard, [Online], http://www-unix.mcs.anl.gov/mpi/

[30] *The NorduGrid User Guide*, 2004, [Online], http://www.nordugrid.org/documents/userguide.pdf

[31] NorduGrid middleware, the Advance Resource Connector, [Online], http://www.nordugrid.org/middleware/

[32] ORACLE Inc.: *Oracle8i Lite: The Internet Platform for Mobile Computing*

[33] M.T.Özsu, P.Valduriez: *Principles of Distributed Database Systems*, 2nd ed. Prentice Hall, 1999.

[34] D. Quass, A. Rajaraman, Y. Sagiv, J.Ullman, J. Widom: Querying Semistructured Heterogeneous Information. In *Deductive and Object-Oriented Databases, Proceedings of the DOOD'95 conference*, 1995, LNCS Vol. 1013, 319-344, Springer 1995.

[35] T.Risch, V.Josifovski: Distributed Data Integration by Object-Oriented Mediator Servers, *Concurrency and Computation: Practice and Experience J.* 13(11), John Wiley & Sons, September, 2001. Available at http://user.it.uu.se/~udbl/publ/ddiooms.pdf

[36] *ROOT: An Object-Oriented Data Analysis Framework – User's Guide*, 2004, [Online], http://root.cern.ch/root/doc/RootDoc.html

[37] Swegrid, [Online], www.swegrid.se

[38] SYBASE Inc.: *SQL Anywhere Studio*, http://www.sybase.com/mobile/

[39] O. Smirnova: *Extended Resource Specification Language*, 2003, [Online], http://www.nordugrid.org/documents/xrsl.pdf

[40] M.Stonebraker, P.Brown: *Object-Relational DBMSs: Tracking the Next Great Wave*, 2nd ed., Morgan Kaufmann Publishers, 1999

[41] O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J.R. Hansen, A. Konstantinov, B. Kónya, J.L. Nielsen, F. Ould-Saada, A. Wäänänen: The NorduGrid Architecture and Middleware for Scientific Applications, *International Conference on Computational Science*, 264-273, 2003

[42] J. Smith, A. Gounaris, P. Watson, N.W. Paton, A.A.A. Fernandes, R. Sakellariou: Distributed Query Processing on the Grid, *Proc. Grid Computing 2002*, pages 279-290, Springer, LNCS 2536, 2002

[43] A. Tomasic, L. Raschid, P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO, *IEEE Transactions on Knowledge and Date Engineering*, 10(5), 808-823, 1998

[44] G. Wiederhold: Mediators in the architecture of future information systems, *IEEE Computer,* 25(3), 38–49, 1992.