CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e–mail: prdoc@itc.it – url: http://www.itc.it

GRID QUERY OPTIMISATION IN THE DATA GRID

Busetta P., Carman M., Serafini L.,
Stockinger K., Zini F.

September 2001

Technical Report # 0109–01

© Istituto Trentino di Cultura, 2001

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of ITC and will probably be copyrighted if accepted for publication. It has been issued as a Technical Report for early dissemination of its contents. In view of the transfert of copy right to the outside publisher, its distribution outside of ITC prior to publication should be limited to peer communications and specific requests. After outside publication, material will be available only in the form authorized by the copyright owner.

Grid Query Optimisation in the Data Grid

Query Optimisation Task - WP2, European DataGRID Project

Paolo Busetta, Mark Carman, Luciano Serafini, Floriano Zini
ITC-IRST, Povo, Trento, Italy

Kurt Stockinger
CERN, Geneva, Switzerland

1 Purpose

Our aim is to realise an agent-based Grid simulator for studying "Grid Query Optimisation Strategies", as partially described in [SSS+01]. The purposes of this document are the following:

- To identify some HEP use cases and, based on these use cases and requirements, to identify some user-level and Grid-level query optimisation opportunities.
- To define an architecture for the simulator, in which the components and services identified in the ATF proposal [ATF] are integrated with components and services, specifically targeted to query optimisation.
- To define how the components and services in the simulator interact in order to perform optimisation based on the identified optimisation opportunities.

Our goal in realising the simulator is twofold:

- To stimulate the discussion in the DataGRID project about the important issue of query optimisation and help the gathering of requirements for it by providing a working software tool that allows experimentation, testing, and validation of optimisation strategies.
- To promote agent technology for the realisation of "intelligent" and flexible Grid services (of which query optimisation is one) that have to take into account several dynamically changing parameters and manage possibly contrasting needs of the different entities (both humans or software systems) that are involved.

We base our use cases mostly on the "CMS Data Grid System Overview and Requirements" [ABH+01] since this is the most "visionary" and complete document currently available. The description of the use cases of other HEP experiments and other applications such as biology or earth observation is beyond the scope of this document but will be part of our future work when the use cases of these applications will be available. We intend to discuss with each application separately what their "Grid Query Optimisation Use Case" looks like and this document will serve as a guideline for future use case studies in order to discover commonalties and differences among the applications in the DataGRID project.

Writing this document helped us to identify specific requirements from one particular experiment, to analyse a fairly complete picture of how the Grid can be used for distributed physics analysis, and to identify which optimisation opportunities there are in this setting. We expect that both the data model and the Grid architecture will evolve over time but we regard this document as an initial collection of requirements for our agent-based Data Grid simulator.

The document is structured as follows. In order to set up the framework of our work, we first introduce the Grid environment, including a typical HEP Data Model and a typical HEP Grid

infrastructure, (both taken from CMS [ABH+01]). Then, we introduce some typical use cases for HEP analysis and identify some optimisation opportunities on these use cases. The following section introduces a proposed Grid architecture, which extends the current Grid architecture [ATF] to include components necessary for Grid Query Optimisation. A HEP use case is then re-examined in the context of the architecture to better explain the interactions and dependencies of different architecture components. Finally, the last section briefly outlines the current status and our future plans in creating a Grid simulator for evaluating different "Grid Query Optimisation" techniques.

2 The Grid Environment

2.1 A Typical HEP Data Model

In order to perform HEP analysis, experiment output data (low-level "raw" data) is converted to higher level "meaningful" and "interesting" data in a number of stages. The different stages in this conversion are shown in Figure 1.

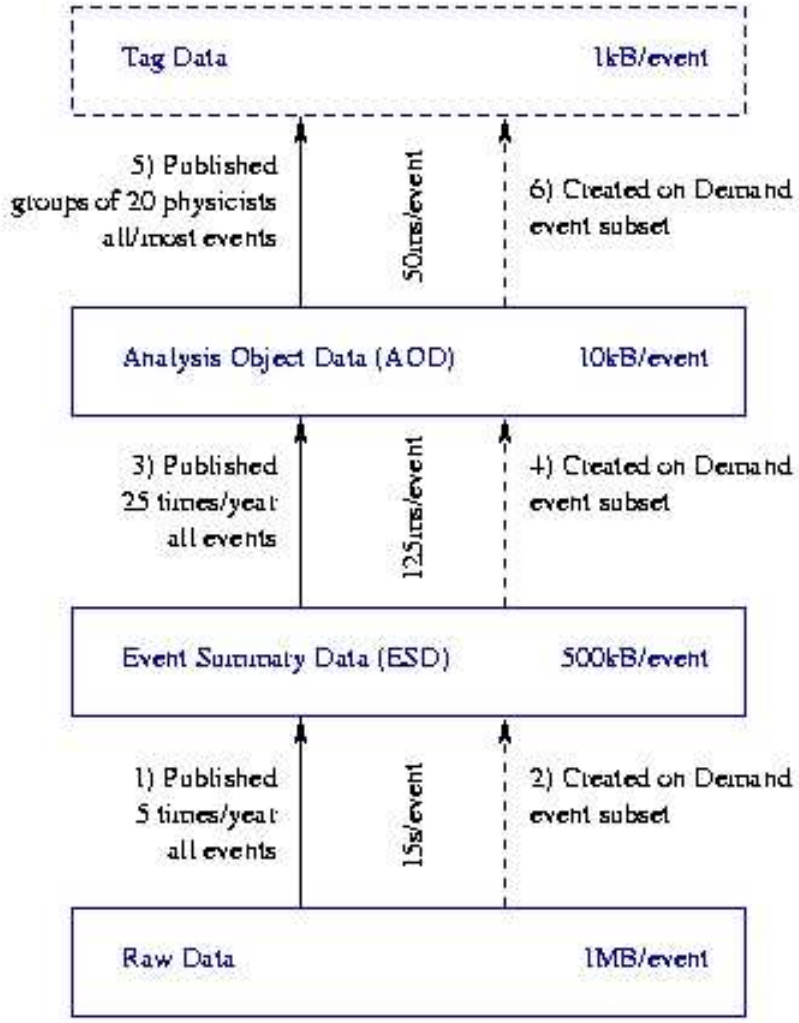

Figure 1: Data model in CMS.

Raw data is produced at a rate of 10^9 events per year for a typical LHC experiment such as CMS. Other LHC experiments, ALICE for example, have lower collision rates but the data volume of a single event is larger.

An event here refers to a single collision between groups of protons inside one of the Large Hadron Collider detectors at CERN. Data is stored in large binary files, where a certain "section" of a binary file corresponds to detector readings for a specific event. A complete set of these binary files will be stored in large storage devices (probably tapes) at CERN, (the so-called "Tier 0" site). Other copies of the raw data (containing 50% to 100% of all events) will be maintained at Tier 1 sites (of which there should be around 5) around the world. It is unknown at this stage whether Data Grid services will have any control over the replication of raw data at Tier 1 sites, or whether all decisions will be made by Tier 1 site managers as part of their local replication policy. We assume the latter to most likely be the case.

For use in analysis by physicists, this raw data must be "reconstructed" form Event Summary Data (ESD). ESD representation of events has a complex (object hierarchy) data structure. Objects contain primitive data types that aggregate some attributes of raw data from which they originated and pointers to these raw data descriptions. Subsets of ESD will be replicated all over the Grid. The choice as to when, where and what ESD to replicate will probably be made both by Grid optimisation services and by local site managers.

It is envisaged that complete sets of ESD will be created five times a year from the original raw data source (arrow labelled 1 in figure 1). These "versioned outputs" of ESD will be produced by the physics community as a whole, using updated and improved algorithms to perform the conversion. Each versioned output of ESD will be replicated widely, and older versions of ESD will be archived for future reference.

In order to develop new and improved algorithms for deriving ESD, some physicists will need to test algorithms on subsets of raw data. This process of on-demand ESD creation is shown in figure 1 by arrow labelled 2. On-demand ESD creation could also be triggered by the Grid in the case where certain (virtual) ESD objects are required by a physicist for subsequent computation, but only the raw data is available. In order for this to be possible, the algorithms used to create versioned ESD outputs must be known to the Grid i.e., exist in the global namespace, and be available for use by Grid components.

Analysis Object Data (AOD) is the next level of abstraction in the information hierarchy of HEP experiments. Physicists spend most of their time analysing this and higher level data. AOD events are made up of a complex hierarchy of objects (but less complex than ESD objects), containing primitive types, and pointers back to parts of ESD objects and raw data descriptions. The size of each AOD event is around 10kB, but will change depending on the event. AOD is stored in storage devices at various sites on the Grid. Its replication (in terms of the choice when, where and what to replicate) will again most likely be managed both by Grid optimisation services and by site managers.

New sets of AOD will be created on average around 25 times per year, from entire sets of the versioned ESD (arrow labelled 3 in figure 1). In addition, AOD objects may be created on demand by the Grid from particular ESD objects (arrow labelled 4 in figure 1). This would be the case if a physicist were testing new algorithms for AOD creation (perhaps with varying parameters or new input ESD objects). It may also be the case that the Grid "materialises" certain sets of AOD locally rather than transports them from another site on the Grid, to minimise bandwidth usage. If this were to be the case, then the algorithms used to create the AOD from the ESD would need to be known and be accessible by the Grid optimisation services.

AOD is often too large or not specific enough for physicists to do "day-to-day" analysis on it directly. Thus, the whole collaboration (i.e. experiment) will perform collective production of

"summary tables" containing values for the most important attributes (or aggregations of them) from all of the events. The tables contain links to objects within AOD, and/or ESD, and/or RAW data. Such summary tables are referred to as Tag data, and are stored on local data resources. Typical tables will contain 1kB worth of data for each of the 10^9 events produced each year. These so-called "collaboration tags" will be produced 25 times per year. In addition, there may be around 25 groups of physicists per experiment who produce Tag data on a subset of all events, so-called "group tags". Tag data can be also created on demand (arrow labelled 6 in figure 1), in order to test new algorithms for Tag data production.

We assume that Tag data will be *invisible to the Grid* and thus will not need to be managed by it. Having created sets of Tag data, physics researchers will spend much of their time analysing the data at that level, until they exhaust the information available at that granularity, and start analysing the extra information available in the AOD.

2.2 A Typical HEP Grid Infrastructure

A typical hierarchical HEP Data Grid infrastructure as proposed by [ABH+] looks as shown in figure 2. Boxes represent typical configuration for the different kinds of sites that will make up the Data Grid.

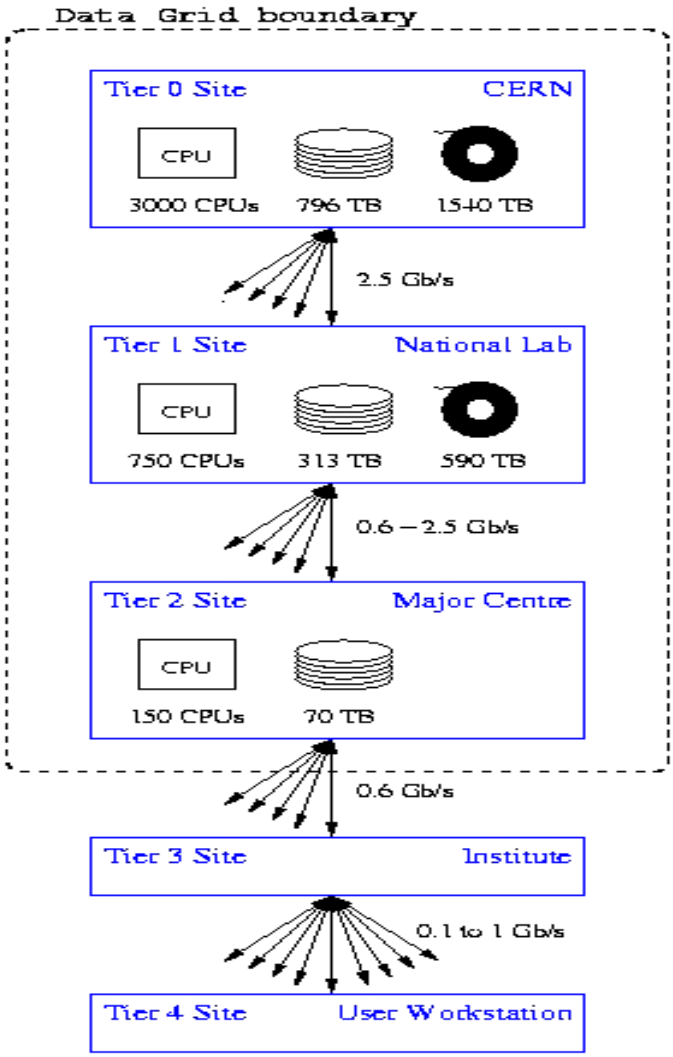

Figure 2: Available HEP Data Grid Infrastructure.

The approximations for network bandwidth, CPU (corresponding to SI95) and data resources are CMS specific. However, other experiment specific estimations can be considered in the future.

As can be seen from the diagram, only Tier 0 to Tier 2 sites will have "Grid aware" data and computational resources. Tier 3 and 4 sites will manage their own resources, running local applications on them, which will then interact with Grid applications/services, to utilise the power of the Grid enabled resources. During the later stages of the project, however, also Tier 3 and Tier 4 sites might become "Grid aware".

3 Defining the Use Cases

There are three main activities that will be performed by the HEP experiments on the Data Grid.

1. ***Testing of algorithms for data production***. Physicists will perform this activity in order to tune the algorithms that they want to use to produce higher-level data types from lower-level data types (e.g., AOD data from ESD data).
2. ***Data production***. This activity will be performed by "data production managers", in order to obtain a full conversion of data from a lower-level format to a higher-level format.
3. ***Analysis***. This activity will be performed by physicists, in order to analyse data and discover new properties of matter.

The following table outlines the similarities and differences between the three types of activities.

Job Type	*User*	*Scheduled*	*Input Data Set*			*Output*
			Known	*Size*	*Spans Datatypes*	
Testing	Physicist	No	Yes	Small Subset	No	Data Subset
Production	Manager	Yes	Yes	All Events	No	Data Set
Analysis	Physicist	No	Partially	Small to All	Yes	Histogram/Tag

The analysis process differs from the other types of activities in certain important ways. Unlike the other activities, the output of the process is generally private data (such as a histogram or a collection of tags), that will not be managed by the Grid. The creation of summary statistical data like histograms also implies the need for an aggregation step not present in the other processes. (The aggregation step brings together the output data from the execution of jobs on individual events). More interestingly, it is harder to know what data an analysis job will access, and in some cases the data accessed in a single job will come from more than just one datatype (e.g. AOD and ESD together). Production and testing activities will be performed massively in the earlier phases of the HEP experiments, while analysis will become far more frequent and important over the lifetime of the experiments. Because of its overall importance and of the difficulty in optimising the process manually, we will focus our optimisation strategies on the execution of the analysis process.

3.1 HEP analysis

Physicists analyse the events generated in the LHC by iteratively selecting a more and more restricted set of them, in order to isolate only the "interesting" events from the 10^9 events generated each year by an experiment. Selection is performed by defining *cut predicates* that are applied in sequence to smaller and smaller sets of events.

A cut predicate p is applied to an input event set S and yields another set of events $S' = \{e \in S / p(e) = true\}$ as output. Cut predicates select those events that satisfy some properties defined on values of data products belonging to the events.

In the earlier phases of the analysis, a cut predicate *p(e)* is evaluated using the values of Tag data products. Data selection is specified by cut predicates defined as set of constraints on certain Tag attributes of the events in the set (e.g., boolean expressions containing "greater-than", "less-than" and "equal-to" operations are used to restrict the input data set). During TAG data analysis, physicists typically "cut" the table data down to around 2% to 20% of its original size, until the selective power of these data type is exhausted.

In the later phases of analysis, physicists exploit values of more "low level" data product, as AOD, ESD or even Raw data to perform further selection on the event set. In these phases, application of a cut predicate on an input data set is performed by submitting a *job* to the Data Grid. A job is a user-supplied piece of code and a specification for an input data set[1] and an output data set. The piece of code executes the event selection based on the values in the input data set and produces the output data set possibly combining or aggregating data values of the selected events.

Cut predicates are created by stepwise refinement. In each step a new version of the predicate is applied to the input event set, then the output event set is examined to decide whether the current version of the predicate is useful or not to continue the analysis.

3.2 Scenarios

We will be considering the following scenarios.

1. A physicist is doing HEP analysis and she has been working on the highest level data (Tag data). These first phases of analysis are performed outside the Data Grid. Having completed part of the analysis, the physicist then "runs out" of information at the Tag level and is forced to access lower level data (AOD) to continue the analysis. The physicist starts to submit jobs to the Grid. These jobs specify some AOD data products as input data set and define a cut predicate on the input data set, in the form of a piece of C++ code. The location for the output data set is also specified by the job, which exists outside of the Data Grid infrastructure.
2. The physicist has continued analysis on AOD data until she has exhausted information at that level. The rest is similar to scenario 1.
3. The physicist has continued analysis on ESD data until she has exhausted information from them. Again the rest is similar to scenario 1.
4. The fourth scenario does not follow from the previous one, but refers to a more general form of physics analysis, in which the physicist performs analysis on multiple datatypes concurrently. This type of analysis is possible, because objects in each datatype may contain pointers to objects in other (lower level) datatypes, as well as to objects within the same datatype.

In all four scenarios above, the data set to which a job refers to can be either formed by "concrete" data products, or by "virtual" data products, to be created from lower-level data products before being analysed by the job.

We are considering the case where a physicist user of the Data Grid is entering an analysis job to be performed on a particular data set (AOD, ESD, Raw Data, or a combination of them). The output of the job could be considered to be TAG data, or some other data format which is transparent to the Grid. To submit the job the user provides the following information:

[1]The input data set might contain specification of "virtual" data. Virtual data is materialized during analysis.

1. The input data set. This is specified by two parameters:
   - the input event set, E_S, that is, a set of *event IDs*;
   - the input data product set $DP_s = \bigcup_{e \in E_s} \bigcup_{i=1}^{n} dp_i(e)$ that contains the collections $dp_i(e)$ of the n data products that are used in the job for each event ID e.
2. A reference to a piece of C++ code, which is to be executed on the input event set. The code contains an algorithm, which is an atomic function for each event, and thus needs to be executed separately on the data for each event. (The C++ code represents the computational job to be performed. Note that we assume this code can be transported for compilation and computation anywhere on the Grid.)
3. A physical location to deliver the output data to. This location may be specified automatically by the local application, and would normally lie "outside" the Grid; i.e. it would exist either on the physicist's local PC or on a private server, which is not managed by the Grid. Thus the output data can be considered to be "invisible to the Grid".
4. (Optional) processing hints, that help the Grid scheduler to find out the best location where to run the job and the Grid optimisation services in their activities.

Note that the input data product set of a job is actually a super set of the real data product set that will be access by the job. In fact, the input data set may not be completely known a priori (before the job starts execution) because the job could need to access some data products based on the values of data products it has already accessed, and of course this is only known at run-time. We refer to this as the *navigational problem* between data products. In many cases (primarily in analysis type jobs) it is impossible to know precisely what data the job will access, or how long it is likely to run for.

Note also that the input data product set could by determined by examining the job code. This means that if some code analysis tool would be available specification of the input data product set could be avoided.

4 Optimisation opportunities for Analysis

We define *Grid Query Optimisation (GQO)* as the optimisation of the time and/or cost of execution of a Grid job submitted by a user while performing analysis. In the following the terms job and query will be used interchangeably.

We can consider query optimisation under various points of view:
1. ***User oriented optimisation (high performance computing).*** Each physicist would like to perform analysis on HEP events as fast as possible and possibly minimising the cost for execution of jobs that she submits to the Data Grid. The optimal situation for a physicist would be being the only user of the Data Grid, thus having all the Data Grid resources available.
2. ***Grid oriented optimisation (high throughput computing).*** Grid designers have a different perspective on the Data Grid. Their task is to define optimisation services that "make happy" all the users of the Data Grid, without favouring any of them (of course, different categories of users can be given different priority of use of the Data Grid). Thus, collective optimisation should be taken into consideration, trying to maximise exploitation of Grid resources while maintaining acceptable time/cost for execution of single physicist jobs.
3. ***Site oriented optimisation.*** Administrators of Data Grid sites would like to decide upon local policies of use for resources at that site. These policies will influence the usage of the site by Grid and non-Grid jobs.

The optimisation services provided by the Data Grid should be designed to take into consideration all three of these perspectives and the "right" trade-off between them. We can consider two kinds of optimisation:

- ***Short-term optimisation of user queries***. This optimisation can be activated whenever a query is submitted to the Data Grid and aims at minimising the execution time and cost of the job. Optimisation is based on the information specified by the job: input data set and job code. We identify some possibilities for optimisation.
  1. ***Job decomposition.*** A job usually operates on a set of events independently. This means that a job can be decomposed as a set of sub jobs, whose code corresponds to the code (or part of the code) of the main job, and an aggregation job, that executes on the output data set of the sub jobs. Job decomposition can be performed on the basis of where the input data products are stored in the Data Grid, or on the structure of the job code.
  2. ***Job execution time estimation.*** The user could exploit this information in order to decide whether or not to submit a certain job to the Data Grid, according to how long she is willing to wait for job completion. Execution time execution can be based on both the location of data and/or code execution time estimation for each of the sub jobs that compose the job.
  3. ***(Sub)Job dispatching.*** Where to dispatch a (sub)job is another important issue to take into consideration for GQO. The optimum location for the execution of a job depends on the location and current status of both the required data and the required computation resources. A trade-off between data optimisation and computation optimisation is important to assure Grid oriented optimisation. As far as data optimisation is concerned, there are two important aspects:
     - *Data selection*. The same input data product for a job could be replicated in several files, placed in different locations on the Grid. The most convenient file copy should be selected for use by the job.
     - *Data relocation*. The relocation and replication of files inside and between sites must be performed in an optimal manner.
  4. ***During execution optimisation.*** When a job requests a set of data that is not available on the site where the job is running, a decision has to be taken on how to optimally access the missing data. Therefore, data selection and relocation could also be performed during the execution of the job, to meet the optimisation needs that arise due to unforeseen data requirements.
- ***Long-term optimisation of use of Data Grid resources.*** As previously stated, the optimised use of Grid resources improves the overall performance of the Data Grid and thus, on average, the performance of jobs submitted by single physicists. Long-term optimisation can be based on statistics on the past use of the Data Grid and forecasts of its future use.
  - *Replication.* A possibility of long-tem optimisation is to set up a suitable replication policy for files stored in the various sites of the Data Grid. For example, suppose that several jobs submitted to the Data Grid from sites placed in the same area have been accessing much the same set of files. Then, an immediate optimisation is to replicate those files in a site easily accessible from the sites in that area. Analogously, if a set of jobs to be executed in the future in a certain area of the Data Grid are going to use a similar set of files, these files could be replicated in advance and stored in sites included in that area.
  - *Reclustering.* Input data products of a job might be sparsely spread over several large files. Reclustering them into a smaller set of files prior to their analysis could improve execution time of the job, which would fully exploit the content of the files. A possibility is to set up some *reclustering policy* for data product. For example, if several jobs are going to access almost the same set of data products in the future, it could be convenient to store these data into the same file (or a set of files).

5 Grid Query Optimisation within a Typical Grid Architecture

In this chapter we present a brief overview of a Grid Architecture as partially defined by [ATF] in order to identify typical "Grid Query Optimisation" (GQO) tasks. Our aim is not to define an explicit "Grid Query Optimisation Service" per se, but rather to discover which services will be required in order to "optimise" data access in a Grid environment.

5.1 WP2 View on the Grid Architecture

The proposed Grid architecture is shown in figure 3. The shaded shapes represent services already existing (or mentioned) in [ATF], while the white shapes represent new services or parts of services, which are as yet not mentioned in [ATF].

The following is a description of the different services provided within each layer of the architecture. Each layer of the architecture is described in a different section.

5.1.1 The Local Application Layer

The top layer of the architecture, the "Local Application Layer", exists outside of the Grid infrastructure. It is in this layer that TAG data analysis will most likely be performed. TAG data will be stored on locally owned storage devices, using a locally managed database system. The optimisation of TAG data analysis is also part of the task "Grid Query Optimisation", and will be achieved through the *Bitmap Indexing* of TAG data values [SDH+00/Sto01]. Once the physicist has exhausted the local supply of information, he begins to analyse on lower level Grid managed data, by submitting a job to the Grid via the local Grid Client.

5.1.2 The Grid Application Layer

The applications that reside in the "Grid Application Layer " act as a means of interface and coordination between the local applications, which reside on a scientist's local PC/server and the services, which are provided by the various components of the Grid. Thus the main task of Grid applications is to reformulate the "user jobs" submitted by the local applications into "Grid jobs", ready for global execution. Such Grid job definitions must be free of any application domain specific semantics and all data must be described as a file, file collection or "data collection"[2] abstraction. Part of the process of job reformulation, therefore, may involve the remapping of the required event objects into the (possibly multiple) sets of logical files / data collections, which contain them.

[2] We define a "data collection" as a collection of data in a database, (take for example AOD version 2, objects 1 to 10, for the event range 1 to 10000). This abstraction is important for the case where a particular database implementation "hides" the file level from the application (i.e. where the mapping between data objects and files is unknown to the application and may change between sites). One can consider a "data collection reference" to be a query to the database implementation, which will then return the required data. (Note that we place no limitations on the syntax or semantics of such a reference, requiring only that it uniquely describes the data to which it refers. The reference would in general be both database implementation and data model dependent.)

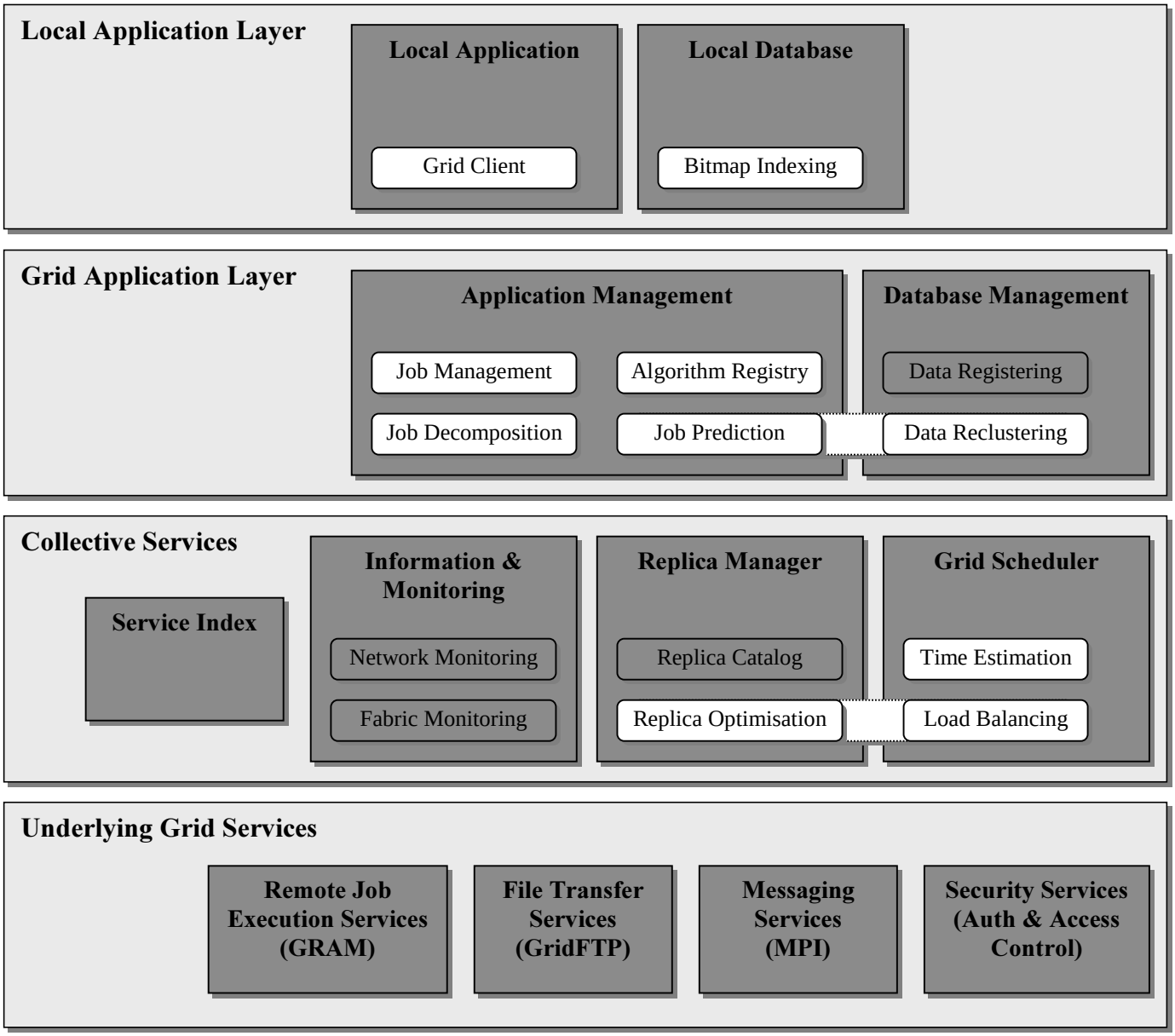

Figure 3. WP2 View on the Grid Architecture.

Application Management

Each application will most likely have some sort of internal *Job Management* system, making decisions regarding which jobs are submitted to the Grid and at which time. Such decisions would be made based on application quotas, user priorities, expected run-times, and so on. (The idea here would be to stop "small time users" from unwittingly submitting very large jobs to the Grid, which drain its resources unnecessarily.) The *Job Management* system would also be responsible for scheduling large production type jobs. However, much of the responsibility could go directly to the *Grid Scheduler*, which still needs to be discussed with WP1.

This application level *Job Manager* will submit each job to the *Grid Scheduler* in the form of a well-defined "Job Description Language" (JDL, see [WP1]). There is need, therefore, for *Job Decomposition* functionality within the *Application Management* system, so as to reformulate jobs into (graphs of) atomic computation steps ready for submission to the Grid. Only the applications themselves can provide enough information to fulfil such a decomposition task. One architecture (not necessarily advocated here) would be to define a standard interface to a *Job Decomposition*

service, which some/all Grid applications could implement, such that the *Grid Scheduler* could access decomposition services at execution time. Another approach would be to decompose all jobs completely before passing them in JDL form to the *Scheduler*.

If we are going to consider the possibility of optimisation through the on-demand realisation of (virtual) input data sets (CMS specific), then the Grid needs to know what algorithm needs to be run on what (low-level) sets of data in order to generate the required input data for the job to be executed. In this case there are two possibilities for passing the required information (a new JDL job definition for the pre-processing job) from the application to the Scheduler:

1) The submitting application passes the information to the scheduler as part of the job description.
2) The scheduler calls methods of a standard interface, which is implemented by the application, to discover what algorithm to run in order to generate the required input.

In either case there is need at the application level for an *Algorithm Registration* service. The aim of this service would be to provide a listing of the algorithms required generating files containing the higher level data (AOD or ESD) from files containing lower level data (ESD or Raw data). (E.g. files containing events of AOD version "X" can be created from files containing ESD version "Y" using algorithm "A".)

We also propose the need for a *Job Prediction* component within each Grid application. Such a component would be responsible for collecting statistics on the execution of Grid jobs, and for making predictions based on those statistics in terms of:

1) Which logical files / data collections a (analysis) job is likely to require.
2) How long a job is likely to run for.

The collection of statistics for the first prediction would require that database access to files/data collections was logged in some way, and that the log information was made available to the *Job Prediction* component. (This may be done either by the Grid intercepting calls to the database, or by the database itself logging its own activities.) Assuming the gathering of statistics is possible, why would such predictions be important, and why wouldn't the information be known already? In fact, in many analysis type jobs, it is impossible to know a priori what data the job will access, or how long it is likely to run for, due to the so-called "navigational problem". Estimations made by a *Prediction* unit would then be used by the *Scheduler* to optimise the execution of the job. Since the similarity between different jobs can only be assessed on a semantic level (i.e. by the application submitting the job), the *Prediction* unit needs to be co-located with the Grid application. The unit may also use other information, such as the physicist's user profile, or processing hints the physicists have given, to make predictions for job execution. As was the case for the *Job Decomposition* system, there are two mechanisms by which the prediction information can be passed from the Grid application to the *Scheduler*. The first option is to include such information as hints in the JDL (i.e. to generate all the prediction information a priori). The second option is to define an interface for the *Prediction* unit, which would be used by the *Scheduler* to gain predictions if and when required.

Database Management

Grid *Database Management* is another important function that needs to be provided at the Grid application level. (Particular database implementations are both the choice and responsibility of individual virtual organisations, i.e. they will not be managed as part of Collective Grid Services.) Two optimisation opportunities exist for optimising the distribution of data in database resources on the Grid. The first opportunity is to re-order data within a database on a particular site, and the second is to replicate data between databases on different sites. The former involves the re-clustering of information within and between data containers (such as files). This kind of

optimisation can only be performed at the application level, because it is at that level that the data schema is known. In order to perform such optimisations automatically, (as proposed in [Hol98]), statistics on data access need to be collected. These statistics are primarily the same ones as used in the Prediction unit, hence the link between this unit and the *Data Reclustering* unit in figure 3.

The second database sub-service shown in the architecture diagram is that of *Data Registering*. To understand the need for a special registration service within the *Database Management* system, we need look at the case where data is to be replicated from a database on one site to second database on another site. Since jobs access the data via a database API, one cannot simply copy the appropriate file from the first to the second, and expect that a job executing at the second site will be able to locate the data. Instead the data (file) must be copied and then registered with the database catalogue at the second site. I.e. the registration module provides support for the importing of files into a database. Another Grid project called Grid Data Management Pilot [GDMP] is presently adding such functionality to both the Objectivity and Root object database implementations. In the case where a relational or object-relational database system was to be used, export functionality as well as import functionality may also need to be provided by the registration unit. (This is because no direct/explicit mapping would exist in this case, between data/table descriptors and the files that contain the data.) The story is even more complicated for any data that is not "read-only" in nature, as the registration unit may then be responsible for ensuring that changes to the data at one site are correctly propagated to the other sites containing replicas of the data. (Note that such "synchronisation services" have also been proposed as part of the functionality of the *Replica Manager*. In that case the services ensure bit-wise equivalence between data files, rather than "information equivalence" between data collections.)

5.1.3 The Collective Services Layer

The collective layer represents the set of high level (but generic) Grid services offered to Grid enabled applications. The services provided in this layer can be roughly separated into three areas, information (monitoring) management, data management, and computation management, corresponding to Work Packages 3 to 1 respectively. A *Grid Services Index* will also be available to help applications find the services they require.

The Information and Monitoring Service

The *Information and Monitoring Service* will aggregate information provided by monitoring services such as the *Network Monitoring Service* (monitoring traffic on Virtual Private Network links between sites) and the *Fabric Monitoring Service* (monitoring the load on computational resources at the different sites).

The Replica Manager

The *Replica Manager* provides a global view on the data stored at the different sites on the Grid. This view is provided by the *Replica Catalog* module. The *Replica Manager* also provides replication and consistency services on this data at the file and "file collection" abstractions.

Replica Optimisation is another functionality of the *Replica Manager*. The aim of the *Replica Optimisation* module is to automatically replicate data throughout the Grid in such a way as to minimise the total cost of data access for all the jobs executing on the Grid. This is part of what we called "long term optimisation". Within a site the module might control which files remain staged on disk and which ones are relegated to tape storage. Between sites the module would control which files are replicated and which ones aren't. The question then becomes, how does *the Replica*

Optimiser decide which files to replicate and which ones not to? I.e. how does the *Optimiser* know which files will be in demand on the Grid and which ones will not? It could do this by:

- collecting its own statistics; (It would be too late to collect information at this point, if the mapping between data collections and files is not constant across the Grid, i.e. if internal data reclustering at each site causes data to be stored differently at each site.)
- asking the *Grid Scheduler*;
- accessing standard services of the application layer *Job Prediction* module; or by
- receiving hints directly from the application. (The application level *Job Prediction* unit could assign importance levels to different logical files / data collections, which the *Replica Optimiser* would then to decide where and how many replicas to create.)

A second function of the *Optimiser* (a kind of short-term optimisation) is to find the "best" replica of a file when the Scheduler demands the file. In this case the *Optimiser* can decide whether it should create a new replica of the file locally, create a temporary copy of the file locally, or (possibly) open the file on the remote location for remote access. In the case where it decides to create a local replica or copy of the file, the *Optimiser* must use an interface provided by the *Data Registering* module of the application level *Database Management* system to import the new replica/copy into the local database implementation.

The Grid Scheduler

Other important functions of the *Optimiser* are to supply the *Time Estimator* module of the *Scheduler* with time estimates for the retrieval of files, and to negotiate with the *Load Balancer* to determine the best location for executing a given job, based on both the data and computational requirements of the job. The negotiation with the *Load Balancer* may require close coupling between the two components (as shown in figure 3).

The *Scheduler* provides high level scheduling services to Grid applications such that the applications needn't know where and how to schedule their work on the Grid, but can simply define the constraints for running a job (such as the amount of memory it requires, the input data collection it needs, etc.) and allow the Grid to schedule it for them. The applications can then view the Grid as a single enormous computation and storage resource (see section 5.2 for a discussion). One of the functionalities that needs to be provided by the *Scheduler* is that of *Time Estimation* for the execution of a Grid job. A time estimate is used by the Grid application or the physicist to decide whether or not to run a job. The *Time Estimator* uses information from the *Job Prediction* module, the *Information and Monitoring* services, and the *Replica Optimiser* to calculate an approximate time for job execution.

The *Load Balancing* unit is responsible for the actual scheduling of jobs to different sites on the Grid. It takes as input a decomposed job from the Grid application, and negotiates with the *Replica Optimiser* to discover the best location for job execution, based on the availability of both data and computational resources. (The *Replica Optimiser* uses information from the monitoring services to compare the costs for replicating data to different sites.)

5.1.4 The Underlying Grid Services Layer

The Grid services on the bottom level of the diagram represent the basic and essential services required in a Grid environment. These services include the ability to schedule jobs on remote clusters, to transfer files efficiently between sites, to pass messages between processes, and to control authorisation and access to files and services. One can view the services as Grid versions of standard LAN services, such as those offered by the UNIX operating system. These services will most likely be provided in EU DataGRID project using modified Globus components.

5.2 Separating Grid Information from Application Information

One of the most important aspects of the architecture outlined above, is the separation it seeks to achieve between application specific information (at the "Application Layer") and Grid specific information (at the "Collective Layer"). The two types of information are defined as follows:

- "Application specific information" which includes domain specific semantics (such as the concept of a "collision event"), and domain specific data structures (such as a particular database schema used for storing these events). It is the primary job of the "Application Layer" to transform "user jobs", which are described in terms of these semantics and structures, into "Grid jobs" which are described using generic Grid concepts (e.g. to map object identifiers to logical file identifiers or "Data collection Id's", such as a set of AOD events).
- "Grid specific information" is all types of information required for the co-ordination and management of distributed computational and data resources. This information includes such things as the location and state of available resources, and how to interface with them.

Separating these two types of information provides important advantages:

1. It simplifies the job of the application programmer, who is not an expert in resource scheduling for a distributed environment, and should not need to become one. The Grid application should "view" the Grid as a single, extremely large data storage and computation resource, to which it submits jobs for execution.
2. It enables the reuse of Grid middleware (optimisation and scheduling) components, rather than requiring that they be reconstructed for each application domain.

It is important to note that much of the information (such as the internal structure of files, and the various steps in computational processes), which is needed to perform optimisations on the Grid (such as locating the best data resources for a particular job, parallelising computation across sites, or re-clustering data to better suit demand) is only available at the application level. Such information is encapsulated in the architectural components Job Prediction, Job Decomposition, and Data Reclustering. Some optimisations, such as the reclustering of data products, are better performed at the application level, while others such as scheduling and data replication should be performed at the Grid (collective) level. As outlined in the discussion of the architecture, different mechanisms exist for passing the required information from the application to the collective layer. One such mechanism is to define interfaces to be implemented by the applications and used by the Grid services. A second option is to pass information from the application layer to the Grid services using the JDL.

5.3 The Task of Grid Query Optimisation within a Typical Architecture

The primary task of GQO is to build a major part of the Replica Optimiser module of the Replica Manager. In this section we look more closely at the required functionality of such a module. The functionality can be viewed in terms of the five "services" it offers:

1) ***Data Access Time Estimation.*** Aids the Time Estimation unit of the Scheduler to calculate approximate execution times for jobs that might be submitted to the Grid. The Replica Optimiser accesses the Network Monitoring Service (to discover the current network bandwidth situation), a Storage Device Monitoring Service (to discover the current load on the devices), and the Replica Catalog (to discover the amount of replication of the required files), so that an estimate of the cost of data movement can be returned to the Scheduler.
2) ***Pre-Execution Optimisation.*** Aids the Load Balancing unit of the scheduler to make scheduling decisions, (i.e. help the Load Balancer decide on which site to run a job). An optimised scheduling decision should take into account both the cost of data movement

between sites and the computational load on sites. A trade-off between data optimisation and computation optimisation will be achieved through the use of a negotiation protocol between the Replica Optimiser and the Load Balancer. (This negotiation/interaction protocol is still to be defined.)

3) ***During Execution Optimisation.*** If a job requests a set of data at a particular site, and the data is unavailable at that site (e.g. the database method returns an exception, and the exception is caught by the Replica Optimiser), then the Replica Optimiser needs to make a decision on how to optimally access the missing data. The Optimiser can then choose between possibly five options:
   - Open data for remote read on a site with a fast network connection.
   - Copy the data locally, register and create a "permanent" replica of the data.
   - Copy the data locally, register a "temporary" replica and de-register it subsequently.
   - Create the data on-demand using "lower level" data, (CMS specific).
   - Ask the Scheduler to reschedule and restart the job on another site (e.g. a tier 1 or 0 site).
4) ***Post-Execution Optimisation***. The aim of this optimisation is to automatically distribute (create replicas of) the Grid managed output files created by jobs running on the Grid. (Such output files are primarily only created by production type jobs, which are not the focus of our work). In making decisions on to what extent to replicate the output files, the Optimiser relies on hints from the user submitting the job, as well as heuristics information on the use of similar sets of data.
5) ***Offline Optimisation***. The aim here is to monitor the usage of logical files / data collections on the Grid as a whole and try to match the supply of replicas to the demand for them.

As well as creating the major part of the Replica Optimiser, the GQO task involves helping the applications groups (Work Packages eight to ten) to create the Grid Application Layer services they require in order to make optimal use of the Grid. The services of interest include the Job Decomposition, Job Prediction, and Data Registering service. The assistance would be in terms of defining standard interfaces to these components and possibly helping to create generic code for use in each application's implementation of the components.

6 Interaction of Services for a Particular HEP Use Case

We now look at a particular HEP use case described previously in the context of the architecture described above.

At the start of the use case, the physicist performs "cuts" on the entire data set, by specifying that she is only interested in those events for which certain conditions on TAG attribute values hold. The local application sends these "cut predicates" to a local (bitmap) indexing system, which returns a list of events adhering to the selection. The physicist then performs some sort of statistical analysis on the events returned by the indexing system, studies the results and repeats the process. Since TAG data is mostly stored locally, the Grid is probably unaware of the analysis being performed by the physicist. After exhausting the information available in the TAG data, the user writes analysis code and submits it as a job to the Grid. The local application sends this code, along with the names of the AOD objects of interest and an output location for the results of the computation, to a Grid Application Layer "HEP Application".

The Grid level application reformulates the request into a job capable of execution on the Grid. It does this by mapping all of the requested AOD objects to a set of logical files or to a data collection description. It may also decompose the job into smaller subjobs via the Job Decomposition service. Having reformulated the request, the application then submits it to the Grid for an estimation of execution time and cost, by sending a request to the "Scheduler" which provides an interface to the

services of the “Time Estimator". The Time Estimator requests information of the Job Prediction module (data and time requirements of the job), the Monitoring Services (current computational load on the Grid), and the Replica Optimiser (cost of data retrieval) to calculate an approximate time for job execution. When the application receives the execution time estimate, it uses that information to schedule the execution of the job on an application level, based on the cost of the job, the user submitting the job, the status of its job queue, etc. Of course the Grid application could also decide to refuse to schedule the job, or just to send the time/cost estimate back to the local application for approval. For more precise estimates, the Time Estimator could also contact the Load Balancer (see below).

In order to decide on which site to schedule the job, the Load Balancer enters a negotiation process with the Replica Optimiser. The Load Balancer first uses the constraints given in the JDL job description (such as memory requirements, software library availability, etc.) to select a set of “possible sites” for job execution. It then requests information from the Fabric Monitoring service and the Job Prediction service in order to calculate the computation cost for job execution on each of these possible sites. It also sends the list of possible sites to the Replica Optimiser, so that the optimiser can use information from the Network Monitoring service and the Replica Catalog to calculate the minimum cost of staging/using/creating the required data at each of the possible sites. The Replica Optimiser then provides this information to the Load Balancer, so that the Balancer can schedule the job at the site with the lowest overall cost. (The “negotiation” between the Load Balancer and the Replica Optimiser given above is highly simplified, and implies a “full search of the available search space, to achieve a global minimum”. Such an exhaustive search may not be possible or simply inefficient, in which case more complicated forms of negotiation may be required. Further discussions with Work Package one are required here.)

The Load Balancer then asks the Replica Optimiser to stage all required files to the site selected for execution. Once the files have been staged, the Load Balancer dispatches the job for execution to the selected site, using the Remote Job Execution service.

During job execution, the navigational access within the job causes it to demand a set of data that is not available in the local database. The Replica Optimiser catches the exception thrown by the database, and remedies the situation by either opening the data for WAN access on a remote site, copying the data locally to create a temporary/permanent replica, or asking the scheduler to stop the job and restart it (from the beginning) on another site. (The latter might be the case if a job on a tier 2 site say, wants to start accessing Raw data, in which case restarting the job on a tier 0 or 1 site, would probably be more advisable than moving Raw data to the tier 2 site.)

Once the job has reached completion, the Replica Optimiser is responsible for deleting any temporary replicas created for the execution of the job. (Deletion also implies the “deregistering” of the data from the local database).

7 Simulating Grid Query Optimisation

Designing query optimisation strategies for the Data Grid is certainly not an easy task. In order not to do a pure theoretical and speculative research, we adopt an experimental approach. Therefore, our aim now is to simulate the Grid environment, and to attempt to optimise data access within such a simulated environment. More specifically, the simulator has the following goals:

- To build a system that can realistically simulate a Grid environment, in which multiple autonomous resources must be managed coherently. (The model will include simulations of the main parts of the architecture components discussed in section 5.1)

- To simulate the optimisation processes of the Replica Optimiser within such a realistic environment. These processes will be defined based on the optimisation opportunities described in Section 4.
- To test different algorithms for making such optimisation processes, based on their ability to optimising globally the use of data and computation resources on the Grid.

We will primarily focus on processes regarding the automated creation and deletion of data replicas between sites. Afterwards, we will focus also on the other optimisation opportunities

Additional goals in building the simulator include:
- To study the effects of local policy decisions on the overall working of the Grid system.
- To validate the design of the different Grid components

It is beyond the scope of this document to describe the specifications of the simulator. Such specifications will be given in due course in another document. At this time a preliminary Grid environment simulator has already been built using Belief Desire Intention (BDI)-based software agents. A first prototype of this simulator was presented by [BCN+01]. We are presently investigating the possible integration of this environment with the MONARC simulation tool [Mon]. Future work will involve the improvement of the model, followed by the incorporation of the Replica Optimiser element and the testing of optimisation algorithms.

8 Acknowledgements

Our special thanks go to Koen Holtman, Peter Kunszt and Heinz Stockinger for useful discussions and contributions to the use cases and the architectural part of this document.

9 References

[ABH+01] Paul Avery, Julian Bunn, Takako Hickey, Koen Holtman, Miron Livny, Harvey Newman, CMS Data Grid System Overview and Requirements (DRAFT V9), http://kholtman.home.cern.ch/kholtman/

[ATF] Architecture Task Force, http://Grid-atf.web.cern.ch/Grid-atf/documents.html

[BCN+01] Paolo Busetta, Mark Carman, Michele Nori, Luciano Serafini, Floriano Zini, Using BDI Agents for a DataGrid Simulator, http://sra.itc.it/events/agents4grid/program.html

[Hol98] K. Holtman, Clustering and Reclustering HEP Data in Object Databases, Proc. of CHEP '98, Chicago, USA.

[GDMP] The GDMP Project, http://cmsdoc.cern.ch/cms/grid/

[Mon] The MONARC Project, http://monarc.web.cern.ch/MONARC/

[SDH+00] Kurt Stockinger, Dirk Duellmann, Wolfgang Hoschek, Erich Schikuta. Improving the Performance of High Energy Physics Analysis through Bitmap Indices. 11th International Conference on Database and Expert Systems Applications, London - Greenwich, UK, Sept. 2000. Springer-Verlag.

[SSS+01] Luciano Serafini, Heinz Stockinger, Kurt Stockinger, Floriano Zini, Agent-Based Query Optimisation in a Grid Environment, IASTED International Conference on Applied Informatics, Innsbruck, Austria, February 2001

[Sto01] Kurt Stockinger, Design and Implementation of Bitmap Indices for Scientific Data, to appear in International Database Engineering & Applications Symposium, Grenoble, France, July 2001, IEEE Computer Society Press.

[WP1] Workload Management Work Package, http://www.infn.it/workload-Grid/documents.htm