LHC Grid Computing Project

COMMON USE CASES FOR A HEP COMMON APPLICATION LAYER FOR ANALYSIS
HEPCAL II

Document identifier:	**HEPCAL II**
Date:	**29/10/2003**
Authors:	**D.Barberis (INFN),** **J.Beringer (CERN),** **N.Brook (Bristol),** **P.Buncic (CERN/IKF Frankfurt),** **F.Carminati (CERN),** **R.Cavanaugh (UF),** **P.Cerello (INFN)** **F.Donno (CERN/INFN),** **D.Foster (CERN),** **C.Grandi (INFN),** **F.Harris (CERN/Oxford),** **L.Perini (INFN),** **A.Pfeiffer (CERN),** **R.Pordes (FNAL),** **D.Quarrie (LBNL),** **A.Sciabà (CERN/INFN),** **O.Smirnova (CERN/Lund),** **J.Templon (NIKHEF),** **A.Tsaregorodtsev (IN2P3)** **C.Tull (LBNL)**
Editors:	**F.Carminati (CERN),** **J.Templon (NIKHEF)**

Document status: **Version v1.2**

Abstract: This document overviews analysis activities and possible execution models, and identifies common use cases for LHC applications to use grid services for data analysis.

In addition the characteristics of interactive vs. batch grid activities are presented, and some special system requirements on middleware are itemised, with key areas being the support of provenance, general reporting facilities, 'persistent' interactive work, and analysis software development.

Issue	Date	Comment
0.0	12/05/2003	Output of HEPCAL week
0.1	21/05/2003	First revision by F.Carminati for discussion within GAG
0.2	7/7/2003	Second revision for discussion
1.0	9/9/2003	Version for feedback from the experiments. All the input received has been analysed and, where possible, introduced. This includes input from CS11.
1.1	30/10/2003	Final version for approval.
1.2	7/11/2003	Minor editorial correction. Version delivered to SC2.

CONTENT

1   INTRODUCTION

The mandate and objectives of the original HEPCAL RTAG were the following:

- Identify and describe a set of high-level use cases of grid technology common to the four experiments;
- Possibly identify and describe which use cases will be specific for the different experiments;
- Identify a set of common requirements for grid middleware;

The resulting RTAG contains the detailed description of use cases that are executable in the distributed "grid" environment. We have simply focussed on what a physicist will actually need to do. These use cases should serve to the middleware developers (both in US and in Europe) and to the experiment framework developers to guide their work.

Almost one year after the publication of the original HEPCAL RTAG, two requests were expressed by the community of physicists using the GRID:

1. Revise the original HEPCAL document both modifying some elements and adding what was felt as missing information;
2. Produce a new document focussing on analysis, which was somewhat neglected in the first HEPCAL.

The purpose of the present document is to respond to the second point, much in the same way and along the same lines of the original HEPCAL document. A revision of the original HEPCAL document is also under way, and it will be soon released under the name of HEPCAL-prime.

As in the original HEPCAL, here we chose not to address the question of whether the needed functionality comes from the grid middleware or the experiments' frameworks.

The end product should help in the development of a common set of services for the four LHC experiments to be used on the timescale of the LHC exploitation for the analysis, both batch and interactive of the LHC data. While we have constructed the analysis scenarios in a quite general way that should cover the majority of the situations, we are aware that other models are possible, that only partially match to the ones described. This document does not aim at being an exhaustive description of the analysis activity of the four LHC experiments, but rather at describing what is the common underlying structure of most analysis activities and which are the common requirements that can be derived by this.

The present document should be seen as an extension of the original HEPCAL, and therefore we decided not to repeat the basic definitions already contained in HEPCAL. Detailed knowledge of HEPCAL-prime is thus a prerequisite for reading this document.

2   EXECUTIVE SUMMARY

This document explores High Energy Physics (HEP) *analysis* activity within the framework of what is commonly called grid computing. A previous document, hereafter referred to as HEPCAL (HEP Common Application Layer), laid the foundation for much of what is discussed here. That document covers many topics essential to HEP computing. We make much use of those topics, and related concepts, in the current document. It is *particularly important* in many cases to understand the HEPCAL nomenclature – HEPCAL makes a serious attempt to provide unambiguous definitions of things like datasets, jobs, catalogues, *etc.* If the reader sees something in the current document that seems strange, it is probably worthwhile for him to consult the original HEPCAL document and make sure he has understood what was really meant by the terms used. A large fraction of objections to HEPCAL, and early versions of the current document, vanished when the terminology was explained. The authors apologize for this situation, but it cannot be helped; consult members of five different HEP experiments and ask them to define a "dataset" and you will get seven different answers.

With this warning in hand, let's turn to the specifics of *analysis* on the grid. Firstly we can define analysis by stating what it is *not.* The original HEPCAL document dealt mainly with issues relevant to organized activities such as Monte-Carlo productions (generation of a large number of simulated HEP event) or reconstruction (construction of "physics quantities" from a large body of real or simulated event data). These activities tend to be planned and scheduled, and they also tend to access (or create) a large set of input (or output) data whose identification and physical location are known well before the task is started. Furthermore these activities tend to use some "official" release of software that is likely to be pre-installed at many computer centres.

In contrast, analysis jobs tend to lack one or more of these characteristics. The first is organization; individual researchers often carry out analysis, and the work is done whenever the person concerned has the time to do so. Secondly is the data organization; the physicist doing analysis normally defines the input data by asking a question, such as "give me all the data taken during a certain time period which has the following characteristics …" The physicist may have no idea where these data are physically located, nor whether the files containing the data are all located at one storage site or scattered all over the world. Finally during analysis, the physicist will likely replace some (or all) of the "official" software with his own algorithms; perhaps he is performing analysis to try and improve the official software, or perhaps the official software lacks capabilities needed for the task being attacked.

Analysis on the grid brings new requirements for data access permissions. Permissions need to be available for individual users to *e.g.* delete or create their own data, but not those of others. Also the concept of group-level permissions needs to be considered, as each of the analysis groups will need to have group-member-level permissions. The concept of roles must be extended as well taking into account the existence of analysis groups.

Since the analyses tend to be less organized and have several "non-standard" elements, it is important for the physicists to keep track of what was actually done. This is especially important for HEP computing since major discoveries are often based on finding a few needles in a giant haystack; such results must be carefully scrutinized. Analysis then involves some method for recording the *provenance* (from Latin *pro* = from and *venire* = to come, i.e. where something comes from) of data products, meaning the path followed from raw data to the plot on the screen. It seems essential, given the amount of information that is potentially important for understanding how a particular physics result was reached, that the grid system makes some allowances for making such an audit trail.

Another feature of analysis is the proliferation of software versions. For production-only grids, a few "official" software releases are likely to be sufficient. With many users (and many groups) doing analysis (part of which is iterative improvement of the software being used), it is likely that some system for management, distribution, and automated installation/purging of software versions would be of great benefit to the users.

A final element that must be considered is *interactivity.* While a computing team might be quite happy to run a reconstruction which takes weeks to process several terabytes of data, an individual user will

often wish to iterate his analysis task as algorithms and parameters are tuned and perfected. Turnaround time becomes a more important consideration. This, in turn, places more demands on the monitoring tools available to the users. An example which is frequently mentioned is an "estimation facility" which when submitted with a job description, gives an estimate on how long the task will take to complete given the current state of the grid resources, user's quota and priority, *etc*. Turnaround time is not the only consideration; a related question is how well the user's working environment is coupled to the grid. Can the user mount the "Grid File System" just as she might a windows "network drive"? Is there some logbook facility from within which a user can inspect what she has done, the status of various running analyses, etc? These issues are important and we did not have enough time to address them carefully.

3   THE ANALYSIS ACTIVITY

In this section we attempt to provide a sketch of what steps are involved when a physicist performs data analysis within a grid environment. "Physics data analysis" refers to the iterative, exploratory activity during the later stages of a physics experiment in which the physicist attempts to transform reconstructed event data into publishable scientific results, via a set of algorithms. The results are quantities derived from the experimental data that are accumulated in histograms, tables or other statistical entities to provide evidence for the subject of the investigation.

We start by presenting a very high level scenario illustrating the general flow of a data analysis activity, which could be batch or interactive.

An experimental collaboration (a Virtual Organisation or VO in grid jargon) has reconstructed raw data and Monte Carlo data generating a set of DSs that contain the events to be analysed. This is done according to the experiment policy by the production process that is explained in the HEPCAL document.

A member of an experimental collaboration wishes to become involved in physics analysis activities. As a pre-requisite he needs to:

1. Register as a user of the computing environment of experiment.

2. Make sure that the analysis activity he intends to do is compatible with the resources (computational, network or storage) he is entitled to use. Physics analysis often happens within the context of a physics analysis working-group (PWG), so one way for the user to gain resource access is for her to become a member of one or more working groups, or to create a new one.

After this, the analysis activity consists of some subset of the following:

1. Perform queries (possibly on the Dataset Metadata Catalogue(s) [DMC] defined by HEPCAL) to determine which DS(s) may contain data meeting their criteria (physics channels, processing runs, type of event data, etc.). Note that this set of input DSs may be already known from a previous query, and therefore this step is not performed every time.

2. Query the input set of DSs, selecting the events or event components of interest, using event-level metadata. The result of such a selection is a list of event component identifiers (experiment-dependent) that allow retrieving the relative event subset within the associated DSs.

3. Optionally save for further use the results of 2 via a procedure similar to that defined in step 5.

4. Perform analysis activity, looping over the event components selected by step 2 above. This activity may involve additional filtering (selecting a subset of the original event set from step 3), reprocessing (augmenting or replacing components of the selected events), generation of new calibration and statistical information. The analysis code is in general some combination of "official" experimental code and new user code being developed for end user analysis.

5. Optionally save the results of 4 and publish them. The role the physicist is playing for this session will determine whether this publication is at the individual level, or is associated with a particular PWG. There are three modes of making the results persistent:

    a. "Create TAG DS". The new DS consists only of references to event components, with no copying of the event data being performed. This is equivalent to 3 above. This is sometimes called "shallow copy", i.e. pointers are copied and not what they address.

    b. "Create new re-clustered DS" The selected subset of input event components is copied to a new DS so that they are available more efficiently to further processing. This is also called "deep copy" because not the references but the information they point to is copied.

    c. "Create new DS". If components of the events are modified (or new components are generated), the user may want to save them to a new DS. Note also that this is the only possibility in the HEPCAL definition of DS, as these are write-once-read-many entities.

Jobs could be submitted for any internally consistent sequence of the above steps and the whole procedure may loop over these steps. Input files may not be known in advance and files may need to be opened dynamically. However the user should be able to evaluate (by using a system-provided estimation service) in some way the cost of this step, to be able to allocate grid resources to the job. Any time after the analysis is complete, the user may wish to inspect the provenance of the output for verification purposes.

A question that we have discussed at length, but on which we could not converge was "private metadata". There was a general agreement among the authors of HEPCAL-II that a user may want to associate her own metadata to the DS used in analysis, be those existing or newly created, as indicated above. However we were not able to go beyond this point and to identify common requirements about these "private" metadata.

Since the analysis is iterative, there may be lots of datasets left as by-products of the intermediate steps. These DSs will be created per node, physicist, physics channel, giving rise to a potentially large number of small files. The question of how to "purge" the unwanted DSs, or even all the intermediate ones when the result of the analysis is discarded, may became non-trivial. This may place extra requirements dataset-deletion functionality, perhaps in conjunction with the provenance information discussed below. Furthermore the existence of a large number of small datasets may have serious implications for efficient access to the data (of this more later in section 4).

3.1 PECULIARITIES OF THE ANALYSIS ACTIVITY

The scenario presented above illustrates many of the characteristics of analysis work that were presented in the executive summary. The main features that have to be kept in mind in the formulation of scenarios and in the derivation of requirements are:

1. Routine use of non-standard algorithms and user- or PWG-specific code together with "official" experiment software release.

2. Input DSs not necessarily known a priori.

3. The possibility to have a very sparse data access pattern (in cases where only a very few event components match the query).

4. Large number of people submitting jobs concurrently and in an uncoordinated fashion. The consequence is a chaotic workload increasing the demand on resource management services.

5. A wide range of user expertise and familiarity with the system.

6. A possibly significant proportion of "interactive" jobs (see Sec. 6).

7. Specific requirements on system response time rather than throughput. Ordered production tends to value total system throughput while response time is more important for end user analysis.

8. Many more "roles" involved in analysis.

9. Need for detailed provenance information. Overlooked in HEPCAL, however important now, as analysis involves longer chains of steps, each chain largely unique to the particular analysis.

10. Need for a resource estimator. As we said already, the number and location of the DSs involved by a query may be unknown a priori, and also the amount of resources needed by an analysis. It would be important to be able to run a pre-analysis that could estimate the relevant parameters of the real job, such estimated time-to-complete and resource consumption.

11. Complementary to the above requirement, it would be important to have a way to account and limit resource consumption or elapsed time for a given analysis.

4 ANALYSIS EXECUTION MODELS

The potential lack of data organization for analysis activities, as discussed in the introduction, can have a serious impact on the efficiency with which the data are analyzed. A naïve approach to the problem could result in very large latencies as data are accessed across the network, or even larger latencies and space-management problems as entire datasets are copied to the execution site, even though the job may use only a fraction of each dataset. We explored a number of execution models for grid analysis job in order to identify problems and possible solutions. These execution models in turn result in different conclusions on what services and capabilities are expected from the middleware and application layer respectively. This is further complicated by the situation where different communities give different definitions of the boundary between "middleware" and "application" or "experiment framework". In this section, we try to explore some of the possible execution models and of the related issues.

The high-level statement of the problem derived from the scenario presented in section 3 is:

- We have an algorithm that we would like to apply to a set of input data in a given environment.

- These input data may be explicitly specified as a set of DS's or selected by the job itself. In most cases this selection is done via a query: the set of data to be processed is the set of data that matches the query.

- The user provides the algorithm and the query to the workload management system.

- "Something happens" and at the end the user has access to the output of the algorithm.

Our task here is to identify ways in which "something happens", and to explore how much of this "something" is handled by the middleware and application layers, respectively. The discussions so far have identified three areas in which it seems necessary to explore the division of labour between the middleware and the application.

- Query execution.

- Workload distribution and execution.

- Data access.

It is clear that we need all three to efficiently support the range of analysis activities for the LHC experiments.

4.1 DEFINITIONS

In all of the discussion, we use the following definitions:

Event. The word "event" in general refers to the raw data associated with a specific trigger number, as well as all the entities derived from them. In the case of real data the derived data products can be associated with a single primary entity to be identified in the original trigger. For simulation the situation is still the same in theory. In practice due to the artefacts of the simulation technique, several primary entities can be at the origin of the same event, and some of them can be common to more than one event. However we may consider that, at a certain level of detail, our definition is valid. We indicate with the name of "event components" or "data objects" the raw data and any derived quantity. In general the event components may also contain metadata, describing e.g. the provenance of the data object (see sec. 7.1).

Metadata. "Metadata" refers to information such as:

1. Information on event components such as which sub-detector or physics channel the component belongs to.

2. Provenance information of datasets or data objects, describing the process of creating the object.

3. Bookkeeping information on data objects (date created, size).

Metadata exists at various levels within the experiment's persistence framework. Of particular interest in this document are the event level metadata and the dataset level metadata.

Provenance. A set of metadata that represents the way a dataset has been produced. See also 7.1 for an extensive discussion.

Input dataset. "Input datasets" (in the context of analysis) generally refer to any pre-existing combination of ESD, RAW, EMD, AOD, or TAGs from RECO (or re-RECO) needed by the job[1]. Of course there may be other required input datasets containing objects besides these (such as calibration information). Additional input is needed like the versions of the software and configuration used for the analysis. The provenance information needs to be recorded; depending on the implementation this information could be recorded in the Dataset Metadata Catalogue (DMC), in some system-provided provenance archival service, or recorded in the EMD.

Output dataset. "Output datasets" refer to AOD, TAGs, new or updated EMD information, histograms or statistical information produced by the job. There may be multiple output streams (representing different kinds of output information from the same job, e.g. a histogram stream plus one or more AOD streams).

Job. The definition of job is given in the HEPCAL document to which we refer.

4.2   SUPPORT FOR QUERIES BY COMMON LAYERS

As we said above, the input data may be the result of a query. The query can invoke middleware services and experiment dependent services. An English-language representation of a typical query might look like this:

```
All the data (or perhaps instead "at least 10,000 events" or "retrievable
within 2 days") from first quarter 2007, taken in trigger configuration
EFG, reconstructed with reco version 7.4 and calibration set 21.d3, for
which pT>150 GeV and a J/ψ was observed.
```

It will not be difficult to generate a query result that represents a very large dataset. This needs attention. Users need to be encouraged to think about the amount of resources that might be needed to process the query results. The system may set some upper limit on the size of the acceptable return; alternately this may be affected by user resource allocation policies elsewhere in the system. As we said in 3.1 it would be very useful to submit the query and get an estimate of the cost for the data access. If this is too high the user may change the query until its cost is acceptable or obtain permission to use the resources required and then submit the job. We do not discuss here the question that the query itself may have a cost in term of resources. The middleware should allow for an experiment-dependent mechanism that could prevent the selection of duplicate events and in general ensure that the selected set is statistically sound. Again we did not have time to elaborate on the requirements that this implies for the middleware.

The most naïve and straightforward way to implement such a query is for each experiment to have a gigantic table (like a classical DBMS) which has a structure similar to the following one:

evID	date	Trigconf	type	algversion	calibv	p_T	Tag particle	LDN	Object reference
1231	31/12/2006	EFG	AOD	7.3	21.d2	120	J/ψ	A1	Ccdfe
1232	01/01/2007	EFG	AOD	7.3	21.d2	121	K_L	B1	C45ea
1233	01/01/2007	EFA	AOD	7.4	21.d3	164	Bs	A2	E43fe

---

[1] These are common terms in HEP, but if needed definitions can be found in the HEPCAL document.

1234	01/01/2007	EFG	ESD	7.4	21.d3	79	None	C4	Ae236
1234	01/01/2007	EFG	AOD	7.4	21.d3	160	J/ψ	B3	910e3
1235	01/01/2007	EFG	AOD	7.4	21.d3	141	Λ	Fred	F3e12

This example is vastly oversimplified. If such a table were to exist, it would likely at least need many more columns, and even more likely would need to have a richer structure. We hope it suffices for illustration purposes.

The query described above will match only the next-to-last entry in this table. The last two fields are needed to enable the user program (in cooperation with the middleware) to access the matched event. The LDN is needed to supply to an "open" call, and here we explicitly assume that the middleware will assist the user. In HEPCAL we supposed to provide the LDN to a routine that would return a filename to be opened via a POSIX call. Alternative we could imagine to feed the LDN to a "gridified open", and get back a file handle or object from which we can read the bytes just as if the file were on a disk local to the worker node. The "object reference" is something the experiment software can understand and use to find the selected event inside the dataset referred to by the LDN. Examples might be a simple event number, a byte offset into the dataset plus the size to read, or a sort of object pointer – this is completely up to the experiment framework. The execution model for the job, at this level, would be:

1. Apply the query to the event table.

2. Take the list of object references so generated as input

3. Generate job(s) that access these objects and run the algorithm on them

4. Return the output.

Experiments are in general not enthusiastic about the use of such a table. First the table would be very large. Such tables would typically have on the order of 10^{12} entries per year. Furthermore the experiments were not planning to construct such a single flat table themselves, nor were they prepared to let a middleware layer provide and manage such a table on their behalf.

There was some interest in various hybrid approaches. Part of the query example above can be handled by facilities of the Dataset Metadata Catalogue as described in HEPCAL. Columns two (date) through six (calibration version) could be part of the dataset metadata, since within a single dataset the events will almost certainly have the same value for each of those columns. Therefore we decided it was interesting to consider a "split" query – one part that acts on a dataset-level catalogue, and another that operates essentially at the experiment-framework level. The split can be at an arbitrary level, which is advantageous since the various experiments have different plans for the Dataset Metadata Catalogue usage.

The experiments also were very interested in a hierarchical metadata space or, more generally, a clustered metadata space. Such a hierarchy can be an efficient way to cut down the search space compared to the flat-table order of 10^{12} mentioned.

The experiments agreed that however the list of event objects is returned, these event "identifiers" or "references" should be suitable for handing to the experiment framework, which can essentially just "open" them to get the bytes. There is no explicit requirement on the ordering of events returned back from a query.

The conclusion is that the support for queries will be split between the Dataset Metadata Catalogue (provided by the middleware layer as described in HEPCAL) and the experiment framework layer. The contents of the DMC will be experiment specific, but the way in which these queries are executed will be the same across all experiments. Typically this DMC-level query will result in a list of LDNs. Since the complete query is a logical AND of the DMC and the experiment dependent part, all DSs containing objects that match the complete query will be in this list, but there may be more. We do not require (since it appears impossible to guarantee) that all DSs matching this middleware-level query

will contain events matching the complete query. The approach requires that a set of matching DSs will not contain duplicate copies of events, or that the experiment software will have to have some sort of guard against inclusion of duplicate events in a single pass. In some of the scenarios below, this second case is still not sufficient.

4.3   SUPPORT FOR ANALYSIS JOB EXECUTION BY A COMMON LAYER

In HEPCAL, the middleware does not provide any special support for this type of analysis job. At first glance this seems undesirable, since such a job has the potential to access many datasets and consume a considerable amount of computing resources. If there is no special middleware support, the job may not benefit from being run in the grid environment, and analysis may even take a step backward from pre-grid days. Therefore assistance from the middleware seems desirable and we describe here several scenarios for how the Workload Management System (WMS) - experiment-independent middleware - might support execution of analysis jobs as described above.

There are several options for running analysis jobs as have been specified in the previous sections:

  a) **"User program does it all – no WMS support"**: the user submits the job that is sent all the way to the WN where the execution starts. The query is executed from the worker node. Input data is accessed via standard mechanisms as described in HEPCAL – relying on local access is not possible since the WMS does not know the list of DSs to be accessed and hence cannot submit the job to a computing resource from which local copies are accessible. The resulting data access would probably be so inefficient that experiments would use a two-step submission: in a first step the list of DS's is extracted by a metadata query and in a second step the jobs are submitted with full specification of the input DS's.

  b) **"WMS queries DMC and then submits job"**: the user submits the job, the WMS performs the query to the DMC to optimise the CE selection, the job is sent all the way to WN with the experiment dependent query and the list of LDNs, job execution starts (see previous case)

The following items (c) and d)) are implemented making use of a workflow representation mechanism. Experiment dependent tools that can be invoked by the WMS (*plugins* that we describe later on) are required.

  c) **"WMS queries DMC, submits multi-jobs and merges output"**: the user submits the job, and the WMS performs the DMC query as in b). The WMS generates several sub-jobs, one for each CE *close* to at least one of the input DSs. Each of the sub-jobs (experiment-level query plus algorithm) runs the algorithm on its local data. Some experiment-dependent merging of the sub-job outputs must take place at the end.

  d) **"WMS queries DMC, performs multi-queries and merges input"**: the user submits the job, and the WMS performs the DMC query as in b). The WMS generates several sub-jobs as in c). Each of these sub-jobs selects the events matching the experiment-dependent part of the query and places these on its output. This output is merged at a final job in the pipeline, where it forms the input of the user-specified algorithm.

Below we will describe these cases in greater detail, and also the possibility of some mixed scenarios.

4.3.1   No special support by WMS

The job description is prepared, which includes a specification of the program to run, any required environment specification, and as input the job receives the query parameters in some format that the specified program can understand. The Workload Management System executes the job on a site based on the job requirements (e.g. software environment or available resources). Data access is not considered by the WMS since it has no idea which data will be accessed, as they will be determined by the query when the job is already in execution.

1. The job starts, and if there is a "dataset level" piece of the query, the job can execute the "DS Metadata Access" use case of HEPCAL to get a list of LDNs; alternatively it can access an experiment specific metadata catalogue.

2. The job accesses each of these DS's in turn. For each DS it applies the rest of the query as a filter to pick out only the selected event components. How the job does this (as a pipeline, or as an "if" statement in an event loop) is not considered – we simply say, "the job does it". The job will make use of the middleware layer data access tools as described in HEPCAL (see sec. 4.3.4 for more discussion).

3. The job returns the data by uploading it to the grid, or arranging for transfer back to the user.

4.3.2 Support at the Dataset Level

In this scenario, we suppose that the WMS has the capability to execute queries to the Dataset Metadata Catalog (as defined in HEPCAL) on behalf of the user. As an example, in the EDG middleware at the moment we can tell the WMS

```
InputData = "LF:stanlib.3"
```

This tells the WMS that the job will require access to the DS with the logical name "stanlib.3". We suppose in the current scenario that we can replace this input data specification with a query. In the example above, the query would be:

```
Return all (or perhaps "at least N") datasets from first quarter
2007, taken in trigger configuration EFG, reconstructed with reco
version 7.4 and calibration set 21.d3.
```

Of course we expect this query would have to be presented in some format understood by the DMC instead of in plain English. The WMS would execute this query and formally replace the query by the list of LDNs that match it. At this point the job would execute as in HEPCAL:

1. The WMS will optimize the selection of the computing resource according to the program selection, environment requirements, computing power availability, *as well as* optimization with respect to the list of LDNs generated by the query (as described in HEPCAL).

2. The job is executed at the remote site, and receives as input the list of LDNs and the experiment-dependent part of the query.

3. The job itself is responsible (as in the previous section) to select the objects from the supplied datasets, according to the experiment-dependent part of the query, and to apply the algorithm to these objects. Furthermore the job will have optimal access to the specified set of input data, but in most cases at least some of the DSs will need to be access remotely as in 4.3.1

4. The job returns the data by uploading it to the grid, or arranging for transfer back to the user.

4.3.3 Support via job pipelines

A higher level of support can be provided if the WMS implements job pipelines or workflow representations as described in HEPCAL. Given this mechanism, a job can supply its output as input to one or more subsequent jobs. We suppose the existence of plug-in tools provided by the experiments (or possibly end users) that would be used by the grid in these pipelines. In the following, we call these tools: select, dump, input merge and output merge. It needs to be clear that these are tools provided by the experiment, but they have to be package in a way that makes it possible for he WMS to invoke them.
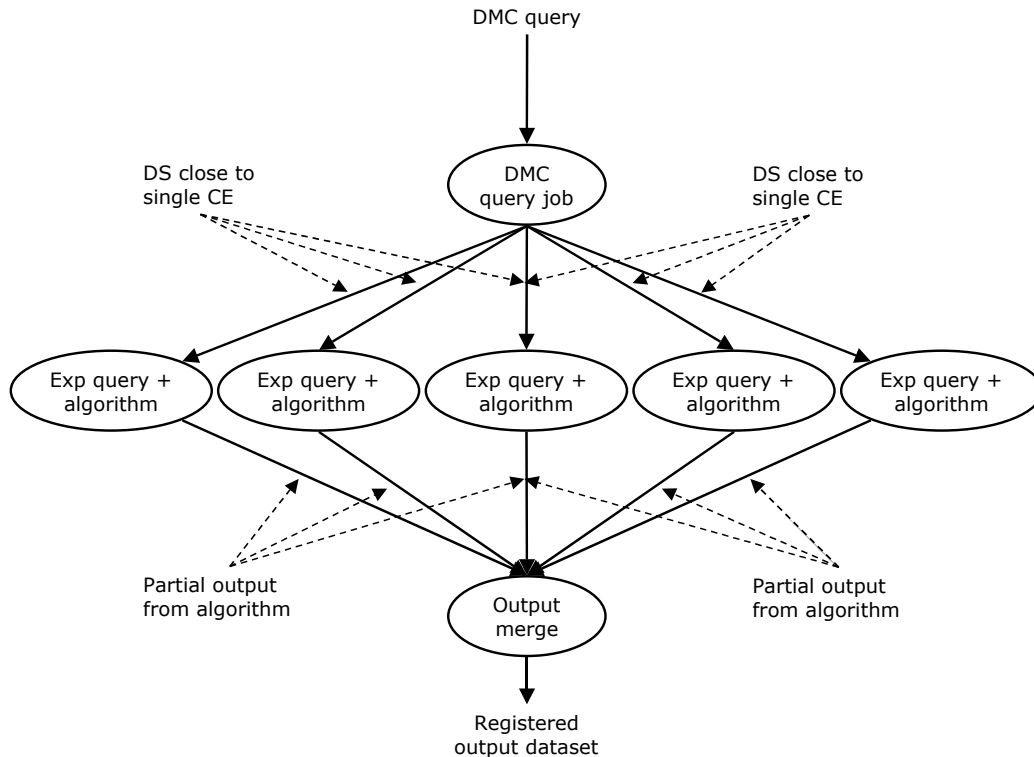
select   takes as input an experiment-dependent query and returns list of object references.

dump   takes as input a list of object references, and returns as an output stream the bytes of those objects, it is a "deep copy" tool.

input merge   query and dump event components into an single input file.

output merge merge the output of several copies of the same job run on different input DSs.

There are several ways in which such a construct could be used to effectively implement an analysis job. Here we will list some.

4.3.3.1 Distributed Execution with no special analysis facility support

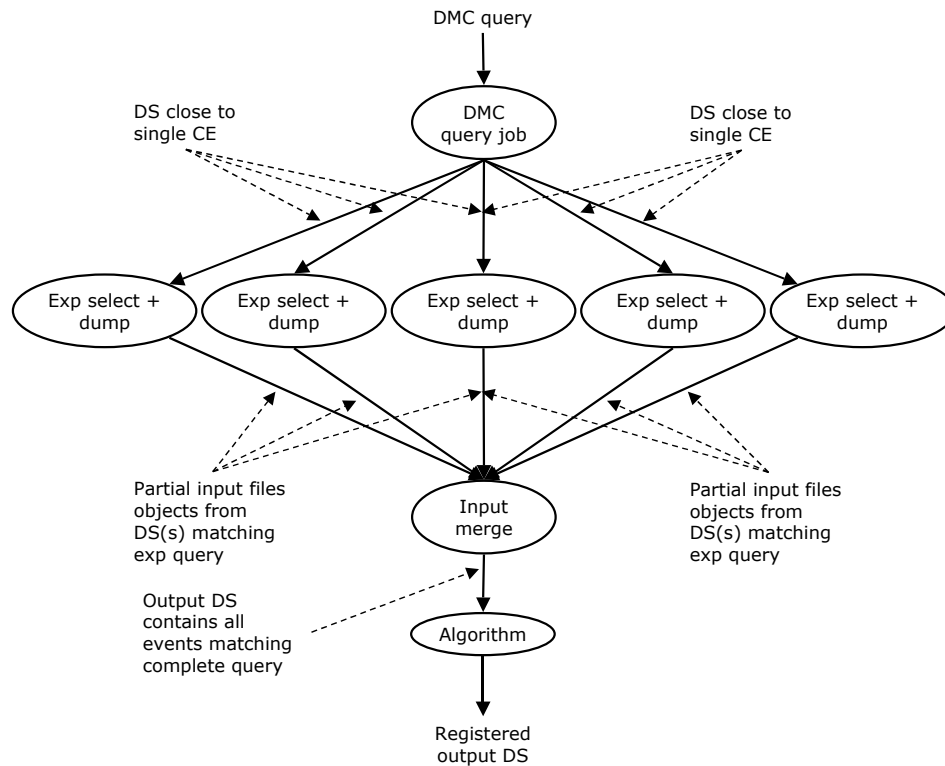This mechanism is illustrated in the following figure.

The flow for this is as follows:

1. The first node in the pipeline is a "dataset metadata select" job. The input is the dataset part of the query. The job to be run is either a "DS Metadata Access" use case of HEPCAL, or it could be an experiment-dependent tool that knows how to access an experiment's private DS Metadata Catalogue. In either case, the output of this node is a list of matching LDNs.

2. The next stage in the pipeline is a set of jobs, each one will have as input a subset of the LDNs generated by the first node. The input to each job is

   a. The LDNs for the job. The WMS may optimize by executing a single job at a computing site with good access to physical copies of several matching datasets.

   b. The experiment-dependent (event component level) part of the query.

   c. The specification of the algorithm to run (executable, environment, input parameters and so on). This will be "inherited" from the original job submitted by the user.

3. Each of these jobs can proceed as in the previous case, except that now the DS access has been optimized by the WMS.

4. The output of each of these jobs is collected together at the last job of the pipeline, which runs an experiment-dependent or even user-specific tool that knows how to merge the results together into a single meaningful dataset.

5. The job returns the data by uploading it to the grid, or arranging for direct transfer back to the user (e.g. sandbox).

4.3.3.2 Distributed Data mining with experiment plugin support

The original HEPCAL team identified many problems with the concept of merging partial output files. Another workflow representation construct can avoid this problem if it turns out that merging of partial input files is simpler. The disadvantage of this solution is that it can be very inefficient (with regard to data movement) if the query selects a significant fraction of all the events, or in other words if the "merged input file" is nearly as large as the sum of the sizes of all the DSs matching the DMC part of the query.

The mechanism is illustrated in the following figure.

The flow of work in this case is as follows.

1. The first node in the pipeline is a "dataset metadata select" job. The input is the DS part of the query. The job to be run is either a "DS Metadata Access" use case of HEPCAL, or it could be a tool that knows how to access an experiment's private DS Metadata Catalogue. In either case, the output of this node is a list of matching LDNs.

2. The next stage in the pipeline is a set of jobs, one for each group of DS matched by the first node. The input to each job is

   a. The LDNs for this job. It might be several LDNs since the WMS can optimize by executing a single job at a computing element with good access to physical copies of several matching datasets.

   b. The experiment-dependent part of the query.

   c. A combination of the experiment-dependent tools "select" and "dump" loops over all events in a DS (or list of DSs). Select applies the query and generates a list of object id's, and dump in turn receives the ids and generates from them an output stream. The input of these jobs has been optimized by the WMS. The output of each of these jobs

is a stream of bytes corresponding to the objects, contained in the input DS(s) for the job, which match the experiment-dependent part of the query[2].

3. The output of each of these jobs is collected at the following (single) node of the pipeline, which is an "input merge" job. It takes all the byte streams as input, and generates an output DS that has the same format as a normal DS for that experiment. "input merge" also must be provided by the experiment.

4. This DS is passed to the last node of the pipeline, which runs the algorithm. It doesn't need the query as input, since the dataset it receives contains all the events matching the query, and none other.

This output is either registered via DS Upload or else returned to the user. Note that we could stop at step 3 and register the output – this is essentially a "re-cluster" pipeline scenario.

4.3.3.3 Middleware Support for Dataset Queries

The previous two pipeline scenarios could be implemented as described, but this seems to imply that the query job must be able to interact with the WMS to submit the distributed second stage of jobs. It is necessary for the query job to have this capability, as the set of matching CEs is only known after the query job runs. We are aware of serious concerns with such a scheme coming from middleware security groups. An alternate scenario that appears to be equally effective is to replace the query job with the dataset query mechanism described in section 4.3.2. In this case the WMS directly receives the matching set of LDNs and job submission proceeds normally (i.e. the job-submits-a-job scenario does not occur).,

4.3.3.4 Mixed scenarios

The previous two pipeline scenarios are two "extreme" examples, as we can imagine an almost continuous variation between them. Suppose that the experiment-dependent part of the query implies the calculation of some complicated property of the event, not included directly in the data. In this case the distinction between query and algorithm becomes rather blurred, and we can imagine that one part of the algorithm run "close" to the DS, and only those event components satisfying certain properties, determined after applying one part of the algorithm, are sent to a common node for further processing.

Also the way in which the data is sent from one node to the next of the graph can vary substantially. At one extreme we have each job in the "central" layer of the graph registering one or more DS and passing to the next layer the LDNs. At the other extreme we could imagine jobs communicating via some sort of socket or pipe opened for the whole duration of the processing, a sort of WAN parallel processing similar to what can be achieved on a local cluster with MPI. We tend to consider those as implementation issues and we will not elaborate further on them.

4.3.4 Support for Data Access by a Common Layer (or mechanism)

The issues regarding DS access in support of analysis jobs are largely addressed in HEPCAL, which assumed that the Data Management System would transparently optimize data access on the user's behalf. HEPCAL anticipated that at least the following options would be considered by the DMS:

1. Access (possibly via remote protocol) to an existing physical copy of the DS;

2. Making a new replica to an SE – because this SE has file-systems mountable from the chosen worker node, or perhaps it supports the protocol requested by the application – and arranging for the user program to access this new one;

3. Making a local copy to temporary storage at the worker node where the job is running;

---

[2] Depending on how the experiment structures its metadata, it might be better to implement this in a single tool, "qdump". The scheme mentioned in the main text is more suited to the case in which the experiment keeps a metadata table separate from the dataset itself.

4. If a virtual definition of the dataset exists, materializing the DS to either a suitable SE or local temporary storage at the node where the job will run.

The user will in general not be aware of this; her program will just "open the DS". Subsequent reads on the returned handle will "get the bytes".

For analysis jobs, this may not be enough. In many cases the dataset access may be very sparse, meaning for example that a thousand-event analysis may attempt to access one thousand separate datasets. The previous sections discuss how the Workload Management System might support such access.

The application itself can support such access via intelligent use of the DMS. An example mechanism is an experiment-specific program (or servlet) that dispenses events. This servlet would receive the results of the query, e.g. as a list of (LDN, objref)-tuples as input, and as output would dispense event data objects in a manner compatible with the experiment software. In the background, we expect the event dispenser to contact the Data Management System to discover access costs for the various DSs, and then start accessing them in an advantageous order, queuing the events for consumption as the bytes arrive. From the application's viewpoint, the servlet object provides data on demand, and the application need not be concerned with access optimization.

5   USERS, GROUPS, QUOTAS, AND PERMISSIONS

The "ownership" context in which analysis activity proceeds spans a broad spectrum. At one end is the lone graduate student working in his office, perhaps collaborating with one other student with who he shares a perhaps brilliant idea on how to reanalyze a particular channel. At the other end are tasks like (re-)re-construction of data with improved parameters and algorithms, bordering on an experiment-wide production activity. A particular user may be involved with both these activities, for which different data access permissions and resource quotas may apply. The situation will require a flexible yet effective system for organizing quotas, permissions, and users.

5.1   USER AND GROUP SCENARIOS

We present here three scenarios, or rather categories of analysis scenarios, for categorising analysis scope within the collaboration, from the one of the single user to the "experiment-organised" analysis. The major impact of these different scenarios on grid Middleware developers and resource managers comes mainly from the scale of the resources required by each instance of the scenarios, and from the granularity of roles and access rights implied. This requires that the authorization of resource access be done at a different scale for each of the scenarios. We begin with the scenario "End-User Analysis", since this scenario will be the one most frequently executed.

End-User Analysis ("EUA")

EUA is conducted at the level of an individual researcher (user) within the collaboration. The user executes the analysis steps defined above in 3. She may apply a private algorithm to the input data. The user may be using a private dataset as input, and this dataset might not even be in the grid Data Management System (DMS). The interval between consecutive iterations of this activity probably ranges from a few hours to a few days (or as frequently as resources permit!). The analysed information may then be given back to the experiment in the form of new selections/cuts/algorithms to be used in a subsequent Group Level Analysis (GLA) or Production Analysis (PA, see below). Alternatively the output might be a new (grid or non-grid) DS. Users wish to be able to create segments of the grid file system (analogous to a unix subdirectory) to hold these results, with access rights tied to the user's Distinguished Name (DN, see below).

The resources used by EUA instances are tied to the individual user. This is typically a relatively small slice of resources per user, but can be quite significant in aggregate as there are many concurrent users executing EUA. In the VO management system the authorization of resources is tied to the users' Distinguished Name (DN) which today maps onto an anonymous user account at each Regional Centre. We expect that users' desktop machines will provide a relatively large proportion of the compute resource for this type of analysis.

Group-Level Analysis ("GLA")

A PWG gives to (one of) the group production manager(s) a set of input DSs (based on a selection on the Event Metadata, EMD, Analysis Object Data, AOD, or TAG from the reconstruction step) and an algorithm to be run on it. The input DSs are selected by queries against EMD or TAGs from the experiment, including the information from the (previous) Production Analyses. Depending on the policies of the experiment, the group might initiate replication of the output data to other sites, or this might happen automatically for "official" data products. We expect this scenario to be executed about every month (per group). Usually all DS are on the grid and the results are also made available to the other group members on the grid. This analysis is often a refinement of the preceding PA in preparation for the following PA. The corresponding data products will, in many instances, need to have access controls based on group identification, and the groups will likely wish to have their own "group areas" in the grid file system in which they have full control.

GLA uses a share of resources allocated to the PWG by policies within the collaboration. It is expected that the relative proportion of resources allocated to PWG will be quite dynamic. This requires that the authorization mechanism for resource allocation recognize a granularity of rights between that of the

VO and the DN. I.E. Something akin to the Unix group concept. In the past, this type of analysis was typically done on institutional resources local to the researchers within the PWG such as smaller university and laboratory clusters or farms. The "Production manager" needs to be a grid *role* (that can be granted to individual users by the VO) rather than a person or persons. These roles are typically assigned to one or a few individuals for a limited period of time, and then reassigned to a successor. This implies that the VO management system allows the definition of such roles. Resource providers need only be concerned with allocating resources to the role, and the VO is responsible for the policy of mapping that role onto individuals and/or small groups. The use of individual production manager DNs will generate many bookkeeping and administration problems in the distributed environment.

Production Analysis ("PA")

The various analysis groups of an experiment each suggest to the experiment production manager a set of input DSs (based on a selection on the EMD or AOD/TAGs from the reconstruction step) and the algorithms of a given version of software and configuration information. After the experiment management endorses these algorithms and configuration data, the experiment production manager starts the "Production Analysis" running the software on the selected (from the Reconstruction TAGs) events from the input data sets. This program will produce a set of output DSs and EMD, potentially creating multiple output streams that will be published on the grid (with "deep" or "shallow" copies of the relevant event components). Running the whole collection of "Physics groups algorithms" is an experiment-wide centrally managed "production". All DS are on the grid and the results are also made available on the grid.

The resources allocated to PA will be global in nature, and of a scale that merits the VO's attention. Typically PA is performed by individuals charged with special roles within the VO (e.g who have done this kind of work: production managers) and allotted major resources as the VO's proxy at each Regional Centre.

The three examples given before try to capture different aspects of what is really a continuum. We are aware that the difference between the "RECO" and "PA" as described may be mainly sociological and concerning quality assurance and resource allocation. However in the context of this document, going from "ESD" to "ESD-prime" for a subset of "RAW" is defined *not* to be in the scope of a "PA".

Section 10 describes the 3 generic use cases PA, GLA and EUA using the HEPCAL notation.

5.2 QUANTITATIVE REQUIREMENTS

Almost all physicists working in an experiment will (hopefully) be involved in the analysis activity at some point. They will probably do so in a combination of the three scenarios described above, and this makes it difficult to express quantitative requirements. The figures that follow have to be taken only as a very preliminary guess that will need to be refined by discussions within each collaboration. The starting parameters (number of files, total size of the data and so on) will be expressed in the HEPCAL prime document due to appear soon.

It is expected to have about 10-15 physics analysis groups in each experiment with probably 10-20 active people in each extracting the data from the earlier scenarios above (PA or GLA). For the later stages (scenario GLA or EUA) the produced data may not necessarily be registered on the grid. In addition, it is expected to have about 30(?) people per subdetector in each experiment (total of 300-500? per experiment) accessing the data for detector studies and/or calibration purposes. So a total of 400-600 people in each experiment are expected to do the extraction of (possibly private) results. This number is representative; depending on the stage of the experiment the profile might be quite different.

6 INTERACTIVE VS BATCH GRID ACTIVITY

Analysis activity will truly be integrated with the grid only when users are able to "interact" with their jobs in meaningful ways. Part of this has to do with persistent environment support – a logbook facility, properly integrated with the environment seen by the user, could provide the grid equivalent of the "up-arrow" key we use at the Unix command prompt. Another example is the equivalent of leaving your PAW analysis session open and running, coming in and inspecting the results the next morning; the analysis session is *persistent*.

The above examples say something about the interactivity of the analysis environment. It is much more difficult to explore what one might reasonably expect from interaction with running analysis tasks or "jobs", partially because it is very difficult to define interactivity in this area. We judged it essential to attempt to develop taxonomy of different classes of "batch" and "interactive" jobs. This helps us define a common vocabulary to be used by physicists and middleware developers, and gives us a well-defined structure for defining user requirements and for associating them the different categories. Following this classification we present some preliminary comments on the requirements for interactive jobs.

We now propose a distinction between those tasks that are typically viewed by end-users (physicists) as interactive versus those that are considered batch. The single, most obvious characteristic of an interactive job (relative to a batch job) is the presence of a human interacting with the job **during** its execution. A batch job involves no significant human interaction between job submission and completion verification. However, the **amount** of human interaction with the job and feedback to the human can vary considerably both quantitatively and qualitatively.

In Figure 1 we try to capture the gradual transition between batch an interactive work.
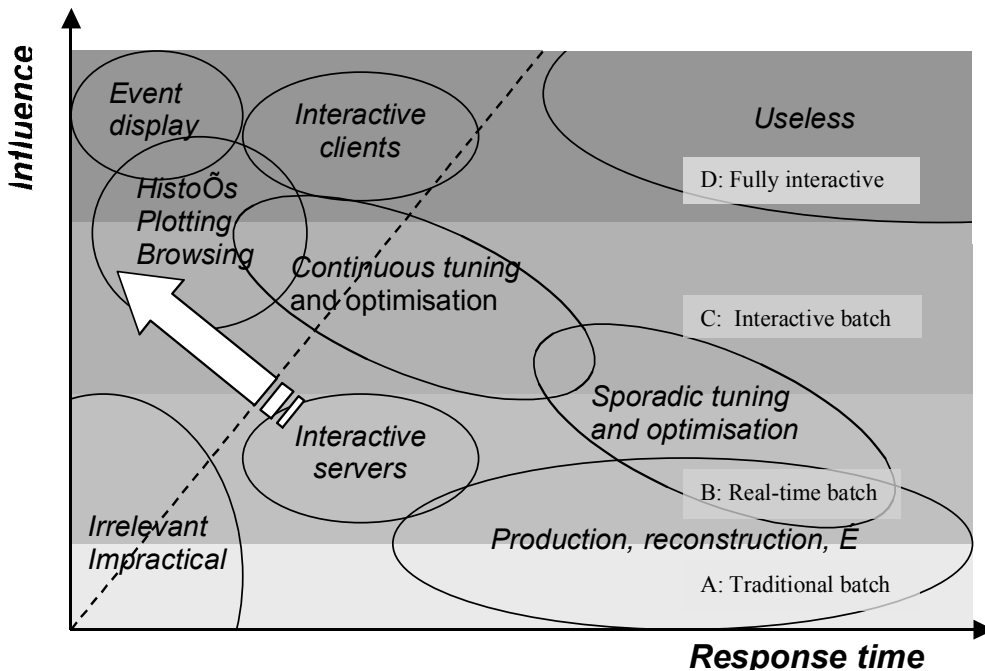

Figure 1: Characterisation of batch and interactive jobs

The vertical axis we label as "Influence" and it denotes the amount of control over job execution/job behaviour that the user exerts (N.B. The user will typically <u>have</u> more influence than she exerts, but it

is the upper-limit of influence which she may wish to exert which defines the requirements on the system.).

The horizontal axis of Figure 1 we label as "Response Time". By response time we mean the wall clock time elapsed from the issuing of a command to the completion of the action that this command entails. This applies to different situations such as job start-up time, request execution latency, user feedback update period, and job completion time.

We define 4 general categories of Influence (Regions A, B, C, D). They are described from more batch-like to more interactive:

Region A. This region contains traditional Batch and Standard Scheduled Jobs. These jobs are usually considered indisputably batch. They are typically submitted to a batch queue or grid job scheduler for execution. The user can effectively ignore the job until it finishes. These are also called "black box" according to the definition of PPDG-CS11.

- User Input:

    o Only lifecycle commands (e.g. start, stop, suspend, resume)
    o Communicated to the job scheduler, not to the running application.

- User Feedback:

    o Only high-level job status and job health monitoring, once again via the job scheduler, not directly from the running application.

- Examples:

    o All modern batch systems.

Region B. Jobs that are submitted to a batch queue or grid job scheduler, but with a communication channel available for minor influence on job behaviour and some ability to examine intermediate results before job completion. Fundamental flow of job (i.e. algorithmic code run, input parameters used) is fixed. These are also called Real-Time Batch in the terminology of PPDG-CS11.

- User Input:

    o Uni-directional/asynchronous, pre-defined commands to the application that only change the behaviour of the job in predictable, and relatively minor ways (e.g. turn on/off monitoring, end execution gracefully, etc.).

- User Feedback:

    o Partial results (e.g. streamed, or incremental) available to user during job execution. (Can for instance be used as input to decisions for job termination.)

- Examples:

    o Many modern batch systems (allow examination of stdout/stderr).
    o LSF/BOSS
    o GMA-Instrumented Athena
    o Job control via semaphore termination

Region C. Jobs that use a batch queue or grid job scheduler to schedule, place, and execute part of the job, but which have many aspects and capabilities of a traditional interactive job, including direct and strong control of program execution. These are also called "Interactive Batch" (PPDG-CS11)

- User Input:

    o Bi-directional/synchronous control of the application at a level similar (identical?) to that of a traditional interactive job.

- o Usage of the communication channel is optional and it can be opened and closed as desired.

- o Default behaviour of job (e.g. when no new user input is being received) is defined. Similar to Region B/A if little or no influence is exerted.

- o Ability to change control flow of job and algorithm, and input/control parameters.

- o Restart of a job step does not necessarily require re-init of environment/application (e.g. re-open the DSs). This requirement is sometimes expressed as support for same level of <Ctrl-C>-like handling.

- User Feedback:

  - o Can be through dedicated, distributed service (eg. distributed DB).

  - o If done manually, is typically chunked for efficiency.

- Examples:

  - o DB I/O

  - o AliEn + ROOT session.

Region D. This corresponds to the common definition of fully interactive jobs. They are also called Dedicated Interactive by PPDG-CS11.

- Jobs that meet most the layman idea of interactivity.

- User Input:

  - o User has full control (at level defined by application) of application during execution.

  - o Provides a <Ctrl-C>-like interrupt facility.

  - o Use of the control channel is mandatory. Closing the control channel is tantamount to ending the application.

  - o Default behaviour of job is to wait for user input and display a prompt.

- User Feedback:

  - o Immediate and finely grained.

  - o May be chunked for efficiency, but at a much smaller granularity than Region C.

- Examples:

  - o LSRUN

  - o PROOF

  - o Distributed JAS

  - o Interactive applications such as PAW, ROOT, JAS, interactive event displays, etc.

  - o Most commercial desktop applications.

The horizontal axis can be divided into general regions (though we do not do so on the figure), based largely on human time-scales:

- < 1 sec: Instantaneous. User's attention is continually focused upon the job.

- < 1 min: Fast. Time periods spent waiting for response or results is short enough that user will not start another task in the interim.

- < 1 hour: Slow. User will likely devote attention to another task while waiting for response/results, but will return to task in same working day.

- > 1 day: Glacial. User will likely release and forget. Will return to task after an extended period or only upon notification that task has completed.

To clarify different regions of the interactivity phase space, we draw contours associated with different analysis scenarios. These scenarios are intentionally vague and the contours are completely qualitative. We have had extensive discussions on where various contours belong and stress that the answer depends upon the perspective one is taking and/or the point one is trying to illustrate.

The scenarios identified in FIGURE 3.1 are:

- Interactive Event Display
    - Demands instantaneous response to all commands/actions

- Histogramming/Plotting/Browsing
    - Human-time scale response to all commands/actions
    - Can be orders of magnitude different than instantaneous
    - Users are used to waiting a reasonable time period for feedback

- Continuous Tuning/Optimisation
    - Redoing calculation with different properties (e.g. control parameters, algorithm code, etc)
    - Human-time scale processing time

- Sporadic Tuning/Optimisation
    - Redoing calculation w/ different properties (e.g. control parameters, algorithm, etc)
    - Human needs to be prompted when input required or check-in occasionally

- Reconstruction, other Production Jobs
    - long (> 24 hours), large scale jobs with no human interaction except monitoring

- Useless
    - This region is useless in any practical sense as the response latency of system incapacitates any human-time-scale interactivity

- Irrelevant/Impractical
    - This region is irrelevant to any grid discussion, but is identified for completeness.
    - For instance if completion time is much faster than time for interacting or if the sum of initialisation, startup, scheduling and so on is much larger than the execution time.

- Client/Server or bi-modal
    - Some tasks in which a user engages may span multiple domains.
    - For instance an interactive client being fed by autonomous, batch-like servers.

We have focused primarily on the distinction from the users' perspective at this point. There are some aspects of the problem that we have not addressed in detail but are corollaries of interactivity.

- Is predictability inversely proportional to influence?
    - Not in principle, but probably in practice. Batch jobs can specify sparse data sets as well as dense data sets, or can use navigational aspects of the event data model, or can use complex criteria for determining resources used. However, in practice most batch-like jobs will have a higher level of predictability than more interactive jobs.
    - How interactive-job "influence" will work with typical middleware job optimisation and allocation strategies? Influence may change unpredictably the algorithm flow and the DSs accessed.

- Does interactivity impose special requirements on software

- o Portability/Distributability? No more than for any other kind of grid activity. Portable, distributable control technologies like Java byte-code, Python scripts, CINT macros, etc. are already used by many HENP systems

- Does interactivity impose special requirements on communication and security protocols?
    - o Probably yes. Both control and feedback imply communication one way or the other between a CE node and a user's desktop or laptop.

We do not here try to address the technical issues, but are aware that they exist and are significant.

7 SYSTEM REQUIREMENTS

While discussing the analysis activity, we realised that independently from the way in which this activity is performed, the grid middleware should provide advanced services not contemplated in HEPCAL. During the HEPCAL-II preparation, we have discussed several of these services, but we had time only to describe few of them in some detail, those that we felt to be the most important or at least the most immediately useful:

- Provenance and job traceability.

- Log books and reports.

- Persistent interactive environment.

- Analysis software deployment.

7.1 PROVENANCE AND JOB TRACEABILITY

Knowing the "history" of a set of data is essential in analysis given the "needle in the haystack" syndrome, where we are looking for very small signals against very large backgrounds. Hence the validity of data is a crucial issue, and being able to trace the history of transformations that lead to a given dataset is an important requirement that is discussed in this section.

Consider the following scenario of two geographically distinct collaborators who wish to work together on a B-physics analysis: one person works to improve the B-tagging of an analysis, while another person works to optimize statistical fits of the data used to interpret the final result according to a theoretical model. Neither collaborator is an expert in the activities of the other. Hence, the latter collaborator uses datasets, which are based upon default reconstructed B-tag information to develop the necessary complex statistical fits, while the former collaborator continues to work independently on vertex fitting. As the B-tag is optimised and new, derived datasets are recorded, the statistician must merge his/her own statistical fit work with the B-tag collaborator's latest datasets. The collaborators become excited upon discovering a new peak in a mass plot. After reviewing the strange mass plot for possible collaboration approval for a conference contribution, the PWG leader is naturally suspicious and requests the provenance of all events contributing to the peak, beginning from the analysis datasets composing the plot and going all the way back to the Raw physics data. Using this information, the PWG leader discovers that, at a critical juncture in the analysis chain, most of the peak events were processed on the same Computing Element (CE). A check of the execution environment on that CE, during the time window when those events were processed, reveals an outdated version of an object library, known to induce systematic shifts in certain mass distributions. Upon re-performing the analysis, ensuring that jobs are only sent to CE sites with the proper execution environment installed, the B-physics team is disappointed to observe that the mystery peak has disappeared.

Every good physicist is acutely aware that one's results must be both verifiable and reproducible from one's peers. Therefore some system of recording, publishing, and discovering the provenance of one's analysis results to the scientific collaboration must be present. Validation of scientific results is often based on user defined tests as well as understanding the complete derivation history of the result. Recording the derivation history of a result is important in a well controlled, local environment but, becomes critically important in a less controlled, highly distributed, grid computing environment. When a result fails a validation test, the result must be debugged. Given the heterogeneous, distributed environment, debugging a physics analysis will be challenging and hence, the ability to propagate error messages and access provenance information about each dataset will be vital.

Complicating the provenance of datasets and scientific results, physicists in HEP tend to work together in teams which are tightly tied to the data-flow of the experiment. Not only do individual physicists within a team need to track the work (i.e. datasets) of other scientists within the same team, they need to track the data deliverables from teams outside their own. This is evident from the fact that each

successive stage in a HEP data-flow builds upon the work of a (typically) separate expert team in the collaboration. A typical collaborative workflow in HEP might be loosely characterised in the following manner:

On-line Detector Teams are responsible for recording quality data, and ensuring the health of the detector. Each sub-detector has a team of experts (often functioning in shifts) that initially calibrate, monitor and log (annotate) sub-detector performance.

Calibration Teams are responsible for fully understanding the response of each sub-detector as well as approving and applying later calibration corrections to the on-line data. Such teams are typically the same as, or closely related to, the on-line detector teams.

Monte Carlo Production Teams are responsible for the coordinated large-scale simulation of data according to different physical models and different detector models.

Data Processing Teams are responsible for large-scale coordinated processing (reconstructing) and reduction of data (real and simulated) into formats, which are more readily usable for data analysis. Such processing would, for example, produce Event Summary Data (ESD) from the Raw data, Analysis Object Data (AOD) from the ESD, and possibly TAG data from the AOD. This is often performed automatically (as in Production). However the processing teams are responsible for inserting or modifying new reconstruction and processing algorithms into the data-flow. There can be a large overlap between the Data Processing Teams and the Monte Carlo Production Teams.

Physics Data Analysis Teams are responsible for developing data analyses projects corresponding to different scientific interests of the Collaboration. The analyses are performed on processed real and simulated data. Occasionally, an analysis will require a change in the way the data is processed and/or simulated; this is usually communicated to the Data Processing and/or Monte Carlo Production Teams. Typically each analysis project consists of several individuals, contributing a particular piece of the work. An analysis project always depends upon derived data (e.g. the Processed Data) and may extend (or, possibly specialise) a separate, pre-existing analysis from a different (or, possibly the same) research team.

Publication Review Board Teams are responsible for auditing and approving the results from a physics analysis project for publication in the name of the collaboration.

In such a collaborative environment, the cross-knowledge between different teams is often necessarily on a "need to know basis" (because of the expertise level required) and asynchronous progress of the various data-flow teams is common. Nevertheless, the interaction of various teams with the data-flow must be performed in a coordinated, synchronous manner. To date, in existing non-grid environments, this coordination often takes place within the local environment of the host laboratory and is typically accomplished by human initiated communication (email, collaboration meetings, chats over coffee, etc).

In a globally distributed environment (such as a grid), a system which records the provenance of datasets and controls the versioning of the data-flows will be required to validate scientific results as well as to allow physics analysis teams to function semi-autonomously in much the same way that software developers use code versioning repositories (e.g. somewhat analogous to CVS). No particular implementation is implied, however, CVS-like terminology is used to help convey ideas related to how a provenance system might be used.

A provenance system should:

- Maintain references to the entries in the Job Metadata Catalogue of jobs used to produce the dataset. This means incrementally record the complete history of how each dataset was produced, including the executable name and version, local execution environment, the input dataset, and the output dataset. The provenance information recorded for each dataset should be complete enough to allow the dataset to be reproduced (at least in principle). Connecting provenance information for related datasets ultimately defines a provenance graph, or "data-flow," from the final output dataset, through all intermediate derived datasets, to the original dataset.

- Enable the Collaboration to sanction (or ban) various data-flows, defining "data-flow trunks" for use in physics analyses for publicly released scientific results. This would be important for defining which datasets are allowed for use in upcoming conferences.
- Enable sub-detector teams and data processing teams to precisely control the way the main data-flow trunk is versioned (new calibration corrections, new reconstruction algorithms, etc) and released to the Collaboration.
- Enable a user to invoke a transformation on any dataset contained in a Collaboration sanctioned data-flow trunk, producing a derived dataset and defining a new "data-flow branch." This process is in general recursive: users would typically define new data-flow branches from existing data-flow branches.
- Enable analysis developers to contribute generic, lower-level scientific results (datasets), which other, higher-level, analysis projects could build upon.
- Provide information to an interactive logbook (see next section) enabling
  - Scientists in a geographically distributed environment to collaborate on a data analysis project in a coordinated way and allowing individual scientists to focus on and contribute particular parts of a data analysis project. The scientists could merge their individual contributions into a complete analysis (defining a new branch of the data-flow) at their convenience.

  - Leaders of PWGs to keep track of the progress various analysis teams in preparation for upcoming conferences.

  - Independent review committees, which often have no intimate knowledge of a particular data analysis, to audit the history of plots, tables, and statistical fits, thereby giving more (or possibly less) confidence in the final scientific result.

  - The collaboration to re-visit any earlier versions of an analysis to better understand why an early candidate "signal" (possibly even presented at a conference) is no longer seen in the current data (or vice versa). Provenance information for a dataset should be available for the lifetime of the experiment.

Several challenging issues related to recording provenance information are recognised. One issue relates to identifying the minimal set of information required to fully describe the local execution environment for a job. Relevant run-time error propagation, related to the local execution environment as well as the application itself, will be an important tool for debugging in a distributed, heterogeneous environment. A possible solution is to keep at each site the software configuration as function of the node and of the date. This information combined with what recorded in the job metadata catalogue should allow obtaining the exact run-time environment used to produce a given set of data. Another issue includes how to handle the provenance of partial results and incremental updates for new, real (or simulated) datasets. The intersection between two successive, incrementally updated, datasets can be large. For example, as new calibration constants are applied to different individual data products over different periods of time, a dataset becomes only partially outdated. In such a case, how can the analysis developer know how to re-compose a similar dataset with all the latest updates? On the other hand, physicists may want to "commit" datasets to a provenance system at regular, but interesting checkpoint intervals, and not for every incremental update to data products comprising a dataset.

While we are hesitant to address issues of implementation, we are aware that the functional description of the provenance service appears rather heavy. It is possible that the same functionality could be achieved by a careful design of metadata recorded in various places (the CE software configuration log, the JMC, the DMC, etc.) coupled with a well-designed tool to mine and combine these data. The important issue is that the information must be reconstructable without the intervention of system administrators.

There is a close relationship between provenance and materialisation information of virtual Datasets. A provenance record is the realisation of the materialisation information contained in a virtual Dataset.

7.2 LOG BOOKS AND REPORTS

For the following discussion, we define a task as an activity that produces a set of output data, based on an algorithm and a (possibly empty) set of input data. A typical example is a single (batch) job, but a task may consist of several such jobs, may involve interactive analysis sessions, and may include sub-tasks. On the other hand several tasks can be included in the same job.

In the course of a given physics analysis, many such tasks will be executed. Some of them will depend on the output of previously executed tasks. The output of some tasks may partially or completely supersede (logically if not physically) the output of previously executed tasks, e.g. when the algorithm used by the task has been improved or the input data used by a previous task has been revised. As physics analyses are often performed by groups of people, tasks may depend on other tasks that have been executed by different people. A typical physics analysis will involve a large number of such tasks that one will have to keep track of.

We would therefore like to have an electronic bookkeeping tool (called "logbook" in the following) with the following features:

a) A record should be kept of every task whose output is not immediately discarded as useless. This record should reference the input data, algorithms, scripts (or log files from interactive sessions), and all output data. The information stored should be sufficient to understand what happened during the task and to possibly repeat it. It should be possible to attach user comments and explanations to this record, including particular plots and intermediate results of interest. If the provenance information introduced in the previous section is available, one only need to present and annotate it in a convenient fashion to implement this request.

b) The logbook should be tightly coupled with the WMS, so that the current status of a task can be queried, or a list of all pending tasks can be obtained. The recording of jobs submitted should be automatic and should not require the user to enter the job information manually.

c) The logbook should make it easy to repeat a set of tasks under different conditions. For example, one may want to rerun a set of tasks with a slightly changed algorithm.

d) The logbook should have some reporting capabilities, like the number of failed jobs, CPU time and other resources consumed, etc.

e) The logbook should be usable by individuals as well as by larger groups, with the possibility to import and export subsets of the information.

f) The logbook should be usable concurrently by several physicists working together on the same analysis, and each one of them should be able to add information and query the information on tasks entered by everybody else.

In HEPCAL we have described a Job Metadata Catalogue (JMC) that should implement most of the described features. This section further complements and specifies these features. These features can be all implemented by the HEPCAL JMC, or by a combination of other services, as long as they are made available to the grid users.

7.3 PERSISTENT INTERACTIVE ENVIRONMENT

To describe this requirement again we will resort to an imaginary scenario. A physicist comes to CERN to discuss an analysis channel with her colleagues from different institutes. A set of algorithms is quickly developed and tuned interactively, during very fast iterative cycles of discussion, code development and preliminary runs performed on a limited set of data. Results are promising, but more work is needed in tuning the parameters of the selections used and of the algorithms developed, and more statistics is needed. After one week, the visit of the physicist at CERN comes to an end, and she returns to her home institute. Here she wants to continue the work starting exactly where she left it. Before leaving she has saved her interactive session and now she wants to resume it from her institute to continue the work, exactly in the same conditions. To do so, before leaving CERN she has saved her complete session with a grid-wide identifier. Once back in her home institute, she authenticates herself

as a member of her experiment's VO, and then re-opens the saved interactive session. The complete environment is restored and the work can continue where she left it.

This scenario implies the ability for a user to "save" an interactive session with a grid-wide identifier by assigning a name (in user's private namespace) to which she can subsequently refer in order to:

- Get additional information about analysis status, estimated time to completion and so on.
- Find and retrieve partial results of her analysis.
- Re-establish complete analysis environment at later stage.

To describe this example further, we report here an example of pseudo-code that illustrates an analysis session:

```
// create a new analysis Object ( <unique ID>)
Analysis* analysis = new Analysis("MyAnalysis");
// set the program, which executes the Analysis Macro/Script
analysis->Command("MyCommand.sh","file:/home/user/test.C");
// submit query
analysis->Query("200210/V3.08.Rev.04/00110/%galice.root?pt>150.0");
// split the task in at most 10 subjobs
analysis->Split(10);
// submit subjobs
analysis->Submit();
// display job information
analysis->GetInfo();
// download partial/final results and merge them
analysis->GetResults();
// save results in with grid wide identifier
analysis->Write("grid:/home/user/MyAnalysis-10-jun-2008","n"); not clear how this is grid-wide
since it is stored in a file on a single machine!
// quit the session
.q
```

Some time later the same user wants to continue this analysis, possibly from a different location, and the resulting session could look like:

```
// Create an empty analysis
Analysis* analysis = new Analysis();
// read previously saved analysis object
analysis->Read("grid:/home/user/MyAnalysis-10-jun-2008");
// submit new enlarged query
analysis->Query("200210/V3.08.Rev.04/*/%galice.root?pt>150.0");
// split the task in at most 10 subjobs
analysis->Split(10);
// check resource usage
analysis->CheckResources();
// submit subjobs
analysis->Submit();
// display job information
analysis->GetInfo();
// download partial/final results and merge them
analysis->GetResults();
// save results in with grid wide identifier, overwrite previous
analysis->Write("grid:/home/user/MyAnalysis-10-jun-2008","o");
```

```
// quit the session
.q
```

We did not have time to discuss in detail which requirements this scenario implies for the middleware. What we want to achieve is something very close to the essential grid vision. A user should be able to disconnect from one terminal, possibly change continent and then reconnect at another one and continue working as she would do with her laptop when putting it to sleep, and then waking it up again.

We believe that we should extend the discussion with middleware developers to understand these requirements in more detail. We nevertheless identified some environment parameters that we expect to be saved and restored:

- All open DSs, and the current position within them, if sequential access is used.

- All loaded shared libraries.

- All environment variables.

- Temporary files.

- Result of queries.

- Histograms and other objects created in memory.

This is a partial and non-exhaustive list and we leave it here mostly as a pointer to future work on this subject.

7.4 ANALYSIS SOFTWARE DEPLOYMENT

This is another item that we discussed at some length, but where we could not come to a clear conclusion. The issue is that the analysis activity, particularly EUA, will be performed with a mix of "official" experiment software and private user code, which can become rather large. In old times, this was achieved in batch by sending the code with the batch job, to be compiled and linked prior to execution. Interactively this was achieved with interpreted languages, such as FORTRAN in PAW, which was interpreted by the COMIS package, or kumacs, which were interpreted by the PAW KUMAC package. PAW was usable also in batch. Private user libraries could also be accessed and linked, but not with PAW. Modern data analysis programs using shared libraries allow this possibility.

It is quite possible that this model can be extended to the grid-enabled analysis, of course with more modern tools such as ROOT and CINT. However there is an important difference. In a pre-grid situation, the environment where the user job was run was predictable. Users knew where their job was running and they could make sure that the right environment was there (compiler, linker, installed libraries, environment elements, such as variables and shells and so on). In a grid-enabled analysis scenario, the user job will likely be executed on a heterogeneous set of resources. User-imposed limits on the heterogeneity of the computational resources used will limit the computing power available for the job. So the question is how to make sure that the user code, whether it is meant to be interpreted or compiled, and whether or not it makes use of private libraries, can execute and provide a correct result wherever it "lands".

A colourful image evocated during HEPCAL-II discussions was the "paratrooper paradigm". Our code should be capable to be dropped "anywhere", and still perform its mission. There are two extreme simplified scenarios that would allow this to happen.

a) The job carries a "complete" specification of the environment it expects, including version of the compiler and system libraries, together with a full specification of the version of the experiment environment. During the discussion we identified several difficulties in defining what we mean by "complete" specification, but assuming this can be done, this scenario has a serious drawback. The number of Computing Elements satisfying this condition may be at any given time rather small, effectively reducing the amount of resources at the disposal of each job.

b) The job has very relaxed requirements, but it carries with it most of the environment, possibly even including compiler and system libraries. This of course allows the job to run on "any" system that is binary compatible with the libraries carried by it, but at the price of increasing the amount of data shipped even for very small jobs, and of having several versions of the "payload" carried with the job for the different system configurations.

During the discussion we concluded that the most favourable scenario would be between the two described. An interesting idea, already expressed in HEPCAL, was to consider the necessary libraries as input DS. The WMS would then decide whether send the job where these DS are already instantiated (i.e. the software is installed) or to replicate the DSs in a new location (i.e. install the software) and send the job there.

What is missing now is the elaboration of a "most probable" scenario, and the definition of the parameters that define an acceptable "environment" for a given job.

An issue related to this, again discussed but on which no common ground was found other then principle statements, is the validation of a given site. One elegant possibility to solve the above questions is to introduce the concept of validation of a site. A VO should have a "validation suite" for the experiment environment. Supposing this suite exhaustive, then a job would have only to specify that it has to be run on a validated site. However we felt that this is somewhat moving the problem, and more discussion, possibly after some practical experience, is needed to come to precise requirements in this area.

8   RECOMMENDATIONS FOR FUTURE WORK

This section contains a list of the issues that we have identified but that we have not been able to flesh out in sufficient detail in the current HEPCAL round. This was partly due to the limited time we had at our disposal, but also to a lack of experience with grid analysis environments. We believe that may issues will be clearer as soon as a prototype analysis environment is deployed in the framework of LCG-1 and users can acquire some experience with analysis on the grid.

The points we have identified for further discussion are:

a) Requirements on private metadata.

b) Better specification of the interactive persistent environment.

c) Better specifications of the requirements for a resource estimator.

d) Discussion on the importance of the QoS in interactive analysis. For a high-priority interactive task we may want to be able to ask a better service. The simple concept of "priority" of batch jobs becomes more complicated and the concept of Quality of Service has to be introduced. More work is needed to define what are the relevant parameters here and what requirements can be derived on the middleware.

e) We need to elaborate more about pools of resources that can be linked to groups of users (group-level policies, roles, quotas).

f) Current experience of the experiments indicates that it may be beneficial if the middleware could transparently save the results of the query on behalf of the user (see section 6). In some situations this would be fine, but in some situations the user will want the up-to-date latest version of the query every time it is executed, and the full set of data matching the query may be changing rapidly. The user should be able to control this on a per-query basis, with some reasonable user-settable default.

g) As we said above, since the analysis is iterative, there may be lots of datasets left as by-products of the intermediate steps. This may places extra requirements dataset-deletion functionality. This can be done via the "analysis logbook" if implemented, or specifying "expiration dates" for this type of dataset. Of course one must be careful not to cancel DS's useful to other analyses. This issue is briefly discussed in HEPCAL.

h) With many users doing analysis on the grid, the number of datasets registered can potentially be quite large. This seems a trivial point, but the impact of a large number of small datasets being registered in the system needs to be considered carefully by grid software providers and site administrators, so that fast access to data is not compromised.

i) We recognize that we did not describe some emerging scenarios of end-to-end applications in an environment where (a) the resources will tend to be oversubscribed, (b) the collaborations will want to carefully specify and prioritize how the resources are used for the various tasks, and (c) where some parts of a given task will complete successfully, and other parts will (in many cases) fail to complete. There is no consensus among the authors on the relevance of these scenarios in the short and medium term, which may give importance to the user behavior and to his interaction with the system. In particular tools for system performance monitoring and tracking, feedback mechanisms, interactive task execution "guidance" may become important. These topics need to be better understood and require additional work together with careful technology evolution tracking, in order to be included in the next revision of the HEPCAL documents.

The question of output datasets is probably linked closely to the issue of namespaces in the DMS. If we do not have namespaces, the requirements on uniqueness and synchronization for the LDNs in the DMS are much tougher.

9   REFERENCES AND GLOSSARY

9.1   APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

R1   DataGrid-08-TEN-0201-1-11

            'Common Use Cases for a HEP Common Application Layer-HEPCAL', F.Carminati et al,

            http://lcg.web.cern.ch/LCG/SC2/RTAG4

R2   HEPCAL report

10 DESCRIPTION OF USE CASES

USE CASE: PRODUCTION ANALYSIS (PA)

Identifier	*UC#prodanalysis*
Goals in Context	*Create AOD/TAG data from input for physics analysis groups*
Actors	*Experiment production manager*
Triggers	*Need input for "individual" analysis*
Services needed	*job submission* (from GridMiddleware (GMW))
	access to exptMetaData to select (GMW)
	access to calibration/conditions data (GMW)
	access to input data based on the selection (GMW)
	selection of s/w to be used for PA (from expt infrastructure) [or installation on the flight?]
	storing and registration of output data (GMW)
	storing and registration of updated exptMetaData (GMW)
	trigger the creation of replicas of (part or all of) output data and updated exptMetaData (GMW)
Specialised Use Cases	
Pre-conditions	*The experiment management has endorsed the input data and s/w for the analysis.*
Post-conditions	*"Ad-hoc groups" can analyse the output data from this step.*
	The next iteration of Production Analysis can be performed.
Basic Flow	*User specifies job information including*
	Selection criteria;
	Metadata DS (input);
	Information about s/w (library) and configuration versions
	Output TAG and/or AOD DS (typical);
	Program to be run;
	User submits job;
	Program is run;
	Selection Criteria are used for a query on the Metadata DS;
	Event ID satisfying the selection criteria and LDN of corresponding DSs are retrieved;
	Input DSs are accessed;
	Events are read;
	Algorithm (program) is applied to the events;
	Output DS are uploaded;
	exptMetaData is updated;
	Report summarizing the output of the jobs is prepared for the experiment (mgt) (eg. how many evts to which stream, ...) extracting the information from the application and GridMW
Devious Flow(s)	

Importance and Frequency	*High importance. Frequency about a few times per year.*
Additional Requirements	*Input will probably be of the order 10^9 events*
	Will have order of tens of output streams, each dealing with 10^7 (??) events.
	Latency for data access: low ?
	Latency for access to calibration/condition data: low ??
Example	

USE CASE: (SUB-)GROUP LEVEL ANALYSIS (GLA)

Identifier	*UC#mgrpanalysis*
Goals in Context	*Refine AOD/TAG data from a previous analysis step*
Actors	*Analysis-group production manager*
Triggers	*Need input for refined "individual" analysis*
Services needed	*job submission* (from GridMiddleware (GMW))
	access to exptMetaData and groupMetaData to select (GMW)
	access to calibration/conditions data (GMW)
	access to input DataSets based on the selection (GMW)
	selection of s/w to be used for GLA (group specific info from expt infrastructure; e.g., from config mgt tool)
	storing and registration of output DataSet (GMW)
	storing and registration of updated exptMetaData and groupMetaData (GMW)
	trigger the creation of replicas of (part or all of) output data and updated exptMetaData (GMW)
Specializing Use Cases	*Production Analysis*
Pre-conditions	*The physics group management has endorsed the input data and s/w for the analysis.*
Post-conditions	*"Individuals" can analyse the output data from this step.*
	The next iteration of Production Analysis can be performed.
Basic Flow	*User specifies job information including*
	Selection criteria;
	Metadata DS (input);
	Information about s/w (library) and configuration versions
	Output AOD and/or TAG DS (typical);
	Program to be run;
	User submits job;
	Program is run;
	Selection Criteria are used for a query on the Metadata DS;
	Event ID satisfying the selection criteria and LDN of corresponding DSs are retrieved;
	Input DSs are accessed;
	Events are read;
	Algorithm (program) is applied to the events;
	Output DS are uploaded;
	exptMetaData is updated;
	Report summarizing the output of the jobs is prepared for the group (eg. how many evts to which stream, ...) extracting the information from the application and GridMW

Devious Flow(s)	
Importance and Frequency	*High importance. Frequency about a few times per year.*
Additional Requirements	*Input will be of the size of the output streams of "PA" (and can consist of several of the output streams from "PA"*
	Will have order of ten of output streams
	Latency for data access: low ?
	Latency for access to calibration/condition data: low ??
Example	

USE CASE: END USER ANALYSIS (EUA)

Identifier	*UC#enduseranalysis*
Goals in Context	*Find "the" physics signal*
Actors	*End User*
Triggers	*Publish data and get the Nobel Prize :-)*
Services needed	*job submission* (from GridMiddleware (GMW))
	access to exptMetaData and groupMetaData to select (GMW)
	access to calibration/conditions data (GMW)
	access to input DataSets based on the selection (GMW)
Specializing Use Cases	*Production Analysis*
Pre-conditions	*The user has defined a selection and an algorithm to extract the data*
Post-conditions	*Local data analysis can be done*
Basic Flow	*User specifies job information including*
	Selection criteria;
	Metadata DS (input);
	Output DataSet (optional);
	Program to be run;
	User submits job;
	Program is run;
	Selection Criteria are used for a query on the Metadata DS;
	Event ID satisfying the selection criteria and LDN of corresponding DSs are retrieved;
	Input DSs are accessed;
	Events are read;
	Algorithm (program) is applied to the events;
	Output DS are uploaded to user's local store;
Devious Flow(s)	
Importance and Frequency	*High importance. Frequency a few times per week.*
Additional Requirements	*Input will be of the size of the output streams of "PA" (and can consist of several of the output streams from "PA"*
	Will produce (several) output files which will be copied to "local storage"
	Latency for data access: low ???
	Latency for access to calibration/condition data: low ??
Example	