LIGO's Virtual Data Requirements

**Bruce Allen   Ewa Deelman   Carl Kesselman   Albert Lazzarini
Tom Prince   Joe Romano   Roy Williams**

Document **LIGO-T000135-00-D**
December 2000

This document aims to represent the GriPhyN/LIGO problem as three layers (Figure 1):

1. The Physics Layer, which looks at GriPhyN/LIGO from the point of view of physics research: what data is collected and what kind of analysis is performed on the data.
2. The Virtual Data Layer, which describes the opportunities of the Virtual Data Grid for support of the LIGO data and the operations performed on it.
3. The GriPhyN Layer, which describes how the virtual data will be stored and handled with the use of the Globus Replica Catalog and the Metadata Catalog. Also, how the data transformations will be provided and their results stored.

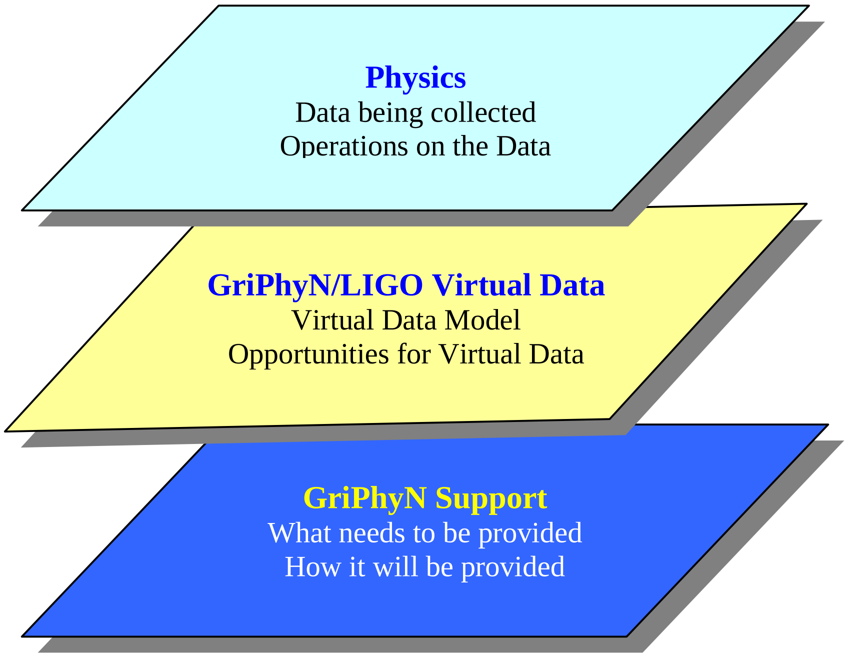


Figure 1: GriPhyN Design Decomposition

1 Physics

LIGO (Laser Interferometer Gravitational-Wave Observatory) is a multi-university (Caltech, MIT, …) project designed to build detectors for gravitational waves. Currently, there are two observatories in the United States, one of which recently went through the "first lock" phase[1], in which initial calibration data was collected. More information on the project can be found at http://www.ligo.caltech.edu.

The observatories aim to detect gravitational waves predicted by Einstein's theory of relativity, which described gravity as a curvature of the fabric of time and space. The gravitational waves are believed to be generated by moving masses, however they are so weak, that so far they have not been directly

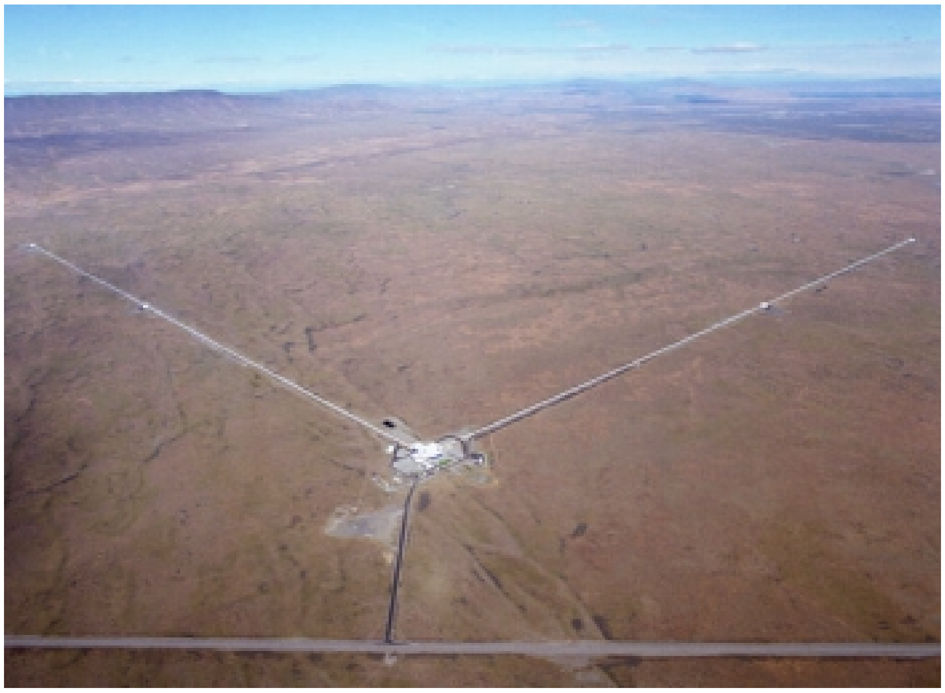


Figure 2: High aerial shot of the LIGO Hanford Observatory shows the twin arms of the installation stretching into the Washington desert. The two-kilometer optical cavities span between the corner station building and the buildings midway down each arm. Evacuated beam tubes between the buildings provide a clear path for the laser light. A four-kilometer interferometer is currently being installed that shares these same beam tubes[1].

detected (although their indirect influence has been observed in 1974 in a binary pulsar[2]). One of the measurable manifestations of gravitational waves is expected to be such pulsars. As the binary pulsars draw closer together, the gravitational wave is expected to increase until the actual coalescence and then eventually decrease. Although, the occurrence of coalescing pulsars is believed to be infrequent, estimates predict one observation per year[3]. Additionally, LIGO will be used to detect the mergers of black holes and neutron stars and black holes with black holes, and these are expected to occur more frequently. Other signals may come from supernova explosions and from "starquakes" in neutron stars. Besides abrupt signals such as these, continuous-wave signals are expected, for example from rapidly spinning neutron stars.

[1] http://www.ligo.caltech.edu/LIGO_web/firstlock/
[2] "Laser Interferometer Gravitational Wave Observatory Data Analysis System", by J.K. Blackburn, LIGO document #P000009-00-E, http://admdbsrv.ligo.caltech.edu/pubdcc
[3] "An Antenna Tuned to the Songs of Gravity", by G.H. Sanders and D. Beckett, Sky & Telescope, October 2000.

The current Hanford interferometer (Figure 2) has two two-kilometer “arms” attached to a central station. Each of the arms contains test masses, which can be acted upon by gravitational waves. A passing gravitational wave is expected to reduce the distance between the test masses in one arm and increase it in the other (because of the L-shape of the arms). To detect that change, a laser light is generated at the central station (central point of the L), then split into two beams and sent into each arm of the interferometer. The changes in the mass positions cause a misalignment of the light in the two instrument arms and that interference can then be detected. However, phenomena such as earthquakes, acoustic noise or laser fluctuations can also cause beam interference. In order to obtain a clean gravitational wave signal, significant amount of data needs to be collected (including data from seismometers, microphones, etc.) and analyzed (for example, to eliminate noise).

The raw data collected during experiments is a collection of continuous time series at various frequencies (e.g., 16kHz, 16Hz, 1Hz, etc.). The amount of data expected to be generated and cataloged each year is in the order of tens of terabytes. The gravitational channel is less than 1% of all data collected. Analysis on the data is performed in both time and Fourier domains. Requirements are to be able to perform single channel analysis over a long period of time as well as multi-channel analysis over a short time period.

1.1 Data in LIGO

The data in LIGO is gathered as a collection of time series ranging in frequency from 16 kHz down to 1 Hz. Each scalar time series is represented by a sequence of 2-byte integers. Time is represented by GPS time, the number of seconds since an epoch in 1981, and it is therefore a 9-digit number, possibly followed by 9 more digits for nanosecond accuracy. Data is stored in **Frame** files, a standard format accepted throughout the gravitational wave community. Such a file can hold a set of time-series with different frequencies, together with metadata about channel names, time intervals, frequencies, file provenance, etc. In LIGO, the Frames containing the raw data encompass an interval of time of one second and result in about 2Mb of data.

In addition to the raw time series, there are many derived data products. Channels can be combined, filtered and processed in many ways. In addition to the time domain, data can be transformed into other domains, such as frequency (Fourier transform, power spectrum), the wavelet basis, or other spaces. Knowledge is finally extracted from the data through pattern matching algorithms; these create collections of candidate events, for example inspiral events or candidate pulsars. Each event comes with a significance (signal-to-noise ratio): we will pay close attention to highly significant (high SNR) events, and make statistical analyses of low SNR events.

1.2 Data organization in LIGO

Much effort in LIGO has been devoted to the development of a naming scheme for the data, which is appropriate for LIGO Data Analysis System (LDAS). The LIGO data model splits data from metadata explicitly. Bulk data is stored in Frame files, as explained above, and metadata is stored in a relational database, IBM DB2. There is also an XML format called LIGO-LW[4] for representing annotated, structured scientific data, that is used for communication between the distributed services of LDAS.

In general, a file may contain more than one Frame, so we distinguish a new word: FrameFile, for a file that may contain many frames. Raw data files contain only one frame, and they are named by the interferometer that produced the data (H: Hanford, L: Livingston), then the 9-digit GPS time corresponding to the beginning of the data. (Note: in 2011, this time will become a 10-digit quantity). There is a one or two letter indication of what kind of data is in the file, (F: full, R: reduced, T: trend, etc). So an example of a raw data frame might be H-276354635.F.

[4] Also called Extensible Scientific Data Language, http://www.cacr.caltech.edu/XSIL/

For long-term archiving, rather larger files are wanted than the 2 megabyte, one second raw frames, so there are collection-based files, some as multi-frame FrameFiles, some as Unix tar format. In either case, an additional attribute is in the file name saying how many frames there are, for example H-276354635.F.n200 would be expected to contain 200 frames.

One table of the metadatabase contains FrameSets, which is an abstraction of the FrameFile concept, which recognises that a FrameFile may be stored in many places: perhaps on tape at the observatory, on disk in several places, in deep archive. A FrameSet is a FrameFile together with the physical locations where it is stored. One of the primary reasons for the metadatabase is to keep a searchable catalog of FrameSets.

Each frame file records the names of all of the approximately one thousand channels that constitute that frame. In general, however, the name set does not change for thousands or tens of thousands of one-second frames. Therefore, we keep a ChannelSet object, which is a list of channel names together with an ID number. Thus the catalog of frames need only store the ChannelSet ID rather than the whole set of names.

The metadatabase also keeps collections of Events. An event may be a candidate for an astrophysical event such as a black-hole merger or pulsar, or it may refer to a condition of the LIGO instrument, the breaking of a feedback loop or the RF signal from a nearby lightning strike. The generic Event really has only two variables: type and significance (also called Signal to Noise Ratio, or SNR). Very significant events are examined closely, and insignificant events used for generating histograms and other statistical reports. The event type distinguishes the remaining data in the record, which may be times, filter number, or other numbers. Also the Event may store Blob (Binary large object) data, whose meaning depends on the event type, for example a snippet of a time series or a power spectrum.

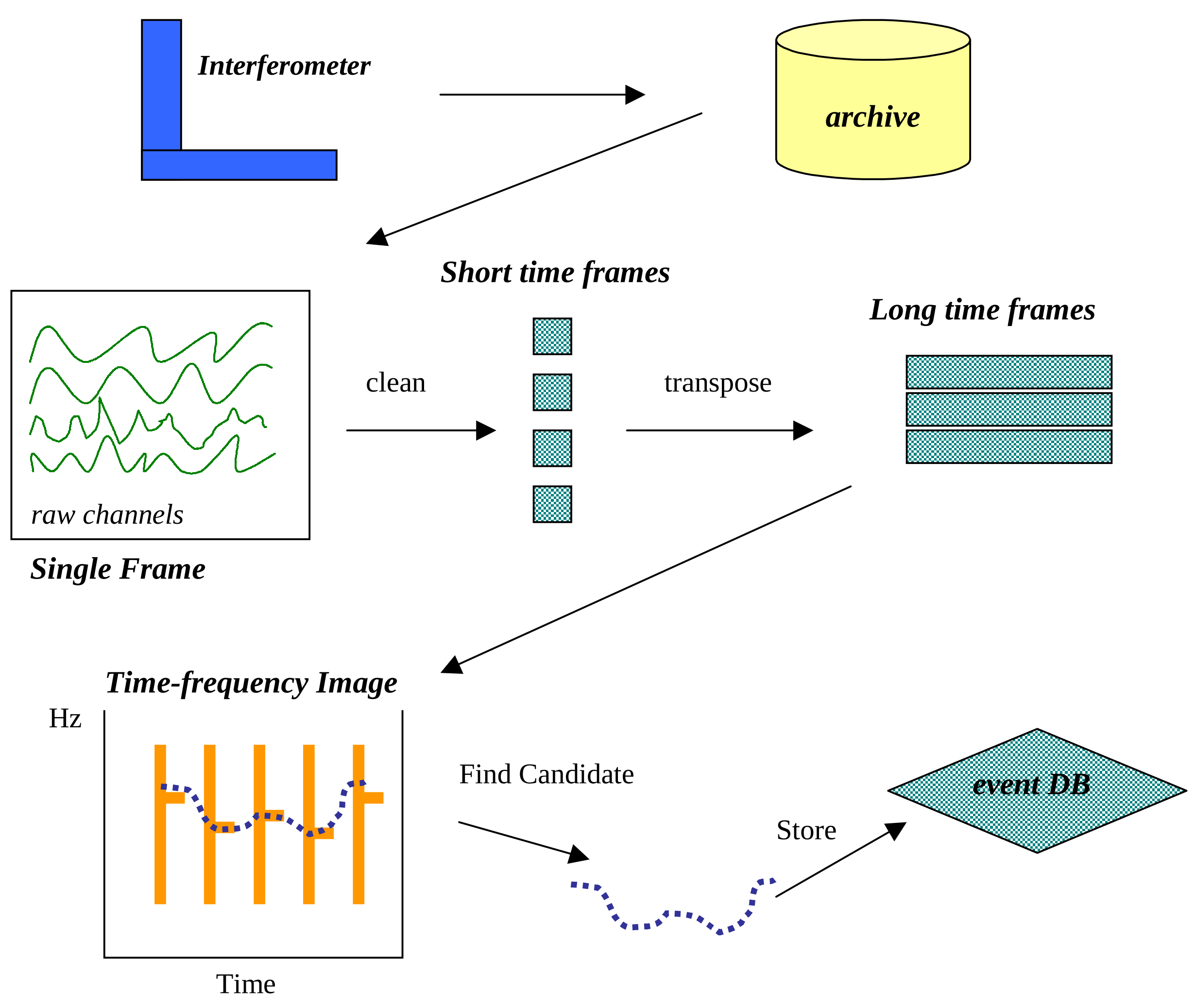
1.3 Computational aspects, Pulsar Search

LDAS is designed primarily to analyze the data stream in real time to find inspiral events, and secondarily to make a long-term archive of a suitable subset of the full LIGO data stream. A primary focus of the GriPhyN effort is to use this archive for a full-scale search in this archive, for the continuous-wave sources. This search can use any amount of computational resources, since the search can be done at essentially arbitrary depth and detail.

If a massive, rotating ellipsoid does not have coincident rotational and inertial axes, then it emits gravitational radiation. However, it is very weak unless the object is extremely dense, rotating quickly, and has a large quadrupole moment. While the estimates of such parameters in astrophysically-significant situations are vague, it is expected that such sources will be very faint. The search is computationally intensive primarily because it must search a large parameter space. The principle dimensions of the search space are position in the sky, frequency, and rate of change of frequency.

The search is implemented as a pipeline of data transformations (Figure 3). As a first step, instrumental data is stored into an archive. Next, since, as noted above, the raw data comes from the instrument as short (1 second) Frames with all the channels, some processing geared towards the removal ("cleaning") of certain instrumental signatures needs to be done. For example, naturally occurring seismic vibration can be subtracted from the data using the channels from the sensitive seismometer that is part of the LIGO data stream. The short-duration frames are then combined into much longer frames in a transpose operation, and further data-conditioning filters are applied. We now have removed instrumental signature, and measured the actual deformation of space-time geometry at the LIGO site.

Figure 3: Data Analysis Pipeline for Pulsar Search



This data is then used to conduct the pulsar search. Since the data needed to conduct the search is a long (~6 months, 2×10^{11} points) stretch of a single channel—the gravity-wave channel, the 1D time-series is broken into many small segments, and the power-spectra of these segments are stacked to make a large frequency-time image, perhaps 4×10^{5} on each side. The pulsar search consists of searching for coherent signals in this image. A source would appear on the frequency-time image as a wavering line, whose frequency might be 1 kHz, but modulated by several Hz over periods of 1 day and 1 year, and also with a secular variation due to slowing of the source.

The pulsar search can be parallelized by splitting the possible frequencies into bins, and each processor searching a given bin. The search involves selecting sky position and frequency slowing, and searching for statistically-significant signals. Once a pulsar source has been detected, the result is cataloged as an event data structure, which describes the pulsar's position in the sky, the signal-to-noise ratio, time etc…. LIGO has invested a significant amount of effort into software development[2], however, efficient data delivery is still an open problem.

1.4 Event Identification Computation Pipeline

During the search for astrophysical events a long duration one dimensional time series is processed by a variety of filters, most of which (~90%) consist of FFT transformations. These filters than produce a new time series which represents the signal to noise ratio in the data. A threshold is applied to each of the new time series in order to extract possible events. These events are cataloged in the LIGO database. If an event appears to be interesting from the point of view of astrophysics, it is further investigated. In order to determine if the event is significant, the raw data containing instrumentation channels needs to be reexamined. It is possible that the occurrence of the event was triggered by some phenomena such as lightning strikes, acoustic noise, seismic activity, etc… These phenomena are recorded by various instruments present in the LIGO system and can be found in the raw data channels. To eliminate the influence of the above occurrences, multiple instrumentation channels must be examined and compared to the occurrence of the event. The location of the raw data channels can be found in the LIGO database. Since the event is pinpointed in time, only small portions of the many channels (that possibly needs to be processed) need to be examined. This computational pipeline clearly demonstrates the need for efficient indexing and processing of data in various views:

- a long time interval single channel data, such as the initial data being filtered, and
- the many channel, short time interval such as the instrument data needed to add confidence to the observation of events.

2 GriPhyN/LIGO Virtual Data

A key component needed to implement the data pipeline is a virtual data service; a system to dynamically create data products requested during the various stages. The data could possibly be already processed in a certain way, it may be in a file on a storage system, it may be cached, or it may need to be created through computation. The full elaboration of this system will allow complex data pipelines to be set up as virtual data objects, with existing data being transformed in diverse ways.

In the extreme, only raw data needs to exist in some archive(s). The rest of the requests for data, such as obtaining a single channel of data ranging over a large time interval can be derived from the original data set. At the other extreme, every single data product that has been created (even if it represents an intermediate step not referred to again) can be archived. Clearly, neither extreme is an efficient solution, however with the use of the Virtual Data Grid (VDG) technology, one can bridge the two extremes. The raw data is of course kept and some of the derived data products are archived as well. In this light, there are many opportunities for virtual data technology along the entire data pipeline, as each step does not need to refer to the raw data, but make use of data products already computed. Additionally, data can be distributed along various storage systems, providing opportunities for intelligent data retrieval and replication.

The virtual data in the two examples above can be listed as:

- single frames containing all the channels of raw data
- cleaned, short time duration frames
- long time duration, single channel raw data frames
- signal to noise ratio time series (resulting from applying filters to raw time series)
- short interval power spectra for a given channel
- frequency images for a given channel and time interval

- candidate signals from a given frequency-time image
- fully characterized event

VDG will provide transparent access to above virtual data products. To efficiently satisfy requests for data, the VDG needs to make decisions about the instantiation of the various objects. The following are some examples of VDG support for LIGO data:

- Raw data vs. cleaned data channels. Most likely, only the virtual data representing the most interesting clean channels should be instantiated.
- Various views of data. Support for easy access to data composed from smaller pieces of data, such as access to long duration frames that could have been already processed from many short duration frames. Not all such long time data frames need to exist physically. Most likely, only the often-used gravitational wave channel frames should be archived, but many of the engineering channels do not.
- Time-frequency image. Most likely the entire frequency-time image will not be archived. However, all its components (short power spectra) might be instantiated. The VDG can then compose the desired frequency-time images on demand.
- Signals retrieved from images. Although many signals can be extracted from a time-frequency image, some signals might have a very high noise-to-signal ratio and thus might not be often accessed. In this case, the VDG might not instantiate the corresponding virtual data. However, very strong signals, likely to be accessed often, might be replicated more than once.
- Interesting events. Given a strong signal representing a particularly promising event, the engineering data related to the time period of the occurrence of the event will most likely be accessed and filtered often. In this case, the VDG might instantiate preprocessed engineering data channels, data that might otherwise exist only in its raw form.

2.1 Simple Virtual Data Request Scenario

We assume that the interaction of the LIGO system with the Virtual Data Grid will be made in the form of requests. First let us take a simple model of the virtual data Universe, by assuming a Cartesian product of a time interval with a set of channels: In reality, of course, it is not so simple. There are sections of missing data, multiple interferometers, data recorded in different formats, and so on. The interesting data has been transformed and filtered, there are multiple interferometers. Thus there is (in principle) a map from (channel C, time interval I) to a 2-byte integer whenever the channel is in C and the time in I. The raw data are one-second files containing all the channels and these can be processed (for some special channels) to on-the-fly files with a much longer time interval, but each with only one channel.

In the following, we consider requests for the data from a subdomain of the full domain. Each request is for the data from a subset of the channels for a subinterval of the full time interval. Thus, a request might be written in as:

T0=700004893, T1=700007847; IFO_DCDM_1, IFO_Seis_*

where IFO_DCDM_1, is a channel, and IFO_Seis_* is a regular expression on channel names that maps to the three coordinates of the seismometer IFO_Seis_x, IFO_Seis_y, and IFO_Seis_z. Our first task is to create a naming scheme for the virtual data object, each name being a combination of the name of a time interval and the name of a set of channels.

This could be satisfied if there is a suitable superset file in the replica database, for example this one:

T0=70004000, T1=700008000; IFO_DCDM_1, IFO_Seis_x, IFO_Seis_y, IFO_Seis_z, IFO_DP

Thus, we need to be able to decide if a given Virtual Data Object (VDO) contains another, or what set of VDO's can be used to create the requested VDO. Tools could be used to combine multiple files (C_1,I_1), (C_2,I_2), ... perhaps as:

- The new file could be (union C_i, intersect I_i), a *channel union tool*, or
- The new file could be (intersect C_i, union I_i), an *interval union tool*.

We could thus respond to requests by composing existing files from the distributed storage to form the requested file.

In addition to these compositions, there will be requests for transformations performed on the data, such as request for files, which are Fourier transforms of the time series data, as those used in the frequency-time image.

Issue: We need to know more about the transformations (capture the knowledge about how they are performed which results are temporary and which need to be persistent). We need to know how many transformation there are (order of magnitude), are they well defined in terms of inputs, outputs and computational requirements. This information is needed in order to be able to execute the transformations on data sets as well as to determine how to describe them.

2.2 Generalized Virtual Data Description

The goal of the GriPhyN system is to make it easy for an application or user to access data. As a starting point, we will require that all requests be made in the form:

(1) a range of time t_0 to t_1 (specified in GPS seconds) , followed by

(2) a list of virtual channels (described below).

Consequently, GriPhyN would return a set of virtual channels for the specified time interval.

Virtual Channels

A virtual channel is a time series, like a real channel, but it may be *derived* from actual channels, but not correspond to a channel in the raw data. Some examples of virtual channels are:

- An actual recorded channel, "raw".
- An actual recorded channel, but downsampled or resampled to a different sampling rate.
- An arithmetic combination of channels, for example $2C_1 + 3C_2$, where C_1 and C_2 are existing channels.

- The actual channel, convolved with a particular calibration response function, and scaled. For example, the X component of the acceleration.
- The virtual channel might be computed from the actual data channels in different ways depending upon what time interval is requested (eg, the calibrations changed, the channels were hooked up differently, etc).
- A virtual channel could be defined in terms of transformations applied to other virtual channels.

In short, the virtual channels are a set of **transformations** applied to the real data.

The set of virtual channels would be extendable by the user. As the project progresses, one may want to extend the set of virtual channels to include additional, useful transformations. Thus, if a user is willing to define a virtual channel by specifying all the necessary transformations, it will be entered in the catalog and will be available to all users, programs, and services. New channels can be created from the raw data channels by parameterized filters, for example decimation, heterodyning, whitening, principle components, autocorrelation, and so forth.

Data naming

A crucial step in the creation of the GriPhyN Virtual Data model is the naming scheme for virtual data objects. Semantically, we might think of names as a set of keyword-value pairs, extended by transformations, perhaps something like (T0=123,T1=456,Chan=[A,B*,C?]).pca().decimate(500). The first part in parentheses is the keyword-value set, the rest is a sequence of (perhaps parameterized) filters. We could also think of using names that contain an SQL query, or in another language. The syntax could also be expressed in other ways, as XML, or with metacharacters escaped to build a posix-like file name. We could use a syntax like protocol://virtual-data-name to express these different syntax in one extensible syntax. However, decisions on naming Virtual Data must be premised on existing schemes described in Section 1.2.

3 GriPhyN Support

The goal of GriPhyN is to satisfy user data requests transparently. When a request for a set of virtual channels spanning a given time interval is made, the application (user program) does not need to have any knowledge about the actual data location, or even if the data has been pre-computed. GriPhyN will deliver the requested virtual channels by either retrieving existing data from long term storage, data caches containing previously requested virtual channels, or by calculating the desired channels from the available data (if possible).

When satisfying requests from users, data may be in slow or fast storage, on tape, at great distance, or on nearby spinning disk. The data may be in small pieces (~1 second) or in long contiguous intervals (~1 day), and conversion from one to another requires computational resources. A given request for the data from a given time interval can thus be constructed by joining many local, small files, by fetching a distant file that contains the entire interval, or by a combination of these techniques. The heart of this project is the understanding and solution of this optimization problem, and an implementation of a real data server using GriPhyN tools.

3.1 User Requests

The initial implementation of theGriPhyN system will accept requests in the semantic form:

$\mathbf{t_o,t_1;\ A,B,C,}$ … where t_o, t_1 is a time interval, and A, B, C, … are virtual channels.

We assume that the order in which the channels are listed does not affect the outcome of the request. The syntax of the virtual channel is yet to be determined. In the simplest form, a virtual channel resulting from a transformation Tr_x on a channel C_y would be $Tr_x(C_y)$; additional attributes (such as transformation parameters or other input channels) can be specified as additional parameters in the list: $Tr_x(C_y, C_z, t_0,$ ….), depending on the transformation. The transformation specific information will be stored in the **Transformation Catalog** described below.

3.2 Data access, cost performance estimation

The request is initially received by the **Request Manager** (Figure 4) and sent for processing to the **Metadata Catalog**, which provides the set of logical files that satisfies the request, if such exists. The files names are retrieved from the Metadata Catalog based on a set of attributes. A possible metadata catalog is the *SRB* system developed at SDSC http://www.npaci.edu/DICE/SRB/index.html.

The logical files found in the Metadata Catalog are sent to the **Replica Catalog**, which maps them to a unique file id. The information about the actual file existence and location (provided by the Replica Catalog) are passed to the Request Manager, which makes a determination about how to deliver the data.

If the requested data is present, the Request Manager still needs to determine whether it is cheaper to recalculate the data or access it. When considering the cost of referencing data, the cost of accessing various replicas of the data (if present) needs to be estimated. If the data is not present, the possibility and cost of data calculation needs to be evaluated. In order to make these decisions, the Request Manager queries the **Information Catalog**. The latter can provide information about the available computational resources, network latencies, and bandwidth, etc…

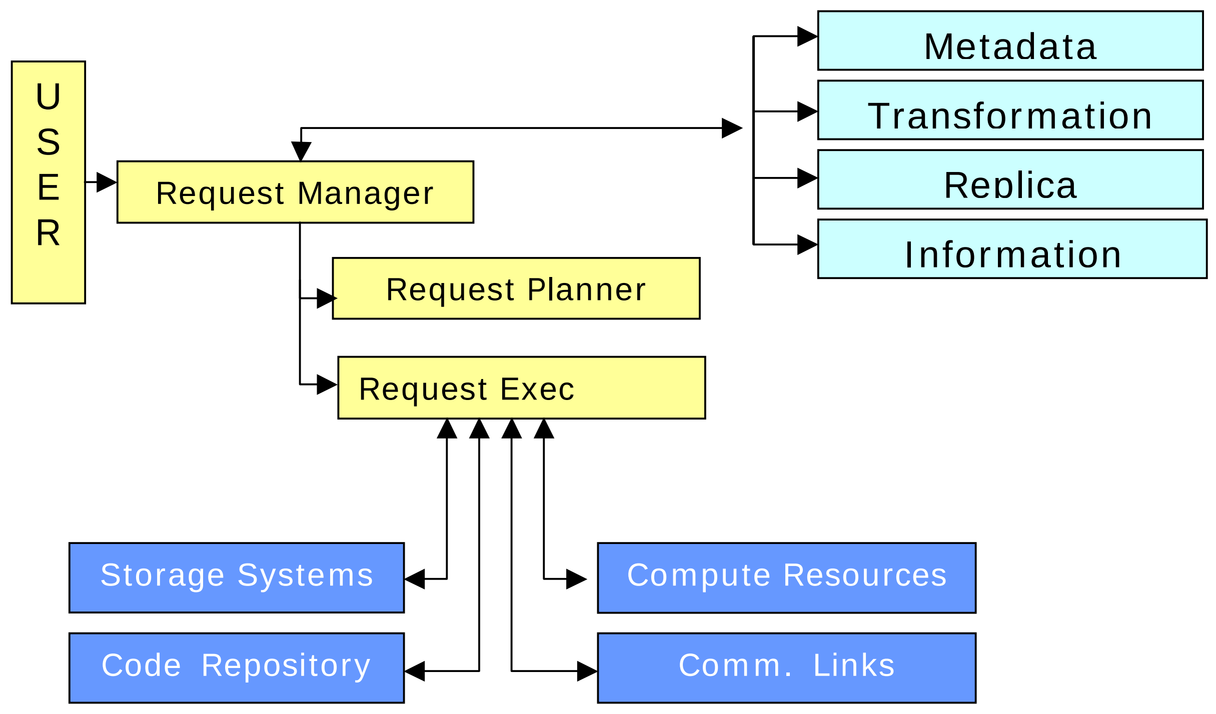


Figure 4: GriPhyN Architecture

The **Request Planner** is in charge of creating a plan for the execution of the transformations on a given data set and/or creating a plan for the retrieval of data from a storage system. The Request Planner has access to the system information retrieved by the Request Manager. To evaluate the cost of re-computation, the cost of the transformations needs to be known. This information, as well as the input and parameters required by a given transformation, code location, etc… are stored in the **Transformation Catalog**. Initially, it will be up to the user to specify the transformation, to indicate whether the transformation is deterministic, and, if so, how the cost of the transformation varies with the size of the set being transformed (linear, quadratic, N log N, …). This would enable the Request Planner to estimate the relative costs of computation vs. caching, and so on. The system would possibly keep a record of how long the various transformations took, and could use this performance history to estimate costs. This record and the analytical performance estimates will be maintained in the Transformation Catalog.

The performance data needed in the evaluation of re-computation and replica access costs (such as network performance, availability of computational resources, etc.) will be provided by information services such as the *NWS*: ws.npaci.edu/NWS/ or *GIIS*: www.globus.org/toolkit/information-infrastructure.html.

Once the request planner decides on a course of action the **Request Executor** is put in charge of carrying out the plan which involves the allocation of resources, data movement, fault monitoring, etc. The Request Executor will use the existing Globus infrastructure to access the data and the computational Grid.

As a result of the completion of the request, the various catalogs might need to be updated.

3.3 Proactive Data Replication

Simply, just retrieving data from the replica catalog is not sufficient. The system must take a proactive approach to creating replica files and decide whether the results of transformations will be needed again. For example, if there is a request for a single channel spanning a long time interval and the replica catalog contains only files which are multi-channel spanning short time periods, then a significant amount of processing is needed to create the requested file (many files need to be opened and a small amount of data needs to be retrieved from each of them). However, once this transformation is performed, the resulting data can be placed in the replica catalog for future use, thus reducing the cost of subsequent requests. New replicas should also be created for frequently accessed data, if accessing the data from the available locations is too costly. For example, it may be useful to replicate data that initially resides on tape to a local file system. Since the Request Manager has information about data existence, location and computation costs, it will also be responsible for making decisions about replica creation.

Issues:

- Since physicists use both single channel data over long periods of time and multiple channel data over short time intervals, we need to provide a dual view of the data: both time indexed and channel indexed. We also need to do that efficiently, therefore we need to consider how we will view the data: for example as a small chunk: a 1-second Frame with a single channel, or will we allow files that span multiple channels and longer time intervals. Obviously, indexing into a regular data set is easiest, but the cost of data retrieval can be much higher.

- Not all data will be viewed as a series, as some data analysis will produce events and these need to be accommodated in the virtual data world.
- We need to decide whether to use the LIGO metadata catalog (need to find out more about it), or start something new with SRB for the purposes of the project. Similarly, the use of the LIGO DB2 replica catalog needs to be evaluated.
- We need to capture the knowledge behind the transformations and their inter-relationship in order to be able to index the data efficiently, and provide consistent results.
- We need to be able to determine how long the results of a given transformation are kept. Maybe allowing the user to indicate the data priority would be a good solution. We can have a range of priorities: do not catalog, keep if space available, … never delete.
- We need a decision-making facility that can support proactive replica creation.

Existing Software:

- LIGO Data Analysis System (LDAS).
- Frame manipulation: we are currently developing Frame manipulation tools based on the FrameCPP library provided by LDAS.
- Metadata catalog: SRB: http://www.npaci.edu/DICE/SRB/index.html, MCAT: http://www.npaci.edu/DICE/SRB/mcat.html, LIGO DB2, running under Solaris, accessible by a Tcl-based language.
- Replica catalog: Globus: http://www.globus.org, LIGO DB2
- Transformation catalog: Globus, Condor: http://www.cs.wisc.edu/condor/.
- Request planner: HRM , Matchmaker: http://www.cs.wisc.edu/condor/publications.html.
- Request executor: HRM, Condor, GridFTP: http://www.globus.org/datagrid/, SRB, WAFT: http://www-cse.ucsd.edu/~marzullo/WAFT/index.html.