LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATOR

- LIGO -

CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	**LIGO-T990001-06**	**E**	02/19/99

LIGO Data Analysis System Preliminary Design.

A. Lazzarini, J.K. Blackburn, S.B. Anderson

Distribution of this draft:

This is an internal working note
of the LIGO Project.

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (818) 395-2129
Fax (818) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

TABLE OF CONTENTS

1 Introduction 4
1.1 Purpose 4
1.2 Scope 4
1.3 Changes to Conceptual Design 6
1.4 Prototype Test Results 7
1.4.1 RAID: Testing and benchmarking on-line data acquisition disk cache system. 7
1.4.2 Clustered parallel computing (Beowulf system) 10
1.4.3 Data transmission tests 11
1.4.4 Database archival technology 11
1.5 Acronyms 12
1.6 Applicable Documents 13
1.6.1 LIGO Documents 13
1.6.2 Non-LIGO Documents 14
2 General description 14
2.1 Product Perspective 14
2.2 General Requirements 15
3 Design Overview 16
3.1 System Hardware Architecture 16
3.2 LDAS Software Design 20
3.2.1 Distributed LDAS APIs 22
3.2.2 Interprocess communications via sockets 24
3.2.3 Extensions of the baseline (genericAPI) template: managerAPI 25
3.2.4 Software Development Tools 30
3.2.5 Database Management System for LDAS Metadata 32
3.2.6 Data Types and Databases 32
3.3 LDAS Software Configuration 33
3.3.1 LDAS Software Development Resources 33
3.3.2 LDAS Runtime Resources 34
3.4 LDAS Startup Procedures 34
3.4.1 Initialization 34
3.4.2 Abnormal Runtime Condition Management 34
3.5 Interfaces 35
3.5.1 Connections to CDS and Observatory General Computing networks 35
3.5.2 Interfaces to CACR 37
3.5.3 User interfaces to LDAS 38
3.6 Wide Area Network 39

4 LDAS CONFIGURATION 39
4.1 Data Server: 41
4.2 Relational Database Server: 44
4.3 Data Conditioning & Regression Processor: 44
4.4 LDAS Control & Monitoring Server: 45
4.5 Compute Server: 46
4.6 Data Ingestion: 47
4.7 LDAS data analysis workstations: 48
4.8 Data Storage (On-line): 48
4.9 Archive Server (Off-line only): 49
5 LAYOUTS 51
5.1 Hardware locations at observatories 51
Appendix A Software Validation 53

LIST OF FIGURES

Figure 1 LDAS Functional Units. LAN or WAN refers to the networks either within a Laboratory site (local area) or between sites (wide area).15
Figure 2 LDAS On-Line Configuration Overview 18
Figure 3 LDAS Off-Line Configuration Overview. Note that the left hand panel represents hardware configurations that may exist at a number of sites.19
Figure 4 LDAS Software Design 21
Figure 5 Nature of the Distributed LDAS API...23
Figure 6 Communications Provided by GenericAPI...24
Figure 7 LDAS Command Flow Using the ManagerAPI 28
Figure 8 LDAS-CDS Interface at the observatories...36
Figure 9 LDAS Networking Interfaces at the Observatories 37
Figure 10 LDAS Networking Interfaces to CACR at Caltech 38
Figure 11 Mapping of LDAS APIs onto the on-line hardware. 40
Figure 12 Mapping of the LDAS APIs onto the off-line hardware. The HPSS will have its own software system which is not developed by LIGO.41
Figure 13 Tape Storage Sequence 49
Figure 14 HPSS Configuration for LIGO 51
Figure 15 Layout of LDAS components within OSB rooms 52

LIST OF TABLES

Table 1 RAID Performance dependence on number of drives 8
Table 2 RAID Dependence on file system type...8
Table 3 RAID Dependence on chunk size 9
Table 4 RAID Performance for directly attached disks...10
Table 5 RAID Performance of Solstice Disksuite on directly attached disks 10
Table 6 Transmission performance of various networks for sustained throughput...............11
Table 7 List of Applicable LIGO Documents...13
Table 8 WAN/LAN Connectivity among LIGO Laboratory Sites 39

1 INTRODUCTION

The LIGO detector and associated PEM system will generate large amounts of continuous data (approximately 6 MB/s for LIGO Hanford Observatory alone). The data must be handled so that a number of different analyses can be supported. Analyses include both on-line processes (at the observatories) and off-line processes (utilizing archived data). These uses include:

- Computationally-intensive and parallel pipeline processes to detect the presence of astrophysical signals in the datastream; these must run at an effective speed to keep up with the data acquisition process.
- Data preprocessing and conditioning to provide reduced data products; these processes produce data subsets and also provide the datastream for pipeline processing.
- Data distribution for detector diagnostics.
- Data archival and retrieval (including a limited archival capacity at the observatories for diagnostics purposes); this includes archival and retrieval of metadata summaries as well as time-series ("raw") data; these processes will support data exploration, algorithm prototyping and other analyses by the LIGO/LSC data analysis teams.

The system of software and hardware which performs these tasks is denoted the LIGO Data Analysis System (LDAS). The present intent is to participate in certain aspects of the hardware implementation of LDAS collaboratively with Caltech's Center for Advanced Computing Research (CACR). This development will be undertaken in accord with an MOU that has been established between LIGO Laboratory and CACR. For additional information to CACR, please see the CACR home page and links therein, **http://www.cacr.caltech.edu.**

1.1 Purpose

This technical note presents the preliminary design for the LIGO Data Analysis System (LDAS). This design follows from the analysis requirements laid out in the *LDAS Design Requirements Document, LIGO T970159* and represents an update to the conceptual design presented in *LDAS Conceptual Design, LIGO-T970160*. This update also incorporates relevant recommendations made by the Design Requirements Review Panel in its report. It is the intent of this document to present a baseline design which is flexible enough to incorporate new features in the future.

All designs presented in this document can use technology available at the time of writing. However, computer hardware continues to improve in its performance-to-cost ratio. LIGO intends to delay as long as possible procurement of hardware to take full advantage of this technological trend. Moreover, the approach will be to incrementally add capacity only on an *as needed* basis. This will open the design to better, faster, cheaper options in the future. Also, while specific manufacturers and model numbers of equipment are shown as part of the design in this document, they may not necessarily be the ones used, but rather are meant to be representative of the equipment to be used.

1.2 Scope

LDAS will be implemented to meet the requirements set forth in the LDAS DRD. The lists below appear in the DRD [LIGO-T971059] and have not changed appreciably [*changes are noted with parenthetical remarks*]. LDAS shall provide the facilities to:

[A] Provide on-line data analysis capability to each of the LIGO Observatories. This capability includes the following

- A means to extract physical strain from the interferometer output(s) and to utilize relevant ancillary channels (e.g., PEM) to remove instrumental or environmental signatures.
- A means to process strain data through real-time detection algorithms for both performance monitoring and scientific purposes. Sufficient computing power to allow processes to keep up with the incoming datastream shall be provided. Sufficient margin shall also be provided to accommodate maintenance down times and other system inefficiencies.
- A means to cross-correlate data (either time series or event lists) from multiple interferometers.
- A means to store data frames and analysis results (local to the observatory) to short term storage media. This functionality will be provided by the LIGO DAQS resources, with possible augmentation by LDAS.
- A means to access both “live” and short term archived data via the local area and wide area networks. Access shall be subject to available bandwidth and demand.
- Means to retrieve, concatenate and extract specific channels of recent data from the on-line storage system.
- Sufficient automation to run continuously and autonomously during periods of normal operation.
- A means to display and visualize results of analyses.
  [*NOTE:* Since the DRR, the interface to detector diagnostics (GDS) has been refined. As a consequence of this, data visualization will be handled using any one of several commercial or publicly available packages, such as MATLAB [from MathWorks] or ROOT [from CERN]. LIGO, together with VIRGO, has developed frame I/O libraries to enable researchers to manipulate frame data within these environments.]

[B] Provide off-line data analysis capability. This capability is likely to be concentrated at one LIGO Laboratory site but shall be available “seamlessly” throughout the Laboratory. This capability includes

- Sufficient computing margin to enable multiple analyses to be conducted in parallel.
- A means to reduce the raw data to science data representing calibrated GW strain data and a reduced subset of ancillary data and a data quality descriptor.
- A means to archive, retrieve and distribute reduced datasets acquired over a period of time at least 2 years in duration. [***NOTE:*** Since the DRR, a LIGO I science run has been defined which will run for 2 years. This defines the length of the data archive required.]
- A means for duplicating reduced datasets either for backup or for distribution.
- A means to access the data archive via LAN and WAN by the LIGO Laboratory and LIGO Scientific Collaboration with sufficient bandwidth to support database manipulation at the off-line site by remote users.

- A standardized interface for visualization tools to allow experimenters to see and interpret results from various analyses. ***See NOTE in [A] above***.

Specifically not considered to be within the scope of the LDAS are:

- Data analysis functions performed at centers other than the LIGO Laboratory facilities.
- The on-line diagnostics system used for stimulus-response characterization, transfer function determination, and calibration functions. However, it is expected that software developed for the LDAS will find utility within the diagnostics system (and vice versa). Specifically, the algorithm numerical libraries for data analysis are being developed jointly. In this manner, the same effort will serve multiple functions and there can be a standardized convention for carrying out the same analysis irrespective of which LIGO component is performing the analysis (i.e, LDAS or GDS).
- Simulations shall be provided separately from, but coordinated with, LDAS. The primary interface with simulated data for purposes of data analysis will be through the use of LIGO standardized data types,i.e., either frame or LigoLW data formats.

1.3 Changes to Conceptual Design

Changes and refinements of design definition which have occurred since the DRR are discussed below.

Hardware Design

Observatories

- Interface to DAQS has been revised. The original design called for a multi-host RAID disc system providing independent access to the on-line disc farm by CDS and by LDAS. Prototyping experience at Caltech identified bandwidth limitations in this approach. Subsequent discussions with CDS led to a design in which CDS would provide a Sun 450 server with 4 CPUs, 1GB RAM and ~300+GB of disc space. In one possible implementation scheme, a second LDAS server could be connected to the CDS machine with a dedicated high-speed port. In this way, CDS can distribute data to detector operations functions [including diagnostics] and LDAS would distribute data for analysis functions.
  Note that initially it is planned to install the needed LDAS software onto the CDS 450 framebuilder to more closely integrate the acquisition and distribution functions. If experience dictates that it is desirable to separate these functions, then a separate LDAS server, with a high throughput connection [e.g., fibre channel or ATM] to CDS/DAQS, will be implemented. This will be done using TCP/IP protocols LDAS will provide tape writing facilities at the observatories for archiving data.
- Tape robot for observatory data retrieval. At the DRR, the requirement was identified to provide for data retrieval from the local [observatory] near-term archive of data tapes.

Off-line

No significant changes have been made.

Software Design

- Multi-layered API design. Key elements of the layered software architecture have been defined since the DRR. Tcl/Tk has been chosen as the scripting command language (i.e. “steering language”) for the software environment.
- LIGO Lightweight data format has been defined [based on XML].
- The database management system will be a relational database. LIGO is presently prototyping a design using IBM’s DB2 and will use portable ODBC interfaces to isolate the LDAS-specific design from the DBMS as much as possible. This will permit future changes of DBMS with the least impact.
- Interface to users will be based on the Tcl/Tk plugin for browsers. Uses of LDAS software outside a browser environment will be supported using the Tcl/Tk application environment (Wish shell) to execute LDAS user interfaces.

Networks

No significant changes have been made. However, suitable hardware will be procured to permit use of OC12 communications bandwidth whenever it becomes necessary without the need to replace major switching hardware.

1.4 Prototype Test Results

1.4.1 RAID: Testing and benchmarking on-line data acquisition disk cache system.

Testing of a multi-port SCSI Redundant Array of Inexpensive Disks (RAID) system indicates that this technology will not scale to the needs of LIGO (at least not with the current generation of technology). A decision was made that higher-end RAID systems are too expensive and that the data acquisition architecture would be switched from a multi-host/single RAID system to a single-host with multiple processors and directly attached disks.

The following configuration was tested:

- Sun Ultra30 (300MHz/128MB).
- Seagate 9GB 7200RPM drives (ST19171W).
- Solaris2.6 (process class = real-time).
- UFS Size is for (newfs -i 16384 -m 0).
- UFS and directio().
- dd bs=1024k.

Here MB and GB refer to 2^{20} and 2^{30} Bytes, respectively. ***Table 1 - Table 5*** report the observed RAID performance under different test conditions.

Table 1 RAID Performance dependence on number of drives[a]

RAID level	Description	UFS (3 disks / 2GB files)				UFS (6 disks / 2GB files)			
		Size (MB)	Read	Write	Read + Write	Size (MB)	Read	Write	Read +Write
0	Striping without parity		17.0	14.0	1.3 + 12.9 =14.2	51687	17.9	15.3	7.3 + 9.3 =16.6
1	Mirroring								
0+1	Striping and mirroring								
4	Striping with fixed parity drive					43052	17.8	11.6	8.0 + 6.4 =14.4
5	Striping with floating parity drive		11.8	8.5	3.5 + 7.0 =10.5	43052	17.3	11.7	7.3 + 6.6 =13.9
JBOD	"Just a Bunch of Drives"								

a. Chunk size = 256 blocks.

Table 2 RAID Dependence on file system type[a] [b]

RAID level	Description	Raw Partition (6 disks /2GB files)				UFS (6 disks / 200MBfiles in cache)				NFS (6 disks /2GB files)	
		Size (MB)	Read	Write	Read + Write	SIZE (MB)	Read	Write	Read + Write	Read	Write
0	Striping w/o parity	52098	20.9	19.3	10.2 + 10.2 = 20.4	51687	38.5	37.7	49.5		
1	Mirroring										
0+1	Striping and mirroring										

Table 2 RAID Dependence on file system type[a] [b]

RAID level	Description	Raw Partition (6 disks /2GB files)				UFS (6 disks / 200MBfiles in cache)				NFS (6 disks /2GB files)	
		Size (MB)	Read	Write	Read + Write	SIZE (MB)	Read	Write	Read + Write	Read	Write
4	Striping with fixed parity drive	43414	20.2	13.1	9.3 + 7.0 = 16.3	43052	37.4	21.6	37.7		
5	Striping with floating parity drive	43414				43052				6.6	5.1
JBOD	"Just a Bunch of Drives"	52098	21.1	17.8	10.3 + 9.0 = 19.3						

a. Chunk size = 256 blocks
b. NFS is to Sun Ultra30 (140MHz/128MB) over LANE/ATM OC-3.

Table 3 RAID Dependence on chunk size

Chunk Size (Blocks)	Raw Partition (6 disks / 2GB files)					
	RAID 0			RAID 4		
	Read	Write	Read + Write	Read	Write	Read + Write
32	19.1	9.1	0.6 + 8.7 = 9.3	19.4	5.0	6.8 + 3.3 = 10.1
96	19.7	16.5	6.8 + 10.8 = 17.6			
256	20.9	19.3	10.2 + 10.2 = 20.4	20.2	13.1	9.3 + 7.0 = 16.3
1024	21.2	20.4	6.0 + 14.2 = 20.2			

Table 4 RAID Performance for directly attached disks[a]

RAID level	Description	Direct Disk/Raw Partition (3 disks/2GB files)			Direct Disk/Raw Partition (6 disks/2GB files)		
		Read	Write	Read + Write	Read	Write	Read + Write
JBOD	"Just a Bunch of Drives"	31.9	24.8	14.1 + 14.0 = 28.1	36.9	36.7	18.3 + 18.3 = 36.7

a. Limited to the 38MB/s peak bandwidth of a single SCSI channel

Table 5 RAID Performance of Solstice Disksuite on directly attached disks[a] [b]

RAID level	Description	Disk Suite/Raw Partition (6 disks/200MB files in cache)			Disk Suite/UFS (6 disks/200MB files in cache)		
		Read	Write	Read + Write	Read	Write	Read + Write
0	Striping w/o parity	33.3	---		29.9	22.4	4.7 + 17.8 = 22.5
1	Mirroring						
0+1	Striping and mirroring						
5	`Striping` w/ floating parity drive	32.3	4.9		29.3	2.3	4.3 + 1.6 = 5.9

a. Limited to the 38MB/s peak bandwidth of a single SCSI channel
b. RAID-5 with interlace = 32 blocks.

See http://www.srl.caltech.edu/personnel/sba/raid/index.html for further details

1.4.2 Clustered parallel computing (Beowulf system)

A prototype Beowulf cluster of 16 Pentium II processors has been installed at Caltech. The current configuration is 8 motherboards with 2 333MHz Deschutes processors and 512MB of memory each. Initial acceptance testing is still proceeding to obtain a stable hardware and software configuration and develop an easy to maintain system where additional or replacement processors may be added to the system in just a few minutes.

Initial benchmarks indicate that for template matching, this cluster is capable of ~1GFLOP FFT performance for large (1Mpt) out-of-cache transforms and ~2.5GFLOP for smaller transforms (1kpt). Work is currently under way to benchmark distributed memory FFTs (~100Mpt) and out-of-core FFTs (~10Gpt) for pulsar searches.

The particular cluster was purchased as an integrated system from Alta Technologies. As part of the prototyping activity, LIGO is also evaluating the cost-effectiveness of purchasing completely integrated systems versus in-house built systems. Preliminary experience indicates that procuring an integrated hardware system will involve significant in-house effort to install the latest versions of MPI and linux OS kernels. This is because the versions available bundled with the integrated systems are typically one or two versions behind the latest ones being used at CACR.

1.4.3 Data transmission tests

Several technologies have been benchmarked for suitability as the primary network for off-line LIGO data analysis and archiving. Emphasis is being place on ATM with current demonstrated transfer rates of 6MB/s from LIGO in the CACR HPSS. It is anticipated with the recent upgrade from OC-3 to OC-12 and additional HPSS hardware being installed at CACR that during 1999 realistic LIGO bandwidths and load usage will be testable. The following table lists sustained throughput over several networks:

Table 6 Transmission performance of various networks for sustained throughput

Network/Protocol		*Mode*	*Speed [MB/s]*
HIPPI	FP (Paragon <--> HPSS server)	Memory to Memory Parallel File System to Memory	50 30
	TCP/IP	HPSS server to HPSS server Paragon to HPSS server	6 2.4
ATM (OC3) <--> HIPPI	TCP/IP	Sun to HPSS server HPSS server to Sun Sun to IBM/SP2	1 0.2 2
ATM (OC3)	TCP/IP	Sun to Sun IBM/SP2 to IBM/SP2 Sun to HPSS server	9-14 10-15 7-9

1.4.4 Database archival technology

1.4.4.1 Testing and benchmarking HPSS:

As of this writing, work continues on gaining practical experience with HPSS to determine its viability as the primary storage system for the LIGO off-line data analysis system. Several hundred Gigabytes (including all of the 1997 40m data) have been archived and successfully retrieved from the CACR HPSS system at sustained data rates from 1-6MB/s. To date, all of the major problems encountered in this work have been adequately solved. Further critical evaluation is needed to determine both the hardware and operating costs for a system suitable for the needs of LIGO.

LIGO is positioned to leverage on a planned upgrade to HPSS by CACR. This upgrade is scheduled to take place in Jan/Feb 1999 and will result in a system that should closely match the needs of LIGO, allowing for a well informed decision to be made regarding the sizing of the archive, choice of drives and network connectivity (this type cooperation is identified in the LIGO-CACR MOU). In particular, the upgrade of the CACR system will consist of the following configuration changes:

1. Increased data mover performance by moving the HPSS processes onto higher performance 4-way processors running as nodes with in an SP2 frame,
2. Improved network connectivity through the addition of a High Performance Gateway Node (HPGN) which will connect the SP2 nodes to several external high speeds networks, e.g., HIPPI and ATM, and
3. Increased storage capacity to 300+TB by utilizing a recently acquired StorageTek SILO with Redwood tape drives.

In addition, both CACR and LIGO have made substantial commitments to utilizing ATM as a high speed network between supercomputers and HPSS. Until Jan 99, host-to-host bandwidths between LIGO and CACR were limited by a single OC-3 connection to a single HPSS servers (~10MB/s). However, with the recent addition of a CACR ATM switch and the upgrading of the LIGO switch at Caltech, there is currently an OC-12 interconnection with a theoretical 4-fold bandwidth improvement. When combined with the new HPGN interface to HPSS LIGO should be able to benchmark the full bandwidth requirements of the LIGO off-line data analysis system.

1.4.4.2 Research into recording technology

Research into emerging technologies is being actively pursued to see if it is technically and fiscally feasible to permanently archive all of the LIGO data stream. One promising technology is optical tape. A start-up company named LOTS is currently going into beta testing with an optical tape device which is write-once-read-many (WORM). We recently learned that IBM is also working on a similar storage technology. The drive and tapes are form compatible with the current tape robotics in the CACR HPSS system and hence are of interest to both CACR and LIGO. Each 1/2" optical tape cartridge holds 1.3TByte of data for a projected cost of \$200-\$300. If this technology comes to market with actual devices it presents the possibility of permanently (shelf life estimated at greater than 50y) archiving the raw LIGO data stream for 75-100K/year in media costs (an order of magnitude reduction in price).

1.5 Acronyms

•	API	Application Programming Interface
•	ATM	Asynchronous Transfer Mode (network communication protocol)
•	CACR	Center for Advanced Computing Research at Caltech
•	CDS	Control and Data System
•	DAQS	Data AcQuisition System

•	DRD	Design Requirements Document
•	LDAS	Data Analysis System
•	DDU	Diagnostic Distribution Unit
•	GUI	Graphical User Interface
•	HPGN	High Performance Gateway Node (to HPSS)
•	HPSS	High Performance Storage System (tape archival robot)
•	IFO	Interferometer
•	IP	Internet Protocol
•	LIGO	Laser Interferometer Gravitational Wave Observatory
•	MTBF	Mean Time Before Failure
•	MTTR	Mean Time To Repair
•	NFS	Network File Services
•	RAID	Redundant Array of Inexpensive Disks (disk farm)
•	RH	Relative Humidity
•	SCU	Signal Conditioning Unit
•	TBD	To Be Determined
•	TCP	Transport Control Protocol
•	UDP	User Datagram Protocol.

1.6 Applicable Documents

1.6.1 LIGO Documents

Table 7 List of Applicable LIGO Documents

Description	*Document ID*
LDAS White Paper	LIGO-M970065
LDAS Design Requirements Document	LIGO-T970159.
LDAS Conceptual Design Document	LIGO-T970160.
Data Format Specifications	
Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD)	LIGO-T971030
LIGO Lightweight Data Format Specification	LIGO-T980091.
LIGO Metadata, Event and Reduced Data Requirements	LIGO-T980070.
LDAS Software Specifications:	
GenericAPI Baseline Requirements	LIGO-T980093.
GenericAPI Baseline Specification	LIGO-T980094.

Table 7 List of Applicable LIGO Documents

Description	*Document ID*
GenericAPI.tcl source code map -- genericAPI.tcl ManagerAPI Baseline Requirements	on-line TclDoc[a] LIGO-T980115.
ManagerAPI.tcl source code map -- managerAPI.tcl	on-line TclDoc[a]
ManagerAPI.rsc resource file -- managerAPI.rsc	on-line TclDoc[a]
asstMgrAPI.tcl source code map -- asstMgrAPI.tcl	on-line TclDoc[a]
FrameAPI Baseline Requirements	LIGO-T9800117.
FrameAPI.tcl source code map -- frameAPI.tcl	on-line TclDoc[a]
FrameAPI.tcl emergency procedures source code map -- frameEmProc.tcl	on-line TclDoc[a]
FrameAPI.tcl operator procedures source code map -- frameOpProcs.tcl	on-line TclDoc[a]
MetadataAPI Baseline Requirements	LIGO-T980119
DataConditioningAPI Baseline Requirements	LIGO-T990002

a. link accessible via http://docuserv.ligo.caltech.edu/~prince/LDCG.html. Note that some of these documents are still in the process of creation.

1.6.2 Non-LIGO Documents

None

2 GENERAL DESCRIPTION

2.1 Product Perspective

The LDAS is divided into two primary functional units as shown in ***Figure 1***. Both functional units are designed to operate under identical software environments.

These are:

- On-line LDAS: The on-line segment consists of two functionally identical but independent units located at the LIGO Observatories (Hanford, WA and Livingston, LA). Each provides the capability to run real-time detection algorithms and also the ability to provide end-to-end insight into interferometer behavior for specific signal types. The system interfaces to the LIGO DAQS for accessing the real-time data. It also has a limited one-way interface (data displays for operators) for interferometer diagnostics to provide performance metrics (e.g., based on non-Gaussian noise characteristics). The site LAN may be used to access the data cache by scientific analysis workstations at the site. There will also be the ability to access the real-time data remotely by other LIGO Laboratory sites within the bandwidth limitations of the LIGO WAN.

- Off-line LDAS: The off-line component will be situated at Caltech and provides the following functions: data reduction and compression for long term archival; data retrieval; off-line pipeline-based analyses of LIGO data. Remote access to the archive will be via a wide area network (WAN) capable of providing high throughput access to the archive. The baseline design relies on the regular transfer of data tapes from the LIGO Observatories to the off-line analysis center. Once data tapes are received, they will be ingested to extract/compress/refine the science data for the permanent archive. The ingestion process will include calibration, regression (if suitable), data QA checking using the multiple ancillary channels, channel reduction, data compression and trend summarization (statistical descriptors of raw time series data). These procedures will be needed in order to reduce the overall volume of the archived database.

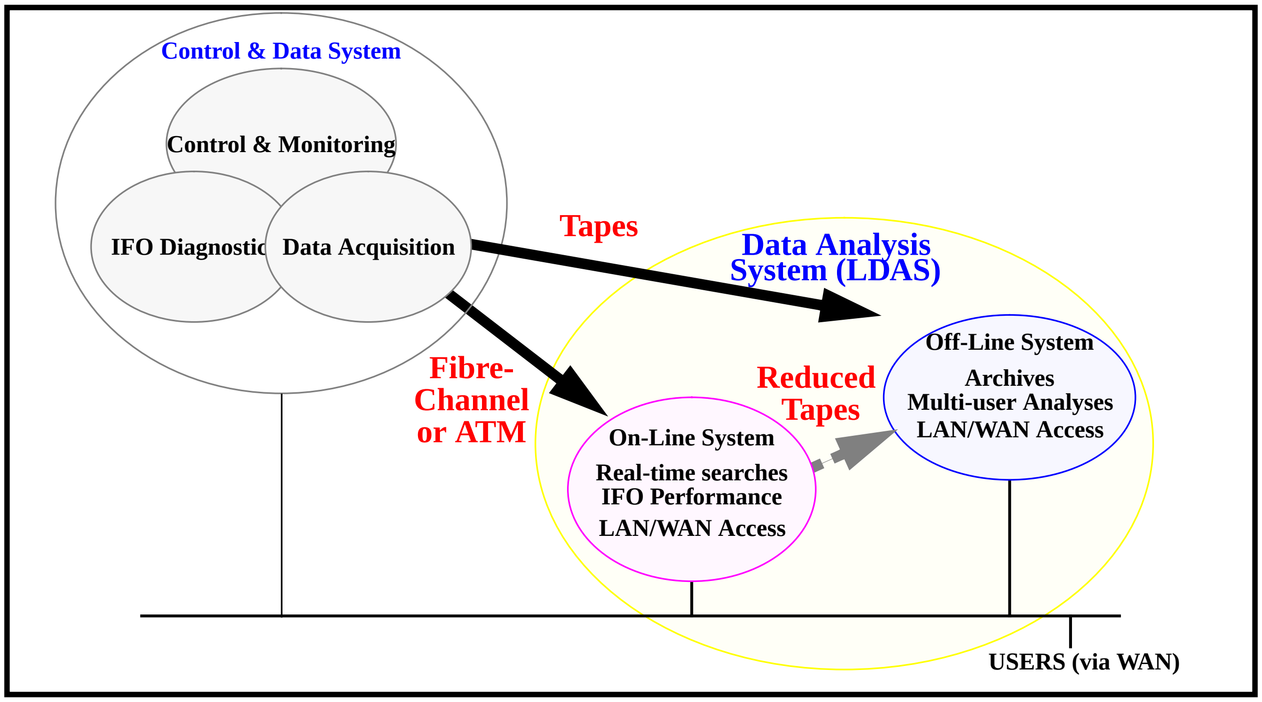


Figure 1 LDAS Functional Units. LAN or WAN refers to the networks either within a Laboratory site (local area) or between sites (wide area).

The on-line and off-line systems are independent; however, where it is required, those critical databases or components will be “mirrored” between them. The systems will be highly integrated so that, for example, a remote user will be able to access either the on-line or off-line system with nearly identical user interfaces and commands.

When the technology becomes available (and affordable) LIGO will make a transition from a media-based data exchange from observatories to the data repository. to a network-based data transmission mode.

2.2 General Requirements

The specific requirements which this system must meet are given in the DRD, LIGO-T970159. The primary requirements on the system, which drive the design, are to be able to:

- Provide on-line analysis at the observatories for a limited class of detection algorithms running at sufficient speed to keep up with the acquisition rate. These are designed to flag the possible presence of certain types of gravitational waveforms as soon as possible after acquisition of data.
- Provide access to LIGO data from all LIGO Laboratory sites and also from member institutions of the LIGO Scientific Collaboration for the LIGO I search.
- Process and reduce the raw LIGO datasets at the off-line center to prepare the data for archival storage and retrieval.
- Provide computational and storage resources for off-line analysis using the archived data
- Provide a flexible design which can be reconfigured to reflect new analysis or computational requirements as they evolve.

3 DESIGN OVERVIEW

3.1 System Hardware Architecture

The general configuration of LDAS and its relationship to other LIGO systems is shown in ***Figure 2*** and ***Figure 3***.

- **Metadata/Data Server:**
  - Provides access to frame data and metadata. This high-performance unix server that is connected via a dedicated high-speed port to the CDS framebuilder. It complements the CDS Network Data Server (NDS, see Section ***3.5***, Interfaces and ***Figure 8*** therein) and provides much of the same functionality to users who are not authorized to request data through CDS (i.e., are not involved in machine performance and diagnostics). The disk cache on which the recent data are resident is part of CDS/DAQS.
    See the note in Section***1.3***, “Hardware Design, Observatories”
  - Users request data via the LDAS front end interface9s), depicted in ***Figure 4*** (LDAS Users) and ***Figure 7*** (CMD, GUI and WEB APIs). Simple filtering operations such as data decimation and channel subset selection from raw frame data can be provided by the data serve directly: more complex tasks, such as spectral cross-correlation and regression, will be handed off to the signal conditioning processor.
  - Users may request returned data in the form of frames or lightweight data sets, written to files or via ftp download to the user’s workstation. Depending on resources and bandwidth, it will also be possible to open a socket interface to stream data (a predetermined and finite amount) directly to a remote process.
  - The data server provides a means to isolate the (real-time) CDS LAN from users not involved in detector operations. It supports users logged on to the LDAS LAN (private and dedicated to data analysis at the observatory) or who are logged on the LHO General Computing LAN, or who are remotely located and coming in through the LIGO WAN.
  - The data distribution server also accesses LIGO metadata created by GDS, CDS/DAQS

and LDAS at the observatory and which are stored and retrieved by the (commercially licensed) database management system or DBMS.

- **<u>Database Server</u>:**
  - Database management system server to perform sorts/queries on the metadatabase.
- **<u>LDAS Control & Monitoring Server</u>**:
  - Configures, starts and synchronizes operation of the LDAS parallel computing resources, load balancing. etc.
  - Contains libraries and shared objects for exporting to rest of the system.
- **<u>Compute Server (linux cluster)</u>:**
  - Provides MPI-based multi-processor parallel computation for a number of analysis processes, including signal processing and detection algorithms.
  - Manages the template database needed for optimal filtering.
  - Generates events for further processing.
- **<u>Data Conditioning & Regression Processor</u>:**
  - One or more workstations or dedicated hardware units (e.g., DSPs) that provide a variety of functions to pre-process raw data for analysis. This includes calibration; regression; Fourier transformation; data formatting for MPI processing; etc. Simpler filtering tasks, such as decimation and resampling, are handled by the data distribution server (see above).

    For concreteness, a typical LDAS signal conditioning task is to take the GW channel and a (small) number of other interferometer channels for which the cross-spectral covariance matrix is already known (e.g., from prior diagnostics tests and calibrations) and to produce a best-estimate regressed GW channel. This FT is re-sampled at lower frequency resolution (corresponding to lower bandwidth) and is passed on to the compute server for MPI-based Wiener filtering. The same data can be provided via the LDAS LAN to users or can be written to media for later off-line analysis.
- **<u>Data Ingestion</u>:**
  - Provides the means to read/write media at observatories and archive.
  - Provides the means to generate reduced datasets from archived data.
- **<u>LIGO Archive Server (HPSS; Off-line only)</u>:**

Experience of the users of large volumes of data at major facilities (e.g., SDSC) indicates that if a standard set of [IBM] hardware is used it is possible to obtain 95% uptime with HPSS. While software ports of HPSS to non-IBM platforms are currently under way and may offer cheaper hardware configurations in the long run, LIGO does not have the resources nor the mandate to play a significant role in porting or further developing HPSS. LIGO plans to obtain a proven stable, scalable solution.

  - Provides the means and resources to archive LIGO reduced data in frames for ~2 years. Permits retrieval of data to the local disk cache for data analysis.
- **<u>LDAS LAN(s)</u>:**
  - Each site (LHO, LLO, Caltech, MIT) will have an LDAS LAN, based on ATM, to provide for high bandwidth data transmission between processors.

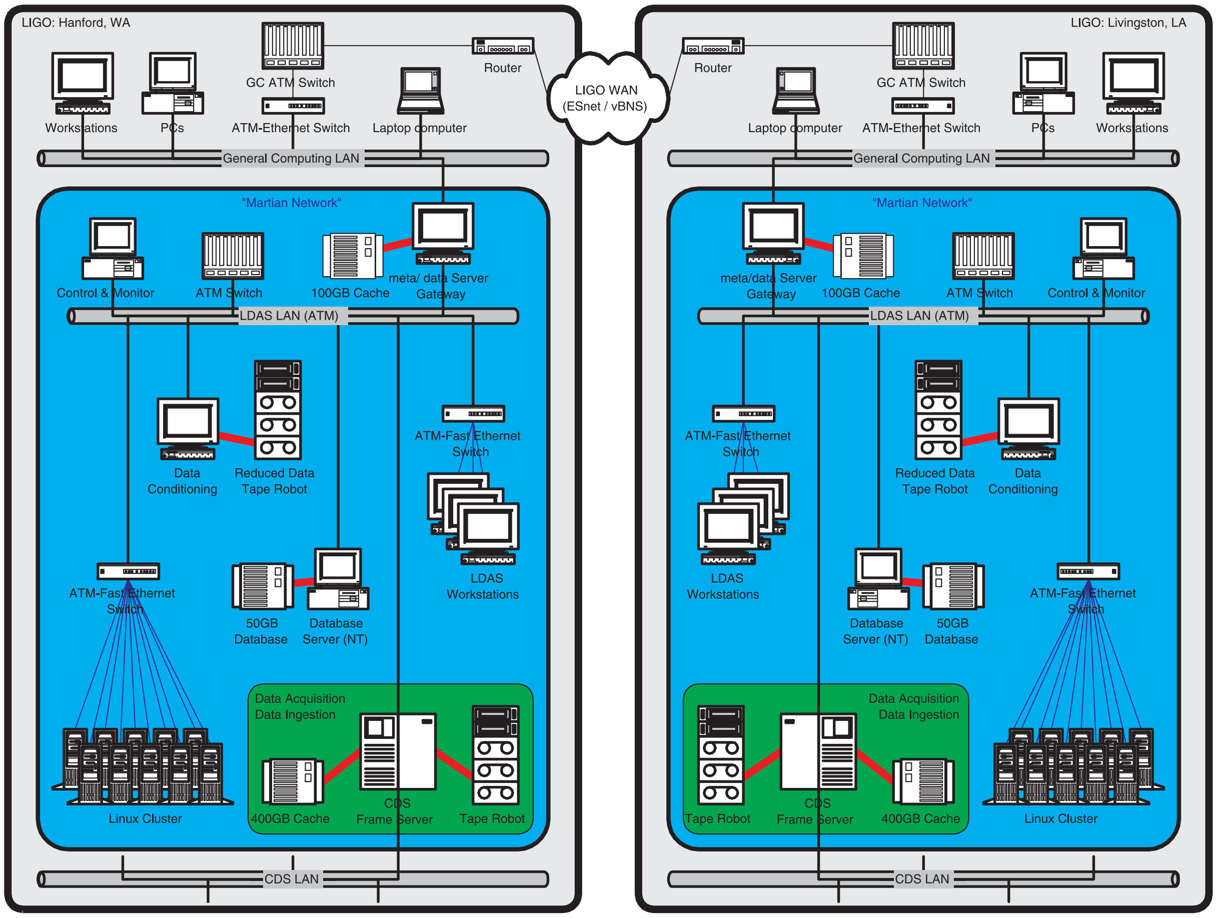


Figure 2 LDAS On-Line Configuration Overview

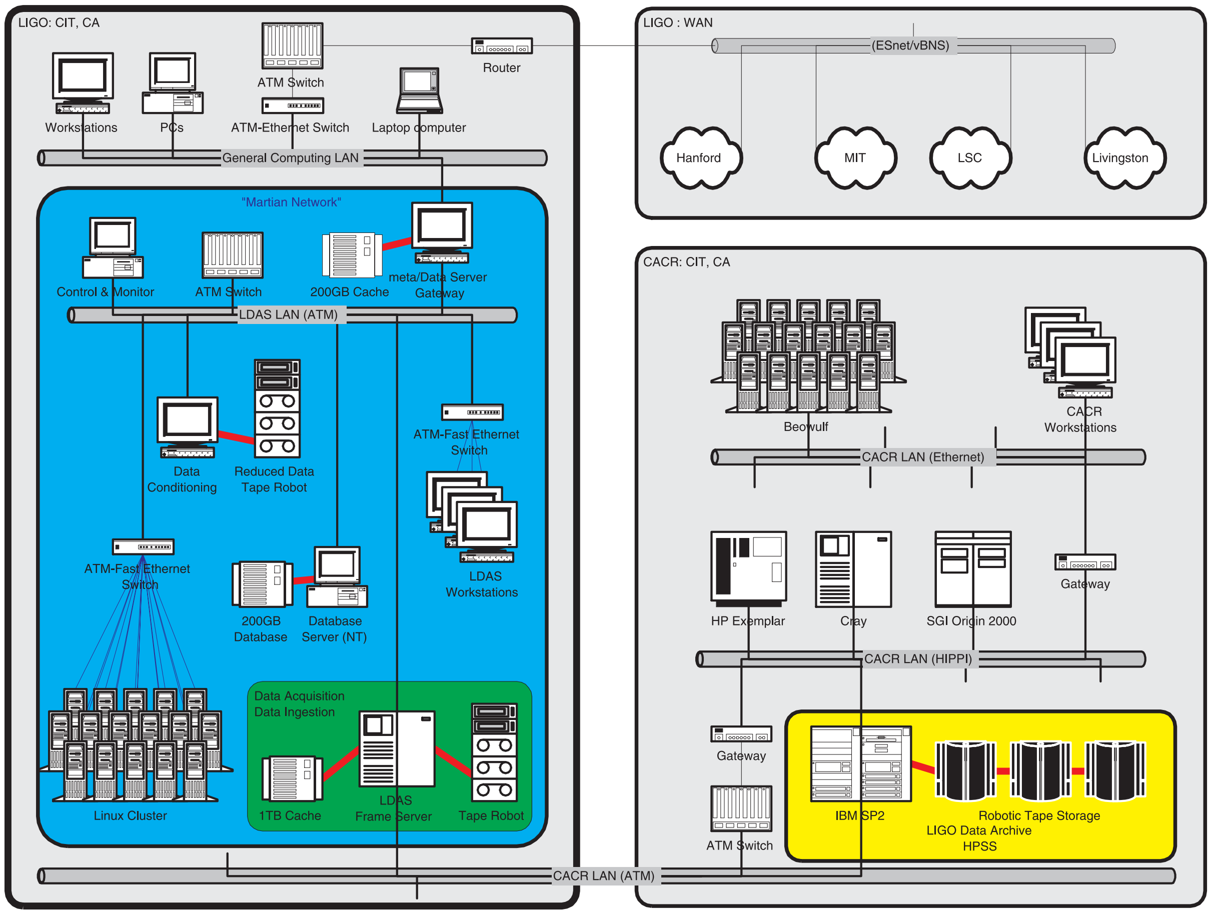


Figure 3 LDAS Off-Line Configuration Overview. Note that the left hand panel represents hardware configurations that may exist at a number of sites.

- **<u>LDAS WAN</u>:**
  - The LIGO sites are linked at $\geq$T1 bandwidth by one of several methods. LHO is linked over the DOE ESnet via PNNL in Richland, WA. LLO, Caltech and MIT are linked over vBNS to the internet.

The general features of the design follow from the following requirements:

- The ability to provide flexible but powerful parallelized computing resources;
- The ability to serve multiple (~< 5 - 6) researchers accessing the on-line and off-line data-bases.
- The ability to provide pipeline-based streaming analysis of real-time and off-line data at rates commensurate with the acquisition rates of new data.
- The ability to provide access to resources to users across LIGO Laboratory and Collaboration Institutions. In general, access to resources will be subject to terms of usage that will be defined by LIGO Laboratory and the LSC.

3.2 LDAS Software Design

The primary software components of LDAS are shown in ***Figure 4***. These components are discussed in further detail in Section ***4*** where the design of each of the LDAS major subsystems is presented. The design is based on a modular and layered paradigm which provides an efficient implementation of a distributed processor configuration. At the time of this writing, one of the most effect methodologies for the design of software architectures such as LDAS is to "design for change". This is what we have intended in the development of LDAS.

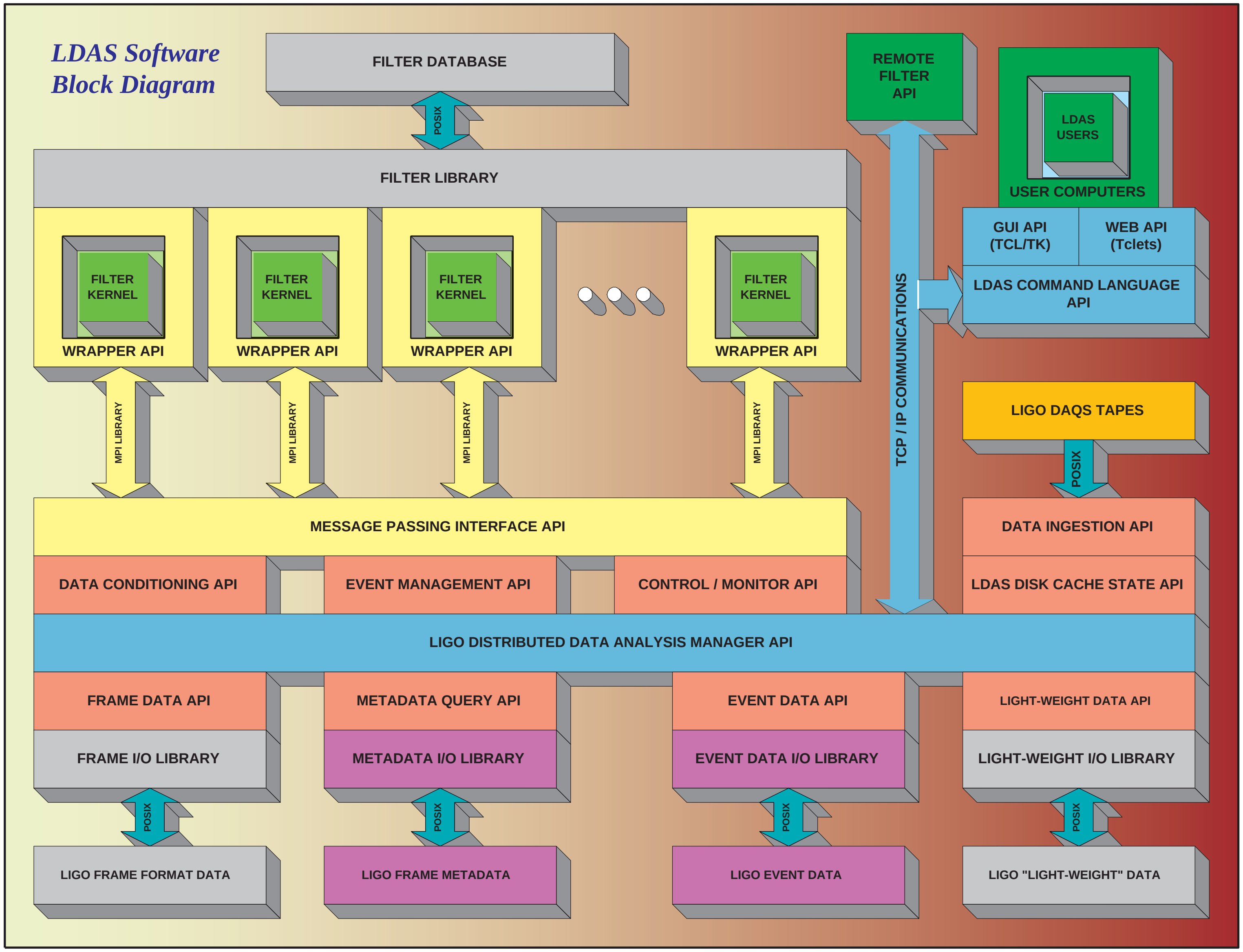
LDAS will be implemented using a combination of (i) TCL/TK as a scripting and command language for communication with users, sequencing tasks and passing commands and parameters and (ii) ANSI C++ as the object oriented language for performing computational tasks requiring the speed achievable with compiled languages.

Care in the design has been taken to ensure a fully POSIX-compliant, ANSI-standardized implementation which should ensure the ability to migrate software components among different hardware platforms and operating systems with the minimum of maintenance effort.

All software components are based on a *genericAPI* (application program interface) template. The genericAPI provides the base set of functionality in the form of an interpreted command language that exists in all LIGO Data Analysis System (LDAS) distributed computing API components:. The features of the LDAS software design may be summarized as follows (this is a top-level summary of the details provided in the various LDAS design documents identified in Section 1.6.1):

- The interpreted command language to be used is TCL/TK, which provides a command line, scripting and a graphical interface.
- The TCL/TK commands are extended to support low level system interfaces and greater computational performance using C++ code that utilizes the standard TCL C code API library in the form of a TCL/TK package.
- The genericAPI TCL/TK script accesses a genericAPI.rsc file containing needed information and resources to extend the command set of the TCL/TK language using the genericAPI package, which exists in the form of a *shared object* (genericAPI.so).
- The genericAPI will provide setup and configuration for all socket and file based communications used in all other LDAS distributed computing API components.

Figure 4 LDAS Software Design



- The genericAPI.tcl script will use a resource file named genericAPI.rsc to configure communications via TCL/TK sockets used to communicate commands and messages, and the C language-API sockets used in the packages to send data. These sockets are often referred to as the *operator* and *emergency* sockets in the TCL/TK layer of the API and as the *data* sockets in the C++ layers of the API. This resource file will also manage the loading of generic TCL/TK extensions found in the genericAPI package.
- The genericAPI.tcl script will provide a separate interpreter for each socket that is open in the “listen” state to allow thread-like response to messages and commands passed through these sockets.
- The genericAPI.tcl script will provide procedures for receiving LDAS XML tags (LDAS internal representation for lightweight data) from the underlying package layer and for sending LDAS lightweight data to the underlying package layer. Refer also to Section ***3.2.6*** on data formats.
- The genericAPI.so package will be developed in C++ using the C language interface to TCL/TK to communicate with the TCL/TK command layer. The wrappers between C++ functionality and TCL/TK command language extensions will be *machine generated* using the *SWIG* API code writer.
- The genericAPI.so package will be responsible for establishing and maintaining socket communications using a C++ class library for sockets developed by *ObjectSpace* and distributed as their *C++ Component Series Class Library.* These sockets will be used to send binary data between LDAS’s various distributed API components in the form of streamed sequential bytes and in the form of registered C++ class objects.
- The genericAPI.so package will translate registered LDAS lightweight data C++ class objects into XML documents for passing up to the TCL/TK layer, as well as translation of XML documents containing registered LDAS lightweight data tags into lightweight class objects in C++.

3.2.1 Distributed LDAS APIs

Figure 5 presents an overview of the structure of the distributed LDAS APIs.

Each LDAS distributed API is made up of two major layers.

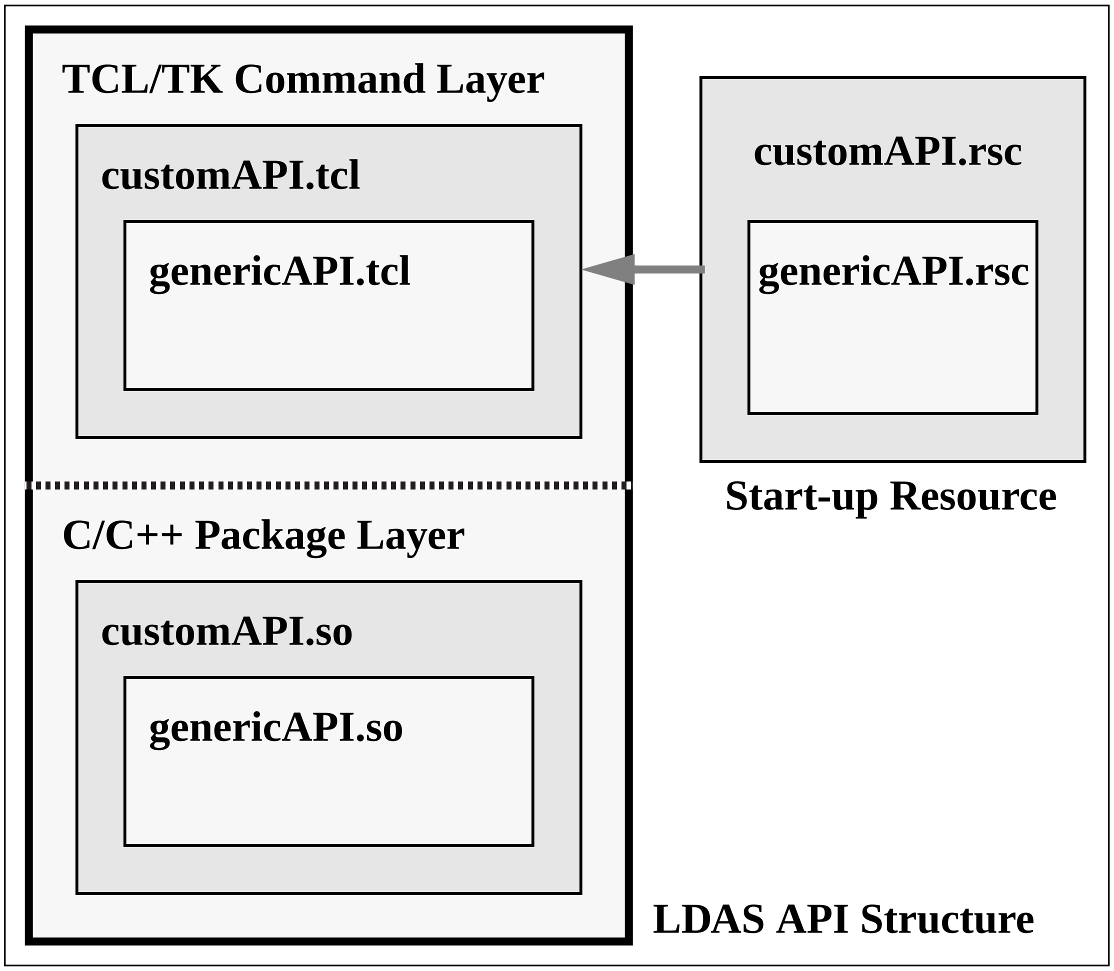
- TCL/TK Layer - this layer is the command layer and deals primarily with commands and/or messages and their attributes and/or parameters, as well as communicate with the underlying Package Layer through TCL/TK extensions.
- C/C++ Package Layer - this layer is the data engine layer and deals primarily with the binary data and the algorithms and methods needed to manipulate LIGO’s data

The TCL/TK layer consists of two internal and two external components, designed to optimize code reuse at the level of the command language used in all LDAS APIs.

- The customAPI.tcl - this TCL/TK script contains specialized TCL/TK procedures and specialized command language extensions which are particular to each custom API in the LDAS architecture.
- The genericAPI.tcl - this TCL/TK script contains the common TCL/TK procedures and command language extensions found in all LDAS APIs.

- The customAPI.rsc - this TCL/TK script contains the start-up and configuration defaults which are unique to each custom API.
- The genericAPI.rsc - this TCL/TK script contains the start-up and configuration defaults which are common to each LDAS API.

Figure 5 Nature of the Distributed LDAS API



The C/C++ package layer consists of two internal components, each developed in C++ and C to take advantage of the higher performance associated with compiled languages which is needed for the types of activities that are being carried out in this layer and loaded as shared objects.

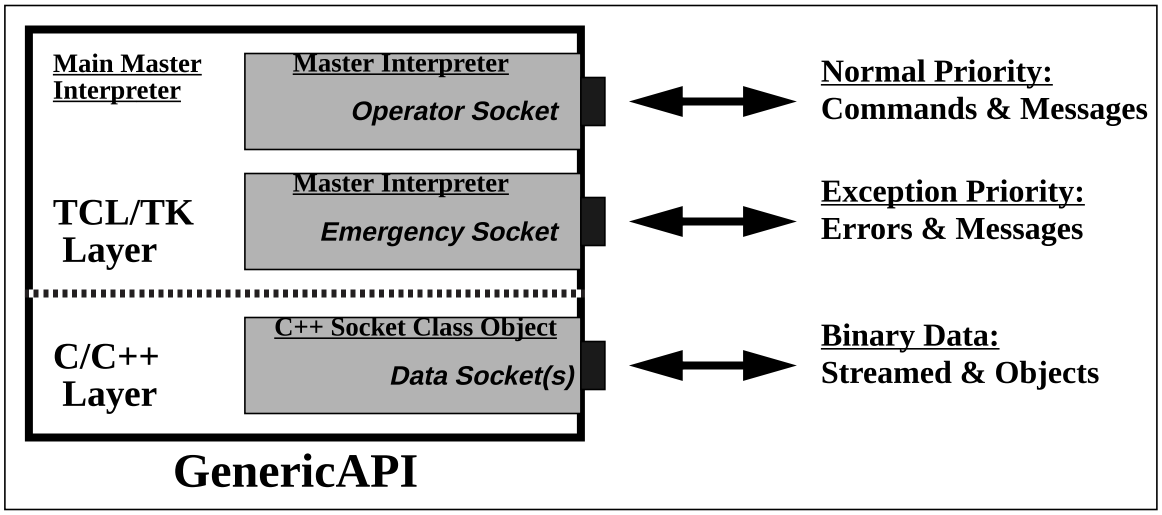
- The customAPI.so - this shared object contains the C++ classes and C interface functions needed to extend the command language set of each custom API, allowing it to more efficiently perform its duties.
- The genericAPI.so - this shared object contains the C++ classes and C interface functions needed to extend the command language set of all API's in LDAS, allowing efficiency and optimal code reuse.

3.2.2 Interprocess communications via sockets

Figure 6 depicts the implementation of socket based communications common to all LDAS APIs. The genericAPI will provide an internet socket within the TCL/TK layer that is the primary communication port for commands and messages of a normal priority. This port is commonly referred to as the *Operator Socket* to reflect its association with normal operations. Requirements on this socket are that

- it run in either a Standard or Safe Master Interpreter depending on the requirements of each specific LDAS API,
- it have the option of requiring an encrypted key attached to each incoming command for authentication prior to execution of the command as a security safeguard (*depending on the nature of each specific API*),
- it will provide a queue for incoming commands, allowing for at least 25 commands to be staged in a FIFO during peak levels of communications. If the FIFO ever fills up the manager API will be notified using the *Emergency Socket* discussed below.

Figure 6 Communications Provided by GenericAPI



The genericAPI will provide an internet socket within the TCL/TK layer that is an exception communication port for commands and messages of a highest priority. This port is commonly referred to as the *Emergency Socket* to reflect its association with critical operations. Requirements on this socket are that

- it runs in a Safe Master Interpreter supporting only a few exception handing commands for each specific API through the TCL/TK aliasing mechanism,
- it have the option of requiring an encrypted key attached to each incoming command for authentication prior to execution of the command as a security safeguard (*depending on the nature of each specific API*),
- it will provide a queue for incoming commands, allowing for no more than 5 commands to be staged in a FIFO during peak levels of communications. If the FIFO ever fills up the

manager API will be notified.

The genericAPI will provide dynamic internet sockets within the C/C++ layer that is used to communicate all data (*typically binary data*) in the form of streamed binary data or distributed C++ class objects using the ObjectSpace C++ Component Series Socket Library. This port is commonly referred to as the *Data Socket* to reflect its primary duty in communicating data sets. Requirements on this socket are that

- it be capable of managing up to 5 simultaneous socket connections at any given time by using spawned TCL/TK master interpreters (*requests for more socket communications will remain in the Operator Socket queue*),
- it be capable of streaming raw LIGO data directly,
- it be capable of streaming raw LIGO Light-Weight data directly,
- it be capable of sending and receiving instances of C++ internal LDAS light-weight data objects (*a simplified subset of the full LIGO Light-Weight data*),
- it be capable of storing the states of connected sockets to a file and restore those socket states at the local request of the API.

3.2.3 Extensions of the baseline (genericAPI) template: managerAPI

The most important specific API within LDAS is the managerAPI. This is by virtue of its interaction with all other APIs and with user interfaces. This will be described below in some detail. The design is depicted in ***Figure 7***. Similar information is available on other APIs and this is documented in various requirements and specifications documents listed in ***Table 7***.

The managerAPI is responsible for centralized administration of all LIGO Data Analysis System (LDAS) distributed computing API components using an interpreted command language.

- The interpreted command language to be used is TCL/TK, which provides a command line, scripting and graphical interface.
- The first generation(s) of the managerAPI will be developed entirely in TCL/TK. If a performance requirement is not being satisfied by the purely TCL/TK based managerAPI, then C++ code can easily be used to extend the language utilizing the standard TCL/TK C code API library in the form of TCL/TK packages.

The managerAPI TCL/TK script will be responsible for configuration and initialization of the LDAS system. This involves testing communications with all API's registered in the managerAPI.rsc resource file, initializing the state of all API's and start-up of missing API's as outlined in the resource file.

All default behavior (*e.g., conduct binary inspiral searches at sites*) will be established after initialization from its resource file managerAPI.rsc.

The managerAPI will act as the broker for user requests to the LDAS. All request from User API's will be in the form of high level language commands to the managerAPI which are queued in a FIFO queue.

<u>*The managerAPI.tcl Script's Requirements:*</u>

The managerAPI.tcl script will provide two additional communication sockets beyond the Operator and Emergency sockets provided by the underlying genericAPI, allowing interactions between LDAS API's and the manager. These sockets are referred to as the Send and the Receive Sockets in the managerAPI. As with the Emergency and Operator Sockets from the genericAPI, the Send and Receive Sockets will each have their own interpreter. The Send Socket will only connect to listening Operator Sockets on the LDAS API's and the Receive Socket will always listen for connections from the Operator Sockets on the LDAS API's.

The managerAPI will maintain three command queues used to store commands and messages between users and the LDAS API's.

- The Command Queue is used to hold incoming high level language commands from User API's. Each command in this queue is passed off to the next available assistant manager interpreter for processing. This queue will be capable of holding 25 high level commands. An attempt to communicate with the Operator Socket by a User API when this queue is full results in a *currently unavailable* message being sent via the Emergency Sockets to the User API.
- The Send Queue is used to send commands from individual assistant managers to the LDAS API's. This queue will be capable of holding 100 API specific commands before being full. If the queue is full the assistant manager requesting to use the queue is told to wait. All commands placed in this queue have an index associating them with the specific high-level command from the Command Queue and the ID of the assistant manager that pushed the command onto the Send Queue.
- Each Assistant Manager requires a completion of requested command response from the LDAS API before proceeding to the next LDAS API command instruction associated with processing high level commands. The LDAS API's report completion of commands to the Receive Socket on the managerAPI. The completion message includes the index to the high level command from the Command Queue and the ID of the assistant Manager that requested the LDAS API to perform the command. This queue will be capable of holding 100 API command completion messages. If this queue ever becomes full, the Receive Socket will not accept a new message from the API, forcing it to retry the connection when the queue is available.

The managerAPI will use Assistant Manager interpreters to supervise the execution of high level commands. There will be a minimum of three assistant managers running in the managerAPI, with the possibility of more being started if needed. Each Assistant Manager shares equally the CPU time with the managerAPI while carrying out commands. Assistant Managers will be comprised of three sub-processing kernels.

- Command Parser - This sub-process is responsible for parsing the high level command taken from the Command Queue and parsing it into a recognized procedure (TCL script) and the associated parameters that customize the procedures behavior.
- Command Scheduler - This sub-processor is responsible for integrating the managerAPI's configuration knowledge base with the output of the Command Parser to produce a script of control commands that can successfully be run based on the current LDAS configuration map. This includes the integrations of parameters from the Command Parser into the script.
- API Command Sequencer - This sub-process is responsible for carrying out each command

statement in the resulting script produced by the Command Scheduler. Each individual command is sent to the Send Queue where the ManagerAPI forward the command to the appropriate API. The API Command Sequencer then polls the Receive Queue waiting to be notified that the involved API has completed has completed the command before repeating the procedure with the next command in the script.

The managerAPI will inherit functionality from the genericAPI.tcl script for the purpose of providing logging, help, socket communications for the Operator and Emergency ports, resource management and other features found in the genericAPI.tcl script.

The managerAPI will provide a GUI which shows the current status of all sockets, queues, and system resources. This GUI will also provide a command interface which allows an operator at the ManagerAPI to issue commands to the Command Queue. It will also allow for viewing of log files and help documents using the standard set of tools provided by the genericAPI's TCL/TK script.

The managerAPI.tcl script will continue to be expanding with newer more detailed functionality as the LDAS API are developed. This means that a complete specification of the functionality will evolve as the command sets of the LDAS API's are implemented.

The managerAPI will not buffer data sets as a broker, in the Frame Format or the LIGO Light Weight Format, as it is transferred between LDAS API's. The LDAS API's will make direct socket connections between themselves as instructed to do so by the managerAPI and report back to the manager upon completion of data transfers.

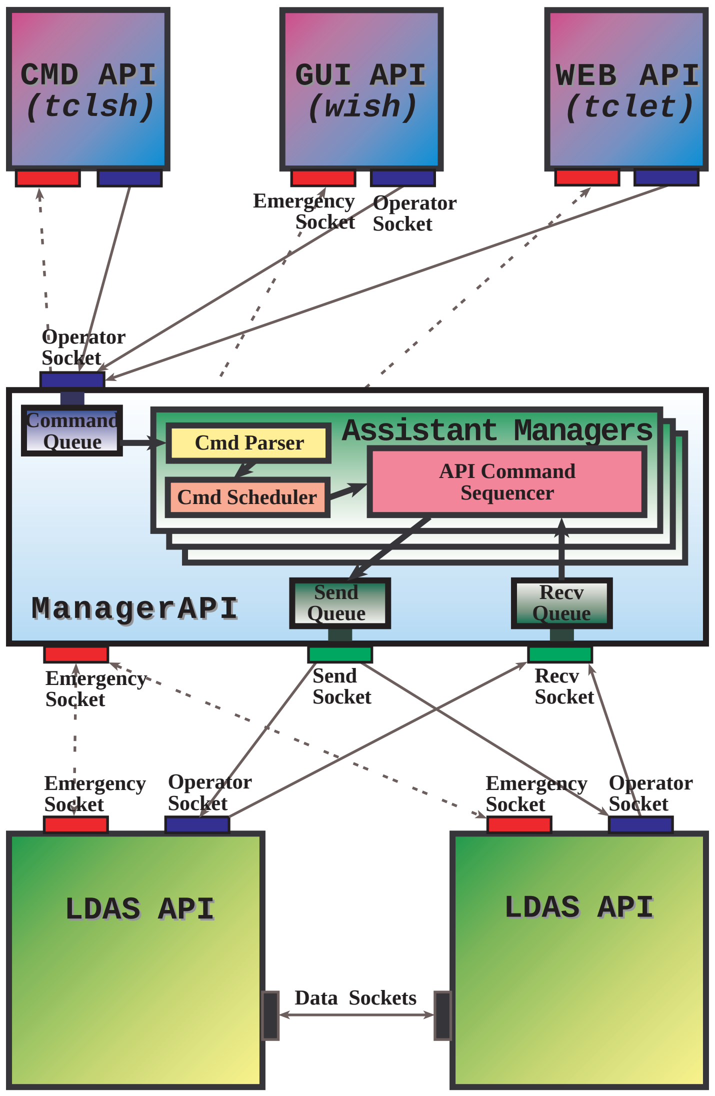
The managerAPI will maintain a list containing the performance statistics for each high level command that appears on the command queue. This list will act as a mini-database containing minimum, average, maximum execution times for each high level command, along with the number of assistant managers interpreters running for these execution times and the number of times each particular command has been executed. The times will be measured using wall clock times starting when an assistant manager first takes a command off the command queue and finishing when the assistant manager completes the command sequence for that high level command. This time will be reported to the manager which uses it to update the list information

The managerAPI.so Package Requirements:

A managerAPI.so package will not be implemented in the initial version of the managerAPI.

If a future requirement for performance in the managerAPI requires a managerAPI.so then it will be developed in C++ using the C language interface to TCL/TK to communicate with the TCL/TK command layer. The wrappers between C++ functionality and TCL/TK command language extensions will be machine generated using the SWIG API code writer.

Figure 7 LDAS Command Flow Using the ManagerAPI



Command Flow using the ManagerAPI:

Operator Socket: This socket receives incoming commands from the UserAPI's and places them on the Command Queue. Prior to placing the command in the Command Queue, the managerAPI will collect the execution statistics from the Command Statistics List and send these statistics back to the UserAPI, along with the current load as measured by the number of active Assistant Manager Interpreters. The UserAPI will then be given the opportunity to continue or cancel the command based on this summary. If the UserAPI decided to continue the high level command, then the command will be placed on the Command Queue. If a UserAPI has opened a communications socket with the Operator Socket at a time when the Command Queue is full, the Emergency Socket in the ManagerAPI reports to the Emergency Socket on the connected API that the Queue is full, interrupting commands from the UserAPI until such time that the Command Queue is again available for new commands. Commands received at this socket will have return address information attached to the command along with an optional encrypted key needed by the ManagerAPI in order to carry out the requested command. Assuming that the key matches, the command is pushed onto the Command Queue.

Command Queue: This queue receives commands from three sources, the Operator Socket discussed above, the initialization script for the ManagerAPI (*managerAPI.rsc*) and the managerAPI's Command GUI interface. Each command in the queue includes information about the source of the command (*e.g., the UserAPI, or managerAPI's Command GUI, etc.*). This information includes return IP address, port number, etc. needed to identify and report back if necessary to the originator of the command. This queue FIFO (*first in, first out*) queue and is supervised by the ManagerAPI with one exception - all commands from the managerAPI's Command GUI interface are pushed to the top of the queue for immediate execution.

Assistant Manager: The Assistant Manager processes are responsible for seeing that the high level commands that are in the Command Queue are carried out using the distributed LDAS API components (*servlets*). The ManagerAPI starts up with three such Assistant Managers (interpreters), but can add more if commands are found to reside on the Command Queue for periods longer than 2 seconds. However, the system will never support more than 10 Assistant Managers in order to minimize overhead associated with interpreter context switching. Each Assistant Manager begins by taking the next available command off the Command Queue, starting a timer to measure execution time, and sending it to the Command Parser.

- **Command Parser**: This parser separates out all return address information form the command, extracts the parameters which are used to specialize the commands behavior and identifies the ManagerAPI procedure that the command maps into. If the command is found to be unrecognizable by the parser, then an illegal command message is returned to the source of the command via the Emergency Socket (*via a dialog box in the case of commands from issued from the resource file or the GUI*). The accepted command and its parameters are passed from the Command Parser to the Command Scheduler.
- **Command Scheduler**: This scheduler takes commands (*scripted procedures*), command parameters, and LDAS configuration information (*IP numbers, port numbers, etc.*) and produces a complete, ready for execution script which has had all variable entries assigned from the parameters and configuration data. This script along with an index to the source of the high level command which was used to produce the completed script are then sent to the API Command Sequencer.
- **API Command Sequencer**: This sequencer executes each command found in the output script from the scheduler, one at a time, in order to guarantee synchronization of the LDAS system in executing the high level command which triggered the procedure. The sequencer send each command from the scheduler script one at a time to the Send Queue. The next com-

mand from the scheduler script will not be sent until the API Command Sequencer finds a command completed message associated with that scheduler command in the Receive Queue. Association is determined via an ID that determines the particular Assistant Manager, the particular high level command and the line number in the scheduler script. This ID information is sent out with the command to the Send Queue and then to the Send Socket to the LDAS API which will perform the command and is then returned with the command completion command to the Receive Socket and the Receive Queue for matching by the corresponding Assistant Manager's API Command Sequencer when reading from the Receive Queue. When the last command in the sequence has been completed, the high level command execution timer is stopped and the wall time needed to complete the high level command is reported to the managerAPI for inclusion in the statistics list.

Send Queue: This queue takes commands from each Assistant Managers' API Command Sequencer along with the unique ID used to match up command completion messages in the Receive Queue. The ManagerAPI uses this queue as a FIFO to send out LDAS API command using the Send Socket. Each command in the Send Queue is delivered to the Send Socket as soon as the Send Socket is ready to send again.

Send Socket: Commands (including the scheduler ID information discussed above) along with the optional encryption key are sent out this socket to the appropriate LDAS API for execution.

Receive Socket: Once an LDAS API has completed the requested command delivered to the API form the managerAPI's Send Socket, it will send back a message stating such to the Receive Socket on the managerAPI with also contains the unique ID associated with the Assistant Manager which placed the requested command in the Send Queue. As these messages are received at the Receive Socket, they are placed on the Received Queue.

Receive Queue: This queue holds all messages coming back from LDAS API's upon completion of their requested commands from the Assistant Managers. All of the Assistant Managers monitor this queue waiting for the unique message reporting that the last sent command has been completed, thereby allowing the Assistant Managers to issue the next command from their API Command Sequencers.

Emergency Sockets: These sockets are used to report exceptions in the execution of commands by the LDAS API's. The Assistant Managers will monitor the Emergency Socket Queue looking for messages that one of their issued commands was unsuccessful, causing the scheduled script to abort and an exception message to be sent to the Emergency Socket on the UserAPI which issued the original high level command that was used to generate the command script that failed.

3.2.4 Software Development Tools

TCL/TK:

TCL is a string based command (or "steering") language. The language has only a few fundamental constructs and relatively little syntax making it easy to learn. TCL is designed to be the glue that assembles software building blocks into applications. It is an interpreted language, but provides run-time tokenization of commands to achieve near to compiled performance in some cases. TK is an TCL integrated (as of release 8.x) tool-kit for building graphical user interfaces. Using the TCL command interface to TK, it is quick and easy to build powerful user interfaces which are portable between Unix, Windows and Macintosh computers. As of release 8.x of TCL/TK, the language has native support for binary data.

C and C++:

The C and C++ languages are ANSI standard compiled languages. C has been in use since 1972 and has become one of the most popular and powerful compiled languages in use today. C++ is an object oriented super-set of C which only just became an ANSI/ISO standard in November of 1997. It provided facilities for greater code reuse, software reliability and maintainability than is possible in traditional procedural languages like C and FORTRAN. LIGO's data analysis software development will be dominated by C++ source code.

SWIG:

SWIG is a utility to automate the process of building wrappers to C and C++ declarations found in C and C++ source files or a special *interface file* for API's to such languages as TCL, PERL, PYTHON and GUIDE. LDAS will use the TCL interface wrappers to the TCL extension API's.

Make:

Make is a standard Unix utility for customizing the build process for executables, objects, shared objects, libraries, etc. in an efficient manor which detects the files that have changed and only rebuilds components that depend on the changed files.If/when LDAS software becomes architecturally dependent, it will be necessary to supplement make with auto-configuration scripts.

CVS:

CVS is the Concurrent Version System. It is based on the public domain (and is public domain itself) software version management utility RCS. CVS is based on the concept of a software source code repository from which multiple software developers can check in and out components of a software from any point in the development history.

Documentation:

DOC++ is a documentation system for C/C++ and Java. It generates LaTeX or HTML documents, providing for sophisticated on-line browsing. The documents are extracted directly from the source code files. Documents are hierarchical and structured with formatting and references.

TclDOC is a documentation system for TCL/TK. It generates structured HTML documents directly from the source code, providing for a similar on-line browsing system to the LDAS help files. Documents include a hyper-text linked table of contents and a hierarchical structured format.

3.2.5 Database Management System for LDAS Metadata

In October 1998 LIGO held a small workshop with invited speakers representing a number of scientific research groups who use databases. After 1 1/2 days of discussion, LIGO determined that its database requirements could be satisfied for the foreseeable future by using relational database technology. The reasons for this is that LIGO does not intend to archive its [raw] framed database in a database management system. At present, scientific uses of object oriented databases [i.e., Objectivity] have been in applications where very large amounts of [intrinsically object-oriented] data, such as astronomical images or high-energy physics events are archived. Object oriented databases allow encapsulation of query methods directly with the data objects. However, this is achieved at the price that researchers who wish to ingest new data into the database must be proficient at programming in C++.

LIGO also performed a brief trade between the adoption of public domain database management tools versus commercially licensed products. Public domain solutions which were considered included miniSQL and postgresSQL. Commercial products which were considered included Sybase, Informix, DB2 and Oracle. After consideration of the advantages and disadvantages of each approach, it was determined that intrinsic database size limitations and the need to develop extensive customized interfaces into public domain products made these options less attractive than adoption of commercially available DBMS products. Caltech’s existing licensing agreements with both IBM(DB2) and Oracle made these two products the most attractive ones. At present LIGO is prototyping a metadatabase for existing 40m datasets in DB2. The decision has been made, however, to produce all LIGO-specific APIs and codes for database queries [SQL] in a manner consistent with the ODBC standard. In this manner, it will be possible to migrate the LIGO-specific developmental work to other DBMSs with minimal code modifications.

Latest versions of relational databases allow incorporation of object oriented data to some degree by supporting BLOBs [binary large objects] within table structures. These will be utilized where absolutely needed (such as allowing spectra, images, or other intrinsically object-oriented data types) within LDAS; however there is a price for encapsulating data objects as BLOBs. It is that the LDAS-specific code which interfaces to a specific DBMS will be less transportable to other DBMS products. There will need to be some code modification to reflect differences in BLOB support between products.

3.2.6 Data Types and Databases

3.2.6.1 Frames

Data are acquired using a frame format standardized jointly with the VIRGO project. The frame specification is defined in LIGO-T971030. The CDS/DAQS is capable of generating data simultaneously in several different frame types. The bulk (raw) data will be acquired in a so-called full frame. A more streamlined frame type, the analysis frame, will be dedicated to the GW channel and a few critical auxiliary channels which serve to qualify data quality or validity. In addition, a trend frame will be acquired that provides summary statistics, such as max/min/rms/avg/etc. for the channel data. The summary statistics are sampled at much reduced bandwidth (e.g., 1 Hz or 0.1 Hz) and can be used to perform initial data QA scans by assessing such top-level performance metrics.

3.2.6.2 Metadata

LIGO metadata will be written into the DBMS (presently DB2 is being evaluated) using a LIGO-develop API (see LIGO-T980119). The metadata will include tables for the following types of data:

1. Frame description tables -- includes frame names, locations, descriptive high-level statistics to indicate data quality (first five moments of the distribution; max/min), spectra for selected channels as BLOBs, and other TBD descriptors.
2. LDAS event tables -- includes descriptions of events generated by numerical filters within LDAS, including astrophysical paramterizations, statistical confidences, event generator, etc. The event data format is still TBD.
3. CDS/GDS trigger tables -- similar in design to 2. These are generated by the on-line diagnostics suite of software.
4. Electronic log tables -- includes references to graphics, operator log entries and other electronically archived information about detector status which is created by CDS and which needs to be archived and linked to the frame database to allow interpretation of the results of analyses.
5. Detector system state vector tables -- includes references to CDS-created databases needed to interpret the frame data.
6. LDAS log file tables -- if required, it will also be possible to incorporate a subset of the LDAS managerAPI log database. This database records LDAS states, resources, exceptions, etc.

3.2.6.3 Lightweight Data

A lightweight data format has been defined to allow efficient and easily used manipulation of (small) datasets.

A machine-internal representation has been defined for the socket-level data transmission and communication among LDAS C++ API software. See the LIGO genericAPI specification, LIGO-T980094 for this definition.

An external representation for writing datasets to storage is in a draft form and is being reviewed. This external representation is based on XML. The specification is presented in LIGO-T980091.

3.2.6.4 Events

Events will consist of diagnostics-derived triggers and astrophysical template search results. An XML (lightweight) format for events will be defined. In addition the LIGO-developed API for accessing the DBMS will be able to take events and ingest them into the datable or retrieve them as lightweight data objects.

3.3 LDAS Software Configuration

3.3.1 LDAS Software Development Resources

The LDAS system will need centralized access to software resources such as compilers, interpreters, editors, etc., which are part of any software development environment. These same resources will be centrally located on the LDAS “Control and Monitor Server”. They will be exported only to the workstations shown in the LDAS Hardware Architecture Diagrams (both the on-line and off-line configurations). The exact content of the centralized software development resources will be a mirror of the LDCG development server located in the /ldcg partition on the server “SPICA”. The exported partition will contain platform specific subdirectories for all platforms in used by the LDAS system. The one possible exception to this centralization will be the commercial database server, which because of licensing constraints my have to be maintained locally to the database server hardware as has proven to be the case with IBM’s DB2 database product.

3.3.2 LDAS Runtime Resources

The LDAS system will need to centralize access to all the LDAS executables, shared objects, libraries, scripts, logging files, help files and TCL/TK interpreters. These runtime software resources will be located on the same LDAS “Control and Monitor Server” as mentioned above. It will export these files to all hardware components found in the LDAS system, including workstations. The mount points for these resources on all LDAS hardware will be based on NFS and use identical directory naming conventions allowing standardized paths for environment variables.

3.4 LDAS Startup Procedures

3.4.1 Initialization

The LDAS initialization upon original startup is controlled by the managerAPI and its associated resource file. Upon startup, the managerAPI runs the script found in the managerAPI.rsc file. This script will cause the managerAPI to ping all LDAS API using the specified host/port identifications for each API found in the resource file. If the API is not “alive”, the managerAPI will attempt to start the API on the designated host using a LDAS customized “remote shell” command. If the remote shell command is unsuccessful on the designated host, the managerAPI will read from the resource file an alternate host for the API and then perform the remote shell command to start the API on the alternate host. All steps will be logged allowing tracebacks to the system configuration and in the event that the alternate host is used a warning dialog box will prompt the operator that the nominal configuration has been modified to use the alternate configuration. In the event that the alternate host is also unable to sustain the API, the managerAPI will report an error to the operator, forcing manual system administration activities to take place.

3.4.2 Abnormal Runtime Condition Management

After the LDAS system has been initialized successfully by the managerAPI, the system begins processing each high level command that appears in the command queue of the managerAPI. Each of these high level commands is passed off to an individual “assistant manager” process within the managerAPI’s TCL script. The assistant managers turn the high level commands into complex TCL scripts involving customized command extensions found in each LDAS API resource. The assistant managers send each command in the resulting script out to the LDAS APIs, one at a time, waiting for a reply from the API before issuing the next command. Replies can come in one of two ways from the resource APIs. If the command was successful, the managerAPI will receive notification at its “receive socket” that the API completed the requested command. If the API recognizes that the command was unsuccessful, as in the case that the command resulted in an exception being thrown in the C++ or TCL layers of the API, then the emergency socket of the managerAPI will be notified that the command behaved abnormally resulting in the managerAPI logging the exception and reporting to the origin of the command (typically an operator or user) that the high level command request could not be performed for the reported reason. If the API behaves abnormally and does not through an exception due to a software bug, power outage, etc., then this to will be detected in the managerAPI by the associated assistant manager. This is done by a timeout clause with a default time set by the managerAPI's resource file, (typically 30 seconds for any command issued to an API). However, the individual scripts associated with each high level command being handled by the assistant managers can override the timeout value on a per command bases allowing very computationally intensive API commands to have a more appropriate timeout value.

In the event that a timeout occurs, the assistant manager that detects the timeout conditions will force the managerAPI to ping each LDAS API again in order to detect if a resource has disappeared. The managerAPI will, in the event of a non-responsive API(s), restart the API(s) and the assistant manager will then begin executing the high level command script all over. If a timeout occurs a second time for the high level command script, the operator or user is notified that the command can not be properly completed and system administration steps in to determine if a high level command request is syntactically in incorrect, while the managerAPI continues to process other high level commands using other assistant managers.

3.5 Interfaces

3.5.1 Connections to CDS and Observatory General Computing networks

The interfaces to the LDAS systems at the observatories are shown in ***Figure 8*** and ***Figure 9***.

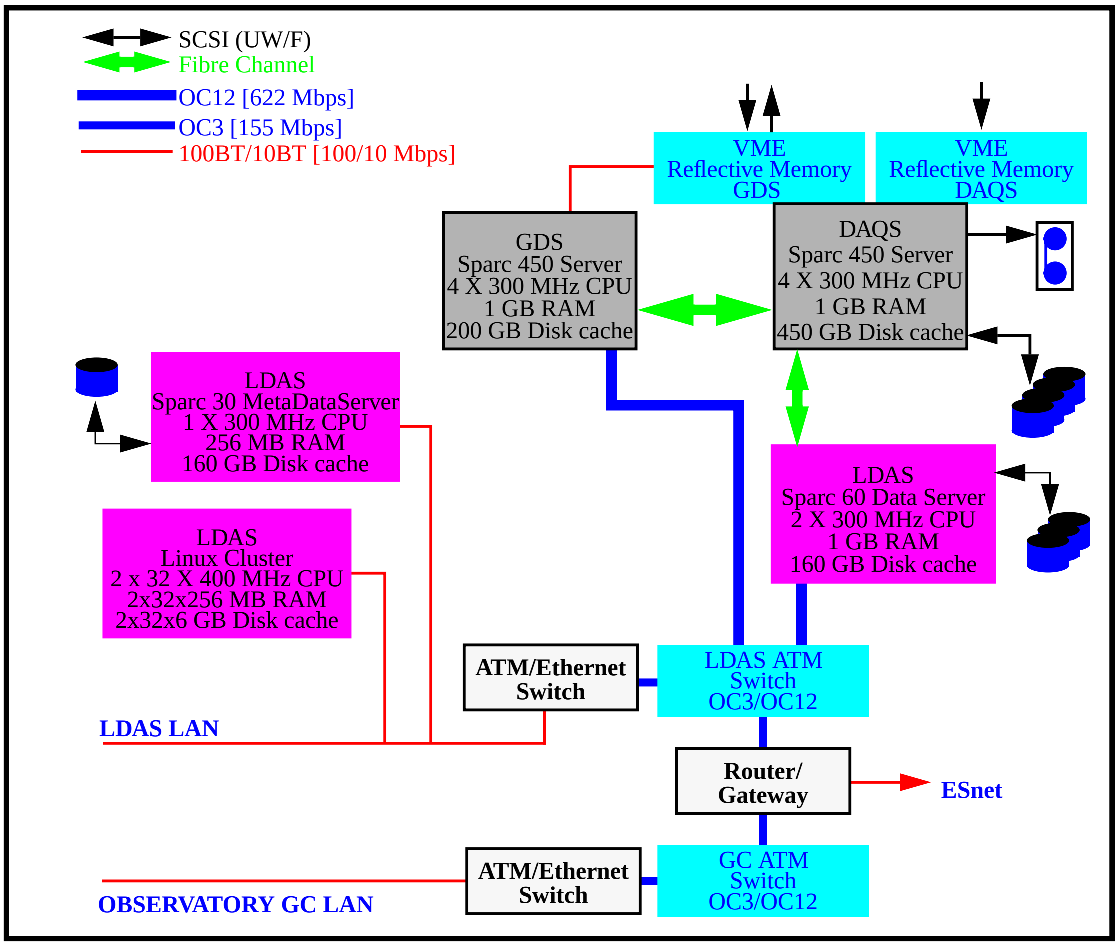

SCSI (UW/F)
Fibre Channel
OC12 [622 Mbps]
OC3 [155 Mbps]
100BT/10BT [100/10 Mbps]
VME Reflective Memory GDS
VME Reflective Memory DAQS
GDS Sparc 450 Server 4 X 300 MHz CPU 1 GB RAM 200 GB Disk cache
DAQS Sparc 450 Server 4 X 300 MHz CPU 1 GB RAM 450 GB Disk cache
LDAS Sparc 30 MetaDataServer 1 X 300 MHz CPU 256 MB RAM 160 GB Disk cache
LDAS Linux Cluster 2 x 32 X 400 MHz CPU 2x32x256 MB RAM 2x32x6 GB Disk cache
LDAS Sparc 60 Data Server 2 X 300 MHz CPU 1 GB RAM 160 GB Disk cache
ATM/Ethernet Switch
LDAS ATM Switch OC3/OC12
LDAS LAN
Router/ Gateway
ESnet
ATM/Ethernet Switch
GC ATM Switch OC3/OC12
OBSERVATORY GC LAN

Figure 8 LDAS-CDS Interface at the observatories

The data distribution server is connected by a high speed connection (fibrechannel or ATM) to the DAQS framebuilder. This provides high bandwidth access to the on-line disk cache, which is managed by the DAQS server. The LDAS server buffers the DAQS server and CDS network from multiple user requests from non-CDS (i.e., non-operations) users requesting on-line data. The LDAS LAN and dedicated ATM switch provide access to the on-line database for the real-time detection algorithms running on the linux cluster. Since the LDAS LAN is private, there will be a gateway to the internet and to the general computing (GC) LAN at the observatory. Casual users of on-line data will be served via the internet and the GC LAN. Intense (local) data analysis tasks will be performed on the LDAS LAN.

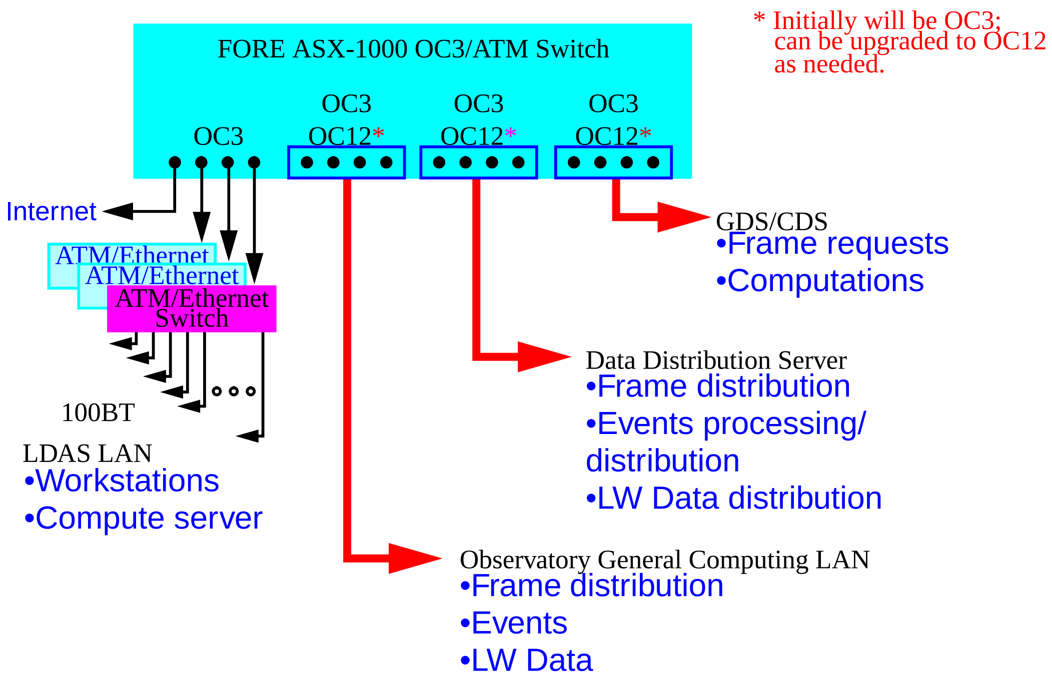


Figure 9 LDAS Networking Interfaces at the Observatories

3.5.2 Interfaces to CACR

The network interfaces between LDAS and CACR computing resources at Caltech are shown in ***Figure 10***. The high speed connection between LDAS and CACR will be from the main CACR ATM switch to the LIGO ATM switch. The LIGO data archive will be physically located at CACR and will be accessible via the OC-12 link to CACR from LDAS. CACR computational resources will be accessed over the CACR LAN. Data from the LIGO archive may be transmitted over the switch-to-switch connection to other LDAS facilities, such as data server, data conditioning workstations and the LDAS linux PC cluster.

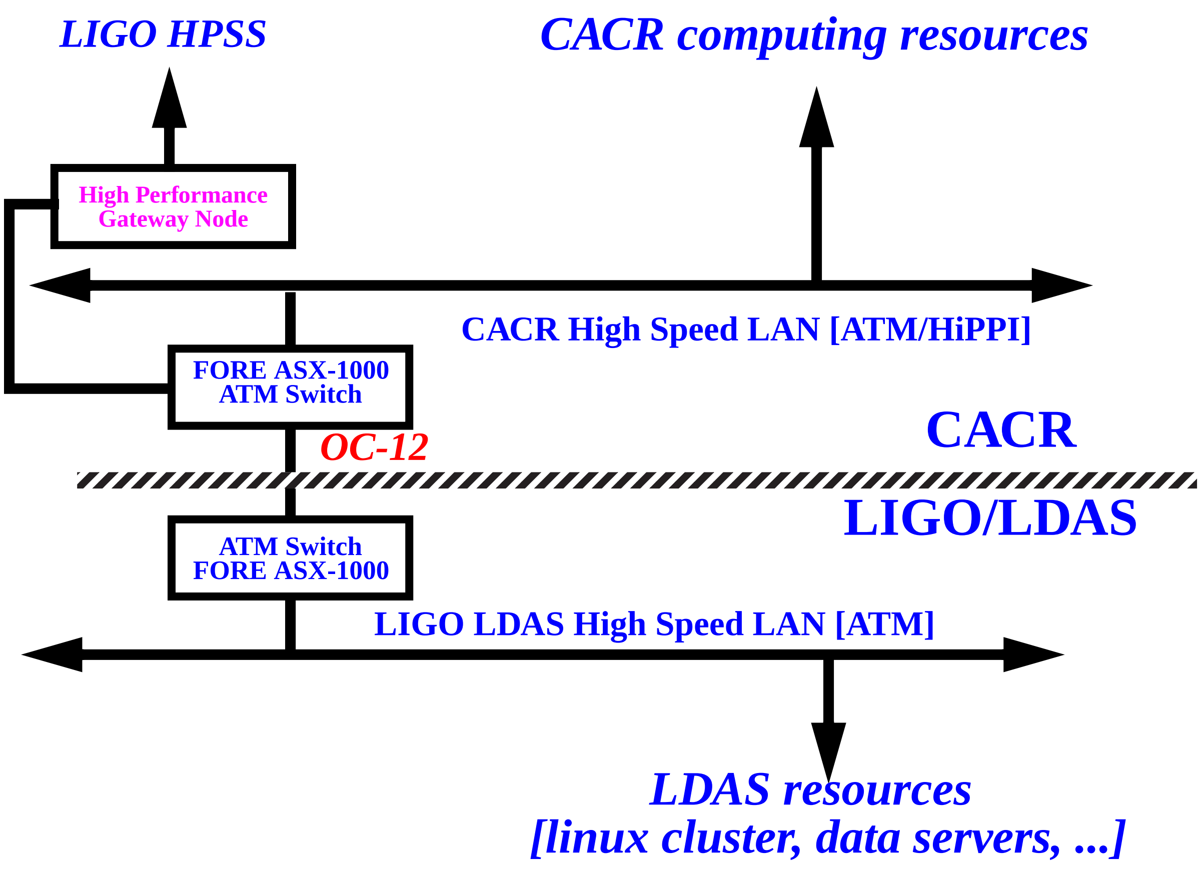

LIGO HPSS
CACR computing resources
High Performance Gateway Node
CACR High Speed LAN [ATM/HiPPI]
FORE ASX-1000 ATM Switch
OC-12
CACR
LIGO/LDAS
ATM Switch FORE ASX-1000
LIGO LDAS High Speed LAN [ATM]
LDAS resources [linux cluster, data servers, ...]

Figure 10 LDAS Networking Interfaces to CACR at Caltech

3.5.3 User interfaces to LDAS

At present the exact details for the user APIs which provide the interfaces for users into LDAS are still in the definition process. However, the following description serves to provide an idea how this is supported. When the requirements and specifications are available, they will be reviewed separately.

There will be a “LIGO Command Language” (or LCL) that is based on Tcl scripts and that is understood by the managerAPI. This LCL will consist of a number of high-level commands that imply a sequence of lower-level operations. These commands can themselves be concatenated into more complex sequences. This follows the paradigm of a standard UNIX shell.

Examples of such commands include: “re-sample channel X at rate Y”; “regress channels {Y,Z,...} from channel X, pass resultant to socket S or process P”; “calibrate channel X (using the calibration data valid at the time channel X was acquired and which are available in the framed data)”; create a subframe for channels {X,Y,Z,...} from time interval {T1,T2} and re-sampled at rate R”; etc.

In addition to the ability to enter such scripts directly into a command line interface or shell, a number of "point and click" GUI screens will be developed that create the same scripts in the background as a consequence of user selections with a mouse.

Users may also want to develop personal scripts as ASCII files which may then be "sourced" by the command line interface.

The dictionary of these commands will be extensible and designed to grow with time as certain sequences are identified as being particularly common and useful. For each LCL-level command, there corresponds a database in the managerAPI which provides the detailed sequences of managerAPI commands that are needed to execute the high-level command. These details will include tasks such as querying the DBMS metadata for frame locations/frameIDs which correspond to particular epochs; extracting calibration data from frames and applying them to specific channels; requesting data conditioning services from the datadonditioningAPI; querying the diskcacheAPI to determine if the needed data are already available on (locally available) spinning media; etc.

3.6 Wide Area Network

LIGO has setup a wide area network, initially capable of sustained T1 data rates and eventually capable of accommodating full vBNS bandwidth for those sites directly connected to the internet and up to 4xT1 at Hanford, where the connection to the internet is made via special arrangement with the DOE's ESnet hub at PNNL in Richland, WA. The connectivity and communications bandwidths for the WAN are shown in ***Table 8***

Table 8 WAN/LAN Connectivity among LIGO Laboratory Sites

Site	***Livingston, LA***	***Hanford, WA***	***MIT***	***Caltech***
Caltech	vBNS/OC3	ESnet (4 X T1) <-> vBNS/OC3	vBNS/ OC3	OC3/ ATM 100BT
MIT	vBNS/OC3	ESnet (4 X T1) <-> vBNS/OC3	100BT OC3(?)	
Hanford, WA	ESnet (4 X T1) <-> vBNS/OC3	OC3 100BT		
Livingston, LA	OC3 100BT			

4 LDAS CONFIGURATION

The design of the LDAS software environment is highly modular and flexible. Many API components can provide identical functionality when installed on different hardware configurations. Similarly, different hardware components can provide multiple functionalities. The discussion below represents one implementation and serves as a basis for presenting the LDAS design. This implementation is depicted in ***Figure 11*** and ***Figure 12***. What is actually implemented within LDAS will evolve from early prototypes to fully commissioned components. The final configuration may differ in some detail from what is presented below.

As a minimum, LDAS will be required where there are databases which need to be accessed by a large number of individuals. LDAS will be installed at both observatories and at the off-line data repository at Caltech. Other LIGO Laboratory sites (e.g., MIT and Caltech campus) are not presently foreseen needing separate installations.

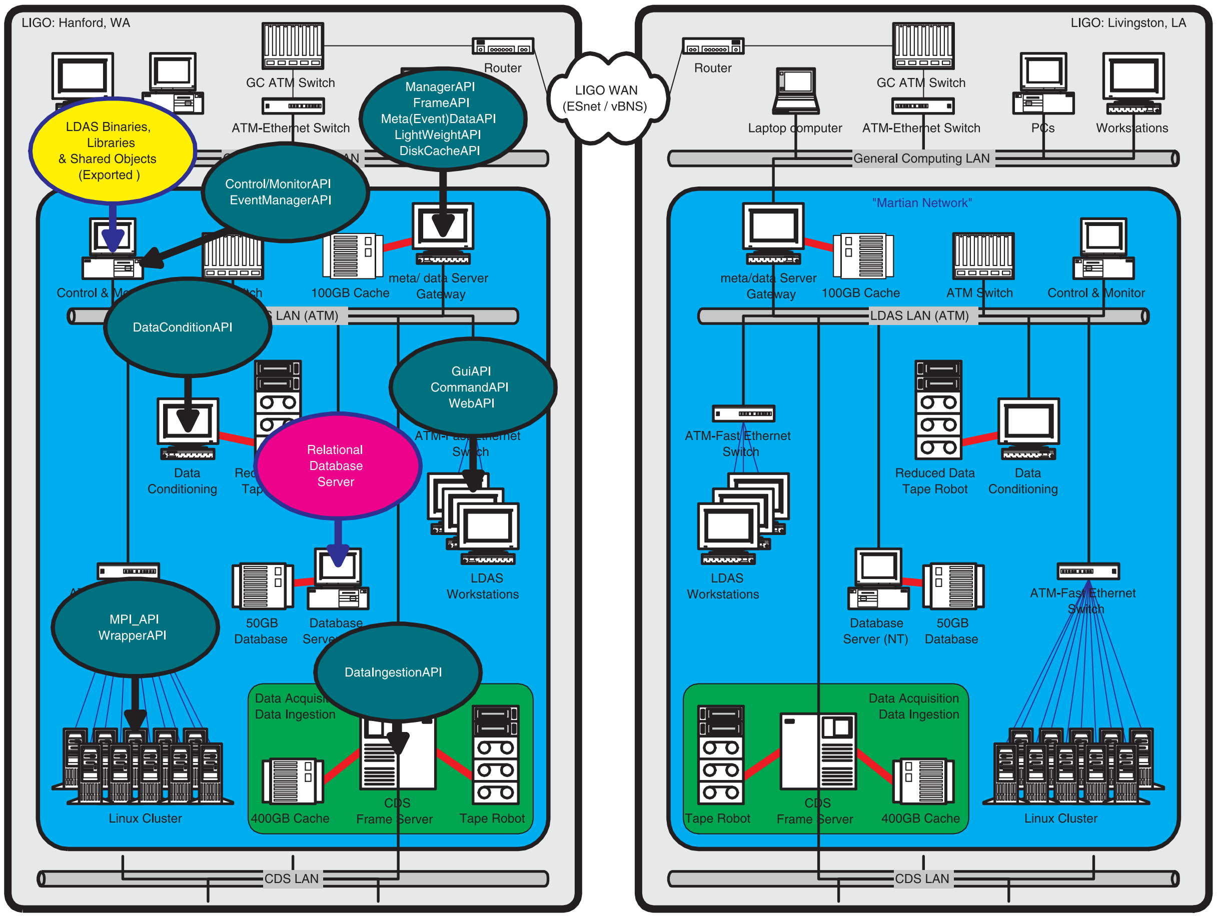


Figure 11 Mapping of LDAS APIs onto the on-line hardware.

In principle, LDAS components could also be installed at LSC member institutions; however issues of software maintenance and support outside LIGO Laboratory will need to be addressed and scope for this support identified and supported within the Laboratory.

Each installation of LDAS can have only one instance running of the ManagerAPI with its various system resource files defining the local LDAS configuration. This API will be installed on a server which serves as a gateway between the (private) internal LDAS network and other public networks. Other APIs are not limited in the number of instances; however it is likely that, depending on the hardware configuration, it will be more efficient for one API to handle multiple users rather than allocating one instance of an API to individual users.

Moreover, there is not a unique one-to-one mapping between hardware components and corresponding software (API) components. LDAS will be installed in a manner designed to optimize the hardware resources at the installation site.

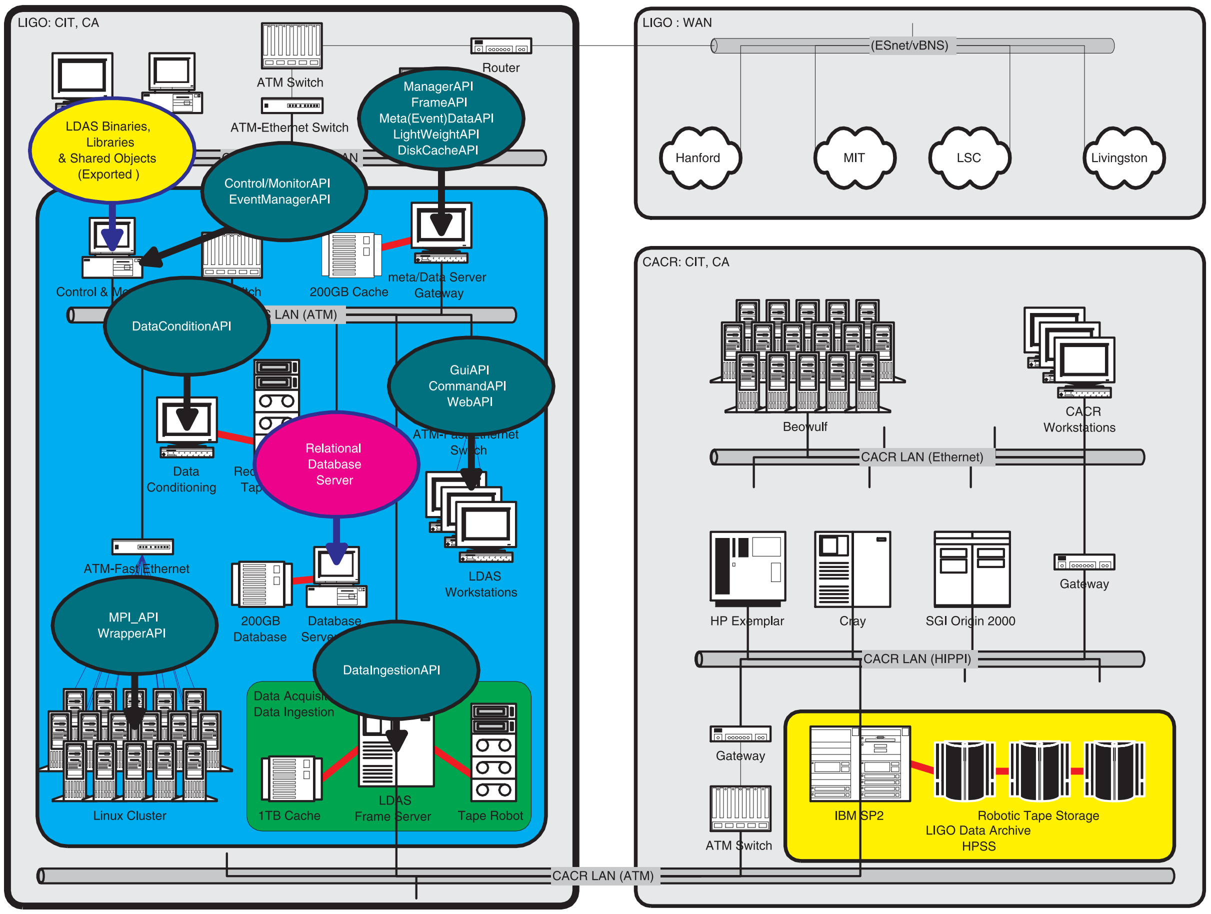


Figure 12 Mapping of the LDAS APIs onto the off-line hardware. The HPSS will have its own software system which is not developed by LIGO.

4.1 Metadata/Data Server:

FUNCTIONS:

- Provides access to frame data and metadata. This high-performance unix server that is connected via fibrechannel (or possibly OC3/OC12 through the LDAS ATM switch) to the to the CDS framebuilder.
- Performs any necessary data decimation and channel selection from the raw frame data.
- Users subscribe to distribution services on this server. They can request data in the form of frames or lightweight data sets. Note that the Disk Cache is part of CDS/DAQS.
- ON-LINE SERVER:
  Server is a gateway between the LDAS LAN (private) and the General Computing LANs at LHO, LLO and other LIGO Laboratory sites.
- OFF-LINE SERVER:

Server is a gateway between the LDAS LAN (private) and the local (CACR) LAN.

- Provides access to clients for LIGO metadata created by GDS, CDS/DAQS and LDAS at the observatory.

CPU:

The CPU will be a UNIX SUN Ultra 60 class server with 2X450 MHz *{CHECK}*CPUs and 1 GB RAM. The machine is configured as a SUN Solaris server. The machine is capable of serving frame data to multiple users at an aggregate bandwidth of 20 MB/s over OC3/OC12 ATM.

DISK:

ON-LINE:
The server will be configured with a local disk cache of ~100GB, independent of the on-line disk farm. This allows data staging from the on-line cache if required to provide sustained throughput.

OFF-LINE:
The server will be configured with a local disk cache of ~100GB, independent of the off-line disk cache. This allows data staging from the off-line cache if required to provide sustained throughput.

NETWORK CONNECTIONS:

ON-LINE:
The server is connected directly to the DAQS framebuilder by fibrechannel capable of providing sustained throughputs of frame data at 1.2 Gb/s. If ATM is used rather than fibrechannel, then the sustained bandwidth will be 622 Mb/s.

OFF-LINE:
The server is connected directly to the HPSS archive server by fibrechannel capable of providing sustained throughputs of frame data at 1.2 Gb/s. If ATM is used rather than fibrechannel, then the sustained bandwidth will be 622 Mb/s.

SOFTWARE:

The managerAPI serves to accept data requests from users, to sequence the scripting commands required to provide service. It is also responsible for system startup and logging.

The frameAPI serves to provide an interface to the raw framed data archived on spinning media or in a mass storage archival system. If frame data objects are requested as part of the data distribution service, this API creates

The metadataAPI serves to provide ingestion and retrieval function for metadata created as part of the data archival process.

The lightweightDataAPI serves to provide access to data stored in this format and to create LigoLW data objects as products of data requests to the data distribution server.

The diskCacheAPI manages data retrieved from the frame archive and stored to local (to the data distribution server) disk space. This may be used to improve data access to frame data which are repeatedly requested by clients. Also, it is expected that some client requests will involve creation of concatenated frames or lightweight data objects of considerable length (e.g., hour-long or even multi-day time series for specific channels). Scratch disk space will be required for the data distribution server to build these data files.

LIBRARIES:

Frame data library for frame data I/O.

Lightweight data library for LigoLW I/O.

POSIX library.

Algorithm library.

4.2 Relational Database Server:

FUNCTIONS:

- Provides access to the metadatabase.
- Requests in ODBC standardized SQL are submitted by the data server.
- Server is on the private LDAS LAN.
- Metadata Database Management System (DBMS) server.

CPU:

The CPU will be an NT machine with 512 MB RAM. The machine is configured as an NT server. The machine is capable of serving metadata to multiple clients at an aggregate bandwidth of 12 MB/s over 100BT.

DISK:

The server will be configured with a local disk cache of 50GB, independent of the on- or off-line disk cache. This allows data staging from the on- or off-line cache if required to provide sustained throughput.

NETWORK CONNECTIONS:

The server is connected directly to the LDAS LAN at 100BT.

SOFTWARE:

Commercial database management system (DBMS) server software.

LIBRARIES:

DBMS library.

POSIX library.

4.3 Data Conditioning & Regression Processor:

FUNCTIONS:

- One or more workstations or dedicated hardware units (e.g., DSPs) that provide a number of required functions to preprocess raw data for analysis. This includes calibration; regression; decimation and resampling; Fourier transformation; formatting for MPI processing; etc.

CPU: This will be 1 - 2 PC linux machines with 512 MB per CPU and 450 MHz CPUs. If the computational requirements dictate, in principle, this function could be migrated into the linux compute cluster by allocating a group of dedicated nodes to this service.

DISK: UWSCSI ~23 GB per CPU.

TAPES: Tape robot capable of handling 5 - 10 cassettes. Exabyte 5 - 10 GB/cassette. These tapes support user requests for specific datasets. In addition, reduced data for long-term archival may be produced.

NETWORK CONNECTIONS: Peripherals will be connected via UWSCSI to the CPUs. This machine will be on the LDAS LAN with 100BT ethernet.

SOFTWARE:

The dataConditioningAPI provides filter algorithms for a specific subset of signal processing methods which will be involved in conditioning and preprocessing time series data for subsequent pipeline processing or distribution. Such filtering methods as decimation, band-limited filtering, calibration, multiple-channel regression and cross-spectral estimation will be supported; trend data summaries can be produced (e.g., rms vs. time; max/min/avg, data QA {1, 0}). The software library providing these functions will be common with CDS/GDS and other LDAS processes.

The frameAPI provides the same functionality as described for the data distribution server.

LIBRARIES:

Frame data library for frame data I/O.

Lightweight data library for LigoLW I/O.

POSIX library.

Numerical filter library.

4.4 LDAS Control & Monitoring Server:

FUNCTIONS:

- Configures, starts and synchronizes operation of LDAS parallel computing resources.

CPU:

Linux PC with 256MB RAM, UWSCSI interface to disk.

DISK:

20 GB, UWSCSI interface.

NETWORK CONNECTIONS:

100BT ethernet via the LDAS ATM LAN.

SOFTWARE:

The eventManagerAPI logs events which are generated by on-line detection algorithms. If hierarchical search algorithms are implemented, then the scheduling of refined or iterated searches will be performed by this API. The API provides access into the event metadatabase. Since events are metadata, they will also be registered with the metadatabase server.

The eventDataAPI accepts results from the on-line detection processes and creates LIGO event objects for ingestion into the metadatabase.

The controlMonitorAPI provides statistics, reports, and performance overview of the overall LDAS system performance. Faults, recoveries and start-ups are managed by this software.

LIBRARIES:

Event library for event data I/O.

Lightweight data library for LigoLW I/O.

POSIX library.

4.5 Compute Server:

FUNCTIONS:

- Provides MPI-based multi-processor parallel computation for a number of analysis processes, including signal processing and detection algorithms.
- Manages the template database needed for optimal filtering.
- Generates events for further processing.

CPU:

Multiple nodes connected via a flat high-speed switch network dedicated to inter-node communications. These will be PCs operating under linux. The number of nodes is extensible. Present estimates indicate ~32 nodes per interferometer will be required to provide on-line detection algorithm processing at rates commensurate with the acquisition of data. Each node will be configured with local disk space and 128MB of RAM for numerical processes. The algorithms will be parallelized using MPI.

DISK: Individual nodes will have 2 - 4GB disks.

NETWORK CONNECTIONS: Internode connectivity 100BT or faster on a dedicated switch. External gateway provided by master node, also ethernet 100BT for the LDAS LAN.

SOFTWARE:

The messagePassingAPI will interface the LDAS environment with MPI for the parallelized computation.

The wrapperAPI will provide a C++ interface into numerical algorithms written in other compiled languages (e.g., C, FORTRAN90).

LIBRARIES:

MPI library.

Lightweight data library for LigoLW I/O.

Numerical filter library.

POSIX library.

4.6 Data Ingestion:

Originally, data ingestion was expected to be a process involving the off-line system only. however, the need to be able to read back in data from media that are stored at the observatories has been identified. This function will be performed by the ingestion API.

FUNCTIONS:

- Provides the means to ingest media written at observatories containing raw framed and metadata for preprocessing and conditioning as part of the data archival process.
- Provides the means to generate reduced datasets from archived data.

CPU:

PC running under linux OS. 450 MHz CPU, with 128MB RAM; UWSCSI interface for peripherals.

DISK: 2 - 4 GB disk for software.

TAPE: Tape drive system identical with reading mechanisms installed at the observatories. Tape handling capability to accommodate ~ 4 - 8 hours of data media (from both observatories and all interferometers) without requiring human intervention.

NETWORK CONNECTIONS:

ON-LINE:
UWSCSI interface for peripherals. LDAS LAN and General Computing LAN connection will be via OC3 or OC12 (at some future time) on ATM running 100BT or 1000BT ethernet.

OFF-LINE:
UWSCSI interface for peripherals. LDAS LAN and CACR LAN connection will be via OC3 or OC12 (at some future time) on ATM running 100BT or 1000BT ethernet.

SOFTWARE:

The dataIngestionAPI will provide for the manipulation of tape media and transfer of framed data from tape to disk cache.

The frameAPI will provide access to frame reading and writing routines in the frame library.

LIBRARIES:

Frame data library for frame data I/O.

Lightweight data library for LigoLW I/O.

POSIX library.

4.7 LDAS data analysis workstations:

FUNCTIONS:

- Provides individual users with desktop computing workstations to access the LDAS environment, perform analyses and interpret results.

CPU:

PC running under linux OS. 450 MHz CPU, with 128MB RAM; UWSCSI interface for peripherals.

DISK: 2 - 4 GB disk for software.

NETWORK CONNECTIONS:

UWSCSI interface for peripherals. LDAS LAN connection will be via OC3 or OC12 (at some future time) on ATM running 100BT.

SOFTWARE:

The GUIAPI provides a visual interface into LDAS. Analysis tasks may be launched or requested from LDAS via point-and-click interfaces. More sophisticated sessions may require the command line scripting interface.

The commandLanguageAPI enables users who prefer a c-shell like scripting environment.

The WEBAPI enables users to invoke plugin software via their browsers. This interface will he the primary method of interaction for remote users.

The remoteFilterAPI will enable users to obtain archived data for proprietary prototyping analyses using personally developed and used filtering algorithms that are not available within the LDAS libraries.

FrameAPI and LightweightAPI to read/write data to local disk spaces.

LIBRARIES:

Lightweight data library for LigoLW I/O.

POSIX library.

4.8 Data Storage (On-line):

This refers to the tape robot shown in ***Figure 2***. The following is excerpted from the CDS/DAQS design documentation and reflects the presently expected implementation of the tape writing facility to record LIGO framed data.

FUNCTIONS:

- Provides the means to record to long-term storage media a raw framed data. Performs continuous backup of data from the local on-line disk cache for shipment of data to the archive repository.

The DAQS disk drive unit is set up in a circular buffer arrangement, such that once full, it begins to remove old frame files to write the new frame files. As an example, the arrangement from the DAQS prototype is shown in ***Figure 13*** and the LIGO DAQS will be arranged in a similar manner.

At present, directories are set up on the disk such that the frame writer produces frame files in a directory for one hour. After an hour, the frame writer starts writing frame files to the next directory and signals the tape control process. The tape control process now backs up the previous hour of data to tape and erases all data files from the directory one hour in advance of where the frame writer is presently active. In the prototype, this continues such that the previous 8 hours of data are always available on disk.

The tape control process also controls the tape robot unit. The units proposed for initial LIGO commissioning contain two drives and 10 tapes in a robotic unit. The tape control process monitors and controls this system, automatically removing tapes as they become full and inserting new tapes. Subsequent to initial operations, the tape write heads will be upgraded to units capable of recording the full LIGO acquisition bandwidth. Additionally, it may be the case that all data from one observatory are recorded to the same tape (or other media) cassette.

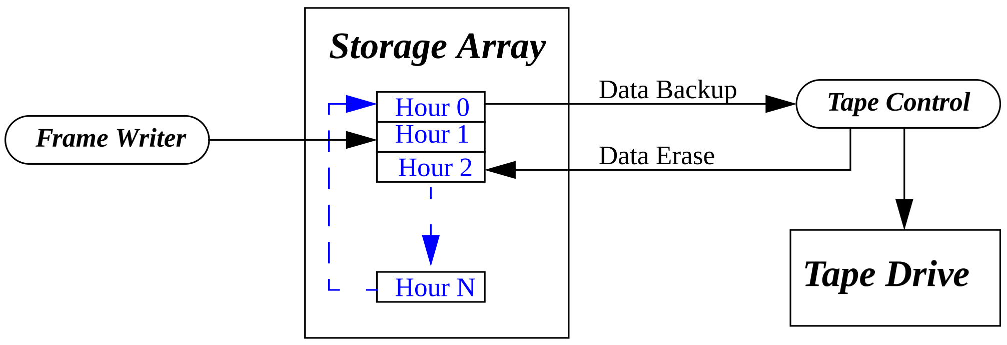


Figure 13 Tape Storage Sequence

4.9 Archive Server (Off-line only):

The software for the off-line server will comprise of the HPSS operating system. Since HPSS appears as a UNIX file system outside the archiving hardware (disk cache/tape cabinets/tape readers/OS), the interface to LDAS will be through data I/O libraries and associated APIs (FrameAPI; LightweightAPI; MetadataAPI).

FUNCTIONS:

- Provides the means and resources to archive LIGO reduced data in frames for ~2 years. Permits retrieval of data to the local disk cache for data analysis.

Experience of the users of large datasets at major computing facilities (e.g., SDSC) indicates that if a standard set of [IBM] hardware is used it is possible to obtain 95% uptime with HPSS. While software ports of HPSS to non-IBM platforms are currently under way and may offer cheaper hardware configurations in the long run, LIGO does not have the resources nor the mandate to play a significant role in porting or further developing HPSS. LIGO plans to obtain a proven stable, scalable solution.

HARDWARE:

The HPSS configuration which LIGO will pursue is as follows:

- A single SP2 rack with a backplane switch.
- 5 4-way PCI nodes.
- 500GB of SSA RAID disk.
- 1 high performance gateway node (HPGN).
- 4 Redwood tape drives.
- few 100GB of TBD disk storage for non-HPSS database.

This assumes that LIGO will have long term access to CACR tape robotics, in particular most of one of the current StorageTek silos. Under this scheme, the LIGO HPSS configuration would run as a storage subsystem within the CACR system, thereby negating the need for a separate licensing fee fro HPSS software. This approach also offers a highly independent configuration that is not subject to performance degradation from non-LIGO uses of HPSS at Caltech as all of the tape drives, data movers, and metadata servers would be operating on LIGO hardware.

Figure 14 depicts the HPSS configuration for the LIGO component. The basic idea is to use the SP2 backplane as a high speed, low latency network to communicate between the various HPSS processes. A single 4-way PCI node should suffice for the metadata services while the other 4 nodes would operate as parallel data movers, allowing for 4-way striped disk speeds.

The gateway node, would interface directly to the SP2 backplane switch and allow high speed access into HPSS via ATM, HIPPI, gigabit, or other high performance network technology. An initial configuration with 1 ATM OC-12 port is listed in the initial configuration.

The redwood drives offer 11MB/s using 50GB uncompressed 1/2inch cartridges. CACR currently has two 6000 slot StorageTek silos, which can each hold ~300TB if populated with these tapes.

It is possible that both the SP2 backplane and 4-way SP2 nodes will have undergone a major performance increment by the time of LIGO's purchase.

No extrapolation has been made for a next generation of tape drive as it is not clear that there will be a superior product available in the time frame that LIGO needs. However, we will continue to monitor the 1/2 inch optical tape drive technology being developed by LOTS, as it is form factor compatible with the existing tape robotics.

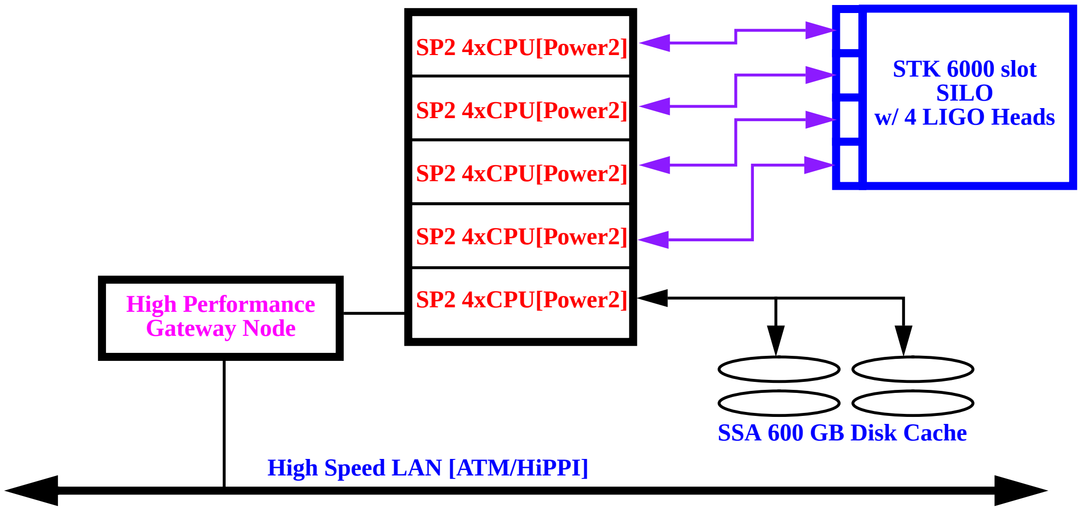


Figure 14 HPSS Configuration for LIGO

5 LAYOUTS

5.1 Hardware locations at observatories

The LDAS hardware will reside in three locations within the Operations and Support Building (OSB) at the observatories. These are discussed below.

The Mass Storage Room (MSR) is where the CDS DAQS framebuilder and disk cache system are located. Here, too, will be located the following LDAS components:

- ATM and other networking hardware
- Data and metadata server(s)
- Data conditioning workstation(s)
- The linux compute cluster (beowulf)
- Media writing hardware (tape drives and robots)

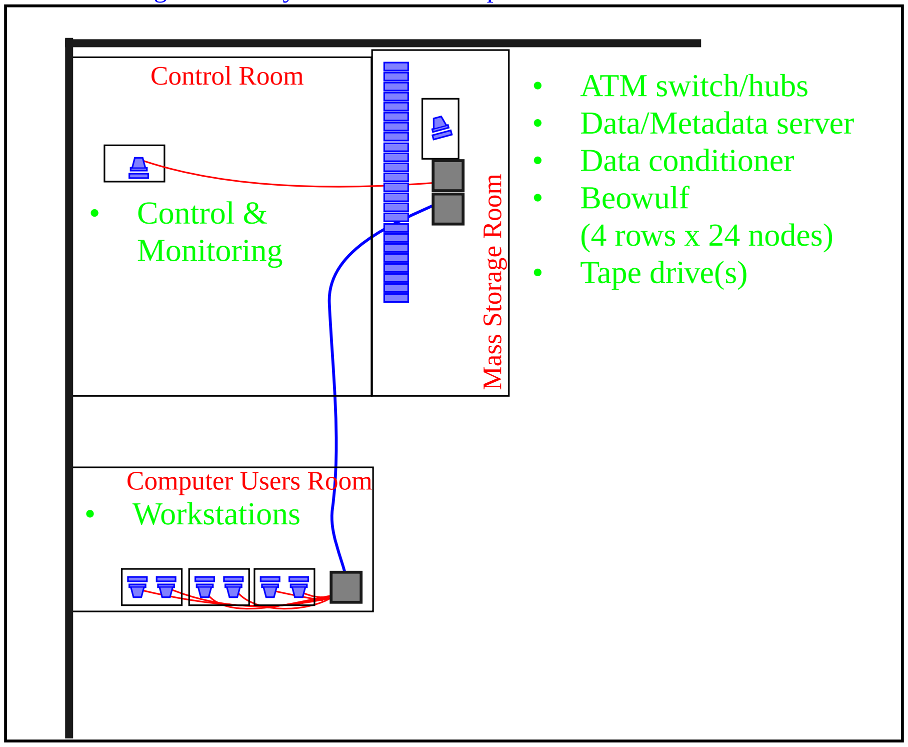
The Control Room (CR) is where the CDS operations workstations are located and where technical and scientific staff will spend time monitoring detector performance. Here, too, will be located the following LDAS components:

- Control and Monitoring server

The Computer Users Room (CUR) is dedicated to data analysis activities that are observatory-based. The following LDAS components:

- LDAS ATM/Etnernet hub
- Analysis workstations

Figure 15 Layout of LDAS components within OSB rooms



APPENDIX A SOFTWARE VALIDATION

The validation of software for LDAS involves a number of different scopes. (i) The system-level APIs and libraries that are being developed exclusively within LIGO Laboratory will have undergone a sequence of validation steps before release. (ii) Other components of LDAS associated with the numerical(filter) libraries that will be used for data processing and event detection will undergo a regimen of validation that will be defined jointly by the LSC and LIGO Laboratory.

System-level software development

Software developed by LIGO Laboratory for LDAS undergoes several levels of validation cycles. The first is at the module level, where individual software modules are tested by their writers using interface programs developed to exercise as fully as possible code modes of operation and functionality.

As different components of LDAS are integrated to interoperate, additional levels of software tests are performed to confirm the integrity of the interfaces that provide for code integration. At this stage, various prototypes will be developed and released to users so that through exercise of the LDAS components any further programming bugs may be identified.

Since most of the processes supported by the LDAS environment at the level described above are not quantitative analyses, but rather data management functions, the integrity of code components is validated by writing and reading or transmitting via sockets large volumes of representative data objects that are created specifically for such tests (i.e., large volume files with random or deterministic numbers).

Timing performance tests are run on representative operations requiring the bulk transfer of large volumes of data to identify hardware or other bottlenecks or limitations that need work-arounds.

LDAS has adopted a form similar to the one developed for CDS to report and track program errors or other flaws that need to be corrected (see the attached *LDAS Maintenance Form*)

Once the LDAS software is in a relatively mature state, complex test suites will be implemented in EXPECT to simulate interactive sessions in which all components of the system are tested at a system level. These EXPECT scripts can then be used with standardized data sets to verify the behavior and results of the system as bug fixes and modifications are made to the software.

Numerical algorithm software development

Numerical libraries and other analysis software will be developed as procedural C code jointly by LIGO and the LSC. The detailed validation methods are presently being identified and requirements are being defined. These will be issued as separate documents. Basically, however, validation will consist of a set of specified numerical analyses to be carried out on specified standard datasets, for which the expected results are known and verifiable. In some cases, validation may be made by comparison of output results between the code component under test and that from previously accepted or validated software

LDAS Maintenance Request Form

Bug/Change Request No.: _________

Problem Report.

Issued by: ______________________ Date: _______________ API: ______________

System: Caltech ☐ MIT ☐ LHO ☐ LLO ☐ other ☐

Problem is: hardware-bug ☐ software-bug ☐ change-request ☐ new-requirement ☐ other ☐

Description:

Priority:
☐ 1) Prevents LDAS operations on-line/off-line (circle one)
☐ 2) Affects essential capability, no work-around available
☐ 3) Affects essential capability, temporary work-aound available
☐ 4) Inconvenience/annoyance
☐ 5) Other (Minor)

Analysis.

Done by: ______________________ Date: _______________

Impact on other systems:____

Estimated time to implement: ______________________

Fix.

Done by: ______________________ Date: _______________

Description:

Time Taken: ________________ Problem tested and signed off by:________________________ Date:_________