Correlator Backend
&
Fast Formatter

Status, Design, Plans, and Schedule

Development tools

old CBE	**new CBE**
no version control	CVS
ad hoc Makefiles	GNU autotools
no testing framework(s)	*dejagnu*, *check* (with *gcov*)
no profiling tool(s)	*oprofile* (*gprof*)

Development tools

old CBE	new CBE
no version control	CVS
ad hoc Makefiles	GNU autotools
no testing framework(s)	*dejagnu*, *check* (with *gcov*)
no profiling tool(s)	*oprofile* (*gprof*)

Development standards

old CBE	new CBE
low modularity	high modularity
little–known libraries	widely–known (standard) libraries
single, documented test procedure	automated system and unit tests

Development tools

old CBE	new CBE
no version control	CVS
ad hoc Makefiles	GNU autotools
no testing framework(s)	*dejagnu*, *check* (with *gcov*)
no profiling tool(s)	*oprofile* (*gprof*)

Development standards

old CBE	new CBE
low modularity	high modularity
little–known libraries	widely–known (standard) libraries
single, documented test procedure	automated system and unit tests

Library usage

~~CWP~~, ~~SU~~, ~~PVM~~, glib, popt, fftw, cblas, *atlas*, *check*

Current performance

Test environment

- Two, dual 3 GHz processor machines connected by gigabit Ethernet
- Lag sets consisting of a single lag frame
- CBE data processing comprising (time-domain) windowing, Fourier transform, integration and ASCII file output
- Simulator used for frame generation

Current performance

Test environment

- Two, dual 3 GHz processor machines connected by gigabit Ethernet
- Lag sets consisting of a single lag frame
- CBE data processing comprising (time-domain) windowing, Fourier transform, integration and ASCII file output
- Simulator used for frame generation

Results

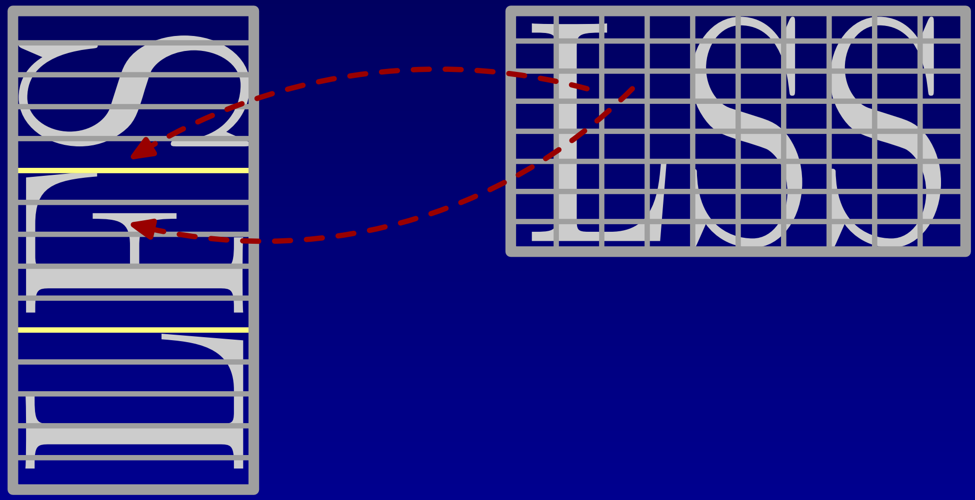
- Achieved frames rates of up to approximately 106,000 frames per second (*i.e.*, $\sim$ 94% of gigabit ethernet total bandwidth.)
- Processor utilization

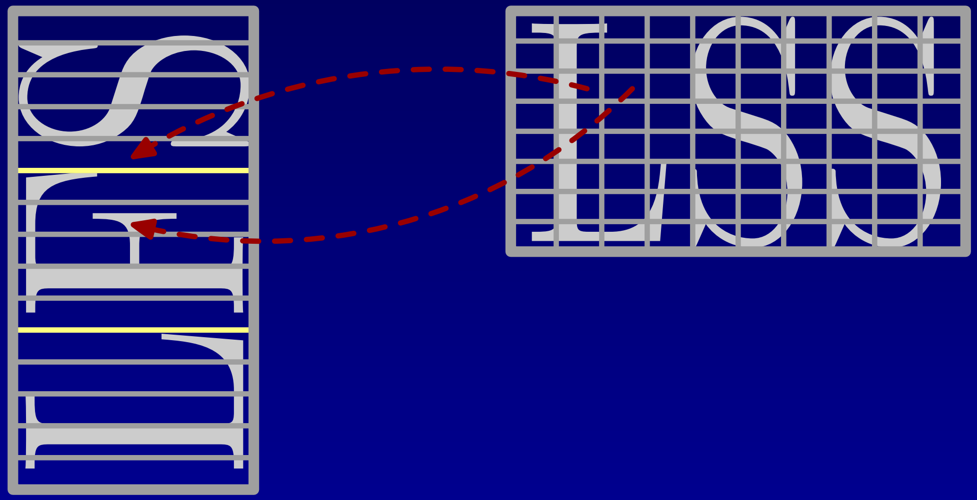
processes	**utilization**
input/sorter	$\sim$ 95%
lagset_proc	$\sim$ 45%

Significant design changes

- Elimination of lag frame store
- Integration of *input* and *sorter* processes
- One–to–one relation of lag set processing processes to lag sets
- Flat (non–hierarchical) indexing of lag frames to lag sets
- Direct lag frame indexing on encoded header words

Elimination of lag frame store

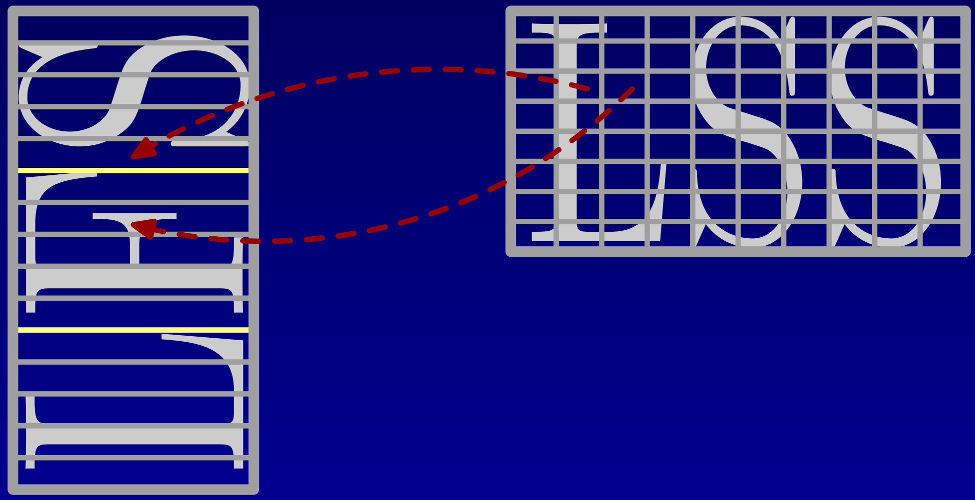




Disadvantages of LFS

- Block structure adds processing latency
- Referenced data can change

Elimination of lag frame store



Disadvantages of LFS

- Block structure adds processing latency
- Referenced data can change

Advantages of LFS elimination

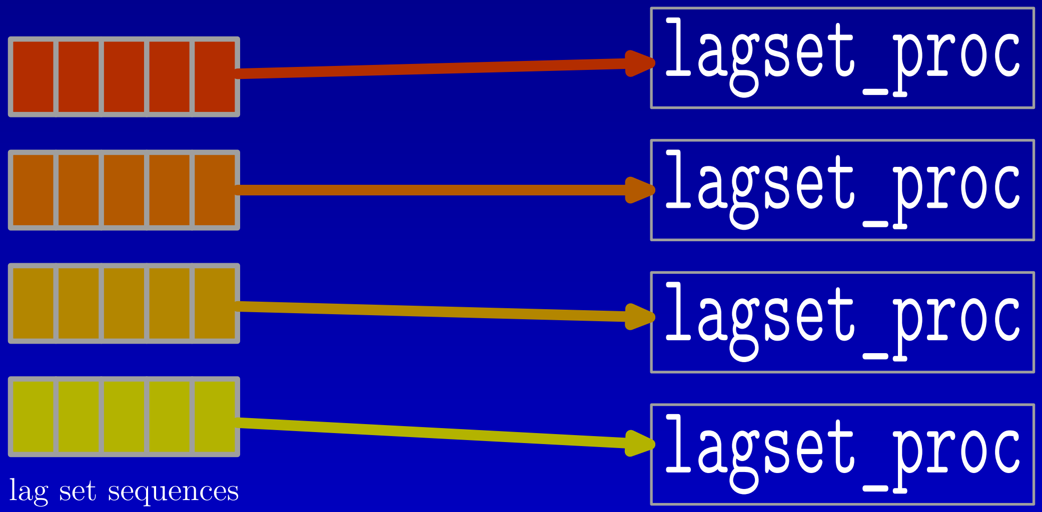
- No added latency in construction of lag sets
- No lag frame references
- Elimination of LFS requires on–the–fly sorting of lag frames into lag sets, inspiring the integration of the *input* and *sorter* processes.

Original design

Lag set processes *vs.* lag sets

Original design

New design



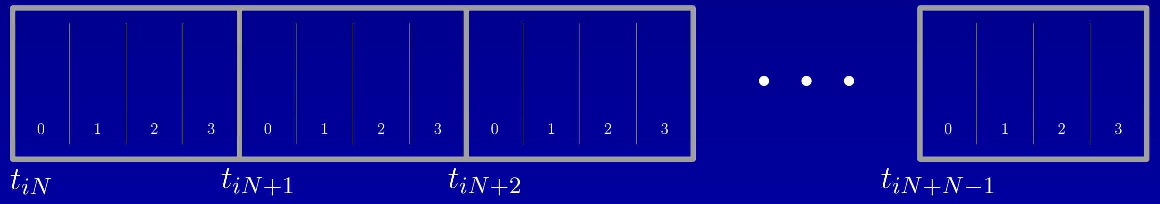
Benefits of new design

- Structure of lag set sequences is simpler than structure of lag set store.
- Lag set processing program is simplified.
- Shared memory accessed by each `lagset_proc` process isolated to one lag set sequence.
- Process isolation for processing of each lag set sequence.
- Lag set sequences are free of structure imposed by lag set store.

Benefits of new design

- Structure of lag set sequences is simpler than structure of lag set store.
- Lag set processing program is simplified.
- Shared memory accessed by each `lagset_proc` process isolated to one lag set sequence.
- Process isolation for processing of each lag set sequence.
- Lag set sequences are free of structure imposed by lag set store.

Lag set sequence structure



- Each element of a lag set sequence contains a partially decoded lag frame.
- Lag sets in the lag set sequence are indexed by time.
- Lag frames within a lag set are indexed by an offset.

Lag frame sorting: fields

Sort, *v. tr.*

1. To map a lag frame to a lag set sequence and frame offset.

Sort, *v. tr.*

1. To map a lag frame to a lag set sequence and frame offset.

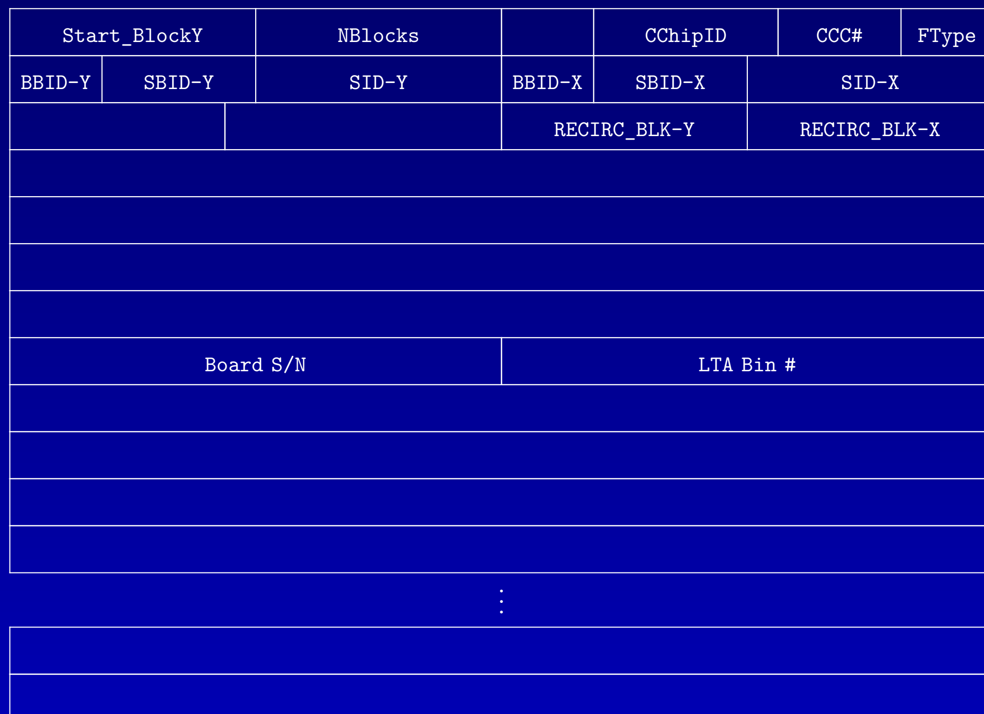
<table>
<tr><td colspan="3">Start_BlockY</td><td>NBlocks</td><td>nlags</td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td colspan="2">SBID-Y</td><td>SID-Y</td><td>BBID-X</td><td>SBID-X</td><td colspan="3">SID-X</td></tr>
<tr><td colspan="2">STATUS_BITS</td><td colspan="2">FRAME_COUNT</td><td colspan="2">RECIRC_BLK-Y</td><td colspan="3">RECIRC_BLK-X</td></tr>
<tr><td colspan="9">TIMESTAMP-0</td></tr>
<tr><td colspan="9">TIMESTAMP-1</td></tr>
<tr><td colspan="9">DVCOUNT-Center</td></tr>
<tr><td colspan="9">DVCOUNT-Edge</td></tr>
<tr><td colspan="4">Board S/N</td><td colspan="5">LTA Bin #</td></tr>
<tr><td colspan="9">Lag0-In_phase accumulator</td></tr>
<tr><td colspan="9">Lag0-Quadrature accumulator</td></tr>
<tr><td colspan="9">Lag1-In_phase accumulator</td></tr>
<tr><td colspan="9">Lag1-Quadrature accumulator</td></tr>
<tr><td colspan="9">⋮</td></tr>
<tr><td colspan="9">Lag127-In_phase accumulator</td></tr>
<tr><td colspan="9">Lag127-Quadrature accumulator</td></tr>
</table>

LTA data frame

Lag frame sorting: fields

Sort, *v. tr.*

1. To map a lag frame to a lag set sequence and frame offset.



Lag frame sorting fields

Lag frame (masked) header words used as sorting key

<table>
<tr><td colspan="2">Start_BlockY</td><td>NBlocks</td><td></td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td>SBID-Y</td><td>SID-Y</td><td>BBID-X</td><td>SBID-X</td><td colspan="3">SID-X</td></tr>
<tr><td colspan="3"></td><td colspan="2">RECIRC_BLK-Y</td><td colspan="3">RECIRC_BLK-X</td></tr>
<tr><td colspan="3">Board S/N</td><td colspan="5">LTA Bin #</td></tr>
</table>

Lag frame (masked) header words used as sorting key

<table>
<tr><td colspan="2">Start_BlockY</td><td>NBlocks</td><td></td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td>SBID-Y</td><td>SID-Y</td><td>BBID-X</td><td>SBID-X</td><td colspan="3">SID-X</td></tr>
<tr><td colspan="3"></td><td colspan="2">RECIRC_BLK-Y</td><td colspan="3">RECIRC_BLK-X</td></tr>
<tr><td colspan="3">Board S/N</td><td colspan="5">LTA Bin #</td></tr>
</table>

Mapping of header words to (lag set sequence, frame offset) pair implemented using a hash table.

- Avoids wasted memory required by an array
- Unstructured
- Speed is not a significant factor
- Mapping can be done without decoding the header fields

Lag frame sorting: mapping

Lag frame (masked) header words used as sorting key

<table>
<tr><td colspan="2">Start_BlockY</td><td>NBlocks</td><td></td><td colspan="2">CChipID</td><td>CCC#</td><td>FType</td></tr>
<tr><td>BBID-Y</td><td>SBID-Y</td><td>SID-Y</td><td>BBID-X</td><td>SBID-X</td><td colspan="3">SID-X</td></tr>
<tr><td colspan="3"></td><td colspan="2">RECIRC_BLK-Y</td><td colspan="3">RECIRC_BLK-X</td></tr>
<tr><td colspan="3">Board S/N</td><td colspan="5">LTA Bin #</td></tr>
</table>

Mapping of header words to (lag set sequence, frame offset) pair implemented using a hash table.

- Avoids wasted memory required by an array
- Unstructured
- Speed is not a significant factor
- Mapping can be done without decoding the header fields

Mapping of lag frame to lag set within a lag set sequence is done by simple timestamp–based indexing of the lag set sequence array.

Review

- All data for one baseline (including all subbands, bins, and polarizations) are sent to one backend cluster compute node.
  - scalability?
- Final data product comprises all backend output data into a single file.

Fast formatter description

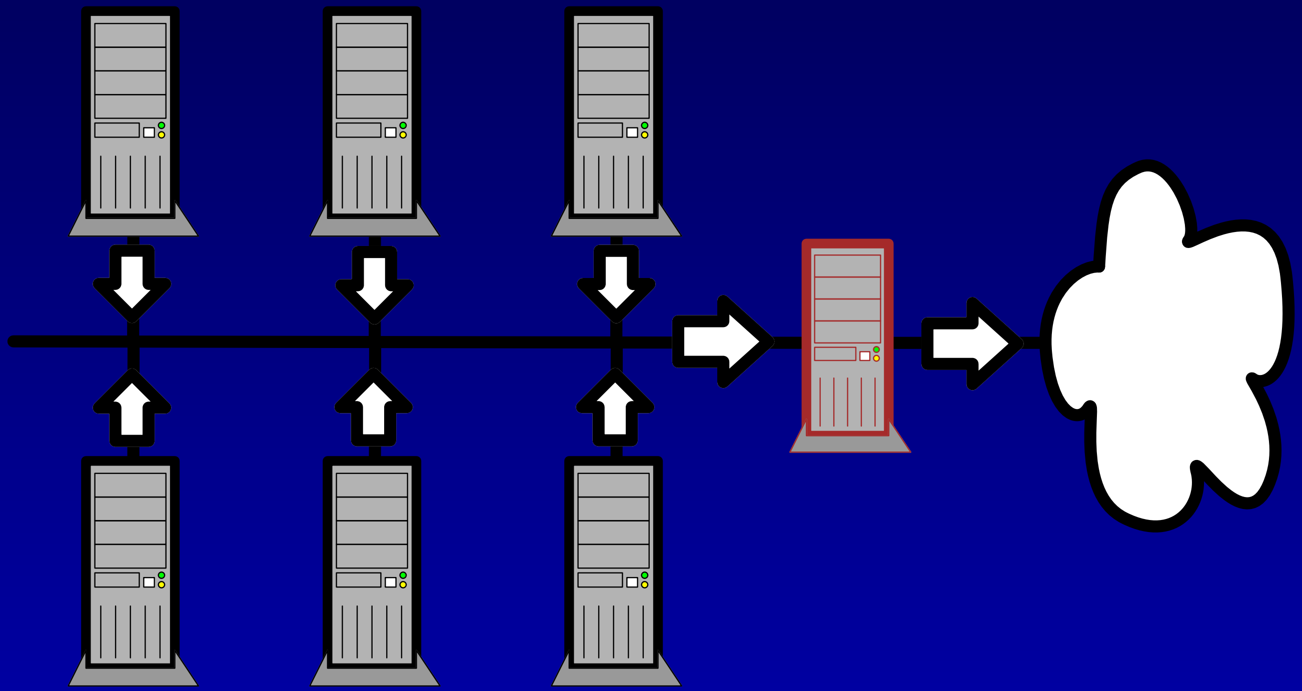
Review

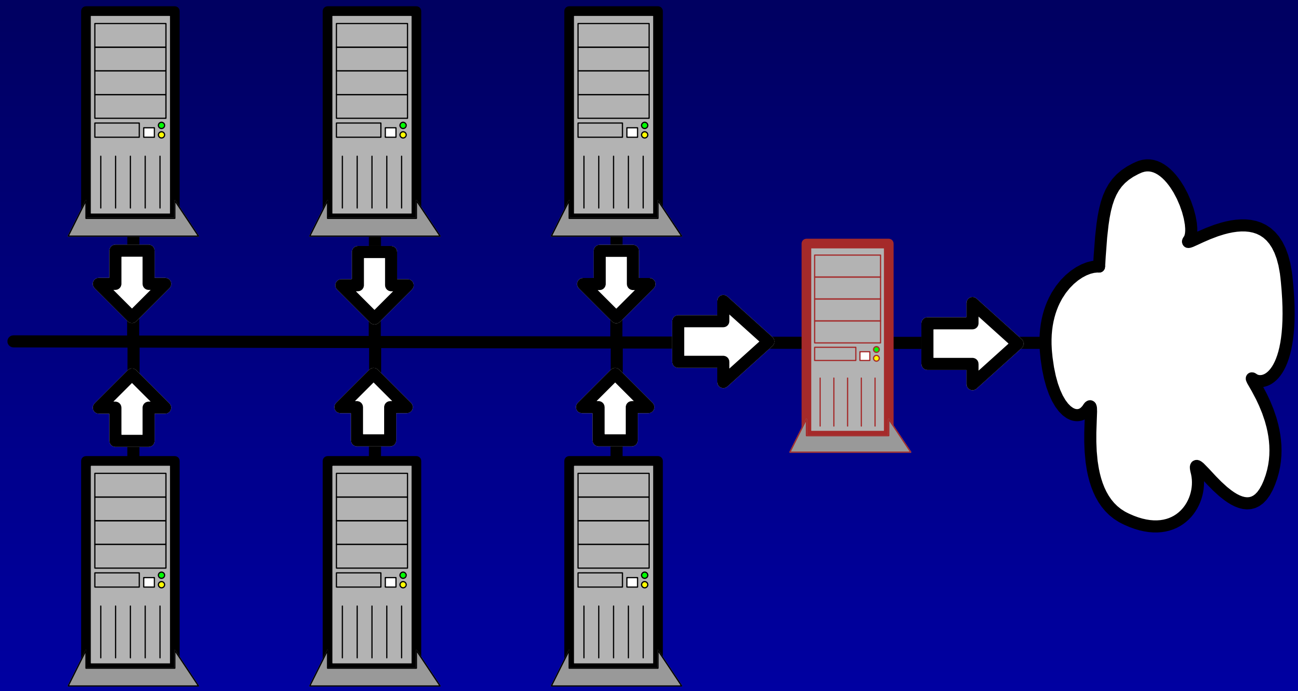
- All data for one baseline (including all subbands, bins, and polarizations) are sent to one backend cluster compute node.
  - scalability?
- Final data product comprises all backend output data into a single file.

Role of fast formatter

- Produce correlator output in binary data format (single file per sub–scan?).
- Binary data format only—no SDM.

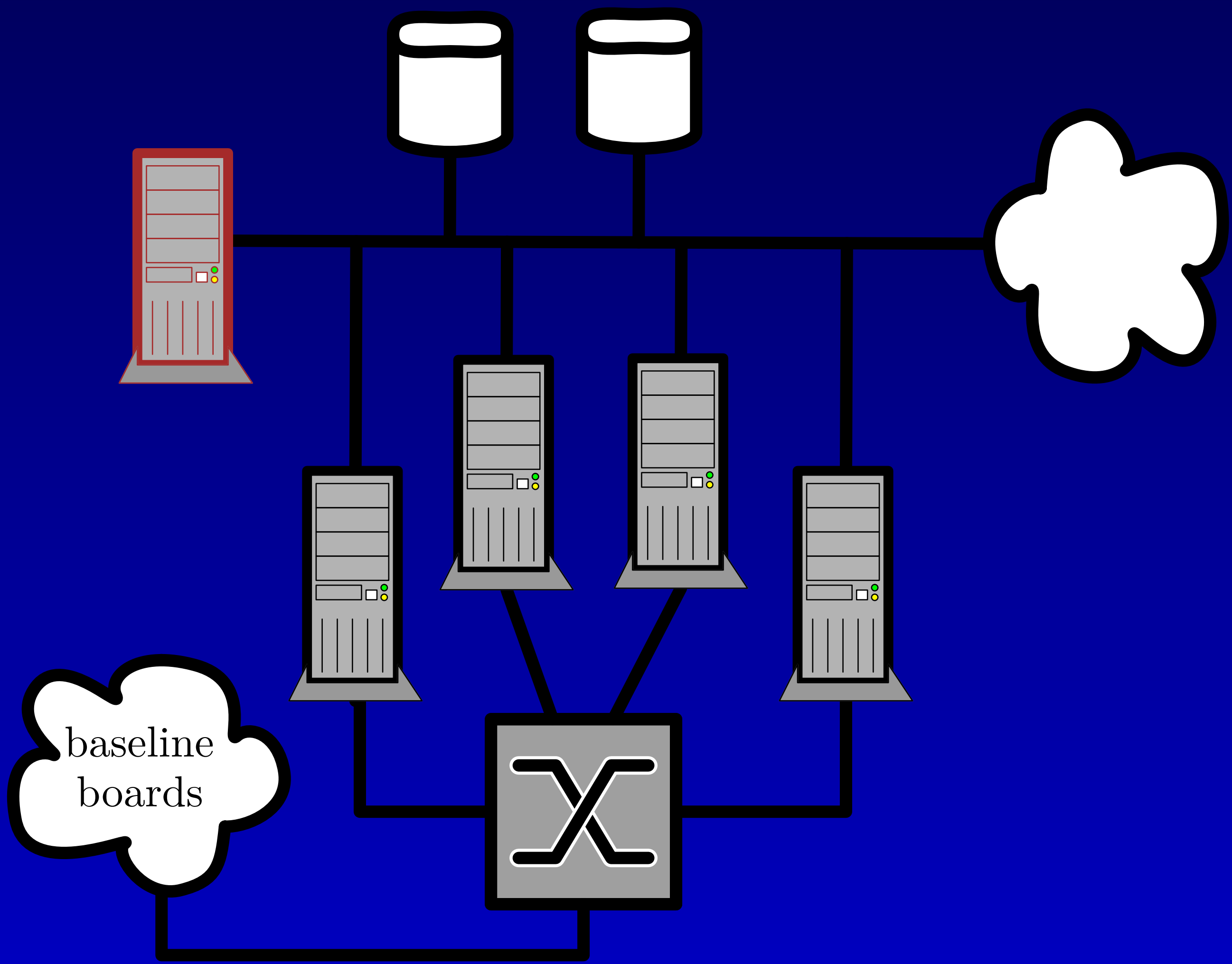
ALMA fast formatter model





Communication channels based on CORBA and/or ACS.

Proposed EVLA fast formatter model



Proposed EVLA fast formatter description

The proposed fast formatter is nothing more than a method by which to write complete files in the required binary data format.

On–line storage for CBE satisfies several requirements

- output data management (3.2.2.18)
  The CBE shall store formatted output data records in a memory buffer with backup disk buffering.
- storage (3.3.2.5)
  The CBE system shall have sufficient disk storage with sufficient access speed to meet short duration correlator bursting demands plus a standby reserve to hold at least 12 hours of output data.
- scalability (3.7.x)

Proposed EVLA fast formatter description

The proposed fast formatter is nothing more than a method by which to write complete files in the required binary data format.

On–line storage for CBE satisfies several requirements

- output data management (3.2.2.18)
  The CBE shall store formatted output data records in a memory buffer with backup disk buffering.
- storage (3.3.2.5)
  The CBE system shall have sufficient disk storage with sufficient access speed to meet short duration correlator bursting demands plus a standby reserve to hold at least 12 hours of output data.
- scalability (3.7.x)

If access to the on–line storage is confined to the CBE, the fast formatter process simply transfers files from the storage area to somewhere else.

Proposed EVLA fast formatter description

The proposed fast formatter is nothing more than a method by which to write complete files in the required binary data format.

On–line storage for CBE satisfies several requirements

- output data management (3.2.2.18)
  The CBE shall store formatted output data records in a memory buffer with backup disk buffering.
- storage (3.3.2.5)
  The CBE system shall have sufficient disk storage with sufficient access speed to meet short duration correlator bursting demands plus a standby reserve to hold at least 12 hours of output data.
- scalability (3.7.x)

If access to the on–line storage is confined to the CBE, the fast formatter process simply transfers files from the storage area to somewhere else.

If the on–line storage is accessible to other subsystems, the fast formatter function is (almost) a NOP.

Proposed EVLA fast formatter

- Scalability
  - File writing distributed across many compute nodes (in parallel)
  - Target filesystem distributed across many storage nodes and/or devices

Proposed EVLA fast formatter

- Scalability
  - File writing distributed across many compute nodes (in parallel)
  - Target filesystem distributed across many storage nodes and/or devices
- No added communication protocol
- Data formatting "intelligence" needed in head and compute nodes
- Some file access synchronization needed
- No dependency on CORBA or ACS
- Some complexity in installing, maintaining and using parallel, distributed filesystem

Proposed EVLA fast formatter

- Scalability
  - File writing distributed across many compute nodes (in parallel)
  - Target filesystem distributed across many storage nodes and/or devices
- No added communication protocol
- Data formatting "intelligence" needed in head and compute nodes
- Some file access synchronization needed
- No dependency on CORBA or ACS
- Some complexity in installing, maintaining and using parallel, distributed filesystem
- Potential for use of parallel, distributed filesystem by other EVLA computing subsystems

Proposed EVLA fast formatter

- Scalability
  - File writing distributed across many compute nodes (in parallel)
  - Target filesystem distributed across many storage nodes and/or devices
- No added communication protocol
- Data formatting "intelligence" needed in head and compute nodes
- Some file access synchronization needed
- No dependency on CORBA or ACS
- Some complexity in installing, maintaining and using parallel, distributed filesystem
- Potential for use of parallel, distributed filesystem by other EVLA computing subsystems
- Candidate filesystems
  - Lustre
  - PVFS2

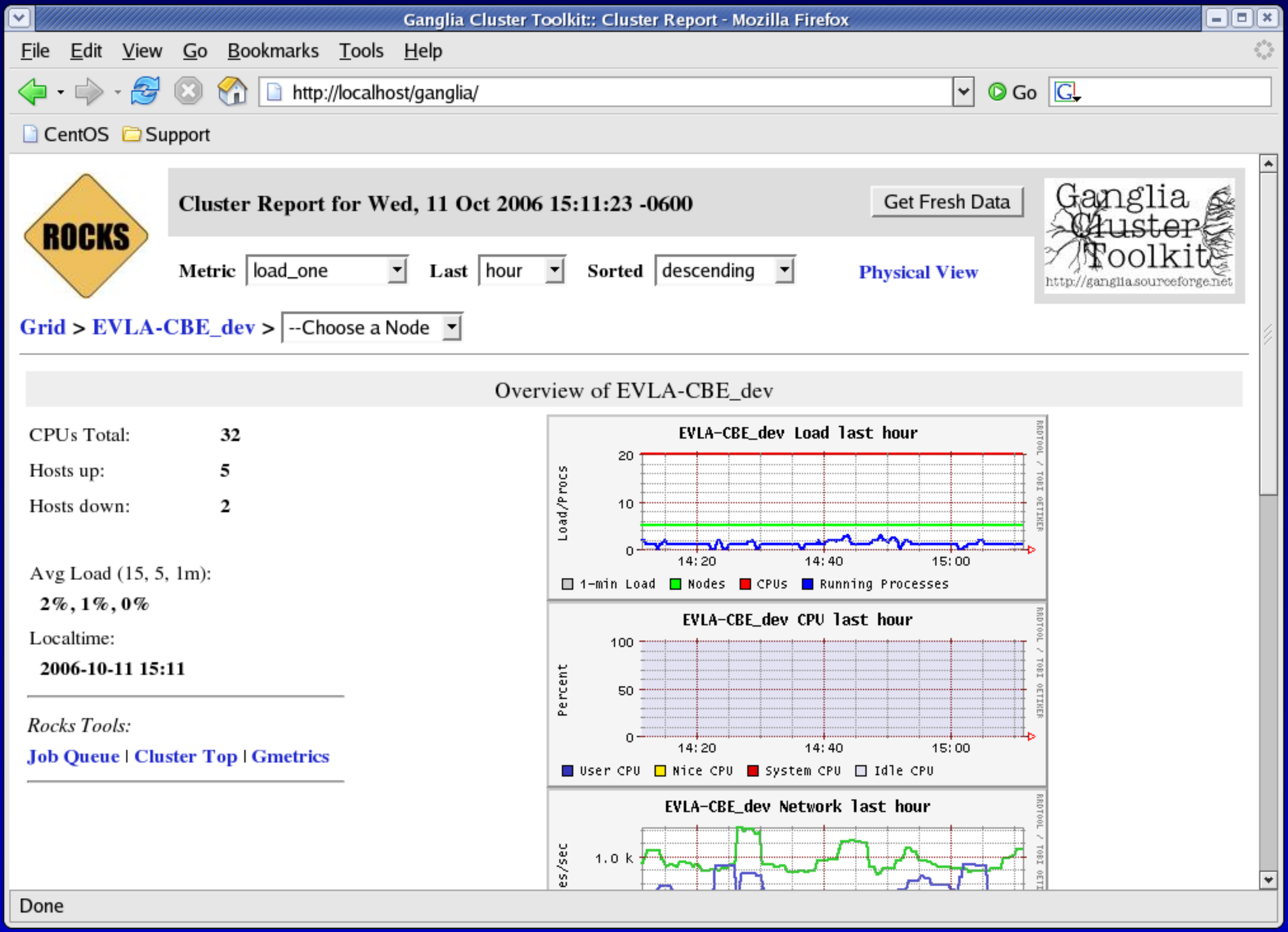
Cluster management

Requirements regarding cluster installation, maintenance, monitoring and control will be met with a third–party package, to the extent possible.

Many packages focus on job control for batch processing using clusters, which we don't need.

Leading candidate under evaluation is "Rocks" (from San Diego Supercomputer Center).

Rocks—1



Top half of cluster monitoring tool webpage

Rocks—2

Bottom half of cluster monitoring tool webpage

CBE M&C

system	framework
old CBE	PVM
current CBE	none
future CBE	OpenRTE (MPI2), PVM, or homegrown? Some tools from cluster management toolkit could also be adopted.

CBE M&C

system	framework
old CBE	PVM
current CBE	none
future CBE	OpenRTE (MPI2), PVM, or homegrown? Some tools from cluster management toolkit could also be adopted.

Communication with MCCC

- Some XML schemas for representing configuration data have been developed based on the original CBE design. These schemas will need updating; for example, more configuration information is needed for fast sorting of lag frames.
- Protocol?

Development outline

target	**needed work**
prototype board testing	• lag set assembly configuration/implementation • 7–bit correlation products
prototype correlator	• binary data format library • configuration schemas • communication protocol with MCCC • auxiliary data (*e.g.*, quantizer power levels) input • Van Vleck correction • timestamp adjustment • residual phase rotation • CBE status and performance monitoring • > 2000 pulsar bins
beyond	• parallel, distributed filesystem support • improved logging (debug, warning, errors, ...) • failover design and implementation • improved system maintenance • more capable lag set processing “scripts” • interference mitigation