P 2.22

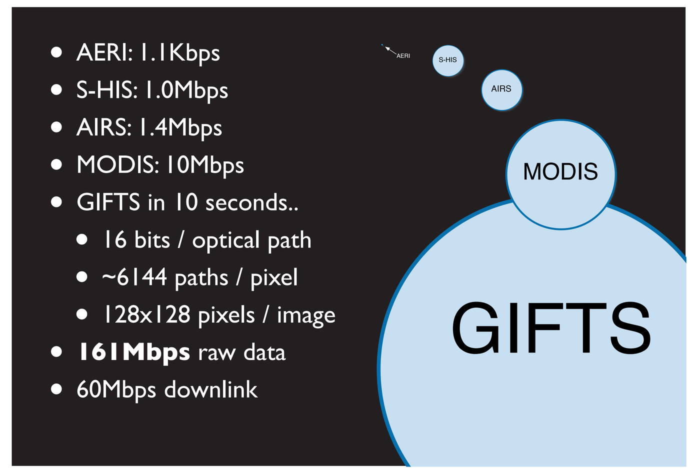
Design Studies for the GIFTS Information Processing System

Raymond K. Garcia, Maciej J. Smuga-Otto*

Space Science and Engineering Center, University of Wisconsin Madison

Objective

Faced with the challenge of enormous data volumes specified by forthcoming imaging infrared interferometers, and the necessity of processing this data in a timely fashion, we have conducted design studies and built prototypes of a **distributed data processing system** capable of meeting throughput and latency requirements.



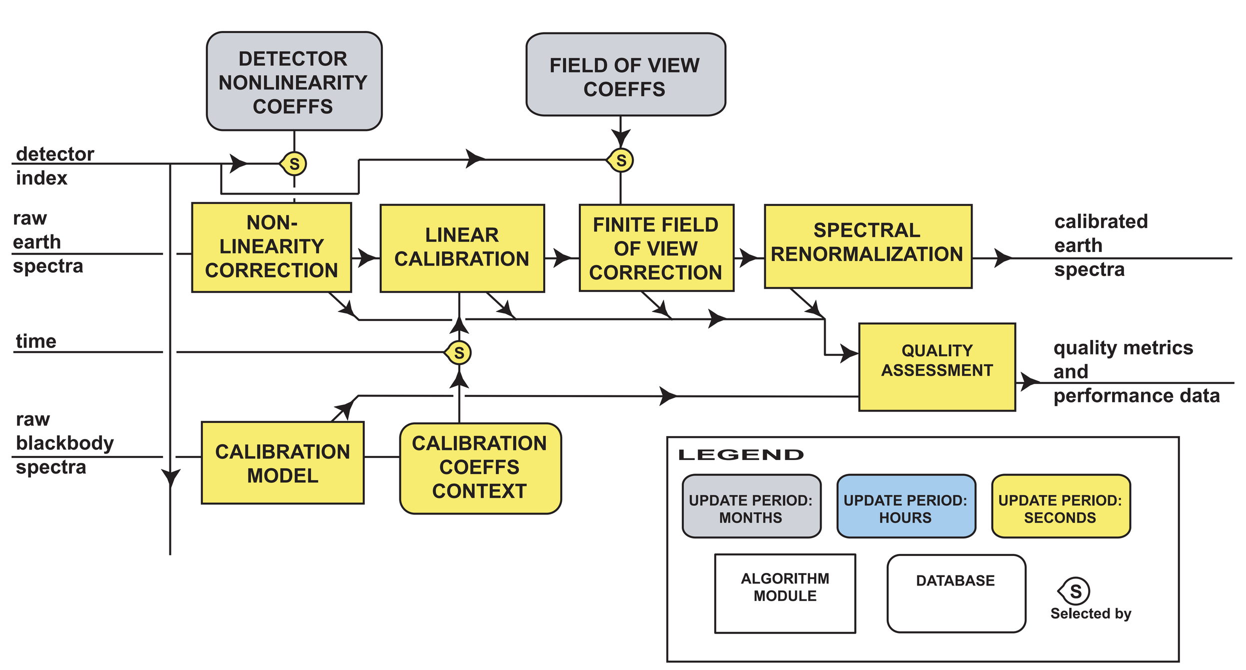
Peak Data Rates

Presented here are architectural concepts and design elements of data processing software leveraging existing expertise and lessons learned from ground-based and aircraft interferometer systems researched, constructed and maintained by the UW Space Science & Engineering Center.

The Science: Data Refinery

Since the algorithms required for the production of L1 GIFTS data (calibrated radiance spectra) are acyclic, the chosen metaphor for the conceptual architecture is that of a **data refinery.** The data is successively "refined" in a series of stages in a numerical pipeline - the stages being the individual science algorithms which act upon the data.

Example of a set of refinery components is the **linear calibration stage.** The purpose of this numerical operator assembly is to assign physically referenced radiances to the instrument measured intensity values.



Calibration pipeline diagram

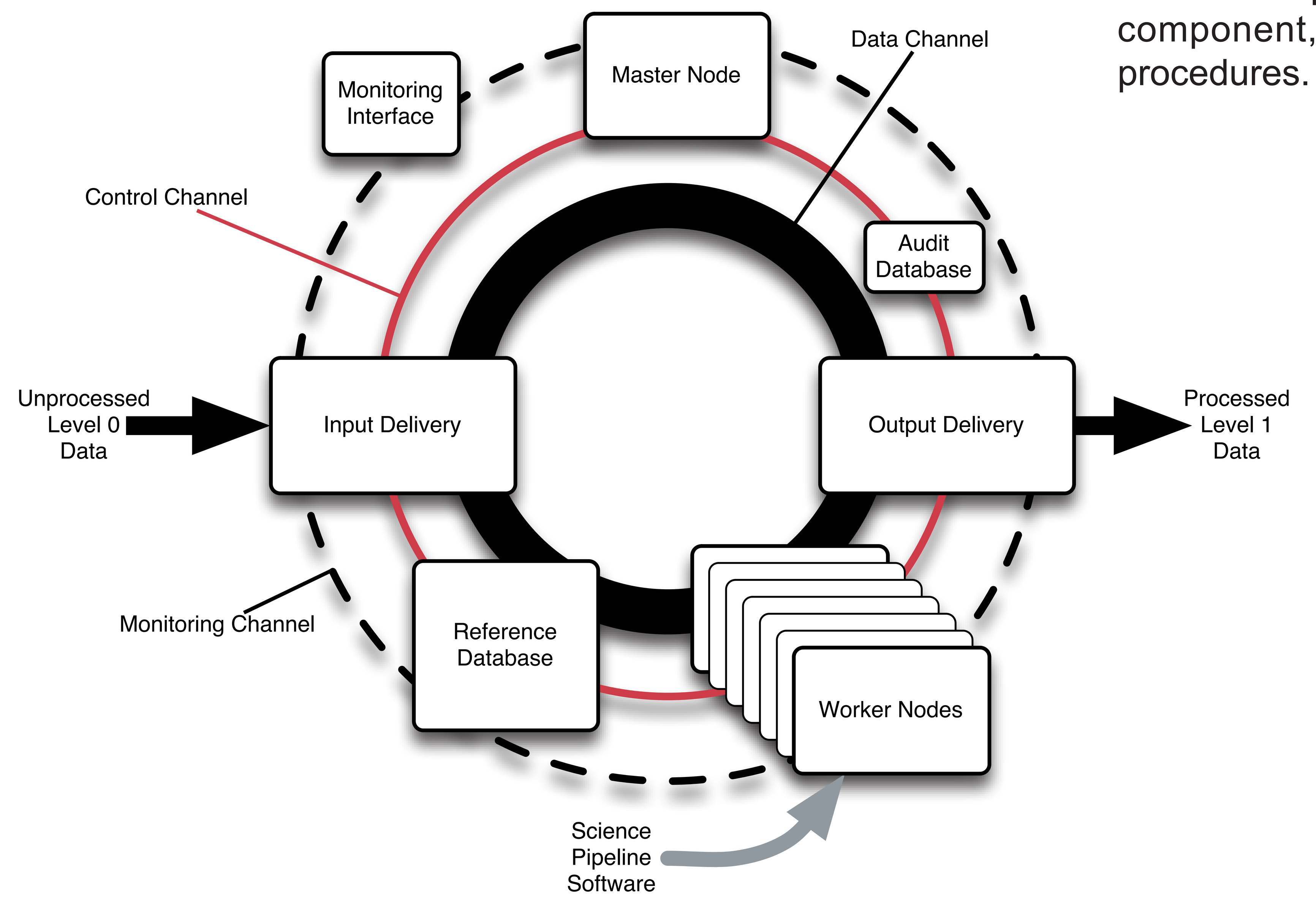
Each spectrum enters the calibration stage as an array of (wavenumber, value) pairs prior to the conversion, and leaves it as an array of (wavenumber, radiance) pairs.

Requirements and Design: Performance, cluster computing, component approach

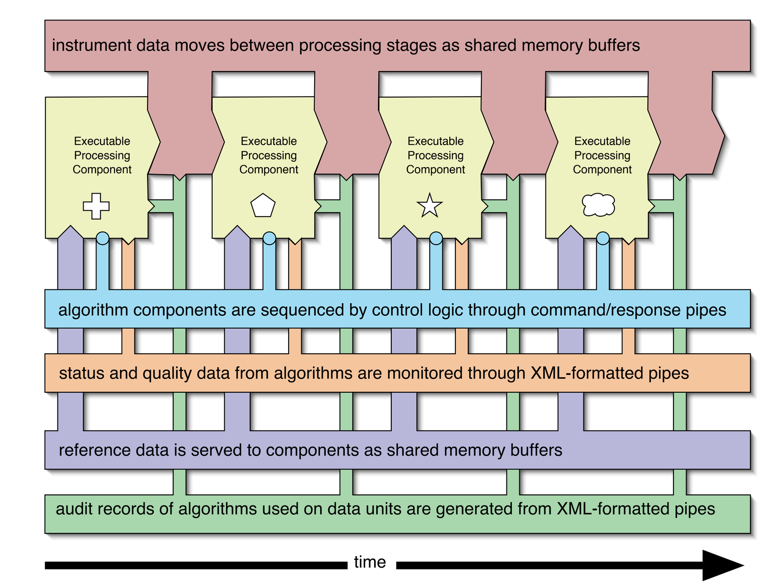
The main challenge lies in running 16384 (128x128) pipelines **with sufficient concurrency** to ensure that the processing system keeps up with data inflow (one GIFTS "cube" every 11 seconds) and with **low enough latency** to meet downstream needs.

To meet this performance demand at reasonable cost, the system will be designed to operate on a **cluster of commodity computers** networked together with a high speed interconnect.

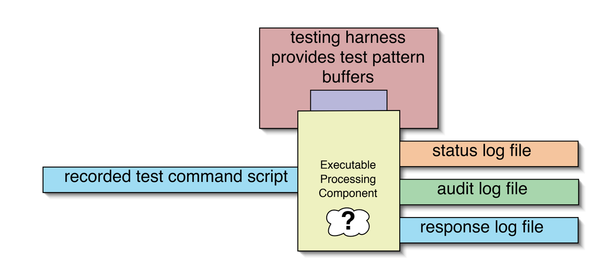
The choice of computing platform drove a design process whose first goal was to **separate the functional components** and communication channels into discrete, individually testable units.
This decomposition enables technology to be chosen per component, and simplifies eventual maintenance procedures.



System Component Diagram



Worker Pipeline Software Design Candidate



Component Unit Testing Harness

Special concern: Audit trails

Data must be annotated with information on which software mix it was processed with, and using what hardware. This helps with:

product reliability- should a hardware instance prove suspect, reprocessing can be scheduled on related data.

algorithm improvement- easy comparison of data re-processed using different versions of refinery stages.

Special concern: Reliability

For a deployed production system, redundancy mechanisms are required so that a failure of any single component does not lead to a failure of the overall system.

Design implications of this, while fairly extensive, do not undermine the fundamental decomposition presented here.

Implementation notes:

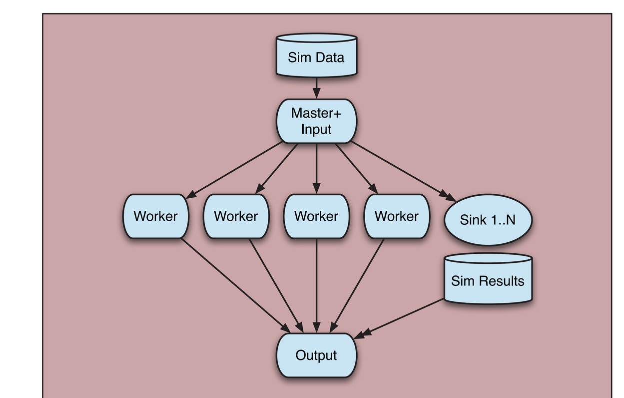
contain the complexity of the task by incremental design and implementation.

First iteration was proof of concept **throughput simulation** on cluster:

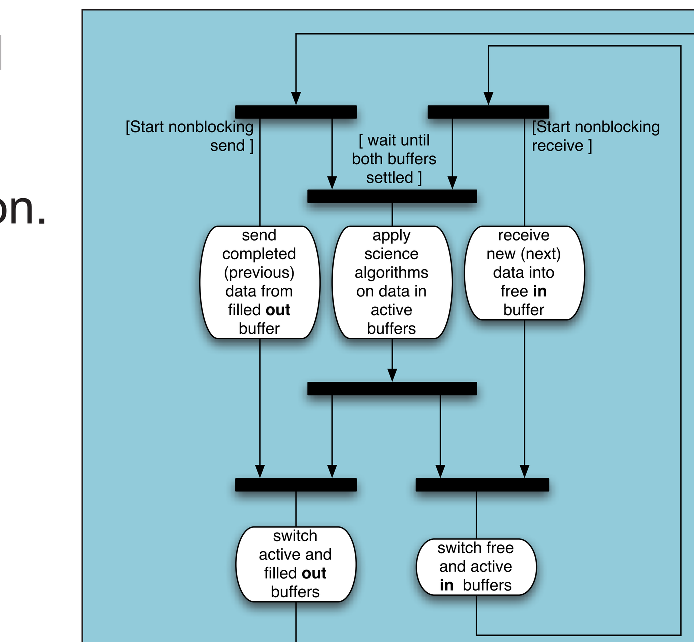
- Emulates larger cluster on small test installation by use of **virtual nodes** and **simulated workloads.**

- Written in C/C++ with MPI

- Worker code parallelizes computation, communication.



Simulation data flow

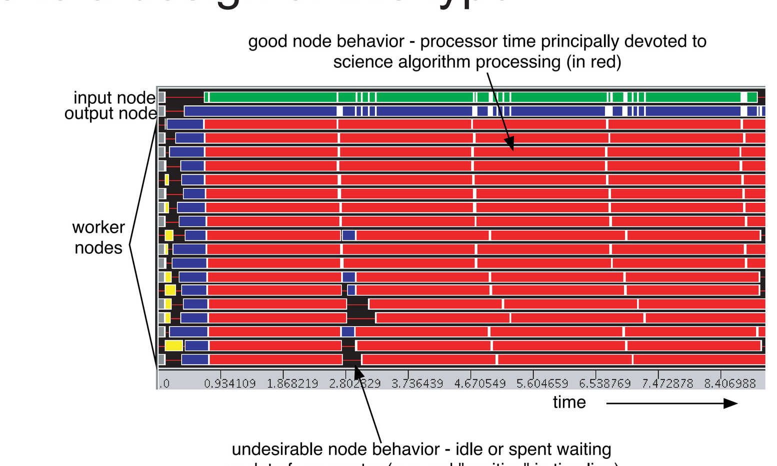


Worker Activity Diagram

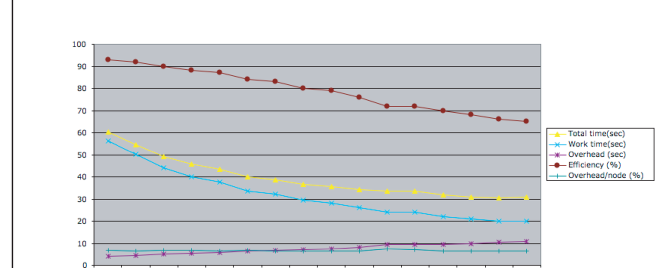
- simulated workloads were based on profile measurements from existing aircraft instrument pipeline.
- Used Jumpshot visualizer and custom timers to evaluate performance
- Demonstrated fundamental tractability of parallel processing approach on cluster, and the utility of modeling the target system using simulated workloads
- Characterized MPI implementation-specific factors and limitations applicable to a design of this type

- per-run startup/shutdown overhead mitigated by increasing size of each run

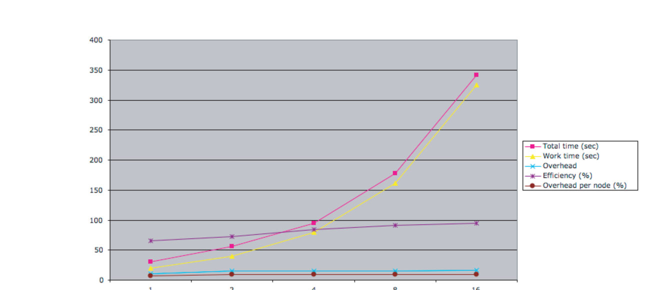
- overcame limitation of MPI on number of simultaneous asynchronous sends from controller by increasing the amount of data to send in each message - which in turn imposed a minimum value for latency of the system.



Jumpshot visualization of a simulation run



Performance vs number of simulated cluster nodes



Efficiency gains when processing multiple data cubes

Second iteration will focus on interfacing pipeline stages to a candidate framework architecture.

* corresponding author: rayg@ssec.wisc.edu