A Robust Framework For Distributed Processing Of GIFTS Hyperspectral Data

Abhishek Agarwal[1], Shahram Tehranian[1], Anand Swaroop[1], Keith Mckenzie[2]

[1]Nortel Government Solutions (NGS)
[2]National Environmental Satellite Data Information Service (NESDIS),
National Oceanic and Atmospheric Administration (NOAA)

Abstract — The Geosynchronous Imaging Fourier Transform Spectrometer (GIFTS) instrument can provide raw data in the order of multiple Terabytes per day. Due to the high data rate, satellite ground data processing will require considerable computing power to process and archive data in near real-time. Cluster technologies employing a multi-processor system combined with a parallel file system is the only cost effective solution for such processing and storage. GIFTS data processing system is required to generate critical products within 5 minutes of gathering observation. In this paper we present an approach for GIFTS ground system processing based on the master-worker paradigm which provides performance, reliability, and scalability of candidate hardware and software using the Message Passing Interface (MPI) standard. The framework used, alleviates the need for earth scientists to understand parallel computing and fault-tolerant operations. Benchmarking results are presented for a selected number of science algorithms for the GIFTS instrument showing that considerable performance can be gained without sacrificing the reliability and high availability constraints imposed on the operational cluster system. A maximum speedup of 54.56 (85.9% efficiency) is obtained for a total number of 64 processors over 64 data cubes of 128 x 128 pixels in the long wave and short-medium wave spectral range. This prototype system shows that considerable performance can be gained for candidate science algorithms without sacrificing reliability and high availability needed for a real-time system.

Index Terms − **Real Time Systems, Parallel and Distributed Computing, Task Scheduling, Reliability**

I. Introduction

Future satellite instruments can provide raw data of the order of multiple Terabytes per day. Due to the high data rate, satellite ground data processing will require considerable computing power to process and archive this data in near real-time. The primary mission of the National Oceanic and Atmospheric Administration (NOAA) is to understand and predict changes in the Earth's environment which requires a continuous capability to acquire, process and archive data in real-time.

NOAA participation in the GIFTS technology transfer represents a risk reduction activity in the design of the NOAA GOES-R series of imager and sounder instruments and their associated science algorithms. The GIFTS instrument uses a combination of Large area Focal Plane Arrays (LFPA's), and a Fourier Transform Spectrometer (FTS), providing a spectral resolution of 0.6 cm^{-1} for a 128 x 128 set of 4 km foot-prints every 11 seconds [1]. It is anticipated that the GIFTS Level-0 data rate is about 55 Mbps or about 1.5 Terabyte per day [1]. The computing power needed for Level-0 to Level-1 processing is not only due to the data volume but also due to the inverse Fourier transform and non-linearity correction. The volume of Level-1 data is approximately the same as Level-0. Since there is little reduction in data volume from Level-0 to Level-1, producing Level-2 data also requires significant computing power. GIFTS data processing system is required to generate critical products within 5 minutes of gathering observation. Cluster technologies employing a multi-processor system present the only current economically viable option to accommodate the processing needed for GIFTS ground data processing. However such systems are inherently unstable; failure of one component may result in a failure of the system if necessary measures are not taken. Operational real-time systems need to be reliable and fault-tolerant, operate on continuous data streams and be operator-friendly. To sustain high levels of system reliability and operability in a cluster-oriented operational environment, a fault-tolerant data processing framework is proposed to provide a platform for encapsulating science algorithms and hide the complexities involved with an operational cluster system. Many Earth science algorithms are very complex, but they have only a small degree of spatial dependency and thus are ideal for parallel processing.

Designing a highly reliable operational cluster system for NOAA's future ground segments is the focus of this paper. In the rest of this section, work reported in the literature that is most relevant to our work is briefly discussed. In section 2, we describe GIFTS instrument and the associated since algorithms. Section 3 describes framework architecture used for GIFTS processing. Section 4 describes the implementation details of the prototype system. Section 5 shows benchmarking results and tradeoff study for GIFTS ground system processing. Section 6 concludes the paper and comments on future directions of this work.

II. GIFTS Instrument and Science Algorithms

A. GIFTS Instrument Background

The GIFTS instrument consists of large area focal plane detector arrays (128 x 128 pixels) within a Fourier Transform Spectrometer (FTS), mounted on a geostationary satellite. The instrument provides observations of Earth infrared radiance spectra at high spectral resolution (0.6 cm^{-1}) and high spatial resolution (4 km x 4 km pixels). Depending on spectral resolution, GIFTS views a large area (512 km x 512 km) of the Earth within a 1 to 11 second time interval. Extended Earth coverage is achieved by step-scanning the instrument field of view in a contiguous fashion across any desired portion of the visible Earth. A visible camera provides daytime imaging of clouds at 1 km spatial resolution. Figure 1 shows a selection of GIFTS measurement modes.

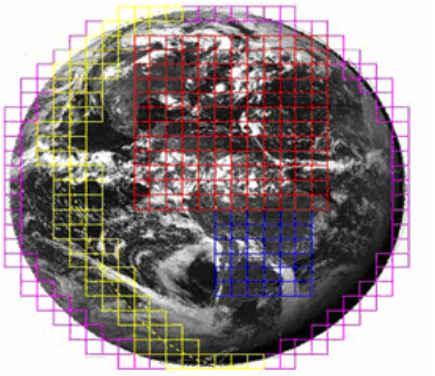

Figure 1: A selection of GIFTS measurement modes. Each box represents the 128 × 128 Large area Focal Plane detector Array (LFPA) [1].

GIFTS uses two detector arrays to cover the spectral bands 685 to 1130 cm^{-1} and 1650 to 2250 cm^{-1}, as shown in Figure 2, and a Michelson interferometer to obtain the spectrum of radiance within these bands [1]. The spectral resolution of the measurements is sufficient to resolve, within 1-2 km vertical resolution, dynamic features of the atmospheric temperature and moisture profiles. The geostationary platform enables the tracing of fine scale features of the atmospheric water (cloud and vapor) distribution to permit the derivation of altitude resolved wind profiles. Nevertheless, GIFTS will cover a major portion of the visible disk with high vertical resolution soundings in less than 30 minutes [1]. This feature is important for obtaining wind profiles from geostationary temperature and moisture sounding data. As part of University of Wisconsin's (UW's) algorithm development [2], simulations of expected top of atmosphere (TOA) radiances are being used for algorithm development and processing.

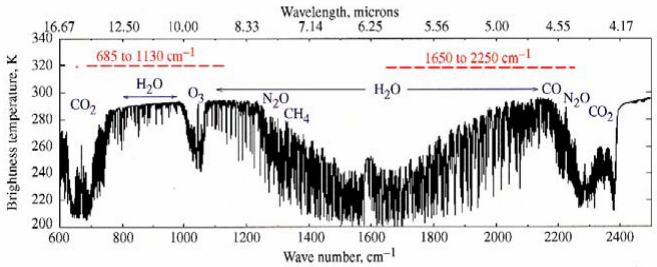

Figure 2: GIFTS spectral coverage with two detector arrays [1]

B. GIFTS Science Algorithm Pipeline

The GIFTS sensor will sample the interferogram from each detector as a function of optical path delay and numerically filter the data in real-time to reduce the data rate before transmission to the ground-based X-band receiver. The sensor collects views of the onboard calibration references and deep space at regular intervals. The ground reception facility will decode the telemetry stream and pass the GIFTS sensor data in real-time to a ground data processing facility [2, 3]. The GIFTS science algorithms are developed by Space Science and Engineering Center (SSEC) at University of Wisconsin (UW). These algorithms can be described as a pipeline consisting of a set of modules including an initial Fast Fourier Transform (FFT), Non-linearity Correction, Radiometric Calibration, Spectral Calibration, Instrument Line Shape Correction and Spectral Resampling as shown in Figure 3. These modules are described below but complete details can be obtained from Knuteson [1].

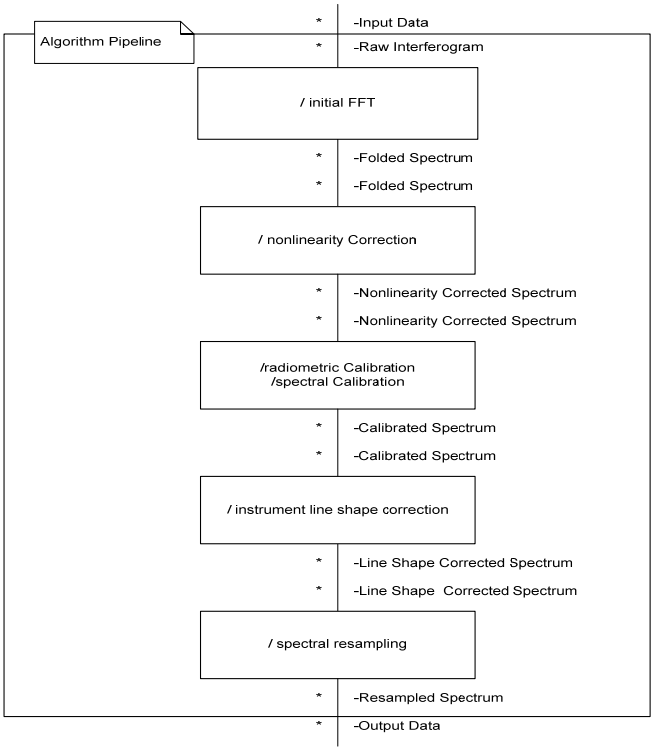

Figure 3: GIFTS science algorithm pipeline

Initially, FFT operations are applied to GIFTS data to convert the measured interferograms into complex spectra. A complex Fourier transform and data folding is performed to convert the complex interferograms to complex spectra corresponding to a wave number scale. In second step we apply non-linearity correction. This work is still under investigation by SSEC, UW. It is expected that the GIFTS detector material is highly linear in the range of photon fluxes used, but the electronics readout of the focal plane array can introduce small signal non-linearity. The non-linearity correction algorithm is being designed using ground-based AERI [4] instrument as well as the Scanning-HIS instrument [4]. In the third step, we perform radiometric calibration. This step ensures the GIFTS instrument requirement to measure

brightness temperature to better than 1 K, with a reproducibility of ±0.2 K. GIFTS uses views of two on-board blackbody sources (300 K and 265 K) along with cold space, sequenced at regular programmable intervals. The temperature difference between the two internal blackbody views provides the sensor slope term in the calibration equation, while the deep space view corrects for radiant emission from the telescope by establishing the offset term. This is followed by spectral calibration, which uses ground-based and aircraft FTS systems. The spectral characteristics of these instruments are defined by an Instrument Line Shape (ILS) and a spectral sampling interval. The spectral sampling scale is maintained very accurately by the stable laser used to trigger sampling at equal intervals of Optical Path Difference (OPD). The next stage is to perform the instrument line shape correction. It has been evaluated that this effect is very negligible for GIFTS instrument since it has an extremely small range of angles contributing to each individual detector pixel (< 1 mrad in the interferometer). As a result, the variation of it is extremely small and could even be ignored without introducing significant errors. Once the spectral calibration is determined we perform wave number resampling. The GIFTS radiance spectrum can be resampled from the original sampling interval to a standard reference wave number scale. The resampling is performed in software using FFT, "zero padding", and linear interpolation of an oversampled spectrum. The results of the wavenumber resampling operation are equivalent to GIFTS spectra with a common wave number scale independent of their location in the focal plane array.

Our benchmarking results as presented in section 5 include the initial Fast Fourier Transform (FFT), the radiometric calibration and the spectral resampling stages, since the other modules are still in the development phase and not yet available for use in the framework.

III. GIFTS Framework Architecture

In this section we propose an architectural model for a robust and reliable operational satellite data processing framework [5] consisting of:

- Active/Standby Master
- Active/Standby Data Input Server
- Active/Standby Data Output Server
- Reference and Audit Database Servers
- Workers

The master is responsible for cluster management and task scheduling. The data input server provides the real-time data which will be retrieved by the workers while the data output server gathers the results produced by the workers. The reference database server provides access to a database storage unit which may contain instrument factory parameters, science algorithm specific parameters, algorithm descriptions, and algorithm initialization parameters. The audit database server provides monitoring capability of algorithms used, produced products and so forth. The workers do the actual processing of the science algorithms.. A Front End Processor (FEP) receives (FEP) receives the data from the satellite, processes the data into packets and frames according to the CCSDS format description. The frames constitute interferometer measurements, performance, engineering and diagnostics data. The FEP provides the input data to the data input server and is not part of the framework. Redundancy for the master, data input and data output servers are provided through an active/standby mechanism. Parallel file systems with server fail-over capability may serve as input and output servers. The databases, consisting of the reference database and the audit database, are envisioned to consist of their own commercially available database clusters such as the MySQL Cluster [6] for fault tolerance purposes.

The master is responsible for cluster management and task scheduling. A *task* is referred to as a unit of work which contains its own set of data, initialization parameters, reference id, timestamp and so forth. Each task has a unique identifier which differentiates it from other tasks within the system. Tasks may also be prioritized in which case tasks with higher priorities will be executed before low priority tasks.

The data input server retrieves the new incoming data, packages them appropriately and assigns unique identifiers to tasks encapsulating the new data sets. The identifiers for the new tasks produced by the data input server are sent to the master, which in turn schedules the new tasks to be assigned to workers.

The master may also define scheduling policies in a way such that certain tasks are assigned to specific workers. We refer to this as *selective* scheduling. Such policies are beneficial in cases where specific tasks may need initialization parameters, which may reside on specific workers and may be computationally time consuming if they had to be recomputed. By assigning such tasks to the same worker or the same group of workers and using application-aware caches [7], a large amount of computation may be saved resulting in higher performance. All communication between the various servers may be performed asynchronously to overlap computation with communication.

A task assigned to a worker by the master may not complete on time for various reasons such as network or worker node failure. A task is considered completed by the master when the task identifier is returned to the master by the worker. To implement task and data redundancy mechanisms, the task execution time, also referred to as task latency, has to be known in advance. Specifying task latency statically is unfeasible for most practical parallel applications. In the current design, task execution times are estimated dynamically based on previous actual task execution times and an over-estimation factor. The over-estimation factor for each task can be calculated as the inverse ratio of its estimated and actual execution time. The estimated execution time may be computed by taking the average of a certain number of previous task execution times multiplied by an over-estimation factor. The over-estimation factor may or may not be a constant.

A task which has not been completed within its estimated execution time is considered lost by the master and will be

reassigned to a worker requesting tasks [8]. The master keeps two queues of tasks, one for new tasks which have not yet been assigned to workers, and one for assigned tasks. When the master schedules a task to be assigned to a worker process, the task in the head of the queue for new tasks is retrieved and assigned to an appropriate worker. This task is then moved to the queue of assigned tasks. When the master receives a notification from the worker that the respective task has been computed, the task is removed from the queue of assigned tasks. However if this task is not completed within its estimated execution time, it is considered lost. Therefore the task is removed from the queue of assigned tasks and is placed in the head of the queue of new tasks to be reassigned to a new worker. The worker which was initially assigned the respective task is considered faulty and will be removed from the list of available workers.

IV. GIFTS Prototype Implementation

The prototype system includes an HP Linux cluster consisting of 32 dual core AMD Opteron DL145 compute nodes with 1 GB RAM per CPU and 4 dual core AMD Opteron DL385 management nodes with 1 GB RAM per CPU. The system runs Red Hat Linux Enterprise Server for AMD64 and has a Gigabit Ethernet as well as Myrinet interconnect for the entire system. HP Serviceguard is used to provide failover capability for the master node. HP Serviceguard monitors the health of each master node and rapidly responds to failures in a way that minimizes or eliminates application downtime. This system is also supplemented by an HP StorageWorks Scalable File Share (HP SFS) which is used to access data in parallel from all compute nodes. It has 16 Terabytes of usable storage and provides a bandwidth of 1064 Mbytes/s for READ and 570 Mbytes/s for WRITE. It uses the Lustre parallel file system which is one of the most extensively used parallel file systems [9].

A. Framework Implementation

The framework is implemented using the Message Passing Interface (MPI) [10, 11] and the C++ programming language. Since there are many implementations of MPI, we evaluated various implementations such as OpenMPI [12], MPICH-2 [13] and LAM-MPI [14]. Finally we choose the MPICH-2 implementation considering its support for MPI-2 [11] standards. Our framework prototype is a complete implementation of all the various components as described in section 3 except for the FEP. For the current framework implementation, the FEP was not used to provide the data; rather the data was directly fetched from the input server. The framework separates the science algorithm layer from the cluster management layer and provides an operational platform within which algorithm software pipelines can be deployed for satellite data processing. There are classes and API's for each service such as master, input delivery, output delivery, reference database, audit database, and workers. The system can be configured at startup using configuration files where hostnames may be specified for the various worker nodes, reference and audit database and so forth. Once the master finishes initialization, it starts the workers and communicates a unique identifier to each worker. Once connection is established, if there is an outstanding task to be processed, it will be assigned to a worker by the master. In case a worker is removed at run-time, its tasks are reassigned to other workers as described in section 3. In case of a master failover, all worker – master connections are lost and the worker processes are cleaned up. The failover process from standby to active involves reading the last check-point file written by the active unit, re-creating the system state accordingly and failing over from standby to active unit. The standby master takes over and creates new worker processes as well as redistributes tasks to the workers.

The framework provides algorithm independence through a set of base classes and interfaces for the algorithmic tasks [5]. Each independent work unit is encapsulated in the *Task* class containing fields for start time, completion time, compute time, unique task identifier, algorithm identifier, number of pixels, input and output data sizes, and other initialization parameters common to most satellite data processing algorithms we have evaluated so far. A task is started as soon as the master schedules it for a specific worker and is considered completed as soon as the master receives a notification from worker that the respective task has been completed. Hence the completion time for each task includes communication, read, write and computation time, respectively. The compute time stored in the task contains the actual computational time for the task as observed by the worker. Each task has a unique task identifier to differentiate it from other tasks. Each task contains a field for algorithm identifier specifying the algorithm/algorithms to be applied to the data. Each task contains reference to its own set of input and output data files. Workers execute a task simply by calling the *execute()* method on an instance of the *Task* class. This way the science algorithm layer is clearly separated from the framework layer.

V. Results

In this section we investigate our framework prototype in terms of performance and reliability. The framework prototype is evaluated using GIFTS science algorithms as described in section 2. Our benchmarking results show end to end performance using simulated data cubes as well as the effect of task size on the performance.

A. Performance

Our benchmarking results include the initial FFT, the radiometric calibration and the spectral resampling stages, since the other modules are still in the development phase and have not yet been released. For the benchmarking, a total of 64 data cubes are used. This results in sufficient amount of work to be provided to each worker to hide any anomalies. Each data cube has a total of 128 x 128 pixels. In the first experiment we have used a constant task size of 64 data cubes and shown the execution time, speed up and efficiency for the GIFTS pipeline. Two spectral ranges for the datasets are used: the long wave infrared band (685-1129 cm^{-1}) and the short-medium wave band (1650-2250 cm^{-1}). Figure 4 shows the

total execution time versus number of processors using the end-to-end GIFTS pipeline processing. Figure 5 shows the speedup versus number of processors for the same experiment. It is shown that we can achieve a linear speedup of 54.56 compared to an ideal speedup of 64. Figure 6 shows the

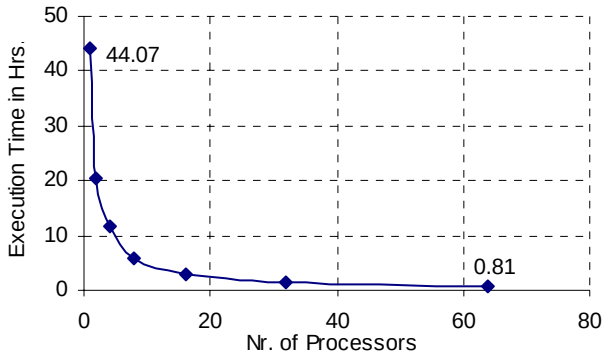

Figure 4: Execution time versus number of processors for 64 data cubes

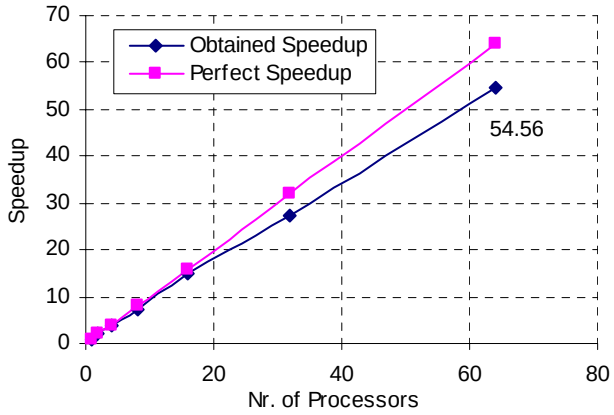

Figure 5: Speedup versus number of processors for 64 data cubes

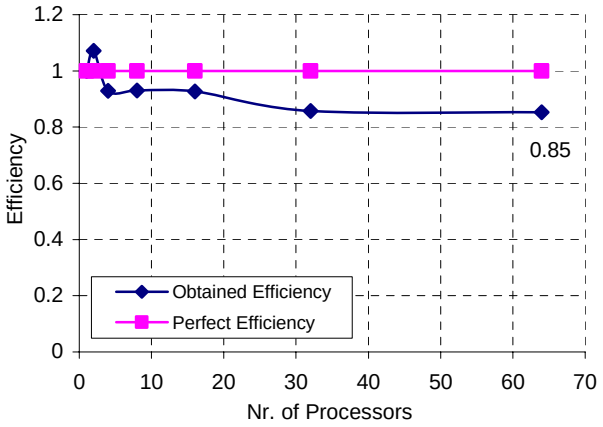

Figure 6: Efficiency versus number of processors for 64 data cubes

the efficiency versus number of processors. It can be seen that we are able to maintain an efficiency of 85 percent throughout the experiment.

In the second experiment, we evaluated the effect of network bandwidth by increasing the task size in terms of number of pixels per task. This resulted in reduced number of messages between the workers and the master which in turn reduces the worker – master communication. This experiment was conducted using a total of 64 data cubes running on 64 processors. The experiment was conducted using tasks containing 128, 256, 1024, and 2048 pixels. Figure 7 shows the total execution time versus task size. Increasing the task size from 128 pixels per task to 2048 pixels per task, resulted in an increase of the total execution time of about 0.7 percent. Hence the effect of task size on the total execution time is negligible. It was observed that the total amount of computation time remained constant as the total work remained constant. Figure 8 shows the effect of task size on READ and WRITE times. For a task size of 128 pixels per

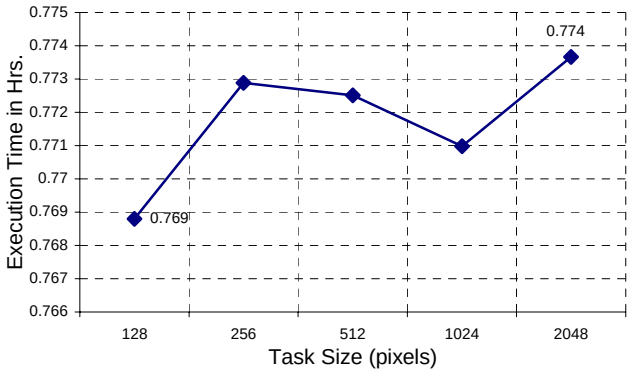

Figure 7: Execution time versus task size for 64 data cubes running on 64 processes

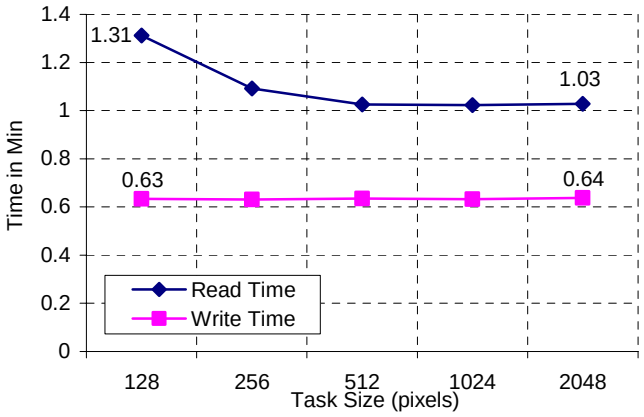

Figure 8: Read/Write time versus task size for 64 data cubes running on 64 processes

task, blocks of 128 pixels were used to perform READ and WRITE, whereas for task sizes greater than or equal to 256

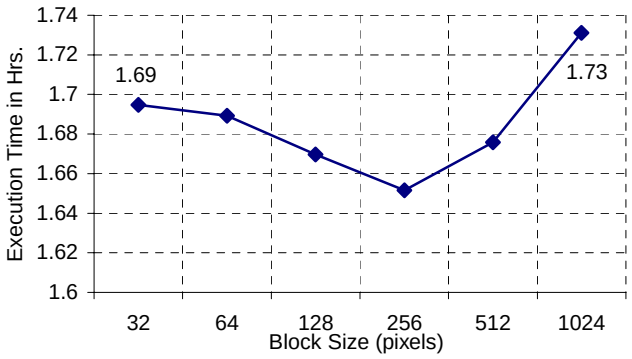

Figure 9: Execution time versus block size for 64 data cubes running on 64 processors

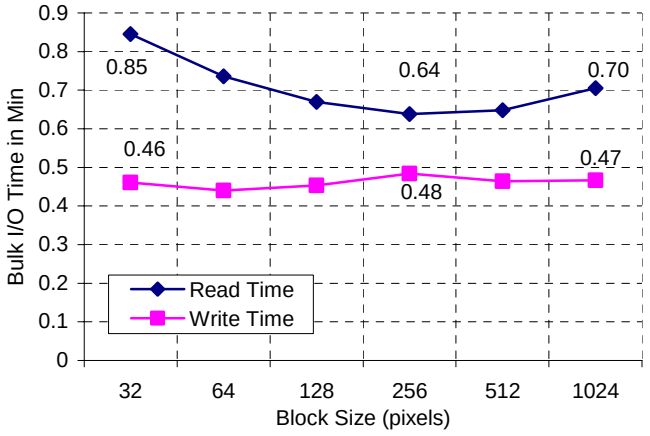

Figure 10: Bulk I/O time versus block size for 64 data cubes running on 64 processors

pixels per task, blocks of 256 pixels were used.

In the third experiment, we investigated the effect of the bandwidth of the parallel file system to determine the optimum number of pixels which should be used for performing I/O to obtain the maximum speedup. In this experiment we kept the task size to a constant value of 1024 pixels per task while increasing the block size to perform bulk I/O. Block sizes of 32, 64, 128, 256, 512, 1024 pixels were used. Figure 9 shows the total execution time versus block size in pixels. It can be seen that an optimal value of a block size of 256 pixels results in the least execution time. Figure 10 shows the effect of bulk I/O time versus block size in pixels. An optimal value of a block size of 256 pixels is obtained for READ while the value of block size has negligible effect on WRITE.

*B. **Reliability***

In this section, we evaluated the reliability of the framework in case of worker failures and their impact on the performance. A task assigned to a worker by the master may not complete on time due to a worker node or worker process failure, respectively. In our framework an estimated execution time is computed by taking the average of a certain number of previous task execution times multiplied by an over-estimation factor as described in section 2. A task which has not been completed within its estimated execution time is considered lost by the master and will be reassigned to another worker requesting tasks. The worker which was initially assigned the lost task is considered faulty and is removed from the list of available workers. It is assumed that an operational system will have a safety margin to ensure the availability of the minimum required number of worker processes at all times. To investigate the capability of the framework in case of worker failures, artificially induced faults are introduced in the system. Worker processes are shut down according to an exponential distribution after they are assigned tasks. An example of the worker failure distribution during the course of one of the experiments is shown in figure 11. In our experiments, we are assuming that the required number of worker processes is 32 with a safety margin of 100 percent. Hence 64 worker processes are shut down until only half of the initial processes are available. We verified that the master can successfully identify failed workers and reassign tasks to available worker processes. Figure 12 shows the speedup for a

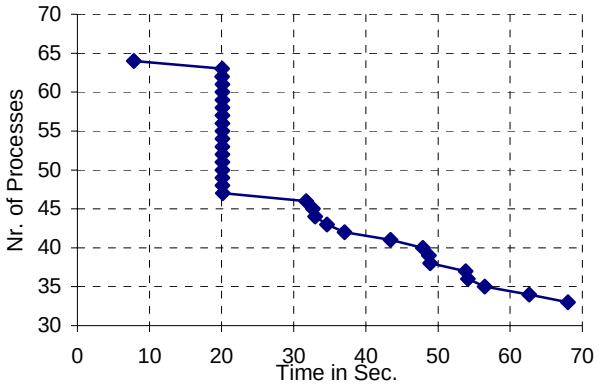

Figure 11: Example sequence showing the time when 32 processes terminated

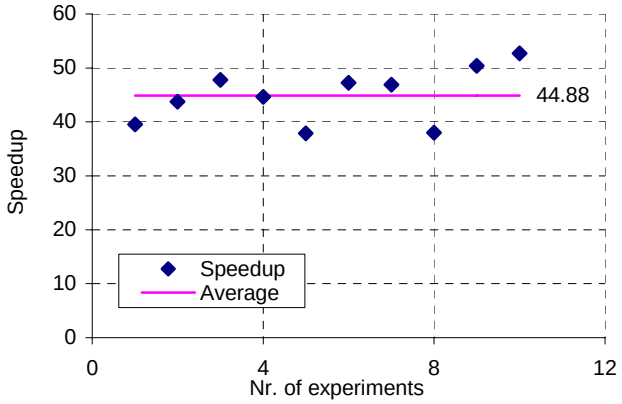

Figure 12: Speedup Comparison for consecutive runs with 32 processes

series of experiments where the speedup varied from about 38 to 53. A mean value of 44.88 is obtained for the speedup as shown in figure 12. This value may be compared to the speedup of 54.56 shown in figure 5 without worker failures.

VI. Conclusions and Future Work

In this paper we have proposed a highly robust and reliable framework for the distributed real-time processing of satellite data for a GIFTS ground system. Furthermore we presented an architectural model for providing performance, reliability, and scalability of candidate hardware and software for such a framework. It has been shown that considerable performance can be gained for GIFTS science algorithms without sacrificing the reliability and high availability requirements for the operational system. Future work involves the processing of GIFTS science algorithm pipeline using real thermal vacuum data. Depending on the flight of the GIFTS instrument, the framework prototype as shown in this paper needs to be transitioned into an operational environment requiring data to be processed in near real-time. This requires further scalability studies in terms of number of processors needed to keep up with the GIFTS data rate.

Acknowledgment

The authors would like to acknowledge Robert Knuteson, Ray Garcia, Graeme Martin, and Maciek Smuga-Otto at the

Space Science and Engineering Center (SSEC), University of Wisconsin (UW) for providing valuable insight into their science algorithms and for providing source code for GIFTS science algorithms. Support for this work was provided by NOAA Instrument Processing Engineering Support (IPES) Task 1.

REFERENCES

[1] R. Knuteson, F. Best, R. Dedecker, R. Garcia, S. Limaye, E. Olson, H. Revercomb, D. Tobin, Level 0-1 Algorithm Description for the Geosynchronous Imaging Fourier Transform Spectrometer, 20th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, 84th Annual American Meteorological Society Meeting, Seattle, January 2004.

[2] R. K. Garcia, M. J. Smuga-Otto, Design Studies for Real-Time Distributed Processing of Infrared Hyperspectral Image data, 20th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, 84th Annual American Meteorological Society Meeting, Seattle, January 2004.

[3] R. ,K. Garcia, M. J. Smuga-Otto, Component-oriented Design Studies for Efficient Processing of Hyperspectral Infrared Imager data, The International Society of Optical Engineering Conference, Denver, CO, August 2004.

[4] H. Revercomb, V. Walden, D. Tobin, J. Anderson, F. Best, N. Ciganovich, R. Dedecker, T. Dirkx, S. Ellington, R. Garcia, R. Herbsleb, R. Knuteson, D. Laporte, D. McRae, M. Werner, Recent Results from Two New Aircraft-Based Fourier Transform Interferometers: The Scanning High-Resolution Interferometer Sounder and the NPOESS Atmospheric Sounder Testbed Interferometer, 20th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, 84th Annual American Meteorological Society Meeting, Seattle, January 2004.

[5] S. Tehranian, Y. Zhao, T. Harvey, A. Swaroop, K. Mckenzie, "*A Robust Framework for Real-Time Distributed Processing of Satellite Data*", Journal of Parallel and Distributed Computing 66 (2006) 403-418.

[6] M. Ronstrom, L. Thalmann, MySQL Cluster Architecture Overview, High Availability Features of MySQL Cluster, MySQL Technical White Paper, April 2004.

[7] J. Lopez, D. O'Hallaron, T. Tu, Big Wins with Small Application-Aware Caches, SC2004 High Performance Computing, Networking, Storage Conference, Pittsburgh, November 2004.

[8] Data Redistribution and Remote Method Invocation in Parallel Component Architectures, 19th IEEE International Parallel and Distributed Processing Symposium , Volume 1, Nr. 1, 2005, pp. 40-54.

[9] Selecting a parallel file system, http://www.lustre.org/docs/selecting-a-cfs.pdf

[10] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. DonGarra, MPI-The Complete Reference, Volume 1, The MPI Core, The MIT Press, Cambridge, Massachusetts, London, England, 1999.

[11] W. Gropp, S. Huss-Lederman, A. Lumsdaine, and E. Lusk, MPI-The Complete Reference, Volume 2, The MPI Extensions, The MIT Press, Cambridge, Massachusetts, London, England, 1999.

[12] Open MPI: Open Source High Performance Computing: http://www.open-mpi.org/papers/heteropar-2006/heteropar-2006-paper.pdf

[13] MPICH2: Users Guide http:// www. unix.mcs.anl.gov / mpi / mpich2 / downloads / mpich2-doc-user.pdf

[14] LAM/MPI Parallel Computing, http://www.lam-mpi.org.