Vector nonlinear time-series analysis of gamma-ray burst datasets on heterogeneous clusters

Ioana Banicescu[a,b,*], Ricolindo L. Cariño[b], Jane L. Harvill[b,c] and John Patrick Lestrade[d]
[a]*Department of Computer Science and Engineering, PO Box 9637, Mississippi State University, Mississippi State MS 39762, USA*
E-mail: ioana@cse.msstate.edu
[b]*Center for Computational Sciences ERC, Mississippi State University, PO Box 9627, Mississippi State University, Mississippi State MS 39762, USA*
E-mail: rlc@erc.msstate.edu
[c]*Department of Mathematics and Statistics, Mississippi State University, PO Box MA, Mississippi State University, Mississippi State MS 39762, USA*
E-mail: harvill@math.msstate.edu
[d]*Department of Physics and Astronomy, Mississippi State University, PO Box 5167, Mississippi State University, Mississippi State MS 39762, USA*
E-mail: lestrade@ra.msstate.edu

Abstract. The simultaneous analysis of a number of related datasets using a single statistical model is an important problem in statistical computing. A parameterized statistical model is to be fitted on multiple datasets and tested for goodness of fit within a fixed analytical framework. Definitive conclusions are hopefully achieved by analyzing the datasets together. This paper proposes a strategy for the efficient execution of this type of analysis on heterogeneous clusters. Based on partitioning processors into groups for efficient communications and a dynamic loop scheduling approach for load balancing, the strategy addresses the variability of the computational loads of the datasets, as well as the unpredictable irregularities of the cluster environment. Results from preliminary tests of using this strategy to fit gamma-ray burst time profiles with vector functional coefficient autoregressive models on 64 processors of a general purpose Linux cluster demonstrate the effectiveness of the strategy.

Keywords: Heterogeneous computing, dynamic load balancing, data analysis

1. Motivation

One of the difficult problems in data analysis involves multivariate time series. A vector time series is a set of observations of multiple related phenomena across time. The mathematical underpinnings of the statistical analysis of time series incorporate the correlation across time and between series – properties that complicate statistical theory. This is especially true for nonlinear models, where mathematical theory is difficult. The calculations involved in the analysis of nonlinear vector time series are highly computationally intensive, requiring several minutes to a few hours on a single processor, even for a series with length of a few thousand observations.

The very long computation time associated with the analysis of multivariate time series has tempted statisticians to tackle smaller problems in the interest of "computational expediency" and productivity. For example, the number of replications in a simulation may be re-

[*]Corresponding author.

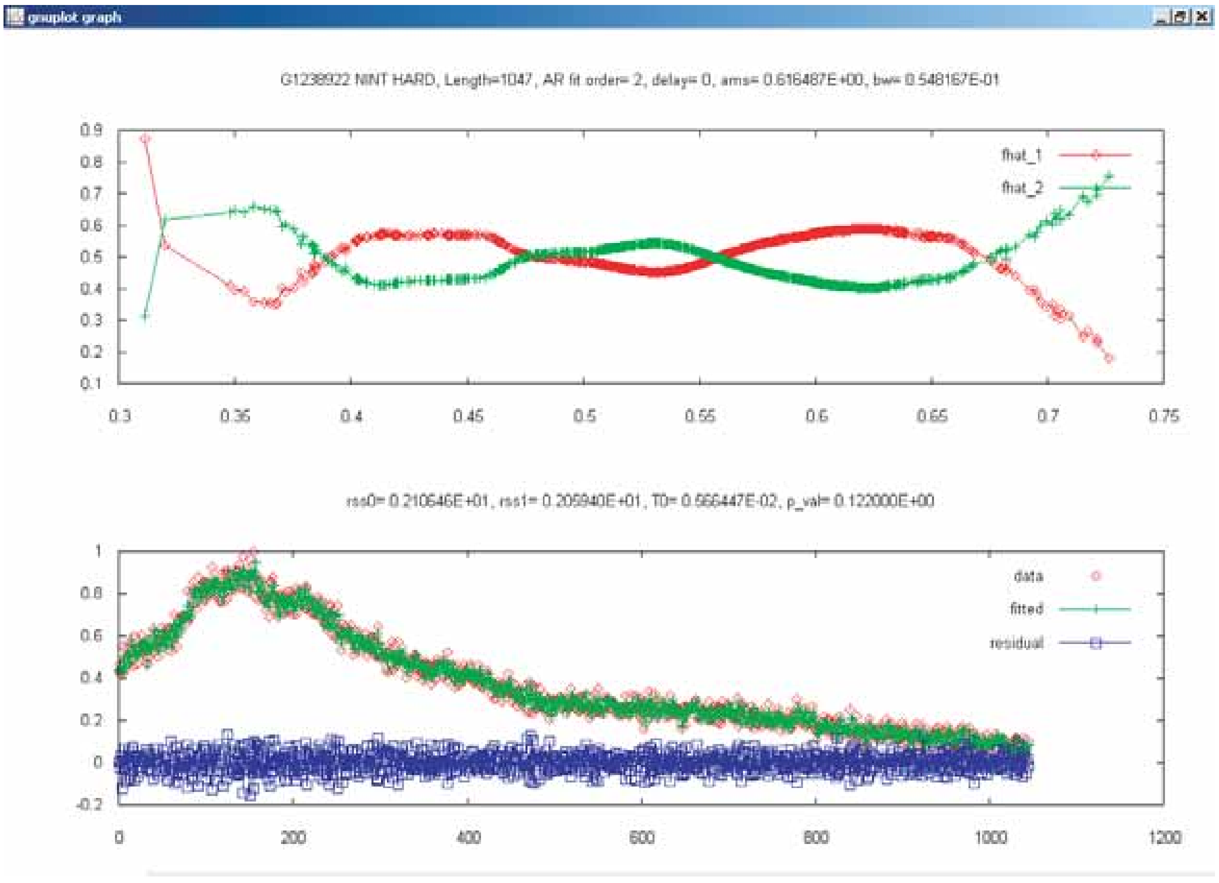


Fig. 1. Sample plots from the results of a nonlinear vector time-series analysis of the G1238922 gamma-ray burst dataset.

duced, or very large sample sizes may be excluded from study, for reasons of computational costs. These temptations have been successfully overcome by utilizing parallel processing technology [2,3,19]. The intensive computations in statistical codes typically occur inside parallel loops. The presence of these loops allows the strategy of integrating dynamic loop scheduling techniques into a statistical analysis program for parallelization and for achieving high performance through load balancing.

Applying multivariate nonlinear time series analysis to a large collection of datasets dramatically increases the computational cost to days or even weeks. The simultaneous analysis of related datasets is sometimes necessary, for example, to uncover underlying structure, to extract important variables, to test underlying assumptions, and to develop compact models, for the phenomena described by the datasets. In addition to the parallel loops inherent in the implementation of the vector nonlinear time series analysis technique, the multiplicity of the datasets to be analyzed defines another level of concurrency. These two levels of concurrency occurring within a single application presents an interesting load balancing problem, especially when executing in a heterogeneous cluster environment.

This paper proposes a load balancing strategy that exploits the large number of processors available in a cluster environment for the simultaneous analysis of multiple datasets. The context in which the strategy was developed is as follows. The datasets are gamma-ray burst (GRB) time profiles recorded by the NASA Burst And Transient Source Experiment (BATSE) [8]. The datasets are to be fitted with Vector Functional Coefficient Autoregressive (VFCAR) models [15,16] under various analysis scenarios. The analysis aims to expose discontinuities or evolution in complex time profiles, and to help classify the GRB profiles into groups. The analysis was conducted on the EMPIRE cluster of the Mississippi State University Engineering Research Center. Although developed in this context, the implementation of the load balancing strategy is orthogonal to the code for the statistical analysis. This implementation can be used in other applications that involve computationally intensive data analysis.

The remainder of this paper is organized as follows: Section 2 provides an overview of gamma-ray bursts, the statistical analysis of GRB time profiles using VFCAR models, and the programming environment used. Section 3 describes a load balancing strategy which is based on the partitioning of processors into small groups appropriate for the level of concurrency in the statistical analysis procedure, and employing a dynamic loop scheduling approach to balance group loads. Section 4 presents results of preliminary tests of the strategy on a general purpose Linux cluster. Section 5 concludes with previews of ongoing efforts to further improve the strategy.

2. Background

This section provides brief descriptions of gamma-ray bursts (GRBs), the statistical analysis of GRB time profiles using VFCAR models, and the programming environment used. This representative application exhibits two opportunities for parallelism: the concurrent analysis of datasets, and the computationally intensive parallel loops within the analysis procedure. Load balancing is a crucial issue in this application due to the disparity in the dataset sizes, the potential non-uniformity of the loop iteration execution times, and the unpredictable runtime irregularities in a heterogeneous cluster environment.

2.1. Gamma-ray bursts

Early astronomical observations were restricted to the visible range of the electromagnetic spectrum. However, the past century has seen the invention and use of new detectors and telescopes that are capable of observing the radio, infrared, ultraviolet, x-ray, and gamma-ray regions. Without this wider spectral vision, we might never have discovered a type of exploding star whose brightness surpasses the well-known supernova. Since 98% of their power is released in the form of gamma radiation, they are called gamma-ray bursts (GRB). There is approximately one burst each day somewhere in the observable universe.

Discovered in the mid-1960's by US military satellites, the causes of GRB have risen to the top of the list of perplexing astrophysical questions. Their incredible distances, known to be far outside our own Milky Way, and their short lifetimes (average brightening duration of 20 seconds) have hampered investigators. For example, with more than 3,000 recorded, we have reliable distance measurements for only a few dozen burst sources.

A complete understanding of the underlying phenomena will require a concerted effort from many different scientific disciplines. Such an understanding could provide us with important, new information about the structure and evolution of the early universe. In addition, this is an opportunity to learn more about the behavior of matter and energy under extreme conditions of pressure, temperature, and magnetic field which are impossible to create in the laboratory.

In practice, when a burst occurs, a gamma-ray detector onboard an orbiting satellite measures a sudden and intense increase in the photon count rate above the normal, relatively constant, gamma-ray background. This flood of radiation from an explosion that lies perhaps billions of light years away can last for only a fraction of a second or for several minutes. Some simply brighten, then dim, but many show multiple peaks in intensity that often overlap, resulting in highly complex profiles. Just as quickly, it is over and our record of the event is most often limited to the energies and the times of arrival of the gamma-ray photons.

The typical time profile shows features, such as sharp jumps and other hints of intermittency, that cannot be modeled by linear time series analyses (Fig. 1, bottom plot). These brightness peaks are a reflection of the energetics of the explosion and the environment. A likely possibility is that many of the peaks are the result of shock interactions between an expanding fireball and the circumstellar material. A nonlinear vector time-series analysis as outlined below could shed light on both the nature of the burster environment and the dominating physical processes.

2.2. Multivariate nonlinear time series

Although complicated in theory, vector nonlinear time series are especially useful for describing complex nonlinear dynamic structures that exists in many time-dependent multivariate series. Let $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{k,t})'$ denote the vector time series at time $t = 1, 2, \ldots, T$. Then the vector functional coefficient autoregressive model of order p (VFCAR(p)) is defined as

$$\boldsymbol{Y}_t = \boldsymbol{f}^{(0)}(\boldsymbol{Z}_t) + \sum_{j=1}^{p} \boldsymbol{f}^{(j)}(\boldsymbol{Z}_t)\boldsymbol{Y}_{t-j} + \boldsymbol{\varepsilon}_t, \qquad t = p+1, \ldots, T, \tag{1}$$

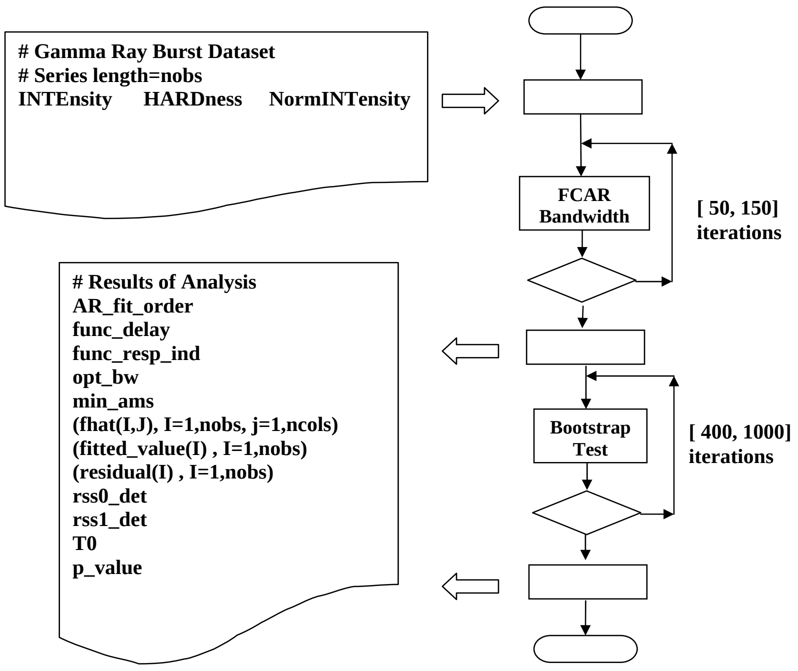


Fig. 2. Sample structure of a GRB dataset, flowchart of VFCAR analysis and outputs of the analysis.

where $\boldsymbol{f}^{(j)}$, $j = 0, \ldots, p$ are $k \times k$ matrices whose elements are real-valued measurable functions that change as a function of the (possible vector-valued) $\boldsymbol{Z}_t$, and which have continuous second-order derivatives. The error terms ε_t in Eq. (1) are such that for each i, the series $\{\varepsilon_{i,t}\}_{t=1}^{T}$ is a white noise sequence, independent of $\{\boldsymbol{Y}_t\}_{t=1}^{T}$. However contemporaneous cross-correlation may exist between $\{\varepsilon_{i,t}\}$ and $\{\varepsilon_{j,t}\}$, $i \neq j$. The primary motivation for studying this model is that specific choices for the elements of the $\boldsymbol{f}^{(j)}$ yield parametric models.

The VFCAR(p) model may be considered a hybrid of parametric and non-parametric models since the autoregressive structure is assumed, but there is little or no information about the form of the elements of the $\boldsymbol{f}^{(j)}$. As such, estimation of the parameters of the VFCAR(p) model is done nonparametrically via local regression. Simultaneous estimation of the elements of the $\boldsymbol{f}^{(j)}$ provides improved statistical efficiency when the error terms have positive cross-correlation. In the process of fitting the model Eq. (1), modified multifold cross-validation is used to determine an optimal bandwidth and value for p by finding the pair of values that minimize the accumulated prediction error. This multi-stage procedure requires an immense number of arithmetic operations on a univariate series. That number increases exponentially for multivariate series.

The VFCAR model is also used in statistical tests of model misspecification. However, the null distribution is unknown, and so a procedure that fits the model to a large number of bootstrapped realizations of the series under the null model is used to obtain a numerical p value for the test. Specifics for fitting the model and the testing procedure are found in [15]. Forecasting can be accomplished using the VFCAR model either recursively or via bootstrap as in [16]. The mathematical complexity of the statistical theory of the estimators, testing procedures, and forecasts using the VFCAR model are highly complicated, requiring arithmetically intensive computations.

Figure 2 illustrates the major steps in fitting the VFCAR(p) model to a GRB time profile. The bulk of the computations occur in the determination of the optimal bandwidth (FCAR bandwidth) and in the statistical test of model misspecification (Bootstrap Test). The analysis can be done on a single processor, but a dataset with a few thousand observations may require up to two days. The bandwidth computations involve a parallel loop with 50–150 iterations, while the bootstrap test involves a parallel loop with iteration count of 400–1000 as decided by the statistician. Thus, the degree of loop-level concurrency in the analysis of single dataset for the bandwidth computations is small compared to the few thousands of processors available in a typical supercluster. The outputs of the VFCAR analysis of a GRB time profile are used to produce visualization plots like Fig. 1 for interpretation by the physicist.

2.3. Heterogeneous clusters

A very popular high performance computing platform is the cluster of Linux workstations, in view of the easily available off-the-shelf hardware components and clustering software. Such clusters are usually built over time, and thus tend to be made up of heterogeneous processors, memory, and networking components. An example is the EMPIRE cluster of the Mississippi State University Engineering Research Center (MSU ERC). The cluster was built in four phases from February 2001 through May 2002. It has a total of 1038 Pentium III (1.0 GHz and 1.266 GHz) processors, and runs the Red Hat Linux operating system. Two processors reside in a node, 32 nodes are connected via 100 Mbps Ethernet switch to comprise a rack, and the racks are connected via Gigabit Ethernet. The cluster queuing system (PBS) attempts to assign compute nodes from a single rack to a job, but this is not guaranteed. Jobs also contend for network bandwidth and disk I/O resources. For all practical purposes, unpredictable system-induced load imbalance will affect most parallel applications running on this cluster, with varying degrees of severity. Therefore, applications running on the cluster must incorporate dynamic load balancing to achieve highest possible efficiency.

3. Load balancing strategy

The proposed load balancing strategy for the simultaneous statistical analysis of multiple datasets on a heterogeneous cluster is driven by considerations arising from the characteristics of the analysis program and the computational platform. The number of datasets to be analyzed and the sizes of parallel loops in the statistical procedure are known before execution, but information about processors and communication distances are available only after cluster resources have been committed to the job. Thus, a dynamic load balancing strategy is necessary for high performance. A simple master-slave strategy where a single processor is assigned to analyze one dataset at a time may seem appealing, except that it potentially achieves low processor utilization due to the disparate sizes of the datasets. A dataset may require a few seconds or over one day of analysis time on a single processor. The processors assigned to the big datasets will finish very long after the other processors have finished all of the smaller datasets. Likewise, the strategy of having all the processors cooperatively analyzing one dataset at a time restricts the maximum number of processors that can be utilized for the job. This maximum is determined by the amount of parallelism in the analysis procedure, which may be a small number relative to the total number of processors available for the job. Thus, an alternative load balancing strategy is needed which takes advantage of a large number of processors available in a cluster environment, and keeps all the processors busy doing useful work. Such a strategy described in the rest of this section.

The proposed strategy takes two preparatory steps before any statistical analysis takes place: (1) the partitioning of the processors assigned to the job into groups, and (2) the initial distribution of datasets to the processor groups. After the completion of these two steps, the strategy employs a dynamic loop scheduling approach to schedule the analysis of datasets in each group, for load balancing group loads. If the analysis procedure involves parallel loops, dynamic loop scheduling can also be used within a group for balancing the loads of processors of the group while executing the parallel loops. Datasets are migrated between groups as necessary to address the load imbalance induced by the initial distribution of datasets among processor groups or induced by unpredictable system irregularities.

3.1. Partitioning of processors into groups

In a cluster environment, it is intuitively obvious that the communications between processors assigned to a job will be more efficient if the processors reside in a single rack instead of being spread across several racks. For example in the EMPIRE cluster, a message between two processors p1 and p2 located in the same rack requires at most two hops (p1 $\rightarrow$ Mbps switch $\rightarrow$ p2), as opposed to four hops for a message between two processors p3 and p4 located on different racks (p3 $\rightarrow$ Mbps switch $\rightarrow$ Gbps switch $\rightarrow$ Mbps switch $\rightarrow$ p4). Thus, the cluster scheduler attempts to assign nodes from a single rack to a job for efficient communications. Despite these attempts, fragmentation of processors across several racks occurs with a high probability, especially for jobs that request large numbers of processors.

The fragmentation of processors assigned to a job across a few physical racks is actually useful in the context of simultaneous analysis of multiple datasets. The proposed load balancing strategy considers the group of processors residing in a rack as a single computational unit, cooperatively analyzing one dataset at a time. A very large group can broken into two or three smaller groups to fit the degree of concurrency in the analysis procedure. Tiny groups can be merged to avoid the long analysis time in the event that a tiny group is assigned a very large dataset. The resulting number of groups determine the number of datasets that can be analyzed simultaneously. This setup exploits the efficient communications among processors residing in a single rack, and ensures that all processors assigned to the job actively participate in solving the big problem.

The typical naming convention for the processors in Linux clusters make it easy to determine which processors belong to the same physical rack. In the EMPIRE cluster for example, the processors are named `Empire-<rn>-<pn>`, where `<rn>` is the rack number and `<pn>` is the processor number. Thus, processors with the same `<rn>` belong to the same rack. In MPI, the routine MPI GETHOSTNAME() may be invoked with appropriate arguments to determine name of the processor.

3.2. *Initial distribution of datasets to processor groups*

After the processors assigned to the analysis job are identified and the processor groupings are established, the datasets are retrieved from disk for distribution to the processor groups. Keeping the datasets in memory offers some advantages over "on demand" retrieval of datasets from disk. In the case of GRB analysis, a dataset with n observations requires $(k + 1) \times n$ real memory locations, where k is the number of components of an observation. The preliminary analysis that were conducted in this work involved 555 datasets with $k = 1$ and $46 \leqslant n \leqslant 9037$. Only 5 datasets have $n >$ 5000 and 55 datasets have $n >$ 1500. Thus, the memory requirement for all the datasets is relatively small compared to the total amount of memory available on the cluster. In addition, moving a dataset from one processor to another is less expensive than retrieving the dataset from disk.

An important issue that has a major effect on group load balance is the initial partitioning of the datasets among the processor groups. It may be possible to correlate the analysis time of a dataset to the number of observations, depending on the nature of the computations. In such a case, the correlation can provide useful guidance in the initial load distribution. For example, if the analysis mainly uses matrix-vector multiplications, then the time complexity may be $O(n^2)$. Initial group loads can then be assigned based on quantities derived from the correlation, with the expectation that further load balancing will no longer be necessary. However, this correlation has limited usefulness in a dynamic environment, where the effective speed of a processor is affected by factors such as operating system daemons waking up to perform short monitoring tasks, or delays in I/O operations due to disk access contention. Also, conditional statements embedded in the analysis procedure may trigger an execution sequence whose time complexity is not captured by the correlation. Therefore, in the interest of generality, the load balancing strategy should utilize dynamically collected information instead of an assumed static correlation between the size and the computational load of a dataset. Nevertheless, the size of a dataset is an important input to the load balancing decisions.

Given only the number of observations in each dataset and the sizes of the processor groups, the proposed strategy uses the following heuristic to initially distribute the datasets among the groups. Let D denote the number of datasets, G the number of processor groups, n_i the number of observations in dataset i, and s_j the size of group j. Therefore, the total number of observations is $W = \sum_{i=1}^{D} n_i$ and the number of processors in the groups is $P = \sum_{j=1}^{G} s_j$. Then, the datasets are distributed such that group j has a total of approximately $s_j \times W/P$ observations. Thus, the number of observation in a group is proportional to the number of processors in the group. To avoid the situation of all the big datasets being lumped together into one group, the distribution procedure is as follows. The datasets are first sorted according to decreasing n_i. Then, for each dataset in sorted order, the group which is farthest from its quota of observations is identified and the processor with the minimum number of observations in that group will store the dataset. This ensures that the big datasets are effectively scattered among the groups, and that the processors in a group store comparable numbers of observations.

3.3. *Dynamic loop scheduling*

The proportionality of the size of a processor group to the total number of observations stored by the group is not a guarantee for good load balance among groups. This is because the number of observations in a dataset may not be an appropriate measure of the computational load of that dataset. Even if the load is known from the size of the dataset, the dynamic nature of a cluster environment induces other types of load imbalance that must be addressed during the actual analysis of datasets.

The proposed strategy employs a dynamic loop scheduling approach for load balancing group loads. Two of the authors have extensive experience with dynamic loop scheduling techniques [4,10–12], and the implementation of these techniques on both distributed and shared memory environments [9], as well as the integration of these techniques into scientific applications [2,3,5–7,13] and systems [1]. In loop scheduling, a parallel loop with N iterations is to be executed on P processors. Chunks of iterations are assigned to processors with the objective of minimizing the loop completion time. The sizes of chunks are determined according to a loop scheduling technique. Mapped to the present context of simultaneous analysis of multiple datasets, the N loop iterations correspond to the W observations, and the P processors correspond to the G groups, each group being a single computational unit. A chunk of iterations is essentially a fraction f of the total N iterations; this chunk corresponds to a collection of datasets whose cumulative size is approximately $f \times W$ observations. Using these correspondences, the dynamic loop scheduling techniques are applicable in the present context, with the possible exception of techniques like adaptive factoring [4] that require measurement of individual iterate execution times. The correspondence between a single iteration and a single observation point may not be valid because the execution of an iteration can timed, while observations are not analyzed individually, but only collectively in a dataset.

Dynamic loop scheduling can also be employed within a processor group if the analysis procedure involves parallel loops. This is the case in fitting a GRB dataset with VFCAR models. Loop scheduling is used to execute the FCAR Bandwidth loop and the BootStrap Test loop.

3.4. *Processor organization*

Figure 3 illustrates the organization of the processors in the proposed strategy. The analysis job is submitted to a heterogeneous cluster, and the cluster scheduler commits the number of processors requested by the job. One of the processors is designated as the coordinator C, which is responsible for: (1) organizing the rest of the processors into groups of workers W and appointing a foreman F in each group; (2) retrieving the datasets from disk and distributing these to the groups; and (3) scheduling the analysis of the datasets by the groups. The scheduling proceeds as follows.

The foreman of a group sends to the coordinator a request for information about the datasets which the group should analyze next. When the coordinator receives the request, it consults the loop scheduling technique to determine the upper limit on the number of observations to be scheduled next. If the requesting group has datasets that are not yet analyzed, the coordinator selects the datasets in the group whose combined observations approximate the upper limit. The coordinator responds to the requesting foreman with the selection, and the foreman relays the selection to its group. The processors in group examine the selected datasets one at a time. The owner of a selected dataset first broadcasts the dataset to the rest of the group, then the group collectively invokes the analysis routine. The foreman participates in the analysis, and the group can invoke dynamic loop scheduling if the analysis involves parallel loops.

If the requesting group has no remaining datasets, the coordinator identifies a source group and selects from that group the datasets to be migrated to the requesting group. This selection is sent to the requesting foreman, who relays the information to its group. Based on this information, the foreman will send a message to the owners of the datasets to begin the transfer operations, then all processors in the requesting group post receive operations. Once the transfers are complete, the datasets are examined as described previously.

4. Preliminary tests

The proposed strategy for the simultaneous analysis of multiple datasets on a heterogeneous cluster is currently implemented in Fortran 90 and utilizes the Message Passing Interface (MPI) library for communications. The choice of the programming language was influenced largely by the earlier decision of the statistician to develop the VFCAR analysis procedures in Fortran 90. The executables were produced by the Portland Group Fortran compiler, linking with the MPICH implementation of the MPI library.

Preliminary tests of the strategy in analyzing 555 datasets of GRB time profiles (total of 398238 observations) by the VFCAR model were conducted on 64 processors of the MSU ERC EMPIRE cluster. The results that follow are for a particular run in which the processors were spread across racks 8, 10, 14, 15 and 16 of EMPIRE, with the racks contributing 20, 16, 16, 8 and 4 processors, respectively. Rack 8 has 1.0 GHz processors, while the rest of the racks have

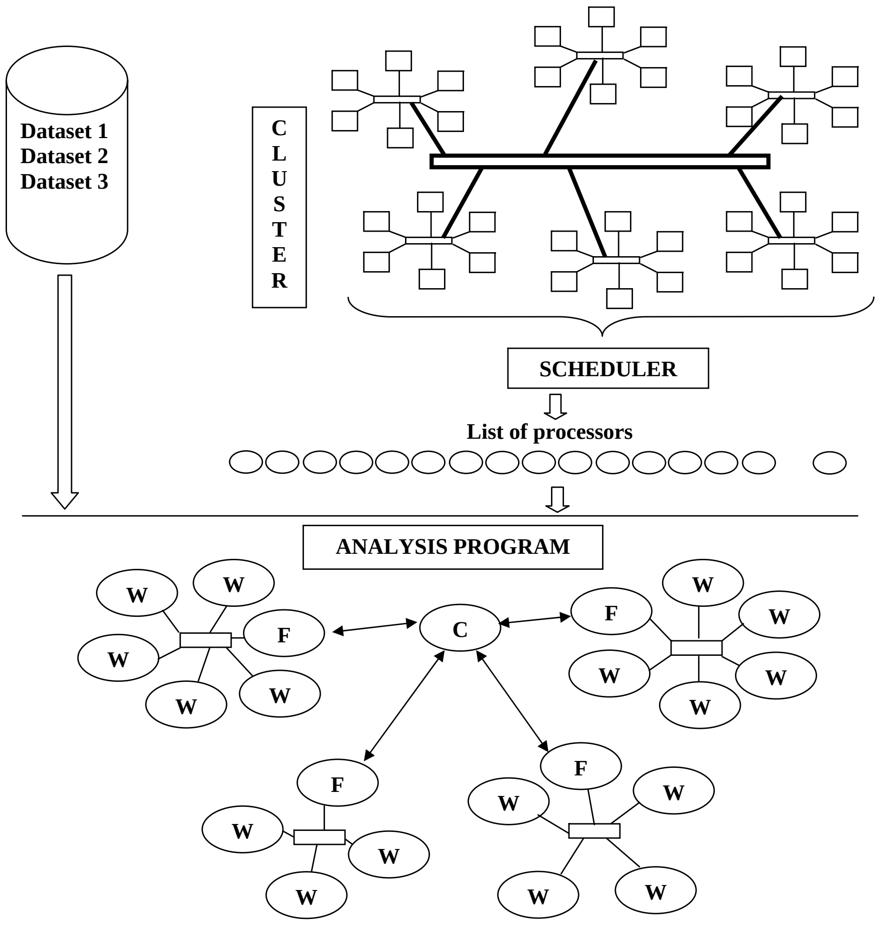
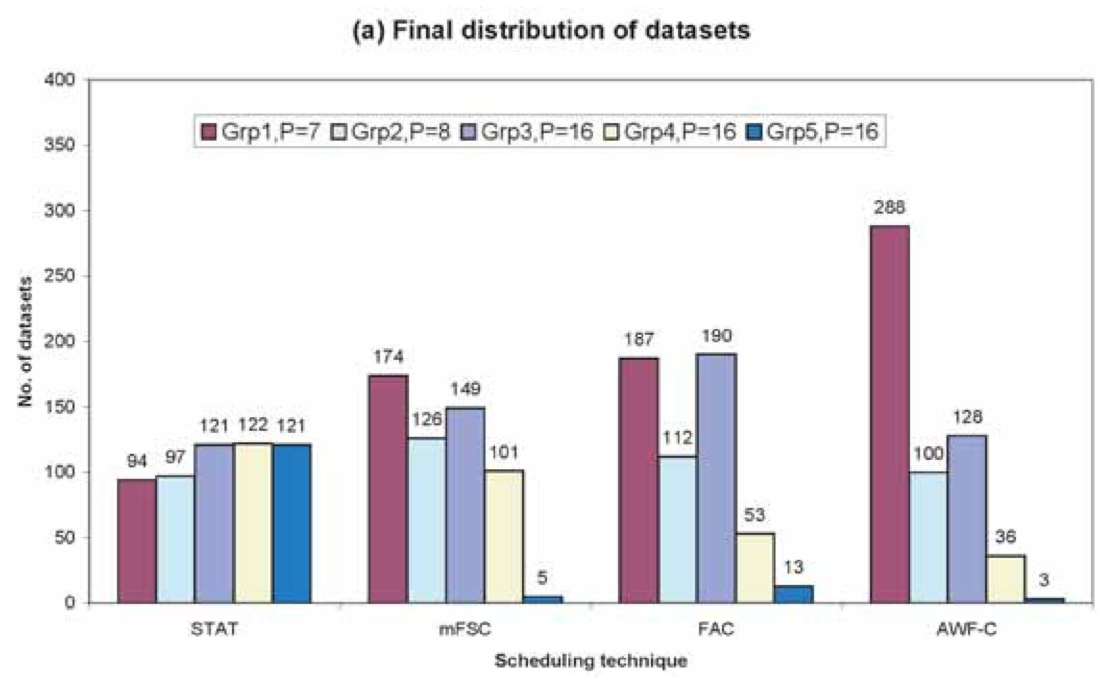


Fig. 3. Organization of processors for the simultaneous analysis of multiple datasets on a cluster: C = coordinator, W = worker, F = group foreman (also a worker).

1.266 GHz processors. The coordinator C formed five groups: Grp 1 with 7 processors split between racks 8 and 16; Grp 2 with 8 processor in rack 15; Grp 3 with 16 processor in rack 14; Grp 4 with 16 processor in rack 10; and Grp 5 with 16 processor in rack 8. The coordinator resided in rack 16. Aside from the imbalance in computational powers of groups, the cluster was also running other jobs along with the tests and the analysis has to write some files. Thus, the load imbalance induced by network contention was unpredictable.

The tests were aimed primarily gain preliminary insights into the characteristics of the strategy in order to identify areas for improvement, and secondarily to compare the performance gains achieved by the strategy with various loop scheduling techniques. For the second objective, only the following techniques were attempted for the sake of brevity in the testing process: no load balancing after the initial distribution of datasets (STAT), factoring (FAC) [14], fixed size chunks based on the number of chunks generated by the factoring (mFSC), and a variant of adaptive weighted factoring (AWF-C) [10,11]. The chunk size in mFSC is $W/$(no. of chunks in FAC); thus, mFSC has the same scheduling cost as FAC. Results for other techniques will be reported in the future. The same scheduling technique was used to balance the load among processor groups, as well as within a group to execute the FCAR Bandwidth loop and the Bootstrap Test loop.

The tests were conducted as a single parallel job, so the same set of processors was used by all the techniques. The performance metric is the *Cost*=$P \times T_P$, where T_P is the parallel time. The cost gives the aggregate time devoted by all of processors (including the coordinator C) to the run. To assess the benefit of dynamic load balancing achieved by a given technique, the percent cost improvement *%CI* is computed as *%CI*=$100\times$(*Cost*$_{STAT}$-*Cost*$_{DLB}$)/ *Cost*$_{STAT}$, where *Cost*$_{STAT}$ is the cost of the analysis without dynamic



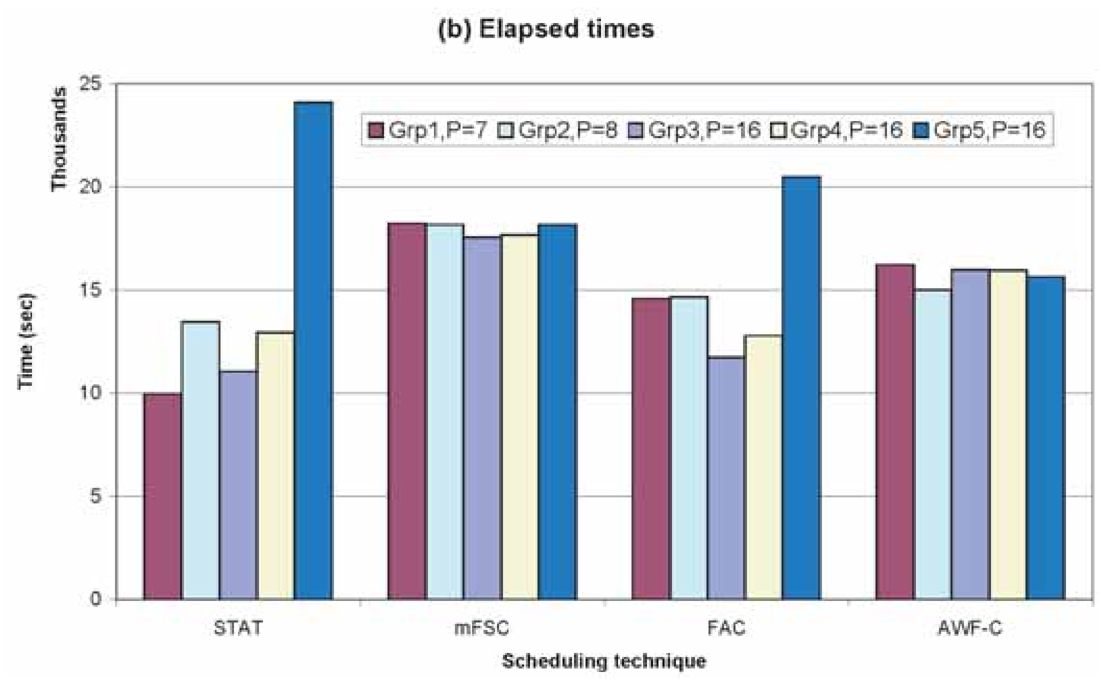


Fig. 4. Comparison of (a) the distribution of datasets among groups after load balancing, and (b) the group elapsed times.

load balancing and $Cost_{DLB}$ is the cost using the dynamic load balancing technique "DLB", both costs being measured on the same set of processors.

Figure 4(a) and (b) summarize the performance of the strategy using the STAT, mFSC, FAC and AWF-C load balancing techniques. The processor groups have different sizes, Grp 5 has 1.0 GHz processors, Grps 2–4 have 1.266 GHz processors and Grp 1 has both types of processors. The initial distribution of datasets is the same as the final distribution for STAT. Except for

STAT, datasets migrated from Grps 4 and 5 to Grps 1–3. The cost improvements *%CI* over STAT of mFSC, FAC and AWF-C are 24.3, 14.9 and 32.7, respectively.

The preliminary results support the following preliminary conclusions regarding the strategy. The total number of observations stored in a processor group is not a good measure of the work load for the datasets owned by the group. The procedure employed to initially distribute the datasets among the processor groups has limited effectiveness as a static load balancing heuristic, especially if the processors are not homogeneous, since the heuristic uses information about dataset sizes and group sizes only. The load imbalance factors induced by the initial dataset distribution, the differences in dataset analysis times and processor group sizes, are corrected, to varying levels of effectiveness, by using dynamic loop scheduling techniques for dataset redistribution during runtime. Of the techniques used in the experiment, the AWF-C achieved the highest cost improvement over STAT.

5. Concluding remarks

This paper describes a load balancing strategy for the statistical computing problem of the simultaneous analysis of multiple datasets, on a cluster environment. The problem presents some interesting load balancing challenges, including the limited degree of concurrency in the analysis procedure, large disparities in the computational loads of datasets, and unpredictable irregularities in a cluster environment. The strategy exploits the fragmentation of the processors assigned to the job across several physical racks, by organizing the processors residing in a rack into a processor group – a single computational unit cooperatively executing the analysis procedure. This organization benefits from improved communication efficiency, and at the same time addresses the limited degree of concurrency inherent in the analysis procedure. The organization also allows multiple instantiations of the procedure, thereby enabling the utilization of a large number of processors for solving the statistical computing problem. The strategy also employs a dynamic loop scheduling approach to balance the loads of processor groups. Such an approach is capable of addressing load imbalance factors that may be induced by the heuristic used during the initial distribution of datasets among the processor groups, or by the disparities in the computational loads of datasets, or by unpredictable irregularities in a cluster environment. Dynamic loop scheduling can also be utilized by a processor group if the analysis procedure involves parallel loops.

Results of preliminary tests of the strategy as applied to the analysis of gamma-ray burst time profiles using the vector functional coefficient autoregressive time series model on a heterogeneous Linux cluster highlight the effectiveness of the strategy. The authors will continue the analysis of additional gamma-ray burst time profiles; the scientific results derived through this analysis will be reported elsewhere. Investigations on the the performance of the strategy with loop scheduling techniques that are more sophisticated than those tested in this work are ongoing. The authors also plan to undertake the analysis of other of datasets, such as those from the areas of finance and meteorology, using the strategy.

Acknowledgments

The authors thank the National Science Foundation for partially supporting this work through the following grants: CAREER #9984465, ITR/DMS #0313274, ITR/ACS #0081303, ITR/ACS #0085969, #0132618, and #0082979.

References

[1] M. Balasubramaniam, K. Barker, I. Banicescu, N. Chrisochoides, J.P. Pabico and R.L. Carino, *A Novel Dynamic Load Balancing Library for Cluster Computing*, Proc. 3rd Int. Symposium on Parallel and Distributed Computing, in association with the International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04), IEEE Computer Society Press, 2004, 346–353.

[2] I. Banicescu, R.L. Carino, J.L. Harvill and J.P. Lestrade, Computational Challenges in Vector Functional Coefficient Autoregressive Models, in: *Lecture Notes in Computer Science 3514, Computational Science – ICCS 2005*, V.S. Sunderam, ed., Springer-Verlag, 2005, pp. 237–244.

[3] I. Banicescu, R.L. Carino, J.L. Harvill and J.P. Lestrade, *Simulation of Vector Nonlinear Time Series on Clusters*, Proc. 19th Int. Parallel and Distributed Processing Symposium, IEEE Computer Society Press, 2005, on CDROM.

[4] I. Banicescu and Z. Liu, *Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes*, Proc. High Performance Computing Symposium, 2000, 122–129.

[5] I. Banicescu and R. Lu, *Experiences with fractiling in n-body simulations*, Proc. High Performance Computing Symposium, 1998, 121–126.

[6] I. Banicescu and V. Velusamy, *Load balancing highly irregular computations with the adaptive factoring*, Proc. 16th Int. Parallel and Distributed Processing Symposium, IEEE Computer Society Press, 2002, on CDROM.

[7] I. Banicescu, V. Velusamy and J. Devaprasad, On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring, *Cluster Computing: The Journal of Networks, Software Tools and Applications* **6** (2003), 215–226.

[8] Burst And Transient Source Experiment (BATSE), http://www.batse.msfc.nasa.gov/batse

[9] R.L. Cariño and I. Banicescu, A load balancing tool for distributed parallel loops, *Cluster Computing* **8**(4) (2005), to appear.

[10] R.L. Cariño and I. Banicescu, *Dynamic scheduling parallel loops with variable iterate execution times*, Proc. 16th IEEE Int. Parallel and Distributed Processing Symposium, IEEE Computer Society Press, 2002, on CDROM.

[11] R.L. Cariño and I. Banicescu, Load balancing parallel loops on message-passing systems, in: *Proc. 14th IASTED Int. Conf. on Parallel and Distributed Computing and Systems* S. Akl and T. Gonzales, eds., ACTA Press, 2002, pp. 362–367.

[12] R.L. Carino, I. Banicescu, T. Rauber and G. Runger, *Dynamic loop scheduling on processor groups*, Proc. 17th Int. Conf. on Parallel and Distributed Computing Systems (PDCS'04), 2004, 78–84.

[13] R.L. Cariño, I. Banicescu, R.K. Vadapalli, C.A. Weatherford and J. Zhu, Message-passing parallel adaptive quantum trajectory method, in: *High performance Scientific and Engineering Computing: Hardware/Software Support*, L.T Yang and Y. Pan, eds., Kluwer Academic Publishers, 2004, pp. 127–139.

[14] S. Flynn Hummel, E. Schonberg and L.E. Flynn, Factoring: A method for scheduling parallel loops, *Communications of the ACM* **35** (1992), 90–101.

[15] J. Harvill and B. Ray, Functional coefficient autoregressive models for vector time series, *Computational Statistics and Data Analysis* (2005), To appear.

[16] J. Harvill and B. Ray, A note on multi-step forecasting with functional coefficient autoregressive models, *International Journal of Forecasting* (2005), To appear.

[17] C. Kouveliotou, C.A. Meegan, G.J. Fishman, N.P. Bhat, M.S. Briggs, T.M. Koshut, W. Paciesas and G.N. Pendleton, Identification of two classes of gamma-ray bursts, *Astrophysical Journal* **413** (1993), L101–L104.

[18] J.P. Lestrade, A New Variability Parameter for Gamma-Ray Burst Time Profiles, *Astrophysical Journal Letters* **429** (1994), L5–L8.

[19] J. Racine, Parallel Distributed Kernel Estimation, *Computational Statistics and Data Analysis* **40** (2002), 293–302.