# A DATA ANALYSIS SOFTWARE ARCHITECTURE FOR PARALLEL AND DISTRIBUTED COMPUTATION

D. M. Brockett
BBN Systems & Technologies

## ABSTRACT

Real-time high-volume telemetry data analysts have needs which require access to ever-increasing amounts of data, which must be processed in a seamless and coherent manner. BBN has developed a data analysis software architecture for use in distributed- and parallel-processing environments which is particularly well-suited for telemetry streams. BBN is currently using this software at two Navy sites to do real-time data analysis. The architecture provides data-source management, data-stream fusion, and data extraction all in a modular, scalable framework. Because of the scalable nature of the software, it can easily accommodate high data rates.

Keywords: Parallel processing, data analysis, real-time analysis, distributed computing.

## INTRODUCTION

Real-time analysis of telemetry data has been an elusive goal. The principal difficulty in doing such analysis is the computational complexity of reducing telemetry streams to intuitively useful information on the fly. Formerly, the best analysts could hope for was to archive all of the data they could, and do a post-run reduction of the data. Continuing advances in computer processing power, however, have put real-time telemetry analysis well within reach.

BBN is currently applying parallel and distributed processing technologies to real-time data reduction at two Navy laboratories. One of these applications is discussed in here, with particular attention to the hardware and software architectures employed to provide the users with seamless and coherent access to their data. The purpose of one of these labs is real-time analysis of interactions between various aircraft subsystems.

Several types of data sources need to be analyzed, including Mil Std 1553 busses, discrete signals, and special purpose interfaces. Data from each of these sources is packaged into a standard format to provide data packets which are independent of the

the data source. This use of a standard format allows the front-end analysis tool to be independent of the origin of the data.

## DATA FORMATTING

Many telemetry data sources enter the data reduction system as a raw bit-stream, whether from a decommutation device, an A/D board or some other interface. This data must usually be put into a coherent format from which useful data can be extracted. This frequently means searching through the bit-stream for time-tag codes, synchronization markers, et cetera in order to compose a data frame. Data frames from any one data source may have different frame types. Data frames of any given source and type generally contain a fixed set of variables, in a fixed format, with a different time for each instance of that frame type. For instance, all frames of type 1 from a given source might contain 3 words of data, representing the values of variables A,B, and C. Every instance of a type 1 frame would have a different time tag associated with it, identifying the data as samples of variables A,B, and C at the stated time. Similarly there might be frames of type 2, from the same source, which contain data for variables X,Y, and Z. Thus each data frame provides contemporary samples of several variables in a fixed format. This frame unpacking process of formatting raw data into frames of a standard format can be both CPU and I/O intensive.

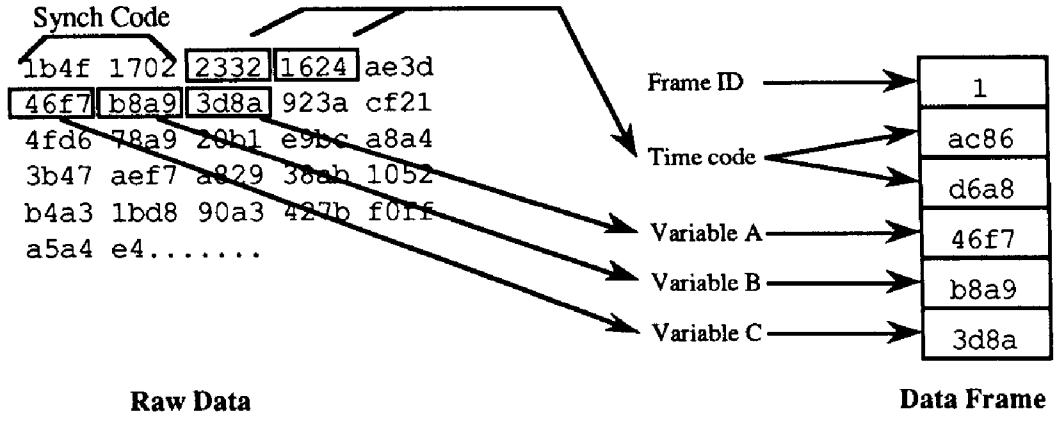


Figure 1. Frame Unpacking. By taking data from the raw bit-stream and packaging it into data frames with a time-code and ID number, the process of organizing the data at hand is simplified.

To analyze data in frame format, however, a key is needed to decipher which fields of which frames correspond to which variables. This key is provided by a data dictionary. The dictionary is a database which has an entry for every variable, containing information about the source type, units, scale factor, size, offset, data type,

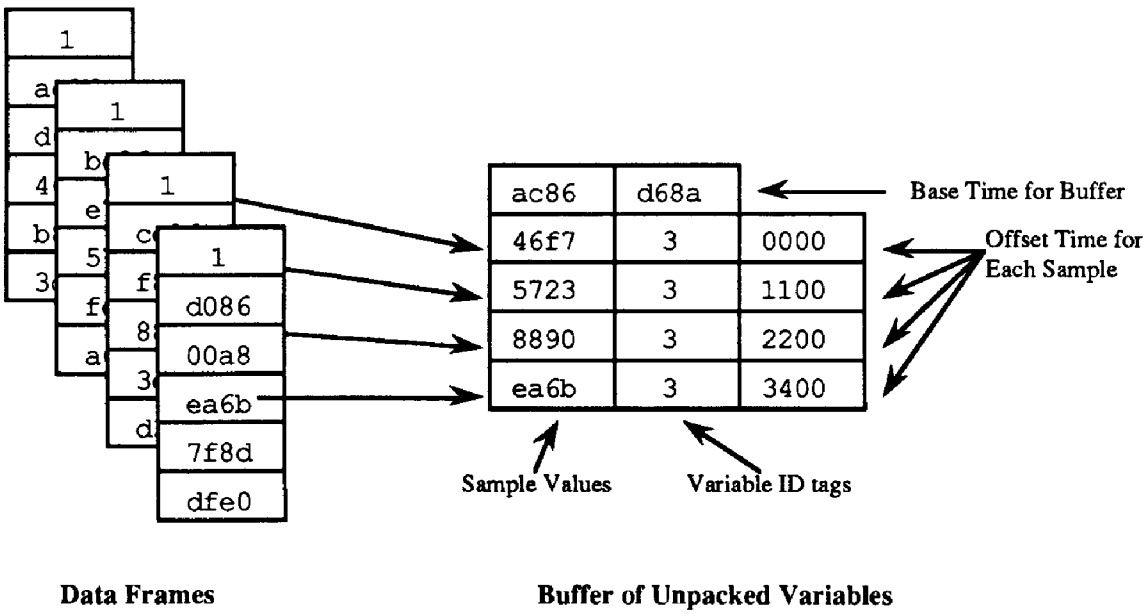


Figure 2. Variable Unpacking. Extracting samples of individual variables from multiple data frames and organizing them into a buffer of samples greatly reduces the processing required by other processors. A buffer of variable unpacked data can be used as input to an analysis package, which could display the data with virtually no computation required.

and bit ordering of the variable, as well as the frame type of the frame in which that variable is found.

The variable unpacking process applies the information found in the dictionary to the proper frame, yielding the value of the desired variable(s) in engineering units. The variable unpacking process can be a time consuming one, since it is here that bit- and byte-reversals are made, dependency conditions are evaluated, and E.U. conversions are performed. The result of variable unpacking is a buffer filled with samples. The buffer itself has a base time stamp, and each of the samples bears a time code which is an offset from the base time. This base-time-plus-offset scheme reduces the number of bytes required to represent the time of each sample, which is useful if the variables are to be sent across a network.

## HARDWARE SYSTEMS

A typical post-run analysis system would archive the raw data during a test run, and subsequently reformat the data file into data frames to serve as input to a popular analysis package like DataProbe. DataProbe would read the dictionary and the data frames and extract the requested variables for display, tabulation or computation.

All of this processing is done on the fly, in a multi-stage process, on a parallel processing system. Some processors collect raw data and format it into frames, while other processors extract the requested variables, handle requests for data from networked computers, and provide "X-Window" displays for a user interface. The key to coordinating this system is a software library known as the the Data Exchange (DX). The Data Exchange relies on a shared memory, parallel-processing model to coordinate senders (sources) and receivers (sinks) of data.

The laboratory real-time analysis system consists of a BBN Advanced Computers GP1000 parallel processor, several VME front end systems, and a Digital Equipment Corp. MicroVAX. The GP1000 runs both the Mach (multiprocessor UNIX) operating system and pSOS (a real-time operating system) concurrently on different processor nodes. The VME systems run pSOS, while the MicroVAX runs VMS. Both the VAX and the GP1000 also run the Cronus distributed operating environment, which provides object-oriented network services. A major function of the Data Exchange is coordinating the interaction between these computers and their operating systems.

The GP1000 parallel processor at the laboratory is a twenty-processor system. Each processing node contains a Motorola 68020 CPU with a floating point accelerator, a memory management unit, and 4 megabytes of RAM. The processing nodes are interconnected by a Butterfly Switch (a switched network which allows for dynamic allocation of processor-to-processor connections, and a scalable system bandwidth). The memory on each processor node is accessible by every other node, enabling shared memory parallel computing. The GP1000 also has VME system interfaces, which allow access to memory on a connected VME system as though it were part of the GP1000's shared memory.

## THE DATA EXCHANGE

The Data Exchange library is a set of routines which manages senders and receivers of data. Data senders are processes which generate data that may be used by other processes; for example, a data acquisition process or a real-time monitor might be a data sender. Data receivers consume data. They are processes like data archives or display generators. A third type of process is a data transceiver, which acts as both a sender and a receiver of data. Some common transceivers are unpackers, simulation processes, and data desamplers.

The Data Exchange library references a shared memory database to keep track of active senders and receivers. Each sender registers itself with a source type. The source type identifier is used to distinguish the source of the data that that sender will provide; for instance, different source types might correspond to different telemetry

channels. When receivers register themselves, they specify the source types of the data that they wish to receive, and they are matched to senders of those types, if they exist.

Processes that send data, whether they are frame unpackers, variable unpackers or data servers, do so on a buffer-by-buffer basis. A Data Exchange library call provides the address of a global memory buffer into which the data to be sent may be written. The sender must also fill in the values of a data structure that describe the data to be sent, including the time, data type, and size of the buffer. When the data buffer and its corresponding descriptor are completed, another Data Exchange library call publishes the buffer, making it available to receivers.

Just as senders must fill in the values of a buffer descriptor before they can publish their data, receivers must fill a similar data structure when they request data. Receivers must create data selectors that describe the data they wish to receive, including source types, the buffer types for each of those source types, and the period of time from which they want data. Once these selectors have been created, a Data Exchange call requesting a buffer will return the buffer with the earliest time stamp which matches the selector criterion. By only returning to the receivers the data actually requested, sometimes only a small fraction of the total data available in the memory pool, the volume of the data flow can be dramatically reduced as it progresses through the system.

For a remote system connecting to the shared memory pool, requests to send and receive data are done by remote procedure call (RPC) through a server process running on the host on which the shared memory pool resides. Care must be used when designing a system with remote connections, however, not to exceed the limited bandwidth of most network media. For example, while a remote host might suffice as a receiver of tag-data information at several thousand samples per second, it is inappropriate for high-volume frame unpacking, in which data rates can reach megabytes per second.

A critical issue in the design of a real-time system is how the system handles data overflow: In other words, what should happen when data enters the system faster than it can be processed. The Data Exchange allows for three solutions. A sender can declare itself to follow any one of these three regimens. The first mode is called request driven. In request driven mode, the sender will fill up as much of the shared memory pool as it has been allocated with data, and then wait until some of that data has been read by a receiver before sending any more data. The receiver gets a chance to examine as much of the data as it likes before the shared memory area is reused by the sender. Unfortunately, if the sender is blocked while waiting on a receiver, the

possibility exists that the sender might not be able to send its current data, and information would be lost.

A second mode in which senders can operate is in the free-running mode. In this mode, the sender is never blocked, and can recycle regions of shared memory as fast as it wishes. Any process receiving data from this sender will lose data if it cannot process its desired data faster than the free-running sender can send it. If this happens, the receiver is notified that data has been lost, and the earliest available frame matching the selector criterion is provided. Systems with free-running senders are most appropriate when the receiver is dealing with a very small subset of the data sent, when the shared memory allocated to the sender is large, and when the processing to be performed by the receiver is not extremely complex. In these cases the receiver can often keep up with the sender. If the receiver is only slightly slower than the sender, then the system will degrade gracefully, since the number of samples lost increases in proportion to the disparity in speed.

The final mode of operation is called change-from-under mode. In this mode, the receiver is given a pointer to the shared memory buffer, with the understanding that the sender may update that data at the same time that the receiver is reading it. The receiver is notified if this happens while it is reading the buffer.

## THE LABORATORY ENVIRONMENT

The aircraft laboratory generates large volumes of data which require real-time analysis. The instrumented airframe generates 400 to 600 kilobytes per second of 1553 format data alone. There are also several hundred discrete signals monitored at 600 Hz, and several other data sources which may push the aggregate data rates past 1 megabyte per second. BBN's approach to meeting these data throughput requirements is to break down the incoming data by stream. Each of the airframe's six 1553 busses is directed to its own GP1000 processor node. A seventh GP1000 processor node monitors the discrete signals. Each of these seven processors runs under the pSOS real-time operating system, which offers a very precise control over the allocation of CPU time on each of the processors. Under pSOS, one can ensure that there will be no unwanted interruptions to processing, caused by events such as virtual memory page faults, which are present in other operating systems. This control allows one to limit process latency in response to events from the airframe. Each of the pSOS processors declares itself to be the sender of the data stream to which it is assigned. The processors read in raw data via VME interfaces, frame unpack it and publish the frames to the data exchange.

Users of the system can request data by issuing commands in DataProbe on the VAX which cause data collection, such as commands for tabulation or plotting. These commands issue requests via remote procedure call to the GP1000. The request specifies the names of the variables required and the VMS dictionary file in which their descriptions can be found. A server process on the GP1000 relays the VAX's requests for variable data to a variable unpacking process which gathers dictionary information from the VAX via Cronus, through the Data Exchange. The variable unpacker then checks the dictionary entries, determines which frame types are required, and requests those data frames from the shared memory pool.

The variable unpacker runs under the Mach operating system. Because each of the frame unpackers has enough shared memory to buffer several seconds worth of frame unpacked data, less-than-immediate response from the variable unpacker can be tolerated; thus, response time is sacrificed for the convenience of running under Mach. The variable unpacker extracts the variables which have been requested by the user for display or tabulation. The variables are repacked into buffers, each containing multiple samples of the requested variables, and sent back into the shared memory pool. The values of these variables are now in a tag-data format, which consists of a time value, a data value, and a code identifying the variable. The variable buffers are then retrieved by the Mach server process, which are then passed over the ethernet to the VAX, where they are displayed by DataProbe.

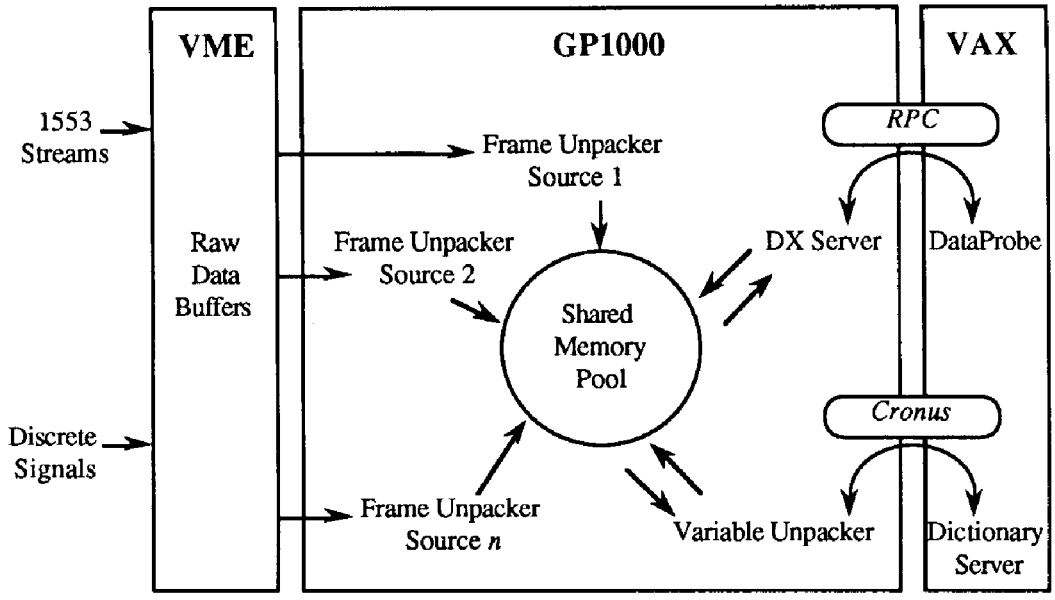


Figure 3. A Typical Data Exchange Application. Distributed and parallel computing require careful design of interprocessor communication, but can offer substantial benefits in power and flexibility. The data analysis application shown here uses the Data Exchange library to coordinate the operations of the many processors involved. The Data Exchange makes use of RPC and Cronus to incorporate remote hosts into the system.

The data may also be extracted from the shared memory pool in either frame or variable format and sent to another process, such as a data archiver, or perhaps a Unix-based analysis application running on the GP1000. The Data Exchange supports multiple fan-in and fan-out of data; that is, any number of receivers can elect to receive data from any one sender, and any one receiver can receive data from an arbitrary number of senders. Each receiver process that accesses Data Exchange data can run on its own CPU, independently from the other processes. Once data buffers are in the shared memory pool, the only limitation on the number of processes that can access that data is that limitation which is set by the memory access bandwidth. One benefit of this fan-out capability is that the Data Exchange can support multiple remote analysis workstations. Each workstation can connect to a different variable unpacking process on the GP1000, which would extract the variables of interest to the analyst.

To facilitate the use of the Data Exchange, BBN has incorporated an X-Window based control process for use in the laboratory. This menu driven system helps the novice user control all of the different processors in the system by automatically spawning the processes to provide the required data. Because the GP1000 supports the X-Window standard, adding graphic displays and user interfaces to Data Exchange applications on the GP1000 is easy.

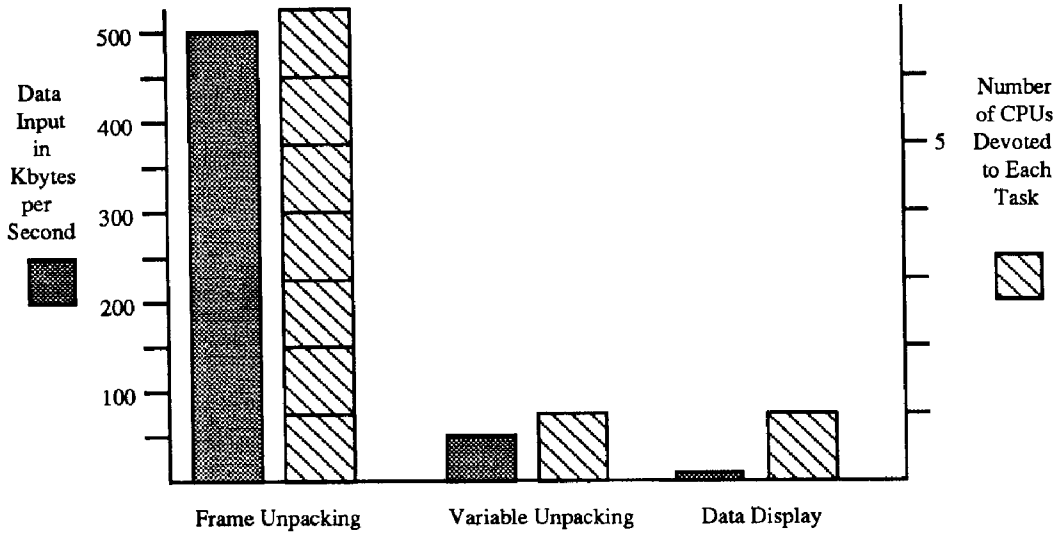


Figure 4. Applying Processing Power Where It Is Needed. One of the advantages of parallel processing is the ability to devote large amounts of processing power to the tasks that need it the most. Here, multiple CPUs are used where the dataflow is the highest. As the dataflows are reduced, the tasks can be handled by a single processor.

## FUTURE DIRECTIONS

Parallel processing clearly offers advantages in real-time telemetry processing over a serial processing system. The ability to dedicate a processor to managing each data stream, for instance, yields the benefit of reduced response time with respect to that stream. The ability to add processors to the parallel machine allows the system to increase in scale, with respect to both the amount of data which is passed through the system, and the complexity of the processing which is done on that data.

The Data Exchange software architecture has been designed to take advantage of this hardware architecture. As a new software module is required, it can simply be added as a stand alone process, receiving data from one source, and/or sending to another. For instance, if one wished to have a complex function evaluated in real-time, based upon the values of incoming data, one would merely have to create a software module which would post a request to the Data Exchange for that variable. The variable unpacker would send the requested data, and the function could be computed, and perhaps that value might, in turn, be passed to yet another process.

Because it is written in an operating system-independent manner, the Data Exchange operates with nearly uniform functionality across Mach, Unix, VMS, and pSOS. Furthermore, use of RPC and Cronus facilitates interoperation of disparate operating systems. Without a tool such as the Data Exchange library, attempting to make a system such as the one described here analysis system would be nearly impossible with current technology. The Data Exchange permits full advantage to be taken of the strengths of different operating systems, while still providing a cohesive system.

Further advances in parallel computing will increase the attractiveness of parallel computers for real-time data analysis. For instance, the TC2000 computer, the immediate successor to the GP1000, offers ten times the processing power per node, as well as ten times the interprocessor bandwidth. A system based upon a TC2000 would be capable of dramatically greater real-time throughput. As the capabilities of such computers grow, it is important that software architectures such as the Data Exchange be exploited to the fullest, to ensure that our abilities to do real-time telemetry processing remain as cost-effective as possible.

## ACKNOWLEDGEMENTS

The work described in this paper was supported by Contract N00123-90-D-0428.

## REFERENCES

1. Fidell, S,. Moss, P., Fortmann, T., Sneddon, M., Milligan, S., "Distributed Processing for Real-Time Data Collection, Display, and Analysis," Proceedings of The International Telemetering Conference, Volume XXIII, October, 1987.

2. Lynch, T., Fortmann, T., Briscoe, H., Fidell, S., "Multiprocessor-Based Data Acquisition and Analysis," Proceedings of the International Telemetering Conference, Volume XXV, October 1989.

3. BBN Advanced Computers Inc., "TC2000™ Technical Product Summary," July, 1989.

4. BBN Laboratories Inc., "Butterfly Parallel Processor," 1985.