A Distributed Signal Processing Facility for Speech Research

Nathaniel Polish

Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027

polish@CS.COLUMBIA.EDU

CUCS-386-88

Published in: Proceedings of the America Voice I/O Systems Conference, Oct 4-6, San Fransisco, CA, 1988.

Abstract

An interactive digital voice laboratory facility has been developed to allow the sharing of expensive signal processing resources among large numbers of interactive display devices. The environment considered in this work is a facility in which there are a moderate number of heterogeneous systems with extensive floating point numerical capability. In this environment there are a number of voice I/O terminals with the capability to display and play voice in a variety of ways. Using shared file access and remote procedure calls, any of the voice workstations may have signal processing jobs performed on any of the available floating point processors without explicit knowledge of where the computation is being performed. The appearance to the user of the voice workstation is that of a workstation with enormous resources.

Introduction

A voice research facility has been built that allows the user access to time and frequency domain presentations. Other traditional functions such as selecting intervals and placing markers are supported. What makes this facility particularly useful however is easy access to distributed signal processing resources.

Prior facilities have focused on presentation issues. In most approaches it is assumed that the underlying resource is simply a local computing system. To this end, powerful computers have been teamed with fast graphics to form voice workstations. As with any highly interactive application, a great deal of time is spent either showing the user something or waiting for a user input. In effect, the computing power of such a workstation must be matched to the maximum expected load. As a result expensive computing resources are left idle. The system that has been built effectively separates the display and interactive tasks from the computational tasks. This separation allows better utilization of computational resources within a larger research organization.

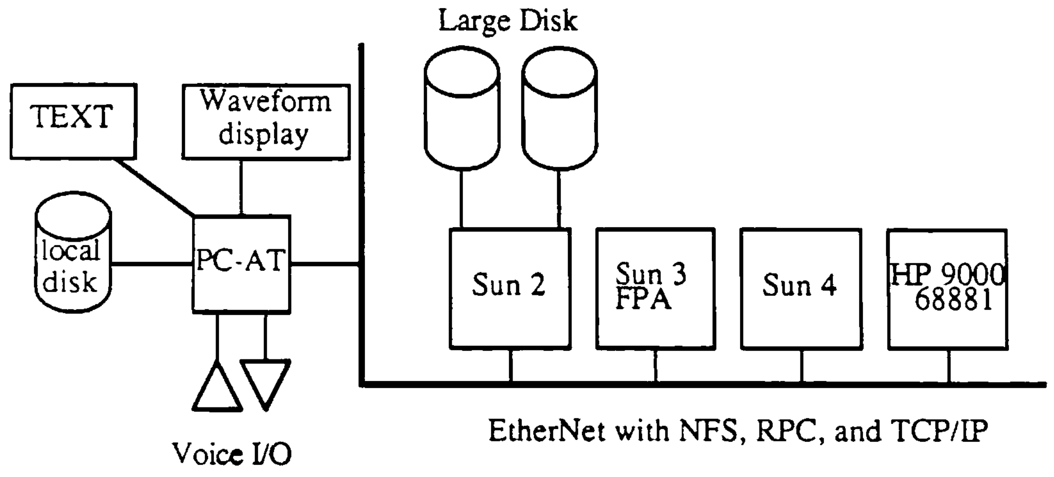

Figure 1: System Components

Why Build Such a System

It is common in research computing environments to have a significant number of powerful computers distributed around a facility. Sometimes there are large numbers of moderately powerful workstations (only a fraction of which are in heavy use at any one time). Sometimes there small numbers of unique computing resources such as a supercomputer or special purpose parallel machine. All of these machines are usually accessible over a moderate bandwidth digital network.[1]

In addition to the availability of networked computing services, the commercial popularity of personal computers has lead to inexpensive analog input and output capability as well as inexpensive high-quality graphics. Unfortunately the computational capacity of these computers is poor relative to common signal processing requirements. Special signal processing boards are available but these require very special and unique software. For under four thousand dollars it is possible to build a workstation using IBM PC compatible hardware that has good graphics[2] and 16 bit 35Khz analog I/O. These graphical and I/O capabilities are entirely adequate for most voice work so the problem is to provide storage and computing resources more in line with the requirements of speech research.

Another, considerablely more expensive speech research system is the SPIRE system developed at the Massachusetts Institute of Technology. This system utilizes a Symbolics 3600 and is written in LISP. The system is highly interactive and extensible (in LISP) however the hardware cost is in the tens of thousands of dollars per station. Further, the computation speed of the workstations themselves, even with a floating point processor, is not very good. The major point of this sort of workstation is fast prototyping and very good user interface. Unfortunately, the cost is high and computation speed low.

Often the distributed computing resources available are not entirely compatible. For example, methods of representing 16 bit integers vary from processor architecture to processor architecture. Floating point representations, despite the IEEE standard in this area, are notoriously different from machine to machine. Even when the processors are the same, the executable code format varies from operating system version to version. It is therefore important to have a methodology that will make these differences transparent.

There are many signal processing applications in which course-grain parallelism is appropriate. In most recognition approaches, the signal to be recognized is compared to a large number of stored templates. These comparisons are usually independent and so could be implemented on parallel processors. Parallel processors are difficult to prototype. In a distributed computing system it is possible to simulate a dedicated parallel processor. In addition, the distributed system itself forms a course-grained parallel processing system.

Design

In order to link to these resources, a client/server model of computer to computer interaction was chosen. In this model some computers act as servers while others act as client computers. In this system, the servers are computers with significant computational resources to share and the clients are workstations with graphical and signal I/O capability. When using a multitasking operating system such as UNIX, several, virtual servers may be running on a machine at one time.

In general, the way to assure machine independence in data representation is to define a canonical representation. It is then the responsibility of each computer to transform data from the canonical form to the processor's internal representation and back again. Since the graphics and I/O computer has minimal computing resources, the canonical form chosen was the internal format of this computer. In the system built, the processor is an 80286 processor that stores 16 bit integers with the high-order byte in a lower memory address than the low-order byte in

[1]The particular environment in which this work was done consists of 6 Sun-3s, 12 Sun-2s, 45 HP workstations (68020 based), 6 Vax 750s, and a 1024 node DADO tree machine.

[2]The Enhanced Graphics Adapter (EGA) has a resolution of 640 x 350 by 4 bits deep and a pixel draw rate with reasonable software in excess of 35,000 pixels per second.

contrast to the 680xx processors in the other computers that store the high-order byte in a higher address than the low-order byte.

In addition to machine independence for data, it is important to have machine independence in coding. This requires that the coding of the algorithms involved be done with some care. In the case of the system built, all the server machines were running variations on the UNIX operating system. Care simply had to be take to avoid word size dependent coding in the C language.

Implementation

The basic system components are indicated in figure 1. The underlying network used in this system is an Ethernet. Ethernet interfaces are pervasive. All of the computers that were of interest for this work had Ethernet interfaces available for them. Ethernet also has sufficient bandwidth for the voice work contemplated. In particular, the transmission time is small compared to the computation time. It is easy to imagine situations in which this might not be true. In this case, optical networks hold great promise. Transmission bandwidths available with optical networks are several orders of magnitude greater than current coax cable based networks.

The method chosen for implementing data transmission over the network was file sharing. Explicit data transmission could have been used, however, file sharing offers considerable benefits while the costs can be mitigated. File sharing simply means that there is a common disk available to all of the potential servers and communication is achieved by writing data to an agreed upon location on the disk. A major advantage of this approach is that the destination machine need not actually be available to receive data. Debugging is simpler with files than data streams. The only penalty involved in using the file sharing approach is the delay for actual disk writing and reading. With appropriate buffering this delay can be made nearly invisible.

File sharing is implemented using the Network File System (NFS) from Sun Microsystems. NFS is widely available and well documented. An important reason for using NFS is the fact that it is built on remote procedure calls. The use of remote procedure calls will form the basis for the rest of the system. NFS allows the mounting of a remote disk system as if it were local storage. NFS takes care of all network level error handling and provides reasonable translations between different operating systems. For example, in UNIX a file name is case sensitive and of nearly unlimited length; while on the IBM PC (running MS-DOS) file names are case insensitive and are at most 8 letters with at most a single three character extension. NFS provides naming translation in this case. Conversion of information such as dates, time, owner, permissions, last writer, and length are also performed by NFS. NFS is stateless so that if a server crashes and is restored, the client machines need not reinitiate their connections -- they resume as before the crash.

Actual requests for computational services are made through the remote procedure call (RPC) mechanism. RPC is a system service implemented by Sun to support the Network File System. In RPC a server process on a machine registers each of its callable routines with the operating system. The parameters for the remote procedures are declared as well as data translation routines to support interpretation of the parameters. In the case of this system, the parameters passed are just the file names of the files to be operated on. For example, there is a remote fast fourier transform (FFT) routine in the system. The parameters are just the name of the input file and the name of the output file. Since all of the machines on the Network File System have the same view of the file system, the names passed are valid on any machine.

The call made on the client computer is made after contact is established with the server process. Contact is established with an "open" call. The choice was made to have a few server programs each of which supports several signal processing functions. The particular function desired is encoded as a parameter to the server. Consider the case of the FFT call. First an open call is made to the machine that is running the server program. Then the data to be transformed are written to a file and a call is made to the remote server with the predetermined function number for FFT as well as the name of the file to be transformed. As a return result from the remote call we get the name of a file which contains the result of the transform.

As discussed above, there are several good reasons for using files as the transfer medium. A cleaner method would be to pass the data to be transformed as a parameter to the remote routine and then expect a return value to be the result of the transform. Unfortunately the transmission protocol is based on a transmission protocol (UDP) which limits parameters to 8K bytes of data. It is however possible to use TCP as the protocol and have nearly unlimited parameter size. At some point in the future we will move the system to TCP. In the mean time the performance is not significantly impacted by the file writing as the calculation time dominates the file write time.

The interface to the system is primarily through an IBM PC equipt with both a monochrome monitor for text and an enhanced graphics adapter. The graphical interface represents three speech buffers. Each of the three buffers are displayed simultaneously in time domain. A low resolution plot of energy is provided as one line for each speech buffer. A cursor can be moved left or right along each of the speech buffer energy plots. The cursors have definite width. There are three high resolution windows (one for each speech buffer). The high resolution windows display the waveform corresponding to what is indicated by the low resolution cursor. The high resolution windows are updated as the low resolution cursors are moved. Marks can be placed at any point in any of the speech buffers. The marks are displayed as yellow vertical lines.

While any of the speech buffers may be used for any purpose, the general intent is more specific. The first buffer is the primary buffer. Operations such as filtering, smoothing, and voice/unvoiced detection are typically performed on the primary buffer. The results of these operations are placed in the second buffer. The cursors for each of the buffers may be moved in lock step. In this way, it is simple to judge the results of particular algorithms. Binary operations such as distance measures are performed on the first two buffers with the results displayed in the third buffer. The markers are also used to delimit sections to be saved as separate voice files or to be played as short segments. Parts of the different buffers can also be concatenated together to be played as one piece. The functionality of the facility is always changing as signal processing functions are added and manipulated.

All of the files used by the system are stored on one or more network file system servers. As a result, all of the files are available to all of the machines simultaneously. It is straight forward to have large numbers of graphics workstations on the network at the same time accessing these files. The file system manages the problems of multiple access. In general it is best for each of the stations to have its own file space with common, read-only pools of voice files.

There must be a server process for each client workstation. Idle server processes occupy almost no resources on their machines. As a result it is not a problem having many machines doing signal processing at the same time. It is important, however, to have a facility to check the CPU load on each server. It is usually best to pick a server with the lowest load -- though this may not always be the case. Many load balancing mechanisms are possible with this system and only a few have been explored.

Applications

Sampled data such as speech must be band limited before it is sampled to avoid aliasing of frequencies above the nyquist rate. The analog filters used to band limit always leave some artifact in the high frequency sections of the signal due to the ripple inevitable with any real-world filter. To provide speech input that has no ripple, a very high order low pass digital filter is provided to further band limit the signal below the limit of the analog input filter.

This brickwall filter is provided as a simple remote procedure call as the data are received by the I/O computer.

Linear predictive coding (LPC) is a very common speech signal processing technique and is the starting point of a great many other signal processing algorithms. A remote procedure call version of LPC has been built for the system. The call takes a time series voice file as input and produces files for the LPC parameters and errors. Fast fourier transforms (FFT) are also very useful for further processing and spectrogram displays. FFTs of various widths and window functions have been implemented for the system.

A very important signal processing application that is used in many, larger systems is distance measure computation. Distance measures take as input two samples of speech signal and return a number or a time series which is interpreted as the distance (with respect to something) between the two inputs. A simple distance measure for example is just the difference in squared energy over time. More complex distance measures such as dynamic time warping, itakura, and cepstrum have been implemented on the system. Over a dozen different distance measures have been implemented on the current system to support several different investigations. Distance measures are at the heart of most recognition tasks as well as the concatenative synthesis project described below.

The major use that the system has been put to is a project involving the selection of vocabularies for a concatenative synthesis project. In concatenative synthesis systems there are vocabularies of elements used in the construction of speech sounds. In prior work, the sets of elements are selected by hand using some sort of editing system. It is usually not possible to be sure that the choice of elements is optimal in any sense or in fact to investigate whether the pool of elements should be expanded or can be shrunk without loss of overall quality. This project involves making large numbers of comparisons between natural and synthesized speech to determine an optimal vocabulary set. This large number of comparisons are implemented as remotely executed distance measurements on many different computers.

Summary and Future Directions

The major motivation for this work is access to greater processing power than would otherwise be available. Taken as a start, the 8 Mhz 80286 machine with math coprocessor is one sixth the speed of the system with server running on a Sun 2 (10 Mhz 68010). A Sun 3 with a floating point accelerator board is fully ten times the speed of the Sun 2. We have therefore obtained a 60 fold speed up over the personal computer. Further the server computer can be used by several clients at once with little apparent degradation.

The servers have been implemented on Sun 2s, Sun 3s, and HP 9000 workstations. In all of these implementations, the code was developed on the Sun 2. Porting consisted only of recompiling the code with the appropriate floating point options. Of course all three of these machines use the same data representations. However, software that detects what the byte ordering is in a given machine is straight forward.

Probably the most interesting aspect of this work has been the investigation of client/server models of computation in an instrumentation environment. One can easily imagine a standard protocol being established for data and remote functions. Once such a standard were established (as it has been within our lab) instruments could simply be hooked to each other via a digital network. Such a network would probably need to be optical for many interesting high performance applications, though coax is fine for some applications as we have seen.

One could have a generic spectrum analyzer that is hooked to a particular input system via the network. In this case the input system would act as a server of data to the analyzer. Any input system could be used so long as it conformed to the protocol. Many other such combinations of analysis tools and input/output systems are possible with a digital interconnection using a client/server protocol. The flexibility and noise immunity of such a system make an attractive possibility.