# Implementation of *Software Radio* Systems Using Distributed Architectures

Henrique C. Miranda[†‡], Gustavo J. A. M. Carneiro[†], Pedro C. Pinto[‡], Sérgio B. Silva[‡]

[†]**INESC Porto** - Instituto de Engenharia de Sistemas e Computadores do Porto
Rua Dr. Roberto Frias, s/n - 4200–465 Porto. Telf.: 22 2094000, Fax: 22 2094250
[‡]**FEUP** - Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200–465 Porto, Telf: 22 5081474, Fax: 22 2094250
e-mail: hmiranda@fe.up.pt

## Abstract

This paper proposes an alternative architecture for the implementation of Software Defined Radio (SDR) systems, providing low-cost, high degree of flexibility and scalability. The architecture is based on a cluster of COTS (Commercial Off-The-Shelf) computers interconnected by standard network devices (LAN network interface cards and switches). This architecture is mainly intended for testing software radio algorithms in parallel/distributed computing environments capable of real-time processing.

**Keywords:** Software defined radio, parallel computing, UNIX clusters

## 1 Introduction

Currently, a new concept is evolving in the wireless telecommunications technology: the concept of Software Defined Radio or Software Radio [1]. Software Radio terminals will be able to support a large variety of communications standards without the need of any hardware modification in order to switch between standards. The Software Radio should be based on a wide-band, high dynamic range transceiver architecture that is fully and flexibly programmable instead of "hard-wired", application specific functional units. By defining appropriate interfaces and protocols together with the reconfigurable hardware enables the use of multi-standard, multi-mode radio terminals and base stations with broad interoperability capabilities. Realization of the software radio principle has many issues ranging from the problems of RF circuits and signal processing aspects to Software Radio API definitions and download protocols.

The proposed architecture discussed in this paper is the outcome of the *ClusteRadio* project. Within the framework of this project, a low-cost, scalable and distributed radio system was developed, with all radio functions implemented in software, including channelization, one of the most demanding ones. In order to use this system in a real-life context, a RF downconverter is required to shift the received signal to the IF range, where it is digitized by the cluster A/D converter. Also, the project development was done on commodity computers, also known as COTS computers, so that powerful development and debugging tools could be used, reducing development time. Real-time operation is achieved using I/O data buffering, circumventing processing jitter inherent to non-real-time operating systems.

## 2 Trade-offs in SDR architectures

The most common hardware options for SDR systems have been ASICs, FPGAs and DSPs. These devices range from high speed and minimum flexibility to maximum flexibility and very little hardware optimization. However, when it comes to implementing complex SDR algorithms, the availability of powerful software tools and languages for the hardware platform is essential. This is a strong argument favoring the use of another alternative: the general purpose processor (GPP) used in commodity PCs. The capacity of this alternative is continuously increasing in terms of memory and computational power, assuming a very promising position in the SDR arena. An example of the utilization of GPPs in SDR is described in [2].

Table 1 summarizes the trade-offs one should consider when designing an SDR device.

Unfortunately, for the time being, a single computer is not usually fast enough to handle the demanding amount of processing required in SDR systems. To cope with this limitation, computer clustering techniques can

Table 1: Different trade-offs in hardware solutions for SDR.

| Hardware | ASICs | FPGAs | DSPs | GPPs |
|---|---|---|---|---|
| Programability | low | medium | good | very good |
| Power consumption | very low | low | medium | high |
| Platform size | very low | low | medium | high |
| Software availability | low | low | medium | very high |
| Software portability | none | low | medium | good |
| Software cost | very high | high | medium | very low |
| Availability | low | low | medium | very high |

be employed. A *cluster* can be defined as a set of computers, called *nodes* in this context, inter-connected by a network, forming a single virtual computer. Together, all the nodes in the cluster sum up to achieve a level of computing power much greater, and still much cheaper, than most single high-performance computers, such as mainframes. The most important advantages of cluster computer systems include:

**Economy:** a better performance/price ratio is attained;

**Performance:** a higher total performance can be achieved;

**Scalability:** computational power can be added incrementally without rigid bounds;

**Availability:** cluster nodes are based on widespread and very low cost hardware, available from multiple vendors;

**heterogeneity:** different computer platforms can be mixed within the cluster (for instance, x86, SPARCs, Alphas and PowerPCs can all coexist cooperatively).

# 3 Proposed architecture

The SDR architecture proposed in this paper is presented in Figure 1.

The architecture is mainly built by computers based on general purpose processors (that can be of different types) and an internetworking device (usually an Ethernet switch) providing data paths between nodes. Some ports of the switch should support very high data rate network interface technologies, such as Gigabit Ethernet, to establish critical paths between two or more nodes (as an example, the interconnection between the digitizer node and channelizer node(s) could benefit from this facility). All the nodes are inside a private virtual LAN (VLAN) isolated from the outside network; this is a desirable configuration since the cluster should not be bothered by outside traffic and vice-versa.

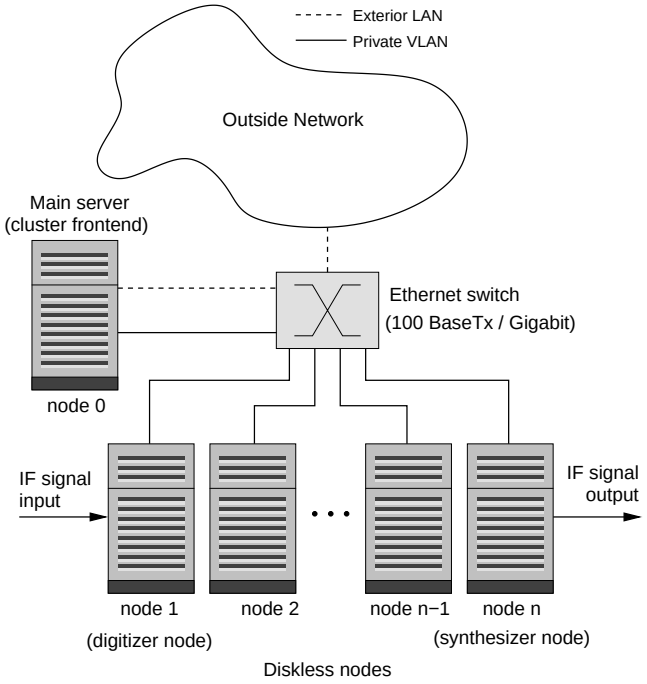


Figure 1: Proposed distributed SDR architecture.

The frontend node is the cluster interface to the outside world where applications can be launched and processes can be monitored. Apart from this node, the remaining ones are diskless, and they all boot from the frontend node. These nodes can still perform file operations either by using the server network mounted file system (NFS) or by using a virtual temporary filesystem of configurable size that each node creates in its memory at boot time. By being able to build the cluster with only a single storage disk, the cost per node is significantly reduced.

When it comes to software running on distributed platforms like this one, the following communication methods are usually available: TCP/IP sockets, PVM (Parallel Virtual Machine), MPI (Message Passing Interface), RPC (Remote Procedure Call) and CORBA (Common Object Request Broker Architecture). In the authors' opinion, PVM and MPI are more naturally suited for data flow-oriented systems, like this one.

PVM and MPI APIs are centered around sending and receiving messages. Here messages have defined data types, and data conversion is performed automatically as need. This type of communications is said to be network transparent. Conceptually, there are no major differences between these two systems. In fact, for the most part they have similar APIs. Between PVM and MPI, the latter was chosen because a better internode communication performance can be attained and it is actually a programming standard [3], with several implementations available that are regularly updated. Some early experiments were performed with PVM, but it had some annoying flaws that motivated the switch to the MPI architecture. The MPI implementation selected for this project was LAM (Local Area Multicomputer), most often referred to as LAM/MPI [4].

# 4 Experimental setup and results

The next paragraphs describe the hardware and software used to build the *ClusteRadio* platform. As a test application, a distributed AM/FM receiver was used with several parameters defined in software such IF center frequency and bandwidth, AGC level and time constant, and the system sampling rate.

## 4.1 Hardware

The cluster is mostly PC based. There is a server (cluster frontend) with a 20 GB hard disk and four diskless nodes. All of them have a 500 MHz AMD K6-2 with 128 MB of RAM. Additionally, a Sun Ultra 10 workstation is used as the A/D conversion node. It has 512 MB of RAM and a 466 MHz UltraSPARC-IIi processor.

For sampling the IF signal, an UltraView Corp. ADDA12-100DMA[1] board was plugged into the UtraSPARC's PCI bus. It is capable of sampling two simultaneous 12-bit channels at up to 25 MHz, or a single channel at up to 50 MHz. This board has also a pair of 14-bit, 18 Msps D/A converters that can be used in software defined transmitter applications.

Figure 2 shows a picture of the *ClusteRadio* testbed described in this paper.

## 4.2 Software

The platform uses two separate OSes. The UltraSPARC node runs SunOS 5.7 (Solaris 7) and the PCs all run Linux 2.4. Linux is the primary choice of OS because of its well known stability, performance and flexibility. The need for the UltraSPARC node arises from the fact that software drivers for the digitizer board were only available for the Solaris OS.

[1] http://www.ultraviewcorp.com

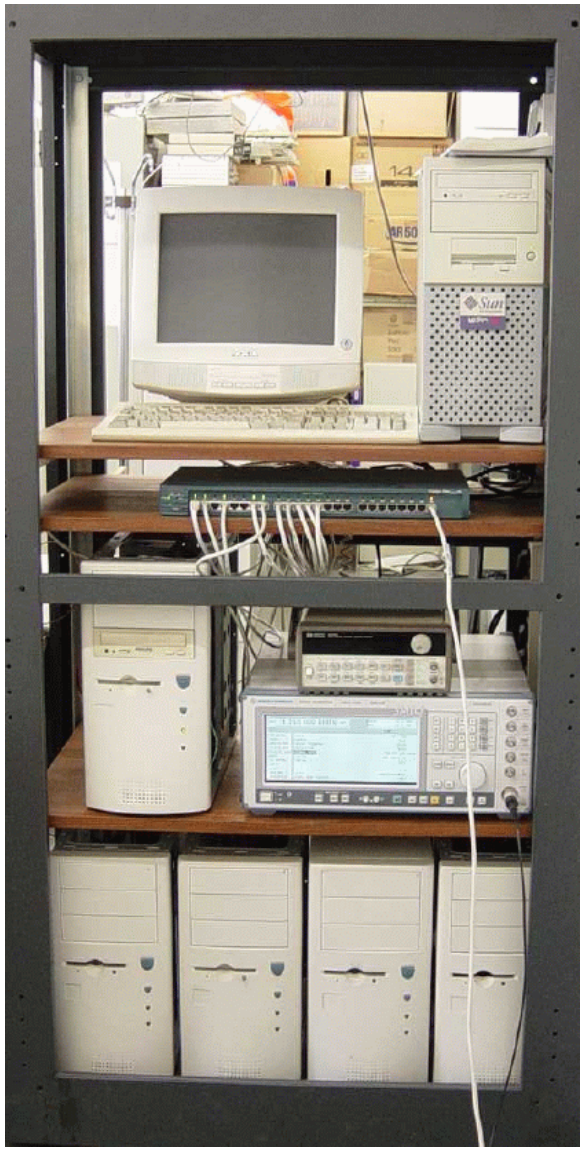

Figure 2: Picture of the *ClusteRadio* testbed.

Among other development tools and libraries, the LAM implementation of the MPI standard was used to provide transparent interprocess communication.

## 4.3 Network setup

Network performance is of vital importance to the overall cluster operation. With that in mind, all network devices operate at the relatively fast speed of 100 Mbit/s. The main network interconnection device is a 100-Mbit/s, 24-port Ethernet switch which is VLAN capable. Each node has a 100 Mbit/s full duplex network interface card (NIC), except for the cluster frontend that has two of such cards, one connected to the global LAN (campus network), and the other connected to the private cluster VLAN. The only PC with a hard drive is the cluster frontend. The diskless PC nodes

as well as the UltraSPARC workstation (equipped with the digitizer board) are connected to the private VLAN. NICs at each diskless node contain an EPROM plugged in, programmed with software needed for the booting process (BOOTP and TFTP clients)[2]. This netbooting approach further reduces the cost/node as these no longer have the need for a floppy disk drive[3].

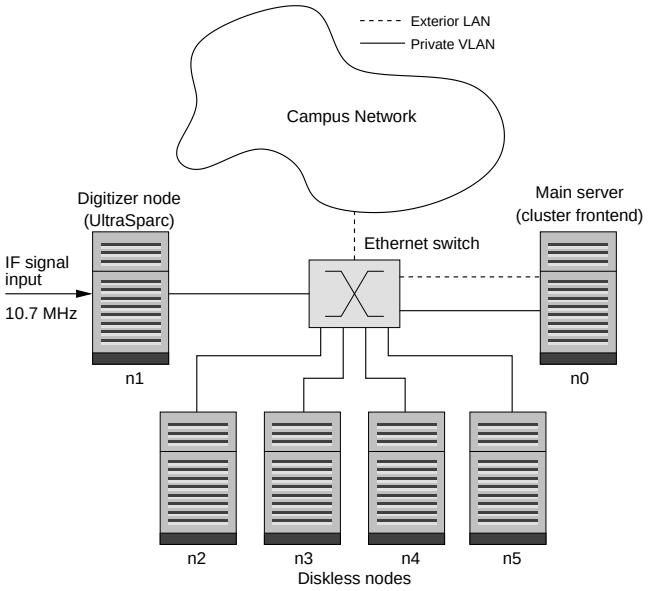


Figure 3: Network setup for the distributed receiver.

## 4.4 Signal processing blocks

Figure 4 shows the overall signal processing chain. Each block actually represents one process running on a node. It does not matter on which node a particular process runs, with the following restrictions:

1. The `acquire` process is in charge of digitizing the incoming IF signal, and so it must run wherever the digitizer board is — currently the UltraSPARC station.

2. The `output` process's mission is to dump the demodulated audio signal onto the sound card — currently installed in the cluster frontend.

Besides these two exceptions, there is absolute freedom to place any process anywhere. The MPI libraries take care of the rest.

The other processes are:

`tune` This process implements a digital downconverter (DDC). In other words, it tunes a specific channel frequency. It is composed by a complex NCO (Numerically Controlled Oscillator), a complex multiplier and a CIC decimator.

[2] http://etherboot.sourceforge.net/

[3] Each diskless node costed approximately 200 USD

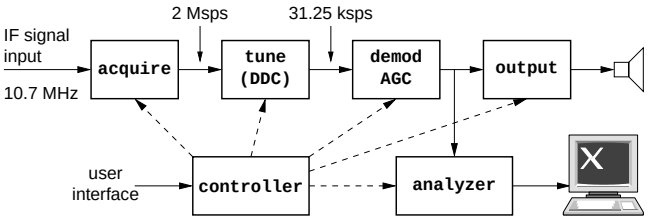


Figure 4: Processing block diagram.

`demod` This process runs the AM/FM demodulator, including the AGC controller.

`analyzer` This is simply an auxiliary program that displays the demodulated signal in the time and frequency domains.

`controller` This constitutes the "control center" of the receiver. It consists of a window with controls that adjust the following parameters: sample frequency, IF center frequency, AGC time constant, demodulator mode (AM/FM) and analyzer display. This window is depicted in Figure 5.

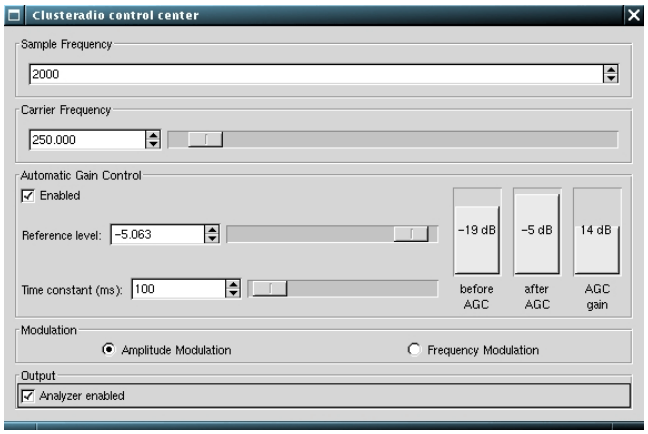


Figure 5: Application control center.

The description of the signal processing algorithms used in this receiver can be found in [5].

## 4.5 Experimental results

Tests were performed using the configuration shown in Figure 4. It shows a specific allocation of processes to nodes. Also, to emulate an AM/FM IF signal, a programmable signal generator with modulation capabilities was used.

The `acquire` process runs at a sampling rate of 2 Msps, with the `tune` process employing a decimation rate of 64. It can be estimated that the average network

bandwidth required in the communication between `acquire` and `tune` is 4 Mbytes/second (2 million samples and 16 bits per sample). The sample rate after decimation is 2000/64 = 31.25 kHz. At 32 bits per sample (it is a complex sample with 16 bits for real and imaginary parts), the network bandwidth required between `tune` and `demod` is approximately 122 kBytes/sec.

It is clear now that moving the `tune` process into the same host as `acquire` will eliminate a 4 MBytes/sec connection (actually more, because of the TCP/IP overhead). This is what motivated the configuration used in the present case. It should be noted, however, that moving the `tune` process into a different node is as simple as changing a single digit in the cluster configuration file. Here is the configuration file used for this application:

```
n1 acquire -s 2000 -d 64
n1 tune
n4 demod
n0 output
n3 analyzer -display=clusteradio:2
n0 controller
```

The `n0`, `n1`, `n3` and `n4` labels refer to a cluster node. Moving the `tune` process away from the acquisition computer will greatly increase the network load but, on the other hand, will alleviate the processing burden of that node. Here we can choose which limit we want to submit to: either network bandwidth or CPU load.

Each node had its CPU load measured and the results are shown in Table 2.

Table 2: CPU loads for two sampling rates (in MHz) decimation ratios (d).

| Process | Node | CPU | CPU Load | |
|---|---|---|---|---|
| | | | fs = 2.000 d = 64 | fs = 3.125 d = 128 |
| `acquire` | n1 | SPARC | 23 % | 42 % |
| `tune` | n1 | SPARC | 19 % | 23 % |
| `analyzer` | n3 | K6 - 2 | 15 % | 10 % |
| `demod` | n4 | K6 - 2 | 6 % | 6 % |
| `output` | n0 | K6 - 2 | $< 1$ % | $< 1$ % |

Some interesting conclusions can be drawn from these figures:

- `demod` does not significantly load the CPU, which is expected since it runs after the DDC (at a much lower sample rate) and employs simple algorithms;
- The programs `demod` and `analyzer` did not consumed more CPU when the sample rate changed to 3.125 Msps, which was to be expected since the decimation factor also increased to 128, yielding a final sample rate of 24.4 ksps, which is actually slightly lower than the original one;
- The relation between the sample rate and the CPU load related to the `acquire` process is almost, but not exactly, linear;
- It is surprising to note that `tune` only consumes slightly more CPU time at 3.125 Msps (a 18% CPU increase for a 56% sample rate increase).

Another important indicator is the time delay between the input and output nodes, also known as latency. Measured values presented in Table 3 have been measured by using timestamps. As the data transfer in the acquisition node is performed in blocks of $2^{16}$ samples, that number of samples divided by the sample frequency results in the additional sampling delay that has to be considered to calculate the total system latency.

Table 3: I/O latency (ms).

| Sample rate | Decimation | delay | acq. time | total |
|---|---|---|---|---|
| 3.125 Msps | 128 | 393 | 21 | 412 |
| 2.000 Msps | 64 | 472 | 33 | 505 |

This table shows that latency slightly decreases as the sample frequency is increased. Some of the factors that influence delay are network speed, network latency, complexity of the DSP algorithms, processor speed, number of DSP processing blocks between input and output and data block size.

# 5 Conclusions and future work

The goal of this project was to build an inexpensive cluster for software radio signal processing. Also, it was required to evaluate the cluster's performance for real-time processing with DSP algorithms for radio systems using the LAM/MPI parallel processing libraries. The flexibility of the proposed architecture has been demonstrated by configuring the cluster into an AM/FM radio receiver with adjastable AGC parameters, IF signal frequencies and sample rates. Software processing and network introduce a noticeable delay of about half a second which is irrelevant for broadcast receiver applications like this one. However, for interactive audio/video communications (mobile telephony) this delay is of vital importance.

The cluster potential has not been fully exploited; there are plenty of resources still available, as shown in Table 2, which gives room for increasing the complexity of the processing or the possibility to increase the number of channels simultaneously demodulated. Furthermore, the architecture functionality can be easily reversed, making the construction of a software transmitter a trivial task (the acquisition board includes two high-speed D/A converters).

Currently, this platform is being used as a test bed for the implementation of adaptive digital receivers with

self-reconfigurability – a much more demanding application than the one described here. These receivers are capable of identifying key signal parameters such as symbol rate and constellation and, in an autonomous way, they can modify their operation accordingly [6].

Future research work will evaluate the feasibility of using this distributed architecture to implement *cognitive radios* — radios that are environment aware and by incorporating artificial intelligence can independently determine their operation [7].

This architecture is also very suitable for adaptive beamforming antennas (smart antennas) in which a high degree of parallelism is required. This kind of application will also be addressed in the near future.

# References


[1] J. Mitola, "The Software Radio Architecture," *IEEE Communications Magazine*, vol. 33, pp. 26–38, May 1995.

[2] V. Bose, M. Ismert, M. Welborn, and J. Guttag, "Virtual Radios," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 596–602, Apr. 1999.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. The MIT Press, 1996.

[4] The LAM/MPI Team, *The LAM/MPI Users's Guide*. `http://www.lam-mpi.org/using/docs/`.

[5] M. Frerking, *Digital Signal Processing in Communication Systems*. ITP/Chapman & Hall, 1994. ISBN 0-442-01616-6.

[6] P. Pinto, S. Silva, and H. Miranda, "An adaptive Software Radio Receiver architechture for linear bidimensional modulations," *Sixth Baiona Workshop on Signal Processing in Communications*, Sept. 2003.

[7] J. Mitola and G. Q. Maguire, "Cognitive Radio: Making Software Radios More Personal," *IEEE Communications Magazine*, vol. 6, pp. 13–18, Aug. 1999.