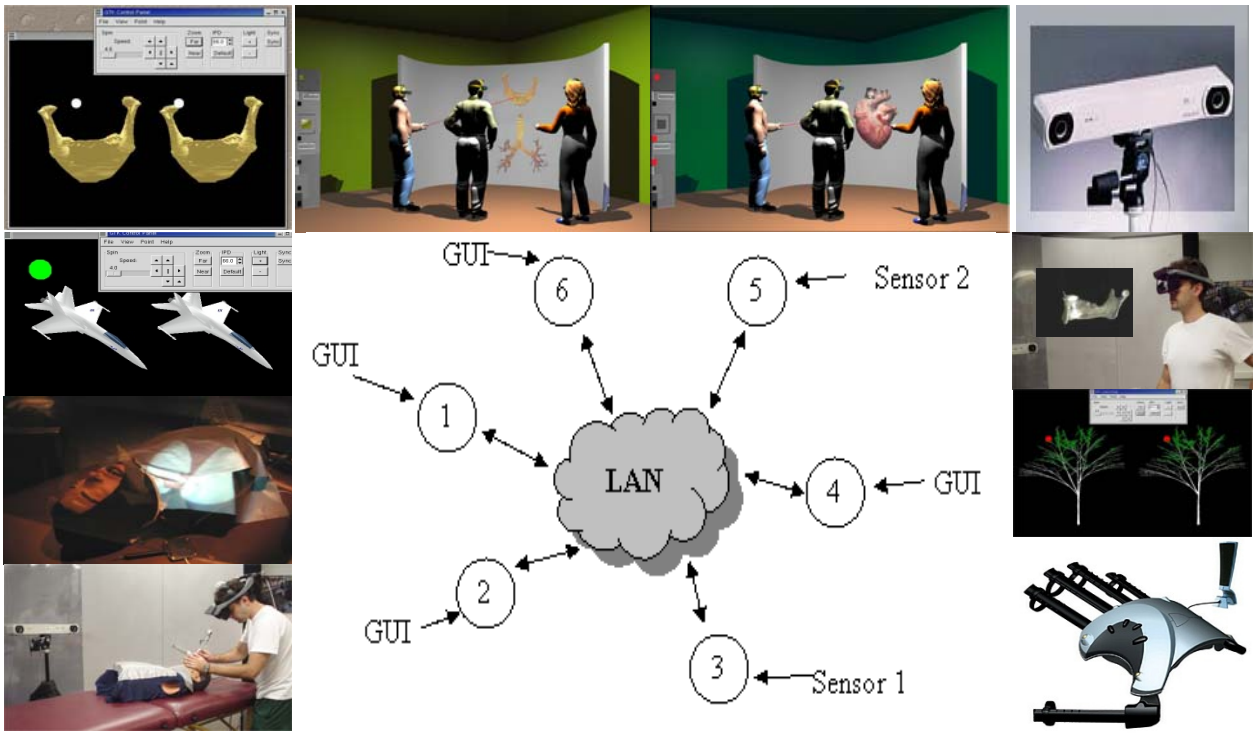
A Less Intrusive System Monitoring Scheme for Distributed Virtual Environments

Felix G. Hamza-Lup



Technical Report
School of Computer Science
University of Central Florida
August 2004

Contents

CHAPTER 1 - MONITORING TOOL

1.1 Introduction

This work is driven by the need to maintain a consistent shared state among several participants in a virtual collaborative environment. Monitoring the node resources will allow compensation for the system's delays at the application level.

Monitoring a computer system's resources is not a new concept. There are several software packages and applications for various operating systems that perform system monitoring (Fontaine 2004). What sets this work apart is a simple monitoring scheme that uses the underlying operating systems facilities and can be triggered remotely using scripting languages.

Monitoring can be done in hardware or in software. Hardware monitoring is non-intrusive, but more expensive and rigid than software monitoring. Extrusive, hardware monitoring usually collects system statistics using specialized hardware without consuming resources on the system being monitored. While this method yields more accurate values for the collected data, the overhead of the approach consist in the specialized hardware that must be built and set up for the specific application. In contrast, software monitoring is an intrusive approach that is flexible and low cost. Software monitoring is by default intrusive since the resources needed to accomplish data collection will add to the computational system load. Although the accuracy of the data being collected will be slightly affected, accurate results can be obtained by extending the monitoring period and through statistical analysis.

1.2 Installing the Monitoring Tool

In the following sections, a software monitoring tool is presented. This section contains important information for the configuration of the monitoring tool under Linux Red Hat 7.2 distribution.

The monitoring module is distributed in a ***tar*** file. Simply unpack the tar file, *dareSysMonitor.beta.tar*, into your home directory. An example of how to use the modules is given in *example_of_use.cpp*.

Example:

```
dareStartMonitor(300,true,false,true);      // This would cause the monitor to run
                                            // for 300 seconds and output
                                            // CPU and Network.
```

Save, and compile using the provided "*Makefile*" to automatically link all necessary components. An executable ***SysMon*** file will be generated. You can rename this file as you wish and call it in your own script. The log files will be generated at the end of the execution. The monitor will generate a maximum of seven log files: one for the CPU, one for memory and five for the network. See the next section for more information about the content of the log files.

1.3 Monitoring Tool Log Files

The monitoring tool generates a set of log files at the end of the monitoring sequence that contains information about the system resources. We now detail the names of the log files generated and their content.

- **CPU_Log.dat**. Holds a single value corresponding to CPU utilization.

- **MEM_Log.dat**. Shows various memory values in bytes.
  - MemUsage: The amount of memory in use at that time.
  - MemFree: The amount of free memory left.
  - SwapUsage: The amount of swap memory being used.
  - SwapFree: The amount of free swap memory left.

- **NET_MISC_INFO.Log.dat**. Contains overall packet information

- Bytes Rcv'd: The total number of bytes received in a second.
- Pkts Rcv'd: The total number of packets received in a second.
- Bytes Trn'd: The total number of bytes transmitted in a second.
- Pkts Trn'd: The total number of packets transmitted in a second.

- **NET_MISC_EXT.Log.dat**
  - Sockets Used: The total number of sockets in use
  - TCP inuse: The number of TCP sockets currently in use
  - UDP inuse: The number of UDP sockets currently in use.
  - RAW inuse: The number of RAW sockets currently in use.
  - Def Rcv Wnd: The default Receive_window size.
  - Max Rcv Wnd: The maximum receive window size.
  - Def Snd Wnd: The default Send_window size.
  - Max Snd Wnd: The maximum send window size.
  - Mem Rcv TCP: There are three values under this title which indicate memory allocated for the TCP receive buffer.
  - Mem Snd TCP: These three values indicate the memory allocated for the TCP transmission buffer.

- **NET_IP_INFO.Log.dat**
  - tot pkts rcv: Total packets received
    - ivld hdrs: With invalid headers
    - frwrd : Forwarded
  - pkts discd: Incoming packets discarded.
  - pkts dlvd: Incoming packets delivered.
  - rqst snt: Requests sent out.
  - frgm drppd: Fragments dropped after timeout.
  - rss mbrqrd: Reassemblies required.
  - pckts ok: Packets reassembled ok.
  - pckt failed: Packet reassembles failed.

- **NET_TCP_INFO.Log.dat**
  - actv conn: Active connections openings.
  - pasv conn: Passive connections openings.
  - failed conn: Failed connection attempts.
  - conn rsts: Connection resets received.
  - conn estb: Connections established.
  - sgmts rcvd: Segments received.
  - sgmtssnd: Segments send out.
  - sgmts rtrns: Segments retransmitted.
  - bad sgmts rcv: Bad segments received.
  - rsts sent: Resets sent.

- **NET_UDP_INFO.Log.dat**
  - Pkts rcvd: Packets received
  - Pkts unknown: Packets to unknown port received.
  - Pkt errors: Packet receive errors.
  - Pkts sent: Packets sent.

In the current implementation, the information collected is limited by the available RAM. The basic CPU utilization measurement technique in Linux is sampling-based, meaning the monitor measures the CPU load every 10 milliseconds (the sampling interval). In other words, the system checks whether a process is running on the CPU every 10 milliseconds. If a non-idle process is running at the start of a sampling interval, the monitor assumes that the CPU was busy throughout the last 10 milliseconds due to this process. Such a level of granularity is sufficient for most monitoring applications. To get a good estimate for the CPU utilization, we can compute the average over several measurements in a 30 to 60 second time interval (e.g. a 60-second interval implies an average of 6000 measurements). CPU utilization at the process level, however is likely to be more error prone (e.g. a process which exists for less than 10ms and thus consumes less than 10ms of CPU time, will be charged 0ms or 10 ms of CPU time).

1.4 Contact Information

Felix G. Hamza-Lup
School of Computer Science
University of Central Florida
Orlando, FL, 32816
E-mail : fhamza@cs.ucf.edu
Web : www.cs.ucf.edu/~fhamza

1.5 Copyright and Disclaimer

CHAPTER 2 - DISTRIBUTED VIRTUAL ENVIRONMENTS

2.1 Virtuality Continuum

Figure 1 illustrates how real and virtual worlds can be combined in different proportions.

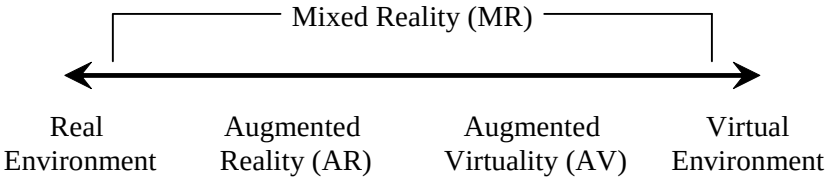


Figure 1. Virtuality Continuum

Mixed reality (MR) is the open interval between the real world and a completely virtual world, or virtual reality (VR). Therefore, to capture the entire spectrum of applications that involve environments containing virtual components we refer to MR/VR environments as Virtual Environments (VEs). A distributed virtual environment is an environment where multiple users participate within the environment from separate computers (remote or local) and the computations involved with presenting the environment are distributed accordingly to each participant.

2.2 Dynamic Shared State

In distributed VEs, a critical component is the dynamic shared state. The dynamic shared state represents the changing information that multiple systems must maintain about a distributed VE.

A node (computing system) in a distributed VE manages several resources (e.g. position tracking sensors, haptic devices and peripherals). Each resource consumes computing resources. Besides the communication delay, additional factors that must be considered for shared state management are the systems delays. If the VE is built on top of a homogeneous system (i.e. each participant to the environment runs similar hardware and

software) the computational loads for each cue (i.e. visual, haptic, and auditory) are nearly identical. These computational loads will not be exactly the same since the complexity of the shared scene might be slightly different for each participant.

On the other hand, if the distributed VE is deployed over a set of heterogeneous nodes (i.e. significant differences in the hardware exists), large differences in the rendering (visual, haptic, audio) times could occur. Such differences will negatively affect the shared state in the environment and the participant interactivity. To ameliorate this problem, a preliminary calibration can be performed by monitoring each system node for a short period of time. The information gathered can be used to adapt the application execution on each node based on the resources available (e.g. the resolution of the 3D models rendered by the node can be decreased automatically if the node's resources are limited).

2.3 Monitoring System Architecture

For the proposed monitoring tool, we have chosen the "proc" file system, a very useful feature of the Linux operating system (Danesh and Das 2000). Similar features are available in other operating systems, however, controlling/accessing these features can be more cumbersome. The "proc" file system is a virtual file system, i.e. it is not associated with a block device but exists only in memory. Such a feature allows us to reduce the intrusiveness of the monitoring scheme by avoiding costly I/O operations.

The Linux kernel has two primary functions: to control access to physical devices on the computer and to schedule when and how processes interact with these devices. The "proc/" directory contains a hierarchy of special files that represent the current state of the kernel. This allows system observation from the perspective of the kernel. Within the "proc/" directory a wealth of information exists, detailing the system hardware and any processes currently running. In addition, some of the files within the "proc/" directory tree can be manipulated by users and applications to communicate configuration changes to the kernel.

For a participating node in a distributed VE, we developed an initial implementation that allows real-time data monitoring of several system parameters (e.g. CPU load, Memory load and Network communication load). A preliminary implementation of the monitoring module consists of four classes: *dareCpuMonitor*, *dareMemMonitor*, *dareNetMonitor* and *dareMonitor*. The monitoring module is part of a larger research effort called DARE (Distributed Artificial Reality Environments) (Hamza-Lup 2002).

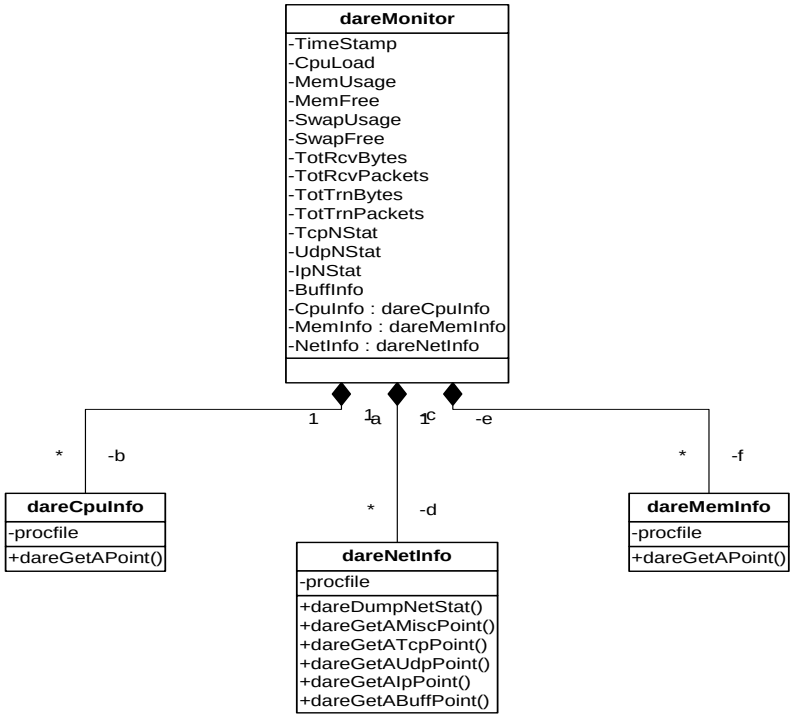


Figure 2. Class Diagram

The C++ implementation allows instantiation of a "*dareMonitor*" object in conjunction with the VE application. On each participating node in the distributed VE, the collected information is stored in memory during the monitoring period. At the end of this period, the memory contents are dumped to log files for further analysis.

The log files are time-stamped at each node. Accordingly, the clock drift of each node from a reference time can be computed using the Network Time Protocol (NTP) (Mills

1994; Mills 2004). A preliminary time synchronization with millisecond accuracy can also be performed concurrently on each node using the following script:

```
for ((i=0; i<=counter; i++))
do
        /usr/sbin/ntpdate rolex.usg.edu        // Call to common time server
        echo "Sync signal sent / received"
        sleep 1
done
```

The "counter" variable can be set to 10. However, higher values can be used for unstable networks. The script above synchronizes the nodes internal clock using the NTP protocol and allows also clock drift value computation.

CHAPTER 3 - USING THE MONITORING TOOL, AN EXAMPLE

3.1 Testbed - Hardware Components

We have performed a set of experiments within a distributed Augmented Reality (AR) environment, and collected data from six heterogeneous nodes. The configuration is illustrated in Figure 3.

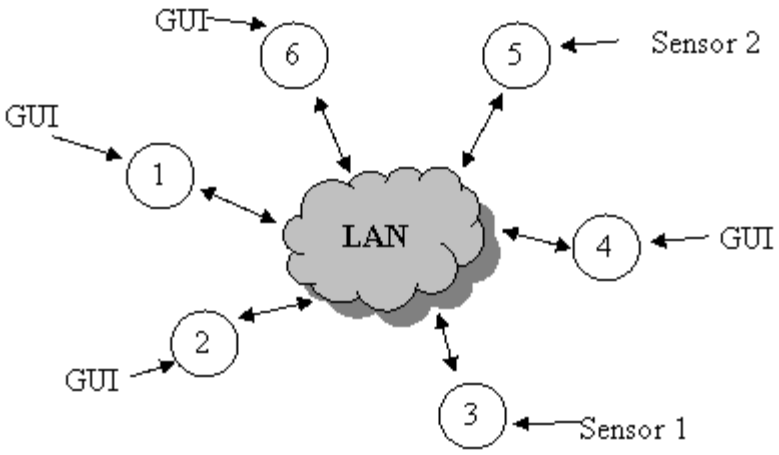
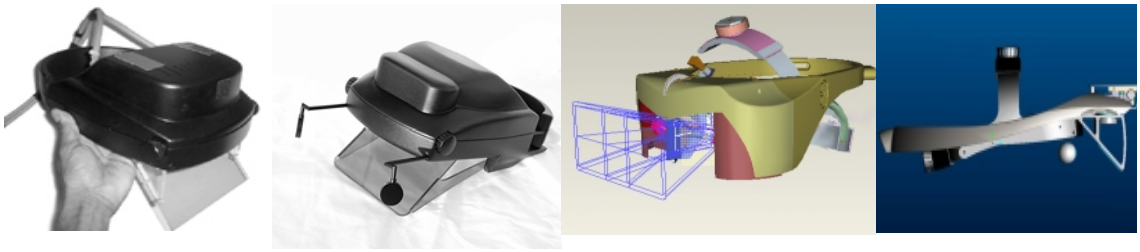


Figure 3. A distributed VE consisting of 6 participating nodes

Each node in the system consists of a Head Mounted Display (HMD), a Linux based desktop/laptop, and a quasi-cylindrical room, called an Artificial Reality Center (ARC), having walls covered with retroreflective material.

3.1.1 HMD & ARC

The HMDs employed are extremely lightweight (< 800g) and utilize compact custom-designed projection optics (< 8g per eye) to provide computer-generated images to the participant within a field of view (FOV) that may be as large as 90 degrees. The displays were designed for indoor settings. Figure 4 shows the HMD technology evolution since 1999, when it was first conceived (Rolland, Biocca et al. 2005).



(a) **(b)** **(c)** **(d)**

Figure 4. Head-Mounted Displays

(a) First prototype; (b) 2001 AR Display & Face Recording; (c) 2003 Side Mounted AR display; (d) 2004 Ultra-compact model

The first prototype (a) had a relatively low resolution (i.e. 4 arcmin), low brightness due to the use of 640 x 480 pixels backlight LCDs, and a weight of 750g which is high compared to the 8g optics per eye. The second prototype (b) added a teleportal capability (THMD) (Biocca and Rolland 2000), which consisted of about 1 inch convex mirrors mounted side-frontal of the user's head and coupled with temple-mounted lipstick cameras to capture stereoscopic images of the face. A recently engineered more compact system with more compact electronics and side-mounted optics is shown in (c). An even more compact prototype (<600g) based on Organic Light Emitting Displays (OLED) and extremely compact electronics is shown in (d).

The HMDs work in conjunction with Artificial Reality Centers (ARCs). The ARCs are cylindrical rooms (Hamza-Lup, Davis et al. 2002), with walls consisting of a set of panels covered by retroreflective material as illustrated in Figure 5. Several ARC rooms can be interconnected on a network. This setup enables remote collaboration through distributed applications that span the entire virtuality continuum (Davis, Rolland et al. 2003).

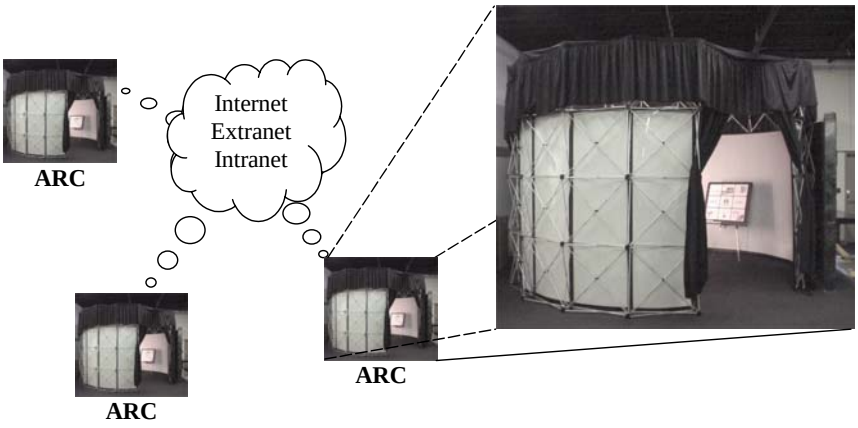


Figure 5. Artificial Reality Centers (ARCs)

3.1.2 Sensors - Optical Motion Tracking Systems

A distributed virtual environment often includes sensors for user interface with the environment. In this specific instance, the sensor used is the Polaris©, an optical motion tracking system built by Northern Digital™. (Polaris 2004) The Polaris is a deployable tracking system that provides accurate orientation and positioning (6DOF) information in real-time. The system has an update rate of up to 60 Hz and is accurate to 0.35 mm in position.

To determine the position and orientation of objects in the virtual environment, tracking probes are utilized with the Polaris. We define a tracking probe as a rigid configuration of markers, as illustrated in Figure 6, that are attached on the real objects in the scene to determine their pose in 3D space (Davis, Hamza-Lup et al. 2004). The sensors' data is collected by the corresponding node and distributed to all participants in the collaborative session. The position and orientation of the virtual objects in the scene can be directly determined by the position and orientation of the real objects.

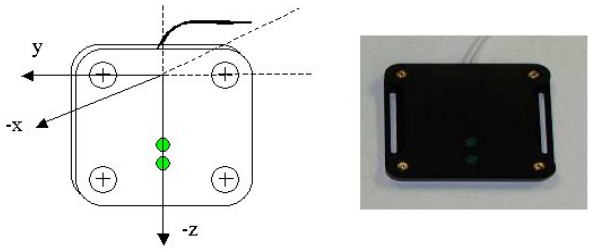


Figure 6. Tracking probe with 4 active markers

The *position sensor* shown in Figure 7 detects the position of the tracking probe while in its tracking volume. The tracking volume has a conical shape with height 1.5 meters and a base radius of 0.5 meters. Newer versions of this system provide a position sensor with a larger pyramidal tracking volume. The position sensor has two infrared cameras that detect the position of the active markers (IRED) or the position of the passive markers by reflection from an infrared source.

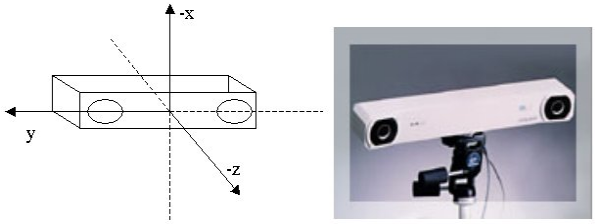


Figure 7. Polaris position sensor

3.1.3 Distributed System Nodes

The nodes in the system are composed of desktops and laptops. The participants at nodes 1,2,4,6 are interacting on the shared scene through a Graphical User Interface (GUI) while the participants at nodes 3 and 5 interact through sensor-based devices. The interaction results in position and orientation changes for the virtual objects in the shared scene. The table below contains a brief specification of each node's hardware components.

Table 1. Hardware Systems Attributes

Node No.	Arch.	CPU (GHz)	RAM (MB)	GPU (GeForce)	Network Card (10/100 Mbps)
1	Desktop	1.5 AMDx2	1024	4 Ti 4600	3Com 3C920
2	Desktop	1.7 AMDx2	1024	4 Ti 4600	3Com 3C920
3	Desktop	1 AMD	512	2 Mx	Netgear FA310 TX
4	Desktop	1.7 Intel	512	4 Mx 440	C.Net Pro200WL
5	Laptop	2 Intel	1024	4 Go 440	3Com 3C920
6	Desktop	2.8 AMD	512	4 Ti 4200	3Com 3C996-BT

3.2 Testbed - Software Components

In terms of software components, the VE distributed application uses OpenGL Performer 2.5 on a Red Hat 7.4 operating system platform. The Graphical User Interface (GUI) was developed using the GIMP Tool Kit (GTK 2.0).

As participants wearing HMDs enter the ARC, (Davis, Rolland et al. 2003), they gradually start immersing themselves in virtuality. Initially, the participant's reality is augmented with 3D computer-generated objects; however, they can be also immersed in a complete virtual setting. The virtual objects may appear to multiple remotely located participants, if they share the same scene. The participants are interconnected on a high bandwidth local area network (100 Mbps).

Participants can interact with the virtual objects in two ways. Using the GUI with 3D pointing capabilities, they can manipulate these objects and they can point in the virtual space to different parts of the objects (e.g., in the experiments we have used 3D medical models of mandibles and several 3D crosses) as illustrated in Figure 8.

Figure 8. GUI, Local collaboration, remote participant.

An alternative way of manipulation is through the motion tracking sensor. The participant holds a tracking probe in his or her hand, and the position and orientation of the tracking probe is associated with the pose of a virtual object in the scene.

A distributed virtual environment experience is created using the following steps:

1. All five users join the interactive VE.
2. The participant at node 1 starts interacting with the virtual objects in the scene using the GUI.
3. The participant at node 2 starts interacting with the virtual objects in the scene through its GUI.
4. The participant at node 3 starts interacting with the virtual objects in the scene using the tracking probe attached to Sensor 1.
5. The participant at node 4 starts interacting with the virtual objects in the scene through its GUI.
6. The participant at node 5 starts interacting with the virtual objects in the scene using the tracking probe attached to Sensor 2.
7. The participant at node 6 starts interacting with the virtual objects in the scene through its GUI.

A preliminary step in this experiment consists of internal clock synchronization and monitoring procedure initiation over a period of time of 25 minutes.

3.3 Results

Below, we have plotted the CPU utilization for each node over a 25-minute monitoring interval. An average CPU utilization is computed offline over each 30-second interval (and may contain up to 3000 measurements)

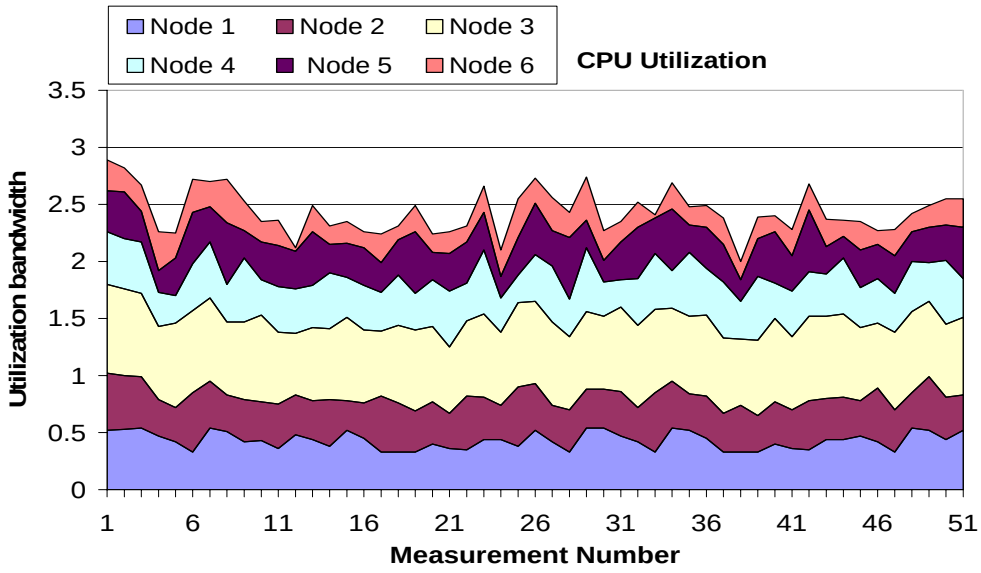


Figure 9. CPU utilization for the distributed system's nodes

As seen in Figure 9, node 6 CPU utilization band is narrower than node 3 utilization band. This is due to the fact that the processing power of node 6 is superior to that of node 3. Moreover a large amount of the graphical rendering burden on node 6 is absorbed by the GeForce 4 GPU.

CHAPTER 4 - CONCLUSIONS

We have presented a method to collect information from a node participating in a Virtual Environment experience using the underlying operating system calls. The intrusiveness of such a monitoring system is maintained at a low level through the avoidance of costly I/O operations.

Since the monitoring module will run on the same system as the application being tested, there will be a limit to how fast data points can be collected. Currently the underlying OS infrastructure provides a CPU monitoring sampling rate of 100 Hz (10 milliseconds). Averaging the utilization over brief intervals of times (i.e. 30-60 sec) an accurate representation of the CPU utilization can be obtained.

We are in the process of extending the monitoring tool to the GPU utilization as this component is under heady load especially in VEs that employ high resolution 3D models.

BIBLIOGRAPHY

Biocca, F. and J. Rolland (2000). Teleportal Face-to-Face system: Teleconferencing and tele-work augmented reality system. USA.

Danesh, A. and G. Das (2000). Linux Process Management and Daemons. Special Edition Using Linux System Administration. B. Walters. Indianapolis, Que**:** 118-141.

Davis, L., F. G. Hamza-Lup, et al. (2004). A Method for Designing Marker-Based Tracking Probes. International Symposium on Mixed and Augmented Reality, Arlington, VA.

Davis, L., J. P. Rolland, et al. (2003). "Enabling a Continuum of Virtual Environment Experiences." IEEE Computer Graphics & Applications **23**(2): 10-12.

Fontaine, J. L. (2004). Modular Object Oriented Dynamic SpreadSheet (MOODSS). SourceForge.Net, SourceForge.Net. **2004**.

Hamza-Lup, F. G. (2002). Distributed Augmented/Artificial Reality Environment. Orlando. **2004**.

Hamza-Lup, F. G., L. Davis, et al. (2002). The ARC Display: An Augmented Reality Visualization Center. International Symposium on Mixed and Augmented Reality, Darmstadt, Germany.

Mills, D. (1994). "Internet Time Synchronization: The Network Time Protocol." IEEE Computer Society(Global States and Time in Distributed Systems).

Mills, D. (2004). The Network Time Protocol (NTP) Distribution. **2004**.

Polaris, N. D. I. (2004). Northern Digital Polaris tracking system.

Rolland, J., F. Biocca, et al. (2005). "Development of Head-Mounted Projection Displays for Distributed, Collaborative, Augmented Reality. Applications." MIT Presence (in press).