



Cluster Profiling

Stéphane Simon
Michel Courson

August 1998 -

NIST Supervisor:
Alan Mink

Abstract

During my trainee period, I worked as a Guest Researcher at the National Institute of Standards and Technology (NIST) from August 1998 until January 1999. I was involved in a cluster profiling project. The project is divided into three phases: data collection and storage, data analysis and automation of the whole process. My internship work covered the first phase of the project: collection and storage of performance data. This report explains the different options that we were offered, our choices and their applications.

Résumé

Lors de mon stage de fin d'étude, j'ai travaillé au National Institute of Standards and Technology (NIST) d'août 1998 à janvier 1999. Je m'occupais d'un projet d'analyse de données de performance d'un ensemble d'ordinateurs travaillant en parallèle. Le projet est divisé en trois phases : collecte et stockage des données, analyse des données et automatisation de l'ensemble du processus. Mon travail couvrait la première phase du projet : collecte et stockage de données de performance. Ce rapport explique les différentes options que nous avons à disposition, nos choix ainsi que leurs applications au projet.

Table of contents

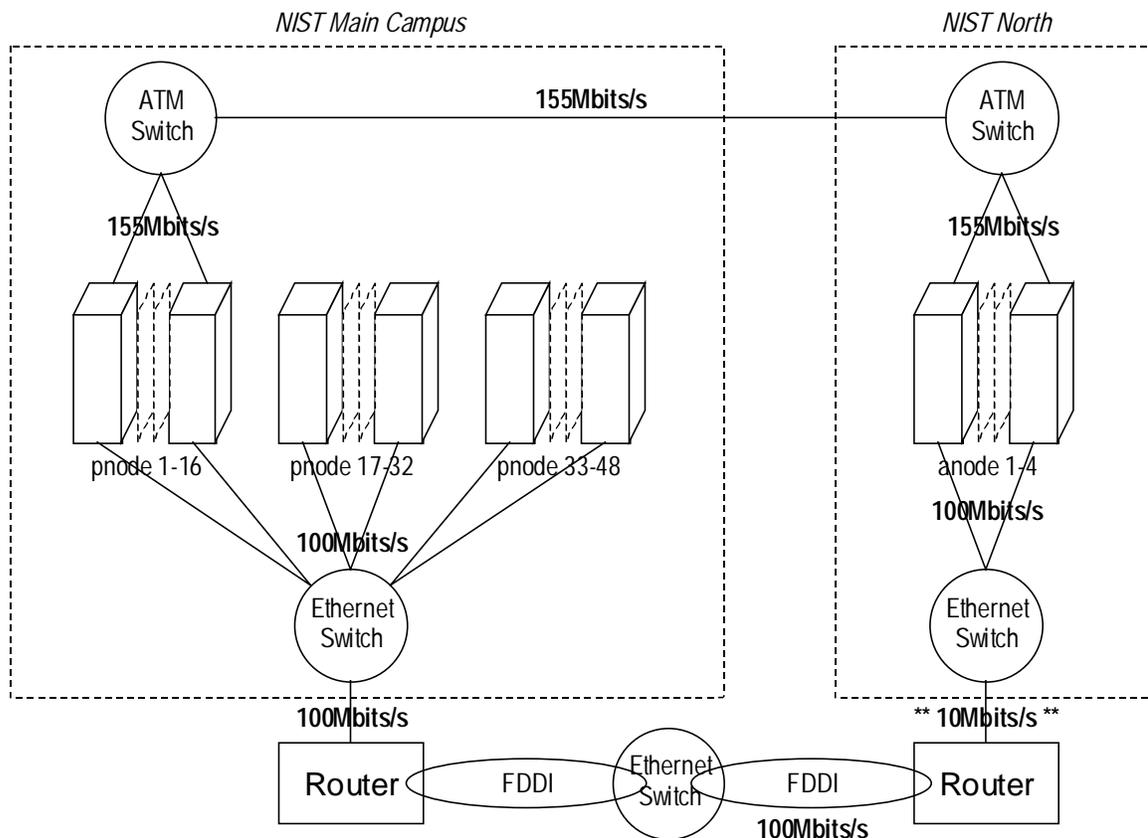
PRESENTATION	4
1 THE CLUSTER PROFILING PROJECT	4
2 PHASE I	5
2.1 Data Abstraction.....	5
2.2 Collection.....	5
2.3 Storage.....	5
3 EXISTING WORK	5
DATA DESCRIPTION	7
1 OVERVIEW.....	7
2 INVENTORY.....	7
3 DATA TYPES.....	8
3.1 Basic Data Types.....	8
3.2 Summary Values and Times Series	8
4 DATA MODEL.....	9
DATA COLLECTION.....	11
1 MECHANISMS	11
1.1 No specific mechanism	11
1.2 Client-server architecture.....	11
1.3 Summary.....	13
2 SAMPLE IMPLEMENTATION	14
2.1 SNMP Libraries	14
2.2 SNMP Manager	15
STORAGE.....	16
1 DATABASE SURVEY	16
1.1 Naïve solutions.....	16
1.2 Object-Oriented Database Management Systems (OODBMS).....	17
1.3 Relational Database Management Systems (RDBMS).....	19
1.4 OLAP and Multi-Dimensional Databases (MDBMS).....	21
1.5 Database Access	22
2 REFERENCE IMPLEMENTATION	24
2.1 General description	24
2.2 Example of an SQL query.....	25
2.3 Customization.....	26
CONCLUSION.....	28
BIBLIOGRAPHY	29
A NIST PRESENTATION	31
1 A BRIEF HISTORY OF NIST	31
2 NIST TODAY	32
3 INFORMATION TECHNOLOGY LABORATORY	33
4 HIGH PERFORMANCE SYSTEMS AND SERVICES DIVISION	34
5 SCALABLE PARALLEL SYSTEM AND APPLICATIONS GROUP.....	34
B SNMP AGENT FILES.....	36

1	MK.H	36
2	MK.C.....	39
3	MULTIKRON-MIB.TXT	45
C SNMP AGENT DOCUMENTATION		51
D SNMP MANAGER SCREENSHOT		56

Presentation

1 The Cluster Profiling Project

The Distributed Systems Technology Group focuses on research and advanced development of measurement techniques for high performance parallel computing, especially cluster environments.



• Figure 1: Cluster network at NIST

Commodity clusters, built around low-cost ordinary PCs, provide an inexpensive and scalable solution for scientific computing. Our group built several experimental PC clusters interconnected with both ATM and Fast-Ethernet, some at different physical locations on the campus, allowing a wide range of network configurations. Based on the Linux operating system, the cluster offers several parallel computing environments, including PVM, MPI (LAM), TreadMarks and traditional Unix applications with sockets. It is used to test the type of scientific applications run at NIST, as well as to evaluate the performance

and the limits of such a platform. One of the clusters was successfully transferred to production in mid-98.

In addition to conventional performance metrics such as system information and software profiling, NIST has developed advanced performance measurement techniques. S-Check [SNE97] is a statistical tool to detect bottlenecks in parallel code. The MultiKron [MIN98] provides hardware, high-resolution (100ns) tracing facility with very low perturbation – the cost of a memory write. In order to use the MultiKron in a distributed environment, hardware instrumentation was developed, using the Global Positioning System (GPS), to synchronize the MultiKron timestamps clocks. The combination achieves global synchronization of all MultiKron boards to within one microsecond worldwide.

The goal of this project is to propose a method to capture an extensive performance profile of the cluster, in order to study its general performance, bottlenecks, possible interactions and many other characteristics. It focuses on thorough analysis of performance data through extensive reuse of existing data and will propose different analysis and visualization schemes for this specific application. The project is divided into three phases: data collection and storage, data analysis and run-time toolkit.

This document discusses the first phase: data collection and storage.

2 Phase 1

The data collection is divided itself into three tasks.

2.1 Data Abstraction

Once an extensive inventory of the known sources of performance data and their properties was prepared, an abstract model was designed to represent the cluster environment in a compact but accurate fashion (section 2).

2.2 Collection

After reviewing the tools available to conduct the actual collection of the data from the different sources, a simple reference implementation was provided to demonstrate the collection process (section 3).

2.3 Storage

Once the data has been retrieved from the nodes, it must be used in an efficient and accessible way to promote sample data reuse. Several popular models were evaluated. One solution was selected and implemented, along with basic management and retrieval tools.

3 Existing work

Many research groups have focused on some aspects of performance evaluation of parallel computing systems in general and clusters in particular.

Several tools provide advanced application-level tracing specifically designed for parallel software. For example, VAMPIR [VAM99], from the Cornell Theory Center is an instrumentation and tracing tool for MPI programs. AIMS [YAN96] from NASA Ames Research Center has similar features. MAD [MAD99] from the GUP Linz in Austria provides event-oriented tracing and analysis. Paradyne [MIL95] features performance tracing down to the procedure and statement level through dynamic instrumentation.

Other tools propose visualization techniques, such as Pablo [AYD96] and Paragraph (part of the AIMS package), Paradyne, Upshot or even PARADE [PAR99] from Georgia Tech that provide advanced navigation tools.

However, as far as we know none of them addresses the storage issue. They may use a proprietary trace files, or one of the public format, such as Pablo's SDDF (Self Defining Data Format), but managing the trace files resulting from multiple experiments is left up to the user.

Data Description

1 Overview

The purpose of this module is to collect distributed performance snapshots of the cluster environment (network, nodes, other devices, application data), to be integrated and stored for later analysis. In this section, we list the major data classes available from the cluster, then we describe a simple, extensible data model to represent each snapshot. We also introduce a higher-level data structure to model experiments.

2 Inventory

Extensive instrumentation of a cluster environment can generate a very wide range of data. This section lists some of the most important sources of performance data, which will be formally organized later in this document.

Physical Machine (computer)
Machine description (processor, memory, cache size, operating system).
Process list (ID, command line, CPU and memory usage, time).
Memory information (free, swap, cache misses).
Network information (dropped packets, output of tcpdump or netperf).
Load.
Status and data of MultiKron (clock, flags, timestamps), GPS (time and date, position, status) and other devices.
Process
Process ID
Response time
Current usage (memory, CPU, swap, interrupt counts, I/O)
Application trace
User-defined trace (usually counters, timers and flags)
Compiler options
Compiler type
Version and General configuration
Gprof data: {function name, CPU, I/O times, percentage of runtime, call count}
Communication Libraries
Data transfers
Number of calls
Average message size
Current Receiving and Sending parties
Timing information
Network device
Configuration

Dropped packets
ARP Tables
Virtual machine
List of machines
Configuration and status
Queuing System / Resource Manager
Configuration
Status
Waiting time
Queue information

3 Data Types

3.1 Basic Data Types

All the information described above uses any one of these basic data types:

- Integer;
- Floating point;
- Strings;

Most systems add semantics by:

- Constraining these types, for example small 16-bit integers, signed and unsigned numerical values, limited ranges (such as a percentage, an integer in [0:100]), or even dynamic constraints (for example SNMP counters that can only be incremented);
- Providing arrays of the above;
- Supporting declared subtypes, such as time ticks (unsigned long integers counting tens of milliseconds).
- Supporting compound types such as dates.

3.2 Summary Values and Times Series

Two different types of metrics lead to different abstraction models to describe them accurately and efficiently:

- One of the basic or compound data types described above can directly represent single value metrics, or summary data, such as response time or number of messages sent.
- Many metrics however consist of consecutive readings over the course of the experiment and need a special representation for time-series. In order to limit the storage burden for very long experiments, we decided to use the Paradyn's data abstraction for time histograms [MIL95]. The number of samples is fixed and the granularity (sampling rate) grows as the experiment runs longer.

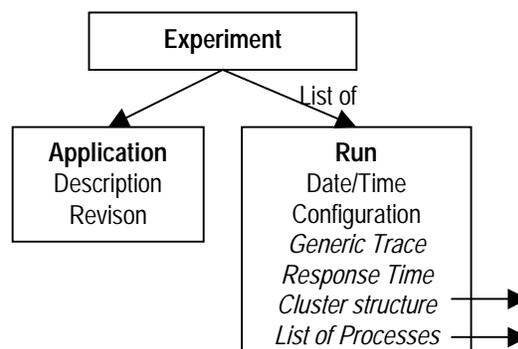
“Paradyn stores performance data internally in a data structure called a time histogram. A time histogram is a fixed-size array whose elements (buckets) store values of a metric for successive time intervals. Two parameters determine the granularity of the data stored in time histograms: initial bucket width (time interval) and number of buckets. [...] If a program runs longer than the initial bucket width times the number of buckets, we double the bucket width and re-bucket the previous values. The change in bucket width (time interval) can cause a corresponding change in the sampling rate for performance data, reducing instrumentation overhead. This process repeats each time we fill all buckets. As a result, the rate of data collection decreases logarithmically, while maintaining a reasonable representation of the metric’s time-varying behavior.”¹

4 Data Model

A data model is an abstraction that represents reality. It encompasses both abstract data structures and mechanisms to describe the relationships within the data, while leaving out extraneous details. A cluster environment and its sensors generate highly hierarchical and mostly independent data that fits nicely in a tree structure. The data model below also introduces abstract objects such as network nodes or hardware devices. It also makes a distinction between category values (settings, options, which are more or less fixed) and summary values (measured data, usually the target of statistical operations). This classification is not rigid, however, as what one user may consider fixed (e.g. the number of nodes) may not be for another (e.g. for a scalability test).

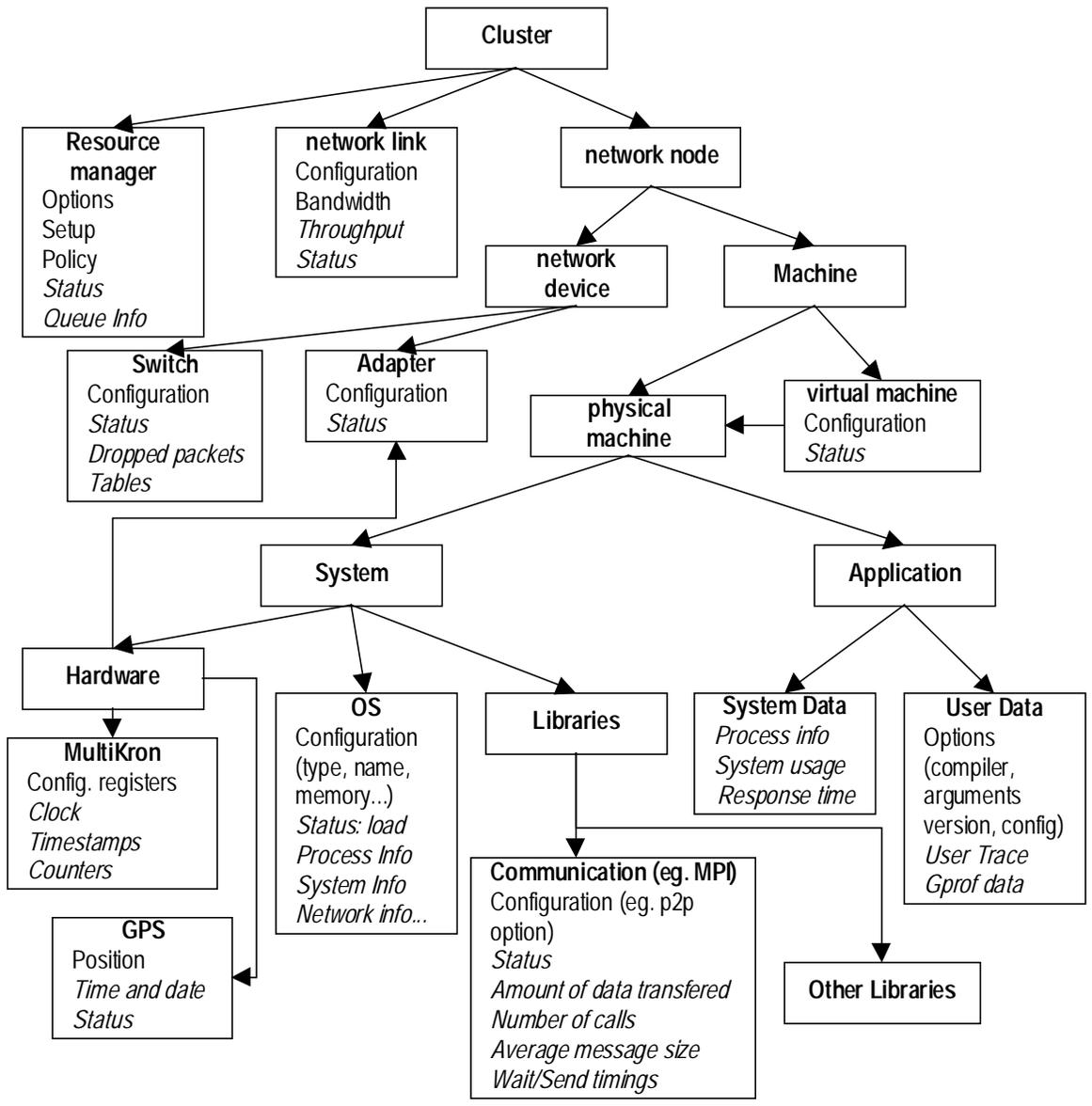
The cluster model (figure 3) accurately holds a snapshot of the whole cluster configuration during a measurement session.

The actual data to be stored in the database as a result of an experiment is a set of cluster snapshots (*runs*). An extra layer of abstraction (figure 2) is then required to group these runs into a logical structure that describes the *experiment*. This powerful abstraction provides the ability to design *virtual experiments* after the fact by recombining existing runs. This is the key to reusable performance data.



• Figure 2: Experiment model

¹ Quote from [MIL95].



• Figure 3: Cluster model

Data collection

Data collection being one of the main parts of the project, we present in this section different options, from the simplest procedures followed when running an experiment by hand to more complex industrial grade protocols.

1 Mechanisms

In this part, we review different mechanisms available to retrieve data from the sources of our cluster data model (figure 3).

1.1 No specific mechanism

This is the simplest approach. It consists of little shell scripts using the Telnet protocol and directly executing commands (displaying information or results on screen or in a file) or using the FTP protocol to retrieve files. This method needs a lot of managing and parsing of files manually. However, it does not require a lot of initial effort. We can use it quickly without preparation but for each new experiment, we have to start over in the creation of new scripts. In addition, it is not easily extensible.

1.2 Client-server architecture

1.2.1 Rsh

Rsh (remote shell) is a protocol available under Unix and other platforms that allows remote execution of commands or shell scripts on a machine. A daemon runs on each host. In our case, we can use it to retrieve information from different machines to process incoming requests by running commands the same way as above, but remotely. This method is readily available on all Unix systems and easy to implement. However, a major drawback for this mechanism is that a user must have full access to all the machines. This is obviously a security problem. Moreover, on some architectures, including the version of Linux running on the cluster, the rsh daemon tends to hang if called to many times or too often.

1.2.2 Custom-made server

This option requires a lot of effort initially although it is somewhat easy to develop. It requires the development of two applications:

- a server application running on each node;
- a client application that gathers data from each server.

Since everything is specifically designed for this application, the server can provide all the required functions in a compact and efficient way. This option leads to a generic and

extensible tool. There are no extraneous features, reducing the server footprint (and therefore potential perturbation) to a minimum. In addition, the server may implement security mechanisms, such as passwords or encryption, to protect the data and restrict access. It will also probably perform better than the mechanisms described earlier since all the transactions can be performed over a single connection. However, some nodes (e.g. a switch) cannot run such a server. Moreover, interoperability requires some form of network standard data representation, such as ASN1 or XDR to transfer the data, which adds to the initial development cost.

1.2.3 Standard interfaces

Standard interfaces offer accrued interoperability for a fraction of the initial development cost of a custom-made server.

FTP

The File Transfer Protocol provides easy data transfer with a basic security scheme (username, password and limited file access). This protocol supports multiple simultaneous sessions allowing faster retrieval of files; there is no need to repeatedly open and close connections. However, FTP is not available on all the nodes although some network devices do support TFTP, a simple non-secure alternative to FTP.

All transfers are file-oriented, so an FTP-based collection device has access to:

- The results of Unix commands or combination of those using “pipes”, this solution leads nearly to the “rsh” one without its security issue.
- The ones found in the /proc tree of some Unix systems, they are updated in real time by the operating system giving general information about the machine and various kernel modules.
- Files created and possibly updated by a running daemon on the remote machine. This option adds extensibility and is quite simple to implement.

FTP itself is a passive solution: it can only retrieve files. Their content must be updated by other agents, such as the kernel, a daemon or an application.

HTTP

The Hyper Text Transfer Protocol, used in web sites, gives the same features as FTP in terms of file transfer but with limited security options. On the other hand, it gives access to active content using CGI scripts or Java applets. The data can then be stored as files and/or easily appear (using HTML tags) on any kind of machine having a web browser. Many modern network devices, for example our Ethernet switch, have an HTTP interface.

SNMP

The Simple Network Management Protocol gives the user the capability to manage a remote network node by setting values and monitoring network events. It uses:

- Several agents, which are applications running on each machine to be monitored or managed.

- One (or more) manager that connects to the agents to monitor and manage the devices by getting and setting properties and listening to events.

The properties are data variables representing resources on the SMNP node handled by this agent. The variables are organized in an MIB (Management Information Base), a kind of dictionary ordering them in a tree structure (the MIB tree) that defines them (name, type and description) and arranges them in a logical way.

SNMP provides three main types of requests:

- GET requests to retrieve values from the agent.
- SET requests to update values on the agent.
- TRAP messages from an agent to notify the management station of significant events that have occurred.

Implementing the SNMP standard requires a significant initial development effort (detailed protocol, complex data representation), but there are many tools available in the form of libraries, generic agents and managers, both free and commercial. More importantly, many network devices, such as switches and router, support SNMP management. This gives SNMP the vastest outreach of all the mechanisms evaluated in this document.

The cluster application uses only a limited subset of the features provided by SNMP – traps and setting values are not used. Most agents available on the market do implement all the functionalities, which can lead to a very large agent in memory.

Security is a weakness of the protocol. In the first version of SNMP, all communicating agents are assigned two community strings (one to read and one to write data), as a makeshift for passwords. These strings are sent unencrypted over the network. Virtually anyone can get information and change the configuration of the SNMP nodes. SNMPv2, the second version of the protocol, specifies an authentication service using DES encryption.

1.3 Summary

The following table rates different features important to us for each mechanism described above:

X	Basic solution	Rsh	Custom made server	FTP	HTTP	SNMP
Extensibility	■	■■	■■■	■■	■■	■■■
Security	■■	■	■■■	■■■	■	■
Performance	■	■	■■■	■■■	■■	■■■
Maintenance	■	■	■■	■■	■■■	■■
Size	■■■	■■■	■■	■■■	■■■	■■
Complexity of client side	■	■■■	■■■	■■■	■■■	■■■
Complexity of server side	■■■	■	■	■	■	■
Initial development effort	■■■	■■	■	■■	■■	■■
Tools available	■	■	■	■■	■■■	■■■
Access to all nodes	■■	■	■	■	■	■■■
Interoperability	N/A	■	■	■■■	■■■	■■■

• Table 1: Our ranking of the importance of our feature selection against the various approaches (■=bad - ■■■=good).

Based on this table, we selected SNMP because:

- SNMP is a standard, thus it is available on computers as well as on other network nodes such as switches or routers.
- It is extensible.
- As a standard, it should be available for new hardware in the future.
- Many SNMP tools are already available reducing the amount of work needed.

2 Sample implementation

Once the choice of SNMP was made, we looked for some available agents and managers.

2.1 SNMP Libraries

We first found SNMX from New Line Software, Inc. [ACE99], a free SNMP package using scripting language. It was possible to make little programs, but the source code was not available. Thus, extensibility of the agent was difficult to achieve.

Then, we investigated Carnegie Mellon University SNMP package (CMU-SNMP) [CMU99]. This free software consisting of a library only, still in development and with very limited documentation. However, source code was available for modification.

We then investigated the University of California at Davis SNMP package (UCD-SNMP) [UCD99]. This is an improved version of CMU-SNMP that is enhanced and supported. It comes with utilities performing several SNMP operations. The package is free; source code and detailed documentation are available, making it easily extensible. We selected this package for the reference implementation.

The agent of the package since was our focus since we want to augment it to implement new sensors. Written in C language, its modular structure allows new sensing modules, using the provided templates.

2.2 SNMP Manager

As an added feature to using SNMP for our collection infrastructure, any SNMP manager can monitor and manage the cluster. Most managers have a graphical interface to display the status and properties of each machine on one single screen. This feature was not an essential part of the project but could help in managing the machines.

The UCD-SNMP package provides a robust and easily extensible agent but only a command-line manager. We then tried to look for a graphical SNMP manager with customization possible for new sensors on the agents. We evaluated several commercial managers with a graphical interface and customization capability to support the new attributes:

- Net Inspector from MG-SOFT: no customization possible;
- Novell Managewise: overly NetWare-centric;
- SNMPc from Castle Rock is quite customizable but lacks some simple features such as updating at a given rate.

A screenshot of SNMPc is presented in appendix D.

We used a Java SNMP API from AdventNet, Inc. To write portable Java applets and applications.

Storage

Before describing different models available to support the cluster profiling tool, here are the general requirements of any scientific database. A good database system should provide:

- 1) A data model providing the abstract representation of the data structures and semantics.
- 2) A high-level query language to easily access the data.
- 3) Support for concurrent access by multiple users.
- 4) Mechanisms validating the integrity of the data.
- 5) Safe recovery from possible system failures.
- 6) Efficiency.

In this section, we will describe several database models: a flat-file storage system, an object-oriented model and an implementation with a traditional relational database.

1 Database Survey

1.1 Naïve solutions

The simplest database system possible is a set of flat files, each one holding the data from one or several runs, or a subset of it. Recording and storing the performance profile are extremely easy with a simple record-oriented format. Accessing the records, however, is very difficult: even if the record format is well designed, querying the database requires appropriate tools and is most likely to be very slow. Tedious management (removing, sorting, adding records, etc.), and limited extension capabilities make such a simplistic database quite unpractical for a large number of measured experiments.

There are ways to improve this solution.

- A richer file format with embedded structure description can improve data availability. Possible candidates are the Self-Defining Data Format (SDDF) [AYD96] or open standards such as SGML or its subset XML.
- A simple Web site can offer an easy solution to the query issue, by hosting all the data in a standard format (most likely HTML or XML). The site then runs a search engine (e.g. Harvest [HAR96], which supports SGML, HTML and RTF) to index and then help to select records. As described in the Harvest documentation, using meta-data (such as the META tags in HTML) can drastically improve the efficiency and accuracy of the queries. This solution, although somewhat inefficient, is very easy to implement and can lead to fairly interesting results.

1.2 Object-Oriented Database Management Systems (OODBMS)

Object-oriented concepts provide the tools to design a rich data model that naturally matches the structure of data, as well as the abstraction power of inheritance and encapsulation. Object models offer a uniform treatment of a wide variety of data types, including highly complex objects, beyond the standard types available to a traditional database.

1.2.1 Data Model

Our model augments the schema from section 2 (figures 2 and 3) by extending the basic hierarchical structure with strong object-oriented concepts, as illustrated in figure 4.

The cluster is viewed as a set of interconnected nodes. The node class can then be subclassed into a network device (a switch for instance) or a computational node. Note that an Ethernet bus or a token ring share the same structure as a switch. Similarly, we use an abstract "Machine" class that can contain other machines, which is then subclassed into a virtual and physical machine. This schema easily and elegantly describes a virtual machine such as PVM, that runs on top of a set of actual workstations. On the other hand, a Java virtual machine running on a workstation fits in this model too. Similar abstract classes organize several other objects in the system.

1.2.2 Available OODBMS

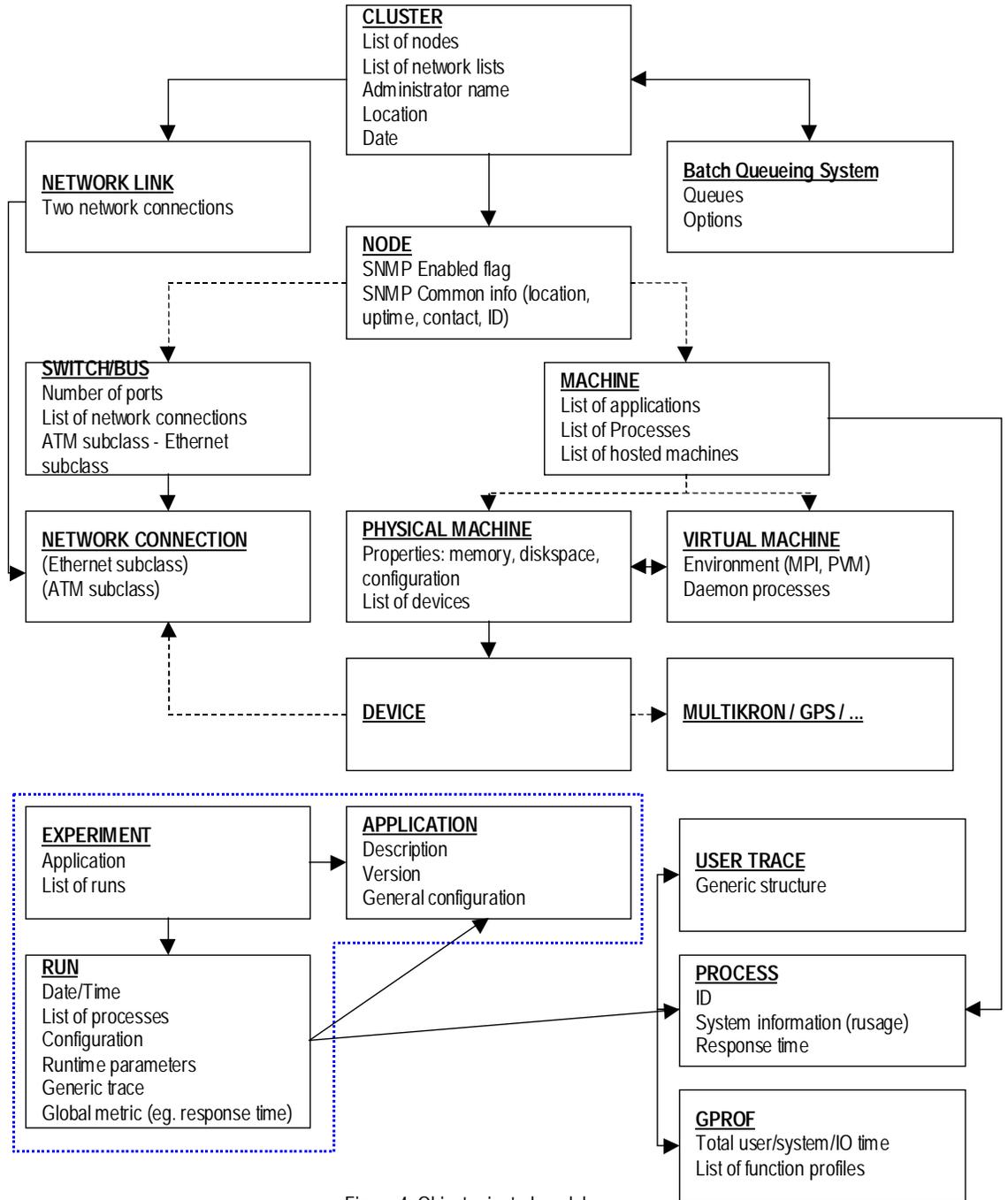
Among the main vendors of object technology, Object Design is the leader with 38% of the object database market, followed by Versant and Objectivity. Softwarebuero distributes an all-Java Linux-based object database, free for non-commercial use.

Vendor	Product	Price	Website
softwarebuero m&b	Ozone	Free	http://www.softwarebuero.de
Object Design, Inc.	ObjectStore PSE/Pro	N/A \$245	http://www.odi.com
Objectivity	Objectivity/DB	N/A	http://www.objectivity.com
UniSQL	UniSQL/X	N/A	http://www.unisql.com
Versant Object Technology	Versant	N/A	http://www.versant.com
Ardent Software Inc.	O ₂	\$5000	http://www.vmark.com/object
Poet Software	Poet	N/A	http://www.poet.com

• Table 2: OODBMS survey

1.2.3 Adequacy

The object model matches the natural structure of data so well that it makes storing the performance traces collected from the cluster a very simple procedure, as virtually no translation is needed. An OODBMS is probably quite efficient, for the same reasons.



• Figure 4: Object oriented model

However, free or open OODBMS are still at an experimental stage and somewhat unstable, while commercial systems are quite expensive. But the main weakness of OODBMS available today is the lack of standard query language. In spite of the efforts of the ODMG [ODM98] for example, who developed OQL (Object Query Language) an extension to SQL92, very few systems were OQL-enabled (at this time, POET and ObjectStore Server).

1.2.4 Example with ObjectStore

ObjectStore PSE Pro for Java from Object Design, Inc. is a Java API that provides mechanisms to create, manage and connect to object oriented databases.

We first need to create the objects. The mapping between the above object model and Java is straightforward. For instance, here is a sample declaration of the machine object:

```
/* Machine.class */

public class Machine extends node {
    private String name;
    private Process processes[];

    public Machine(String name, Process processes[] ) {
        this.name = name;
        this.processes = processes;
    }

    public Machine() {
        this.name = "";
        this.processes = null;
    }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public Process[] getProcesses() { return processes; }
    public void setProcesses(Process processes[] ) { this.processes = processes; }
    public String toString() {
        String s = "Machine:\n" + " name = " + name + "\n";
        s += " processes: " + processes.length + "\n";
        for (int i=0; i<processes.length || i==1; i++)
            s += processes[i].toString();
        return s;
    }
}
```

Then, we need to create an instance of the object. Once the instance is created, we can easily insert data as objects in the database. Indeed, by inserting the top object of the structure, the API inserts all objects reachable from this top object. However, this ease of insertion has a downside: we can query the top object only i.e. the cluster in our example. There is no way, with ObjectStore PSE Pro, to query an object under (in the structure) the top object without recursively searching “by hand” the children. This is not satisfactory in a context where querying is very important.

1.3 Relational Database Management Systems (RDBMS)

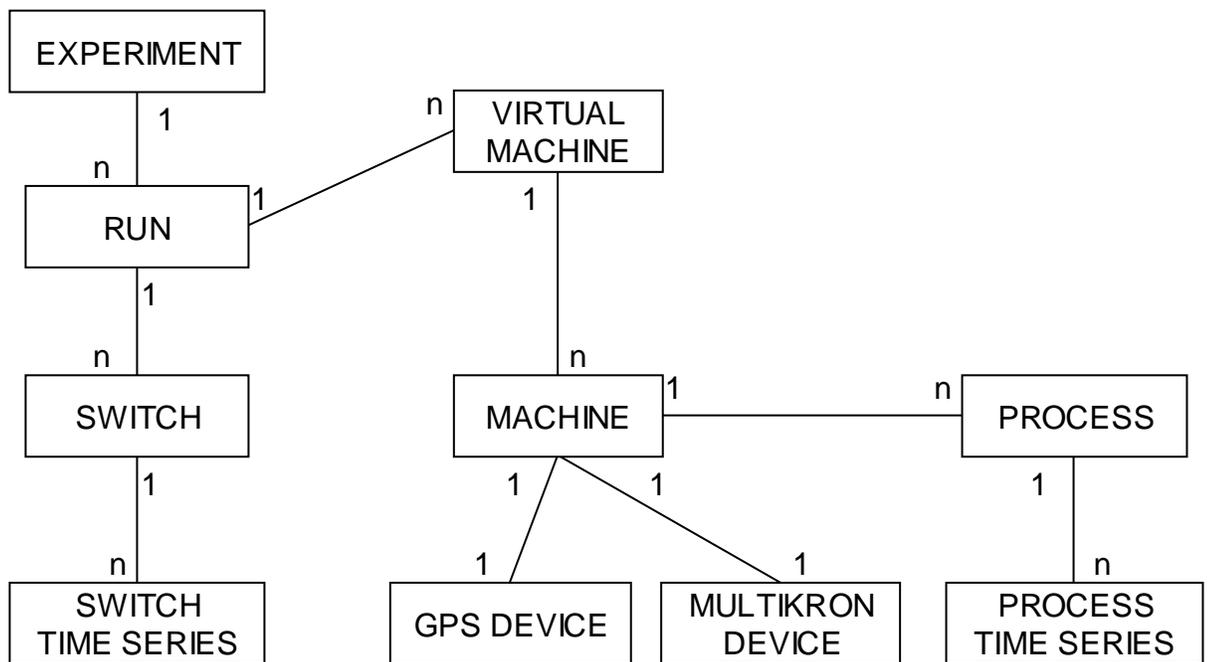
Relational databases are the most popular in the database world. Contrary to object oriented databases where all the data are well structured and logically bound together, relational databases rely on several tables linked with relations.

1.3.1 Model

Description

We used an Entity-Relationship Diagram (ERD) (see [OCC92]) to represent a simplified view (figure 5) of a possible implementation of a relational database for this application. Each box represents a table, the basic entity type in the relational model. Each link between two boxes represents a logical relation between the data in the tables. Three types of relation exist:

- (n:n) “many to many” and not used in our example.
- (1:n) “one to many” which is used for example between an experiment and a run. This is the relation used to represent a list. Indeed, here, an experiment is a list of several runs and one run can belong to only one experiment.
- (1:1) “one to one” relation meaning that, for instance, one machine can have only one Multikron device and that one Multikron device belongs to only one machine.



• Figure 5: Entity-relationship diagram

All those different relations have some consequences in the building of the tables, especially the primary keys (see [OCC92]), which are attributes of tables that make each entity occurrence unique.

Pros/Cons, performance

The relational model is a mature technology, one of the first ones and still widely used. It is simple, efficient and, in general, less expensive than object oriented technology.

This model, contrary to the object-oriented scheme, does not reflect the data structure. This fact leads to two kinds of complexity:

- Complexity at design-time when all the relations and tables have to be defined. As we will see in the sample implementation section, the data model diagram of figure 5 leads to a complex primary key management. However, this complexity is only encountered by the designer.
- Complexity in querying since we will need to make a lot of joins and projections (operations on tables, see [GRU90]) on large tables, resulting in poor performance for large databases. The database management system handles most of the work. The consequence is that querying, which is, in our case, very important for the analysis of stored data, is fairly easy and flexible for the end-user, even though a lot of relations may appear in the data model.

In addition, the primary key management involves some redundancy in the different tables as we will see in the reference implementation.

1.3.2 Available database solutions

Many relational database system products are inexpensive or even free (for non commercial products). We looked for Linux based software.

Commercial products provide, in general, many unneeded features. They are usually more affordable than object oriented solutions, but still expensive. Some free products seem to be sufficient though.

Some relational databases are [SQL98]:

- mSQL (MiniSQL), commercial, inexpensive, query language = subset of SQL, too restricted
- GNU SQL, free (GNU license), in development
- BeagleSQL, free, very much in development, not stable.
- PostgreSQL, free, stable, simple
- MySQL, free extension of MiniSQL, simple use

1.4 OLAP and Multi-Dimensional Databases (MDBMS)

OLAP (On-Line Analytical Processing) is a database software technology that enables analysts to view and access a traditional database as a multi-dimensional structure.

Salesperson	Product	Sale Amount
Bob	Nuts	2000
Mary	Bolts	6000
Bob	Nuts	3000

• Table 3: Traditional relation

	Nuts	Bolts
Bob	2000	3000
Mary	0	6000

• Table 4: Multi-dimensional view

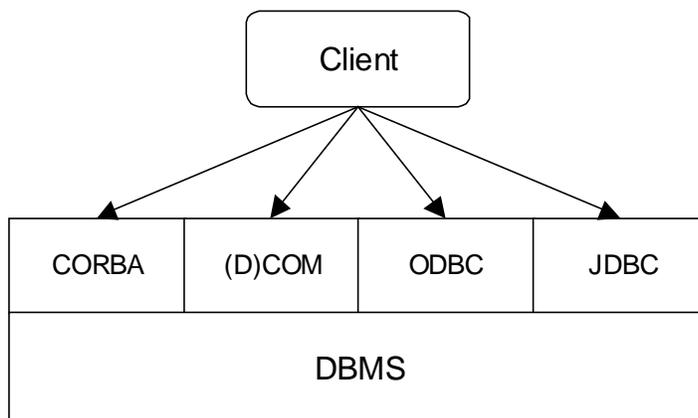
This model, along with the underlying implementation and accompanying tools, allows for fast, natural and powerful data analysis along all the dimensions. It is becoming popular for high-volume trend analysis and data mining of large-scale databases. The data model also includes hierarchies within dimensions and multi-dimensional vector arithmetic and tools (including extensions to SQL). OLAP servers often support time-series (time being one dimension), a very useful tool for trend analysis.

OLAP may be the architecture for the next generation of high-performance business databases. It can be implemented on top of traditional relational (or other) database management systems, although with poorer performance.

OLAP is quite appealing for many applications. Such systems are designed – and priced – for high-volume industrial applications.

1.5 Database Access

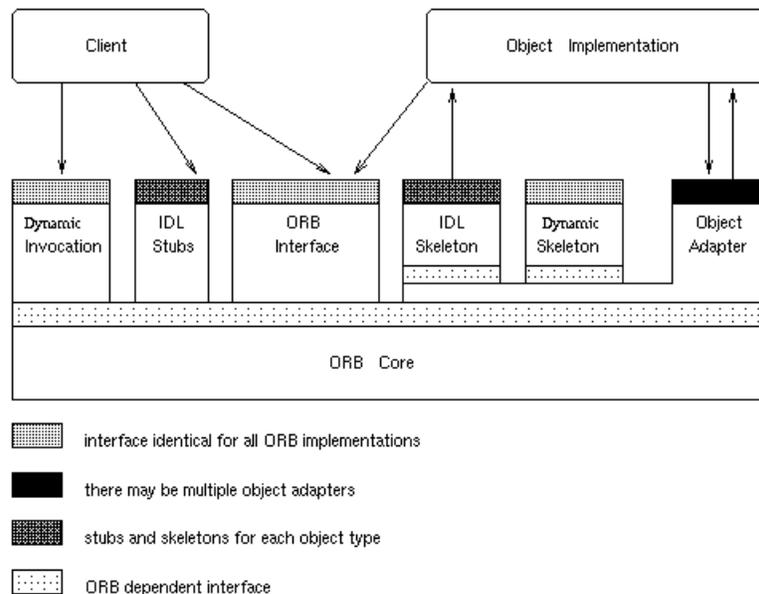
There exist several standard ways to access a DBMS, all of them supported to some extent by leading database system vendors.



• Figure 6: Standard ways to access a DBMS

1.5.1 CORBA

The Common Object Request Broker Architecture (CORBA), developed by the Object Management Group [OMG99], is an open, distributed object infrastructure that allows applications to communicate through a standard object interface provided by an Object Request Broker (ORB). The OMG describes the ORB as “[...] the middleware that establishes the client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object’s interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.”



• Figure 7: CORBA Architecture [KEA97]

There are no free fully compliant CORBA solution but many commercial implementations are available.

The object model described (figure 4) may be implemented with CORBA, although this architecture was designed for distributed and heterogeneous systems; for this application, it adds little benefit to a local Java or C++-based object broker, except that the ORB manages the objects similarly to an object-oriented database. As before, querying, which is essential, is left up to the object implementation.

CORBA, as well as Microsoft’s COM [COM99], provides essential distributed object services but still requires an underlying data management system. Major commercial databases have a CORBA interface.

1.5.2 ODBC and JDBC

JavaSoft's JDBC and Microsoft's ODBC (Open DataBase Connectivity) are both APIs for executing SQL queries. They provide a standard interface to nearly any database on the market. Microsoft's ODBC is the most widely used database drivers but JDBC integrates naturally into Java – a JDBC-ODBC bridge is available for those database systems that do not support JDBC.

Example of JDBC query, from the SunSoft documentation:

```
Connection con = DriverManager.getConnection(
    "jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b FROM Table1");
while(rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
}
```

2 Reference Implementation

2.1 General description

Our choice to implement the data storage solution is the relational model. After having encountered querying problems with object oriented databases, technology that seemed to be best suited for what we wanted to do, we started evaluating some relational database management systems. Our priorities were reuse, access and easy retrieval of stored data. These features were all present in relational database systems whereas the querying in object oriented database systems was not satisfactory. Multidimensional databases are business-strength solutions that do not suit smaller-scale application.

We started the implementation using MySQL [MYS99], free relational database software running under the Linux operating system. We could have also chosen PostgreSQL [POS99] since it has similar features.

A named table is the representation of an entity type in the relational model. Table columns represent attributes of the entity type and each row in the table corresponds to an entity occurrence. The relational model requires that each entity occurrence of each table be unique. This uniqueness is achieved using primary keys: they are embedded in other tables to make each entity occurrence unique and to achieve cross-references between tables.

We had to translate the entity-relationship diagram (figure 5) into an actual database. The sample result, which is not yet complete, is shown in figure 8.

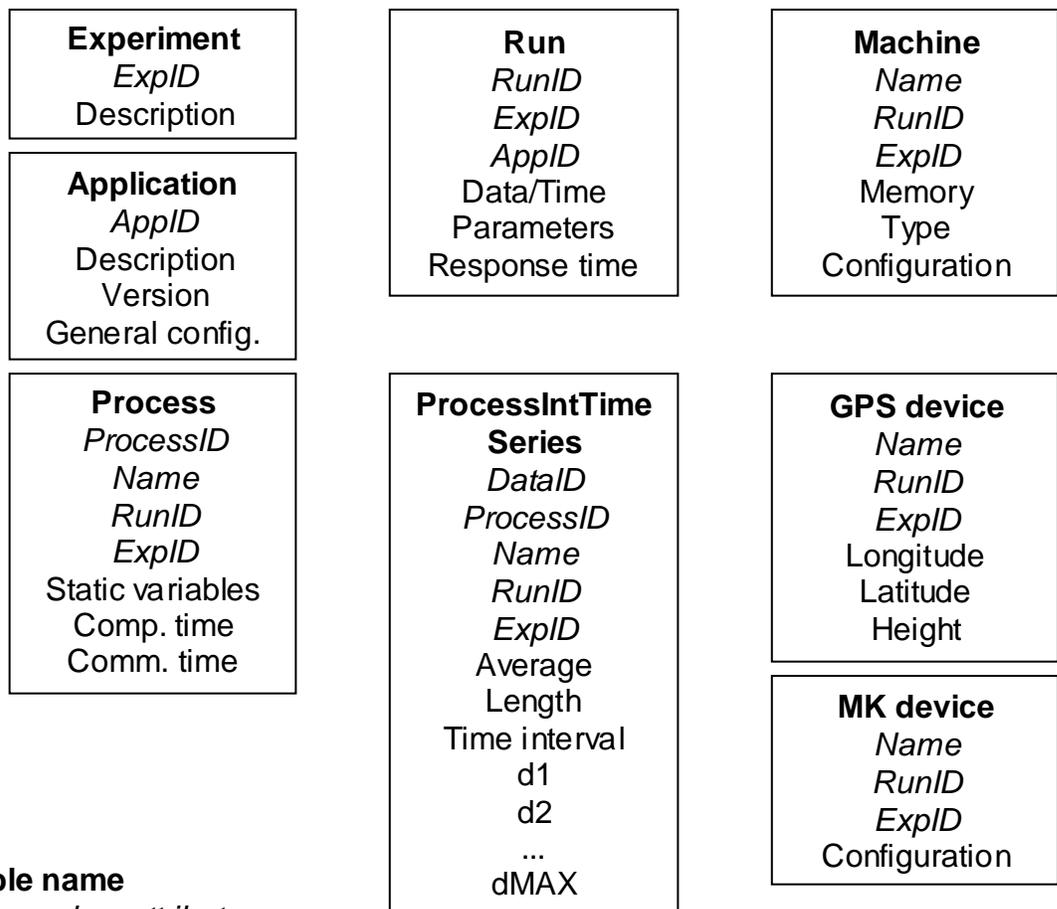


Table name
Primary key attribute
 Attribute

• Figure 8: Relations

The type of data is, commonly, integer, float or string. We notice the redundancy of the model: For each “one to many” relation, the primary key of the content is made of the primary key of the container and a unique attribute of the content itself. For each “one to one” relation, the primary key of the container is the key of the content itself. These are ways to ensure the uniqueness of each record.

Each table in the figure 8 contains attributes: some are parts of the primary key of the table and some are actual data. Some data needs a special type of storage – time series – so we can analyze variations of those data with time. A possible way to implement those time series is to use one specific table for each type (integer, float) of data and for each table that may need time series (Process, Multikron device). This table is then filled the same way that Paradyne [MIL95] fills its time histograms (see above), so that we can store a trace of a long or short experiment using the same amount of memory. In addition, this approach provides, in one single query, all interesting information to plot graphs. This is a significant feature for our application.

2.2 Example of an SQL query

As an example, we have stored data from an experiment of 4 runs, consisting of 1,2,4 and 8 machines with one process on each machine (there can be other processes running on each machine!). In addition, we have gathered the memory usage as a process time series. We want to query the average memory usage for each process of each run. The simple SQL query below provides this information:

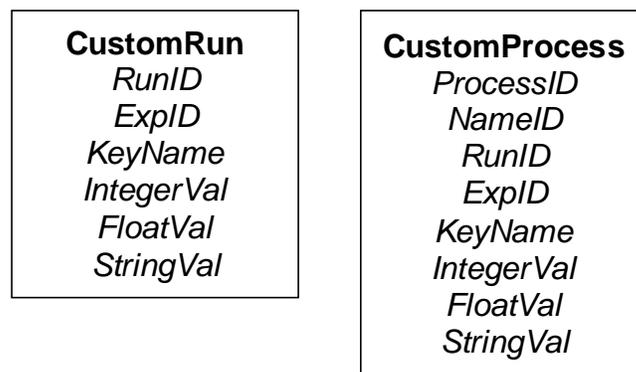
```
Select P.RunID, P.ProcessID, PTS.Average
from Process as P, ProcessIntTimeSeries as PTS
where P.ProcessID = PTS.ProcessID AND P.Name = PTS.Name AND
P.RunID = PTS.RunID AND P.ExplID = PTS.ExplID AND
P.ExplID = 1 AND PTS.DataID = 1
```

*what we want to retrieve
tables that are going to be part of the query
joins between tables
selection of ExplID and DataID (1 = memory usage)*

The result of this query yields 15 different average values. We can see that the database is not too laborious to query although it might require significant work by the database application for large tables.

2.3 Customization

Each kind of performance experiment needs some specialized data, which can be of different types (integer, float and string). To avoid creating one table for each different type, we chose to create one table that included a key name and 3 attributes for integer, float and string values. This facilitates the access to data. 1 attribute out of 3 is used, the others have the NULL value which does not use any extra memory. Run and Process tables may need this technique. Therefore, we can add two tables (figure 9).



• Figure 9: 2 tables added

Example: demonstration of the effect of optimization in compilers

We have stored data corresponding to an application with some CustomRun values for each run such as the number of processes and compiler optimization (e.g. -O2 or nothing with gcc). We want to compare the average response time of all those runs where the number of processes is set. To accomplish this, we make two queries: retrieving the average response time with and without optimization as shown below.

```
Select avg(response time)
from Run as R, CustomRun as CR1, CustomRun as CR2
where R.ExplID = CR1.ExplID AND R.RunID = CR1.RunID AND
```

*selection of what we want
tables part of the query
joins between tables*

CR1.ExplID = CR2.ExplID AND CR1.RunID = CR2.RunID	
R.AppID = 1 AND	selection of application
CR1.KeyName = 'numProcesses' AND CR1.IntegerVal = 2	selection of number of processes = 2
CR2.KeyName = 'optimization' AND CR2.StringVal = '-O2'	selection of optimization

This query should return a single value, the desired average. The other query is the same except the last clause: CR2.StringVal = "".

Connection to the database server

A way to connect to a relational database can be using JDBC (Java Database Connectivity). This is the one we chose for several reasons: portability of Java, existence of JDBC drivers for almost all relational database servers (in case we need to change the DBMS, the Java source codes will stay the same). We are using Mark Matthews' MySQL JDBC Driver Version 0.9e [MYS99] and started implementing some basic tools to query the database.

Conclusion

After establishing an inventory of all the data available on a cluster, we started looking for a collection technique among several different ones. After evaluating each of them, we selected the Simple Network Management Protocol as the most suitable to our application.

Storage required the choice of a database technology. Even though the object oriented model seemed to be well suited to our needs, we chose the relational model because of some problems encountered in our object oriented sample implementation. We then chose a Linux based relational database management system: MySQL.

The data collection and storage phase of the project is now almost complete. However, we did not have time to run experiments and significantly test the database. A collection-storage link can be developed to provide a complete tool, this could be done using the SNMP and MySQL Java APIs.

Bibliography

- [ACE99] Diversified Data Resources, Inc., "ACE-SNMX Scripting Executive" http://www.ddri.com/acesnmp/snmx_info.htm, 1999.
- [AYD96] Ruth A. Aydt, "The Pablo Self-Defining Data Format", <http://www.pablo.cs.uiuc.edu>, 1996.
- [BAK57] Seán Baker, "CORBA Distributed Objects", *ACM Press*, 1997.
- [BAN92] François Bancilhon, Claude Delobel, Paris Kanellakis, "Building an Object Oriented Database, the O₂ experience", *Morgan Kaufmann Publishers*, 1992.
- [CAT91] RGG Catell, "Object Data Management", *Addison-Wesley*, 1991.
- [CAT97] R.G.G. Catell et al., "The Object Database Standard: ODMG 2.0", *Morgan Kaufmann Publishers*, 1997.
- [CMU99] Carnegie Mellon University, "CMU SNMP Library", <http://www.net.cmu.edu/projects/snmp>, 1999
- [COM99] Microsoft, "Microsoft COM, Component Object Model", <http://www.microsoft.com/com>, 1998.
- [DAT96] "OLAP And OLAP Server Definitions", *Datamation*, April 15, 1996.
- [GRU90] Martin Gruber, "Understanding SQL", *Sybex*, 1990.
- [HAR96] Harvest Web Indexing, <http://www.tardis.ed.ac.uk/harvest>.
- [KEA97] Kate Keahey, "A Brief Tutorial on CORBA", <http://www.cs.indiana.edu/hyplan/kksiazek/tuto.html>, 1997.
- [KIM96] Ralph Kimball, "The Data Warehouse Toolkit", *John Wiley and Sons, Inc.*, 1996.
- [MIL95] B.P. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam and T. Newhall, "The Paradyn Parallel Performance Measurement Tools", *IEEE Computer*, 28, 1995.
- [MYS99] T.c.X. DataKonsultAB, "MySQL", <http://www.tcx.se/>, 1999.
- [OCC92] Val Occardi, "Relational Databases: Theory and Practice", *NCC Blackwell*, 1992.
- [OMG99] The Object Management Group, <http://www.omg.org>, 1999.

- [PAR99] Georgia Institute of Technology, "Visualization of Parallel and Distributed Programs", <http://www.cc.gatech.edu/gvu/softviz/parviz/parviz.html>, 1999.
- [POS99] "PostgreSQL Home Page", <http://www.pyrenet.fr/postgresql/>, 1999.
- [SQL98] Linas Vepstas, "Linux SQL Databases and Tools", <http://linas.org/linux/db.html>, 1998.
- [SNE97] Robert Snelick, Mike Indovina, Michel Courson, Anthony Kearsley "Tuning Parallel and Networked Programs with S-Check", *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, Nevada* (June 30 - July 3, 1997), <http://cmr.ncsl.nist.gov/scheck>, 1997
- [UCD99] University of California at Davis, "The UCD-SNMP project home page", <http://www.ece.ucdavis.edu/ucd-snmp>, 1999.
- [VAL89] Patrick Valduriez, Georges Gardarin, "Analysis and Comparison of Relational Database Systems", *Addison-Wesley*, 1989.
- [VAM99] Cornell Theory Center, "VAMPIR", <http://www.tc.cornell.edu/UserDoc/Software/PTools/vampir/>, 1999.
- [YAN96] J. C. Yan and S. R. Sarukkai, "Analyzing Parallel Program Performance Using Normalized Performance Indices and Trace Transformation Techniques", *Parallel Computing* Vol. 22, No. 9, November 1996. pages 1215-1237, 1996
- [ZBI91] Zbigniew Michalewicz, "Statistical and Scientific Databases", *Ellis Horwood*, 1991.



NIST Presentation

1 A Brief History of NIST

The National Institute of Standards and Technology (NIST), formerly the National Bureau of Standards (NBS), was established by Congress in 1901 to support industry, commerce, scientific institutions, and all branches of Government. For nearly 100 years the NIST/NBS laboratories have worked with industry and government to advance measurement science and develop standards.

NBS was created at a time of enormous industrial development in the United States to help support the steel manufacturing, railroads, telephone, and electric power, all industries that were technically sophisticated for their time but lacked adequate standards. In creating NBS, Congress sought to redress a long-standing need to provide standards of measurement for commerce and industry and support the "technology infrastructure" of the 20th Century.

In its first two decades, NBS won international recognition for its outstanding achievements in physical measurements, development of standards, and test methods – a tradition that has continued ever since. This early work laid the foundation for advances and improvements in many scientific and technical fields of the time, such as standards for lighting and electric power usage; temperature measurement of molten metals; and materials corrosion studies, testing, and metallurgy.

Both World Wars found NBS deeply involved in mobilizing science to solve pressing weapons and war materials problems. After WWII, basic programs in nuclear and atomic physics, electronics, mathematics, computer research, and polymers as well as instrumentation, standards, and measurement research were instituted.

In the 1950s and 1960s, NBS research helped usher in the computer age and was employed in the space race after the stunning launch of Sputnik. The Bureau's technical expertise led to assignments in the social concerns of the Sixties: the environment, health and safety, among others. By the Seventies, energy conservation and fire research had also taken their place at NBS. The mid-to-late 1970s and 1980s found NBS returning with renewed vigor to its original mission focus in support of industry. In particular, increased emphasis was placed on addressing measurement problems in the emerging technologies. Many believe that the Stevenson-Wydler Act implemented, throughout the federal laboratories, the practices that had been developed at NBS over the years: cooperative research and technology transfer activities.

The Omnibus Trade and Competitiveness Act of 1988 – in conjunction with 1987 legislation – augmented the Institute's uniquely orchestrated customer-driven, laboratory-based research program aimed at enhancing the competitiveness of American industry by

creating new program elements designed to help industry speed the commercialization of new technology. To reflect the agency's broader mission, the name was changed to the National Institute of Standards and Technology (NIST).

These efforts, and the organizational changes brought by the NIST Authorization Act for 1989 which created the Department of Commerce's Technology Administration to which NIST was transferred, served as a critical examination of the role of NIST in economic growth. These mission and organizational changes, initiated under the Bush Administration were reaffirmed and strengthened by the Clinton Administration.

In addition to the reviews by Congress, the Administration, and the Department of Commerce, the Visiting Committee on Advanced Technology (VCAT) of NIST reviews and makes recommendations regarding the general policy, organization, budget, and programs of NIST. The VCAT holds four business meetings each year with NIST management, and summarizes its findings each year in an annual report that is submitted to the Secretary of Commerce and transmitted by the Secretary to Congress.

As an agency of the U.S. Department of Commerce's Technology Administration, NIST's primary mission is to promote U.S. economic growth by working with industry to develop and apply technology, measurements, and standards. It carries out this mission through a portfolio of four major programs:

Measurement and Standards Laboratories that provide technical leadership for vital components of the nation's technology infrastructure needed by U.S. industry to continually improve its products and services;

- a rigorously competitive Advanced Technology Program providing cost-shared awards to industry for development of high-risk, enabling technologies with broad economic potential;
- a Manufacturing Extension Partnership with a network of local centers offering technical and business assistance to smaller manufacturers; and
- a highly visible quality outreach program associated with the Malcolm Baldrige National Quality Award that recognizes business performance excellence and quality achievement by U.S. manufacturers, service companies, educational organizations, and health care providers.

2 NIST today

BUDGET

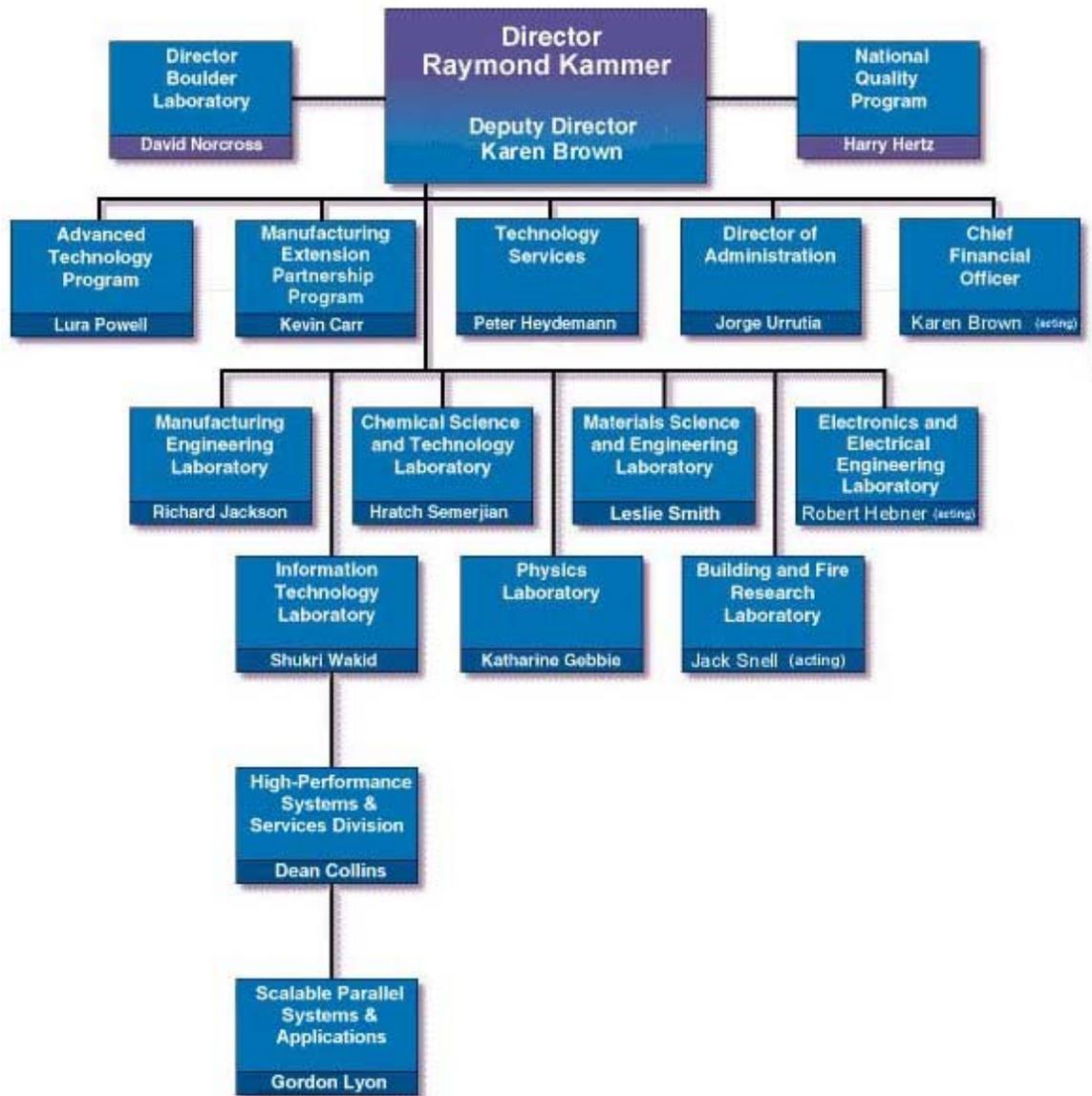
\$760 million (FY 1999 estimated operating resources from all sources).

STAFF

About 3,300 scientists, engineers, technicians, and support personnel, plus some 1,200 visiting researchers each year.

SITES

Gaithersburg, Maryland (headquarters – 234-hectare campus) and Boulder, Colorado (84-hectare campus).



• Figure 10: NIST Organizational Chart

3 Information Technology Laboratory

About 60 percent of all U.S. workers have jobs that depend on the information they generate and receive on advanced information networks. Innovations in computer hardware, software, and digital communications will challenge managers to apply new technology for productivity increases throughout the American economy in the coming years and will have pervasive effects on industry structure as well as on the quality of government services. Also, since computers will be distributed throughout society, both at home and in the workplace, computer integration, interoperability, usability, reliability, and security will become even more important.

NIST's Information Technology Laboratory (ITL) concentrates on developing tests and test methods for information technologies that are still in the early stages of development--long

before they are available in new products. But even once information technology products are available, tests developed by ITL provide impartial ways of measuring them so developers and users can evaluate how products perform and assess their quality based on objective criteria.

Technical research, industry collaborations, and standards-related work of the ITL address issues emerging from today's information revolution.

4 High Performance Systems and Services Division

The High Performance Systems and Services Division (895) of the Information Technology Laboratory enables effective application of high performance computing and communications systems in support of the U.S. information technology industry and NIST by:

Conducting research, development and evaluation of advanced hardware and software components, new architectures, novel application technologies, and innovative measurement and test methods for improved computing performance, scalability, functionality, interoperability, flexibility, reliability and economy;

Serving as a testbed for R&D in high-performance computing and information technologies such as embedded computing, displays, and data storage, gaining experience in the deployment of these technologies, and developing metrics for the representative technologies;

Serving as a responsive, effective mission-critical resource spanning computational, communication, mass storage, security, archival, and scientific visualization services; and

Providing and managing state-of-the-art computing and networking facilities which integrate and support an enterprise-wide heterogeneous information technology environment for NIST.

5 Scalable Parallel System and Applications group

This is the group where I worked, directed by Gordon Lyon. I worked under Dr. Alan Mink supervision on the Distributed Systems Technology project (figure 10: NIST organizational chart).

Scalable parallel computing, in which problems are broken into parts and solved simultaneously by many processors rather than sequentially by one processor, constitutes a growing fraction of high end computing. Yet the technology is still immature; large-scale parallel programming is not easy and system features are not firmly established. Parallel architecture is also a technology bellwether – it now appears widely in the form of systems of networked workstations as well as in less expensive workstations. Adapting the process to work on different kinds of hardware is difficult.

The NIST Distributed Systems Technology project is investigating performance measurement that promotes High-Performance Computing and Communications scalable systems. Its projects focus on:

- **HARDWARE:** In cooperation with DARPA, NIST has designed and built the MultiKron VLSI chip series and support boards for measuring the performance of advanced processors and very high speed networks. Such accurate measurement permits researchers to understand the source of performance bottlenecks and therefore learn how to scale their system designs upwards. Intel Paragon systems incorporate concepts from MultiKron.
- **SOFTWARE:** With DARPA support, NIST has devised a novel measurement technique Synthetic Perturbation Screening and developed it into a software tool, S-Check, for assaying processing bottlenecks in parallel code. Besides solving code bottleneck assessment in a sound mathematical manner, the tool's powerful measurement principles are independent of any particular parallel system, language or computational algorithm.
- **NEW RESEARCH:** While advent of LAN-clustered workstations promises flexible, inexpensive and scalable computing, these dividends today elude most scientific users. The attractions of networked workstation architectures are abundantly clear: incremental upgrading, personal, custom immediacy, large scale recovery of mostly unused cycles in institutional workstations and tantalizingly low compute costs. The state-of-the-art offers a unique opportunity to explore networked workstations in the context of more advanced, moderately-priced interconnection schemes such as ATM (asynchronous transfer mode) and ever-swifter RISC (reduced instruction set computer) workstations. NIST has started the Scalable Computing Testbed Project to explore such issues.

Both NIST hardware and software approaches address current concerns in achieving the economic promise of cheap massively parallel systems.



SNMP Agent Files

The following files are part of the modules added to the UCD-SNMP package, in order to add new sensors in each agent. The sample files concern Multikron support only since they are based on a common template file.

1 mk.h

This file is based on a template provided in the UCD-SNMP package. The end of the file is Multikron specific declarations used to access the chip and has been cut to save space.

```
/* mk.h - multikron module for the SNMP agent */
/* Handles enterprises.6478.65 requests */

/* Don't include ourselves twice */
#ifndef _MIBGROUP_MK_H
#define _MIBGROUP_MK_H

config_require(util_funcs);

/* Adding the ASN1 description of the multikron module */
config_add_mib(MULTIKRON-MIB)

/* Define all our functions using prototyping for ANSI compilers */
/* These functions are then defined in the mk.c file */

long toto;
void init_mk();
u_char *var_mk __P((struct variable *, oid *, int *, int, int *,
int (**write) __P((int, u_char *, u_char, int, u_char *, oid *, int))
));

/* Magic number definitions. These numbers are the last oid index
numbers to the table that you are going to define. For example,
lets say (since we are) creating a mib table at the location
.1.3.6.1.4.1.2021.254. The following magic numbers would be the
next numbers on that oid for the var_mk function to use, ie:
.1.3.6.1.4.1.2021.254.1 (and .2 and .3 ...) */

#define MK_BOARDVERSION 1

#define MK_INTERNCPUID 2
#define MK_TESTMODE 3
#define MK_OUTPUTDIRECTED 4
```

```

#define MK_EXTERNCPUID      5
#define MK_WAITSTATE        6
#define MK_TESTBMODE        7
#define MK_OUTPUT           8
#define MK_MEMWRAPAROUND    9
#define MK_EXTCLOCK         10
#define MK_FIFOSTATE        11
#define MK_TIMESTAMPLOCK    12
#define MK_NODECLOCK        13
#define MK_MEMWAITSTATE     14
#define MK_ADDRESSPOINTER   15

#define MK_SAMPLING          16
#define MK_WRITEWAIT        17
#define MK_READWAIT         18
#define MK_FIFOFULL         19
#define MK_SHADOWREG        20
#define MK_FIFOVERRUN       21
#define MK_RESOURCEOVERRUN  22
#define MK_BIT161           23
#define MK_CPUWAITSTATE     24
#define MK_32BITNODE        25

#define MK_TIMEVAL           26

/* only load this structure when this .h file is called in the
   snmp_vars.c file in the agent subdirectory of the source tree */

#if defined(IN_SNMP_VARS_C) || defined(USING_DLMOD_MODULE)

/* Define a 'variable' structure that is a representation of our mib.
*/

/* first, we have to pick the variable type.  They are all defined in
   the var_struct.h file in the agent subdirectory.  I'm picking the
   variable2 structure since the longest sub-component of the oid I
   want to load is .2.1 and .2.2 so I need at most 2 spaces in the
   last entry. */

struct variable2 mk_variables[] = {
    { MK_BOARDVERSION, ASN_OCTET_STR, RONLY, var_mk, 1, {1}},
    /* Load the first table entry.  arguments:
       1: MK_BOARDVERSION: magic number to pass back to us and used to
          check against the incoming mib oid requested
          later.
       2: ASN_OCTET_STR: type of value returned.
       3: RONLY: Its read-only.  We can't use 'snmpset' to set it.
       4: var_mk: The return callback function, defined in mk.c
       5: 1: The length of the next miboid this will be located at

       6: {1}: The sub-oid this table entry is for.  It's appended to
          the table entry defined later.
    */

    { MK_INTERNCPUID, ASN_OCTET_STR, RONLY, var_mk, 2, {2,1}},
    { MK_TESTMODE, ASN_INTEGER, RONLY, var_mk, 2, {2,2}},
    { MK_OUTPUTDIRECTED, ASN_OCTET_STR, RONLY, var_mk, 2, {2,3}},

```

```

    { MK_EXTERNCPUID,      ASN_INTEGER,    RONLY, var_mk, 2, {2,4}},
    { MK_WAITSTATE,      ASN_INTEGER,    RONLY, var_mk, 2, {2,5}},
    { MK_TESTBMODE,     ASN_INTEGER,    RONLY, var_mk, 2, {2,6}},
    { MK_OUTPUT,        ASN_INTEGER,    RWRITE, var_mk, 2, {2,7}},
    { MK_MEMWRAPAROUND, ASN_INTEGER,    RONLY, var_mk, 2, {2,8}},
    { MK_EXTCLOCK,     ASN_INTEGER,    RWRITE, var_mk, 2, {2,9}},
    { MK_FIFOSTATE,     ASN_INTEGER,    RONLY, var_mk, 2, {2,10}},
    { MK_TIMESTAMPLOCK, ASN_OCTET_STR, RONLY, var_mk, 2, {2,11}},
    { MK_NODECLOCK,    ASN_OCTET_STR, RONLY, var_mk, 2, {2,12}},
    { MK_MEMWAITSTATE,  ASN_INTEGER,    RONLY, var_mk, 2, {2,13}},
    { MK_ADDRESSPOINTER, ASN_OCTET_STR, RONLY, var_mk, 2, {2,14}},
    { MK_SAMPLING,     ASN_INTEGER,    RWRITE, var_mk, 2, {3,1}},
    { MK_WRITEWAIT,    ASN_INTEGER,    RONLY, var_mk, 2, {3,2}},
    { MK_READWAIT,     ASN_INTEGER,    RONLY, var_mk, 2, {3,3}},
    { MK_FIFOFULL,     ASN_INTEGER,    RONLY, var_mk, 2, {3,4}},
    { MK_SHADOWREG,    ASN_INTEGER,    RONLY, var_mk, 2, {3,5}},
    { MK_FIFOVERRUN,   ASN_INTEGER,    RONLY, var_mk, 2, {3,6}},
    { MK_RESOURCEVERRUN, ASN_INTEGER,    RONLY, var_mk, 2, {3,7}},
    { MK_BIT161,       ASN_INTEGER,    RONLY, var_mk, 2, {3,8}},
    { MK_CPUWAITSTATE, ASN_INTEGER,    RONLY, var_mk, 2, {3,9}},
    { MK_32BITNODE,    ASN_INTEGER,    RWRITE, var_mk, 2, {3,10}},
    { MK_TIMEVAL,      ASN_OCTET_STR, RONLY, var_mk, 1, {4}}
};

/* Now, load the above table at our requested location: */
config_load_mib(1.3.6.1.4.1.6478.65, 8, mk_variables)
/* arguments:
   1.3.6.1.4.1.6478.65:  MIB oid to put the table at.
   8:                    Length of the mib oid above.
   mk_variables:        The structure we just defined above
*/

/* IMPORTANT: If you change the contents of this macro, you *must*
   re-run the configure script!!! */

#endif
#endif /* _MIBGROUP_MK_H */

/*****
/* mk.h file of the multikron toolkit */
*****/

#ifndef _MK_MODULE_H
#define _MK_MODULE_H
/*****
*****/
/* mk.h -- header file for the MultiKron combined PCI card.  This
header */
/* is to be used along with mk_init.c to map the MultiKron
card. */
/* The defines used for mapping locations on the card are based
on */
/* pointers declared inside mk_init.c.  If you want to use your
own */
/* initialization routines, then you must declare these
pointers: */

```

```

*/
/*
*/
/* SYSTEM:      x86 Linux
*/
/*
*/
/* BY:          Wayne J. Salamon
*/
/*
*/          NIST
*/
/*****
*****/

```

[Cut here]

2 mk.c

This file, based on a template, is the actual module added in the agent to provide Multikron sensors on each machine.

```

/* mk.c - multikron module for the SNMP agent */

/* Initializes the multikron board if not done yet.
   Allows setting of some bits
   I use here, in the switch clause, the source code of mk_status and
   its header file
   which is at the end of mk.h */

/* include important headers */
#include <config.h>

/* needed by util_funcs.h */
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
# include <sys/time.h>
# else
# include <time.h>
# endif
#endif

/* mibincl.h contains all the snmp specific headers to define the
   return types and various defines and structures. */
#include "mibincl.h"

/* header_generic() comes from here */
#include "util_funcs.h"

/* include our .h file */
#include "mk.h"

```

```

/* and string library... */
#include <string.h>

int writeMk (int, u_char *, u_char, int, u_char *, oid *, int);
    /*****
    *
    *   Initialisation & common implementation functions
    *
    *****/

/* this is an optional function called at the time the agent starts up
   to do any initializations you might require.  You don't have to
   create it, as it is optional. */

/* IMPORTANT: If you add or remove this function, you *must* re-run
   the configure script as it checks for its existence. */

void init_mk( )
{
    if (mk_init(JUST_MAP, 0L, 0L, 0L)) {
        system("/dev/mk.LOAD");
        mk_init(JUST_MAP, 0L, 0L, 0L);
    }
    /* call auto_nlist to load the nlist symbols.  We
       actually don't need it, so its commented out. */
    /* auto_nlist( "example_symbol" ); */
}

#if USING_DLMOD_MODULE

static struct subtree mk_subtree = {
    {1,3,6,1,4,1,6478,65}, 8, mk_variables,
    sizeof(mk_variables)/sizeof(*mk_variables),
    sizeof(*mk_variables), "mk"
};

int mk_init() {
    load_subtree(&mk_subtree);
    return 0;
}
#endif
    /*****
    *
    *   System specific implementation functions
    *
    *****/

/* define the callback function used in the mk_variables
   structure in mk.h */

u_char      *
var_mk(vp, name, length, exact, var_len, write_method)
    register struct variable *vp;
    oid      *name;
    int      *length;
    int      exact;

```

```

        int      *var_len;
        int      (**write_method) __P((int, u_char *, u_char, int, u_char
*, oid *, int));
    {
        /* define any variables we might return as static! */
        static long long_ret;
        static char string[300];
        static oid oid_ret[8];

        /* specific variables */
        unsigned long ulong;

        /* header_generic is a simple function for finding out if we're in
        the right place.  This only works on scalar objects.  Use
        checkmib for simple tables, and write your own for anything
        else. */
        if (header_generic(vp, name, length, exact, var_len, write_method))
            return NULL;

        /* We can now simply test on vp's magic number, defined in mk.h */

        switch (vp->magic){
        case MK_BOARDVERSION:
            /* set up return information */
            sprintf(string, "0x%01x", (mib_csr>>24) & 0xFF);
            *var_len = strlen(string); /* set the length of the returned data
*/
            return (u_char *) string; /* return everything as mapped to a
u_char * */

        case MK_INTERNCPUID:
            sprintf(string, "0x%02x", mib_csr & CPUID_MASK);
            *var_len = strlen(string);
            return (u_char *) string;

        case MK_TESTMODE:
            /* note: var_len defaults to the length of a long */
            long_ret = (mib_csr & BD_TEST) ? 1 : 0;
            return (u_char *) &long_ret;

        case MK_OUTPUTDIRECTED:
            sprintf(string, "%s", (mib_csr & BD_LOCAL) ? "Local Memory" :
"S16D");
            *var_len = strlen(string);
            return (u_char *) string;

        case MK_EXTERNCPUID:
            long_ret = (mib_csr & BD_EXT_CPU) ? 1 : 0;
            return (u_char *) &long_ret;

        case MK_WAITSTATE:
            long_ret = (mib_csr & BD_WAITSTATE) ? 1 : 0;
            return (u_char *) &long_ret;

        case MK_TESTBMODE:

```

```

    long_ret = (mib_csr & BD_NOTESTB) ? 0 : 1;
    return (u_char *) &long_ret;

case MK_OUTPUT:
    long_ret = (mib_csr & BD_OUTEN) ? 1 : 0;
    *write_method=writeMk;
    return (u_char *) &long_ret;

case MK_MEMWRAPAROUND:
    long_ret = (mib_csr & BD_NOWRAP) ? 0 : 1;
    return (u_char *) &long_ret;

case MK_EXTCLOCK:
    long_ret = (mib_csr & BD_EN_EXT) ? 1 : 0;
    *write_method=writeMk;
    return (u_char *) &long_ret;

case MK_FIFOSTATE:
    long_ret = (mib_csr & BD_DIS_FIFO) ? 0 : 1;
    return (u_char *) &long_ret;

case MK_TIMESTAMPLOCK:
    sprintf(string, "1/%d of Node Clock", (mib_csr & BD_TS_3TO1) ? 3 :
4);
    *var_len = strlen(string);
    return (u_char *) string;

case MK_NODECLOCK:
    sprintf(string, "%s board oscillator", (mib_csr & BD_NODE_2) ? "1/2
of" : "equal to");
    *var_len = strlen(string);
    return (u_char *) string;

case MK_MEMWAITSTATE:
    long_ret = (mib_csr & BD_WAIT_MEM) ? 1 : 0;
    return (u_char *) &long_ret;

case MK_ADDRESSPOINTER:
    sprintf(string, "0x%06lx", mib_mem_ptr & 0x3FFFFFF);
    *var_len = strlen(string);
    return (u_char *) string;

case MK_SAMPLING:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_ENABLE_SAMP) ? 1 : 0;
        *write_method=writeMk;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_WRITEWAIT:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_ENABLE_WO) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

```

```

case MK_READWAIT:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_ENABLE_RWCB) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_FIFOFULL:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_FIFO_FULL) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_SHADOWREG:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_SHAD_FULL) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_FIFOVERRUN:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_FIFO_OVERRUN) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_RESOURCEOVERRUN:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_RSC_OVERRUN) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_BIT161:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_FIFO_161) >> 10;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_CPUWAITSTATE:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_CPU_WAIT) ? 1 : 0;
        return (u_char *) &long_ret;
    }
    else return NULL;

case MK_32BITNODE:
    if (mib_csr & BD_OUTEN) {
        long_ret = (mk_csr & mk_ENABLE_32_BIT) ? 1 : 0;
        *write_method=writeMk;
        return (u_char *) &long_ret;
    }
    else return NULL;

```

```

case MK_TIMEVAL:
    ulong = mk_time & 0xFFFFFFFF;
    sprintf(string, "0x%08x%08x", mk_hi_register & 0x00FFFFFF, ulong);
    *var_len = strlen(string);
    return (u_char *) string;

default:
    ERROR_MSG("mk.c: don't know how to handle this request.");
}
/* if we fall to here, fail by returning NULL */
return NULL;
}

int writeMk (action, var_val, var_val_type, var_val_len, statP, name,
name_len)
    int    action;
    u_char    *var_val;
    u_char    var_val_type;
    int    var_val_len;
    u_char    *statP;
    oid    *name;
    int    name_len;
{
    int bigsize = 1000;
    int count, size;
    long buf;
    if (var_val_type != ASN_INTEGER){
        printf("not integer\n");
        return SNMP_ERR_WRONGTYPE;
    }
    size = sizeof(buf);
    asn_parse_int(var_val, &bigsize, &var_val_type, &buf, size);
    if (action == COMMIT){
        switch((char)name[8]){
            case 2:
                switch((char)name[9]){
                    case 7:
                        if (buf)
                            mib_csr |= BD_OUTEN;
                        else
                            mib_csr &= ~BD_OUTEN;
                        break;
                    case 9:
                        if (buf)
                            mib_csr |= BD_EN_EXT;
                        else
                            mib_csr &= ~BD_EN_EXT;
                        break;
                    default:
                        break;
                }
            case 3:
                switch((char)name[9]){
                    case 1:
                        if (buf) {
                            mk_csr |= mk_ENABLE_SAMP;
                            mk_csr |= mk_ENABLE_SAMP;
                        }
                }
        }
    }
}

```

```

        else
            mk_csr |= mk_DISABLE_SAMP;
        break;
    case 10:
        if (buf) {
            mk_csr |= mk_ENABLE_32_BIT;
            mk_csr |= mk_ENABLE_32_BIT;
        }

        else
            mk_csr |= mk_DISABLE_32_BIT;
    default:
        break;
    }
    default:
        break;
    }
}
return SNMP_ERR_NOERROR;
} /* end of writeMk */

```

3 MULTIKRON-MIB.txt

This file is used by the manager as a dictionary, converting object Ids (numbers) into more meaningful information for the user. The language used by SNMP is the ASN1 description.

```

MULTIKRON-MIB DEFINITIONS ::= BEGIN

IMPORTS
    enterprises, OBJECT-TYPE
        FROM SNMPv2-SMI
    DisplayString
        FROM SNMPv2-TC;

-- private    OBJECT IDENTIFIER ::= { internet 4 }
-- enterprises OBJECT IDENTIFIER ::= { private 1 }
nist         OBJECT IDENTIFIER ::= { enterprises 6478 }
mk           OBJECT IDENTIFIER ::= { nist 65 }
mkInterfaceBoard OBJECT IDENTIFIER ::= { mk 2 }
mkStatus     OBJECT IDENTIFIER ::= { mk 3 }

MkBoardVersion OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Multikron board version."
    ::= { mk 1 }

InternalCPUID OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only

```

```

STATUS mandatory
DESCRIPTION
    "Internal CPU ID"
 ::= { mkInterfaceBoard 1 }

MIBTESTmode OBJECT-TYPE
SYNTAX INTEGER {
    ENABLED(1),
    DISABLED(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "MIB TEST mode."
 ::= { mkInterfaceBoard 2 }

MkOutputDirectedTo OBJECT-TYPE
SYNTAX DisplayString
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Multikron output directed to."
 ::= { mkInterfaceBoard 3 }

ExternalCPUID OBJECT-TYPE
SYNTAX INTEGER {
    ENABLED(1),
    DISABLED(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "External CPU ID."
 ::= { mkInterfaceBoard 4 }

MkWaitState OBJECT-TYPE
SYNTAX INTEGER {
    ENABLED(1),
    DISABLED(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Multikron wait state."
 ::= { mkInterfaceBoard 5 }

MkTESTBMode OBJECT-TYPE
SYNTAX INTEGER {
    ENABLED(1),
    DISABLED(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Multikron TESTB mode."
 ::= { mkInterfaceBoard 6 }

```

```

MkOutput OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "Multikron output."
    ::= { mkInterfaceBoard 7 }

MIBWrapAround OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "MIB local memory wrap-around."
    ::= { mkInterfaceBoard 8 }

ExternalClock OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "External clock & reset."
    ::= { mkInterfaceBoard 9 }

FIFO OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "FIFO state machine."
    ::= { mkInterfaceBoard 10 }

TimeStamp OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Timestamp clock selected as."
    ::= { mkInterfaceBoard 11 }

NodeClock OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION

```

```

        "Node clock selected as."
 ::= { mkInterfaceBoard 12 }

MemoryWaitState OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Memory wait state."
 ::= { mkInterfaceBoard 13 }

LocalMemAddress OBJECT-TYPE
    SYNTAX DisplayString
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Local memory address pointer."
 ::= { mkInterfaceBoard 14 }

Sampling OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "Sampling."
 ::= { mkStatus 1 }

WriteWaitOverrun OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Write wait on overrun."
 ::= { mkStatus 2 }

ReadWaitCounters OBJECT-TYPE
    SYNTAX INTEGER {
        ENABLED(1),
        DISABLED(0)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Read wait on counters."
 ::= { mkStatus 3 }

FIFOFULL OBJECT-TYPE
    SYNTAX INTEGER {

```

```

TRUE(1),
FALSE(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "FIFO FULL."
 ::= { mkStatus 4 }

ShadowReg OBJECT-TYPE
SYNTAX INTEGER {
FULL(1),
notFULL(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Shadow register."
 ::= { mkStatus 5 }

FIFOOverrun OBJECT-TYPE
SYNTAX INTEGER {
TRUE(1),
FALSE(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "FIFO overrun."
 ::= { mkStatus 6 }

ResourceOverrun OBJECT-TYPE
SYNTAX INTEGER {
TRUE(1),
FALSE(0)
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Resource overrun."
 ::= { mkStatus 7 }

FIFOOutput OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "FIFO output bit 161."
 ::= { mkStatus 8 }

CPUWaitState OBJECT-TYPE
SYNTAX INTEGER {
ENABLED(1),
DISABLED(0)
}
ACCESS read-only
STATUS mandatory

```

```
DESCRIPTION
    "CPU wait state."
 ::= { mkStatus 9 }

32BitMode OBJECT-TYPE
SYNTAX INTEGER {
    ENABLED(1),
    DISABLED(0)
}
ACCESS read-write
STATUS mandatory
DESCRIPTION
    "32 bit mode."
 ::= { mkStatus 10 }

TimeVal OBJECT-TYPE
SYNTAX DisplayString
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "Time value."
 ::= { mk 4 }

END
```



SNMP Agent Documentation

The agent was made by the University of California at Davis (version 3.5.2 running under linux). It is located on *lamd@dirk.ncsl.nist.gov:~/steph/ucd-snmp-3.5.2*.

The package includes an SNMP agent (*~/agent/snmpd*) and SNMP utilities to get and set SNMP data (*~/apps*).

The only modified source code was the agent's. It was made in a way that modules could be added quite easily to provide extensibility.

Multikron and GPS supports were added as well as some statistical information from */proc/stat* files.

What was done?

Memory information for each process:

2 entries were added in *hrSWRunTable* : Size and RSS (Resident Set Size), entries that correspond to those in */proc/PID/status* (the program actually parses the content of this file). The MIB file *HOST-RESOURCES-MIB.txt* was modified as well as *hr_swrn.h* and *hr_swrn.c* files (located in *~/agent/mibgroup/host*). *hr_swrn.h* contains declarations of the *hrSWRunTable* MIB sub-tree where lines 26-27 and 40-41 were added. *Hr_swrn.c* contains the actual C code of the module where lines 542-570 were added.

Multikron support:

3 files (located in *~/agent/mibgroup/mk*) were necessary to add the multikron support:

- *MULTIKRON-MIB.txt*: ASN1 description of the multikron MIB sub-tree
- *mk.h*: header file for multikron module
- *mk.c*: C code

The MIB sub-tree used for the multikron data is *enterprises.nist(6478).mk(65)*. The multikron information is then divided in 3 parts: *mkInfo(1)*, *mkInterfaceBoardInfo(2)*, *mkStatusInfo(3)* that correspond to the *mk_status* program of the multikron toolkit.

The header and C files were based upon an example provided in the package in order to add modules to the agent (given in *~/agent/mibgroup/examples*).

The agent has the possibility to set three bits of the multikron, this feature is handled by the *writeMk()* procedure.

The multikron library is used to compile this module, it is located in *~/snmplib/libmk.a*. Consequently, the *Makefile* located in *~/agent* needed to be modified. '-lmk' was added at the end of the 'LIBS=' line.

GPS support:

As well as the multikron support, 3 files were added (located in *~/agent/mibgroup/gps*):

- *GPS-MIB.txt*
- *gps.h*
- *gps.c*

The MIB sub-tree used for the multikron data is *enterprises.nist(6478).gps(477)*.

The header and C files have the same structure as above.

The module just parses */tmp/gps_status_0* to get GPS data. No setting of SNMP data is currently available.

Some statistics:

Files in *~/agent/mibgroup/sys*:

sys.h

sys.c

The MIB sub-tree used for those statistics is *enterprises.nist.sys(797)*.

The module just parses */proc/stat*.

How to add a module?

Copy the 3 files of the multikron module and modify them.

The header and C files are obviously necessary. The MIB file is not although it can help understanding the data without having only numbers for each object ID and this file is often required for SNMP managers.

Let us suppose you take the multikron header file and you want to add a *foo* module:

- modify the *ifndef* and *define* lines (6-7) with something like `_MIBGROUP_FOO_H`

- modify line 12 if needed to add an MIB to the agent
- change all occurrences of procedures and variables names like *init_mk()*, *var_mk()*, *mk_variables* to *init_foo()*, *var_foo()*, *foo_variables*
- define SNMP entries (no more than 255...)
- change the whole *foo_variables* structure using the definitions above
- change the *config_load_mib* line to have the MIB sub-tree loaded where you want with *foo_variables* (I think that 1.3.6.1.4.1.6478.xx is not so bad as it is the just under the NIST branch, 65, 477 and 797 values for xx are already used.)

This new file will be *foo.h*.

Source code file:

- change *mk.h* for *foo.h*
- adapt line 29 to your library needs
- delete line 31 if there won't be any setting of variable needed or change the procedure name
- change occurrences of *init_mk()*, *mk_init()*, *var_mk()* to *init_foo()*, *foo_init()*, *var_foo()*
- modify the *mk_subtree* structure as needed (*foo_subtree*, *foo_variables* and 6478.xx)
- modify *foo_init()* to have *load_subtree(&foo_subtree)*
- delete specific variables and add some if needed
- we're now in the main part of the program, the *switch (vp->magic)* section where should appear the different cases related to the definitions of the header file. Just write the piece of code needed for each case, return a string with its length or just a long as explained in the file.

Next part is the *writeFoo()* procedure to be able to change the data remotely. In the *mk.h* file is an example of integer variable setting. For a string variable, you can go and see at the end of *~/agent/mibgroup/mibII/system.c*.

MIB file:

The MIB files modified and created are useful

Just modify *MULTIKRON-MIB.txt*. I think it's quite easy to understand the ASN1 description syntax. The sub-tree structure must be the same as the one declared in the header file. Some samples of ASN1 files are in *~/mibs*.

Compilation of the package:

First step is to execute the *configure* program located in *ucd-snmp* directory but to add the different modules, type:

```
configure --with-out-mib-modules="v2party ucd_snmp" --with-mib-modules="mibII/system  
host mk/mk gps/gps" (and other modules if needed like x/y where x is the directory of the  
module in ~/agent/mibgroup and y the name of the .c and .h files).
```

This program makes Makefiles and *config.h*.

If needed, modify Makefiles to include libraries (see multikron support above) and build the package with *make* in the *ucd-snmp* directory.

Then *make install* should normally be done to install the package in different directories as follows (you need to do it as root):

/usr/local/bin: utilities

/usr/local/sbin: agent daemon (*snmpd*)

/usr/local/share/snmp: agent configuration file and MIB definitions

But it can be annoying to have a lot of utilities we don't care about on each node that's why a script can be used to install only the agent on each node.

To work, the agent must be placed in */usr/local/sbin* and run by root, it also needs a configuration file *snmpd.conf* that can be in */usr/local/share/snmp* by default or anywhere else (with command line option). It also looks for MIB files in */usr/local/share/snmp/mibs*. The agent does not require those files, the manager usually does.

The sample EXPECT script, located in the *ucd-snmp* directory, installs the agent in */usr/local/sbin* and the configuration file in */usr/local/share/snmp*, and execute the agent on each node. The script needs a *snmp.tar* file containing *snmpd* and *snmpd.conf*.

Optional: To change the default locations of the agent files, 2 lines have to be changed in the *config.h* file:

```
#define DEFAULT_MIBDIRS
```

```
#define SNMPSHAREPATH
```

and then rebuilding is needed.

The default log file for the agent is */var/log/snmp.log*, the location of this file can be modified in the configuration file.

The configuration file defines read-write access. We apparently have the possibility to limit the access by IP address but I didn't really find out the way it works. At the moment, the read community name is "public" and the write one is "private" and anyone can get and set any variables from anywhere.

Directories and files modified/added in the package

(To use in case of a package upgrade...)

Multikron library: *~/snmplib/libmk.a*

Multikron directory: *~/agent/mibgroup/mk*

GPS directory: *~/agent/mibgroup/gps*

Statistic information: *~/agent/mibgroup/sys*

Memory information for each process addition: *~/agent/mibgroup/host/hr_swrn.c* and
~/agent/mibgroup/host/hr_swrn.h



SNMP Manager Screenshot

