Towards a DSS for performance evaluation of VAX/VMS-clusters
Hoogendoorn, J.

Published: 01/01/1988

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Oct. 2018
Memorandum COSOR 88-22

Towards a DSS for performance evaluation of VAX/VMS-clusters

by

J. Hoogendoorn

Eindhoven, the Netherlands

October 1988
EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

MASTER'S THESIS
Towards a DSS for Performance Evaluation of VAX/VMS-clusters

by
J. Hoogendoorn

Supervisor: dr.ir. J. van der Wal

Eindhoven, August 1988
## Contents

1 Introduction ................................................................. 3

2 Problem Specification .................................................... 6
   2.1 Configuration Description ......................................... 6
   2.2 The Monitor Utility .................................................. 12
   2.3 Modeling the VAX/VMS-cluster ..................................... 15
      2.3.1 Introduction .................................................... 15
      2.3.2 The Model ...................................................... 15
      2.3.3 The Parameterization of the Model .......................... 17

3 Improving the accuracy of Performance Calculation .............. 22
   3.1 Introduction .......................................................... 22
   3.2 The VAMP-Algorithm ................................................ 23
   3.3 Specifying Inputs .................................................... 30
   3.4 Seek Optimization .................................................... 32
      3.4.1 The Disk Unit .................................................. 32
      3.4.2 Seek Distance Distributions .................................. 34
      3.4.3 Implementation of the Seek Optimization .................. 38
      3.4.4 Seek Optimization and Performance ........................ 40
   3.5 Suggestions for Future Improvements ............................. 41

4 User Interface .................................................................. 43
   4.1 Introduction ............................................................ 43
   4.2 Starting the User Interface .......................................... 47
   4.3 Management of Daily Data Files .................................... 48
      4.3.1 Model Parameter Determination ............................... 48
      4.3.2 Amalgamation of Daily Data Files of a Specific VAX into Monthly
            Data Files ......................................................... 50
   4.4 Performance Calculation of Existing Configuration .......... 52
   4.5 Cluster Management ................................................... 56
      4.5.1 Build Clusters .................................................... 56
      4.5.2 Remove Clusters ................................................ 62
      4.5.3 Performance Calculation of an Example of a built Cluster . 63
Chapter 1

Introduction

Nowadays, system managers of computer systems are constantly faced with problems concerning the performance of their computer systems. These problems often reflect heavily used system resources, called bottlenecks. Once a problem is identified as a bottleneck, the right strategy has to be found in attacking the bottleneck. Such a strategy always includes a system adaptation. Either a short term adaptation consisting of an improved allocation of the user workload over the system resources or a long term adaptation by extending the actual system configuration. Especially in the academic and research environment, the changes in system configurations are rapidly responded by an altered behaviour of users. Releasing new projects is postponed till after the actual change and running projects will be extended by making larger calculations in order to obtain more reliable results. Very soon, the user workload upon the system asks for a new system adaptation.

In order to survive the battle against the ever increasing automating, the system manager should have a tool which supports decisions concerning performance problems. First of all such a tool should be able to measure the current system workload and to use these measurements in an analysis in order to obtain performance statistics. In combination with the experience and knowledge of the system manager and the environment the system is placed in, these statistics could indicate whether it is time for a short term or long term adaptation. It is obvious that the requirements about response times in the commercial and military area are more strict than in the academic and research world. The tool should also be able to predict the effects of certain short and long term system changes upon performance. Thus supporting the system manager in choosing the best alternative in attacking the environment dependent bottlenecks. Long term improvement like increasing background memory is often applied in the the academic and research areas. Computer upgrading or main memory extension can be seen in each environment. Moreover, in practise the performance improvements of the alternatives will always be weighted against cost aspects, since enormous amounts of money are involved in the purchase of hardware devices.

We, performance analysts, are capable of building such tools, also known as Decision
Support Systems (DSS). We have the knowledge of modeling computer systems and developing software packages. When the measurements made upon the system are collected by monitor utilities, our models have to be tuned to these measurements. A problem that we find with monitor utilities is that they often monitor other information that we would need to calculate our models. The information is often specifically gathered for technical and computing purposes. Therefore it can be stated that performance analysts should be involved in future developments of monitor utilities.

The alternative for monitor utilities is to extract information about the internal behaviour of executing programs. Since these measuring programs themselves affect the performance (in what way?), this method isn't very much applied in the area of performance evaluation.

Due to long term research followed by standardizations of computer systems, extensive documentation is available allowing for construction of detailed models. Since these models aren't analytically tractable, the analysis has to be based upon expensive simulations with large programming and computing complexity. Considering the cost aspect, simulations can only be used for a limited scope of short term and long term adaptations. In the context of decision supporting, the tool should be able of analyzing a wide range of adaptations. Therefore mainly simple models are used, in spite of very restrictive modeling assumptions imposed by the algorithms used in the mathematical analysis.

In this report we will discuss some aspects of the design and development of a DSS aimed at system managers of VAX/VMS-clusters. Further, some case studies with the initial DSS will be described. The hardware components in the VAX/VMS-clusters are manufactured by Digital Equipment Corporation and are operating under the VMS operating system. Around the world over 30,000 of these clusters have been placed in technical, commercial, military, research and academic environments. Hence, the clusters have capability proven in all sorts of applications. The clusters have high growth potential, since new VAX hardware is compatible and will operate effectively without extensive changes to software and hardware. Therefore the development of the DSS has been aimed at flexibility, allowing for a great variety in applications and system configuration changes. The initial DSS is based upon measurements obtained by a monitor utility installed standardly on each VAX computer and a simple but analytically tractable model. In this context we named the DSS the VAX/VMS Analysis and Measurement Package (VAMP).

In August 1986 the initial attempt was made to model the VAX/VMS-cluster of the Eindhoven University of Technology (EUT). Implementing in the academic area means unpredictable user behaviour, since many users are free to choose any computer system they like. Moreover, the users of various Departments at a University often behave completely different, since each Department has its own characteristic workload. We have continued developing the VAMP packet by collecting the measurements more robustly, improving the adaptability to configuration changes and designing a clear user interface.
In april 1988, we found that the development had reached a phase that it could be implemented elsewhere. The Wageningen University of Agriculture (WUA) has given us the opportunity to see whether VAMP could be implemented in a much greater and semi-academic environment. In return, we could give the system manager performance statistics. It seemed that with some minor adjustments VAMP could be used in finding and attacking performance problems in this different environment. The control over VAMP remained ours, since the packet is still in development.

In Chapter 2 we will discuss the characteristics of VAX/VMS-clusters, both the hardware components and the internal operations, and the tuning of the model to the measurements. In spite of the fact that this initial and crucial part has been developed by others (see [2]), we thought it necessary to briefly discuss it again in order to fully understand the following Chapters.

In Chapter 3 we will discuss the implemented algorithm and some improvements concerning the accuracy.

Chapter 4 handles the designed user interface, emphasized on management of obtained measurements and both current and predictive performance calculations.

Chapter 5 discusses a case study done on the EUT-cluster. Further, in this chapter some examples concerning performance predictions on the VAX/VMS-clusters of both EUT and WUA will be described.

Finally, Chapter 6 contains a brief reference manual of the VAMP package.
Chapter 2
Problem Specification

2.1 Configuration Description

The VAMP package has specifically been developed for the VAX/VMS-cluster family. The system configurations of this family are all characterized by at least two VAX processors and a number of disk controllers (e.g. the HSC50), connected by a coupler. The user interaction happens via terminals connected to at least one of the processors. The disk controller takes care of the in- and output of data requests coming from the VAX processors and the in- and output of background memory data (in page format) the processors are asking for. The coupler is the connection point of all VAX processors and disk controllers.

![Diagram of VAX/VMS-cluster]

Figure 2.1: The VAX/VMS-cluster

A VAX/VMS-cluster can contain at most 16 VAX processors and disk controllers, imposed by the required coupler. Each disk controller can support up to 24 disk or tape
units. Moreover, it is possible to connect VAX processors directly with disk or tape units, thus creating local background memory. This means a great variety in configurations within the VAX/VMS-cluster family.

If there are connected disks locally to a VAX processor, this VAX also has to spend precious calculation time to disk controlling, by handling the disk IO traffic between the other VAXes and the local disks.

Concerning the hardware components, this is all we need to model the VAX/VMS-cluster properly. The VAX/VMS-cluster described above is shown in Figure 2.1.

At this moment, August 1988, the VAX/VMS-cluster of the EUT consists of three VAX processors (1 VAX 8530 and 2 VAX-11/750s) and nine disk units (7 RA-81s and 2 RA-60s) connected by one HSC50. The WUA-cluster consists of four VAX processors (1 VAX-11/785, 1 VAX 8600 and 2 VAX 8700s) and fifteen disk units, four locally on the VAX-11 machine (1 RP-07 and 3 RP-06s) and eleven connected by one HSC50 (1 RA-60 and 10 RA-81s). The RP-06 and RP-07 disk are from the early generation of VAX disks and cannot be connected with the HSC50. As a consequence, they can only be local. However, the current and future generation is and will be compatible with the HSC50.

In order to be able to evaluate performance of a computer system, it is necessary to extract data on how the system is being used and how it responds to requests of users. The VAX/VMS operating system contains a standard monitor utility installed on each VAX processor, which is capable of statistically collecting and displaying several data items. Via this utility the VAMP package collects data of each VAX in the cluster it is running for, by means of taking samples of these statistics throughout the day at regular intervals of three minutes. For each VAX processor in the cluster, we take samples of three statistic displays at the same time. The samples can be categorized as follows.

- cat.1 Samples concerning activities of various processes belonging to a specific VAX
- cat.2 Samples which provide the disk IO rates caused by all processes belonging to this VAX
- cat.3 Samples which reflect general VAX processor activity of all processes belonging to this VAX

Besides the description of some of the hardware components, some internal operations have to be specified in order to understand what is measured and why.

We mentioned the concept of processes. The VMS operating system distinguishes three classes of processes:

- Interactive Processes - These correspond to users who have logged into the system and are reacting interactively with the computer via a terminal.
- Batch Processes - Processes which run automatically and require no additional user input from a terminal.
- System Processes - These are processes which are created by the above two sorts of user processes in order to perform a certain task. Once this task is completed, the system process disappears.

Interactive processes have higher priority than batch processes at the CPU, since it is supposed that interactive processes have users waiting behind their terminals, who want as rapid a response as possible. Therefore, as the bottleneck CPU becomes busier with more interactive processes, batch processes receive almost no attention from the CPU. System processes have in fact the highest priority.

Each process appearing in the first category of samples (cat1.), is allocated a limited space in main memory called the Working Set (WS), partly or completely filled with pages. Without pages in main memory the processor cannot edit, run or debug programs. Each page consists of 512 contiguous byte locations used as unit for data transferring inside main memory or between main memory and backing storage on the disks. The concept of disk IOs mentioned in the second category, contains the transferring of pages from main to background memory and in reverse.

When a currently executing process lacks a page in its WS, a page fault occurs. The VMS memory management system contains a system service called the pager, which locates the missing page either in main memory or somewhere on disk and brings the page in. A page fault in a filled WS requires the least recently brought in page to be removed. This page is transferred to either the free page list or modified page list, dependent on whether the page has been modified during its stay in the WS. Both lists are part of main memory.

The concept of these lists allows for reducing the fault time, since copying and transferring of a page from disk, called a page read IO, can be avoided if this page is on one of these lists. The modified page list serves another important purpose. By delaying the writing of modified pages to disk, called a page write IO, many pages never have to be written to disk at all, because the pager brings them in the WS for another modification. Hence, in order to gain as much performance as possible, these lists have to be of reasonable size. These sizes are controlled by the swapper, another service of the VMS memory management system. Exceeding the lower limit of the number of pages on the free page list is followed by an attempt of the swapper to write the whole modified page list to the paging file somewhere on disk, in order to create space for the free page list. In this way the number of write IOs are diminished by grouping the pages. Paging files are used to save the contents of modified pages in background memory, allowing for sorting the pages by process. The pager must have access to these pages, since these pages have no copies like the unchanged pages on the free page list.

If the modified page list is too small to make writing interesting, the swapper tries to adjust the WS sizes of the processes currently competing for system resources. The size of each WS changes constantly over time. The exact adjustment strategy and the mentioned limits concerning the sizes of the lists have been described in [2]. In Chapter 5, we will discuss the WS adjustment in more detail.
If after the WS adjustment the free page list is still too small, the swapper outswaps entire WSs of the longest inactive processes to the swapping file on disk. This certainly creates space for the free page list.

The paging and swapping file can be seen as temporary extendable main memory. They belong to the so-called virtual memory.

It has to be remarked that when a page fault results in a page read IO, a number of contiguous pages are brought in. This affects also the performance, since the contiguous pages are often the next pages necessary for execution.

Besides the page read IO and page write IO, the VMS operating system distinguishes a third disk IO, called the direct IO. It contains IOs due to a page transfer directly from one of the memory buffers.

The VMS operating system contains a scheduler, which selects of all computable processes resident in main memory, the process with highest priority. Priorities 16 up to 31 are reserved for realtime processes, such as the swapper. They can only be run by suitably privileged users. The priority does not change over time and the processes run until completion, preemption by another realtime process or entering a yet to define wait state.

Time sharing processes have fluctuating priority between 0 and 15. These processes include terminal sessions, batch jobs and nearly all system processes. The priority changes due to several priority boosts, which occur at certain events such as terminal input and output completion or disk IO completion (priority increase) or quantum expiration (priority decrease). The priority of the interactive processes fluctuates between 4 and 10, while the batch and system processes are able to obtain priorities 0 up to 15. Each batch and system process has a so-called basic priority between 0 and 15, which is the lower bound for the fluctuating priority. As a consequence, the priority of these processes is always between the basic priority and 15. The time sharing processes are subjected to quantum control, meaning that a selected process can execute at most a certain time quantum. A time sharing process executes until expiration of the quantum unless the process is preempted, enters a wait state or is terminated before expiration. In each VAX processor the time quantum is set on 200 milliseconds. The part of the quantum which a time sharing process consumes is known as time slice.

At the moment, our main interest are the states a process can be in. At any time a process is in one of 14 different states. These states are displayed in the first category of measurements and indicate the particular state a process is in at the time the sample was taken.

---

1This number is called the clustersize.
The states fall out in three groups:
- Current State
- Computable States
- Wait States

A process in the CUR state is currently being executed. A process in the COM state is computable and enters the CUR state after having been selected as the highest priority resident process. A CUR process makes a transition to the COM state when it is preempted and to one of the wait states by making a direct or indirect request for a system operation which cannot complete immediately. This is the only way of entering the wait states. A process in the COM state which is being outswapped enters the COMO state. In Figure 2.2 these state transitions are graphically displayed.

Concerning the wait states, we look only at the following states which are important in the context of our way of modeling:
- **LEF and CEF**, the local and common event flag. These are system service wait states, because entering these states is a result of invoking a system service. A process can use this service for signaling an asynchronous event. An IO completion is typically asynchronous with a process’ execution. In order to synchronize activities within a process, local event flags are used. A process can also use system services to set common event flags to communicate with other processes. One process can reach a critical point in its execution and wait on a common event flag to be set by another process. A common event flag can also be used to gain access to a resource shared among processes.
- **FPW.** This wait state is associated with memory management. A process enters the free page wait when it requests a page to be added in a filled WS, while there are no free pages to be allocated on the free page list. In fact, the process is waiting for the swapper to extend the free page list.

- **SUSP.** A process is suspended and must be restarted by another process. Since a process can be suspended by another process, this is a special wait state.

Via the resident (i.e. not outswapped) wait states, a process can enter the COM state. From certain non resident wait states only the COMO state can be reached, while processes in other wait states can enter both the COM and COMO state. On his turn the COMO state can only be left by entering the COM state by means of inswapping.
2.2 The Monitor Utility

In the preceding Section we mentioned three categories of samples, collected from three different monitor statistics provided by the VMS operating system. In order to get an idea what is measured, we will give examples of samples taken from the three monitor utilities. Further we will specify for each statistic the items we use for the VAMP package.

**Figure 2.3: Sample of MONITOR PROCESSES**

The utility MONITOR PROCESSES provides samples for the first category. In Figure 2.3 a sample is shown. In each sample, for each interactive, batch and system process connected to the system, its name, current priority and state at the time the sample is taken is of importance. Further, some cumulative data collected since the start of the process.

- number of direct I/Os
- number of page faults
- CPU calculation time

In order to be able to sort these items at the process classes (interactive, batch and system), each process has to be assigned a process class. Identifying processes seemed to
be very difficult. For instance, the priority is not an ambiguous criterion due to the non-disjunct priority ranges of the classes. Therefore the following mixture of criterions has been developed for the identification.

A system process is identified by name. These names of all system processes are stored in file CONFFILE.DAT, in order to be read by the VAMP package. This requires a file update each time a system process is added or a name is changed.

A process is recognized as a batch process, if its name starts with "BATCH", in conformity with the default name. Since it is possible to alter this name, this criterion needs a supplement. Therefore, a second criterion sees whether the priority of the process appearing during the samples taken on one day at regular intervals of 3 minutes is at least once less than 4 or more than 10.

A process is said to be an interactive process if it is neither a system process nor a batch process.

As a consequence of this way of identifying, a system process which has its name not in the mentioned file, is considered batch or interactive depending on the priority during its existence. Further, a batch process with an altered name which has during existence its priority constantly between 4 and 10 is considered interactive. However, at working days this last error is not likely to occur, since the batch jobs hardly receive attention from the CPU, thus few priority boosts.

The samples corresponding to the second category are taken from MONITOR DISK. The only items we use are the current disk IO rates (per second) caused by all processes from a specific VAX generated in the preceding three minutes before the sample was taken to each disk in the existing configuration. The logical names of the disks are used to identify the disks. An example is shown in Figure 2.4.

![Figure 2.4: Sample of MONITOR DISK](image-url)
The utility MONITOR SYSTEM/ALL supplies the samples for the last category. It provides some general information about critical system activities. In each sample, the following items, averaged over the last three minutes before the sample was taken, are of importance.

- idle time of the CPU
- page fault rate of all processes together
- free page list size (in pages)
- direct IO rate of all processes together

An example of a generated sample of this monitor utility is shown in Figure 2.5

![VAX/VMS Monitor Utility System Statistics](image)

**Figure 2.5: Sample of MONITOR SYSTEM/ALL**
2.3 Modeling the VAX/VMS-cluster

2.3.1 Introduction

In modeling computer systems three parts are involved. The model, the parameterization of the model and the algorithm to calculate the model. The parameterization forms the kernel of modeling, since the model parameters are determined from measurements, which are fixed the moment the method of measuring is chosen. As a consequence, the model parameters can change within certain small bounds. However, the model and algorithm have many degrees of freedom and can therefore be tuned to the parameters. In Section 2.3.2 we describe the type of mathematical problem and the way we have tuned the model for the VAX/VMS-cluster. In Section 2.3.3 we will discuss the parameterization. We will follow the deduction of the model parameters from samples collected of one VAX. This deduction is VAX independent, since the monitor utility is installed standardly on each VAX.

The algorithm implemented in the VAMP package will be described in Chapter 3, together with some tuning improvements.

2.3.2 The Model

In the terminology of a model, the VAX/VMS-cluster can best be seen as a closed queuing network. A queuing network is characterized by a number of stations, allowing queues to appear. In these queues customers are waiting for service. The customers can be subdivided into classes, each of them having their own routing through the network. It is allowed to have class dependent service times and priority among the classes. A closed queuing network is a queuing network with a constant number of customers, with no customers arriving from outside the network.

In the context of computer systems, we will further speak of processes in stead of customers. In our models, a processor is often called a Central Processor Unit (CPU).

It seemed that the actual time a disk IO request stays at the disk controller is very small compared to the access time of the disk units. The HSC50 can handle up to 120 IO requests at the same time. Even in relatively busy situations the delay will be minimal (see [2]). Therefore, the VAX/VMS-cluster can be modelled as shown in Figure 2.6.

We have modelled the stations in the closed queuing network as follows. The terminals as one infinite server (IS) station. According to the Round-Robin priority scheduling of the CPUs (see [2]), each CPU has been modelled as a pre-emptive resume priority station with processor sharing (PS) scheduling policy at each priority level (processor sharing is Round-Robin with an infinitely small quantum). Each disk has been modelled as a single server station with first come, first served (FCFS) scheduling. Each station has independent exponential service time. The number of classes of processes are defined by the number of VAX processors. Each CPU has three classes - interactive, batch and system processes. Upon leaving the CPU an interactive process either returns to the
terminal station or goes to the disk. Once the necessary information has been read, the process returns to the CPU for further service. The batch and system processes never visit the terminal station.

Concerning the tuning of the model of Figure 2.6, the parameters necessary to calculate this model are the workloads per process class at CPUs, disks and terminal and the routing per process class through the network. These parameters could not be obtained from the samples for the following reasons:

- The VAX/VMS monitoring program does not sufficiently distinguish system processes from user processes. Since system processes are only in existence when called upon by user processes, we have modelled the workload of system processes as a part of the user processes, proportional to their fraction of CPU calculation time. We have therefore reduced the number of classes of processes per VAX processor from three to two.

- It seemed impossible to determine a real terminal workload from the generated samples. We could only obtain a percentage of the time an interactive process stays at the terminal station. As a consequence we had to use relative workloads, meaning that we only consider relations between workloads. Unfortunately, interesting information about real response times could not be obtained anymore.

- It seemed that a reasonable percentage of the interactive processes are in fact inactive, in the sense that they use less than the smallest measured amount of CPU calculation time (100 msec) in a three minutes interval. Since we are only interested in the interactive processes that really have a system workload, we introduced the concept of active processes. In a generated sample, a process is said to be active, if it is an interactive process with at least 100 milliseconds CPU calculation time more than the preceding sample. Therefore, we will speak of the active process classes instead of interactive process classes.
It proved impossible to obtain the probabilities that an interactive process upon leaving the CPU returns to the terminal or goes to a disk. Therefore we had to adjust the model as shown in Figure 2.6 to the model as shown in Figure 2.7. The fraction of time a process of a certain class stays at one of the disks now is based upon numerous disk I/Os, measured in the samples. We will ensure that the measured fraction of disk time equals the yet to determine relative disk workload.

Figure 2.7: The actual Model

Compared to the model in Figure 2.6 we have changed the routing per process class through the network, the number of process classes and real workloads into relative workloads. After a CPU visit, each process (active or batch) selects one of the disks based on the routing probabilities. After having read the necessary information at the disk, each process enters the terminal station with probability one. Since this is a IS station, each arriving process receives service immediately. The batch processes have relative terminal workload zero, since batch jobs require no terminal input.

In the coming Section we will discuss the components used to estimate the parameters of the active class and the batch class belonging to a particular VAX. The components are deducted by joining and sorting all of the samples of this VAX taken on one day.

2.3.3 The Parameterization of the Model

After the threefold generating of samples has been stopped, the samples corresponding to a specific VAX are combined with the intention of sorting the samples twice and deleting the huge samples by creating much smaller files. In this context the sorting for the number of active processes is of importance. In each sample a certain number of active processes is determined by comparing the CPU calculation time of the interactive processes of this sample with the preceding sample. As a consequence the model parameters are defined by number of active processes. Per number of active processes the following data items are selected which are used in our parameterization:
- The number of samples with this number of active processes (cat.1).
- The average number of batch processes in these samples (cat.1).
- The average interval length (about three minutes) of these samples (cat.1).
- The cumulative number of times that active processes were waiting in the LEF, CEF, FPW and SUSP state at times these samples were taken (cat.1). The last three states indicate that a process is neither being served nor is waiting for service at terminal, CPU or disk.
- The direct IO rate, page fault rate and the fraction of CPU calculation time for all active and batch processes and for all processes together. The disk IOs caused by the "inactive" interactive processes are added to the disk IOs of all processes together, and not to the active processes. Further the read IO rate for all processes together are selected. These items are measured in cat.2 and cat.3.
- The disk IO rates to each disk in these samples, measured in cat.2.

Having defined the components of the relative workloads, we can start building these workloads. Remember that we distinguish per CPU two classes of processes and that we have to spread the system workload of system processes over the active and batch processes.

We introduce the term user process to denote a process which is either active or batch. Consequently we will speak in the deduction of the model parameters of user processes, when the model parameter construction of active processes is similar to the deduction of batch processes.

In order to avoid confusion, we will now specify all used indices in the remainder of this Section. Index to will be used for an item selected for all processes together. Indices ac, ba, sy and us denote respectively an active, batch, system and user process. Finally, index dsk will be used to indicate a disk related item.

We define \( dI/O_{ac}, dI/O_{ba} \) and \( dI/O_{to} \) for the direct IO rates, \( pft_{ac}, pft_{ba} \) and \( pft_{to} \) for the page fault rates and \( CPU_{ac}, CPU_{ba} \) and \( CPU_{to} \) for the fraction CPU calculation time. The read IO rate is defined as \( rdI/O_{to} \) and the disk IO rates as \( r_{dsk} \).

The relative CPU workload is the number of seconds per second that an user process is actually receiving attention from the CPU. For each user process (fill in : active or batch) it equals

\[
\frac{CPU_{us}}{CPU_{ac} + CPU_{ba}} \frac{1}{\#user\ processes}
\]
For each other process class, i.e. the active and batch class of the other VAXes, the relative workload of this VAX equals zero.

The relative disk workload is the number of seconds per second that an user process stays at the disks. Let \( m_{disk} \) be the access time of a certain disk, defined as the absolute average service time of a page read IO. In Chapter 3 we will discuss the components of this access time. For each disk and for each user process, the relative disk workload equals

\[
\frac{IO_{us}}{IO_{ac} + IO_{ba} \sum_{disks} r_{disk} m_{disk}} \cdot \frac{1}{\#user\ processes}
\]

The concept of one relative workload for each disk is imposed by the fact that the IO fraction cannot be split up by disk. The \( IO_{us} \) contains an estimate for the disk IO rate caused by user processes and caused by their proportional part of system processes. It equals

\[
\frac{CPU_{us}}{CPU_{ac} + CPU_{ba}} \cdot \left( \text{direct IO + page IO rates}_{sys} + (\text{direct IO + page IO rates})_{us} \right)
\]

We see that the direct IOs and the page IOs caused by system processes are again distributed over the active and batch processes, proportional to their fraction of CPU calculation time. The direct IO rate of system processes equals \( dIO_{to} - dIO_{ac} - dIO_{ba} \) and of the user processes \( dIO_{us} \).

Concerning the page IO rate, the page write IOs are ignored, since they hardly contribute to the total number of disk IOs ([2]). Therefore we assume that the relation between page fault rate and page IO rate is the same for active, batch and system processes since the monitor utility does not make a distinction between the page IOs of different sorts of processes. So the page IO rate for system processes equals \( rIO_{to} (pft_{to} - pft_{ac} - pft_{ba})/pft_{to} \) and for the user processes \( rIO_{to} pft_{us}/pft_{to} \).

The relative terminal workload, also known as think time, is the number of seconds per second an active process stays at the terminal station. This workload equals zero for batch processes. This, together with the infinite server character of the terminal station, ensures the batch processes to circulate between CPU and disk. An active process in the LEF state means that this process is waiting for terminal input or direct or write IO completion. The fraction of time an active process is in the LEF state minus the fraction of time this happens due to a direct or write IO completion should give a good estimate for the relative terminal workload. The rate of write IOs for all processes together equals

\[
\sum_{disks} r_{disk} - rIO_{to} - dIO_{to}
\]

By multiplication with the CPU fraction \( CPU_{ac}/CPU_{to} \), we obtain an estimate for the write IO rate for active processes.
The rate of direct and write IOs for active processes, also known as nonread IO rate becomes
\[ dIO_{ac} + \frac{CPU_{ac}}{CPU_{to}} \left( \sum_{disk} r_{disk} - rIO_{to} - dIO_{to} \right) \]

We want a fraction of time that an active process waits for direct or write IO completion. Besides this rate of direct and write IOs, we need the average absolute disk response time for one such IO. This time depends on the disk IO rate of the other VAX processors in the model (what we need right now are the combined and sorted samples of the other VAXes!). We assume an average disk IO rate from the other VAXes.

An estimate for the average absolute disk response time becomes
\[ \frac{1}{\sum_{disk} r_{disk}} \sum_{disk} r_{disk} \frac{m_{disk}}{1 - [r_{disk} + (average\ rates\ of\ other\ VAXes)] m_{disk}} \]  (2.1)

In order to understand this formula, we return to elementary queuing theory. Let \( \lambda_{disk} \) be the rate of all disk IOs caused by all VAXes to a certain disk. Let \( S_{disk} \) be the absolute average response time of this disk and \( L_{disk} \) the average number of processes waiting. Mean Value Analysis [3] applied for the M|M|1-queue gives the following equations.

\[ S_{disk} = (L_{disk} + 1) m_{disk} \]
\[ L_{disk} = \lambda_{disk} S_{disk} \]

After elimination of \( L_{disk} \), we obtain the following equation.
\[ S_{disk} = \frac{m_{disk}}{1 - \lambda_{disk} m_{disk}} \]

With weights \( r_{disk}/(\sum_{disk} r_{disk}) \) we obtain the absolute disk response time, the real disk time, of (2.1). We are now able to construct the relative terminal think time. For each active process it equals

\[ \frac{(#LEF\ states\ in\ samples) - (nonread\ IO\ rate)(#samples)(real\ disk\ time)}{(+#samples)(#active\ processes)} \]

with the samples corresponding to the number of active processes in the denominator.

A fourth relative workload is determined, called the relative wait time. It is the number of seconds per second that an active process neither is being served nor is waiting for
service at any station in the model. In practise, this only occurs in busy situations and since this phenomenon is measured, it is necessary to model it. The relative wait time will be added to the relative terminal workload, because the mentioned measurements only includes active processes.

The relative wait time equals

\[
\frac{(# CEF, FPW and SUSP states in samples)}{(# samples)(# active processes)}
\]

The remaining model parameters are the relative visit frequencies per process class. These visit frequencies define the routing of processes of a certain class through the model. All classes of processes have visit frequency 1 to the terminal station. The active and batch processes have visit frequency 1 to their corresponding VAX and frequency 0 to all other VAXes. Both active and batch processes of the same VAX have disk visit frequency \( r_{disk} / (\sum_{disks} r_{disk}) \). These relative disk visit frequencies ensure that the distribution of the processes over the disks is quite accurate.
Chapter 3

Improving the accuracy of Performance Calculation

3.1 Introduction

In Chapter 2 we have described a simple queuing network model for the VAX/VMS-cluster. Further, we have explicitly described the way VAMP deducts model parameters from the measurements. In order to complete the modeling of the VAX/VMS-cluster we will discuss in this Chapter the algorithm which calculates the model by using the model parameters and giving performance statistics in return. The algorithm of the VAMP package is implemented in a module, which is linked to the major VAMP program. At performance calculations a data file containing model parameters is created in order to be read by the module. In this way it is possible to add new algorithms by linking new modules to the VAMP program. Clearly, a wider range of algorithms will make the VAMP package a step closer towards a DSS. However, at this moment the VAMP package has still one algorithm. We haven't had time to look at other algorithms, since the existing VAMP-algorithm wasn't working perfectly. It sometimes occurred that the utilizations of both the VAXes and the disks were greater than one and that the response times were negative. It seemed that some of these problems were due to inaccurate determination of the model parameters from the measurements, while others were caused by malfunctioning of the algorithm. Our efforts to improve the accuracy of calculating the performance in order to avoid these occurrences will be described in this Chapter. In Section 3.2 we will describe the VAMP-algorithm at the time we started this project. In Section 3.3 we will discuss an improved interaction between the algorithm and the model parameters, by checking the model parameters before they are stored in the data file. Some improvements concerning the VAMP-algorithm will be discussed in Section 3.4. In the last Section we will make some suggestions for future improvements of the accuracy of performance calculations.
3.2 The VAMP-Algorithm

The queuing network model described in Chapter 2 can be calculated by a number of algorithms. Since many of the algorithms and in particular the VAMP-algorithm, are based on the so-called Mean Value Analysis (MVA) algorithm [3], we will first discuss the latter algorithm.

We have \( N \) stations and \( R \) process classes. Each process class has its own routing through the network, defined by the probabilities \( p^r_{m,n} \), the probability that a process of class \( r \) upon leaving station \( m \) decides to go to station \( n \), \( r = 1, \ldots, R \) and \( n, m = 1, \ldots, N \). Since we only consider closed queuing networks, the following equations for \( r = 1, \ldots, R \), \( m = 1, \ldots, N \) should hold.

\[
\sum_{n=1}^{N} p^r_{m,n} = 1
\]

We define \( f_{n,r} \) as the relative visit frequency of a process of class \( r \) to station \( n \). The value of \( f_{n,r} \) is a measure for the number of times per second that a process of class \( r \) visits station \( n \). The relative visit frequencies hold for the following equation.

\[
f_{n,r} = \sum_{m=1}^{N} p^r_{m,n} f_{m,r}
\]

It is obvious that in closed queuing networks the visit frequencies can be relative. It is easy to verify that \( f_{m,r} / f_{n,r} \) denotes the expected number of visits to station \( m \) per visit to station \( n \).

A process of class \( r \) at station \( n \) receives an exponentially distributed workload with mean \( w_{n,r} \). If station \( n \) is a FCFS station, we assume the workload class independent. We will denote this by the workload \( w_n \).

Finally we define a population vector \( K = (K_1, \ldots, K_R) \), with \( K_r \) the number of processes belonging to class \( r \) and a population vector \( k = (k_1, \ldots, k_R) \), with for each class \( r \) \( 0 \leq k_r \leq K_r \). We define \( K \) the end-population of the queuing network.

The MVA-algorithm is based on a set of recursive relations between expected response times, throughputs and expected number of processes at the stations of the network. Therefore, we introduce the following notations.

- \( S_{n,r}(k) \) : expected response time of processes of class \( r \) at station \( n \).
- \( \Lambda_{n,r}(k) \) : throughput of processes of class \( r \) at station \( n \).
- \( L_{n,r}(k) \) : expected number of processes of class \( r \) at station \( n \)

The argument \( k \) denotes the dependence on the population vector. This vector actually determines the recursive scheme. The following recursive relations form the so-called mean value relations.
\[
S_{n,r}(k) = \begin{cases} 
\sum_{i=1}^{R} L_{n,i}(k - \varepsilon_r) + 1 \ w_n & \text{if } n \text{ is a FCFS queue} \\
w_{n,r} & \text{if } n \text{ is IS queue} \\
\sum_{i=1}^{R} L_{n,i}(k - \varepsilon_r) + 1 w_{n,r} & \text{if } n \text{ is PS queue}
\end{cases}
\]

\[\Lambda_{n,r}(k) = \frac{f_{n,r} \ k_r}{\sum_{m=1}^{N} f_{m,r} \ S_{m,r}(k)} \quad (3.2)\]

\[L_{n,r}(k) = \Lambda_{n,r}(k) \ S_{n,r}(k) \quad (3.3)\]

With \(\varepsilon_r\) denoting the \(r\)-th unit vector. The vector \(k - \varepsilon_r\) is the population with one process of class \(r\) removed. Relation (3.1) is a consequence of the arrival theorem, that states that the limiting distribution at arrival instants of a process of class \(r\) equals the limiting distribution of the network with one process of class \(r\) removed. Relations (3.2) and (3.3) are based on Little's formula.

The initialization is by \(L_{n,r}(0) = 0\) for each \(n, r\). The recursion runs through all vectors in the range of \((0, \ldots, 0)\) up to \((K_1, \ldots, K_R)\). Since the computational complexity is closely connected with the number of recursion steps \((\Pi_{r=1}^{R}(K_r + 1))\), the calculation of the MVA-scheme for even relatively small queuing networks requires a reasonable amount of calculation time. For example, the network of our model with 3 CPUs and 4 processes per class requires 15625 recursion steps. Hence, large values of \(R\) and \(K_r\) will prevent the exact evaluation of this scheme within a reasonable amount of time.

The large computational complexity is one of the reasons why we are forced to develop an approximate algorithm. A second reason forms the modeling of the CPUs as preemptive resume priority queues. The priority queues cannot be included in the MVA-algorithm without losing the exact character of the algorithm. The relation for the expected response time for such a queue will violate the arrival theorem.

Before we deal with priority stations, we will define an approximate algorithm introduced in [4], which reduces the computational complexity enormously. The Schweitzer method is based on the idea to remove recursion from the MVA-algorithm and to concentrate on the last step of the MVA-algorithm, which supplies mean values at the end-population \(K\). Instead of \(L_{n,i}(K - \varepsilon_r)\) in (3.1) in the last MVA step, we use \(L_{n,i}^{*}(K)\) where
As a consequence, the argument $K$ can be omitted, resulting in an iterative method for calculating the Schweitzer relations as shown below.

$$L_{n,i}(K) = \begin{cases} L_{n,i}(K) & i \neq r \\ \frac{K_r - 1}{K_r} L_{n,i}(K) & i = r \end{cases}$$

Initialization of $L_{n,r} = 0$ leads to an $S_{n,r}$ and an $\Lambda_{n,r}$, which can be used to determine a new $L_{n,r}$. The relations are solvable and therefore the iteration will converge to a solution. The accuracy of the solution can be determined by defining a stop criterion.

An obvious improvement of the Schweitzer method is to run the Schweitzer iteration at the population vectors $K - e_1, \ldots, K - e_R$ and to evaluate the last step of the MVA-algorithm at population $K$. This method is called the Schweitzer-FODI (see [1]) method, a first order depth improvement of the Schweitzer method.

The Schweitzer-FODI method is implemented in the VAMP package.

The iterative scheme is as follows. At each iteration, first of all the above mentioned Schweitzer relations are solved once for populations $K - e_1, \ldots, K - e_R$. Consequently, estimates for the expected number of processes at each station $n$ for each process class $r$ at those populations are known. Using these values in the MVA-algorithm at the population $K$ results in estimates for the mean values at the end-population. After each MVA step the stop criterion is applied. The sum over the process classes of $\Lambda_{n,r}(K)/f_{n,r}$ (the throughput for class $r$) in the current Schweitzer-FODI iteration is compared with the
sum in the preceding iteration. If the difference is less than $10^{-4}$, the Schweitzer-FODI algorithm is terminated.

The terminal station has been modelled as a IS-server queue and therefore the Schweitzer-FODI algorithm needs no adjustment.

The CPU stations have been modelled as pre-emptive resume priority stations with PS scheduling policy at each priority level. The expression of the expected response time for such a queue has to be approximated in the Schweitzer iterations as well as in the MVA step. In order to cope with the priority levels, the so-called Shadow Approximation has been used to obtain the expression for the expected response time. This approximation is based on the idea that processes of a certain priority level are queued for a separate CPU and don’t see processes of other priority levels. In this way the CPU consist of a number of parallel queues, the so-called shadow queues. The processes at each shadow queue receive attention from the CPU according to the fraction of time the CPU is available for that priority level. Note that this way of modeling priority queues implies the validity of the arrival theorem. Note further that a priority level may consist of several process classes. However, in our model we distinguish two priority levels per CPU in combination with two process classes per CPU, active and batch. Consequently, processes of each of the two priority levels consist of one class. Therefore, the amount of attention the processes of the lowest priority class receives, depends on the workload of the processes of the highest priority class. If we denote class 1 the highest priority class, this workload equals $\Lambda_{n,1} w_{n,1}$. If the service rate is normalized to unity, the remaining attention for the lowest priority equals $1 - \Lambda_{n,1} w_{n,1}$. In the general case with the assumption of one process class per priority group and class 1 the highest priority, the expression for the expected response time in the MVA-algorithm (3.1) for a pre-emptive resume priority station with PS at each queue satisfies

$$S_{n,r}(K) = \frac{(L_{n,r}(K - e_r) + 1) w_{n,r}}{1 - \sum_{i < r} \Lambda_{n,i}(K - e_r) w_{n,i}}$$

This approximate expression is also used in the Schweitzer iterations of the Schweitzer-FODI method (without argument $K$). In this case the throughput for populations $K - e_1$, $K - e_2$, ..., $K - e_R$ will be approximated by the throughput at population $K$. This estimation is quite accurate, since in general one process less will not result in raising the bottleneck station and a drastic throughput improvement.

The disks have been modelled on a FCFS base. Besides the computational complexity and the concept of priority queues, a third cause for developing an approximate algorithm appears: the disk workloads in our model aren’t class independent. Therefore the following obvious approximation for the expected response time for a disk has been implemented in the VAMP-algorithm.
A correction has been applied for the non-exponentiality of the disk access time (see Section 3.4). It has been assumed that the coefficient of variation of the disk access time distribution approximately equals the square root of $\frac{1}{3}$. This means a residual time of $\frac{2}{3}$ of the average access time. This correction is only applied in the Schweitzer calculations of the Schweitzer-FODI algorithm.

The VAMP-algorithm has been described for absolute workloads, while the workloads of our parameterization as described in Section 2.3.3 are relative. The tuning to these relative workloads requires no special adoptions. However, introducing the following notations will make the calculation of the algorithm smoother and more clear.

\[
S_{n,r}(K) = \sum_{i=1}^{R} L_{n,i}(K - e_r) w_{n,i} + w_{n,r}
\]

The marked expected response is relative. The notation $\Lambda_r(K)$ denotes the throughput of processes of class $r$ through the network.
The implemented Schweitzer-FODI variant in the VAMP package becomes.

- Schweitzer calculation at populations $K - e_1, \ldots, K - e_R$.

\[
S_{n,r}^* = \begin{cases} 
(L_{n,r}^* + 1) w_{n,r}^* \\
1 - \sum_{i < r} \Lambda_i w_{n,i}^*
\end{cases} \quad \text{if n is a CPU}
\]

\[
S_{n,r}^*(K) = \begin{cases} 
(L_{n,r}(K - e_r) + 1) w_{n,r}^* \\
1 - \sum_{i < r} \Lambda_i (K - e_r) w_{n,i}^*
\end{cases} \quad \text{if n is a CPU}
\]

\[
\Lambda_r = \frac{K_r}{\sum_{m=1}^N S_{m,r}^*}
\]

\[
L_{n,r} = \Lambda_r S_{n,r}^*
\]

- The MVA-algorithm at population $K$.

\[
S_{n,r}^*(K) = \begin{cases} 
w_{n,r}^* \\
\sum_{i=1}^R L_{n,i}(K - e_r) w_{n,i}^* + w_{n,r}^*
\end{cases} \quad \text{if n is a Disk}
\]

\[
\Lambda_r(K) = \frac{K_r}{\sum_{n=1}^N S_{n,r}^*(K)}
\]

\[
L_{n,r}(K) = \Lambda_r(K) S_{n,r}^*(K)
\]

28
The algorithm requires the following inputs:
- number of CPUs and disks.
- number of process classes.
- number of processes per process class.
- priority per process class at CPU.
- relative terminal, CPU and disk workload per process class.
- relative visit frequencies (in particular the disk visit frequencies)

As concluding remarks, we note that the accuracy of the VAMP-algorithm can be improved in a number of ways. First of all, the correction for the non-exponentiality of the disk access time can also be implemented in the MVA step. Secondly, the Shadow Approximation for the priority queue is very primitive while other more accurate approximation methods are available. For instance, the Completion Time Approximation [1].
3.3 Specifying Inputs

In the introduction of this Chapter, we mentioned some difficulties which appeared at performance calculations. The disk utilizations could exceed one and the response times could be negative. In this Section we will describe some efforts to avoid the difficulties by checking inputs at performance calculations.

In each performance calculation first the number of active processes per VAX has to be specified. This is followed by a search for the model parameters that correspond to the specified number of active processes. After specification of the number of batch processes per VAX to include in the performance calculation, the model parameters are stored on file, in order to be read by the module with the implemented algorithm. It seemed that many difficulties were caused by negative parameters and relative workloads exceeding one, due to inadequate calculation of the model parameters or a determination based on too few measurements. It is obvious that with these parameters a proper performance calculation couldn't be made.

In order to illustrate the effects of a model parameter determination based on too few measurements, we consider the relative terminal workload of a specific VAX, as described in Section 2.3.3. This terminal workload depends on the number of times that a process is in the LEF state. An active process in the LEF state means that this process is waiting either for terminal input or disk completion. As a consequence, we had to construct an expression for the average time of a disk IO completion, the real disk time as shown in (2.1). The real disk time contained the weighted sum over the number of disks of the following expressions.

\[
\frac{m_h}{1 - \left[ \frac{r_h}{(\text{average rates of other VAXes})} \right]} m_h
\]

Here, \( r_h \) equals the disk IO rate to disk \( h \) from disk I/Os generated by processes of the VAX for which the terminal workload is determined and \( m_h \) equals the disk access time. The expression for disk \( h \) is obtained by considering disk \( h \) an \( M|M|1 \)-queue with FCFS scheduling. The mentioned weight for disk \( h \) equals \( r_h/(\sum_{disk} r_{disk}) \).

Assume that a particular disk \( h \) has access time \( m_h=0.038 \) sec, meaning that theoretically this disk can handle at most 26.3 IO requests per second. If we assume \( r_h=30 \), the denominator becomes negative, in spite of the disk IO rates of the other VAXes. This can be interpreted by saying that disk \( h \) cannot handle the demand for service. In practise the disk can handle this disk IO rate, due to the SSFS scheduling, see Section 3.4.

If the weight of disk \( h \) is of reasonable size (0.80 should be sufficient), the disk times multiplied by the corresponding weights of the other disks cannot compensate the negative term in the sum over the number of disks. Consequently, the total real disk time becomes negative and probably results in the relative terminal workload to exceed one. In this example, measuring an incidental occurrence in few measurements resulted in the heavy weight for disk \( h \). Since, generally, the disk workload is uniformly distributed over

30
the disks, more measurements will make the weights less different.

The concept of a data file for pushing the model parameters to the algorithm, allows for a range check of some of the model parameters. VAMP only passes through those model parameters with which a proper performance calculation can be made. If the check fails, an error message occurs and the user has to specify a different number of active processes.

The following range checks are applied:
- relative disk and CPU workload per active process in $[0.001,1]$
- relative terminal workload plus relative workload for the wait time in $[0.001,1]$
- sum of these four relative workloads in $[0.003,1]$
- if the number of entered batch processes is greater than one
  then relative disk and CPU workload per batch process in $[0,1]$
  else relative disk and CPU workload per batch process in $[0.001,1]$

If the sum of the four relative workloads per active process exceeds one, there has to be a miscalculation like the one described. In one second a process can receive at most one second attention from the system resources. The difference between the determined and the maximum amount of attention is due to waiting for these system resources.

Once the model parameters belonging to the specified number of active processes have been checked, a second improvement concerning input specification becomes possible. Averaging the model parameters corresponding to $k$ active processes with the model parameters corresponding to $k - 1$ and $k + 1$ active processes, weighted by the number of measurements (samples). In this way inaccurate model parameters deducted from too few measurements can be made more reliable. Clearly, inaccurate model parameters can be caused by choosing a too small period of measurements. A large VAX could strengthen this effect, since such a VAX has a great variety in the number of active processes appearing in the samples, which results in a reduction of the number of samples corresponding to each number of active processes. Hence, a large VAX has an increased probability of producing inaccurate parameters.
3.4 Seek Optimization

3.4.1 The Disk Unit

The disk as shown in Figure 3.1 consists of a number of rotating flat magnetic *platters*, containing one or two *surfaces*.

![Figure 3.1: The Disk Unit](image)

Each surface of a platter is divided into concentric *tracks*. Each track is again subdivided into *sectors* as shown in Figure 3.2. The sector often has the size of a 512 byte page. To read or write a sector, an arm with read-write heads above each surface senses or magnetizes the stream of bits in a sector in a certain track on a certain platter. For multi-platter disks, the concept of a track gives way to that of a *cylinder* formed by logical grouping of all tracks at the same radius on each platter.

![Figure 3.2: Surface of a Platter](image)

The access time of a disk for an IO request consists of three components. The positioning of the heads is called a *Seek*. Once the read-write heads are positioned to the track (the heads move all together), it must wait for the correct sector to pass by. This is known as the *Latency Time* and on the average equals half a rotation. Once the read-write head has found the right sector, the data transfer can take place. The time needed to transfer data is called the *Transfer Time* and depends on the number of pages to be transferred.
Both the latency and the transfer time depend on the rotational speed. Further, the transfer time depends on the number of sectors per track, since this determines the transferring speed too. For the exact description of the latency and transfer time, we refer to [2].

In Table 3.1 some characteristics of VAX disks which appear in VAX/VMS-clusters are shown.

<table>
<thead>
<tr>
<th>Tracks per surface</th>
<th>RK-07</th>
<th>RM-80</th>
<th>RP-06</th>
<th>RM-05</th>
<th>RP-07</th>
<th>RA-60</th>
<th>RA-80</th>
<th>RA-81</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sectors per track</td>
<td>815</td>
<td>1118</td>
<td>815</td>
<td>823</td>
<td>1260</td>
<td>1600</td>
<td>1092</td>
<td>2496</td>
</tr>
<tr>
<td>Track-to-track seek time</td>
<td>6.5ms</td>
<td>6ms</td>
<td>10ms</td>
<td>6ms</td>
<td>5ms</td>
<td>6.7ms</td>
<td>6ms</td>
<td>6ms</td>
</tr>
<tr>
<td>Rotational speed (rpm)</td>
<td>2400</td>
<td>3600</td>
<td>3600</td>
<td>3600</td>
<td>3633</td>
<td>3600</td>
<td>3600</td>
<td>3600</td>
</tr>
<tr>
<td>Average seek time</td>
<td>36.5ms</td>
<td>25ms</td>
<td>30ms</td>
<td>30ms</td>
<td>23ms</td>
<td>41.7ms</td>
<td>25ms</td>
<td>28ms</td>
</tr>
<tr>
<td>Average latency</td>
<td>12.5ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
<td>8.3ms</td>
</tr>
<tr>
<td>Average transfer 3 pages</td>
<td>3.4ms</td>
<td>1.7ms</td>
<td>2.3ms</td>
<td>1.7ms</td>
<td>1.0ms</td>
<td>1.2ms</td>
<td>1.6ms</td>
<td>1.0ms</td>
</tr>
<tr>
<td>Average transfer 4 pages</td>
<td>4.5ms</td>
<td>2.1ms</td>
<td>3.0ms</td>
<td>2.1ms</td>
<td>1.3ms</td>
<td>1.6ms</td>
<td>2.2ms</td>
<td>1.3ms</td>
</tr>
</tbody>
</table>

Table 3.1: Characteristics of Disks used in VAX/VMS-clusters

We have given the transfer time based on a transfer of 3 and 4 pages, since these number of pages are most commonly used in VAX/VMS-clusters \(^1\). Note that for each disk, the seek time forms about 75% of the total access time.

It seemed that when a performance calculation was based on measurements with one disk heavily used, the utilization of this disk rapidly exceeded one. In order to avoid this, we have compared FCFS modeling of the disks with the disk IO request scheduling implemented in the VAX disks, which is based on the Shortest Seek First Served (SSFS) policy. As a result the VAMP-algorithm as described in Section 3.2 has been modified by means of an adjustment to the relative disk workload.

In Section 3.4.2 we will derive a distribution for the seek distance for both FCFS and

\(^1\)The number of pages to be transferred is known as **clusterson**.
SSFS scheduling, in order to compare the distributions. As a consequence, statements concerning probability characteristics of both seek times will be possible. In Section 3.4.3 we will discuss an implementation for the seek optimization. In Section 3.4.4, we will give some performance comparisons between the VAMP-algorithm and the modified VAMP-algorithm.

3.4.2 Seek Distance Distributions

The seek time depends on the arm acceleration and velocity and the data distribution over the tracks. We will assume the pages uniformly distributed over the tracks. Then we can evaluate the distribution for the seek distance $X$.

Let $m$ be the number of tracks on a disk surface, with $m$ of considerable size (see Table 3.1). Therefore, the obvious discrete distribution for the seek distance can be approximated by a continuous distribution.

We consider the read-write head at the moment that the transfer of the preceding disk IO request has ended and that the next seek begins for a process selected on FCFS base. The probability that the seek distance $X$ is greater than $s$, $s \leq m/2$, equals the probability that the read-write head has to be removed from position $y$, $0 \leq y \leq m/2$, to a position in the thickened areas of Figure 3.3.

![Figure 3.3: Transverse Section of a Disk Surface, Case $s$ less or equal $m/2$](image)

The distribution of this probability satisfies.

$$
\frac{1}{2} P_{s \leq m/2}[X > s] = \int_0^s \frac{m - y - s}{m} \, dy + \int_{s}^{m/2} \frac{m - 2s}{m} \, dy
$$

The factor $\frac{1}{2}$ is to account for the symmetric case that $m/2 \leq y \leq m$. The probability that the seek distance $X$ is greater than $s$, with $s \geq m/2$, equals the probability that the

\footnote{In the remainder we will assume $0 \leq s \leq m$.}
arm has to be removed from position $y$ to a position in the thickened area of Figure 3.4, $0 \leq y \leq m/2$.

![Figure 3.4: Transverse Section of a Disk Surface, Case $s \geq m/2$](image)

The distribution of these probabilities is shown below

$$\frac{1}{2} P_{s \geq m/2}[X > s] = \int_0^{m-s} \frac{m - y - s}{m} \frac{dy}{m}$$

The combined probability that seek distance $X$ is greater than distance $s$, $0 \leq s \leq m$ satisfies

$$P[X > s] = \frac{(m - s)^2}{m^2}$$

The MVA and Schweitzer algorithm are both based on relations between expected values. Therefore, we are interested in the first and second moment of the seek distance distribution. These moments can be determined as follows.

$$E[X] = \int_0^m s \frac{dP[X \leq s]}{s} = \frac{1}{3} m$$

$$E[X^2] = \int_0^m s^2 \frac{dP[X \leq s]}{s} = \frac{1}{6} m^2$$

Since the acceleration and arm velocity are hard to determine, we assume the relation seek time and seek distance to be linear. The following relation defines the seek time for an arm movement of $s$ tracks.

$$ST(s) = \begin{cases} 0 & \text{if } s = 0 \\ a + b \cdot s & \text{if } 1 \leq s \leq m \end{cases}$$

The $ST(1)$ equals the track-to-track seek time. Function $ST$ is completely determined by the track-to-track seek time and the average seek time which has been specified for
each VAX disk. We assume that the average seek time as shown in Table 3.1 equals the seek time for the average seek distance. The RA-81 disk, for instance, has average seek time $E[ST] = 28$ msec, $m = 2496$ and $ST[1] = 6$ msec. As a consequence $a = 5.974$ and $b = 0.026$. Therefore, the second moment of the seek time equals $965 \text{ (msec)}^2$. The coefficient of variation of the total disk access time (see [2]) approximately equals 0.25. It is obvious that the disk access time has a strong non-exponential character. Therefore, the coefficient of variation of the disk workload in the VAMP-algorithm has been lowered from the exponential value of one to the square root of $\frac{1}{3}$ (approximately 0.58).

In practice, the disk handles the disk IO requests according to the SSFS scheduling. With SSFS that process is selected for service which requires the smallest arm movement compared to the position of the arm at the moment a new request can be handled. The SSFS policy is highly discussed in the performance area. Doubts are mainly based on the suggestion that this scheduling would favor those processes which have their data concentrated in the middle of the disk. This would strongly affect the variance of the disk access time. Moreover, the implementation of the SSFS policy requires the existence of queues, whereas we would like to prevent the occurrences of such queues.

We want to obtain the first and second moment of the seek distance distribution under SSFS scheduling, given that there are $k$ processes waiting at the moment a new seek can be started. We will use a similar deduction as the distribution of the seek distance under FCFS scheduling.

Let $y$ denote the position of the read-write head ($0 \leq y \leq m/2$). The probability that the seek distance is greater than $s$, $s \leq m/2$, equals the probability that all $k$ ($k \geq 1$) processes waiting must have their data in the thickened regions in Figure 3.3. Therefore, the probability satisfies

$$\frac{1}{2} P_{s \leq m/2}[X > s \mid k \text{ processes}] = \int_0^s \left( \frac{m - y - s}{m} \right)^k \frac{dy}{m} + \int_{s}^{m/2} \left( \frac{m - 2s}{m} \right)^k \frac{dy}{m}$$

After some calculations, the following expression could be obtained for the probability that the arm has to be moved more than $s$ tracks, $s \leq m/2$ starting from any position in the whole track range.

$$P_{s \leq m/2}[X > s \mid k \text{ processes}] = \frac{2}{k + 1} \left( \frac{1}{m} \right)^{k+1} (m - s)^{k+1} - (m - 2s)^{k+1} + (1 - 2 \frac{s}{m})^{k+1}$$

For the case that $s \geq m/2$, we obtained the seek distance distribution, given that the start position $y$ is in the region between 0 and $m/2$, which satisfies
\[
\frac{1}{2} P_{s \geq m/2} [X > s | k \text{ processes}] = \int_0^{m-s} \left( \frac{m-y-s}{m} \right)^k \frac{dy}{m}
\]

Concerning the symmetry, the seek distance distribution in the case that \( s \geq m/2 \) equals

\[
P_{s \geq m/2} [X > s | k \text{ processes}] = \frac{2}{k+1} \left( \frac{1}{m} \right)^{k+1} (m - s)^{k+1}
\]

The first and second moment of the seek distance distribution on SSFS base can be obtained as follows

\[
E[X | k \text{ processes}] = \int_0^{m/2} s \, dP_{s \leq m/2} [X \leq s | k \text{ processes}]
+ \int_{m/2}^m s \, dP_{s \geq m/2} [X \leq s | k \text{ processes}]
= \frac{1}{2} m \frac{(k + 3)}{(k + 2)(k + 1)}
\]

\[
E[X^2 | k \text{ processes}] = \int_0^{m/2} s^2 \, dP_{s \leq m/2} [X \leq s | k \text{ processes}]
+ \int_{m/2}^m s^2 \, dP_{s \geq m/2} [X \leq s | k \text{ processes}]
= \frac{1}{2} m^2 \frac{(k + 7)}{(k + 3)(k + 2)(k + 1)}
\]

In the case that \( k=0 \), the distribution equals the distribution for \( k=1 \). Note that for \( k=1 \), the first and second moment of the seek time distribution with SSFS scheduling equals these moments with FCFS scheduling.

The coefficient of variation of the total access time for \( k=2 \) processes becomes 0.43, due to a reduction of the average seek time of 30% and of a reduction of the standard deviation of the seek time of 25%. Clearly, the variance of the disk access time has been increased.

37
3.4.3 Implementation of the Seek Optimization

Our aim has been to obtain a unique seek time correction for each disk appearing in Table 3.1. Further, we want a seek optimization per disk in the model, since each disk has a different number of expected processes. The adjustment we will make is based on the ratio between the average seek time with SSFS scheduling and the average seek time with FCFS. The ratio of a particular disk depends on the expected number of processes waiting for that disk, since the average seek time based on SSFS scheduling depends on the number of processes waiting. The actual implementation in the VAMP-algorithm is a multiplication of the relative disk workload of a particular disk with the mentioned ratio of that disk, in each iteration just before the Schweitzer calculations begin. The ratios will depend on the expected number of processes at the disks in the preceding iteration. We assume that the access time of each disk mentioned in Table 3.1 consists of exactly 75\% seek time and 25\% latency and transfer time.

Let \( d_1 \) denote the track-to-track time and \( d_2 \) the average seek time. Hence, \( a + b = d_1 \) and \( a + b \cdot m/3 = d_2 \). Then, \( a \) and \( b \) satisfy the following relations

\[
a = \frac{md_1 - 3d_2}{m - 3} \quad b = \frac{3(d_2 - d_1)}{m - 3}
\]

The average seek time for the SSFS scheduling, under the condition that there are \( k \), \( k \geq 1 \), processes waiting at the time a new seek can be started becomes

\[
\frac{md_1 - 3d_2}{m - 3} + \frac{3(d_2 - d_1)}{m - 3} \cdot \frac{1}{2m} \cdot \frac{(k + 3)}{(k + 2)(k + 1)} \approx
\]

\[
d_1 + \frac{3}{2}(d_2 - d_1) \cdot \frac{(k + 3)}{(k + 2)(k + 1)}
\]

This approximation will be good, since \( m \) exceeds 800 tracks for each of the disks in Table 3.1. As a consequence, the ratio between average seek time with SSFS and FCFS scheduling policy, becomes

\[
c_1 + c_2 \cdot \frac{(k + 3)}{(k + 2)(k + 1)}
\]

with \( c_1 \) equal to the quotient of \( d_1 \) and the average seek based on FCFS policy and \( c_2 \) equal to the quotient of \( \frac{3}{2}(d_2 - d_1) \) and the same average seek time. In order to estimate the unique values of \( c_1 \) and \( c_2 \), we will now specify these constants for each disk of Table 3.1, as shown in Table 3.2.

We have chosen for \( c_1 = 0.20 \) and \( c_2 = 1.20 \). It has to be stated that these constants can easily be changed in the modified VAMP-algorithm. A suggestion might be to use the
constants in Table 3.2 by linking these constants to the type of the disk, which is stored on file. The RP-06 disk which is used in the VAX/VMS-cluster at the WUA has obviously the least accurate constants.

The adjustment ratio for the seek time in the \( n \)-th Schweitzer-FODI iteration for disk \( h \) becomes

\[
a_h^{(n)} = \sum_{k=1}^{K_1+\cdots+K_R} p_{k,h}^{(n-1)} \left[ c_1 + c_2 \frac{(k + 3)}{(k + 2)(k + 1)} \right] + p_{0,h}^{(n-1)}(c_1 + \frac{2}{3} c_2)
\]

with \( p_{k,h}^{(n-1)} \) the probability that in the \((n-1)\)-th iteration \( k \) processes are waiting at disk \( h \). We assume \( p_{k,h}^{(n)} \) geometrically distributed with mean \( L_h^{(n)} \), the expected number of processes at disk \( h \) after the MVA step at population \( K \) of the \( n \)-th iteration. \( L_h^{(n)} = \sum_{r=1}^{R} L_h^{(n)}_{r} \), with the \( L_h^{(n)}_{r} \)'s the recursive elements \( L_h^{(n)}_{r}(K) \) in the MVA-algorithm at iteration \( n \). The distribution of the \( p_{k,h}^{(n)} \) is given by

\[
p_{k,h}^{(n)} = (\rho_h^{(n)})^k (1 - \rho_h^{(n)})
\]

with \( \rho_h^{(n)} = L_h^{(n)}/(L_h^{(n)} + 1) \). Therefore, the correction ratio for the seek time becomes (with \( \rho = \rho_h^{(n-1)} \))

\[
a_h^{(n)} = (1 - \rho) \sum_{k=1}^{K_1+\cdots+K_R} \rho^k \left[ c_1 + c_2 \frac{(k + 3)}{(k + 2)(k + 1)} \right] + (1 - \rho)(c_1 + \frac{2}{3} c_2)
\]

\[
= c_1(1 - \rho^{K_1+\cdots+K_R+1}) + c_2(1 - \rho) \left[ \frac{2}{3} + \sum_{k=1}^{K_1+\cdots+K_R} \rho^k \frac{(k + 3)}{(k + 2)(k + 1)} \right]
\]

The actual adjustment in the VAMP-algorithm at disk \( h \) per process class \( r \), \( r = 1, \ldots, R \), becomes

\[
w_{h,r}^{(n)} = w_{h,r} \cdot f_{h,r} \cdot \left[ a_h^{(n)} \cdot \frac{3}{4} + \frac{1}{4} \right]
\]

The disk workloads are adjusted just before the first Schweitzer calculation starts at the \( n \)-th iteration. Note that in the first iteration all \( a_h^{(1)} \)'s equal one, since for each disk-\( h \) and process class \( r \), \( L_h^{(n)}_{r}(K) \) is initialized by zero.

<table>
<thead>
<tr>
<th></th>
<th>RK-07</th>
<th>RM-80</th>
<th>RP-06</th>
<th>RM-05</th>
<th>RP-07</th>
<th>RA-60</th>
<th>RA-80</th>
<th>RA-81</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>0.18</td>
<td>0.24</td>
<td>0.33</td>
<td>0.20</td>
<td>0.22</td>
<td>0.16</td>
<td>0.24</td>
<td>0.21</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>1.23</td>
<td>1.14</td>
<td>1.00</td>
<td>1.20</td>
<td>1.17</td>
<td>1.26</td>
<td>1.14</td>
<td>1.18</td>
</tr>
</tbody>
</table>

Table 3.2: Constants used in Adjustment Ratio
3.4.4 Seek Optimization and Performance

The adjustment of the disk workload as described in Section 3.4.3 will especially affect the disk utilizations. Concerning the actual adjustment, it is obvious that a heavily used disk will have a greater utilization reduction than a disk which is hardly used, since a higher average number of processes waiting for IO completion will certainly more reduce the seek distance and thus seek time. In order to verify this statement, we will compare the performance with and without the seek optimization. The performance results are deducted from measurements taken on the VAX/VMS-cluster at the WUA in the beginning of June 1988 and are shown in Table 3.3.

<table>
<thead>
<tr>
<th></th>
<th>utilization</th>
<th>response time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FCFS</td>
<td>SSFS</td>
</tr>
<tr>
<td>VAX-1</td>
<td>0.204</td>
<td>0.215</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.333</td>
<td>0.334</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.351</td>
<td>0.351</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.539</td>
<td>0.542</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>utilization</th>
<th>adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FSFS</td>
<td>SSFS</td>
</tr>
<tr>
<td>disk-1</td>
<td>0.507</td>
<td>0.478</td>
</tr>
<tr>
<td>disk-2</td>
<td>0.198</td>
<td>0.197</td>
</tr>
<tr>
<td>disk-3</td>
<td>0.053</td>
<td>0.053</td>
</tr>
<tr>
<td>disk-4</td>
<td>0.083</td>
<td>0.083</td>
</tr>
<tr>
<td>disk-5</td>
<td>0.148</td>
<td>0.148</td>
</tr>
<tr>
<td>disk-6</td>
<td>0.213</td>
<td>0.211</td>
</tr>
<tr>
<td>disk-7</td>
<td>0.107</td>
<td>0.106</td>
</tr>
<tr>
<td>disk-8</td>
<td>0.166</td>
<td>0.165</td>
</tr>
<tr>
<td>disk-9</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>disk-10</td>
<td>0.130</td>
<td>0.130</td>
</tr>
<tr>
<td>disk-11</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>disk-12</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>disk-13</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>disk-14</td>
<td>0.675</td>
<td>0.632</td>
</tr>
</tbody>
</table>

Table 3.3: Performance Comparison Seek Optimization

The link between the mentioned model names and the names in the WUA-cluster have been depicted in Table 5.12. Disk USERD in this Table has been added in the beginning
of July 1988, so the measurements in June 1988 doesn’t include this disk.

For a certain population without batch processes, the performance results with and without the seek optimization have been expressed in terms of the CPU utilizations of each of the VAXes in the WUA-cluster and the response time per active process for a VAX-11/750 CPU second. Further, for each of the disks the utilization in both performance calculations. Concerning the performance calculation with seek optimization, we have also given the adjustment ratio.

We see that especially disk-1 and disk-14 are heavily used and that the response time for one CPU second for VAX 1 is high compared to the other VAXes. Indeed, we see for disk-1 and disk-14 the highest adjustment ratio, resulting in a large utilization reduction of about 6%. We see that the CPU utilization of VAX-2 is slightly lower, while the utilization of VAX-3 has not changed. However, VAXes 1 and 4 have a definite lower utilization. Since for all VAXes the response time has been reduced, the throughput for each active process class through the network of our model has been improved. Clearly, the heavily used disks were bottlenecks. The response time reduction of VAX-4 is due to the fact that many disk IOs generated by processes of this VAX were to disk-1, the system disk. The explanation for the reduction of the response time of VAX-1 is a bit complicated. At the time of the performance calculation, VAX-1 had three local disks. This means that VAX-1 has to control the disk IO traffic between each VAX and the three local disks. The disk IO control costs an unknown amount of CPU time. At the moment, VAMP distributes this CPU time over the CPU workload of the active processes of VAX-1. This results in a CPU workload per active process which is higher than the CPU workload in reality. Therefore, the response time for one second CPU time at VAX-1 as a result of the performance calculation differs from this response time in reality. Clearly, a heavily used local disk like disk-14 will make the performance of VAX-1 less reliable. Probably, the response time for one VAX-11/750 CPU second is much too high, because the utilization of VAX-1 is only about 20%. Compared to the other VAXes, VAX-1 has an enormous utilization reduction. An explanation for this could be the high CPU workload, which definite increases the utilization of VAX-1 at the slightest throughput improvement.

3.5 Suggestions for Future Improvements

Concerning the input specification for the performance calculation, the accuracy of the performance should be improved by averaging the model parameters corresponding to the specified number of \( k \) active processes with the parameters corresponding to \( k - i, \ldots, k + i \) active processes, with \( i \) of varying character. At the moment, \( i=1 \) for each VAX in the configuration. However, adapting the size of \( i \) to the size of the VAX should certainly supply more accurate model parameters for the performance calculation. A small size of \( i \) should be sufficient for a small VAX, while a large VAX with many active processes and few samples per number of active processes could have a large value of \( i \).

Further, the accuracy will benefit from deleting those parameters at the averaging, which
are deducted from few measurements.

The above mentioned improvement can be avoided by another selecting procedure for the model parameters. At the moment, the model parameters are selected for each measured number of active processes. Especially when measuring a great VAX, this results in inaccurate as well as huge amounts of model parameters. Therefore, selecting the model parameters in groups of 3, 4 or 5 active processes should possibly solve these problems. Moreover, if the group size can be varied, the VAMP package can better cope with VAXes of different sizes.

Concerning the VAMP-algorithm, there is still a possibility that the utilizations at performance calculations of the existing configuration of both VAXes and disks exceed one. This is only the case when batch processes are entered at the performance calculation. Actually, the Shadow Approximation, which only affects the CPU workload per batch process, pushes the utilizations above one. It is well known that when the CPU workload per active process is large compared to the unadjusted workload per batch process, Shadow Approximation is inaccurate. As a consequence, the disk workload per batch process is very large compared to the CPU workload, resulting in a not measured disk utilization increment.

On the contrary, Shadow Approximation can result in a CPU workload per batch process which is too high for the CPU to cope with. Under certain circumstances the Shadow Approximation should be corrected in order to avoid the utilization to exceed one. However, we haven’t had time to develop such a correction. Therefore, we have programmed an error message to indicate that the utilization of a VAX or disk exceeds one, which states that the number of batch processes is too high for proper performance calculation.
Chapter 4

User Interface

4.1 Introduction

Usage of any DSS involves a dialogue between users and the DSS. The user interface provides a framework for this dialogue, within which information is presented and inputs are given. Even if a DSS provides extremely powerful functions, it may not be used if the dialogue is unacceptable. Therefore, our aim has been to develop a user interface which is effective as well as understandable for the ultimate users we had in mind, the system managers of VAX/VMS-clusters.

Before we look at the user interface of the VAMP package, we will first give the separate programs which form the VAMP package.

1. The VAX/VMS Operating System includes a standard monitoring program (MONITOR), which measures current system characteristics at regular intervals of 3 minutes throughout the day. (see Figures 2.3, 2.4 and 2.5).

2. At the end of the day, for each VAX a program (JOINMON) extracts relevant data from the MONITOR information and stores these data on file (the daily data file).

3. A program (VAMP) which provides the actual user interface of the VAMP package. It allows for manipulating the daily data files, in order to obtain the performance of configurations within the VAX/VMS-cluster family.

   - An implemented algorithm to calculate models of VAX/VMS-clusters in order to obtain certain performance characteristics.

   - The parameters for the model of the existing configuration can be deducted from (amalgamated) daily data files. These parameters can be used to obtain the performance in a certain period.

   - A number of routines for altering the model and the parameters of the measured VAX/VMS-cluster, in order to make predictions about the performance of a modified system configuration.
Clearly, calculating the model of the existing configuration provides the current performance, while usage of any other model will provide performance predictions.

Clearly, the VAMP program contains the user interface of the VAMP package. In the following we will speak of the user interface program instead of the VAMP program. In this Chapter we will only consider the third program of the VAMP package, since the programs MONITOR and JOINMON are implemented in batch jobs and require therefore no dialogue.

The user interface is based on menus of options and posed questions. The menus are used to select the functions of VAMP. The questions are posed within each function. Starting the third program leaves the user at the menu at VAMP level, denoted by the prompt VAMP>, see Figure 4.1.

**Within VAMP you can choose the following options:**

- display the system configuration
- determination of model parameters
- amalgamation of daily data files of a specific VAX into monthly data files
- performance calculation of an example of the configuration (model parameters have to be determined)
- build clusters
- remove clusters
- performance calculation of an example of a built cluster (model parameters have to be determined)
- Stop and leave VAMP

**Figure 4.1: Option Menu at VAMP level**

Prompts will appear each time VAMP requires input. In order to avoid confusion, each function has been given its own unique prompt. For example, the function for building clusters has prompt BUILD>.

The user interface includes help options, which can be invoked after each prompt in each function by typing the "?"-command. As a consequence, relevant information will be displayed concerning the input VAMP requires at the particular entry the "?"-command has been entered. For example, typing the "?"-command at the VAMP level returns a display of the functions of VAMP, together with some handy abbreviations.

It might occur that the user wants to stop the dialogue within a function, in order to return to the VAMP level. Therefore, we have developed the "(q)uit"-command\(^1\), which allows for returning to the VAMP level. It has to be remarked that the "(q)uit"-command

---

\(^1\)The expression within the delimiters denotes the command of the minimal abbreviation.
cannot be used at each entry. Typing the "?"-command first, will tell the user if it is allowed to return to VAMP level.

Further, the user interface includes several error messages, all in format "error: ". Some of the error messages appear in each function, while others have been developed to indicate a special occurrence within a function. The global error messages are connected with the read procedures for reading integers and reals or arrays of integers and reals. A syntax error is answered as follows:

  error: check your syntax, REAL expected : enter again
  error: check your syntax, INTEGER expected : enter again

If the syntax check has succeeded, a range check is applied (if necessary). If the range check fails, the following message occurs.

  error: inspected value out of range: enter again

These error messages will continue to occur, until VAMP accepts the values.

Entering large arrays of integers and reals have been made easier. Entering a random number of values is answered as follows.

    BUILD 0.2 0.7 0.5 0.0
    number of expected reals 3
    BUILD 0.45 0
    number of expected reals 1
    BUILD 0.01 0.7

For example, the specification of an array of reals in the function for building clusters. It has to be remarked that entering too many values does not bother. The last value of 0.7 in our example is not accepted.

Other global error messages occur when the read procedures for reading options cannot recognize the entered option.

  error: check syntax OPTION ("?" for information)

It has to be said, that the developed read procedures for reading options can easily be extended, if new functions are added to VAMP. Further, the read procedures are developed with the intention of combining successive entries within a function into one entry. This certainly affects the effectiveness of the user interface in a positive way. In Section 4.6 we will make some suggestions for combining successive entries. Implementing an extended command language will bring the VAMP package a step closer towards a DSS. Therefore future development of the VAMP package has to include the command language.
At some entries, the user has to specify a name for either an input or output file. These files can be placed in several directories or subdirectories, which the user can define in CONFFILE.DAT, the input file for the programs JOINMON and VAMP with relevant information about the VAX/VMS-cluster. Consequently, the mentioned entries require only the name of the file, not the disk or directory.

In Section 4.2 we will discuss the monologue of the user interface program at the start of each user interface session. Since the functions for displaying the system configuration and stopping the program doesn't require a dialogue too, we will discuss these functions in this Section.

The remaining functions are divided in three groups.

1. Management of daily data file
   - determination of model parameters
   - amalgamation of daily data files of a specific VAX into monthly files

2. Performance calculation of existing configuration
   - performance calculation of an example of the configuration

3. Cluster management
   - build clusters
   - remove clusters
   - performance calculation of an example of a built cluster

The daily data files are files with data obtained from measuring a VAX at one day. The names of all daily data files include the name of the corresponding VAX and the date of creation. The measured data contain the items mentioned in Section 2.3.3. In Section 4.3 we will discuss the user interface to manage the daily data files.

In order to avoid confusion, we call a configuration a theoretical cluster, if this configuration has been obtained by altering the model for the existing configuration of the VAX/VMS-cluster. Clusters are created in the function for building clusters and include e.g. an extra disk or a VAX processor with more main memory. The user interface involved in obtaining the performance of the existing configuration will be described in Section 4.4.

Clusters are built for making performance predictions. The user can create a great variety of clusters, which enables the user to predict the effects of many configuration changes upon the performance. Further, the user can easily compare the performance of different clusters, thus choosing the best alternative for adapting the existing system configuration. The management of the clusters will be described in Section 4.5.
The program for the user interface contains several program parameters, also known as *constants*. These constants have been implemented in order to control the calculation time of the procedures within the user interface program and to ensure the displays appearing in the dialogue to be acceptable. In Appendix B we will define each program parameter of the VAMP package.

### 4.2 Starting the User Interface

Before the user sees the menu at VAMP level of Figure 4.1, some relevant information about the VAMP package and the system configuration is displayed. First of all, the following information appears.

```
The VAX/VMS Analysis and Measurement Packet (VAMP)
VAMP offers the possibility to calculate the performance of a VAX/VMS configuration by means of a queuing network model and collected measurements of the configuration. Furthermore it is possible to calculate a prediction of the performance of a cluster which can be created by altering the configuration.

Prompts appear each time the program expects data entry. Each function of the program has its own unique prompt. At each prompt, the '?'-command supplies orientating information.

For running the program the following files in your directory are necessary:
- CONFFILE.DAT (containing relevant configuration data)
- CLUSFILE.DAT (containing built clusters)
- for each VAX in the configuration at least one file, obtained by MONITOR, of the following form:
  <disk>[(<maindirectory>)(<subdirectory>)<date>].DAT

Pressing the return-key, results in a display of the system configuration, divided over two screens. The first screen is filled with relevant information about the VAXes in the VAX/VMS-cluster. The name of the VAX in the model is linked with the type, speed compared to the VAX-11/750 processor and main memory size of the corresponding VAX in the configuration. An example of the first screen is shown in Figure 4.2.
```
Figure 4.2: An Example of the First Screen of System Configuration Display

The second screen contains relevant information about backing storage. The name of each disk in the model is linked with the name, type and speed (access time) of the disk in the configuration. An example of this screen is shown in Figure 4.3.

Figure 4.3: An Example of the Second Screen of System Configuration Display

The second screen is obtained by pressing the return-key after having seen the first screen. If the user presses this key another time, the menu at VAMP level appears.

At the moment, the only query function is a display of the existing system configuration. This function can only be invoked at VAMP level by entering the "(d)isplay"-command. The screen will be filled with the displays of Figures 4.2 and 4.3. It serves the purpose of verification.

Terminating the user interface session can only legally be done by entering the "(s)top"-command at VAMP level.

4.3 Management of Daily Data Files

4.3.1 Model Parameter Determination

Invoking this function can only be done at VAMP level by entering the "(m)odel parameter determination"-command. The function has been given the prompt MOD/PAR>.
The function to determine model parameters has been developed to generate model parameters belonging to a certain period of measuring the VAX/VMS-cluster, in order to obtain the performance of that period. Further, this function stores the calculated model parameters of a period, so that they can be used more than once. The model parameters are used in the functions to calculate an example of the existing configuration or a built cluster.

The model parameters of a period are deducted by first joining and averaging the daily data files for each VAX and for each day in the period into a file with the same format as the daily data file of that VAX. This is followed by the calculation of the model parameters as described in Section 2.3.3. If the determination has succeeded, for each VAX the file with the joined and averaged daily data files is deleted and the model parameters are stored on file.

Now, we will describe each of the entries within this function which lead to a successful model parameter determination.

First, the user has to answer the following question.

Before you enter a period in which you would like to have determined the model parameters, you have to specify whether the measurements in the weekends will be (I)ncluded or (E)xcluded in the determination.

If the user is only interested in the system workload of interactive processes, he will enter the "(e)xcluded"-command. However, if the user is also interested in the behaviour of the batch processes, entering the "(i)ncluded"-command should be considered. Consequently, a period has to be specified, by entering respectively the start date and end date of the period (format "dd mm yy" or "d m yy"). Entering another format results in the following error message.

error: DATE expected (dd mm yy)

If the date of the start of the period is after the end date, the following error message occur.

error: START-DATE greater than END-DATE

If VAMP accepts the entered period, the user has to specify a name for a file in which the model parameters have to be placed. The default name is PAROUT.DAT. This entry accepts the "(q)uit"-command, which means that the filename cannot begin with the character "q".

If at least one of the VAXes has no daily data files in the specified period, the following
error message could occur.

error: no measurements in period on VAX 1. Enter new period

Clearly, the user is returned to the entry of the period specification.
If each VAX has at least one daily data file in the period, the determination will proceed.
The screen will be filled with some information concerning the daily data files.

An example can be seen below. Per VAX the days without measurements are reported.
These days have no daily data file of the mentioned VAX. If the weekends are excluded,
the days without measurements in the weekends are not reported.
If the determination has succeeded, VAMP displays the relative workloads per active
process corresponding to the measured number of active processes. The occurrence of
"* * * * *" indicates that the determination of that particular parameter has failed.
Probably, because the number of samples measured with the number of active processes
Corresponding to the parameter is too small.
This function is closed with the message that the model parameters have been placed in
a file with the specified name.

4.3.2 Amalgamation of Daily Data Files of a Specific VAX into
Monthly Data Files

The function can only be invoked at the VAMP level by entering the "(a)malgamation"-
command. The prompt for this function is AMALG>

The philosophy behind this function is to reorganize the daily data files by joining a
number of these files into one file, followed by the deletion of the daily data files files.
Moreover, this function reduces the private memory allocation.
We have chosen to amalgamate the daily data files which have been obtained by measuring
a VAX in one and the same month. Hence, monthly data files are created. Further,
we have chosen to create two monthly data files out of the amalgamation. One with the
measurements in the weekends included and one with these measurement excluded. We
make a distinction between the measurements in the weekends and on working days, since
the system workload of interactive processes is not representative in the weekends. The
measurements in the weekends mainly concern the system workload of batch processes,
since generally in the weekends few interactive processes are logged into the computer
system. It has to be remarked that the daily data files of public holidays which do not
match weekends are joined with the daily data files corresponding to working days, in
spite of the fact that the system workload on these days approximates the workload in
the weekends. If the user wants to exclude the daily data files of these public holidays in the amalgamation, the names of these files can be altered. The monthly data files can be used in the model parameter determination (VAMP reports the usage of a monthly data file in the model parameter determination). Consequently, the model parameters belonging to e.g. the last week of the corresponding month equal the parameters of the whole month. This certainly affects the reliability of the model parameters. Therefore, only old daily data files should be amalgamated, since these daily data files have a small probability that they will be used in the future for performance calculations. To give an idea, we amalgamate the daily data files about five months after creation.

Invoking this function of VAMP results in the following question.

You are now able to amalgamate the daily data files of a specific VAX into one monthly data file with measurements in weekends included and one monthly data file with weekends excluded. After the amalgamation each daily data file is to be deleted! Do you really want to do this? (Y/N)

AMALG

This is an important possibility to quit the proceeding of this function, since the daily data files are automatically deleted from the directory. If the user wants to continue, the "(y)es"-command has to be entered. Going back to the VAMP level is done by entering the "(n)o"-command.

If the user continues, the number of a VAX has to be specified, followed by a specification of the month and year (format "mm yy" or "m yy") of the daily data files the user wants to amalgamate.

If the specified VAX hasn't any daily data files in the specified month and year, the following error message could occur.

error: no measurements in AUG 1988

If the monthly data files have already been made for the specified VAX, it is useless to proceed this function. The following error message indicate the occurrence of this event.

error: Monthly data files of month MAR 1988 have already been made for VAX 3

Both errors lead to a return to VAMP level. Hence, the user is not forced to continue amalgamating with files of another month.

If the error messages do not appear, VAMP starts amalgamating the files of the specified month and year. During the amalgamation, VAMP displays the names of all currently deleted files.

The function is closed with mentioning the names of both created files.
4.4 Performance Calculation of Existing Configuration

In this context, one function is of importance, the function to calculate the performance of an example of the existing configuration. This function can only be invoked from the VAMP level by entering the "(e)xample (co)nfiguration performance"- command. Prompt PERF/CONF >. With this function the user can obtain performance statistics about the existing configuration over a period, preceding to the day this function is invoked. It serves the purpose of signalizing bottlenecks.

First of all, the user has to enter a name of a file with model parameters. The default name is PAROUT.DAT. Since each performance calculation needs model parameters, there has to be at least one file containing model parameters. The model parameters correspond to a certain period. In the function to determine the model parameters this period has been specified.

All files with model parameters are placed in one and the same directory, defined in file CONFFILE.DAT. If this directory contains no file with the specified name, the following error message could occur.

```
error: file USER4:WSCP:PERF.VAM:PAR]modpar.dat not in directory
```

Consequently, a new file name has to be entered. If VAMP locates the file with the specified file name, an error message occurs when this file does not contain model parameters.

```
error: file USER4:WSCP:PERF.VAM:PAR]modpar.dat contains no model parameters
```

Occurrence of an error like this, results in returning to the VAMP level.

If the file with the specified name contains model parameters, VAMP displays for each VAX and each number of active processes, the number of samples (intervals) which were used to deduct the model parameters corresponding to these number of active processes, see Figure 4.4.

Numbers of active processes other than the numbers displayed weren't measured on any VAX in the period of measurement. The statistic indicates whether model parameters are reliable or not. The parameters corresponding to 2, 3 and 13 active processes respectively on VAXes 1, 2 and 3 will be more reliable than the parameters corresponding to 5, 8 and 20 active processes. Further, this display gives an idea about the average number of active processes during the period of measurement. However, the average number of interactive processes per VAX remains unknown. The only statement that can be made is that this number is at least equal to the average number of active processes. Recall that the inactive interactive processes are eliminated from the measurements (see Section 2.3.3).

Next, per VAX the number of active processes has to be specified, which the user wants
to include in the performance calculation. For each VAX, the specified number of active processes has to be greater or equal to one, since it is supposed that performance calculations are made in order to obtain information about active processes. In order to help the user with this specification, VAMP displays per VAX the average number of active processes measured in the period of measurement.

Average number of ACTIVE processes during the period 10JUL1988 - 18JUL1988:

VAX 1: 1
VAX 2: 3
VAX 3: 10

Enter for the configuration the desired number of ACTIVE processes per VAX.

PERF/CONF

Clearly, VAMP will calculate instantaneous performance, since the number of processes is fixed. If the user wants to know the performance of the computer system at busy moments, the entered number of active processes will exceed the average number. On the contrary, the performance at the quiet moments could be obtained by entering a low number of active processes.

Entering the number of active processes can lead to several error messages. If the model parameters belonging to the specified number of active processes equal zero the following message could occur.

ERROR 1: no measured intervals VAX 1 with 7 active processes

Apparently, in the period of measuring, all samples deducted from VAX-1 contained an-
other number than 7 active processes.
As a consequence, VAMP returns to the entry of specification of the number of active
processes.
If the model parameters corresponding to the specified number of active processes are
out of range (see Section 3.3), VAMP could report the following error message.

```
error: number of measured intervals VAX 2 with 10 active processes
is too low for proper performance calculation.
```

Consequently, VAMP returns to the entry of specification of the number of active pro­
cesses per VAX. The VAX which caused the error requires another number of active
processes. The last two error messages will continue to occur until VAMP accepts the
model parameters belonging to the specified number of active processes.
Secondly, per VAX the number of batch processes has to be specified. In order to get
an idea how many batch processes to include in the performance calculation, VAMP
displays per VAX the average number of batch processes at the samples with the earlier
specified numbers of active processes. For an example of this display see below.

```
Average number of BATCH processes at this given number of active processes:

  VAX 1: 2.3
  VAX 2: 1.0
  VAX 3: 7.0

Enter the number of BATCH processes per VAX.
```

Entering the number of batch processes per VAX could again supply error messages. If
the model parameters per batch process of a certain VAX are out of range (see Sec­
tion 3.3), the following message could occur.

```
error: too many batch processes at VAX 1 for proper performance
```

VAMP returns to the entry for the specification of the number of batch processes per
VAX. In this case, entering a number of zero batch processes for VAX-1 will succeed.

If VAMP accepts the model parameters for both active and batch processes, the perfor­
manCe calculation can start. First, the user has to specify a name for a file in which
the performance results have to be placed. Default name is FODOUT.DAT^2. The files
with the performance results are placed in one and the same directory, defined in the file
CONFFILE.DAT.
When the calculation has succeeded, VAMP displays some relevant information of the
output file, see Figure 4.5.

---

^2The default name is derived from the name of the algorithm, Schweitzer-FODI.
If the utilization of one of the system elements is displayed as "* * * * *", then the algorithm which has calculated the performance couldn't prevent this utilization to exceed one. Recall, that this is caused by the approximation method for handling the lower priority batch processes at the CPU (see Section 3.2). Hence, if the utilization exceeds one, this is due to too many batch processes. Consequently, the results of the batch processes are not reliable.

Remember that we have to make use of relative workloads (see Section 2.3.3) and that as a consequence the performance results, which are based on these workloads, are relative. The performance results as displayed in Figure 4.5 include the relative wait time, the relative response time and the response time per CPU second. All these results are defined per process.

The relative wait time is the fraction of time a process is waiting for service at CPU and disk.

Generally, the response time of a process equals the time for waiting and receiving service at CPU and disk. However, the relative response time is the fraction of time a process is at CPU and disk. For batch processes this fraction always equals 1.000, since batch processes require no terminal input and circulate between CPU and disks.

The response time per CPU second equals the response time for a process which needs exactly one second CPU calculation time. It has to be remarked that when comparing the response times per CPU second of VAXes of different speed, the response times has to be normalized to the same CPU second. Our normalization second is the CPU...
second of the VAX-11/750 \(^3\) machine. For each VAX in the VAX/VMS-cluster, the speed compared to the VAX-11/750 can be displayed by choosing the query function at VAMP level (display system configuration). This function is closed with the message that all performance data, inputs as well as outputs, have been placed in a file with the specified file name.

### 4.5 Cluster Management

#### 4.5.1 Build Clusters

Invoking this function can only be done from the VAMP level, by entering the "(b)uild"-command. Prompt BUILD>.

In the VAMP package, clusters are theoretical configurations created by altering the existing configuration, the configuration which is measured. We will now describe in what ways the configuration can be altered.

In order to select the options for altering the configuration, we have developed a second option menu, a menu at BUILD level, see Figure 4.6.

![Option Menu at BUILD level](image)

It is allowed to store maxclusnumber\(^4\) clusters.

We have developed a display for the clusters based on the display of the existing system configuration. This display costs two or three screens. The first screen displays information about the VAXes, divided over the VAXes which already existed and the VAXes which have been added in the cluster.

Compared to the display of the system configuration, the display of the existing VAXes has been extended by one column, in order to indicate whether this VAX has been modified in the cluster. If such a VAX has been changed the type of the VAX is given by a questionmark. Concerning the added VAXes, the speed compared to the VAX-11/750, main memory size and the number of the existing VAX which equals the added VAX are given, see Figure 4.7.

\(^3\)For historical reasons, we use the VAX-11/750 as reference. However, in the VAX-world the VAX-11/780 is considered the standard.

\(^4\)VAMP parameter. At the moment this parameter equals 5.
VAXes of the CONFIGURATION:

<table>
<thead>
<tr>
<th>Type</th>
<th>Speed cf.</th>
<th>Main Memory size</th>
<th>Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>7</td>
<td>18.0 MB</td>
<td>yes</td>
</tr>
<tr>
<td>VAX-2</td>
<td>VAX 8600</td>
<td>24.0 MB</td>
<td>no</td>
</tr>
<tr>
<td>VAX-3</td>
<td>VAX 8700</td>
<td>32.0 MB</td>
<td>no</td>
</tr>
</tbody>
</table>

VAXes which are added in the CLUSTER:

<table>
<thead>
<tr>
<th>Type</th>
<th>Speed cf.</th>
<th>Main Memory size</th>
<th>Workload equals most with existing VAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-5</td>
<td>6.0</td>
<td>24.0 MB</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.7: An Example of the First Screen of the Cluster Display

The second screen contains information about background memory. More precisely, name, type and speed (access time) of each disk in the cluster. If the speed of an existing disk has been altered, the type is denoted by question marks. The name and type of an added disk is also denoted by question marks.

Number of disks in the CLUSTER: 17

<table>
<thead>
<tr>
<th>Disk</th>
<th>Name</th>
<th>Type</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VAXVMS</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>2</td>
<td>USER</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>3</td>
<td>USER2</td>
<td>RA-81</td>
<td>0.041</td>
</tr>
<tr>
<td>4</td>
<td>USER3</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>5</td>
<td>USER4</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>6</td>
<td>USERS</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>7</td>
<td>FEI1</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>8</td>
<td>FEI2</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>9</td>
<td>USER9</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>10</td>
<td>FEI3</td>
<td>RA-81</td>
<td>0.038</td>
</tr>
<tr>
<td>11</td>
<td>INSTAL</td>
<td>RA-600</td>
<td>0.052</td>
</tr>
<tr>
<td>12</td>
<td>SYS1</td>
<td>RF-06</td>
<td>0.041</td>
</tr>
<tr>
<td>13</td>
<td>USERC</td>
<td>RF-06</td>
<td>0.041</td>
</tr>
<tr>
<td>14</td>
<td>USEPD</td>
<td>RF-06</td>
<td>0.041</td>
</tr>
<tr>
<td>15</td>
<td>SCRATC</td>
<td>RF-07</td>
<td>0.033</td>
</tr>
<tr>
<td>16</td>
<td>?????</td>
<td>???</td>
<td>0.038</td>
</tr>
<tr>
<td>17</td>
<td>?????</td>
<td>?????</td>
<td>0.038</td>
</tr>
</tbody>
</table>

Figure 4.8: An Example of the Second Screen of the Cluster Display

If the cluster has a modified diskload distribution, the diskload transfers appear in the display of Figure 4.8. In order to ensure that the display of the diskload transfers is acceptable in the case that the cluster contains many disks, this display isn't added to the display of Figure 4.8. The diskload transfers are displayed on a third screen if a cluster contains more than 11 disks.

Besides the cluster display, we have developed a brief survey of all stored clusters with identifying information. This survey is used at the entries in the cluster management where the user has to choose a number of a cluster. In Figure 4.9 we have given an example of the cluster survey.
All the clusters in the example of Figure 4.9 have been built out of a configuration consisting of four VAXes and fifteen disks. This survey contains all possible messages.

Cluster 1 has been built by choosing the option to extend a VAX for VAX-1 and the option to add a VAX. The added VAX-5 is a copy of VAX-2. An added VAX which is quickened or extended is not displayed in the cluster survey. Further, disk-3 has been replaced by a slower disk. The option to add a disk has been invoked twice.

Cluster 2 contains a quickened VAX-3 and a VAX-4 which has been quickened as well as extended. One disk has been added.

Cluster 3 has been created by choosing at least once the option to change the diskload of the disks. This message only occurs when no other build option (all options in the menu at BUILD level except the display and store option) has been invoked.

Cluster 4 corresponds to the cluster in the display in Figures 4.7 and 4.8.

Finally, cluster 5 has been stored without invoking one of the build options.

Now that we have defined the displays appearing in the cluster management, we can look at the data entries in the function for building clusters.

When there aren't any clusters stored, the user is directly confronted with the option menu at BUILD level, see Figure 4.6. It is obvious, that at this moment the cluster, cluster 1, equals the existing configuration.

If in earlier sessions of executing the user interface program, some clusters have been built, first the user has to answer the question whether a new cluster will be built or some additional building will be made with an existing cluster. If the number of stored clusters equals maxclusternumber, the menu at BUILD level is preceded by the cluster survey with the message:

```
error: number of clusters exceeds limit. You must remove at least one cluster
```

Only after having removed at least one cluster, the normal procedure can continue. That is, displaying the menu of Figure 4.6 when the user wants to build a new cluster and
displaying the cluster survey with the question which cluster the user wants to continue building otherwise. In the latter case, entering a number of a cluster is followed by the cluster display of this cluster. This display is followed by the question whether the user wants to work with this cluster or not. Entering the "(y)es"-command results in the display of the menu at BUILD level, together with the number of the cluster which the user has chosen to continue building with. Entering the "(n)o"-command returns to the cluster survey with the question which cluster the user wants to proceed building.

Anyway, the user can only start building from the moment the menu at the BUILD level appears. During the building session, this menu will not appear again. However, entering the "(?)"-command at the BUILD level supplies handy abbreviations of the build options. We will now describe the user interface within each build option. Principally, for each cluster the build options can be chosen at most \textit{mazoptnumber} times.

- add VAX
  
  Entering the "(a)dd (V)AX"-command returns the question.

    \begin{verbatim}
    BUILD> a v
    The added VAX is VAX 5. Which existing VAX best approximates
    VAX 5 in speed and user workload (VAXes 1,.., 5) ?
    BUILD>
    \end{verbatim}

  Consequently, the new VAX and its user workload is a copy of the specified existing VAX.

- quicken VAX

  Entering the "(q)uicken (V)AX"-command returns the following questions.

    \begin{verbatim}
    BUILD> q v
    Which VAX is to be quickened (VAXes 1,.., 5) ?
    BUILD>
    How many times faster is the new VAX cf. the VAX-11/750 (max. 10.0 times) ?
    BUILD>
    \end{verbatim}

  The maximum increment is determined by \textit{mazintacnumber}, a program parameter which equals the upper bound of the number of active interactive processes on the largest VAX. The questions are quite obvious.

  If one and the same VAX is quickened more than once, VAMP only accepts the last change in processor speed. It is possible to slow down a VAX by entering a speed less than the current processor speed of the VAX compared to the VAX-11/750.

- extend VAX
Entering the "(e)xten(d) (V)AX"-command returns the following questions.

BUILD>e
Which VAX will have its Main Memory changed (VAXes 1,...,n) ?
BUILD>n
How many times more interactive processes can use this memory compared to the existing VAX (? max. 2.40 times) ?
BUILD>

The maximum increment is determined by mazintacnumber. Logically, the extended VAX-3 cannot have more active processes than the number of mazintacnumber.

VAMP considers a proportional relation between the main memory size and the number of active processes. Therefore, the extension of main memory can be specified in terms of the increment in the number of active processes and, thus, the number of interactive processes.

Upgrading a VAX is often a combination of quickening and extending a VAX. The user should not extend the same VAX more than once! VAMP interchanges the extension factors. It is allowed to enter an extension factor less than one.

- add disk

Entering the "(a)dd (d)isk"-command returns the following questions.

BUILD>a d
Which fraction of the diskload of the existing n disks is to be transferred to the new disk (disk 10) ?
BUILD:0.0 0.0 0.0 0.25 0.20 0.0
Enter the speed of the new disk 10 (in sec) 
access time RA-81 : 0.058 sec
access time RA-60 : 0.052 sec
BUILD>

In this example, the first entry needs 9 reals in the range $[0,1]$. The value of 0.25 means that 25% of the diskload of disk-7 is to be transferred to the new disk, disk-10. In this way, for each existing disk the diskload transfer has to be specified. Entering the access time is obvious.

If the cluster already contains diskload transfers or added disks (options add disk and change diskload) specifying the fraction might result in the following error message.

error: too much diskload removed from disk 7

This means that the diskload transfer from the mentioned disk exceeds 100%. Note that if a disk transfers diskload to two or more disks in successive entries of the options for adding a disk and changing the diskload, the fractions of the transfers are casted up!

- change disk
Entering the "(c)hange (d)isk"-command returns the following questions.

```
BUILD> c d
Which disk is to be changed (disks 1,...,10) ?
BUILD> 1
Enter the new speed of disk 1 (in sec)
  access time RA-81 : 0.038 sec
  access time RA-60 : 0.052 sec
BUILD>
```

With this option disks of different speeds can be exchanged. At successive changes of the access time of one and the same disk, only the last change is accepted.

- change diskload

Entering the "(c)hange (l)oad of disk"-command returns information which depends on whether the build options to add a disk and to change the diskload have been chosen earlier for this cluster or not. If the cluster under consideration has gone through at least one of these build options, a survey of diskload that has already been transferred appears, see below.

```
The diskload transfers are as follows
(horizontal the disks to be unloaded)
<table>
<thead>
<tr>
<th>disk 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
<td>-</td>
</tr>
<tr>
<td>disk 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.10</td>
<td>-</td>
</tr>
<tr>
<td>disk 8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disk 10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Whether this survey appears or not, VAMP poses the following questions.

```
From which existing disk (disk 1,...,9) is to be removed diskload ?
BUILD> 2
Which (existing or added) disk is receiving diskload from disk 2 ?
BUILD> 1
Which fraction ? (between 0 en 1)
BUILD>
```

It might occur that entering a legal part of diskload transfer, result in an error message.

```
error 1 too much diskload removed from disk 7
```

This is caused by the fact that in an earlier option (add disk or change diskload) the particular disk has already lost a fraction of its diskload. The specified fraction at this option results in a total diskload removal of more than 100%.
Entering the fraction of zero, means that VAMP considers the invocation of this particular option to change the diskload as never happened.

- display the cluster
Entering the "(d)isplay cluster"-command returns the extended cluster display as described earlier. This option is a way of verifying the constructed cluster.

- store old/new cluster and leave BUILD
Entering the "(s)top/store"-command leaves the user with some options to store the cluster under consideration, in our example cluster 1. If cluster 1 already existed, cluster 1 contains an old and a new version. Consequently, there are four ways of storing cluster 1. If cluster 1 has been built out of the configuration, the old version of cluster 1 equals the existing configuration, which doesn’t have to be stored. Consequently, the user has to choose whether to store cluster 1 or not. Respectively, the following questions has to be answered.

```
BUILD>s
Do you want to store (B)OTH versions, the old and new one, of CLUSTER 1 only
the (N)ew version, only the (D)ld version or neither of them (G)
BUILD>
BUILD>s
Do you want to (S)tor. this cluster or (N)ot
BUILD>
```

Anyway, the function for building clusters is closed with the most recent cluster survey (if there are any clusters), leaving the user at the VAMP level.

### 4.5.2 Remove Clusters

This function can be invoked only from the VAMP level by entering the "(r)emove cluster"-command. Prompt REM/CLUS>.
The same procedure of removing clusters is implemented in the function for building clusters, with this difference that here the user has explicitly chosen to remove clusters. It is not absolutely necessary to remove clusters, since there is no danger that the max-clusnumber parameter will be exceeded.

Entering this function leads to the cluster survey with the question to enter a number of a cluster the user wants to remove. Typing a number in the right range is followed by the extended cluster display and the question if the user really wants to remove the displayed cluster. The cluster is removed by twicely entering the "(y)es"-command. The cluster is not removed by entering successively the commands "(y)es" and "(n)o" or just the command "(n)o". Anyway, the user comes to the question if other clusters have to be removed. Entering the "(y)es"-command means that the described procedure starts all over again. Only now, the cluster survey may contain one cluster less. Finally, entering the "(n)o"-command at this entry supplies the most recent cluster survey and a return to the VAMP level.
4.5.3 Performance Calculation of an Example of a built Cluster

Invoking this function can only be done at the VAMP level by entering the "(e)xample (cl)uster performance"-command. Prompt PERF/CLUS>. Clearly, when there aren't any clusters stored, a performance prediction cannot be made. The following error message has been developed to indicate this occurrence.

```
error: no clusters stored
```

Consequently, the user leaves this function and returns to the VAMP level.

First, the user has to enter a name of a file containing model parameters. The same question appears in the function to calculate the performance of the existing configuration. If there is exactly one cluster stored, the user is directly confronted with the display of Figure 4.4.

However, when there are two or more clusters stored, first the user has to specify the number of the cluster with which a performance prediction will be made. If the user is not sure about the number, he can invoke the display of the cluster in order to verify the corresponding cluster. After the display the user is able to change the entered number. After entering a correct number of a cluster, the display of Figure 4.4 appears.

From this point, this option runs parallel to the function of calculating the performance of the existing configuration. Concerning the specification of the filename, the same error messages can occur.

VAMP starts adapting the model parameters to the changed model which defines the cluster. This can be observed at the display of Figure 4.4, where an extra column appears for each VAX which has been added in the cluster.

Again, per VAX the number of active and batch processes have to be specified. The same error messages may appear. The ultimate display of the performance characteristics includes the results of the added disks and VAXes.

4.6 Suggestions for Extended Command Language

It has already been mentioned that the read procedures have been developed with the intention of combining successive entries. We haven't had the time to implement such a command language, only to think about the design. In this Section we will give some suggestions for the design of the command language. We will give the original command followed by the shortest abbreviation.

An example of the command for invoking the function to determine the model parameters could be

```
model parameter determination/excluded/1 7 88/2 8 88/MODPAR.DAT
m/e/1 7 88/2 8 88/MODPAR.DAT
```
As a consequence, the model parameters over the period July/1/88-August/2/88 without the measurements in the weekends are calculated and placed in file MODPAR.DAT. The normal procedure of answering the posed questions as described in Section 4.3 can be avoided.

The function to calculate the performance of a cluster can also be done in fewer commands. An example could be

```
example cluster performance/MODPAR.DAT/4
```

```
e cl/MODPAR.DAT/4
```

Now, a performance prediction will be made with cluster 4 based on model parameters of file MODPAR.DAT. The user is left at the specification of the number of active processes per VAX, together with the display of the average number of measured active processes.

Within the function for building clusters, some interesting combinations could save time.

```
add VAX 2/change disk 9 0.038/display
```

```
av 2/c d/9/0.038/d
```

This command results in a cluster with an added VAX which is a copy of VAX-2 and a changed disk-9, which has now access time 0.038 seconds. Further, the cluster is displayed.

Several other combinations of the options from the menu at BUILD level could be designed.

At the moment, we have implemented the possibility to omit the display of Figure 4.4 in the functions to calculate the performance. The commands are

```
example configuration performance *
```

```
e co *
```

and

```
example cluster performance *
```

```
e cl *
```
Chapter 5

Case Studies

5.1 Predicting the Performance of an upgraded VAX Processor

5.1.1 Changing the System Configuration

In October 1987 the system configuration of the VAX/VMS-cluster at the EUT changed into the configuration as described in Chapter 2. A VAX-11/750 computer with 6 Mb main memory had been replaced by the VAX 8530 machine with 16 Mb, which is estimated to have a processor speed 5 times as fast as the VAX-11/750. Further, the number of disks increased from 7 to 9, in order to support the extended computer power. Two RA-60 disks had been replaced by RA-81 disks and two RA-60 disks had been added. Besides these long term adaptations, a short term adaptation was made. As a result of monitoring the old configuration (April’87 to end-September’87) we noticed that the system disk was quite heavily used. About 50% of all the disk visits were to this disk. After notifying the system manager, it was decided to transfer the paging and swapping files to another disk, reducing the IO requests to the system disk by about 40%.

This change meant a unique opportunity to verify the performance prediction capabilities of VAMP. By comparing the performance of the new configuration with the predicted performance of a cluster which approximates this configuration as closely as possible, the accuracy of the performance prediction could be determined. Therefore we built the following cluster out of the old configuration:

1. VAX-3 has been sped up by a factor 5. The main memory of this VAX has been made 2.67 times larger than the original, resulting in an increment of the maximum number of 12 active processes, imposed by the number of ports on VAX-3, to 32. The number of 32 active processes is imposed by the model parameters used for the cluster.

2. The access times of disk-6 and disk-7 have been decreased from 52msec to 38msec.
3. Two disks, namely disk-8 and disk-9 have been added (access times 52 msec), with a disk workload respectively 5% and 1% of the disk workload of disk-4. Observing the disk utilizations of disk-8 and disk-9 in the first weeks of monitoring the new configuration resulted in these percentages.

4. Further we have transferred 40% of the disk workload of disk-4 to disk-6. This 40% is to account for the transfer of the page and swap files and is based upon an examination of disk workloads within the first weeks of monitoring the new configuration.

In order to avoid confusion, we distinguish between the name of a station in our model and the corresponding system name of the hardware device. In this Section we will only use the model names. Therefore the system names at the time of the system change with the corresponding model names are shown in Table 5.1

<table>
<thead>
<tr>
<th>model name</th>
<th>system name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>TUERC1</td>
</tr>
<tr>
<td>VAX-2</td>
<td>TUERC2</td>
</tr>
<tr>
<td>VAX-3</td>
<td>TUEWS1/TUERC5</td>
</tr>
<tr>
<td>disk-1</td>
<td>USER1</td>
</tr>
<tr>
<td>disk-2</td>
<td>USER2</td>
</tr>
<tr>
<td>disk-3</td>
<td>USER3</td>
</tr>
<tr>
<td>disk-4</td>
<td>COMMONSYS</td>
</tr>
<tr>
<td>disk-5</td>
<td>USER4</td>
</tr>
<tr>
<td>disk-6</td>
<td>USER5</td>
</tr>
<tr>
<td>disk-7</td>
<td>POOL</td>
</tr>
<tr>
<td>disk-8</td>
<td>USERBD</td>
</tr>
<tr>
<td>disk-9</td>
<td>APPL</td>
</tr>
</tbody>
</table>

Table 5.1: Model Names versus System Names of the EUT-cluster

We have described our efforts to predict the performance of the new configuration with the described cluster in [13]. In the following we give the performance predictions with an initial cluster without point 4, combined with the performance results of the cluster used in [13]. The initial version of VAMP could not build clusters with diskload transfers from one existing disk to another existing disk. It was only possible to transfer diskload to a disk which had been added in the cluster. Since we started this case study with this version of VAMP, we also give the results of the prediction with this initial cluster. Moreover this gives an idea of the iterative design and development of the VAMP package.

1After the change in configuration the system name of VAX-3 changed from TUESW1 into TUERC5.
It has to be remarked that the performance statistics for the cluster in this Section differ from those in [13], since here we use the seek optimization and the concept of averaging the model parameters as described in Chapter 3.

In order to understand the decision of the system manager of the EUT-cluster to alter the system configuration by means of upgrading VAX-3 and extending and reorganizing backing storage, we first give the performance statistics of the month preceding to the configuration change, September '87, as shown in Table 5.2.

In all performance calculations in this Section, we have to duplicate the effects of batch processes upon interactive in order to obtain fair comparisons. Since in any reasonable busy system, batch jobs hardly receive attention at the CPU owing to their lower priority, we have excluded batch processes in all our calculations. As a consequence, we did not use our measurements in the weekends, since then the number of batch and active processes is not representative for normal interaction between these processes. The model parameters used for the performance calculations of September were based on 17 working days of measurement. In Table 5.2 the first calculation was made with the average number of (7,5,3) active processes during the measurements. The second performance calculation in Table 5.2 was made with a high number of active processes of (11,10,8), in order to give an idea of the bad response times the users had to deal with at that time.

<table>
<thead>
<tr>
<th>VAX-1</th>
<th>7</th>
<th>0.662</th>
<th>2.720</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-2</td>
<td>5</td>
<td>0.709</td>
<td>2.386</td>
</tr>
<tr>
<td>VAX-3</td>
<td>3</td>
<td>0.520</td>
<td>2.021</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VAX-1</th>
<th>11</th>
<th>0.836</th>
<th>5.052</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-2</td>
<td>10</td>
<td>0.845</td>
<td>5.205</td>
</tr>
<tr>
<td>VAX-3</td>
<td>8</td>
<td>0.795</td>
<td>4.374</td>
</tr>
</tbody>
</table>

Table 5.2: Old Configuration - Measured Performance in September 1987

Many complaints of the users about huge response times, especially at times the VAXes were heavily used, led to the upgrading of VAX-3 and extension of background memory. Together with the upgrading, it was decided to transfer most users of VAXes 1 and 2 to the new and faster VAX-3, in order to decrease the system workload of the first two
VAXes. The extremely high disk utilization of the system disk, disk-4, led to the reorganization of backing storage, by transferring the page and swap files from disk-4 to disk-6. Clearly, disk-4 was the bottleneck, since even with many active processes the CPUs were used for about 80%.

5.1.2 Comparison Measured and Predicted Performance

In November 1987 we had collected a number of measurements for the new configuration, with which a representative performance calculation could be made. In [13] we had chosen a small period of 10 working days, October/15/87-October/30/87, near the beginning of the initial measurements. We hoped that we would have a period in which the user workload would stay relatively constant. However, it seemed that the user workload was actually changing, since four extra working days of measuring the new configuration gave a performance which differed considerably. However we had to choose a period in the beginning.

In the mentioned period an average population of (3,3,13) active processes was measured. By using this population in the performance prediction, we were able to make the best prediction possible. We were in the position that we knew exactly how the user workload had responded to the altered configuration. The advantage of looking back at performance does not appear in practise. Predicting the change in user behaviour is very difficult and has to be based on the experience and knowledge of the system manager. In our case, the system manager decided that most users on VAX-1 and VAX-2 would be transferred to the new faster VAX, with the first two VAXes being used almost exclusively for respectively special purposes and computer courses. However, could he predict that within the month the average population would change from (7,5,3) into (3,3,13)? The measured performance for the new configuration is shown in Table 5.3. The performance statistics slightly differ from those in [13], due to the recalculation of the the performance with the improved algorithm.

<table>
<thead>
<tr>
<th>number of active processes</th>
<th>CPU utilization</th>
<th>response time per CPU second</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>3</td>
<td>0.466</td>
</tr>
<tr>
<td>VAX-2</td>
<td>3</td>
<td>0.281</td>
</tr>
<tr>
<td>VAX-3</td>
<td>13</td>
<td>0.450</td>
</tr>
</tbody>
</table>

Table 5.3: New Configuration - Measured Performance in October 1987

Compared to the performance of the pre-October configuration with the average popu-
lation in Table 5.2, we see a definite improvement. If the response times of VAX-3 are compared, the response time of the new faster VAX has to be divided by 5, since the response time in Table 5.3 is compared to a VAX 8530 second. Therefore, the users of the VAX-3 saw their response time reduced by about 70%! Further we note that apparently the disk IO intensity had been reduced, since the disk utilizations are predominantly better.

Our initial efforts to predict the performance in Table 5.3 were made with a cluster without the transfers of the page and swap file to disk-6. This prediction was based on model parameters deduced from measurements taken in September'87 (during 17 working days). We assume that the user workload in this month best approximates the workload in the last two weeks of October. Remember that in a performance prediction the model parameters have to be adjusted, since building a cluster means changing the model. The initial performance prediction is placed in Table 5.4.

<table>
<thead>
<tr>
<th>number of active processes</th>
<th>CPU utilization</th>
<th>response time per CPU second</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>3</td>
<td>0.341</td>
</tr>
<tr>
<td>VAX-2</td>
<td>3</td>
<td>0.539</td>
</tr>
<tr>
<td>VAX-3</td>
<td>13</td>
<td>0.524</td>
</tr>
</tbody>
</table>

Table 5.4: Performance Prediction with Initial Cluster

Compared to the performance in Table 5.3 we see that the predicted disk utilizations are predominantly too high and that the predicted response time for one CPU second for VAX-3 is completely wrong.

Before we could draw conclusions we wanted to know the effect of the transfer of the paging and swapping files on our predictions. Therefore, we adjusted VAMP in order to create clusters with diskload distributed over existing disks.

The performance prediction with the cluster which was used in [13] can be seen in Table 5.5.

This prediction compared to the initial prediction in Table 5.4 shows better response times, which can only be caused by the reduction of the disk utilization of disk-4 by 39%. Apparently this disk was a bottleneck in the initial cluster. The slightly increased CPU utilizations are due to an improved throughput through the network. The utilization reduction of disk-4 has a greater effect upon throughput than the increased utilization of disk-6.

In spite of the improved performance, the second prediction is still not good. The CPU
Table 5.5: Performance Prediction with Cluster

utilizations for VAXes 1 and 2 together are predicted well, but the split over the two is not good. The response times per CPU second are quite good. For VAX-3, the predicted CPU utilization is good, while the predicted response time still remains wrong.

In [13] the performance prediction with the cluster was based on measurements taken in the period March/28/87 - October/2/87. A similar result appeared.

5.1.3 Explanations for the bad predicted Response Time for VAX-3

In search for an explanation for the much too high predicted response time for VAX-3 in Table 5.5, we concentrated on the causes of the differences in disk utilizations. Another way of presenting these differences are the model parameters, in particular the ratio between CPU workload and disk workload per active process of VAX-3, sorted at the number of active processes. In the measured performance of Table 5.3 this number of active processes equals 13 with corresponding ratio 1 : 1.063.

In the predicted performance, the model parameters had been collected from the old configuration. However, VAX-3 had changed in a number of ways. In the VAMP packet it is assumed that the number of active processes grows proportional with main memory extension. Therefore, the model parameters of VAX-3 for 13 active processes in the cluster corresponds to 5 active processes in the old configuration (13 = 5 \times 2.67). The ratio becomes 1 : 0.486. However, besides memory extension VAX-3 has become 5 times as fast. This means that the new VAX-3 with the proportional increased number of active processes, creates in 1 CPU second the same number of disk IOs as the old VAX-3 in 5 CPU seconds. Therefore the ratio used in the performance calculation becomes 1 : 2.427. Using the measured ratio 1 : 1.063 in the prediction with the cluster used in [13] resulted in a predicted response time for one second CPU of 3.013. Concerning this, it is evident that the high error in the relative disk workload contributes significantly to the total error.
In Section 2.3.3 we showed that the ratio between the number of disk IO generated by active and batch processes is crucial in the relative disk workload determination. If the disk IO rates of batch processes before and after the configuration change are approximately the same, we can state that a difference in the relative disk workload is due to an altered disk IO behaviour of active processes. We have no reason to believe that indeed the behaviour of batch processes had been changed. Therefore in the following we only concentrate on differences in the disk IO rates of active processes of VAX-3.

It is obvious that the change in user workload could not be the only cause of the changed disk IO rate. This user workload upon the system had not really changed drastically. Therefore we looked at the VMS memory management system, since this system greatly affects the total number of disk IOs.

As a result of inspecting the VMS memory management system, we found a substantial difference in the size of a certain system parameter between the old and new VAX-3. This system parameter is called PFRATH. Before we are able to speak about the consequences of the difference in the PFRATH parameter, we have to understand the mechanism for allocating main memory to processes.

Let us concentrate on the way of adjusting the WS limits, the maximum number of pages a process is allowed to have in physical memory. At any time, the number of pages in the WS, the sum of the global pages and a process' private pages, cannot exceed the WS limit. The size of a process' WS limit can be altered with the Adjust Working Set Limit system service. Part of this system service is the Automatic Working Set Adjustment (ASWA), provided by VMS operating system to keep a process' page fault rate within certain limits. These limits can be set by several system parameters controlled by the system manager.

For each time-sharing process ASWA takes place each time such a process has completed a time slice, but after ASWTIME msec of CPU calculation time (in practice, ASWTIME msec is reached after a number of time slices). Consequently, only currently executing processes will have their WS adjusted. Clearly, it is not useful to increase the WS of a process waiting for certain events.

At the time of the actual adjustment, the current number of page faults per CPU second of the process is calculated. The concept of ASWA serves two purposes. If the page fault rate is too low, the system can decrease the WS limit without harming the process. Secondly, if the page fault rate is too high, the process can benefit from a larger WS size, without degrading the system's performance. Then, less disk IOs is weighted against a greater WS size.

- If the current page fault rate is too high (greater than or equal to PFRATH),

---

2 The parameters we will discuss are AWSMIN, AWSTIME, BORROWLIM, GROWLIM, PFRATH, PFRATL, WSDEC and WSINC (dynamic SYSBOOT parameters) and WSEXTENT and WSQUOTA (parameters on command level).
a determination is made to see if the WS limit should be increased. If the WS size is greater than 75 percent of its WS limit and the size of the WS limit is below WSQUOTA, the WS limit is extended by WSINC pages. If the WS limit is greater than or equal to WSQUOTA, the number of pages on the free page list is compared to the system parameter BORROWLIM. If there are more than BORROWLIM pages on the free page list, the WS limit is increased by WSINC pages. However, if there are fewer pages on the free page list, the WS limit is not extended. The WS limit can only be extended up to WSEXTENT.

Note that ASWA only affects the size of the WS limit. The WS size can be increased by adding pages. The pager adds pages in a WS with a limit above WSQUOTA only when there are more than GROWLIM pages on the free page list size. GROWLIM is smaller than BORROWLIM.

The interaction between WS adjustment and the number of pages on the free page list serves the purpose of equal distribution of pages over the processes when physical memory is scarce.

- If the current page fault rate is too low (strictly less than PFRATL), the WS limit is decreased by WSDEC pages. However, if the number of private pages is less than or equal to AWSMIN the WS limit is not lowered. If there are few private pages in a process' WS, there are many global pages in relation to private pages. Since the system will not benefit from removing global pages, this has to be avoided.

Note that if PFRATL is set zero, ASWA only extends WS limits. Then, the swapper takes care of the WS reduction, by selecting those processes which are least likely to need large WSs. Therefore a compute-intensive process with many time slices and a low page fault rate will have its WS reduced with a small probability. On the contrary, if the PFRATL is greater than zero, such a process is highly qualified for WS reduction.

For more detailed information about VMS memory system we refer to [11,12].

In the pre-October configuration, for each VAX PFRATL was equal to zero and PFRATH to 120 page faults per 10 CPU seconds. This parameter institution didn't change, meaning that the PFRATH for the new faster VAX was also set on 120. This means an implicit lowering of the PFRATH parameter for the new VAX-3, since the new faster VAX-3 generates per process per CPU second about 5 times the page faults of the old VAX-3 in one CPU second per process, at least when the WS sizes do not differ. However, the WS sizes can not differ very much, since the sizes of WSQUOTA and WSEXTENT didn't change either. They remained on respectively 500 and 1000 pages.

We want to know the effect of this implicit lowering of the PFRATH on the number of disk I/O generated by processes of this faster VAX. It is obvious that the WS limits of these processes are increased more quickly and once their number of page faults stabilizes these limits might be on a higher level (especially small processes which need few pages in main memory). Since in the first weeks of using the new configuration, main memory wasn't the bottleneck this might be an explanation for the reduction of the number of disk I/Os and the bad predicted response time in Table 5.5. (There are about 26000 free
pages to be divided over WSs and free page list, while 13 active processes could use at most 13000 pages).

In order to trace the effect of the PFRATH system parameter on the number of disk IOs, the system manager of the EUT-cluster agreed to reset the PFRATH of VAX-2 from 120 to 90 page faults per 10 CPU seconds.

In the next Section we will discuss the consequences of different PFRATH institutions of VAX-2 on the relative disk workload and the response time.

5.1.4 The Effects of PFRATH on Performance

VAX-2 of the VAX/VMS-cluster at the EUT is almost exclusively used for computer courses. This means a great variety in the user workload, since the computer courses change from day to day, due to different and developing courses. Logically, this affects the reliability of the model parameters deducted from VAX-2. Consequently, the performance results are less reliable.

The PFRATH of VAX-2 was reset on March 16, 1988. The measurement period for the PFRATH of 120 (March/1/88 - March/15/88) contains 10 working days. The measurement period (March/17/88 - April/9/88) with the lowered PFRATH contains 12 working days. We used these two contiguous periods in order to eliminate as much as possible the effects of changing user workload on VAX-2.

Let us first look at the model parameters determined from measurements in these two periods, in particular the ratio of the workload per active process for CPU and disk. In Table 5.6 we have placed these ratios, selected for 1 up to 7 active processes. More active processes has not been measured in these periods.

<table>
<thead>
<tr>
<th>number of active processes</th>
<th>PFRATH 120 Mar/1/88-Mar/15/88</th>
<th>PFRATH 90 Mar/17/88-Apr/9/88</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 : 0.532</td>
<td>1 : 0.407</td>
</tr>
<tr>
<td>2</td>
<td>1 : 0.555</td>
<td>1 : 0.462</td>
</tr>
<tr>
<td>3</td>
<td>1 : 0.624</td>
<td>1 : 0.340</td>
</tr>
<tr>
<td>4</td>
<td>1 : 0.688</td>
<td>1 : 0.371</td>
</tr>
<tr>
<td>5</td>
<td>1 : 0.774</td>
<td>1 : 0.526</td>
</tr>
<tr>
<td>6</td>
<td>1 : 0.715</td>
<td>1 : 0.937</td>
</tr>
<tr>
<td>7</td>
<td>1 : 0.816</td>
<td>1 : 0.949</td>
</tr>
</tbody>
</table>

Table 5.6: Ratio CPU and Disk Workload

Clearly, there is a substantial difference between the ratios of the periods. The relation between the ratio and the number of active processes in the first period tends to be linear, while the second period seems to supply a non-linear relation. In Figure 5.1 this difference is displayed graphically. It has to be remarked that the parameters for 1 up
to 5 active processes are quite accurate, since most measurements were made with these numbers of active processes.

We see that lowering the PFRATH only reduces the number of disk IOs when there are few active processes logged into VAX-2. This has to be caused by the increased WS sizes. However, when the PFRATH is low, main memory becomes scarce at a lower number of active processes, resulting in a sooner applied swapper trimming. In order to obtain a reasonable free page list size, the swapper writes modified pages to disk or reduces the WS sizes of the currently existing active processes, thus causing respectively a direct and indirect increase in disk IO intensity. This probably happens in the second period when there are 6 or 7 active processes in existence. In the first period swapper trimming is not applied or is applied more efficiently. Since in the first period with the high PFRATH, the WS of an active process contains heavily selected pages, nearly all remaining pages after swapper trimming are still useful. However, when the PFRATH is lower, the WS is filled with a number of useless pages. With greater probability, the swapper removes the wrong pages leaving the useless pages in the WS. Consequently, an increase in the page fault rate follows, thus more page read IOs.

We also see that the ratio in the second period with a small number of active processes is actually higher than with 3 or 4 active processes. This might be due to the minor effect of the concept of global pages at a low number of active processes, resulting in more global page faults. In the first period this behaviour cannot be observed. Apparently the number of page read IOs for global pages is relatively small compared to the read IOs for private pages, due to the small WS sizes in the first period.

In spite of the fact that the chosen periods are contiguous, it is likely that indeed some change in user workload has occurred, due to changing computer courses. So, a changed system workload could also explain some of the differences described above.
We state that the enormous reduction of the number of disk IOs at a low number of active processes could not only be caused by a changed user workload on VAX-2, in spite of the fact that this VAX is used for various computer courses. Therefore, such a drastic change in the number of disk IOs as observed, mainly has to be due to the reduction of 25% of the PFRATH parameter.

We have also made performance calculations with the model parameters from the two mentioned periods. We are only interested in the differences in the performance of the second VAX at varying number of active processes. Therefore we have kept at each calculation the number of active processes of the VAXes 1 and 3 constant. In Table 5.7 we have placed the performance of VAX-2 in terms of CPU utilization and response time for one second CPU calculation time at different numbers of active processes.

<table>
<thead>
<tr>
<th>number of active processes</th>
<th>CPU utilization</th>
<th>response time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PFRATH 120</td>
<td>PFRATH 90</td>
</tr>
<tr>
<td>1</td>
<td>0.091</td>
<td>0.068</td>
</tr>
<tr>
<td>2</td>
<td>0.169</td>
<td>0.144</td>
</tr>
<tr>
<td>3</td>
<td>0.248</td>
<td>0.264</td>
</tr>
<tr>
<td>4</td>
<td>0.316</td>
<td>0.420</td>
</tr>
<tr>
<td>5</td>
<td>0.404</td>
<td>0.524</td>
</tr>
<tr>
<td>6</td>
<td>0.447</td>
<td>0.550</td>
</tr>
<tr>
<td>7</td>
<td>0.531</td>
<td>0.561</td>
</tr>
</tbody>
</table>

Table 5.7: Performance Comparison

We see that for 4 or less active processes, lowering the PFRATH leads to better response times. Moreover, the CPU is used more efficiently at 3 and 4 active processes, since the CPU utilizations are higher while the response time is lower. Apparently the increased WS sizes causes a decrease in the number of quantum interrupts at the CPU due to page faults. When there are 5 or more active processes connected with the VAX-2, the relation between CPU utilization and response time in the two periods slightly differs. However, the utilization and response time are definite higher in the period with the lowered PFRATH.

We now return to the model parameters before and after the change in configuration in October 1987. We want to see if the found difference in the ratio of the workload per active process for CPU and disk between measurements with a high and a low PFRATH can also be seen at the pre-October measurements (high PFRATH) and the end-October measurements (low PFRATH) of VAX-3. The result is shown in Figure 5.2.

In comparison with Figure 5.1 we see a much greater vertical distance. This is probably caused by the greater difference in PFRATH of VAX-3 before and after the change than
the 25% of VAX-2. (The difference is about 83%). Further, some different change in
the user workload at the time of the change in system configuration than the workload
change of VAX-2, might cause a greater vertical distance.
It has to be remarked that at 13 active processes the vertical distance is about maximum.
Considering the low number of 13 active processes and the relatively small size of
WSQUOTA and WSEXTENT, the implicit PFRATH reduction might be a plausible
explanation for the bad predicted response time for VAX-3 in Table 5.4 and Table 5.5.

5.1.5 Concluding Remarks

We have described a part of the iterative development of the VAMP packet by using
VAMP in a case study. An adaptation had to be made in order to create a wider range
of clusters with which performance predictions could be made.
With the VAMP packet we discovered the effect of the PFRATH system parameter on the
performance. It seems that when a VAX is used by few active processes the performance
can significantly be bettered by lowering the PFRATH. However, when a VAX is quite
heavily used with many active processes lowering the PFRATH should be avoided. It
might be better to increase the PFRATH.
Besides the need to model the PFRATH parameter, which require a full understanding
how the WS sizes affect the number of disk I/Os, we have to be able to predict changes in
user workload right before and after a change in system configuration. It will require a lot
of effort to solve these problems sufficiently and to make reliable performance predictions.
5.2 Examples

5.2.1 Introduction
In this Section we will compare the performance of the existing configuration with the performance of clusters. The clusters will be built with the intention of improving the current performance, by attacking the bottleneck resources. In this way, the comparison will confirm the intuitive improvements of certain configuration changes. We will ensure to look at each of the configuration changes provided by the build options of the VAMP package.

In Section 5.2.2 we will concentrate on the effects of computer power extension on the performance of backing storage. The clusters will be built out of the VAX/VMS-cluster at the EUT.

Some examples of the effects of reorganizing backing storage upon performance will be discussed in Section 5.2.3. The performance comparisons will be made with the performance of the VAX/VMS-cluster at the WUA.

The performance results in the following Sections will be expressed in terms of the utilization of both the VAXes and the disks. Further, for each VAX the response time compared to the CPU second of the VAX-11/750 machine per active as well as batch process.

5.2.2 The EUT-cluster
In August 1988 the EUT-cluster consisted of three VAXes, two of type VAX-11/750 and one of type VAX 8530. The backing storage included nine disk units, seven RA-81 disks and two RA-61 disks to be precise. In this Section we will only use the model names of the EUT-cluster. The corresponding names in the existing configuration are shown in Table 5.1.

All performance calculations in this Section are based on model parameters deducted from measurements on working days in June 1988. This period contains respectively 19, 21 and 22 days of measuring VAXes 1, 2 and 3. The average population of active processes was (2,3,13). The average population of batch processes at these numbers of active processes was (2.5,1.0,5.5). The value of 1.0 for the average number of batch processes on VAX-2 is the smallest possible number, since during the measurements one batch job is always in existence, the batch job which measures the VAX. This doesn't have to mean that this batch job was the only batch job running on VAX-2 during the working days of June. We can only state that if there were other batch jobs, they existed shortly.

The measured performance with the average population of (2,3,12) active processes and (1,0,3) batch processes is shown in Table 5.8.
Clearly, the batch processes consume a lot of CPU calculation time, in conformity with the basic idea behind releasing batch jobs. That is, filling up the idle times between services given to active processes. The difference in utilization of VAXes 1 and 2 in relation to the number of active processes confirms this. VAX-3 hasn't hardly any idle time, mainly due to the number of three batch processes. We see that disk-4 and disk-7 are most heavily used.

We will now look at the performance at the population of (4,9,25) active and (1,0,3) batch processes, in order to see if the disks could handle the disk IO demand at busy moments at the working days of June 1988. This performance will be used in the comparison with the yet to define clusters. The results are shown in Table 5.9.

We see that the utilization of VAX-2 has increased and that the utilization of VAXes 1 and 3 haven’t really changed. Apparently, the extra active processes slowed down the activity of batch processes. The response times for the batch processes of VAXes 1 and 3 indicates this lowered activity, due to the lower priority of the batch processes at the CPU, resulting in larger wait times.
Obviously, the distribution of the diskload over the disks at busy moments differs from the distribution at the average population. For example, disks 8 and 9 were totally unused in the busy situations. Probably, in practise these disks were used, only the usage was too small for our model to calculate utilizations greater than zero. This is due to both the effects of more processes competing for the system resources and a different user behaviour. More processes and an altered user behaviour could cause much more disk IOs to certain disks (e.g. the system disk, disk-4). As a consequence, the fraction of disk IOs to other disks becomes smaller, resulting in a smaller disk utilization, as calculated in our model.

Suppose that the system manager of the EUT-cluster considers to extend the computer power. More precisely, he wants a fourth VAX of type VAX 8700 with a size of main memory of 32 Mb. This VAX is estimated to have a processor speed 8.6 times as large as the VAX-11/750. Further, suppose that the system manager expects that the user workload of the new VAX would approximately equal the workload of the existing VAX-3. In order to predict the effects of the extended computer power at busy moments upon performance, particularly the performance of background memory, the system manager builds the following cluster, cluster-EUT1.

- add a VAX-4, which is a copy of the existing VAX-3
- quicken VAX-4 to 8.6 times the speed of the VAX-11/750
- extend VAX-4 2 times (VAX-3 has 16 Mb main memory)

Suppose that the system manager expects that in the near future 30 active processes and 1 batch process at VAX-4 will represent a busy situation. Therefore, he makes the following performance prediction with cluster-EUT1, as shown in Table 5.10.

<table>
<thead>
<tr>
<th>VAX</th>
<th>CPU utilization</th>
<th>process class</th>
<th>number of processes</th>
<th>response time per CPU second</th>
<th>utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>0.833</td>
<td>active</td>
<td>4</td>
<td>2.600</td>
<td>disk-1 0.307</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>4.554</td>
<td>disk-2 0.173</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.826</td>
<td>active</td>
<td>9</td>
<td>4.157</td>
<td>disk-3 0.165</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.993</td>
<td>active</td>
<td>25</td>
<td>1.231</td>
<td>disk-4 0.735</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>3</td>
<td>2.592</td>
<td>disk-5 0.771</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.969</td>
<td>active</td>
<td>30</td>
<td>0.900</td>
<td>disk-6 0.476</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.708</td>
<td>disk-7 0.534</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>disk-8 0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>disk-9 0.000</td>
</tr>
</tbody>
</table>

Table 5.10: Performance Prediction with cluster-EUT1

Compared to the performance of Table 5.9, we see that the utilization of VAXes 1 and 2 has been reduced. Apparently, the processes of these VAXes have to wait longer at the disks. The same holds for the active processes of VAX-3, only here the batch processes
have used more CPU time, resulting in a slightly increased CPU utilization. This state-
ment is confirmed by the reduced response times for these batch processes.

We see an overall increment of the disk utilization. Disks 4 and 5 have still the highest 
utilization. Concerning VAX-4, we see that the active processes belonging to this VAX 
actually have the lowest response times. It is well known that a VAX 8700 can handle 
much more than 30 active processes.

Concludingly, we state that the disks can handle an extra VAX 8700, since the response 
times per active process of the VAXes 1, 2 and 3 do not really suffer.

The performance can be improved significantly by removing diskload from disk-4 and 
disk-5, since these disks have utilizations which are much higher than all other disks in 
the performance prediction of Table 5.10. The system manager should think of this when 
he observes the performance prediction.

In order to verify the performance improvement, we have built cluster-EUT2 out of 
cluster-EDT1, by adding disk-10 with 15% and 20% of the diskload of respectively disks 
4 and 5, see Table 5.11.

<table>
<thead>
<tr>
<th>VAX</th>
<th>CPU utilization</th>
<th>process class</th>
<th>number of processes</th>
<th>response time per CPU second</th>
<th>utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>0.885</td>
<td>active</td>
<td>4</td>
<td>2.329</td>
<td>disk-1 0.293</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>4.069</td>
<td>disk-2 0.162</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.841</td>
<td>active</td>
<td>9</td>
<td>3.972</td>
<td>disk-3 0.158</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>3</td>
<td>2.763</td>
<td>disk-4 0.551</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.996</td>
<td>active</td>
<td>25</td>
<td>1.084</td>
<td>disk-5 0.521</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.813</td>
<td>disk-6 0.453</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.977</td>
<td>active</td>
<td>30</td>
<td>0.733</td>
<td>disk-7 0.509</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td></td>
<td>disk-8 0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>disk-9 0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>disk-10 0.245</td>
</tr>
</tbody>
</table>

Table 5.11: Performance Prediction with cluster-EUT2

Compared to the performance of cluster-EDT1 in Table 5.10 we establish a definite 
throughput improvement of the active processes of each VAX, since all CPU utilizations 
have been increased. Clearly, disks 4 and 5 were bottlenecks, since the response times 
per active process on each VAX has been reduced.

We see that adding a disk and reorganizing the distribution of the diskload certainly 
 improves the performance. The question whether to add a disk or not depends on 
the critical performance which the system manager considers acceptable. Therefore, 
he has to be able to "translate" the performance as displayed in the Tables to the 
behaviour of the VAX/VMS-cluster in reality. Of course, the translation should include 
the characteristic user workload on each VAX, since the response times for the VAXes 
in the Tables contain average values and might have large variances due to completely
different users (simulation versus data base queries). Anyway, the system manager cannot make a proper translation without a certain experience. Further, the future developments in the number of users and the sort of users (calculation intensive versus disk IO intensive) should also be considered in the decision making. However, in this case costs aspects should not be included, since the costs of purchasing a disk is low compared to a VAX 8700!

5.2.3 The WUA-cluster

In August 1988, the VAX/VMS-cluster at the WUA-cluster consisted of four VAXes and fifteen disks. The VAX-11/785, the VAX 8600 and the two VAXes of type VAX 8700 had a common background memory of eleven disks, ten RA-81 disks and one RA-60 disk. Further, the VAX-11/785 had four local disks, one of type RP-07 and three of type RP-06. In this Section we will only use the model names. Since we haven't introduced the corresponding names in the WUA-cluster, the link between the names in reality and the model is shown in Table 5.12. The disks 12, 13, 14 and 15 are the local disks.

<table>
<thead>
<tr>
<th>model name</th>
<th>system name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>LUWRVA</td>
</tr>
<tr>
<td>VAX-2</td>
<td>LUWRVB</td>
</tr>
<tr>
<td>VAX-3</td>
<td>LUWRVC</td>
</tr>
<tr>
<td>VAX-4</td>
<td>LUWRVD</td>
</tr>
<tr>
<td>disk-1</td>
<td>VAXVMS</td>
</tr>
<tr>
<td>disk-2</td>
<td>USER</td>
</tr>
<tr>
<td>disk-3</td>
<td>USER2</td>
</tr>
<tr>
<td>disk-4</td>
<td>USER3</td>
</tr>
<tr>
<td>disk-5</td>
<td>USER4</td>
</tr>
<tr>
<td>disk-6</td>
<td>USER5</td>
</tr>
<tr>
<td>disk-7</td>
<td>FEZ1</td>
</tr>
<tr>
<td>disk-8</td>
<td>FEZ2</td>
</tr>
<tr>
<td>disk-9</td>
<td>USER9</td>
</tr>
<tr>
<td>disk-10</td>
<td>FEZ3</td>
</tr>
<tr>
<td>disk-11</td>
<td>INSTAL</td>
</tr>
<tr>
<td>disk-12</td>
<td>SYS1</td>
</tr>
<tr>
<td>disk-13</td>
<td>USERC</td>
</tr>
<tr>
<td>disk-14</td>
<td>USERD</td>
</tr>
<tr>
<td>disk-15</td>
<td>SCRATCH</td>
</tr>
</tbody>
</table>

Table 5.12: Model Names versus System Names of the WUA-cluster

All the performance calculations with the configuration of the existing WUA-cluster and the derived clusters, will be based on measurements taken at working days in July 1988.
VAXes 1, 2, 3 and 4 were measured respectively 13, 14, 14 and 12 working days. In July 1988 many measurements didn’t succeed, due to disk quota exceedment during the joining of the samples. Problems with the deletion of the MONITOR.DAT file of each VAX at the end of each day resulted in a rapid allocation of the private memory. Probably, the problem concerns the calculation of the used private memory by the VMS operating system. In spite of the fact that the MONITOR.DAT files have been removed from the directories, VMS counts them in the calculation of the allocated private memory space. Once the calculation has been corrected, VMS does not count them anymore. Since the correction is made once per week, the huge MONITOR.DAT files resulted in a rapid disk quota exceedment, allowing us to measure the WUA-cluster only a few days per week. At the EUT-cluster, we haven’t had to deal with this problem. Apparently, the correction is applied daily.

In July 1988 the average population of active processes was (2, 10, 8, 19). The average population of batch processes at these numbers of active processes was (3.1, 1.4, 1.3, 1.9). In Table 5.13 the average performance of the WUA-cluster in July 1988 is shown.

<table>
<thead>
<tr>
<th>VAX</th>
<th>CPU utilization</th>
<th>process class</th>
<th>number of processes</th>
<th>response time per CPU second</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>0.119</td>
<td>active</td>
<td>2</td>
<td>2.920</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.386</td>
<td>active</td>
<td>10</td>
<td>0.596</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.619</td>
<td>batch</td>
<td>1</td>
<td>0.788</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.740</td>
<td>active</td>
<td>8</td>
<td>0.390</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>active</td>
<td>19</td>
<td>0.490</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.352</td>
</tr>
</tbody>
</table>

Table 5.13: Measured Performance of WUA-cluster

In Section 3.4.4 we have mentioned the difficulties with the calculation of the response times for processes of VAX-1. Therefore, we excluded batch processes at VAX-1. The response time per active process is based on a relative CPU workload per active process which is too high. Besides the actual fraction of time the active processes receive attention from the CPU, the relative CPU workload consists of the fraction of time the CPU spent on controlling the disk IO traffic between all VAXes and the local disks. At the moment, the relative CPU workload is not corrected, resulting in the too high response times.
times. The disk utilization of the local disk-15 is a measure for the CPU time spent on disk IO controlling, thus also a measure for the correction. We haven't had the time to look at such correction, but it should be implemented in the modeling. We see that in this example, the utilization of disk-15 is quite high, implying that the response time per VAX-11/750 CPU second per active process of VAX-1 is much too high. In the remainder of this Section we will include active processes at VAX-1 in the performance calculations, since these processes affect the diskload and since we are interested in the effects of backing storage upon the performance.

Before we introduce configuration changes concerning reorganizing the backing storage, we will look at the performance at busy moments in July 1988. The busy situation will be represented by a population of (5,20,20,41) active and (0,1,1,1) batch processes. The result of the performance calculation is shown in Table 5.14.

<table>
<thead>
<tr>
<th>Disk</th>
<th>Utilization</th>
<th>Number of Processes</th>
<th>Response Time per CPU Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk-1</td>
<td>0.685</td>
<td>5</td>
<td>4.369</td>
</tr>
<tr>
<td>Disk-2</td>
<td>0.235</td>
<td>20</td>
<td>0.658</td>
</tr>
<tr>
<td>Disk-3</td>
<td>0.094</td>
<td>1</td>
<td>0.967</td>
</tr>
<tr>
<td>Disk-4</td>
<td>0.148</td>
<td>20</td>
<td>0.428</td>
</tr>
<tr>
<td>Disk-5</td>
<td>0.197</td>
<td>1</td>
<td>0.193</td>
</tr>
<tr>
<td>Disk-6</td>
<td>0.255</td>
<td>20</td>
<td>0.857</td>
</tr>
<tr>
<td>Disk-7</td>
<td>0.375</td>
<td>1</td>
<td>0.752</td>
</tr>
<tr>
<td>Disk-8</td>
<td>0.104</td>
<td>20</td>
<td>0.217</td>
</tr>
<tr>
<td>Disk-9</td>
<td>0.228</td>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td>Disk-10</td>
<td>0.025</td>
<td>20</td>
<td>0.015</td>
</tr>
<tr>
<td>Disk-11</td>
<td>0.021</td>
<td>1</td>
<td>0.732</td>
</tr>
</tbody>
</table>

Table 5.14: Measured Performance of WUA-cluster at busy Situation

Logically, all CPU utilizations have been increased. Further, the response times per active process of each VAX has been increased. The response time of VAX-4 has the highest increment, due to the fact that most of the disk IOs generated by processes of this VAX are to disk-1. We state a slightly altered user workload at the busy moments, since the growth in disk utilization differs.

However, the performance at busy moments in July 1988 is acceptable.

Suppose that the system manager isn't satisfied with the current performance and that he wants to reorganize backing storage. The configuration changes he considers are changing the diskload distribution (short term adaptation) and the purchase of a new RA-81
disk (long term adaptation). Both configuration changes are aimed at removing diskload from the bottleneck disks, the disks 1 and 15. Suppose that for technical reasons only 25% of the diskload of disk-1 can be removed to exactly one other disk and that disk-15 can only transfer diskload in portions of 10% of the diskload of disk-15.

In the context of studying the effects of the short term adaptation, the system manager builds cluster-WUA1 out of the existing configuration. This cluster is created by transferring 25% of the diskload of disk-1 to disk-3 and 10% of the diskload of disk-15 to each of the disks 12,13 and 14. Since disk-11 has an access time which is 14 msec larger than the access time of disk-3, it isn’t attractive to transfer the 25% to disk-11. A similar reasoning holds for the transferring of 30% of the diskload of disk-15.

The prediction of the performance at busy moments with cluster-WUA1 can be seen in Table 5.15.

<table>
<thead>
<tr>
<th>CPU utilization</th>
<th>process class</th>
<th>number of processes</th>
<th>response time per CPU second</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>0.239</td>
<td>active</td>
<td>5</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.517</td>
<td>active</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.927</td>
<td>active</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.937</td>
<td>active</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.15: Performance Prediction with cluster-WUA1

Clearly, the performance of cluster-WUA1 is better than the performance depicted in Table 5.14. We see that all response times have been reduced. This has to be due to the improved distribution of the diskload which resulted in a throughput improvement.

Concerning the long term adaptation of adding a RA-81 disk, the system manager constructs cluster-WUA2. The added disk-16 receives 25% of the diskload of disk-1 and 30% of the diskload of disk-15. The prediction of the performance at busy moments with cluster-WUA2 can be seen in Table 5.16.

Compared to the performance in Table 5.15 we see a definite performance improvement, due to a diskload distribution over the disks, which is more uniformly.
Both clusters resulted in an intuitive performance improvement. However, the reorganizing of backing storage by adding a new disk certainly gives the greatest performance profit.

In this example, we have compared a short term adaptation with a long term adaptation. It seemed that the short term adaptation couldn’t improve the performance as much as the long term adaptation. Still, the system manager has to make a choice which adaptation he will implement in the WUA-cluster. The costs of an extra disk have to be weighted against the differences in the performance profit of the clusters and the speculations about the future system workload. Therefore he has to ”translate” the differences in Tables 5.15 and 5.16 into differences in the behaviour of the computer system in reality. In Section 5.2.2 we have indicated some of the difficulties in the translation.

Especially in the research and academic environment, the future developments are hard to predict. It depends on the competition between the various computer systems installed in the environment and on the number of released projects and the sort of projects to be released in the near future.

<table>
<thead>
<tr>
<th></th>
<th>CPU utilization</th>
<th>process class</th>
<th>number of processes</th>
<th>response time per CPU second</th>
<th>utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>0.258</td>
<td>active</td>
<td>5</td>
<td>2.175</td>
<td>0.403</td>
</tr>
<tr>
<td>VAX-2</td>
<td>0.543</td>
<td>active</td>
<td>20</td>
<td>0.558</td>
<td>0.216</td>
</tr>
<tr>
<td>VAX-3</td>
<td>0.941</td>
<td>batch</td>
<td>1</td>
<td>0.797</td>
<td>0.096</td>
</tr>
<tr>
<td>VAX-4</td>
<td>0.958</td>
<td>active</td>
<td>20</td>
<td>0.336</td>
<td>0.137</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.186</td>
<td>0.180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>41</td>
<td>0.673</td>
<td>0.235</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batch</td>
<td>1</td>
<td>0.654</td>
<td>0.393</td>
</tr>
</tbody>
</table>

Table 5.16: Performance Prediction with cluster-WUA2
Chapter 6

VAMP's Reference Manual

6.1 Functional Description

The documentation of the VAMP package is specifically aimed at system managers of VAX/VMS-clusters. As a consequence, certain aspects will not be discussed in full detail, since we assume the system manager to have a basic knowledge of the VMS operating system and programming languages.

We consider the VAMP package implemented in a VAX/VMS-cluster with a number of $n$ VAX processors.

The VAMP package includes certain functions, related to each other by data files. Each function is characterized by at least one input file and exactly one output file. The functions and their connections have been described in the flowchart of Figure 6.1.

The function for measuring the system workload collects per VAX samples of the three MONITOR utilities MONITOR PROCESSES, MONITOR DISK and MONITOR SYSTEM/ALL at regular intervals of 3 minutes throughout the day. For each VAX, these samples are converted into one daily data file. The name of this file includes the node name of the VAX under consideration and the date of creation. For example, the daily data file deducted from measurements on node TUERC5 at July 22, 1988 is named VRC5..22JUL1988.DAT. The daily data files of one and the same month can be amalgamated into two monthly data files. Considering the example, the names of these files will be VRC5.E_JUL1988.DAT and VRC5.L_JUL1988.DAT, respectively an amalgamation of the daily data files with the measurements in the weekends excluded and included. The names of the other files in the flowchart equals those names in the VAMP package. However, the names denoted by a "*" are default names and might have other names due to the choice of the user.

At each function, the file CONFFILE.DAT is input. All data files appearing in the VAMP package will be described in Appendix A.
6.2 Functions of the VAMP package

6.2.1 Introduction

In this Section we will open the boxes corresponding to the functions of the VAMP package. Before we do this, we have to distinguish the function for measuring the system workload from the other functions. The distinction is based on the way the functions are invoked. The function for measuring the system workload is invoked by submitting batch jobs, while all other functions are invoked within an executing program, the user interface program, by choosing options.

Measuring the activities of a VAX processor during a specified period of the day is done by exactly one batch job. The batch job collects samples of the utilities MONITOR PROCESSES, MONITOR DISK and MONITOR SYSTEM/ALL of this VAX during the period and joins and averages these samples in order to obtain one single file, the daily data file. The commands in the batch job which lead to a daily data file of a VAX will be described in Section 6.2.2. We will also discuss the JOINMON program which reads, joins and averages the samples collected from one VAX in terms of the routine construction. Logically, the batch job includes the command for executing the JOINMON program.

The user interface program mainly contains routines. In Section 6.2.3 we will discuss the routine construction within each remaining function.

Concerning the routine construction, we will make the following convention. If routine
A makes use of routine $B$, we will denote this by \[
\begin{array}{c}
A \\
B
\end{array}
\]

If routine $A$ makes use of successively routines $B$ and $C$, we will denote this by one of the following structures:

\[
\begin{array}{c}
A \\
B \\
C
\end{array}
\quad \quad \quad
\begin{array}{c}
A \\
B \\
C
\end{array}
\]

Further, a routine which can be used optionally, is denoted by a "*". If such a routine can also be invoked more than once, we will denote this by a "+" instead of the "**".

6.2.2 Function for Measuring the System Workload

The Batch Job

Measuring a VAX processor in a VAX/VMS-cluster is conducted by one batch job. Each batch job executes in a different directory, preferably named by the node name of the VAX and operates from the command file PERFMON.COM in this directory. The structure of the various directories required by the VAMP package will be discussed in Appendix C. Further, the execution of each command in the batch job is written in a log file in main memory, PERFMON.LOG. The batch jobs belonging to the various VAXes should not end execution at the same time. The construction of the batch jobs prohibits this. Moreover, the private memory space should be unnecessary large, since at the end of execution, each batch job manipulates the files with the MONITOR samples resulting in a temporary increment of the private memory allocation (For example, the file ZPRO.DAT of a VAX 8530 could be 1500 blocks).

The following flowchart represents the operation of the batch process. The boxes include the commands in the batch jobs, the bags the input and output files. In each batch job the files have the names as specified in the flowchart, except the daily data file. File VERZMON.DAT is the standard name of the daily data file.

In the remainder of this Section we will discuss the commands in each of the boxes. We will do this on the basis of the batch job which measures VAX LUWRVA.
1. The batch job starts with the following sequence of commands

```
$purge dlS$user5[wscopeperf]perflmon.log/since=yesterday
$submittdlS$user5[wscopeperf.rva]perflmon.Keep/noprinter/queue=LUWRVA_BATCH/
priority=200/after="tomorrow + 07:40"
$set default dlS$user5[wscopeperf.rva]
$set process/name=monperf/priority=15
$s
$::= f$lctime(,,"weekday")
$goto 'x'
$$
$sunday: goto normal
$monday: goto normal
$tuesday: goto normal
$wednesday: goto special
$thursday: goto normal
$friday: goto normal
$saturday: goto normal
$s
$normal: end="16:45:00"
$goto next
$$
$special: end="10:45:00"
$goto next
$$
$next:
$monitor/begin=07:45:00/end="end"/nodes/inter=150/rec processes,disl,system
$set process/priority=4
```

The purge command is only applied in the batch job of the VAX which starts executing first. In our case, the command deletes the log files of the batch jobs which measured VAXes LUWRVA, LUWRVB, LUWRVC, LUWRVD the preceding day.

Next, the batch job submits itself for the next day. For the batch job which measures VAX LUWRVA this is 7.40 am. However, if the automatic submitting fails, the user can submit the batch job with a command like

```
subrva:==submit/keep/noprinter/queue=LUWRVA_BATCH/priority=200/

"'tomorrow + 07:40"'
```

Figure 6.2: Operation of the Batch Job
A suggestion might be to place this command for each VAX in the LOGIN.COM. The batch job is set in the appropriate directory, directory [WSCOPERF.RVA] and will execute under the name MONPERF at priority 15. The priority have to be this high, since after each interval of 3 minutes samples have to be taken.

The goto statement defines the end instants of collecting samples for each day of the week. In this example, all end instants are set on 16.45 pm, except the instant at Wednesday, which is set on 10.45 am. Note that the period of measurement has to be a multiple of 3 minutes. For Wednesday, this multiple equals 60, for all other days 180.

The command beginning with "monitor" conducts the actual collections of samples between the specified begin and end instant. The term "nodisplay" means that the collected data will not appear on the screen. Finally, "interval=180/record processes, disk, system" means that after each 180 seconds samples of the three mentioned monitor utilities are written to a hexadecimal file MONITOR.DAT. The name of this file as well as the interval size cannot be altered! For more detailed information we refer to [7].

After the collection of samples has stopped, the basis priority is reset on 4 and the joining can start.

2. The samples have been collected over the desired period. Before program JOIN-MON is able to read the measurements, the samples have to be translated into a readable form. For the samples of MONITOR PROCESSES this is done by the following commands.

```$ monitor/inp/display=zzproc.dat processes
$ search/nohead/output=pro.data zzproc.dat ";
$ delete zzproc.dat"```

First, the file ZZPROC.DAT is created. Since this file contains many repeating phrases (the heading of MONITOR PROCESSES), the file is shortened by deleting these phrases. The result is file ZPRO.DAT. Consequently, file ZZPROC.DAT is deleted.

3. File ZDIS.DAT is created similar to the creation of ZPRO.DAT.

```$ monitor/inp/display=zdsl.dat disk
$ search/nohead/output=dis.data zdsl.dat ";
$ delete zdsl.dat;`
4. File ZSYS.DAT is deducted from file ZZSYST.DAT by extracting all phrases beginning with "Idle", "Page", "Free" and "Direct".

```
$ monitor/inp/display=zzsy$t.dat system/all
$ search/nohead/output=zzst.dat zzsst.dat "Idle","Page","Free","Direct"
$ delete zzsst.dat
```

5. The following command supplies data of the VAX LUWRVA

```
$ run disp/user5:[wscoper1]cpurva
```

Program CPURVA is a very small and simple program in the main directory, which creates a file with specific information about VAX LUWRVA needed for the joining of the samples of VAX LUWRVA. The information is read from the CONFFILE.DAT and written to file CPUDATA.DAT in the main directory. It contains the maximum number of active processes and the main memory size of VAX LUWRVA and the number of disks. The file CPUDATA.DAT is read by the program JOINMON to join the samples in conformity with the size of VAX LUWRVA. Right after usage, file CPUDATA.DAT is deleted. Since we use one program JOINMON, the other VAXes also have to write their system information in a file with name CPUDATA.DAT. Consequently, the batch jobs should not end at the same time, in order to prevent interchanging of different CPUDATA.DAT files! In our example, the other programs in the main directory are called CPURVB, CPURVC and CPURVD.

6. The command which results in the execution of program JOINMON equals

```
$ run disp/user5:[wscoper1]joinmon
```

Program JOINMON needs the files ZPRO.DAT, ZDIS.DAT and ZSYS.DAT for the system workload of VAX LUWRVA, file CPUDATA.DAT for the system information of VAX LUWRVA and file CONFFILE.DAT for the logical names of the disks and the names of all system processes appearing on VAX LUWRVA. The disk names are used to identify the names of the disks appearing in the samples coming from MONITOR DISK which the user wants to measure. The names of the system processes are used to select information about these processes from the samples coming from MONITOR PROCESSES.

Program JOINMON deletes the file CPUDATA.DAT before a batch job measuring another VAX invokes this program.

The result of the joining and averaging is file VERZMON.DAT.

---

1 It is not necessary to measure each disk in the VAX/VMS-cluster.
7. The last commands in the batch job concern the following

```plaintext
$ newname = "vrva_" + strftime("%Y%m%d", time()) - "-" - "-" - "
$ rename verzmon.dat 'newname'.dat
$ set protection=0=r-wed monltor.dat
$ delete monitor.dat;
$ delete zpro.dat;
$ delete zdis.dat;
$ delete zsys.dat;
$ set protection=0=r-we 'newname'.dat
$ exit
```

File VERZMON.DAT is renamed into a name with the node name LUWRVA and the date of creation included. The daily data file is also given a protection, so that the file cannot be removed by mistake.

Finally, the files used in the measuring are deleted. However, the deletion of file MONITOR.DAT does not always succeed, in spite of the removal of the protection. In Section 5.2.3 we have mentioned the consequences of this.

**Routine Construction of the JOINMON Program**

Program JOINMON joins the MONITOR data into one file, the daily data file. Therefore, the program consists of three parts. Reading the samples in files ZPRO.DAT, ZDIS.DAT and ZSYS.DAT, joining and averaging the MONITOR data and writing the results in VERZMON.DAT.

Since reading the samples is quite complicated, separate routines have been developed for reading the mentioned files. The second and third part, however, aren't divided into separate routines.

```
joining and averaging the MONITOR samples

read_cpudata read_conf_file read_samples

read_MONPRO read_MONDIS read_MONSYS

search
```

The first procedure `read_cpudata` reads and deletes the file CPUDATA.DAT. Procedure `read_conf_file` reads the logical names of all the disks and system processes which the user has specified in the file CONFFILE.DAT. All samples are read by procedure `read_samples`. The names of the system processes are needed for reading the file ZPRO.DAT in procedure `read_MONPRO`. Subprocedure `search` is applied in order to obtain MONITOR data of the preceding sample. Since all data in MONITOR
PROCESSES are cumulative and we want these data per interval of three minutes, the current data have to be compared to the data in the preceding sample. Procedure `read_MONDIS` reads file ZDIS.DAT. The disk names are used to identify the disk names in the samples. Finally, file ZSYS.DAT is read by procedure `read_MONSYS`.

### 6.2.3 User Interface Functions

#### Introduction

In order to avoid complex structures of routines used in the user interface functions, we will first discuss the read procedures appearing in each of the functions. We will define a method of denoting the occurrences of the read procedures in the routines within the functions. It will distinguish the routines which provide the actual user interface from the routines which are merely built for calculating.

A routine is either a procedure or a function. Do not confuse this kind of function with the functions of the VAMP package!

In the following Section, we will discuss the routines used to start and terminate a user interface session.

In the remaining Sections we will discuss the routine construction within each function of the user interface program as displayed in the flowchart of Figure 6.1.

#### Reading Commands

We have developed procedures for reading integers, reals, arrays of integers and reals, dates and options.

All read procedures are based on procedure `read_command`. This procedure reads every command at each data entry and translates the strings between the spaces in the command in the following way. If a string only contains alphabetic characters, the string is translated into one single character (mostly the first character of the string), otherwise, the string remains the same. Further, procedure `read_command` separates the translated strings by exactly one space. Procedures `rem_space` and `rem_word` are used in the translation, respectively for removing spaces and words in the original command.

Further, we have developed procedure `get_option` for separating the string before the first space from the translated command, leaving the string after the first space as the remaining string. For example, if procedure `read_command` reads an array of integers, all integers are set exactly one space from each other. Procedure `get_option` can be used to extract the integers from the translated command.

The boolean functions `is_an_integer` and `is_a_real` require a translated command to verify whether the string before the first space is an integer respectively a real. If the check succeeds, these functions convert the string into an integer or real.

If in this way an integer or a real has been extracted from the translated command, the boolean functions `in_integer_range` and `in_real_range` can be applied for a range check. If one of the boolean functions become false, procedure `error_message` is accessed, re-
resulting in a display of an adequate error message.

Procedures read_command and get_option make use of the VAX pascal standard subroutines SUBSTR, LENGTH, INDEX and READLN. The boolean is.an.integer and is.a.real functions make use of the subroutine READV (see [8]).

We will now specify for each of the read procedures the described subroutines used, see Table 6.1.

<table>
<thead>
<tr>
<th>read_one_option</th>
<th>get_option</th>
<th>is.an.integer</th>
<th>is.a.real</th>
<th>in.integer range</th>
<th>in.real range</th>
<th>error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_option</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>O2</td>
</tr>
<tr>
<td>read.build_option</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>O3</td>
</tr>
<tr>
<td>read.integer</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>I3</td>
</tr>
<tr>
<td>read.integers</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td>I4</td>
</tr>
<tr>
<td>read.date</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td>R1</td>
</tr>
<tr>
<td>read.mmyy</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td>R2</td>
</tr>
</tbody>
</table>

Table 6.1: Read Procedures and their Subroutines

Error messages for the procedure read_one_option are implemented within this procedure. The similarity between procedures read.mmyy, read.date and read.integers is quite obvious, since each of the procedures require an array of integers. However, in order to copy with different error messages, we have chosen to program three different procedures.

We have given each procedure an identifying number. In the following we will add these numbers to each routine which make use of the read procedures. In several procedures, a file name has to be read. Occurrence of this, will be denoted by the number O4.

Starting and Terminating the User Interface Program

The user interface program can only execute if data file CONFFILE.DAT is in the directory of the user interface program. Further, this data file has to contain the right information about the system configuration.

The following procedures are invoked when the user interface is started. Procedure read_conffile reads file CONFFILE.DAT, in order to supply the user interface program with information about the system configuration. As a consequence, the procedure
configuration_record creates a record by occupying the fields corresponding to configuration features with the information from CONFFILE.DAT. The remaining fields are all set on zero. The exact contents of the record will be discussed in Appendix D. The configuration record is used in procedure display, which displays the existing system configuration. The user can start entering commands after the procedure menu.VAMP has displayed the menu at VAMP level.

At this point, the VAMP> prompt appears. All the options entered after this prompt are read by procedure read_option (O2).

Concerning the terminating of the user interface program, at most one procedure is invoked. If the user has chosen at least one of the functions for the cluster management, the procedure write_clusfile is accessed. Since at the first invocation of one of the functions for the cluster management the file CLUSFILE.DAT is read (a file containing all stored clusters, built in earlier user interface sessions) and the contents might have been changed, this file has to be rewritten. The old file CLUSFILE.DAT is not needed any longer and is deleted.

Amalgamate the Daily Data Files

The basic idea behind this function is to amalgamate the daily data files obtained from measuring a VAX in one month, into two monthly data files with the same format as the daily data file in order to save disk quota. The procedure construction of this function can be seen below. Procedure convert_to_string and the function month_name are used to convert the specified month and year (in format "mm yy" or "m yy") into a string.
which makes part of the name of the daily data files of that month and year. Remember that the name of the daily data file include the date of creation.

Function `days_in_month` is used to generate all dates and thus all names of the possible daily data files of the specified month and year. Consequently, function `textfile.open` sees whether each file with the constructed name of the daily data file exists in the directory of the corresponding VAX.

If a constructed name has a corresponding daily data file, procedure `conj_dayfile` is applied to conjunct the file with the generated name with the amalgamation of the existing daily data files of all preceding days in the month. If the file has been created on a working day (function `day_of_week`), the file is also conjuncted with the amalgamation of the existing daily data files of all preceding working days in the month.

**Determination of the Model Parameters**

This function is also based on the idea of the conjunction of daily data files. Only now, this conjunction is not our purpose, since the conjunction of the daily data files is made in order to deduct the model parameters belonging to the period of conjunction. Moreover, a conjunction of daily data files of each VAX in the VAX/VMS-cluster is made. The procedure `det_modpar` is the only procedure within this function which provides

```
model parameter determination
  convert_to_string
  month_name
  day_of_week
  days_in_month
  textfile_open
```

the user interface. The procedure `convert_to_string` and the functions `textfile.open`, `day_of_week` and `month_name` are used in the same way as in the function to amalgame daily data files. Thus, generating names of daily data files. Usage of procedure `conj_dayfile` is also similar to this function. The result is a file with joined and averaged daily data files over a specified period with weekends included or excluded, dependent on the choice of the user in the beginning of the model parameter determination. This file is an internal file and will be deleted by terminating the user interface program. The file is read by procedure `read_dayfile` in order to create input for procedure `cal_modpar`, which actually calculates the model parameters belonging to one VAX. After calculating the model parameters of each VAX, the latter procedure writes the parameters in a file with a name specified by the user (default PAROUT.DAT). Procedure `display_modpar` provides a short display of the file.
Example Configuration Performance

Procedure `read_modpar` asks a name of a file with model parameters belonging to a certain period (default PAROUT.DAT). Function `textfile_open` is used to locate the file in the directory where all files with model parameters have been placed. The name of the directory is defined in file CONFFILE.DAT (see Appendix A). If the specified file has been found and the file indeed contains model parameters, procedure `read_modpar` reads the file in order to provide the algorithm for calculating the performance with model parameters. The procedure `performance_record` fills the remaining fields of the configuration record with the model parameters. Now, the record contains all relevant information for a performance calculation. That is, data of the configuration and the system workload, expressed in the model parameters, upon this configuration in a certain period. Only when the record is filled like described, procedure `performance` can be invoked. This procedure asks the user how many active and batch processes per VAX he wants to include in the performance calculation. Before the entered numbers of active and batch processes are accepted, range checks are applied by means of the boolean function `in.real.range`.

Procedure `f.in` opens file FODIN.DAT in order to fill this file with inputs for the algorithm. The inputs are the specified number of active and batch processes per VAX and the model parameters belonging to these numbers of active processes. Before procedure `f.in` writes the model parameters in file FODIN.DAT, the procedure averages the model parameters with the model parameters belonging to contiguous numbers of active processes.

Consequently, procedure `performance` opens a file with a name specified by the user (default FODOUT.DAT) for writing the input and especially the output of the performance calculation. Procedure `display` takes care of the input by writing the system configuration. The output is provided by procedure `SFODI`, the procedure with the implemented algorithm. Procedure `swfod` supplies the procedure `SFODI` with data from file FODIN.DAT. The short display of the performance is also generated by procedure `SFODI`.

At the end of procedure `performance`, file FODIN.DAT is deleted.
Build Clusters

The first time that one of the functions for the cluster management is invoked, procedure `read_clusfile` reads the file CLUSFILE.DAT. This file contains clusters built in earlier user interface sessions and has to be in the directory of the user interface program. An empty file is allowed. The result is the creation of a number of records, one for each cluster, with only the fields for the configuration filled. Procedure `build` contains many subroutines, partly optional. If the user wants to continue building an early built cluster, procedure `set_cluster` is invoked. The user has to choose a cluster with aid of the cluster survey, displayed by procedure `survey_clusters`. If the number of stored clusters exceeds `maxclusnumber`, the user has to remove at least one cluster in procedure `remove_cluster`. Subprocedures `survey_clusters`, `display` and `delete_cluster` are used for the removal.

Anyway, procedure `menu_BUILD` is always invoked. It displays the menu at BUILD level. The following six procedures can optionally be invoked and are used to adapt the record of the cluster which the user is building with to the corresponding configuration changes.

2All program parameters are described in Appendix C.
Procedure *display* provides a display of the cluster, like the display of the existing configuration. Besides invoking this procedure within the context of building clusters, it is invoked in the procedures *set_cluster* and *remove_cluster*.

Each build session is terminated with procedure *store_cluster*. If the user wants to delete the built cluster, the procedure *delete_cluster* is invoked.

Procedure *store_cluster* and thus this function is closed with the cluster survey (when there are stored clusters).

**Remove Clusters**

Procedure *read_clusfile* is either invoked or not, dependent on earlier invocations of one of the functions for the cluster management. The procedure *remove_cluster* and

```
remove_clusters
```

```
read_clusfile * remove_clusters 01,11
```

```
display  survey_clusters  delete_cluster
```

the subprocedures *display*, *survey_clusters* and *delete_cluster* are used similar to the function for building clusters.

**Example Cluster Performance**

Many of the procedures which this function accesses, equals the procedures in the function to calculate the performance of the existing configuration. The procedures are capable of handling a great variety of VAX/VMS-clusters. They only require a record with configuration and performance data.

Dependent upon earlier invocations of one of the functions within the cluster management, procedure *read_clusfile* is accessed. Procedure *read_modpar* works exactly the

```
example_cluster_performance
```

```
read_clusfile * choose_display 01,11 * performance_record
```

```
display
```

```
add_vax_perf
```

```
quicken_vax_perf
```

```
extend_vax_perf
```

```
disk_perf
```

```
performance_04,12
```

```
read_modpar 04
```

```
textfile_open
```

same. Procedure *performance_record* is invoked in order to occupy the fields for the
model parameters in the record of the cluster with which a performance calculation will be made. For each stored cluster, the fields concerning the configuration has been altered in the function for building clusters. However, the adaptation of the model parameters to these configuration changes of the cluster are made at this point in the appropriate sub-procedure of the procedure `performance_record`. The procedure `disk_perf` is invoked if the cluster has been created by invoking one of the procedures `add_disk`, `change_disk` or `change_diskload` at the cluster construction. Finally, procedure `performance` is accessed. The record for the cluster is handled the same way as the record for the existing configuration in the function to calculate the performance of the configuration.
### Appendix A

#### Data Files

#### A.1 CONFFILE.DAT

**contents** Relevant information about system elements in the VAX/VMS-cluster and the conjunction of all names of system processes appearing on each VAX.

**directory** `<vam>`

**responsibility** In order to handle changes in the actual system configuration efficiently, without adjusting and recompiling some of the programs of the VAMP package. The programs JOINMON and VAMP read this file.

**structure** The structure will be defined on the basis of Figure A.1. Don’t replace the colons or add empty lines! The first two lines must have position 11 from the left margin. All other colons position 9.

- number of VAXes
- number of disks
- per VAX its type (max. 11 characters), processor speed compared to the VAX-11/750 machine and main memory size (bytes) and maximum number of active interactive processes.
- for each disk its name (at most six adjacent characters of the original logical name of the disk, flushed to the left if the logical names in MONITOR DISK are flushed to the left and flushed to the right otherwise). Further, one space followed by six characters for specifying the type of the disk. The last column is for the access time (seconds).
- name of directory in which files with model parameters are written.
- name of directory in which files with performance data are written.
- per VAX the name of the directory in which the daily data files have been written, together with the beginning of the name of these files (preferably an abbreviation of the node name of the corresponding VAX).

- the conjunction of all names of system processes appearing on the VAXes in the system configuration (max. 15 characters).

```
nombre: va.xes: 3
nombre: d1sis: 9

<table>
<thead>
<tr>
<th>name</th>
<th>speed cf. VAX-11/750</th>
<th>MM size</th>
<th>max. act. pro.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>VAX-11/750</td>
<td>1</td>
<td>8000</td>
</tr>
<tr>
<td>VAX-2</td>
<td>VAX-11/750</td>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>VAX-3</td>
<td>VAX-8530</td>
<td>5</td>
<td>16000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>access time</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1si-1</td>
<td>USER1</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-2</td>
<td>USER2</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-3</td>
<td>USER3</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-4</td>
<td>COMMON</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-5</td>
<td>USER4</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-6</td>
<td>USER5</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-7</td>
<td>FILE</td>
<td>0.038</td>
</tr>
<tr>
<td>d1si-8</td>
<td>XIVAD</td>
<td>0.052</td>
</tr>
<tr>
<td>d1si-9</td>
<td>AFFL</td>
<td>0.052</td>
</tr>
</tbody>
</table>

directory for model parameters (PAR) and input SFODI (FOD)
PAR : USER4: [WSCOPPERF.RAM.PAR]
FOD : USER4: [WSCOPPERF.RAM]

directory and the beginning of the names of daily data files
VAX-1 : USER4: [WSCOPPERF.RC1]VRC1_
VAX-2 : USER4: [WSCOPPERF.RC2]VRC2_
VAX-3 : USER4: [WSCOPPERF.RC3]VRC3_

conjunction of names of all system processes appearing on each VAX
NULL
SWAPPER
NETACP
EVL
ERFFMT
CACHE_SERVER
CLUSTER_SERVER
OPCOM
JOB_CONTROL
CONFIGURE
FSJACF
REMOTE
HETUNE5_Daemon
HETUXD_Daemon
File_Daemon
Mail_Daemon
Prof_Daemon
VAPRT_Daemon
VAKJEB_Daemon
HEITUEI_Daemon
HEITUESI_Daemon
HEIIPDS_Daemon
Syst. usage

Figure A.1: Example of Data File CONFFILE.DAT
A.2 VERZMON.DAT

contents The output of the joining and averaging of the samples taken from one VAX during a contiguous period of the day. First part, the sorting based on the measured number of active processes. Second part, the sorting based on the measured free page list size.

directory <VAX-1>, ..., <VAX-n>

responsibility Creating smaller files by joining the samples and storing only the information that is really needed for the determination of the model parameters.

structure The structure of the first part will be defined on the basis of Figure A.2. The first part begins and ends with the date of the creation of the file. It contains as many repeating sections as the maximum number of active processes defined in the CONFFILE.DAT. Each section includes data measured at intervals in which a certain number of active processes were interacting with the VAX. More precisely, each section contains the following data items.

Line 1
- the number of active processes
- the number of samples measured with this number of active processes
- the average interval length of the samples
- the average number of batch processes in the samples

Line 2, respectively the cumulative number of CEF, LEF, PFW, HIB, SUSP, FPW and COM state occurrences

Line 3
- the direct IO rate of active processes
- the direct IO rate of batch processes
- the direct IO rate of active, batch and system processes together
- the page fault rate of active processes
- the page fault rate of batch processes
- the page fault rate of active, batch and system processes
- the read IO rate of active, batch and system processes together
- the fraction CPU time spent on active processes
- the fraction CPU time spent on batch processes
- the fraction CPU time spent on active, batch and system processes together

Line 4, the disk IO rate of all processes together to each of the disks
The structure of the second part will be defined on the basis of Figure A.3. The second and last part begins with the date of creation of the file. It contains data sorted at the free page list size. Therefore, the free page list size is divided into 20 classes. For example, if a VAX has 24 Mb main memory, class i represents a free page list size between \((i - 1)\frac{24}{20}\) Mb and \(i\frac{24}{20}\) Mb. Consequently, the second part contains 20 repeating sections of the following structure.

Line 1

- free page list size
- the number of measured samples with this free page list size
- the average interval length of the samples
- the average number of active processes in the samples
- the cumulative number of LEF state occurrences in the samples

Line 2-3, the distribution over the numbers of active process which led to the average number of active processes in these samples. In the first section of Figure A.3 the first appearing number of 4 equals the number of samples with 12 active processes measured at a free page list size between 0 and 800 Mb. Note that the maximum number of active processes equals 45.

Line 4-5, contain the same information as in the last two lines in the first part, only now it is sorted differently.
Figure A.2: Fragment of an Example of First Part of Data File VERZMON.DAT

```
23 160.7 10.870 231
23 0 0 0 0 0 0 1 0 2 2 4 3 3 4 0 0 1 0 2 0 1 0 0 0
23 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
84 160.5 11.476 819
5 1 1 2 0 4 5 6 3 9 4 9 9 5 6 8 5 2 2 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1400 0.04 3.73 76.53 6.26 90.45 4.80 0.53 0.36 1.60
1400 0.21 0.24 2.57 0.25 3.83 2.03 0.66 0.00
```

Figure A.3: Fragment of an Example of Second Part of Data File VERZMON.DAT

```
2 181.0 6.000 15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.04 0.01 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20 0.22 0.24 0.26 0.28 0.30 0.32 0.34 0.36 0.38 0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54 0.56 0.58 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80 0.82 0.84 0.86 0.88 0.90 0.92 0.94 0.96 0.98 1.00
```

105
A.3 VAXDATA.DAT

contents Characteristics of a VAX processor, needed for the joining of the samples generated by this VAX. At the time samples of a VAX have to be joined, this file is created.

directory < main >

responsibility In order to ensure the program JOINMON to join MONITOR samples taken from VAXes of different sizes.

structure One line with respectively the number of disks, the size of main memory and the maximum number of logged in interactive processes, who can be active.
A.4  ZPRO.DAT

contents A number of samples taken from MONITOR PROCESSES of one specific VAX during a contiguous period throughout the day. The heading of each sample has been removed. The file is produced by the batch job which measures the VAX.

directory $< VAX - 1 >, \ldots, < VAX - n >$

responsibility In the context of measuring the system workload, measuring the activities of the various processes competing for CPU time.

structure The structure will be defined on the basis of Figure A.4.

The samples form repeating sections between the phrases containing the same date and different time. Per process appearing in a section, the following data is written:

- the Process IDentification (PID) number
- the state of the process at the time the corresponding sample was taken
- the current priority of the process
- the name of the process
- the current number of global/private pages in the WS of the process
- the number of direct IO of the process since its creation
- the number of page faults of the process since its creation
- the used CPU time since the creation of the process in hours, minutes, seconds and tenth of seconds.
Figure A.4: Fragment of an Example of Data File ZPRO.DAT
A.5 ZDIS.DAT

contents  A number of samples taken from MONITOR DISK of one specific VAX during a contiguous period of the day. The heading of each sample has been removed. The file is produced by the batch job which measures the VAX.

directory  $<\text{VAX} - 1>, \ldots, <\text{VAX} - n>$

responsibility  In the context of measuring the system workload, measuring disk IO activity of processes belonging to the VAX under consideration.

structure  The structure will be defined on the basis of Figure A.5. Repeating sections between lines with the date and the time. Per disk the system name and the logical name and the following disk IO activity.

- average disk IO rate in the last 3 minutes
- average disk IO rate over all preceding samples, generated since the start of the batch job
- the lowest average disk IO rate per 3 minutes since the first sample
- the highest average disk IO rate per 3 minutes since the first sample
<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Time</th>
<th>User</th>
<th>FEZ</th>
<th>SYS</th>
<th>USEF</th>
<th>RE</th>
<th>RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDIS.DAT</td>
<td>28-JUL-1988</td>
<td>12:25:54</td>
<td>USER1</td>
<td>0.21</td>
<td>0.00</td>
<td>0.00</td>
<td>8.67</td>
<td></td>
</tr>
<tr>
<td>ZDIS.DAT</td>
<td>28-JUL-1988</td>
<td>12:25:54</td>
<td>USER2</td>
<td>0.00</td>
<td>1.97</td>
<td>0.00</td>
<td>35.54</td>
<td></td>
</tr>
<tr>
<td>ZDIS.DAT</td>
<td>28-JUL-1988</td>
<td>12:25:54</td>
<td>INSTAL</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ZDIS.DAT</td>
<td>28-JUL-1988</td>
<td>12:25:54</td>
<td>USER3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>ZDIS.DAT</td>
<td>28-JUL-1988</td>
<td>12:25:54</td>
<td>USER4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure A.5: Fragment of an Example of Data File ZDIS.DAT
A.6 ZSYS.DAT

contents A number of samples taken from MONITOR SYSTEM/ALL of one specific VAX during a contiguous period of the day. The heading of each sample has been removed. The file is produced by the batch job which measures the VAX.

directory < VAX - 1 >, ... , < VAX - n >

responsibility In the context of measuring the system workload, measuring general system activities of all processes belonging to the VAX under consideration.

structure The structure will be defined on the basis of Figure A.6.

The samples form repeating sections starting with "Idle" and ending with "Direct". It contains the following items, respectively averaged over the last 3 minutes, over all preceding samples, and the maximum and minimum values since the first sample.

- idle time of the CPU
- page fault rate of all processes belonging to the VAX together
- free page list size (in pages)
- direct I/O rate of all processes together

| Free List Size | 16818.00 | 14800.76 | 3181.00 | 29288.00 |
| Direct I/O Rate | 0.42 | 15.36 | 0.10 | 72.82 |
| Idle Time | 96.77 | 76.77 | 11.77 | 99.11 |
| Page Fault Rate | 8.25 | 71.20 | 2.85 | 277.22 |
| Page Read I/O Rate | 0.06 | 1.44 | 0.00 | 5.18 |
| Free List Size | 16855.00 | 14818.59 | 3161.00 | 29288.00 |
| Direct I/O Rate | 0.69 | 15.22 | 0.10 | 72.82 |
| Idle Time | 84.96 | 76.80 | 11.77 | 99.11 |
| Page Fault Rate | 27.27 | 70.74 | 2.85 | 277.22 |
| Page Read I/O Rate | 0.39 | 1.47 | 0.00 | 5.18 |
| Free List Size | 17862.00 | 14847.77 | 3161.00 | 29288.00 |
| Direct I/O Rate | 5.91 | 15.17 | 0.10 | 72.82 |
| Idle Time | 74.67 | 76.84 | 11.77 | 99.11 |
| Page Fault Rate | 58.64 | 70.65 | 2.85 | 277.22 |
| Page Read I/O Rate | 0.38 | 1.45 | 0.00 | 5.18 |
| Free List Size | 17865.00 | 14874.57 | 3161.00 | 29288.00 |
| Direct I/O Rate | 10.40 | 15.09 | 0.10 | 72.82 |
| Idle Time | 96.29 | 77.02 | 11.77 | 99.11 |
| Page Fault Rate | 12.15 | 70.06 | 2.85 | 277.22 |
| Page Read I/O Rate | 0.28 | 1.41 | 0.00 | 5.18 |
| Free List Size | 17012.00 | 14894.51 | 3161.00 | 29288.00 |
| Direct I/O Rate | 0.72 | 14.95 | 0.10 | 72.82 |

Figure A.6: Fragment of an Example of Data File ZSYS.DAT
A.7 PAROUT.DAT (default name)

contents  The model parameters of a certain period, sorted at the measured number of active processes and the measured free page list size. At both sortings, the file also contain comparisons between the relative workloads per active process.

directory < par >

responsibility  Creation of a file with model parameters permits usage of the parameters in various performance calculations, both current and predictive. Further, writing the model parameters in a file allows for inspection. The model parameters always indicate a certain system workload.

structure  The first part contains the model parameters sorted at the number of active processes, as shown in Figure A.7.

VAX-2(VAX-11/750), SORTED AT THE NUMBER OF ACTIVE PROCESSES

<table>
<thead>
<tr>
<th>act. number</th>
<th>per INTERACTIVE process</th>
<th>batch (per BATCH process)</th>
<th>relative visit frequency to disk</th>
<th>workload</th>
<th>thinktime</th>
<th>waittime</th>
<th>number workload</th>
<th>workload</th>
<th>workload</th>
<th>workload</th>
<th>workload</th>
<th>workload</th>
<th>workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>115</td>
<td>0.080</td>
<td>0.011</td>
<td>0.288</td>
<td>0.000</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>126</td>
<td>0.082</td>
<td>0.026</td>
<td>0.429</td>
<td>0.214</td>
<td>1.0</td>
<td>0.025</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>67</td>
<td>0.082</td>
<td>0.022</td>
<td>0.362</td>
<td>0.141</td>
<td>1.0</td>
<td>0.025</td>
<td>0.004</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>157</td>
<td>0.083</td>
<td>0.024</td>
<td>0.328</td>
<td>0.139</td>
<td>1.0</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5</td>
<td>107</td>
<td>0.084</td>
<td>0.022</td>
<td>0.288</td>
<td>0.101</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td>0.067</td>
<td>0.023</td>
<td>0.284</td>
<td>0.131</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
<td>0.087</td>
<td>0.036</td>
<td>0.237</td>
<td>0.143</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0.060</td>
<td>0.036</td>
<td>0.213</td>
<td>0.125</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.071</td>
<td>0.045</td>
<td>0.078</td>
<td>0.000</td>
<td>1.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Figure A.7: Fragment of an Example of the First Part of Data File PAROUT.DAT

For each VAX in the system configuration, the number of allowed active processes is written and per number the number of measured samples with this number of active processes. The maximum allowed number of active processes equals the maximum number of active processes in the CONFFILE.DAT. Further, per number of active processes the relative workload per active process at terminal, CPU and disk and the relative wait time per active process. (Remember that we formulated one disk workload per process class) Next, the measured number of batch processes in
the samples corresponding to the number of active processes and per batch process the relative workload at CPU and disk. Finally, the relative disk visit frequencies are written. These frequencies equal for the active and batch class belonging to one VAX.

For each VAX, this display is followed by a display of the ratio between terminal workload per active process and the CPU and disk workload per active process and a display of the ratio between the CPU workload per active process and the disk workload per active process. For an example of this display, see Figure A.8

<table>
<thead>
<tr>
<th>terminal time</th>
<th>CPUworkload</th>
<th>diskworkload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>0.276</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td>0.196</td>
</tr>
<tr>
<td>3</td>
<td>1.000</td>
<td>0.225</td>
</tr>
<tr>
<td>4</td>
<td>1.000</td>
<td>0.241</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td>0.254</td>
</tr>
<tr>
<td>6</td>
<td>1.000</td>
<td>0.235</td>
</tr>
<tr>
<td>7</td>
<td>1.000</td>
<td>0.267</td>
</tr>
<tr>
<td>8</td>
<td>1.000</td>
<td>0.241</td>
</tr>
<tr>
<td>9</td>
<td>1.000</td>
<td>0.415</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPUworkload</th>
<th>diskworkload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.136</td>
</tr>
<tr>
<td>2</td>
<td>0.299</td>
</tr>
<tr>
<td>3</td>
<td>0.375</td>
</tr>
<tr>
<td>4</td>
<td>0.267</td>
</tr>
<tr>
<td>5</td>
<td>0.267</td>
</tr>
<tr>
<td>6</td>
<td>0.341</td>
</tr>
<tr>
<td>7</td>
<td>0.412</td>
</tr>
<tr>
<td>8</td>
<td>0.648</td>
</tr>
<tr>
<td>9</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Figure A.8: The Ratio between the Relative Workloads per Active Process

The second part contains the sorting at the free page list size. These data are not used in the performance calculations, since memory allocation is not considered in our model.

For each VAX, the main memory size is divided into 20 classes. If the main memory of a VAX equals 24 Mb, then the classes run from 1.2 Mb to 24 Mb. Per class \(i\), the number of samples with a measured free page list size between \((i - 1) \cdot \frac{24}{20} \text{ Mb}\) and \(i \cdot \frac{24}{20} \text{ Mb}\) and the relative terminal, CPU, and disk workload per active process is written, see Figure A.9.

\(^1\)In the VAMP package implemented at the WUA, the frequencies have been multiplied by 100.
VAX-11/780, SORTED AT THE FREE PAGE LIST SIZE

<table>
<thead>
<tr>
<th>list</th>
<th>size</th>
<th>rel. term</th>
<th>rel. CPU</th>
<th>rel. disk</th>
<th>workload</th>
<th>work load</th>
<th>think time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>255</td>
<td>0.056</td>
<td>0.047</td>
<td>0.713</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>420</td>
<td>0.065</td>
<td>0.058</td>
<td>0.727</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>453</td>
<td>0.077</td>
<td>0.036</td>
<td>0.704</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>172</td>
<td>0.078</td>
<td>0.035</td>
<td>0.723</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>43</td>
<td>0.089</td>
<td>0.045</td>
<td>0.656</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td>26</td>
<td>0.125</td>
<td>0.082</td>
<td>0.618</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7000</td>
<td>70</td>
<td>0.055</td>
<td>0.050</td>
<td>0.714</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td>649</td>
<td>0.040</td>
<td>0.036</td>
<td>0.941</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9000</td>
<td>11</td>
<td>0.026</td>
<td>0.026</td>
<td>0.729</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure A.9: Fragment of an Example of the Second Part of Data File PAROUT.DAT

Again, for each class the ratio between terminal workload and CPU and disk workload per active process is displayed as well as the ratio between CPU workload and disk workload per active process. This supplies a display similar to the display of Figure A.8. Only now, the first column contains the free page list sizes.
A.8 CLUSFILE.DAT

contents The system configuration of clusters (max. \texttt{maxclusnumber + 1}) built and stored in earlier user interface sessions. If there are no stored clusters, the file must exist in an empty form.

directory \texttt{<vam>}

responsibility To avoid unnecessary rebuilding of clusters, constructed in earlier user interface sessions.

structure The structure will be defined on the basis of Figure A.10.

Line 1
- number of VAXes in the cluster
- number of disks in the cluster
- the identification number

Line 2, the number of times the various build options has been accessed at the construction of the cluster. Respectively build option
- add VAX
- quicken VAX
- extend VAX
- add disk
- change disk
- change load of the disks

Line 3 corresponds to the added VAXes. The number of the existing VAX which has been copied is written. If there are no VAXes added in the cluster, this line is deleted. Further, for each added VAX the number of the corresponding existing VAX is written on a new line.

Line 4 corresponds to the quickened VAXes. The number of the quickened VAX and its new speed compared to the VAX-11/750 computer is written. The results of successive invocations of this build option are written on successive lines.

Line 5 corresponds to the extended VAXes in the cluster. The number of the extended VAX and the extension factor is written. The results of successive invocations of this build option are written on successive lines.

Line 6-9. For each VAX in the cluster
- the maximum number of active processes
- the size of main memory (bytes)
- the processor speed compared to the VAX-11/750 machine
Figure A.10: *One stored Cluster in Data File CLUSFILE.DAT*

Logically, a different number of VAXes requires a different number of lines.
Line 10. The disk access times (msec) of each disk in the cluster.
Line 11-21. The diskload transfers in the cluster. Vertically, the disks from which diskload is to be removed. In the example, the option for changing the diskload has been invoked twice. This resulted in the values 0.25 and 0.15. Further, the added disk (the last disk) has become 30% of the load of disk-4.
A.9 FODIN.DAT

contents The relative workloads of the terminal station, CPUs and disks and the relative
disk visit frequencies, corresponding to the number of active processes entered at
the performance calculation in the user interface program. It contains an extraction
of the file with default name PAROUT.DAT and is deleted right after usage.

directory < fod >

responsibility To obtain a distance between the implemented algorithm and the user
interface part of the package, in order to make use of other algorithms in the future.

structure The structure will be defined on the basis of Figure A.11

Line 1

- number of VAXes
- number of disks
- number of process classes

Line 2, the number of processes per process class, running from active processes of
VAX-1 and batch processes of VAX-1 to batch processes of VAX-n
Line 3, the priority of each process class
line 4, the relative terminal workload per process class
Line 5-10, the relative workload of the active processes of VAX-1 until VAX-n at
the VAXes and the disks
Line 11-16, the relative visit frequencies of active processes of VAX-1 until VAX-n
to VAXes and disks
Line 17, number of cycles per terminal visit for each process class

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0.11</th>
<th>0.08</th>
<th>0.70</th>
<th>0.19</th>
<th>0.24</th>
<th>0.02</th>
<th>0.07</th>
<th>0.00</th>
<th>0.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.11</td>
<td>0.08</td>
<td>0.70</td>
<td>0.19</td>
<td>0.24</td>
<td>0.02</td>
<td>0.07</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.51</td>
<td>0.10</td>
<td>0.03</td>
<td>0.07</td>
<td>0.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0.51</td>
<td>0.10</td>
<td>0.03</td>
<td>0.07</td>
<td>0.22</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>0.16</td>
<td>0.00</td>
<td>0.30</td>
<td>0.08</td>
<td>0.19</td>
<td>0.27</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure A.11: Example of Data File FODIN.DAT
A.10  FODOUT.DAT (default name)

contents  The file consists of two parts, the input and output of the performance calculation. The input consists of the date period of measurement which was used in the determination of the model parameters and a display of the configuration/cluster which defined the used model. The output consists of the results of calculating the model by the implemented algorithm.

directory  < fod >

responsibility  To support the system manager in efficiently handling the performance of a VAX/VMS-cluster by displaying the performance in a clear form.

structure  The structure of the first part will be defined on the basis of Figure A.12. The first part contains the following input of the performance calculation.

- the period of measurement
- the configuration/cluster display provided by procedure display

The structure of the second part will be defined on the basis of Figure A.13. The second part contains the performance results. More precisely, the relative workload, relative response times, mean queue length, throughput and utilization at

- the terminal for each process class (except the utilization)
- each VAX for each process class
- each disk for each process class

Furthermore, the priority, number of processes, absolute response time, relative response time, relative wait time and response time per CPU second for each process class
PERFORMANCE RESULTS OVER THE PERIOD (weekends included): 10 JUL 1988 - 18 JUL 1988

UNDERLYING CLUSTER (cluster 5)
-------------------------------------

VAXes of the CONFIGURATION:

<table>
<thead>
<tr>
<th>type</th>
<th>speed cf. VAX-11/750</th>
<th>Main Memory size</th>
<th>changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-1</td>
<td>1.0</td>
<td>8.0 MB</td>
<td>no</td>
</tr>
<tr>
<td>VAX-2</td>
<td>2.0</td>
<td>15.0 MB</td>
<td>yes</td>
</tr>
<tr>
<td>VAX-3</td>
<td>5.0</td>
<td>16.0 MB</td>
<td>no</td>
</tr>
</tbody>
</table>

VAXes which are added in the CLUSTER:

<table>
<thead>
<tr>
<th>speed cf. VAX-11/750</th>
<th>Main Memory size</th>
<th>workload equals most with existing VAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX-4 5.0</td>
<td>16.0 MB</td>
<td></td>
</tr>
</tbody>
</table>

Number of disks in the CLUSTER: 10

<table>
<thead>
<tr>
<th>disk name</th>
<th>type</th>
<th>speed</th>
<th>fraction of removed disk</th>
<th>load of disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER1</td>
<td>RA-B1</td>
<td>0.078</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USER2</td>
<td>RA-B1</td>
<td>0.078</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USER3</td>
<td>RA-81</td>
<td>0.058</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>COMMON</td>
<td>RA-B1</td>
<td>0.036</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USER4</td>
<td>RA-B1</td>
<td>0.036</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USER5</td>
<td>RA-B1</td>
<td>0.036</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>POOL</td>
<td>RA-B1</td>
<td>0.036</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LLAD</td>
<td>RA-B1</td>
<td>0.052</td>
<td>-</td>
<td>0.15</td>
</tr>
<tr>
<td>APFL</td>
<td>RA-B1</td>
<td>0.052</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>???</td>
<td>???</td>
<td>0.041</td>
<td>-</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Figure A.12: Example of Input Part of Data File FODOUT.DAT
Table shows calculated performance statistics. The columns represent station, cl. type, and the various performance metrics such as total response time, completion time, and utilization. The rows correspond to different types of input files, including processor and disk unit data. The table includes values for different metrics, such as throughput and response time, for each station type.

Figure A.13: Example of Output Part of Data File FODOUT.DAT
Appendix B

Program Parameters

We will now define the used constraints and their restrictions per program or module of the VAMP package

JOINMON: joins the MONITOR samples.

- max_disk. The maximum number of measured disks in the VAX/VMS-cluster.
- max_processes. The upperbound for the number of interactive processes logged into the largest VAX of the VAX/VMS-cluster (≥ max. number of active processes in the CONFFILE.DAT for this VAX).
- max_sysprocesses. The upperbound for the number of names of all system processes appearing in the VAX/VMS-cluster.

VAMP, MOD1: contains the constants declaration of program VAMP.

- mazintacnumber. The maximum number of active interactive processes (≥ max. number of active processes in CONFFILE.DAT for the largest VAX).
- mazoptnumber. The maximum number of times each build option can be chosen (add, quicken and extend VAX, add and change disk and change disk workload).
- mazvaxnumber. The maximum number of VAXes allowed in the clusters (≥ number of existing VAXes + mazoptnumber).
- mazdisknumber. The maximum number of disks allowed in the clusters (≥ number of existing disk + mazoptnumber).
- mazclusnumber. The maximum number of clusters that can be stored. This maximum can be exceeded by one.
- numberoptions. The number of build options. Standardly set on 6.
- `numbervazoptions`. The number of build options concerning changes in the VAXes. Standardly set on 3.

**VAMP, MODSFODI**: constants concerning the model.

- `maxvazznumber`. The maximum number of VAXes in the model (\( \geq mazvaznumber \)).
- `mazdiskknumber`. The maximum number of disks in the model (\( \geq mazdiskknumber \)).
Appendix C

Directory Structure

It has already been mentioned that the VAMP package requires the creation of certain subdirectories. Besides a number of required directories, the user is able to choose separate directories for in- and output files of the functions of the VAMP package. The following diagram indicates the structure of the directories. Directory denoted by \(<\text{main}>\)

\[
\begin{align*}
\text{<main>} & \\
\text{<vam>} & \text{<VAX-1>} \cdots \text{<VAX-n>} \\
\text{<mod>} & \text{<par>} & \text{<fod>}
\end{align*}
\]

contains program JOINMON and the programs, one for each VAX, which supply the data file CPUDATA.DAT for JOINMON.
The batch jobs for measuring the VAXes execute within the directories denoted by \(<\text{VAX-1}>, \ldots, <\text{VAX-n}>\). Each of these directories contains daily data files and the command file PERFMON.COM.
The directory denoted by \(<\text{vam}>\) contains the user interface program, under the name VAMP, and the data files CONFFILE.DAT and CLUSFILE.DAT.
Subdirectory \(<\text{mod}>\) contains the modules which are linked to the VAMP program in the directory \(<\text{vam}>\). The modules are named MOD1, MOD2 and MODSFODI respectively including the read procedures, the function textfile_open and procedures SFODI and swfod.
The directories \(<\text{par}>\) and \(<\text{fod}>\) are optional in the sense that these directories can be altered without adapting one of the programs. Changing the directory specification in the file CONFFILE.DAT is sufficient. Mostly, these directories coincide with the directory denoted by \(<\text{vam}>\). They contain respectively the files with model parameters and the files with the performance results (default names respectively PAROUT.DAT and FODOUT.DAT).
Appendix D

Data Structure for VAX/VMS-clusters

The configurations of the VAX/VMS-cluster family which the VAMP package allows to build out of the existing configuration (the configuration which is measured) are stored in data records. These records also contain fields for storing model parameters of a certain period, in order to make performance calculations. The fields for the model parameters can be changed when the performance of another period has to be calculated.

Before we describe each field in the data record, we will define the structure. The subrecords are placed between the delimiters.

```
.nvx
.ndsk
.n
.tel
.hulp(.va,.gr)
.vz(.ps,.mp,.fg,.wc,.wd,.wt,.ww,.intv,.bapro,.dskbz,.cor)
.dsk(.tijd,.ovn)
```

The field .hulp has been developed for storing the clusters in CLUSFILE.DAT, without losing characteristic information about the clusters. Consequently, the values in this field are all set on zero for the configuration of the existing VAX/VMS-cluster, since this configuration is stored in the CONFFILE.DAT. The fields from field .wc to field .cor and the field .ovn are used to store the model parameters.

Before we define the contents of each field in an arbitrary configuration, we introduce the following indices. Index v to indicate a VAX, d to indicate a disk, p for the processes, r to indicate the active or batch process class of a VAX, o for the build options, ov for the build options for manipulating the VAXes and t to denote the number of times a build option has been chosen.
- .nvx the number of VAXes in the configuration.
- .ndsk the number of disks in the configuration.
- .n configuration identification number. Equals 0 for the existing configuration and 1 for each built cluster.
- .tel[o] the number of times build option o has been invoked at the cluster building.
- .hulp[ov].va[t] the number of the VAX which has been copied in the t-th invocation of build option add a VAX if ov=1 and the number of the VAX which has been quickened or extended in the t-th invocation of the corresponding build options, respectively in the case that ov=2 and ov=3.
- .hulp[ov].gr[t] the processor speed compared to the VAX-11/750 entered at the t-th VAX which has been quickened if ov=2 and the extension factor of the t-the VAX which has been extended if ov=3.
- .vx[v].ps the processor speed compared to the VAX-11/750 of VAX-v.
- .vx[v].mp the maximum number of active processes of VAX-v.
- .vx[v].fg the size of main memory of VAX-v.
- .vx[v].wc[r,p] the relative CPU workload of class r corresponding to VAX-v, selected at the number of p active processes.
- .vx[v].wd[r,p] the relative disk workload of class r corresponding to VAX-v, selected at the number of p active processes.
- .vx[v].wt[r,p] the relative terminal workload of class r corresponding to VAX-v, selected at the number of p active processes.
- .vx[v].ww[r,p] the relative wait time of class r corresponding to VAX-v, selected at the number of p active processes.
- .vx[v].intv[p] the number of measured samples with p active processes on VAX-v.
- .vx[v].bapro[p] the average number of batch processes measured in the samples with p active processes on VAX-v.
- .vx[v].dskbz[p,d] the relative disk visit frequencies of active as well as batch processes belonging to VAX-v to disk-h, measured in the samples with p active processes.
- .vx[v].cor[p] a correction factor, frequently used at the adjustment of the model parameters to the configuration changes in the cluster. It equals the average disk time. More precisely, the sum over the disks of the following expressions: .vx[v].dskbz[p,d] + dsk[d].tijd.
- $dsk[d].tijd$ the disk access time of disk-$d$.

- $dsk[d].ovn[d']$ the fraction of the diskload of disk-$d$ which is to be transferred to disk-$d'$.
Bibliography


