

A probabilistic model for evaluating the Verso software filter

V. Delebarre, Patrick Richard

▶ **To cite this version:**

V. Delebarre, Patrick Richard. A probabilistic model for evaluating the Verso software filter. RR-0462, INRIA. 1985. inria-00076092

HAL Id: inria-00076092
https://inria.hal.science/inria-00076092

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tél : (1) 39 63 55 11

Rapports de Recherche

N°462

A PROBABILISTIC MODEL FOR EVALUATING THE VERSO SOFTWARE FILTER

Véronique DELEBARRE
Philippe RICHARD

Décembre 1985

A PROBABILISTIC MODEL FOR EVALUATING THE VERSO SOFTWARE FILTER

Veronique DELEBARRE (*,**)
Philippe RICHARD (**)

Abstract

For database management systems,the performance problem is crucial. New technologies have emerged, which are intended to improve database performance. Specialized hardware were designed in order to fasten data access. In this paper we are mainly interested in filters. We develop a model to evaluate the response time of software filters implemented on standard disk exchange units."Off the shelf" processors are prefered to customized hardware for implementing filtering. Although hardware filtering is extremely fast,software filtering is adaptable to the evolution of technology for a low cost and can have an acceptable performance if well designed.This evaluation points out the weaknesses of the software approach and gives means to "optimally" implement software filters. The model has been applied to two kinds of implementation ,one of them is operational.

Resume

Le problème des performances est crucial pour les systèmes de gestion de bases de données relationnelles. Des materiels specialisés ont été concus dans le but d'accélerer les accès aux données. Dans ce papier, nous nous sommes principalement intéressés aux processeurs de filtrage implantés en machines dorsales . Nous avons developpé un modèle permettant d'evaluer le temps de reponse du filtre logiciel Verso implanté sur une unité d'échange disque standard. En effet,bien qu'ayant des performances moindres,les filtres logiciels sont préférés à une implantation materielle car ils sont moins coûteux et plus rapides à développer. De plus la facilité de leur conception permet de suivre l'évolution de la technologie. Cette évaluation met en evidence les faiblesses d'une implantation logicielle et donne des indications sur les points (code,gestion de la memoire,vitesse du processeur) à améliorer. Le modèle est applique à deux implantations dont l'une est operationelle.

(*) IIE- 18 allee J.ROSTAND,les passages BP. 77 -91002 EVRY
(**) INRIA -BP.105 - 78153 Le Chesnay Cedex

PAPIER RÉCUPÉRÉ ET RECYCLÉ

Introduction

This work is devoted to database machines evaluation. Due to the complexity of database management systems,the scientific community has payed a great attention to specialized hardware [Dewitt79, Schweppe83, Babb79, Armisen81, Bancilhon80]. Concurrently, the technology has significantly evolved and DBMS implementation on micro computer architectures are now available with increasing performance. In [Boral81], Boral and Dewitt are wondering whether database nachines time has passed. In the Verso project, two prototypes of a relational DBMS have been successively realized, the first one following the specialized hardware approach and the second one following the software approach. The Verso database machine was designed with the objectives of:

1) justifying the relegation of some tasks to a dedicated hardware close to the mass storage device.

2) checking whether an automaton-like device for the hardware filter was well adapted to the structure of the files to be processed.

The hardware filter first realized was then replaced by a standard disk exchange unit (DEU) utilizing an Intel 8086 processor and a 128 Kbyte Ram memory with an SASI (or SCSI) disk interface. The finite state automaton filter was thus implemented by software on the DEU. Gamerman evaluated the performance of the hardware filter in [Gamerman84], Scholl [Scholl85] performed a deterministic study of the software filter. They both evaluate the response time of the filter architecture and these two approaches are compared in [Gamerman85]. The deterministic study of the software filter shows that the performance of this architecture drastically depends on the DEU itself. Thus, one may question whether it is worth having a more powerful processor such as an 80186 or another disk interface (SMD instead of an SASI one). It was therefore interesting to make previsional and comparative evaluations of different configurations. We chose a probabilistic model because it could give another measure: the maximal throughput of the filter and it allows to study the maximal load which can be submitted to the filter. From the deterministic study, we concluded that the characteristics of the users' queries (selectivity factor, size of the the relation, number of attributes..) determined the filtering response time; we attempt to study the sensitiveness of the performance of a filter architecture to these different factors. As we take in account the load characteristics, we are concerned with random events.

1. Software filtering

1.1. Architecture

The Verso filter is implemented by software on a disk-exchange unit. The hardware frame of this realization is an SM90 micro-computer which has the following characteristics (cf. Fig. 1.1):

- an SM bus is shared by (a) a processing unit (PU), (b) a disk exchange unit.
- the PU plays the role of a host processor as in standard back-end database machines. It is in charge of high level DBMS functions (query decompositions, access path, transaction management and concurrency control) as well as filter control. It compiles data base queries, then sends them to the filter via the SM bus and gets results from the filter.

- the DEU plays the role of a back-end data base machine. It executes the compiled queries and transfers the filtered data to the PU or to the disk. The DEU includes the following components, which share a 16 bits local bus (c) (cf. Fig. 1.1)

1) an 8086 (or 80186) processor in charge of filtering, loading and unloading blocks of data,

2) an EPROM memory where the system Kernel is stored; it includes a real time monitor, tasks for the disk transfers and the dialogue with the PU,

3) a DMA sharing the local bus and the shared memory (MS) with the processor,

4) an 128 (or 512) K-byte dynamic Ram memory LM where is stored the filtering code.

The software filter consists in a sequence of concurring tasks which are executed by the CPU (8086 or 80186 depending on the type of DEU). These tasks are initiated when the PU asks for a filter query and are synchronized by checking the states of two pools of buffers in the memory LM. These buffers are shared by the tasks. (cf. Fig. 1.2.). We give in this section a brief description of these tasks in order to lighten the frame of our evaluation. (For a more detailed description see [Scholl85]).

The first task, called *the loader* is in charge of loading the set of blocks of the source relation from the disk to the source buffers. For each source block to be loaded, the loader asks for a free buffer. If none is free, it waits, otherwise one is allocated. Once the buffer has been loaded, it is entered in FIFO F_L . The loader stops when all blocks of the source relation have been loaded.

The second task called *the filter* asks for a loaded buffer of the queue F_L and a target buffer allocated from the target pool. It filters data from the source buffer SB into a target buffer TB. It stops when SB is empty or when TB is full. It asks then for a new SB or TB and waits if the queue F_L is empty or if the target pool is empty. A full buffer TB is entered into a FIFO queue F_U and the empty SB is given back to the source pool.

The third task called *the unloader* is in charge of unloading the target buffer onto the disk or to the PU via the SM bus. Once a TB has been unloaded, it is returned to the target pool. The unloader pops the FIFO queue F_U and waits if it is empty.

1.2. Logical and physical parameters of the evaluation

The Verso DBM is used for relational applications; data are structured in relations denoted $R(U)$ where U stands for a set of attributes, with cardinality !U!. Every attribute X takes its values in a domain of strings of size A_X bytes. In this paper we evaluate the filter when it executes a stream of selection -projection queries. Let us consider for instance, the relation R(Name,Age,Address,Tel-nb). A possible selection -projection operation may be : R(N,A,Ad,T)/Ad=Paris AND Age >20[N,T], which lists the name and the telephone of young parisians. The filter sequentially scans every block of the source relation,checks the selection clause and writes in the target buffer the projected attributes if the clause is true.
We denote by :

S the number of skipped attributes,
P the number of attributes which are to be projected only,
C the number of attributes which are involved in a selection clause,
σ is the selectivity ratio: $\frac{C+P}{!U!}$. For every tuple of the source relation, (C+P) attributes of the source relation will be written in the target relation if the selection clauses are true. These relational parameters are sufficient to evaluate the filtering time of a source block [Scholl85].

We suppose that all the attributes of U have the same length !A! and that $C = P$;
τ is the cycle time of the CPU,
S_b is the size of a source block,
ϑ_S is the mean number of CPU cycles necessary for skipping an attribute.
t_P , t_C are the mean number of CPU cycles necessary for projecting , comparing and copying respectively one character of an attribute from the source relation into the target relation.

$$\vartheta_P = \vartheta_S + !A!\, t_P$$

is the mean cycle time (in number of CPU cycles) for projecting an attribute.

$$\vartheta_C = 2\,\vartheta_S + !A!\, t_C$$

is the mean cycle time (in number of CPU cycles) for comparing and copying an attribute of the source relation. Thus we obtain the mean number of CPU cycles to filter a character of a given relation:

$$\vartheta = \frac{1}{!A!}\left(\frac{!U!-C-P}{!U!}\,\vartheta_S + \frac{P}{!U!}\,(\vartheta_S + !A!\, t_P) + \frac{C}{!U!}\,(2\,\vartheta_S + !A!\, t_C)\right)$$

$$\vartheta = \left(1 + \frac{C}{!U!}\right)\frac{\vartheta_S}{!A!} + \frac{P}{!U!}\,t_P + \frac{C}{!U!}\,t_C$$

With $C = P$ we have $\frac{C}{!U!} = \frac{P}{!U!} = \frac{\sigma}{2}$ and

$$\vartheta = \frac{\vartheta_S}{!A!} + \frac{\sigma}{2}\left(\frac{\vartheta_S}{!A!} + t_P + t_C\right)$$

Finally we denote the filtering time of a block by T_F . T_F is a random variable which is supposed to be exponentially distributed and

$$E(T_F) = \vartheta * \tau * S_b$$

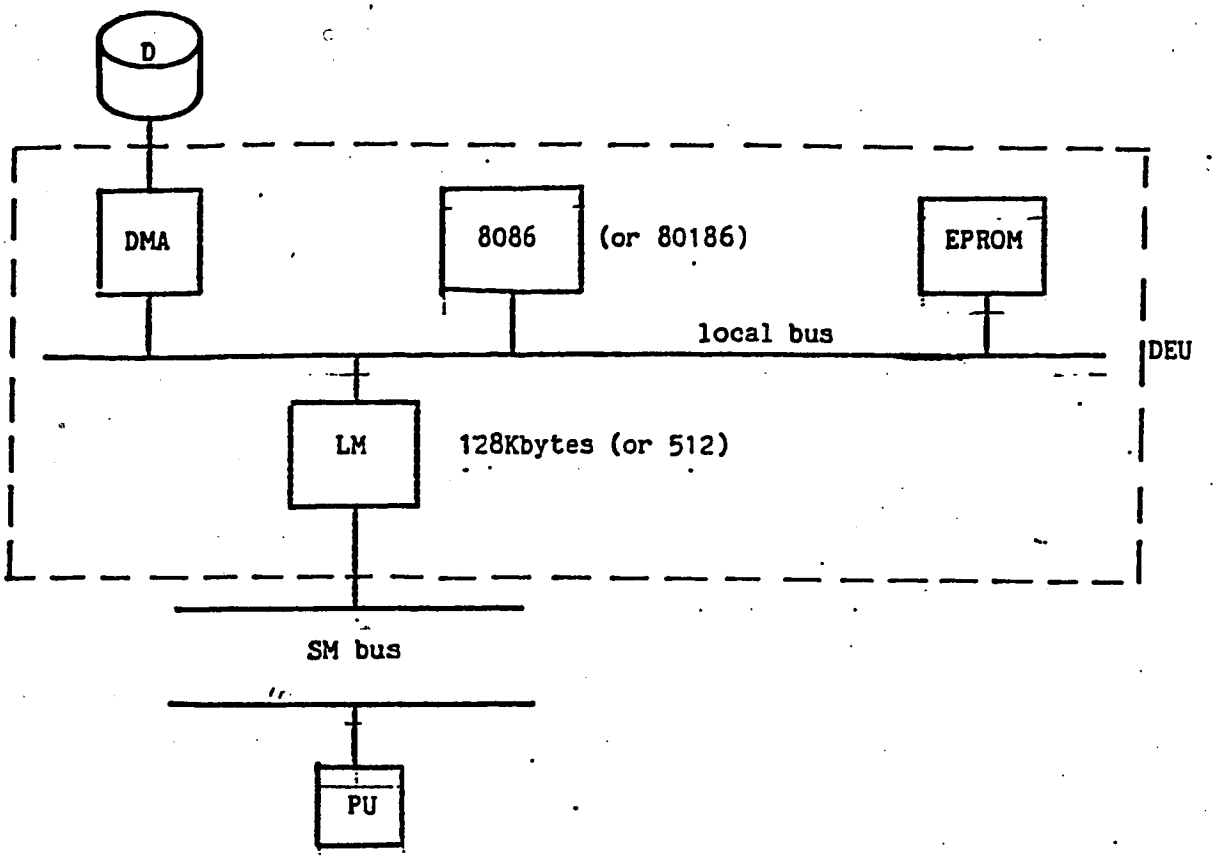


Fig. 1.1. - architecture

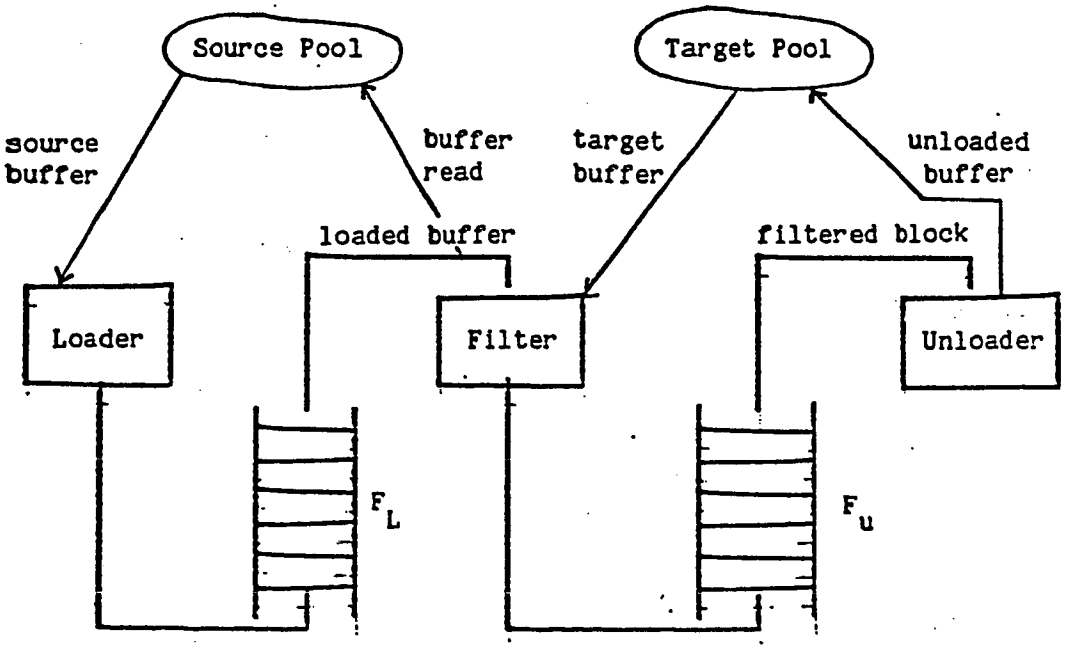


Fig. 1.2. - Filtering process sharing

in the same way, we suppose that the time for loading (unloading) a block (6 Kbytes) is a random variable exponentially distributed and its mean value is denoted by t_L . (t_U) .
Fig 1.3 gives the numerical values of the evaluation parameters.

	SASI	SMD
t_L	38.160 µs	4915,2 µs
t_{μ}	38.160 µs	4915,2 µs
τ	0.2 µs	0.125 µs
θ_s	120	120
t_f	17	17
t_c	22	22

Figure 1.3.

2. Performance evaluation of the Verso software filter

The aim of this section is the evaluation of the mean response time of the Verso software filter to a stream of queries.

When it has processed a user's query, the host processor (PU) sends to the DEU disk-addresses which correspond to blocks; these will be processed by means of relational operations performed by the filter. We are interested in the DEU performance only, so we do not take in account the time spent neither in the host, nor in the converse between the PU and the DEU. We also restrict ourselves to the selection-projection operation in this paper.

Input of the model is the offered load (in blocks) by the PU to the DEU. Outputs will be the mean response time, relative to a user's query and the throughput of the filter in blocks, for given loads.
The parameters of the loader are the nominal speeds of the processes and their synchronization constraints,but also the characteristics of the relational operation we consider.

We make the following hypothesis about the users' queries which are processed by the filter:

let σ be the selectivity ratio of the selection-projection,

let p be the probability that the selection clause for one attribute is positive. Each projected attribute is written with probability p,independantly of others.

let M be the number of blocks of the source relation.

Loading characteristics

We assume that M, σ , p are constant and identical for every query. The interarrival times between two queries (for the filter) are supposed to be exponentially distributed and the arrival rate is denoted by λ_a . The arrival process is a markovian homogeneous process with grouped customers, where all groups have M customers (blocks).

Parameters of the model

The three servers: loader, filter and unloader cooperate by communications through memory. The size of the source pool (target) is N_s (N_c) .

The service times of a block are supposed to be exponentially distributed:

- λ_L^{-1} is the mean loading time of a block from the disk,
- λ_F^{-1} is the mean filtering time of a block for the selection projection defined by σ , p , !A! and M,
- $\lambda_U^{-1}=\lambda_L^{-1}$ is the mean unloading time of a block from the target pool onto the disk.

We assume in this paper that the disk access conflicts between the loader and the unloader are negligible.

Events which occur in the system are :

- the arrival of a query from the host processor (M blocks),

- the end of a loading,
- the end of a filtering,
- the end of an unloading

} of a block

Dealing with blocks, we modelize the filtering architecture by the following queueing network:

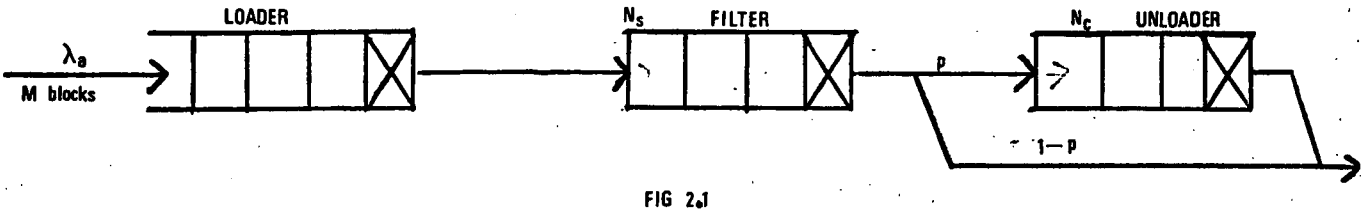


FIG 2.1

We define the mean response time to a selection- projection query as:

W = M * [Mean time spent in the loader subsystem by one block
+ Mean time spent in the filter subsystem by one block
+ p * Mean time spent in the unloader subsystem by one block]

2.1. Definition and resolution of the model

The definition of the performance criterion for a query implies that we can isolate every subsystem of the model in order to compute for each of them, its mean throughput and the mean time spent in it by one block of an user's query. Numerical solutions have been proposed to solve tandem queues but analytical results are known when there are no more than two tandem queues with finite capacity. Our model depends on the operation we consider and it will be necessary to compare results for various parameters of the selection projection (σ, !A! ,p),so the cost of the computations must be quite low.

We propose an approximate resolution which is based upon a hierarchical decomposition of the previous model (fig. 2.1.).

In the isolation method defined by Pujolle [Pujolle78], the interfaces between queues are solved with a not linear system. To avoid the computation of these interfaces in each case,we approximate the service rates to inject in the model so that each of the servers is regulated by the synchronization constraints. These service rates will be used in the model of fig. 2.1. instead of the nominal service rates λ_L , λ_F , λ_U . We calculate the probabilities that the filter queue or the unloader queue are full (inclusive or) with the hypothesis that the input queue (loader) is saturated. This result is given by a first auxiliary model, which is called capability model.

We then calculate the service rates which are to be taken in account in the queueing network in order to integrate the synchronization.

2.2. Capability model

We suppose that there is always one block at least in the loader queue. To calculate the probability that the filter queue and the unloader queue are full, we describe the system by the random vector $X(t) = (i(t), j(t))$ in each instant in time t where:

- i(t) is the number of blocks in the filter queue F_F (they have been loaded but not filtered yet),
- j(t) is the number of blocks in the unloader queue F_U.

X(t)$\in$[0, N_s] X [0, N_c] where N_s and N_c are the capacities of these queues respectively. We also assume that the evolution in the states space S obeys the competitive model [Howard75]. In every state, the evolution of the system depends on the event which occurs first. The events are the end of the loading,or the filtering, or the unloading of a block. Under these assumptions, X(t) is a markovian homogeneous recurrent positive process and we denote by $P^*_{i,j}$ the steady probability of state (i,j).

Let k and l be two states of S. The transitions between two states from $k = (i,j)$ obey the following equations.

$$\lambda_{kl}\, dt = Prob\ (X(t+dt) = l / X(t) = k)$$

with

$$\lambda_{kl} = \lambda_L \text{ if } l = (i+1,j) \quad i\in[0,N_s - 1],\ j\in[0,N_c]$$
$$\lambda_{kl} = \lambda_F \text{ if } l = (i-1,j) \quad i\in[1,N_s],\ j\in[0,N_c-1]$$
$$\lambda_{kl} = p * \lambda_F \text{ if } l = (i-1,j+1) \quad i\in[1,N_s]\ j\in[0,N_c-1]$$
$$\lambda_{kl} = \lambda_U \text{ if } l = (i,j-1) \quad i\in[0,N_s],\ j\in[1,N_c]$$

P^* is the row vector of the probabilities: $P^* = (P^*_{i,j})$ and $\boldsymbol{\Pi}$ the dynamic matrix (matrix of rates) of the process X(t). (See in annex the matrix and the equations of the process) From the solution of the linear system:

$$\begin{cases} P^* \boldsymbol{\Pi} = 0 \\ \sum_{i,j} P^*_{i,j} = 1 \end{cases} \qquad (1)$$

we get the different probabilities of states and especially the probabilities that one queue is full.

$\sum_{j=0}^{N_c-1} P^*_{N_s,j}$ is the probability that the filter queue only is full.

$\sum_{i=0}^{N_s-1} P^*_{i,N_c}$ is the probability that the unloader queue only is full.

$P^*_{N_s,N_c}$ is the probability that both queues are full.

2.3. Evaluation of the service rates of the queueing model

The problem is to calculate the service rates of the servers in the model above (fig. 2.1.), taking in account the synchronization delays between them. We denote by q(f,u) the probability of f customers in the filter queue and u customers in the unloader queue at time t.
We approximate q(f,u) by the corresponding steady states probabilities of the capability model,which is independant of the offered load.
Let Z_L be the random variable equal to the service time of the loader:
let A be the event that the filter queue is not full,whatever the unloader queue is;
let B be the event that the filter queue only is full;
let C be the event that both the filter and the unloader queues are full.

$$E(Z_L / A) = \lambda_L^{-1}$$

$$E(Z_L / B) = \lambda_L^{-1} + \lambda_F^{-1}$$

$$E(Z_L / C) = \lambda_L^{-1} + \lambda_F^{-1} + \lambda_U^{-1}$$

$P^*_{f,u}$ approximates the probability $q(f,u)$ after the last departure from the first queue (loader). The next customer of the loader is served if the system is in a state where $f < N_s$.
Thus, the synchronization delay between the servers, seen by the loader, is equal to the mean recurrence time from the state of the system after the last customer has left out into a state (k,l) where $k < N_s$. This time is calculated with the capability model. The same computation is performed to calculate the delay synchronization between the unloader and the filter. We obtain the approximate values of μ_L , μ_F , μ_U which are the service rates when the servers are synchronized:

$$\mu_L^{-1} = E(Z_L)$$

$$\mu_L^{-1} = \lambda_L^{-1} + \lambda_F^{-1} * \sum_{j=0}^{N_c} P^*_{N_s,j} + \lambda_U^{-1} * P^*_{N_s,N_c}$$

In the same way

$$\mu_F^{-1} = \lambda_F^{-1} + \lambda_U^{-1} * \sum_{i=0}^{N_s} P^*_{i,N_c}$$

and

$$\mu_U^{-1} = \lambda_U^{-1}$$

for there is no output constraint on the unloader service.

2.4. Resolution of the queueing network

The behavior of the software filter may be described now as a tandem configuration of three servers without any reject between successive queues. Since the synchronization delays have been integrated in the service rates of this model, we solve this model with a simplified method derived from the isolation method defined in [Pujolle 78].
The ergodicity condition for the whole system only depends on the loader queue

(see equation 3).

Each queue is isolated in order to be solved as if it was a single queue and we need to precise the interfaces between two successive queues. As the queues are regulated according to the synchronization conditions, the interface between two consecutive queues i and j is reduced to:

- the first and the second moments of the inter-departure time from station i,
- the first and the second moments of the interarrival time into station j,
- the probability that a block which leaves the station i enters the station j: pr_{ij}.

$$pr_{loader_{filter}} = 1$$

and

$$pr_{filter_{unloader}} = p$$

Each queue is solved as a single G/M/1 queue.

2.4.1. Resolution of the loader queue

Let $P_i^L(t)$ be the probability that i blocks are in the loader station at time t and P_i^L the steady state probability of i customers (blocks) in it,
W^L is the average time spent by a block in the loader station,
n^L is the mean number of blocks in the station at the equilibrium.

We recall that the arrival process in the system is a compound Poisson process of M blocks, with arrival rate λ_a . μ_L is the rate of service; the service times are supposed to be exponentially distributed. Therefore the equilibrium condition to satisfy is :

$$\frac{M * \lambda_a}{\mu_L} < 1 \qquad (3)$$

The equations of the limiting probabilities are then given by:

$$\lambda_a P_0^L = \mu_L P_1^L$$

$$(\lambda_a + \mu_L)\ P_1^L = \mu_L\ P_2^L$$

$$(\lambda_a + \mu_L)\ P_M^L = \mu_L\ P_{M+1}^L + \lambda_a\ P_0^L$$

$$(\lambda_a + \mu_L)\ P_{kM+i-1}^L = \mu_L\ P_{kM+i}^L + \lambda_a\ P_{(k-1)M+i-1}^L \qquad (4)$$

$$k \geq 1, 1 \leq i \leq M$$

$$P_0^L + \sum_{k=0}^{\infty} \left(\sum_{i=1}^{M} P_{kM+i}^L \right) = 1$$

$$\bar{n}^L = \sum_{n=0}^{\infty} n\ P_n^L$$

As the arrival rate of *blocks* is $M\ \lambda_a$ and the average time spent by each of them in the loader station is given by

$$W^L = \frac{\bar{n}^L}{M\ \lambda_a}$$

2.4.2. Interface Loader-Filter

Thus the departure process from the loader station is exactly the arrival process in the filter system. We approximate this process by the first and the second moments of the inter-departure time from the loader service

Let P_0^L be the probability that the loader station is empty,

S^L is the random variable equal to the service time in the loader system,

A^L is the random variable equal to the interarrival time to the loader station, Z is the random variable equal to the interarrival time to the filter queue.

$$Z = \begin{cases} S^L \text{ with probability } 1 - P_0^L \\ S^L + A^L \text{ with probability } P_0^L \end{cases}$$

Replacing A^L by λ_a^{-1} and S^L by μ_L^{-1}, the first and the second moments of Z are respectively:

$$E(Z) = \frac{1}{\mu_L} + \frac{P_0^L}{\lambda_a}$$

$$E(Z^2) = \frac{2}{\mu_L^2} + \left(\frac{2\ P_0^L}{\lambda_a}\right) * \left(\frac{1}{\mu_L} + \frac{1}{\lambda_a}\right)$$

We denote by $\lambda_{a_f} = (E(Z))^{-1}$ the arrival rate at the filter station and

$$K_{a_f} = \frac{E(Z^2)}{(E(Z))^2} - 1$$

the coefficient of variation of Z
These two parameters characterize the arrival process in the filter which is solved in the following.

2.4.3. Resolution of the filter queue

This G/M/1 queue is solved by a diffusion method, due to Gelenbe [Gelenbe75].
This method approximates the arrival and the service distributions by their first and second moments and the results are:

the mean response time W_F,
the mean number of customers in the system $\bar{n}_F$
the probability that the queue is empty P_0^F With

$$\lambda_{a_F} = (\mu_L^{-1} + P_0^L \lambda_a^{-1})^{-1}$$

and

$$K_{a_F} = \frac{2 * (\mu_L^{-2} + P_0^L * (\mu_L^{-1} \lambda_a^{-1} + \lambda_a^{-2})}{(\mu_L^{-1} + P_0^L \lambda_a^{-1})^2} - 1$$

The service is exponential with rate μ_F. We also use

$$\rho_F = \frac{\lambda_{a_F}}{\mu_F}$$

$$\tilde{\rho}_F = \exp\left(\frac{2(\lambda_{a_F} - \mu_F)}{(\lambda_{a_F} K_{a_F} + \mu_F K_{s_F})}\right)$$

where $K_{s_F} = 1$.

then we obtain the results:

$$\bar{n}^F = \frac{\rho_F}{1 - \rho_F}$$

the mean number of blocks in the filter station,

$$P_0^F = 1 - \rho_F$$

the probability that the station is empty, and

$$W^F = \frac{\bar{n}^F}{\lambda_{a_F}}$$

the mean residence time in the station for a block, using Little's formula.

2.4.4. Filter-unloader interface and resolution of the unloader queue

We applied the same method for the third subsystem (unloader). In this case, p is the probability that a block which leaves the filter queue, enters the unloader station.We characterize the arrival process in the station as above by:

$$\lambda_{a_U} = p\ (\mu_F^{-1} + P_0^F \lambda_{a_F}^{-1})$$

and

$$K_{a_U} = p * \frac{2(\mu_F^{-2} + P_0^F \lambda_{a_F} (\mu_F^{-1} + \lambda_{a_F}^{-1}))}{(\mu_F^{-1} + P_0^F \lambda_{a_F}^{-1})^2} - 1$$

The service time is exponentially distributed and the rate is $\mu_U = \lambda_U$, $K_{s_U} = 1$. In the same way, we obtain for the unloader queue:

$$\rho_U = \frac{\lambda_{aU}}{\mu_U}$$

$$\tilde{\rho}_U = \exp\left(2\frac{(\lambda_{aU} - \lambda_U)}{\lambda_{aU}\lambda_{aU} + \mu_U \lambda_{sU}}\right)$$

thus,
the mean number of blocks in the station is

$$\bar{n}_U = \frac{\rho_U}{1 - \rho_U}$$

and the mean residence time of a block in the station is

$$W_U = \frac{\bar{n}_U}{\lambda_{aU}}$$

3. Results and dicussion

We applied the method which is described above to an architecture with two blocks in the source pool and two blocks in the target pool. $N_s = N_c = 2$. Each block is 6K-bytes long. This architecture is currently running on an 8086 processor with an SASI DEU, and is being benchmarked [Bitton83].

In this paper, the performance of the software filter is measured by:

- T_f the mean filtering time for one character and for a given offered load; T_f is called the cycle time of the filter as for the hardware configuration [Scholl85].
- the mean maximal load which can be submitted to the filter from the host processor. This load is evaluated in K-bytes/s but corresponds to queries of M blocks. This measure is obtained with the ergodicity condition $\frac{\mu_L}{M \lambda_a} < 1$ where μ_L is the real service rate of the loader task, M the size (in blocks) of the query and λ_a the arrival rate of queries in the filtering system.

With $N_s = 2$ and $N_c = 2$, the capability model has only 9 states and we simply solved the linear system (1). The graph and the matrix are given in annex.
As seen in section 2, the service rates are given by equations (2); the first queue was solved with the recurrent system (4); all other results are analytically known so that the computation time is short: 43 s on a Vax-11/780 computer.

3.1. Mean response time of the software filter

We chose M=1000 for each query and the mean interarrival time between two queries is 100 seconds; we see in the next paragraph that this offered load is close to the maximal load for the SASI configuration. $\sigma = \frac{C+P}{!O!}$ where C+P is the number of the compared and projected attributes, and that p is the probability that the selection clause is true.

We first consider the mean response time of the SASI DEU (Fig.3.1). The cycle time increases with the size of the target relation. For instance, when the target relation is small ($\sigma = 0.05$) and !A! =20 (respectively 5), the cycle time is 2.3 μ s/car (respectively 6.7 μ s/car) but if $\sigma = 1$, it increases up to 7.3 μ s/car (respectively 13.6 μ s/car). On the other hand, the length of the attributes has a non negligible effect on the cycle time, especially for small values of σ. This is specific to the software filter. In the case of the hardware filter, the cycle time of such a back-end processor only depends on the size of the source relation [Scholl85], since it scans the whole source sequentially, one byte at a time. On the contrary, the software filter algorithm is optimized when an attribute must be skipped; the software filter does not scan the value of such an attribute but skips to the next attribute. The gain in cycle time is considerable and increases with the length of the skipped attribute. If every attribute must be scanned ($\sigma = 1$) ,the cycle time is twice, comparing with $\sigma = 0.05$. This result is corroborated by measures on the prototype where the gain is also 2 in the same conditions [Richard 85].

In the case of the SMD DEU, the previous comments remain valid (Fig.3.1). The only difference is the gain of speed due to the internal parameters of the DEU. However, the gain in filtering time is less than expected. This is due to the fact that in each case the filter is CPU bound. (note that, in fact, !A!=5

and $\sigma = 1$ is outside the ergodicity domain for SASI).

3.2. Mean maximal load

Figure 3.2 (respectively Fig.3.3) shows the mean maximal load in K-bytes/s for the SASI DEU (respectively SMD DEU) for the same queries as in the previous paragraph. An interesting point is the gain offered by the SMD unit. In the worst case the gain is a factor 2 ($\sigma = 1$) and in the best case it increases to 4.2 when $\sigma = 0.05$ and !A! = 20.
Thus, even if the gain in cycle time is not very important, the gain in load may justify the choice of an SMD DEU. Indeed, the software filter is intended to be used as a back end processor. It has to execute queries sent by several users and the maximal load is a very important criterion for the architecture designer.

As the filter is CPU bound, the maximal load is drastically reduced when the size of the target relation increases, because of the processors: the transfert rate is 7 times faster for the SMD DEU than for the SASI one, but the processor of the SMD configuration is only 1.7 times faster than the SASI one. Thus, the maximal load for the SMD does not exceed 4 times the maximal load for the SASI interface. This result points out the need for increasing the processor speed and optimizing the filtering algorithm.

4. Conclusion

This paper was devoted to the performance evaluation of the Verso software filter which is currently operational and more generally to specify the conditions of implementation which validate or invalidate the software filtering approach. We first exhibited probabilistic results on the response times for two kinds of software filters. To our knowledge, there is no such probabilistic evaluation for existing database machines.

We intend to compare these previsional results with field tests and measures. We have already obtained partial results about the existing prototype (SASI DEU) [Richard85]. They corroborate the determinant effect of logical parameters, such as the length of attributes and the selectivity ratio on the mean response time (see section 3.2) The main result of this model is the mean maximal load which can be applied to the filter; it could not be obtained from a deterministic study which does not take in account the load streams. The objective of the back-end data processing is the exploitation of the parallelism between high level functions on the host processor and data filtering on the filter processor unit. The maximal load is a determinant estimation for designing the whole architecture of the database machine.

To sum up, we realized a first model which is simple to use and to extend. We intend in a near future to investigate the following points:

- the case of compacted relations as used in the Verso machine [Bancilhon82],
- the case of binary operations,
- our model is limited to a single stream of identical queries but several kinds of requests must be taken in account in a same model,
- the logical parameters of the operations are determinant for the performance of the filter. They are, at the present, internal parameters of the model. We think that it is necessary to elaborate new models in order to

measure the quantitative influence of these parameters. In consequence, they should be input parameters and we should try to define feedback measures .

- the evaluation of the filter must be extended and integrated in a model for the whole DBMS system.

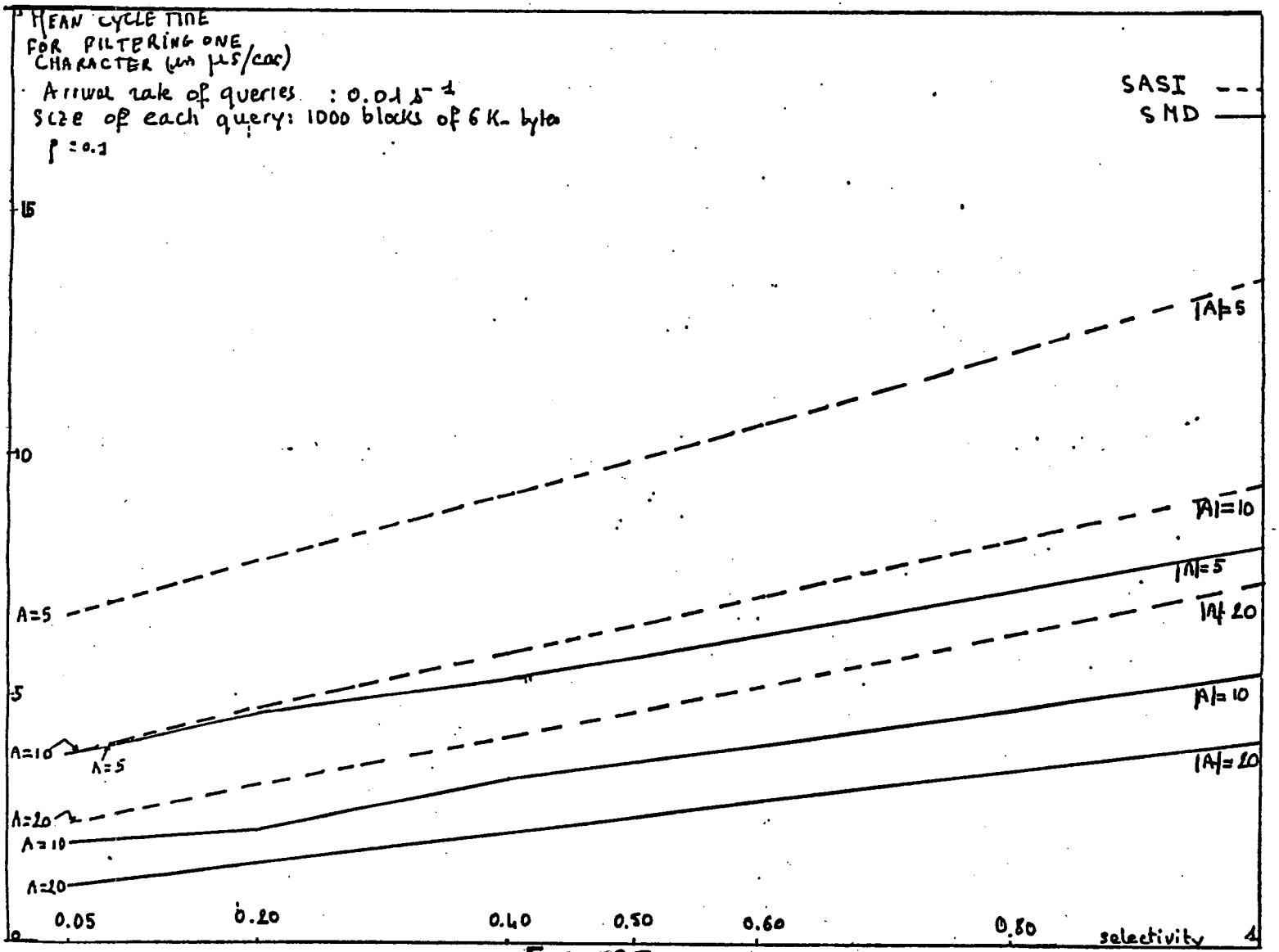

MEAN CYCLE TIME
FOR FILTERING ONE
CHARACTER (in μs/car)
Arrival rate of queries : 0.01 s⁻¹
Size of each query: 1000 blocks of 6 K-byte
p = 0.1
SASI
SMD
15
10
5
0
|A|=5
|A|=10
|A|=5
|A|=20
|A|=10
|A|=20
A=5
A=10
A=5
A=20
A=10
A=20
0.05
0.20
0.40
0.50
0.60
0.80
selectivity

Figure 3.1

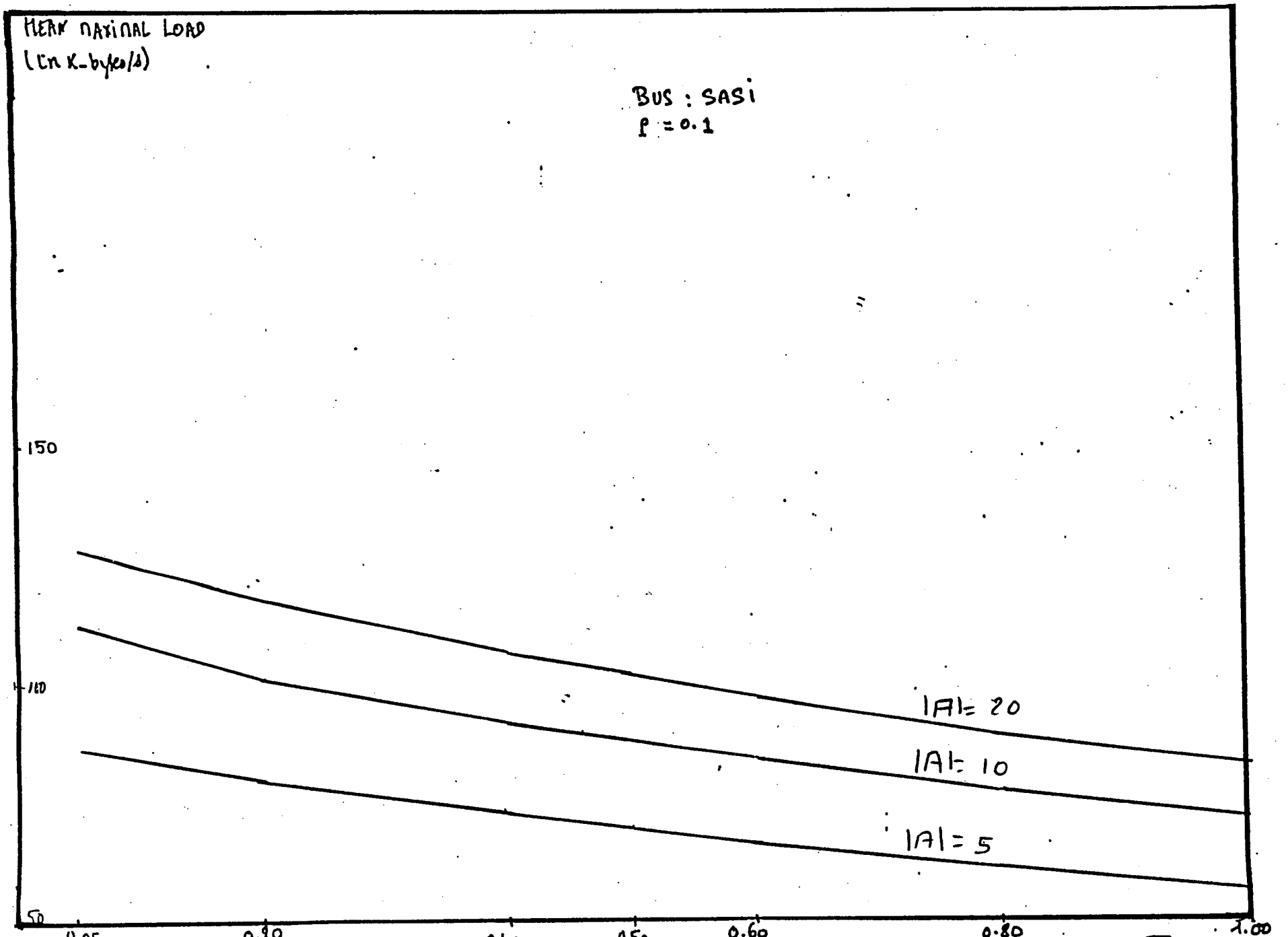
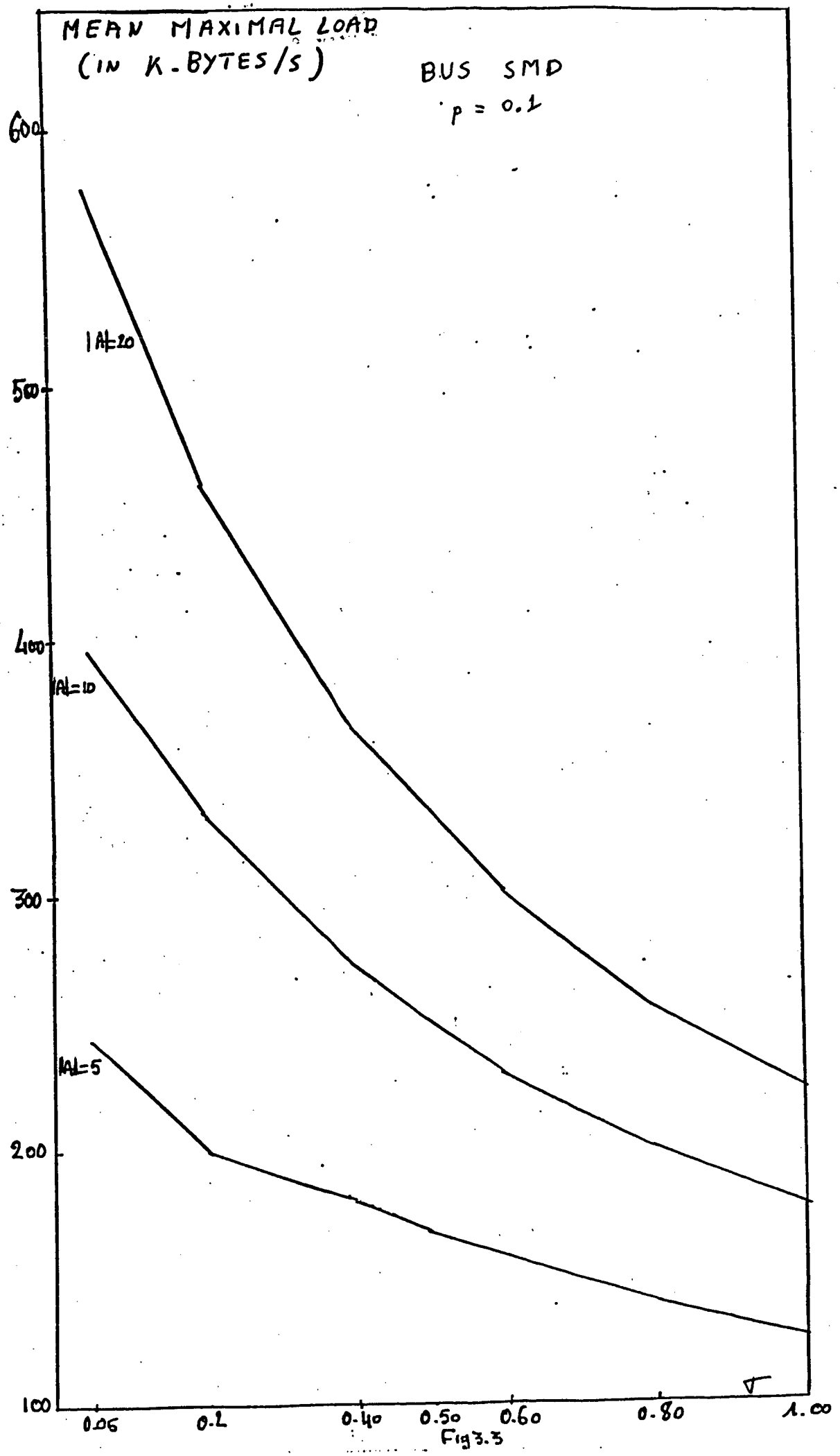

MEAN MAXIMAL LOAD
(in K-bytes/s)
BUS : SASI
ρ = 0.1
150
110
90
|A|= 20
|A|= 10
|A|= 5
0.05
0.20
0.40
0.50
0.60
0.80
1.00
r

Fig 3.2.


MEAN MAXIMAL LOAD
(IN K.BYTES/S)
BUS SMD
p = 0.1
600
500
400
300
200
100
IAL=20
IAL=10
IAL=5
0.05
0.2
0.40
0.50
0.60
0.80
1.00
Fig 3.3

ANNEX

Resolution of the capability model

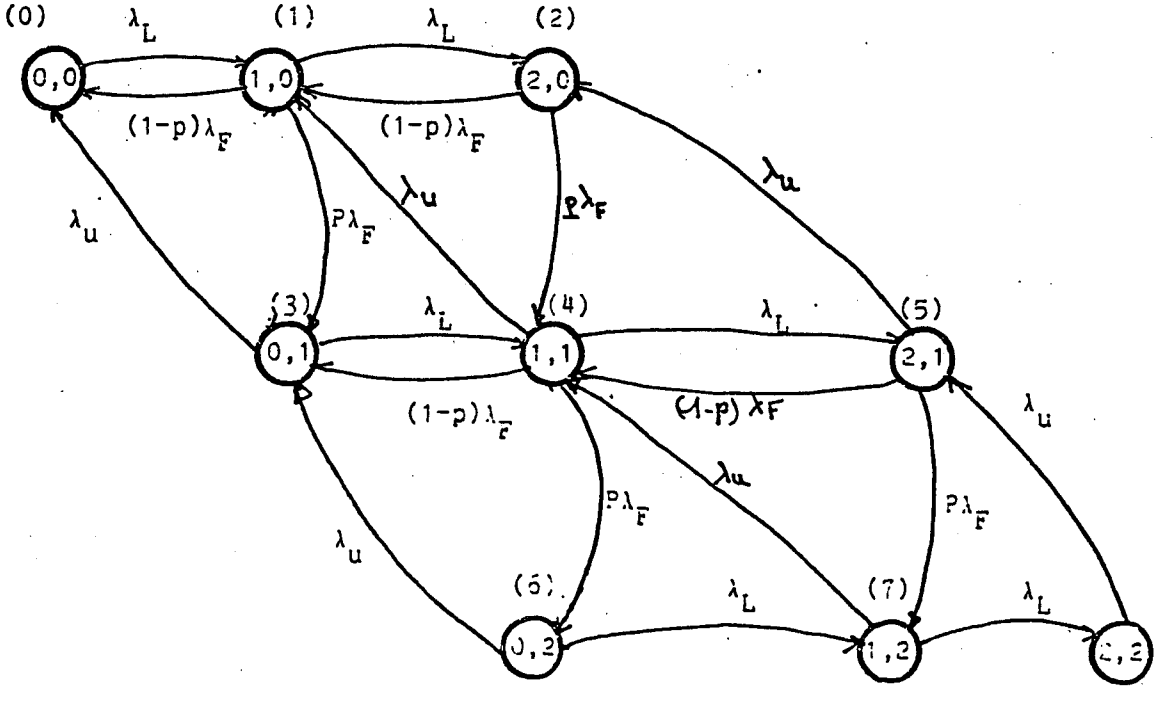
P^*_{ij} is the steady probability that is blocks are loaded and j blocks are filtered.

The equation of the steady state probabilities for the markovian process $X(t) = (i(t), j(t))$ are :

$$\begin{cases} \lambda_L P^*_{0,0} = \lambda_U P^*_{0,1} + \lambda_F P^*_{1,0} \\ \quad \cdots \\ (\lambda_F+\lambda_L)P^*_{i,k} = \lambda_u P^*_{i,k+1} + \lambda_L P^*_{i-1,k} + p\lambda_F P^*_{i+1,k-1} + (1-p)\lambda_F P_{i+1,k} \\ \qquad \forall\ (i,K) \in [1,N_s-1]x[1,N_c-1] \\ \qquad\qquad \vdots \\ P^*_{N_s,N_c} = \lambda_L P_{N_s-1,N_c} \end{cases}$$

with : $\sum_i \sum_j P^*_{i,j} = 1$

When $N_s = N_c = 2$ the graph which describes the behavior of X(t) is :



with $\lambda_L = \lambda_u$.

the matrix of the stochastic process X(t) is :

	0	1	2	3	4	5	6	7	8
0	$-\lambda_L$	λ_L							
1	$(1-p)\lambda_L$	$-\lambda_F$ $-\lambda_L$	λ_L	$p\lambda_F$					
2		$(1-p)\lambda_F$	$-\lambda_F$		$p\lambda_F$				
3	λ_L			$-2\lambda_L$	λ_L				
4		λ_L		$(1-p)\lambda_F$	$-2\lambda_L$ $-(\rho+1)\lambda_F$	λ_L	$p\lambda_F$		
5			λ_L		$(1-p)\lambda_F$	$-\lambda_L$ $-\lambda_K$		$p\lambda_F$	
6				λ_L			$-2\lambda_L$	λ_L	
7					λ_L			$-2\lambda_L$	λ_L
8						λ_L			$-\lambda_L$

References

[Armisen 81] Armisen, J.P. and Caleca, J/Y/ : A Commercial Back-End Database System. Proceedings of the 7th International Conference on VLDB, Cannes 1981.

[Babb 79] Babb, E. : Implementing a Relational Database by means of Specialized Hardware. ACM Trans. Database Systems, Vol. 4, n° 1, 1979.

[Bancilhon 80] Bancilhon, F. and Scholl, M. : Design of a Back End Processor for a Data Base Machine. Proceedings of ACM SIGMOD. Santa Monica, 1980.

[Bancilhon 82] Bancilhon, F., Richard, P. and Scholl, M. : "On line processing of compacted relations", Proc. Inter. Conf. on VLDB, Mexico, 1982.

[Bancilhon 83] Bancilhon, F. et al. : VERSO - A Relational Backend Database Machine. Advanced Database Machine Architecture. D.K. HSIAO, Editor, Prentice Hall, 1983. pp. 1-18.

[Bitton 83] Bitton, D. et al. : "Benchmarking Data Base Systems. A systematic Approach", Computer Science Department, Technical Report # 526, University of Wisconsin, December 1983.

[Boral 81] Boral, H., Dewitt, D. : Database Machines : An Idea whose time has passed ? A critique of the future of Database machine, research report, Univ. of Wisconsin.

[Cinlar 75] Cinlar, F. : Introduction to Stochastic Processes. Prentice Hall Editions.

[Codd 70] Codd, E.F. : A relational model for large shared data banks, Comm. ACM 13 : 6 (1970), pp. 377-387.

[Dewitt 79] Dewitt, D.J. : DIRECT - A Multiprocessor organization for supporting Relational Database Management Systems, IEEE Transaction on Computers, Vol. 28, N° 6, 1979.

[Gamerman 84] Gamerman, S. : Où l'on découvre que ls performances des filtres dans les Machines Bases de Données ne sont pas celles qu'on croyait. Thèse de 3e cycle, Université Paris XI, Orsay, 1984.

[Gamerman 85] Gamerman, S. & Scholl, M. : Hardware versus software data filtering, the Verso experience. 4th Int. Workshop on Database Machines IWDM 85, Bahamas, March 6-8, 85.

[Gelenbe 75] Gelenbe, E. : On approximate computer system Models. JACM 22, 2 pp. 261-269, 1975.

[Gelenbe 81] Gelenbe, E. & Pujolle, G. : Introduction aux réseaux de files d'attente. Editions Eyroles, Paris.

[Lemaire 78] Lemaire, B. : Contribution des systèmes avec attente : les méthodes de conservation. Application de problèmes de RO et d'informatique. Thèse d'Etat, Université de Paris VI, Juin 1978.

[Pujolle 78] Labetoulle, J., Pujolle, G. : Méthode d'isolation en Réseaux de files d'attente, modélisation et traitement numérique. AFCET. Editions Hommes et techniques.

[Richard 85] Richard, P., Scholl, M. : Filtrage des données dans les SGBD relationnels : l'expérience Verso". A paraître dans "Bases de Données. Nouvelles Perspectives". Ed. Eyrolles

[Scholl 85] Scholl, M. : Architecture matérielle et logicielle pour le filtrage dans le sBases de Données relationnelles. Thèse d'Etat, INPG, Grenoble, 1985.

[Schweppe 83] Schweppe, H et al. : RDBM - A Dedicated Multiprocessor Systems for Database Management, Advanced Database Machine Architecture, D.K. Hsiao Editor, Prentice Hall, pp. 36-86.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique