1986

On the Behavior of Programs with Remote Procedures

Dan C. Marinescu

Report Number:
86-636

ON THE BEHAVIOR OF PROGRAMS WITH REMOTE PROCEDURES

Dan C. Marinescu

CSD-TR-636
November 1986

ON THE BEHAVIOR OF PROGRAMS WITH REMOTE PROCEDURES

Dan C. Marinescu
Computer Science Department
Purdue University
West Lafayette, Indiana 47907

Summary

There are preciously few data concerning program execution in a distributed computing environment. The networking software has evolved considerably over the last few years and the availability of Remote Procedure Call (*RPC*) protocols as well as lower costs and higher sophistication and computing power of workstations, invites wide spread use of *RPC* by the casual application programmer. This paper discusses briefly *RPC* programming and reports measurements pertinent to *RPC* execution. A model of a program which invokes a mix of *RPC*'s is presented. Finally, measured data for different mixes of remote procedure types are analyzed using this model and program execution types are estimated for variable rate of *RPC*'s.

PROBLEM FORMULATION

Communication in a distributed system has been explored from different angles, e.g. from the point of view of operating systems, of the hardware architectures, of programming languages and data base systems, but rarely from the end user's perspective.

Operating systems provide now different levels of support for the communication to remote machines and it has become possible to write application software in a distributed fashion. The task of the application programmer is greatly simplified when a Remote Procedure Call (*RPC*) protocol is available, since the use of low level primitives like sockets is by no means trivial and it closely resembles performing I/O operation at physical level.

There are several reasons why it is desirable to perform computations remotely. First of all, the remote host may be capable of performing the computation faster than the local host, as it is the case in scientific and knowledge processing environments when using a highly parallel system or an image processing machine, a LISP machine etc. Database applications offer a variety of cases where remote computations are necessary, for example a query of a large remote data base. Remote operations are also executed implicitly, for example when running a program with a large

working set size on a diskless workstation. In all these cases, it is important to understand the implication of performing remote operations upon the actual execution time of the program.

An accurate model of a distributed system is much too complex for a tractable analysis and it leads generally to multi-server, multi-queue problems. Even approximate models lead to the analysis of networks of queues and usually the delay analysis of such a system requires at least the first two moments of the arrival and of the service process at each node. The experiments necessary to gather such data are difficult and this explains why such data are seldom available.

We present an unsophisticated approximate analysis based on simple measurements. This analysis provides good guidelines for the design of a program performing remote computations. We examine several questions of interest from the application programmer's view point, i.e. how often can he afford to perform an expensive Remote Procedure (*RP*), or when the remote host can perform the computation faster, what is the speeding up factor, and, in general, what is the execution time of a program with a certain mix of remote procedures.

There are cases also when the remote services are provided implicitly by the operating system, as in the case of paging. Then the performance degradation can be significant if the user is not aware of the remote server's location.

In all cases it is important to understand the implications of: the server's location, its load, and the communication delay when remote computations are performed. As expected, the access through a long haul network leads to longer delays. As a general rule tuning of a distributed application is a rather challenging task and it requires a certain level of understanding of the networking environment.

The paper presents briefly techniques for *RP* programming and then *RP* measurements are discussed. In the next sections, a trivial model of program execution is examined, numerical results for different mixes of *RP* are presented, and an analysis of program execution time is carried out.

PROGRAMMING WITH REMOTE PROCEDURES

A good analysis of the remote procedure call mechanism, can be found in Spector [2]. Essentially, in a client-server relationship, the client process passes the flow of control to the server in a request to perform a certain action and blocks until the server process located on a remote machine completes its execution and sends back the result. To hide the details of transporting the data from one location to another, and the problem of addressing in a networking environment, Berkeley Unix (TM) has introduced the concept of a socket as an endpoint of communication as seen from the user's perspective. Sockets are manipulated like file descriptors. Due to the asymmetry of the client-server relationship, a client process can request a connection with a server which can accept or not the connection request.

Two transport mechanisms are available: one based on the unreliable but more efficient transport protocol UDP, (User Datagram Protocol), and one on the reliable but less efficient, TCP (Transport Control Protocol). The transport mechanism has to be specified at socket creation time. Obviously both client and server need to use the same transport mechanism.

To design a distributed application, one has to create the server process on the remote host and obtain its unique identification, given by the port number and network address of the host where it runs. An *RPC* protocol is then a rule concerning the format of the data packets, so that each time the client process establishes a connection with one of its servers, the server is capable to understand the semantics of the request. This was essentially the procedure followed in order to perform the measurements reported in the next section.

This brief presentation of the low level communication primitives available in Berkeley Unix (TM) and their use in order to perform remote operations, deliberately skipped over a number of details concerning: the problem of handling different data types, especially on machines with different architectures, the problem of embedding the error control and flow control mechanisms in the *RPC* protocol, especially when using an unreliable transport protocol as

UDP, and last but not least, a set of details related to the techniques of communicating through sockets, as socket binding, the use of network utility functions to obtain the port number associated with a given service or to obtain the network address of a host, etc.

In order to facilitate the development of system software and to provide a friendlier access to the networking functions, Remote Procedure Call protocols have emerged lately. For example SUN provides an *RPC* protocol [3], with three levels of interfaces. At the first level, all the details of communication are hidden from a user who has access to a library of *RP*'s available system-wide.

A second level provides two primitive operations, one to define new *RP*'s and to register them for system-wide use, *registerpc* and a second one, *callrpc* to actually use them. To register a procedure, the server must provide the service identification (program number, version number and procedure number), a name for the procedure as well as the types of its input and output. A client can call this remote procedure by invoking the *callrpc* primitive giving as arguments the name of the *RP*, the service identification, the type and a pointer to the input parameters and a type and pointer for the result. At the lowest level, the *RPC* protocol allows a greater flexibility. One may use TCP rather than UDP, may perform authentification at either the client or the server site or may use its own memory allocation schemes.

MEASUREMENTS

The measurements reported here were carried out in order to determine the parameters necessary for a model of the execution of a program with remote procedures. A very simple *RPC* protocol based upon UDP was designed. Several server processes were activated on different machines located at different "distances" from the host where a client process was running. A server running on the same machine as the client is said to be at distance 0, one in the same Ethernet at distance 1, and one in another local subnet, connected through a gateway, at distance 2.

The client sends a packet to the remote host where an *RPC* server identifies the type of the remote procedure requested by the client, extracts the input parameters, and branches to the proper procedure on the remote host. Upon completion of the computation, the results are sent to the *RPC* server which packs them together and sends them back to the client.

We have defined eight types of remote procedures according to

- the size of input (arguments) to the remote procedure,
- the amount of computations performed remotely,
- the size of output (returned values) produced as a result of the remote procedure execution.

Each of these parameters could be s (*small*) or l (*large*). For example, an <s,l,l,> type means *small* input, *large* computation and *large* output. The observation of the traffic in the local network (Ethernet) of our department has shown a bimodal distribution of the packet length. About 50% of the packets carried by the network have a length in the 64–241 bytes range and the remaining 50% have length in the 970–1150 byte range. In our measurements *small* packets (input or output to/from the remote procedure) contain less that 100 bytes while *large* packets contain 1 Kbyte of data. A *small* computation requires the execution of a few dozens of instructions while a *large* computation was defined as an empty loop executed 100,000 times.

At the time when the measurements were carried out, the 10 Mbps Ethernet was connecting several VAX 780 and 785, one file server, less than a dozen SUN workstations, a multiprocessor system, FLEX 32, graphic workstations and several other processors, all running Berkeley UNIX (TM) 4.2 BSD. The traffic load through the Ethernet was relatively low, it rarely exceeded 5% of the channel capacity, and the file server was responsible for 10–30% of the traffic. Since then, a VAX 8600 as well as several new SUNs and a new file server, have been added and the traffic load is now in the 10% range of the channel capacity. The client was running on a VAX 785 with

a low load, and several servers were active on

(A) The client machine (VAX 785).

(B) A VAX 780 connected to the Ethernet and acting as a gateway.

(C) A dual VAX connected to the gateway through a 10 Mbps token passing ring (PRONET).

(D) A SEQUENT BALANCE 8000 connected to the INTERNET and located in California.

The results of our measurements are reported in Table 1.

Remote Procedure Type	95% Confidence Interval for the Response Time (msec) for different server locations			
	VAS 785 *Machine A* (client's is location)	VAX 780 *Machine B* (acts also as a gateway)	Dual VAX 780 *Machine C* (connected through the gateway)	Sequent Balance 800 *Machine D* (long haul network)
s s s	(11,11)	(24,24)	(26,28)	(866,898)
s s l	(12,12)	(48,48)	(60,62)	(1911,1971)
l s s	(19,21)	(42,44)	(54,56)	(2185,2237)
l s l	(22,24)	(68,70)	(86,88)	(3174,3216)
s l s	(292,298)	(523,535)	(390,392)	(1426,1454)
s l l	(341,351)	(663,689)	(425,429)	(2532,2592)
l l s	(353,365)	(793,837)	(418,422)	(2728,2766)
l l l	(375,389)	(933,989)	(458,464)	(3816,3876)

Table 1. Measured Response Time (in msec) for different types of Remote Procedures on several hosts.

The error rates were low for the local machines and considerably higher for the long haul networks. The results reported here are based upon a large number of measurement points, 4,000 for machines A and B, 2,000 for C and 1,000 for D. The measurements were carried out over a period of one month at random points in time, at various loads of the server machines and at

various levels of traffic.

The results reported are affected by: errors determined by the limited accuracy of timing measurements in Berkeley Unix (TM), which are in the 10% range, by the variations in the server machine load, by the dispersion of communication delays determined by variations in the network traffic. For these reason, rather than presenting average values for the response time, Table 1 shows the 95% confidence intervals for the measured data. We notice that:

(a) The *RP*'s with *small* computations on local machines (B and C), have a small 95% confidence interval of the response time. The variation in the traffic load is primarily responsible for the slight variation in the response time observed for each machine.

(b) The response time of *RP*'s with *large* computations on local machines (B and C) have a slightly larger dispersion of the response time and this can confidently be attributed to the variations in the load of the server machines. The faster machine C, has a smaller 95% confidence than the slower, but closer machine, B.

(c) In case of the remote machine, D, the variations on the communication delay through a long haul network are the dominant factors as we conclude by comparing the 95% confidence interval of the related *RP*, one with *small* computation and one with *large* computation (e.g. <s s l> and <s l l>).

A QUEUEING MODEL FOR A PROGRAM WITH REMOTE PROCEDURE CALLS

Queueing models for program behavior have been investigated in the past, see for example Ramamoorthy [4] and Spirn [5]. The basic idea is to construct the program flow graph and to treat it as the state transition diagrams of a Markov chain. The study of program behavior is then reduced to the investigation of a homogeneous, finite-state, discrete-parameter Markov chain with an absorbing state. Clearly, the homogeneity assumption leads to an approximate analysis since in a given state, the future program behavior may depend upon the past history. Since there are a

finite number of statements, hence, a finite number of execution regions, the Markov chain is finite.

The absorbing state corresponds to the program termination. If s_1 to s_{n-1} are the transient states and s_n is the absorbing state of a Markov chain with n states, then the transition probability matrix of the chain can be partitioned into a sub-stochastic $(n-1)$ by $(n-1)$ matrix Q describing the transitions only among the transient states, and two more vectors, as follows:

$$P = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{C} \\ \hline \mathbf{O} & 1 \end{array}\right]$$

Here **C** is a column vector and **O** a row vector of $n-1$ zeros. The fundamental matrix defined as:

$$M = (I - Q)^{-1}$$

always exists and has the following property (see for example Trivedi, [6]):

$$m_{ij} = E[v_{ij}].$$

The (i,j)-th element of M, m_{ij}, is the average value of the random variable v_{ij} with $i,j < n$. Here, v_{ij} is the number of times the program visits state s_j before entering the absorbing state, given that it started in state s_i. In order to model the execution of a program which invokes remote procedures, we construct the program flow graph of Figure 1 with one state, s_0, corresponding to execution on the local region, n states $s_1, \ldots, s_n$, each corresponding to the execution of one remote procedure and an absorbing state.

In this graph p_i is the probability of executing RP_i after an execution on the local machine. The model does not allow for recursive RP's. The transition matrix among transient states is:

$$Q = \begin{bmatrix} 0 & p_1 & p_2 & p_n \\ 1 & 0 & 0 & 0 \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The fundamental matrix is:

$$M = \begin{bmatrix} 1/p_0 & p_1/p_0 & p_2/p_0 & p_n/p_0 \\ 1/p_0 & 1+p_1/p_0 & p_2/p_0 & p_n/p_0 \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ 1/p_0 & p_1/p_0 & p_2/p_0 & 1+p_n/p_0 \end{bmatrix}.$$

The average number of execution of RP_j is:

$$v_j = \frac{p_j}{p_0} \quad j = 1, 2, \ldots, n.$$

The average number of passages through the local execution is:

$$v_0 = \frac{1}{p_0}.$$

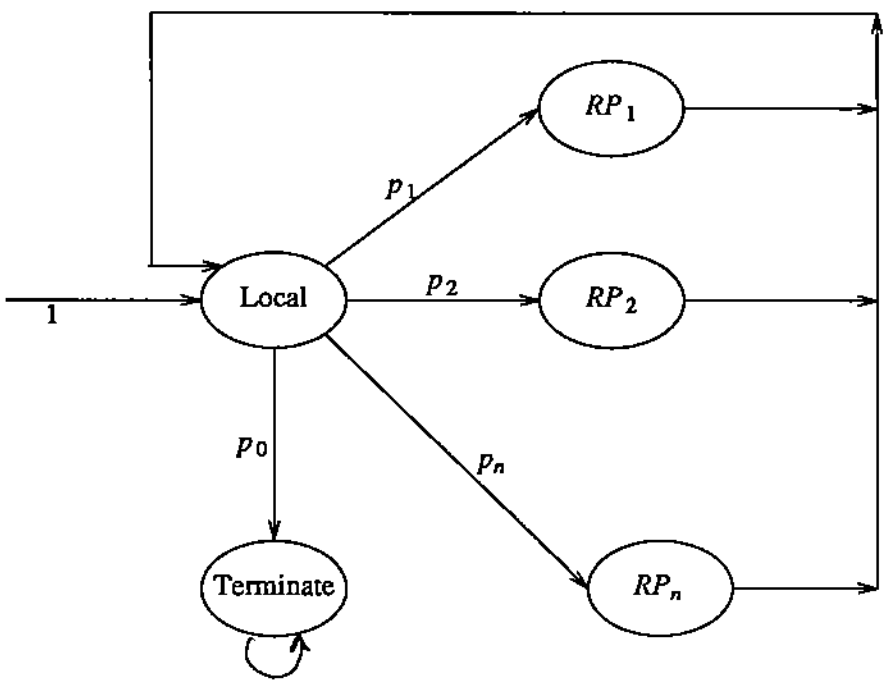

Figure 1. The program flow graph

The total execution time of the program is then:

$$T = \sum_{j=0}^{n} v_j \, t_j = v_0 \, t_0 + \sum_{j=1}^{n} v_j \, t_j$$

with t_j the execution time in the corresponding state. The model assumes that the execution time of remote procedure RP_j is exponentially distributed with average t_j.

EXPERIMENTAL RESULTS

We now apply the model described in the previous section and use the measurements presented earlier to estimate the execution time of programs with remote procedures. One can easily associate with each of the eight types of *RPC*'s defined with an actual remote operation. For example, the <s,s,s> type corresponds to operations performed in handling remote switches, used by voting algorithms or performed by remote commands like *cd*. The <s,l,l> may correspond to a query operation. Similarly, <s,l,l> or <l,l,l> could be related to paging over the network and <l,l,l> to remote execution of computationally intensive tasks, for example the inversion of a matrix, etc.

The objective of our analysis is to compute γ, the ratio between the execution time with remote procedure calls and the equivalent execution time where the remote procedures were executed locally, as function of the rate at which remote procedures are executed. The following notation is used:

T_R - expected execution time with *RP*'s

T_L - expected equivalent execution time with all computation performed locally

γ - T_R/T_L

n - the number of *RP* types

p_j - $1 \le j \le n$, the probability of executing RP_j

t_j - average execution time corresponding to RP_j

t_0 - average execution time in one cycle on the local machine

t_e - average execution time for the equivalent system where all *RP*'s are executed locally

v_j - number of executions of RP_j (visit count for RP_j)

v_0 - number of visits to the local execution region

A number of cases will be analyzed. First of all we examine the average case when the program cycles through all types of *RP*'s, a second example is concerned with the case when only *RP*'s with small execution time are invoked. Then an approximate model for paging over the network is analyzed, and finally a shell script is modeled. The section is concluded with a discussion of the results.

Average Case Analysis for a Uniform Mix of *RP*'s

The important assumptions for this case is that all *RP*'s in our generic group, <s,s,s> , . . . , <l,l,l> are executed with equal probability. In this case $n = 8$.

A simple computation indicates that in this case

$$v_j = \frac{v_0 - 1}{n}$$

Hence

$$T_R = v_0\, t_0 + \frac{v_0 - 1}{n} \sum_{i=1}^{n} t_i$$

or

$$T_R = v_0\, t_0 + (v_0 - 1)t_n$$

with

$$t_n = \frac{\sum_{i=1}^{n} t_i}{n}.$$

For large v_0, T_R can be approximated by:

$$T_R = v_0(t_0 + t_n).$$

According to previous notations, we have:

$$T_L = v_0(t_0 + t_e).$$

Hence

$$\gamma = \frac{T_R}{T_L} = \frac{v_0(t_0 + t_n)}{v_0(t_0 + t_e)} = \frac{t_0 + t_n}{t_0 + t_e}.$$

To compute t_e we assume that on average, we have an equal number of *small* and *large* remote computations. The *equivalent* local execution time of a large remote computation is obtained as the difference between the average time of an <s,l,s> and of an <s,s,s> computations performed on the local machine A. The *equivalent* local short computation time is very small and assumed zero. Hence, we use for t_e the value 142 msec (see Table 1).

The following observation is valid for this example and for all the following ones. Since our model assumes that each local execution is followed by a remote procedure call and t_0 is the time for local execution in one cycle, then $\frac{1}{t_0 + t_n}$ is the rate, in *RPC*'s per msec, at which remote procedure calls are executed.

The values of γ are summarized in Table 2 for t_0 in the range 10^{-1} to 10^4 msec.

	t_0 (in msec) / t_n	10^{-1}	1	10	10^2	10^3	10^4
A	181.00	1.27	1.27	1.25	1.16	1.03	1.003
B	396.25	2.79	2.77	2.67	2.05	1.22	1.025
C	241.25	1.69	1.69	1.65	1.41	1.08	1.009
D	2353.00	16.55	16.46	15.54	10.13	2.93	1.218

Table 2. The gamma factor for a homogeneous mix

Figure 2 presents the gamma factor function of the RPC rate for machines B and C. In case of the

long haul network access (machine D) the increase in execution time is significant, the gamma factor is 16.55 for 0.42 RPC/sec, decreses to 2.93 for 0.29 RPC/sec and is 1.218 for 0.08 RPC/sec.

The Small Computation Case

In this case we consider only the mix consisting of the four types of *RP*'s performing *small* computations remotely (<s,s,s>, <l,s,s>, <s,s,l> and <l,s,l>). Hence $t_e = 0$ and γ is given by:

$$\gamma = \frac{t_0 + t_n}{t_0} = 1 + \frac{t_n}{t_0}$$

The values of γ are summarized in Table 3. The value of t_n is computed considering an uniform mix of *small RPC*'s where all types occur with the same probability.

t_0 (in msec) / t_n	10^{-1}	1	10	10^2	10^3	10^4
A	16.5	166	17.5	2.65	1.16	1.016	1.0016
B	46.0	461	47.1	5.60	1.46	1.046	1.0045
C	57.5	576	58.5	6.75	1.57	1.057	1.0057
D	2057	20571	2058	206.70	21.57	3.057	1.0257

Table 3. The gamma factor for small computations

In case of local network we observe that the speed of the remote machine is no longer the dominant element in determining the gamma factor. In particular host B, which is a slower machine, but located only one hop from A looks better than C, a faster machine, but located two hops away.

Paging Over the Network

Paging is an implicit remote operation associated with execution of large programs on diskless workstations. The paging rate depends upon the ratio between the available memory and the program's working set size.

In our model paging is assimilated with a mix of <s,l,l> and <l,l,l> operations. If the page being replaced has not been modified, paging is modeled as an <s,l,l> operation while if the page has been updated, paging is modeled as an <l,l,l> operation.

The gamma factor is given by:

$$\gamma = \frac{t_0 + t_{rp}}{t_0 + t_{lp}}$$

The paging time on the local machine, t_{lp} will be assumed fixed and a reasonable value for it is 25 msec. For each machine, the remote paging time is approximated by:

$$t_{rp} = 0.5\ t_{<s,l,l>} + 0.5\ t_{<l,l,l>}$$

The resulting values are summarized in Table 4.

	t_0 (msec) / t_{rp}	10^{-1}	1	10	10^2	10^3	10^4
A	364	14.56	14.03	10.68	3.71	1.33	1.03
B	821	32.71	31.61	23.74	7.36	1.77	1.07
C	444	17.76	17.11	12.97	4.35	1.40	1.04
D	3204	127.65	123.26	91.82	26.43	4.10	1.31

Table 4. The gamma factor for paging

Figure 3 shows the gamma factor for machines B and C in case of paging. For machine D, the performance degradation is sensible even for low paging rates: 127.65 for 0.31 pages/sec, 26.43 for 0.30 pages/sec, decreses to 4.10 for 0.23 pages/sec and reaches 1.31 for for 0.075 pages/sec.

A Shell Script

As a last case, we consider the execution of a shell script on a diskless workstation. This is an example of the application of the model described earlier to a case when *RP*'s are executed with different frequencies. As a basis for our estimation of frequency of use of different *RP*'s we use the results reported by Sheltzer and Popek for the most frequently used interactive commands

on the collection of LOCUS system at UCLA [1].

Command	Type	Frequency of Execution (%)
ls	<s,l,l>	11.9
vi	<s,l,l>	8.7
cd	<s,s,s>	8.5
more	<s,l,l>	7.5
rm	<s,l,s>	3.3
dirs	<s,l,s>	2.6
jobs	<s,s,s>	2.6
fg	<s,s,s>	3.2
make	<s,l,s>	2.2
grep	<s,l,s>	2.1
finger	<s,l,s>	1.9
cp	<s,l,s>	1.3

Table 5. The frequency of execution of commands in LOCUS

Examining Table 5, we observe that by considering only two types of computations in modeling a remote procedure, *small* and *large* we are forced to some gross approximations. Probably more than two quantization levels for the remote computations are necessary to have a more accurate model. For example a *make* should be modeled as a very large computation.

Assuming that the rest of remote computations belong to the <s,l,l> class, we use in our model the data summarized in Table 6.

Type	Frequency (%)
<s,s,s>	13.3
<s,l,s>	13.4
<s,l,l>	73.2

Table 6. The frequency of execution of *RP*'s in a shell script

In this case t_e is given by:

$$t_e = 0.133\ t_{e,<s,s,s>} + 0.13\ t_{e,<s,l,s>} + 0.732\ t_{e,<s,l,l>} = 0 + 0.866 \times 284 = 246\ msec$$

t_n will be calculated for each machine according to the expression:

$$t_n = 0.133\ t_{<s,s,s>} + 0.134\ t_{s,l,s>} + 0.732\ t_{<s,l,l>}$$

The results are summarized in the following table.

	t_0 (msec) / t_n	10^{-1}	1	10	10^2	10^3	10^4
A	294.26	1.19	1.19	1.18	1.13	1.03	1.00
B	568.91	2.31	2.30	2.26	1.93	1.25	1.03
C	368.54	1.49	1.49	1.47	1.35	1.09	1.01
D	2185.65	8.88	8.85	8.57	6.60	2.55	1.18

Table 7. The gamma factor for a shell script

Figure 4 shows the gamma factor for machines B and C. In case of D, a machine accesible through a long haul network the gamma factor is: 8.88 for 0.457 RPC/sec, 6.6 for 0.437 RPC/sec, 2.55 for 0.313 RPC/sec and finally 1.18 for 0.082 RPC/sec.

Discussion of the Results

The approximations presented earlier assume a number of execution cycles much larger than one. An execution cycle consists of an execution on the local machine and one remote execution.

The results presented in these sections are somehow optimistic for several reasons. First, the *RPC* protocol used was very simple. The measurements were performed only in case of a successful *RPC*. While a more sophisticated *RPC* protocol would retry in case of a failure and thus the response time would increase, in our measurements the unsuccessful case was simply discharged. Using more sophisticated *RPC* protocols like SUN's the results would probably be more conservative especially if the TCP rather than UDP protocol is used. The TCP protocol

must be used when more than 8 Kbytes of data are transmitted as parameters.

Second, the approximations used to compute the equivalent execution time of a remote procedure on a local machine was conservative.

Third, during our measurements on the local network the actual load on both the network and remote servers was low.

As a general observation, on local machines, *RPC*'s executed at a rate of 1 or less per second lead to acceptable degradation of the computing speed. Paging at a rate higher than one page/second can be very expensive.

In case of a long haul network access, paging over the network becomes unthinkable for rates larger than one page every 4 seconds. Remote command execution is reasonable for rates not larger than one command every three seconds (which is realistic).

CONCLUSIONS

The results presented have an orientative value, they are intended to illustrate the methodology to study the program behavior in a distributed system. The approximations made in constructing the model of the program behavior and especially the measured data used as parameters of the model, affect the results as discussed earlier.

A more accurate model needs a set of *RP* types larger than the one considered here. A qualitative improvement of the model would be to allow for parallel *RPC*'s and for non-blocking ones. In case of a non-blocking *RPC* the client does not block waiting for the result but continues its execution flow and is interrupted when the results become available, like an asynchronous I/O operation.

The speed of the remote hosts, their actual load, the communication overhead, all are significant factors in determining the actual performance of the program.

It would be extremely interesting to repeat the procedure discussed here in a network with 50 or more workstations and at least five file servers, especially in a scientific environment, with large CPU bound-programs and an assortment of parallel machines. In such an environment, one could probably observe speeding-up factors due to execution of intensive computations on fast, remote machines.

ACKNOWLEDGMENTS

The author expresses his thanks to Andrew Royappa, who has carried out the measurements.

LITERATURE

[1] **A.B. Sheltzer and G.J. Popek,** *Internet LOCUS: Extending Transparency to an Internet Environment,* IEEE Trans. on Soft. Eng., Vol. SE–12, (11) 1067–1076 (1986).

[2] **A.Z. Spector,** *Performing Remote Operations Efficiently on a Local Computer Network,* CACM 25, (4), 246–259 (1982).

[3] ******,** SUN MICROSYSTEMS, *Networking on the SUN Workstation Reference Manual,* (1986).

[4] **C.V. Ramamoorthy,** *Discrete Markov Analysis of Computer Programs,* Proc. ACM National Conference, 286–392 (1965).

[5] **J.R. Spirn,** *Program Behavior: Models and Measurements,* Elsevier, New York (1977).

[6] **K.S. Trivedi,** *Probability and Statistics with Reliability, Queueing and Computer Applications,* Prentice Hall (1982).

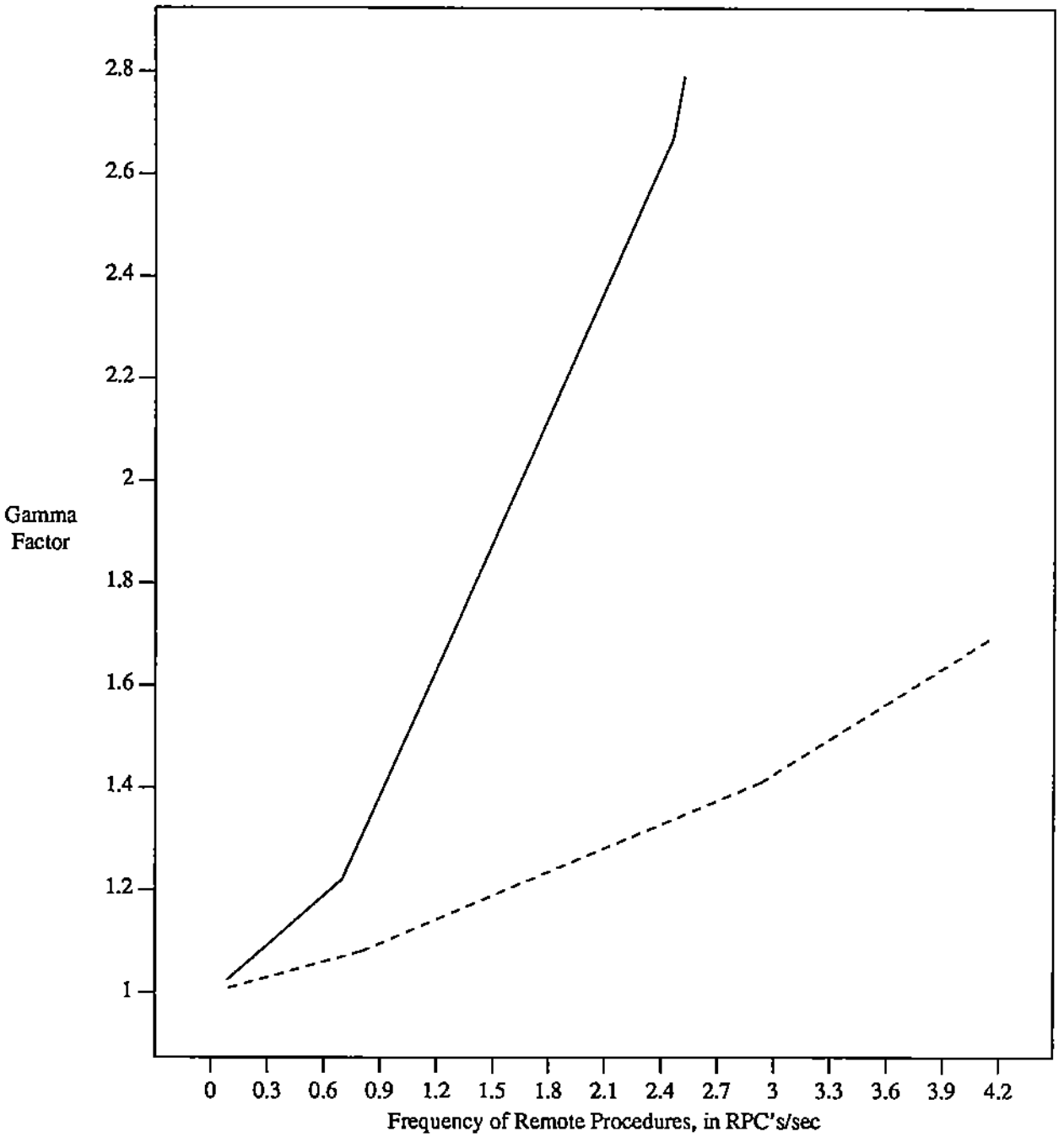

Figure 2. The gamma factor for machines B (solid) and C(dashed) function of RPC rate (homogenous mix case)

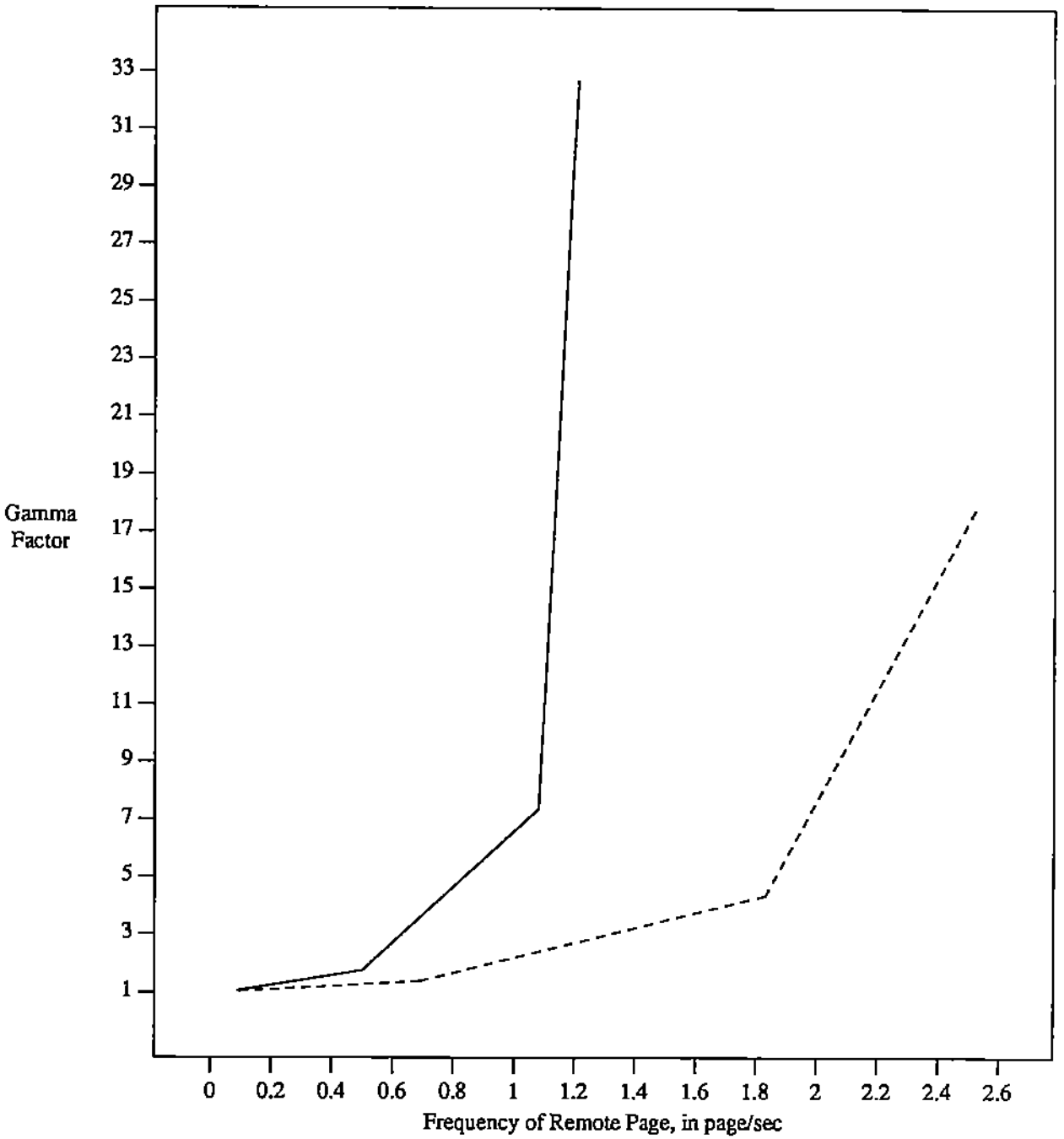

Figure 3 The gamma factor for machines B (solid) and C(dashed) function of the paging rate in pages/sec

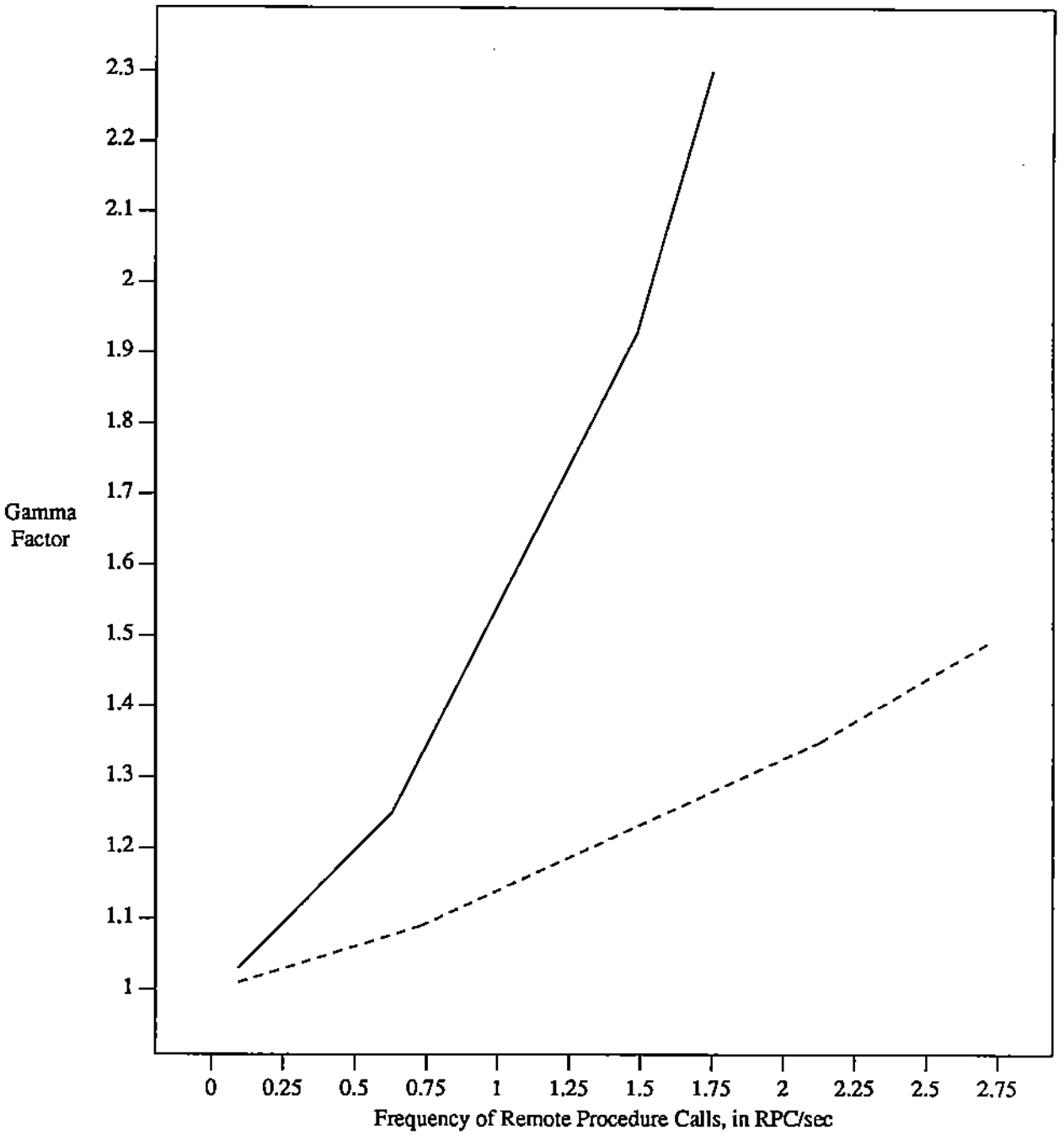

Figure 4. The gamma factor for machines B (solid) and C (dashed) function of the command execution rate for a shell script