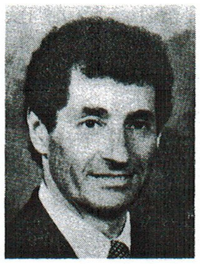
On Distributed Systems Performance *

Leonard KLEINROCK
Computer Science Department, University of California, Los Angeles, CA 90024-1596, USA

Abstract. The behavior of two interacting processes in a distributed processing environment is analyzed. This problem represents a class of problems which we will confront in the next decade as distributed systems are implemented.

Keywords. Distributed systems, distributed simulation, time-warp, synchronization, speedup, parallel processing, rollback, communicating processes.



Leonard Kleinrock is a Professor of Computer Science at U.C.L.A. He received the B.S. degree in electrical engineering from the City College of New York in 1957 (evening session) and the M.S.E.E. and Ph.D.E.E. degrees from the Massachusetts Institute of Technology in 1959 and 1963, respectively.

He joined the Faculty at the University of California, Los Angeles, in 1963. His research interests focus on local area networks, computer networks and performance evaluation of distributed systems. He has had over 150 papers published and is the author of five books. He is principal investigator for the DARPA Parallel Systems Laboratory contract at U.C.L.A. He is also founder and CEO of Technology Transfer Institute, a computer-communications seminar and consulting organization located in Santa Monica, CA.

Dr. Kleinrock is a member of the National Academy of Engineering, a Guggenheim Fellow, and a member of the Computer Science and Technology Board of the National Research Council. He has received numerous best paper and teaching awards, including the ICC 1978 Prize Winning Paper Award, the 1976 Lanchester Prize for outstanding work in Operations Research, and the Communications Society 1975 Leonard G. Abraham Prize Paper Award. In 1982, as well as having been selected to receive the C.C.N.Y. Townsend Harris Medal, he was co-winner of the L.M. Ericsson Prize, presented by His Majesty King Carl Gustaf of Sweden, for his outstanding contribution in packet switching technology. In 1986, he received the 12th Marconi International Fellowship Award, presented by His Royal Highness Prince Albert, brother of King Baudoin of Belgium, for his pioneering work in the field of computer networks.

* This research was supported by DARPA grant MDA 903-87-C-0663.

1. Introduction

The design and performance evaluation of distributed systems is an important and difficult problem and one which will occupy our attention for the next decade. Indeed it represents an example of the class of resource allocation problems with which we have been wrestling for many years in a variety of different contexts. For example, the problem of designing the operating system for time-shared computers was a major issue in the 1960's, the issue of wide-area network design and access occupied our interest in the 1970's, the problem of local area network design was our focus for the decade of the 1980's and we foresee that the general problem of distributed processing will surely occupy our attention for the coming decade of the 1990's.

One aspect of the problem has to do with resolving conflicts. This issue manifests itself both in centralized, as well as in distributed systems. The problem arises when more than one user requires access to the same resource at the same time. Usually we cannot predict exactly *when* a user will require access to the resource, we cannot predict *how long* each user will hold the resource once he gains access, most users only require the *occasional use* of the resource, and, in addition, when a user asks for access he usually expects *immediate access* to that resource. This presents a nasty set of requirements on the part of the user and we refer to such a class of users as being *bursty and asynchronous*. There are four canonical ways of resolving conflicts. The first is *queueing*: here one user gets access to the resource while the others wait for their turn. The second resolution method is that of *splitting*: here, the resource is split into as many pieces as there are competing users and each user gets a piece of the resource. The third canonical resolution method is *blocking*: here, one user gets access to the resource and the others are asked to go away. The fourth method is *smashing*: here, if more than one user asks for access, no one is given access. Examples of each of these systems are prevalent throughout the computer and communication industry. Of course, one may use hybrid mixtures of these four canonical resolution methods.

Queueing is perhaps the most common conflict resolution method and is often found in our current technology. We are all familiar with the price one pays for queueing, namely, an increase in response time due to the sharing of a resource with other users (see for example, [1,2]). However, in a distributed system, we are confronted with additional access problems and delays beyond those due to pure queueing. This comes about for many reasons discussed in the next section, all related to the fact that one cannot form a queue for free in a distributed environment. In particular, we will focus later in this paper on the specific issue of *synchronization* among coupled processes.

2. Problems of Distributed Access

Once we distribute resources and users, we are faced with a number of difficult conflict resolution and access problems. One way to divide these problems into recognizable systems is to consider a two-dimensional description where the first dimension describes the degree of coupling among the distributed processes (i.e. the amount of communication and interaction among them) and where the second dimension describes the distance that separates these processes. This may be seen in Fig. 1.

In this figure, the items above the dashed line within a quadrant refer to *processing* applications, whereas items below the dashed line refer to *communications* applications. Applications in the case of tightly coupled processes which are close to each other include parallel processing (a number of processors cooperating in the execution of a single problem) as well as back plane buses and high speed LANs. In the case of loosely coupled processes which are close to each other, we include such things as distributed processing (a number of processors, each possibly working on a different problem with occasional interaction among these processors) as well as LANs and MANs. Loosely coupled processes which are separated by large distances include applications such as distributed access (e.g. remote access to a data base) as well as wide-area networks. However, in the case of tightly coupled processes which are far from each other, we find very few applications, all of which are extremely difficult due to the large amount of interaction required at what may be long distances and/or long delays.

		COUPLING	
		TIGHT	LOOSE
DISTANCE	CLOSE	PARALLEL PROCESSING ------------------------ BACK PLANE BUS HIGH SPEED LANs	DISTRIBUTED PROCESSING ------------------------ LANs MANs
	FAR	HA I	DISTRIBUTED ACCESS ------------------------ WANs

Fig. 1. Examples of distributed systems.

A major problem that we face in distributed access is that we usually lack *global knowledge* regarding the system state. A number of things contribute to this lack of global knowledge and manifest themselves as problems. For example, a long distance between users which must interact becomes a problem with regard to the speed with which they can interact and the bandwidth of the communications of that interaction (for example, it takes light approximately 15 000 microseconds to cross the United States!); thus, the time for state information to be exchanged between processes may be seriously delayed leading to a lack of global knowledge. Another problem is that not all processes (users) may be in immediate communication with each other; for example, two users may not be able to hear each other and may require intermediate users to relay information between them. Sometimes the state information we get is incomplete and sometimes it is incorrect. Even if it is complete and correct, it may be that the state information is stale by the time we have an opportunity to use it, since it may take a user a certain amount of time to get to the location where he can use the information he gathered previously. Thus, lack of global knowledge prevents perfect use of all system resources in addition to any congestion problems which would arise in a centralized configuration with perfect knowledge.

Another source of difficulty in distributed systems has to do with access to resources themselves. For example, even though there is a set of resources in a system, not all users may be allowed access to certain of the resources. Livny and Mel-

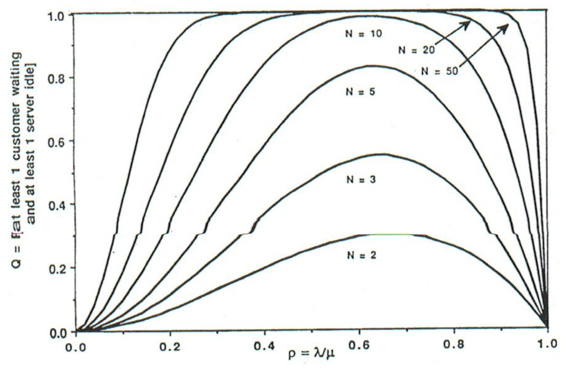


Fig. 2. Performance of N M/M/1 queueing systems.

man [3], studied the case of N independent M/M/1 queueing systems, each with its own independent arrival process at rate λ and each with its own server at rate μ. For this system, the probability Q that at least one customer is waiting in some queue and at least one server is idle is given by

$$Q = 1 - \rho^N + (1-\rho)^N\left[\rho^N - (1+\rho)^N\right]. \qquad (2.1)$$

In Fig. 2, we plot Q as a function of $\rho = \lambda/\mu$ with N as a parameter. From this result we see that Q approaches 1 for all values of ρ in the range $0 < \rho < 1$, as the number of independent systems increases ($N \to \infty$) indicating serious degradation to system performance. Of course, this is only a simple measure of system degradation but does indicate the cost of prohibited access.

One way to improve performance of these N independent queueing systems is to allow some jockeying among the queues. A model which introduces this jockeying is to allow each user from queue n ($n = 1, 2,\ldots, N$) to move to queue $n+1$ (mod N) at a rate α. Clearly, for $\alpha = 0$, we have Livny's model and for $\alpha = \infty$ we have a simple M/M/N queue which provides, in some sense, perfect sharing of the N servers. Unfortunately, this model with jockeying is an unsolved coupled queueing problem (even for the case $N = 2$); coupled queueing problems are very difficult in general.

Another problem with access has to do with the fact that the distributed system may be lossy; by this we mean it is possible, that, as the load increases, the throughput might decrease due to internal waste of resources (see for example [4]).

The access problem may also manifest itself, in some cases, by requiring that resources be used in series (one after the other) rather than all at once. This leads to increased delays and possibly inefficient use of resources. An example here might be that of a message passing through a wide-area network which must hop from channel to channel as it makes its way through the network.

Another manifestation of this access problem is that there may be synchronization and/or precedence constraints in the way in which the tasks required by a user are executed. This arises with the parallel use of processors and possible restrictions on how much parallelism the algorithm or the application allows. It is on this type of problem that we focus in the remainder of this paper. Below, we give a description of the problem, a model of the system, analysis of its behavior and a discussion of the results.

3. A Model for Synchronization Beween Two Processes

Assume we have a job which is partitioned into two processes, each of which is executed on a separate processor. As these processes are executed, we consider that they advance along the x-axis in steps of length one (i.e. they visit the non-negative integers), each beginning at $x = 0$ at time $t = 0$. Each process independently takes an exponentially distributed amount of time, with parameter λ_i ($i = 1, 2$), to advance from position k to position $k+1$ ($k = 0, 1, 2,\ldots$). When process i advances one unit along this axis, it will send a message to the other process with probability q_i ($0 < q_i \leqslant 1$). Upon receiving a message from the other (sending) process, this (receiving) process will do the following:

(1) If its position along the x-axis is equal to or behind the sending process, it will ignore the message.

(2) If it is ahead of the sending process, it will immediately move back (i.e., "rollback") along the x-axis to the current position of the sending process.

This is a simple model of distributed simulation (motivated by the time warp distributed simulation algorithm [5]) where two processors are both working on a simulation job in an effort to speed it up. They both proceed independently until such time as one (slower) process transmits a message in the "past" of the other (faster) process. This causes the faster process to "rollback" to the point that the slower process is at, after which they advance independently again until the next rollback, etc.

Let $F(t)$ = position of the first process (process 1) at time t and let $S(t)$ = position of the second process (process 2) at time t. Further, let

$$D(t) = F(t) - S(t);$$

$D(t) = 0$ whenever (2) occurs (i.e., a rollback). We are interested in studying the Markov process $D(t)$. From our assumptions that $F(0) = S(0) = 0$, we have $D(0) = 0$. Clearly, $D(t)$ can take on any integer value (i.e., it certainly can go negative). We will solve for

$$p_k = \lim_{t\to\infty} P[D(t) = k], \quad k = \dots, -2, -1, 0, 1, 2, \dots \tag{3.1}$$

namely, the equilibrium probability for the Markov Chain $D(t)$. Moreover, we will find the average separation between processes as well as the speedup with which the computation proceeds when using two processors relative to the use of a single processor as described below.

Our model is that of a discrete state, continuous time process. Some previous work on variations of this model already exists. For example, Mitra and Mitrani [6] studied a related model in which they considered a continuous state, discrete time model not unlike the one we have described here. Their results are similar to ours in some ways although their method of analysis appears to be far more complex than ours. Lavenberg, Muntz, and Samadi [7] provide an approximate analysis of a continuous time, continuous state model. (We have also completed the analysis of the discrete time, discrete state case which will be published elsewher.) Moreover, Felderman and Kleinrock [8] give an upper bound on the gain in speedup that P unsynchronized processors can achieve relative to P processors which are forced to synchronize at every step.

4. Analysis

Let us analyze the behavior of these two coupled processes which we have modeled as a one-dimensional discrete state, continuous time Markov Chain. The following balance equations are easily obtained:

$$(\lambda_1 + \lambda_2) p_k = \lambda_1 p_{k-1} + \lambda_2 \bar{q}_2 p_{k+1}, \quad k = 1, 2, \dots, \tag{4.1}$$

$$(\lambda_1 + \lambda_2) p_{-k} = \lambda_2 p_{-(k-1)} + \lambda_1 \bar{q}_1 p_{-(k+1)}, \quad k = 1, 2, \dots, \tag{4.2}$$

$$(\lambda_1 + \lambda_2) p_0 = \lambda_2 q_2 \sum_{k=1}^{\infty} p_k + \lambda_1 q_1 \sum_{k=1}^{\infty} p_{-k} + \lambda_2 \bar{q}_2 p_1 + \lambda_1 \bar{q}_1 p_{-1} \tag{4.3}$$

where $\bar{q}_i = 1 - q_i$.

We solve this system of linear difference equations using the usual approach of z-transforms [1] by defining the two transforms

$$R(z) = \sum_{k=1}^{\infty} p_k z^k, \tag{4.4}$$

$$Q(z) = \sum_{k=1}^{\infty} p_{-k} z^k. \tag{4.5}$$

We further define the relative speed parameter

$$a = \frac{\lambda_1}{\lambda_1 + \lambda_2}. \tag{4.6}$$

Multiplying (4.1) by z^k and summing over the range $k = 1, 2, \dots$ we obtain

$$R(z) = \frac{z(\bar{a}\bar{q}_2 p_1 - a p_0 z)}{a z^2 - z + \bar{a}\bar{q}_2} \tag{4.7}$$

where $\bar{a} = 1 - a$. Similarly, multiplying (4.2) by z^k and summing over the range $k = 1, 2, \dots$ we obtain

$$Q(z) = \frac{z(a\bar{q}_1 p_{-1} - \bar{a} p_0 z)}{\bar{a} z^2 - z + a\bar{q}_1}. \tag{4.8}$$

Note the duality between $R(z)$ and $Q(z)$ with regard to the variables a, q_1, and q_2.

The denominator roots (i.e. the poles) of $R(z)$ are given by

$$r_1 = \frac{1 - \sqrt{(1-2a)^2 + 4a\bar{a}q_2}}{2a}, \quad r_2 = \frac{1 + \sqrt{(1-2a)^2 + 4a\bar{a}q_2}}{2a}. \tag{4.9}$$

Similarly, the poles of $Q(z)$ are given by

$$s_1 = \frac{1 - \sqrt{(1-2a)^2 + 4a\bar{a}q_1}}{2\bar{a}}, \quad s_2 = \frac{1 + \sqrt{(1-2a)^2 + 4a\bar{a}q_1}}{2\bar{a}}. \tag{4.10}$$

It is easy to show that these four poles are real and that $0 \leqslant r_1 \leqslant 1$, $1 \leqslant r_2$, $0 \leqslant s_1 \leqslant 1$ and $1 \leqslant s_2$. Since $R(z)$ and $Q(z)$ are both analytic in the region $|z| \leqslant 1$, it must be that numerator $(R(r_1)) = 0$ and numerator $(Q(s_1)) = 0$. From this observation and from (4.7) and (4.8) we have

$$p_1 = \frac{ar_1}{\bar{a}q_2} p_0, \tag{4.11}$$

$$p_{-1} = \frac{\bar{a}s_1}{aq_1} p_0. \tag{4.12}$$

In addition, by conserving probability we have

$$p_0 + R(1) + Q(1) = 1. \tag{4.13}$$

From these last three we readily find

$$p_0 = \frac{(r_2 - 1)(s_2 - 1)}{r_2 s_2 - 1}. \tag{4.14}$$

From (4.11) and (4.12) we may simply rewrite $R(z)$ and $Q(z)$ as

$$R(z) = \frac{zp_0}{r_2 - z}, \qquad Q(z) = \frac{zp_0}{s_2 - z}. \tag{4.15}$$

We may now invert both $R(z)$ and $Q(z)$ to give us the equilibrium distribution for our Markov Chain, namely,

$$p_k = \begin{cases} p_0\left(\dfrac{1}{r_2}\right)^k, & k = 0, 1, 2, \dots, \\ p_0\left(\dfrac{1}{s_2}\right)^{-k}, & k = 0, -1, -2, \dots. \end{cases} \tag{4.16}$$

Equation (4.16), along with (4.9), (4.10) and (4.14), give us the complete solution to the Markov Chain.

To find the average separation between these two processes on the x-axis, namely, $\bar{K} = \lim_{t\to\infty} E[|D(t)|]$, we calculate as follows:

$$\bar{K} = \sum_{k=1}^{\infty} k(p_k + p_{-k})$$

which gives us

$$\bar{K} = p_0\left[\frac{r_2}{(r_2-1)^2} + \frac{s_2}{(s_2-1)^2}\right]. \tag{4.17}$$

Let us now calculate the speedup S which is defined as the rate at which the two processor system carries out useful processing divided by the rate at which an equivalent single processor carries out useful processing. We define the equivalent single processor as one which moves a process along the time axis at a rate equal to the average rate of the two original processes, namely: $\frac{1}{2}(\lambda_1 + \lambda_2)$. In this single processor case, there is no rollback to worry about and so useful progress occurs at the rate $\frac{1}{2}(\lambda_1 + \lambda_2)$. In the two processor case, useful progress is equal to the sum of the two rates minus the expected rollback for each process. If $D(t) = k$ at time t, and if the next advance along the x-axis is made by the lagging process which also causes the leading process to rollback (with probability q_2), then the leading process will be rolled back only a distance $k - 1$ since the lagging process just advanced one step along the x-axis. Thus, we see that the rate at which the two processor system advances, on average, is given by

$$\lambda_1 + \lambda_2 - \sum_{k=1}^{\infty} \lambda_2 q_2 p_k (k-1) - \sum_{k=1}^{\infty} \lambda_1 q_1 p_{-k}(k-1).$$

Thus, the speedup is given by

$$S = 2 - 2\bar{a}q_2 \sum_{k=1}^{\infty} p_k(k-1) - 2aq_1 \sum_{k=1}^{\infty} p_{-k}(k-1).$$

This leads us to the following general expression for the system speedup:

$$S = 2\left[1 - \frac{\bar{a}q_2 p_0}{(r_2-1)^2} - \frac{aq_1 p_0}{(s_2-1)^2}\right]. \tag{4.18}$$

5. The Symmetric Case $q_1 = q_2$

In this section we consider the symmetric case where $q_1 = q_2 = q$; that is, each process has the same probability of sending a message to the other process. We now have $ar_1 = \bar{a}s_1$ and $ar_2 = \bar{a}s_2$.

The speedup is shown in Fig. 3 as a function of a and q. For $a = \frac{1}{2}$, the speedup rises continously to its maximum value of $S = 2$ as $q \to 0$. For $q = 0$, $S = 2$ for all a but S has a discontinuity for all $a \neq \frac{1}{2}$; this discontinuity is not shown clearly in Fig. 3. (For $q = 0$, no rollbacks occur and it is intuitively clear that $S = 2$.) Note for $q > 0$ that, when $a \to 0$ or $a \to 1$ (that is, $\lambda_1 \to 0$ or $\lambda_2 \to 0$), then the speedup goes to 0; this is the case, since one process moves extremely slowly (compared to the equivalent single process) and it

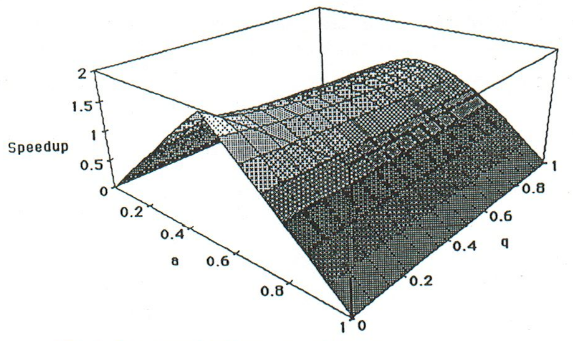


Fig. 3. Speedup for the symmetric case, $q_1 = q_2 = q$.

will occasionally drag the faster process back to its lagging position.

6. The Balanced Case $\lambda_1 = \lambda_2$

Here we consider the balanced case where both processes move at the same rate giving us $a = (1 - a) = \frac{1}{2}$. We see that $r_1 = 1 - \sqrt{q_2}$, $r_2 = 1 + \sqrt{q_2}$, $s_1 = 1 - \sqrt{q_1}$ and $s_2 = 1 + \sqrt{q_1}$. In addition,

$$p_0 = \frac{\sqrt{q_1 q_2}}{\sqrt{q_1 q_2} + \sqrt{q_1} + \sqrt{q_2}}. \tag{6.1}$$

Furthermore, in this case, we find that the speedup is simply

$$S = \frac{2\left(\sqrt{q_1} + \sqrt{q_2}\right)}{\sqrt{q_1 q_2} + \sqrt{q_1} + \sqrt{q_2}}. \tag{6.2}$$

In Fig. 4 we show the speedup as a function of q_1 and q_2. Note, of course, that the speedup goes to 2 for $q_1 = q_2 = 0$ and goes to $\frac{4}{3}$ for $q_1 = q_2 = 1$.

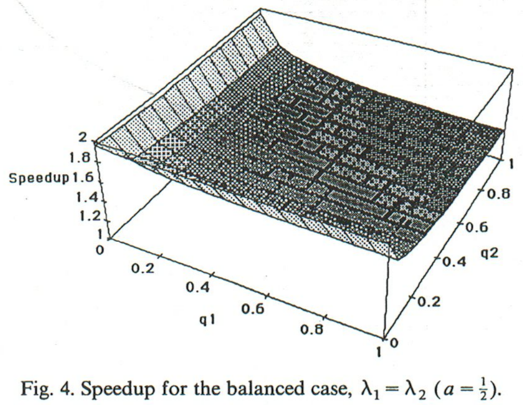


Fig. 4. Speedup for the balanced case, $\lambda_1 = \lambda_2$ $(a = \frac{1}{2})$.

7. The Symmetric Balanced Case: $q_1 = q_2$ and $\lambda_1 = \lambda_2$

In this symmetric balanced case, where both processes move at the same speed and both have the same probability, $q_1 = q_2 = q$, of passing a message to the other process, we obtain great simplifications. In particular, we have $r_1 = s_1 = 1 - \sqrt{q}$ and $r_2 = s_2 = 1 + \sqrt{q}$. Note, again, that $a = (1 - a) = \frac{1}{2}$. In this case we find,

$$p_0 = \frac{\sqrt{q}}{2 + \sqrt{q}}. \tag{7.1}$$

The average process separation becomes

$$\overline{K} = \frac{2(1 + \sqrt{q})}{\sqrt{q}(2 + \sqrt{q})}. \tag{7.2}$$

The speedup is given by

$$S = \frac{4}{2 + \sqrt{q}}. \tag{7.3}$$

The speedup function in this very special case is shown in Fig. 5. Note for $q = 0$ that $S = 2$ whereas for $q = 1$ we have $S = \frac{4}{3}$. We can see this last result intuitively as follows. If each process always sends a message to the other process when it advances, then the time for both processes to advance one unit is equal to the maximum of two exponential delays which we know is equal to 1.5 times the mean. Thus, the rate of progress for each process is simply $\frac{2}{3}$ times the rate of a single process. Since both are moving at a rate $\frac{2}{3}$, the sum is equal to $\frac{4}{3}$ which yields the result for $q = 1$.

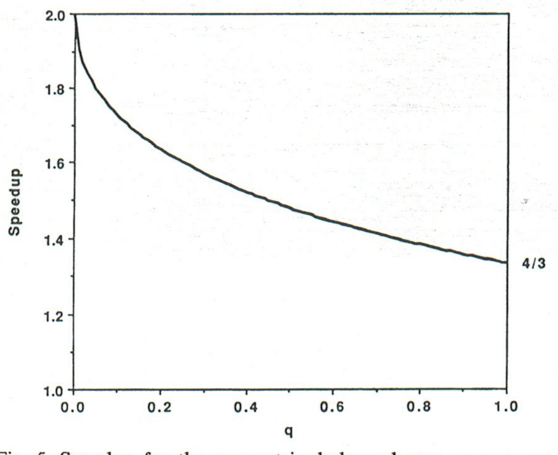


Fig. 5. Speedup for the symmetric, balanced case, $q_1 = q_1 = q$, $\lambda_1 = \lambda_2$ $(a = \frac{1}{2})$.

8. Conclusions

In this paper we have focused on the speedup available when two processes limit each others' rate of progress by passing messages between each other (these messages may cause a process to roll back and therefore waste some of the work it has accomplished). We gave the complete solution for the important system performance variables in the general case. We found for the symmetric balanced case that when $q = 0$ the speedup reaches its maximum value of 2 but that the speedup falls off infinitely quickly as q increases from 0, the limiting speedup being $\frac{4}{3}$ at $q = 1$.

The analysis given here is for only two processors. For larger numbers of interacting processes, the mathematics becomes far more complex and the most likely fruitful approach to understand the interaction among many asynchronous processes would be via approximate solutions.

Synchronization is only one source of overhead one finds in distributed systems. There are numerous other issues which impact the performance of distributed systems. Indeed, we are only beginning to formulate the models and gain the understanding required to analyze, design, and properly implement distributed systems in this coming decade.

References

[1] L. Kleinrock, *Queueing Systems, Vol. I: Theory* (Wiley Interscience, New York, 1975).

[2] L. Kleinrock, *Queueing Systems, Vol. II: Computer Applications* (Wiley Interscience, New York, 1976).

[3] M. Livny and M. Melman, Load balancing in homogeneous broadcast distributed systems, in: *Proc. ACM Computer Network Performance Symposium* (1982) 47–55.

[4] L. Kleinrock, On flow control in computer networks, in: *Proc. Internat. Conf. Communications, Vol. II*, Toronto, Ontario (1978) 27.2.1–27.2.5.

[5] D.R. Jefferson, Virtual time, *ACM Trans. Programm. Languages Systems* **7** (3) (1985) 404–425.

[6] D. Mitra and I. Mitrani, Analysis and optimum performance of two message-passing parallel processors synchronized by rollback, in: *Performance '84* (North-Holland, Amsterdam, 1984) 35–50.

[7] S. Lavenberg, R. Muntz and B. Samadi, Performance analysis of a rollback method for distributed simulation, in: *Performance '83* (North-Holland, Amsterdam, 1983) 117–132.

[8] R. Felderman and L. Kleinrock, An upper bound on the improvement of asynchronous versus synchronous distributed processing in: *Distributed Simulation 1990* (Society for Computer Simulation, 1990) 131–136.