SYNCHRONIZATION ALGORITHMS
AND THEIR EVALUATION
by: Manfred Ruschitzka
DCS-TR 80 A
June 1979

Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey

The work reported in this paper was performed while the author was on leave at the Computer Science Division of the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley.
ABSTRACT

The need for exclusive accesses to shared resources, like memory and communications facilities, requires the synchronization of asynchronous processes which degrades the performance of a distributed system. A Markov chain which models a set of cyclic processes can be used to evaluate the performance degradation. This is demonstrated for systems with a single shared resource (e.g. a communications bus), and ring-structured systems (e.g. as defined by the scenario of the Dining Philosophers [4,6]) for which a distributed, a centralized, and a "fair" synchronization algorithm are considered. Performance data for both system types are plotted and the respective merits of the three algorithms are discussed. The state space of the Markov chain equals that of the status variables used to verify synchronization algorithms, and, thus, provides a common framework for evaluating correctness and performance.
INDEX TERMS

Cyclic processes, Dining Philosophers, distributed systems, interference,
Markov chains, modeling, performance evaluation, shared resources.
FIGURE CAPTIONS

Figure 1. Topology of the single-resource system
Figure 2. The transition-rate diagram of the single-resource system
Figure 3. Interference in the single-resource system (algorithm A)
Figure 4. Topology of the ring-structured system
Figure 5. The transition-rate diagram for four processes under algorithm A
Figure 6. Interference as a function of system size (algorithm A)
Figure 7. Interference in ring-structured systems under algorithm A
Figure 8. The overshoot anomaly in a system of size six (algorithm A)
Figure 9. The transition-rate diagram for six processes under algorithm B
Figure 10. Interference for six processes under algorithm A, B, and C
Figure 11. The transition-rate diagram for four processes under algorithm C
Figure 12. Interference in ring-structures under algorithm C
Figure 13. The degenerate transition-rate diagram under algorithm C for
infinite service-ratios
1. INTRODUCTION

The performance of a distributed system is affected by dependences among the various processes. When these dependences are minimal, the performance will be proportional to the number of processors or concurrently executing processes in the system. Conversely, heavy dependences may effectively degrade the performance to that of a single-processor system. There are two types of dependences which cause this detrimental effect: exclusive access of shared resources and the dependence of the progress of one process on that of another. While the former is common to all distributed systems, the latter varies with the application. In interaction-oriented systems, this type of dependence may not be present at all. On the other hand, many parallel systems run computations which have been decomposed into smaller processes whose initiations depend on their predecessors' completions. Independent of the type distinction, synchronization procedures serve to coordinate dependent processes by delaying some of them temporarily. The design of powerful synchronization mechanisms and the programming of provably correct synchronization algorithms have been given much attention and there is now a wealth of results to draw from [1,3,4,5,6,7]. In particular, the technique of using status variables to formally analyze synchronization algorithms has made their verification a manageable task. In this technique, the system is modeled by a finite-state machine whose state space is defined by the permitted values of the status variables.

In this paper, we show how the intrinsic performance degradation due to exclusive accesses of shared resources can be evaluated as a function of the synchronization algorithm. The model chosen for this purpose is that of a collection of cyclic processes which compete for fixed subsets of the system.
resources during part of their cycle. Its parameters include the number of processes, the number of shared resources, the percentage of time a process requires exclusive access to resources, and the system configuration. Performance measures are obtained from a Markov chain whose state space is defined by the status variables of the synchronization algorithm under consideration.
II. MODELS, DEFINITIONS, AND MEASURES

We consider symmetric distributed systems of \( n \) processes and \( m \) resources. Processes are asynchronous and cyclic, and a process cycle is divided into internal and external service. External service requires the exclusive allocation of a subset of the resources. For each process, this subset is fixed. The basic algorithm of process \( i \) \((i = 0, 1, \ldots, N-1)\) is outlined below.

\begin{verbatim}
while true do
begin
  internal service;
  request(i);
  external service;
  release(i)
end:
\end{verbatim}

Allocation and deallocation are part of the external service, and the synchronization procedures request and release represent the synchronization algorithm. Their implementation may be based on any of the standard mechanisms, like hardware arbitration logic, test-and-set instructions, semaphores [3], or synchronization monitors [1,4,7]. We shall only assume that the times to execute a release or a successful request are negligibly small compared to the internal or external service. An unsuccessful request, however, will cause a process to wait. Thus, a process may be in one of three states: I (internal), W (wait), or E (external).

We assume exponentially distributed service times with parameters \( \tau \) and \( \eta \) for the internal and external service respectively. The service-ratio \( r \) is defined as the ratio of the mean external and the mean internal service time:

\[ r = \frac{\tau}{\eta}. \]

In equilibrium, the (time-average) probability of finding a process in state \( S \) equals the fraction of time that process spends in state \( S \). \( P(I), P(W), \) and \( P(E) \) denote these equilibrium probabilities for states I, W and E respectively. By definition, the average internal service time of a process equals
\( P(I) = r \cdot P(E) \). \hspace{1cm} (1)

An important performance measure of cyclic processes is the cycle time \( \bar{C} \) [2]. When no waiting occurs, the average cycle time equals \( \bar{C}_{\text{opt}} = 1/r + 1/n \).

In general, the average cycle time is

\[ \bar{C} = \bar{C}_{\text{opt}} + \bar{D}, \hspace{1cm} (2) \]

where \( \bar{D} \) is the average delay in the wait state. The cycling rate is given by the inverse of \( \bar{C} \). Applying Little's theorem [10] to the wait state yields

\[ P(W) = \frac{\bar{D}}{\bar{C}}, \hspace{1cm} (3) \]

and combining the last two equations gives

\[ \bar{D} = \bar{C}_{\text{opt}} \cdot P(W)/(1 - P(W)). \hspace{1cm} (4) \]

We define the interference \( i \) as this average delay normalized with respect to \( \bar{C}_{\text{opt}} \):

\[ i = P(W)/(1 - P(W)). \hspace{1cm} (5) \]

Interference is measured on a scale from zero to infinity and we shall frequently express it as a percentage. It is a function of the service-ratio, the number of processes, the number of resources, the request topology (which describes the resource subsets which the various processes require, c.f. Figures 1 or 4 which will be discussed later), and the synchronization algorithm. Note from
equations (2), (4) and (5) that $\overline{C} = (1+i)\overline{C}_{\text{opt}}$.

The state probabilities can be expressed as functions of the interference and the service-ratio. After solving equation (5) for $P(W)$, the expressions for $P(E)$ and $P(I)$ follow from equation (1) and probability conservation ($P(I) + P(W) + P(E) = 1$):

\[
P(E) = \frac{1}{[(1 + i)(1 + r)]} \\
P(I) = \frac{r}{[(1 + i)(1 + r)]} \\
P(W) = \frac{i}{(1 + i)}
\]

(6)

These equilibrium probabilities are valid for arbitrary symmetric systems. Their characteristics are captured by the interference. The service-ratio will typically be a given parameter.

The configuration of a distributed system is often guided by throughput and cost considerations. We define the system throughput $t$ as the average number of cycles performed per unit time:

\[
t = \frac{n}{\overline{C}} = \frac{(n/\overline{C}_{\text{opt}})}{(1 + i)}.
\]

(7)

Since interference is itself a function of $n$, throughput need not be proportional to $n$. System cost, however, is proportional to $n$. (We assume that the cost of resources is distributed over the $n$ processes.) We define costeffectiveness as the ratio of throughput and the system cost $nD$. The relative costeffectiveness $e$ denotes the ratio of the costeffectiveness of an $n$-process system and the costeffectiveness of a single-process system (no interference):

\[
e = \frac{(n/\overline{C}_{\text{opt}})/[(1 + i) nD]}{(1/\overline{C}_{\text{opt}})/D} = 1/(1 + i).
\]

(8)

The simple form of this measure permits cost problems to be treated as inter-
ference issues.

With 3 states per process, a distributed system of n processes may assume up to $3^n$ different states. The actual number depends on the synchronization algorithm. For instance, only one process may be allowed in state E at any one time if there is only a single resource which is shared by all processes.

In general, the state of a process depends on all other n-1 processes and the system state probabilities cannot be expressed as a product of the process state probabilities. It is then necessary to first obtain the joint system state probabilities (typically by solving the system of simultaneous linear equations resulting from the appropriate Markov chain) and to derive the marginal process state probabilities from them. While this approach is straightforward for small n, the rapid growth of the system state space may render it impractical for large n. For simple cases, it is sometimes possible to obtain explicit expressions as a function of n. While this seems to be the exception, the results for small system sizes may already display a strong convergence indicative of large systems. Generally, the dependence of the process states on each other is an intrinsic property of distributed systems. Without it, such a system represents little more than a collection of non-interacting modules.
III. SINGLE-RESOURCE SYSTEMS

The synchronization of \( n \) cyclic processes which need to access a single resource exclusively is a common operating system task. The resource may be a device, a set of storage locations, or a sequence of instructions (critical section). A similar situation occurs when \( n \) processors share a common bus structure. The request topology for such systems with \( m=1 \) is depicted in Figure 1. Arrows are used to point from a process to the subset of resources it requires. In this case, all subsets are identical. This property results in a drastic reduction of the system state space.

The synchronization algorithm for this system can be quite simple. Whenever a process requests the resource and all other \( n-1 \) processes are in state I, it will immediately proceed to state E. Otherwise, it waits until the resource becomes available. When a process exits from state E while other processes are waiting, one of them will make the transition from state W to state E. (The selection of this process is arbitrary in view of the symmetric arrangement; any neutral policy, like random or first-come first-serve, will suffice.) The synchronization procedures request and release which implement this synchronization algorithm, termed Algorithm S, are sketched below. For simplicity of presentation, we assume the availability of a monitor mechanism \([1,7]\) which assures exclusive access to the synchronization procedures and provides process queues which are manipulated by the primitives wait and signal. Wait causes a process to block on the specified queue, and signal unblocks one of the processes on the queue (neutral policy). \( k \) counts the number of processes in the internal state; it is globally defined (within the monitor) and initialized to \( n \). There is one process queue for all processes.
Figure 1. Topology of the single-resource system.
Algorithm 5

procedure request(i);
begin
    k := k-1;
    if k < n-1 then wait(queue)
end;

procedure release(i);
begin
    k := k+1;
    signal(queue)
end;

In accordance with algorithm 5, we denote system states by [k], where k is the number of processes in state i. Note that the number of waiting processes is zero in states [n] and [n-1], and n-k-1 in states [0] through [n-2]. Figure 2 shows the transition-rate diagram for single-resource systems. In state [k], any one of k processes may complete its internal service; the transition-rate to state [k-1] is therefore k times the completion rate \( \tau \). If k < n, the process using the resource completes the external service with rate n which is therefore also the transition-rate to state [k+1]. This transition-rate diagram may be recognized as that of the classical single-server system with finite population (M/M/1//n) [9]. In fact, if the system boundary is drawn around the external resource, i.e. k denotes the number of processes in states W or E, the single-resource system is by definition such as M/M/1//n system. Furthermore, this transition-rate diagram also equals that of an n-server loss system with infinite population (M/M/n/n [9]) where k denotes the number of occupied servers. That system would model the internal activities of the n processes; its Poisson arrivals are assured by the Poisson output from the external resource.

Equating the flow rates into and out of each state in Figure 2 yields n equations for the probabilities \( P_k \) of state [k]. One of these is redundant and should be replaced by the probability conservation law. Solving
Figure 2. The transition-rate diagram of the single resource system.
these equations (e.g. [9]) yields

\[ p_k = p_n r^{n-k} \frac{n!}{k!}, \quad k = 0,1,...,n, \quad (9) \]

where

\[ p_n = \frac{1}{n!} \frac{r^n}{\sum_{j=0}^{n} 1/(j! r^j)}. \quad (10) \]

\( p_n \) denotes the fraction of time that the external resource is idle. This quantity is known as Erlang's loss formula and frequently plotted as \( B(n,1/r) \).

Making use of the symmetric nature of the processes, the derivation of the marginal process state probabilities is straightforward. Specifically, we have

\[
\begin{align*}
P(T) &= \sum_{k=1}^{n} \frac{k}{n} p_k = \frac{(1 - p_n)}{nr} \\
P(E) &= \sum_{k=0}^{n-1} \frac{1}{n} p_k = \frac{(1 - p_n)}{n} \\
P(W) &= \sum_{k=0}^{n-1} \frac{(n-k-1)}{n} p_k = 1 - \frac{(1+r)(1-p_n)}{nr} \\
\end{align*}
\]

(11)

The interference in the single-resource system follows from equations (5) and (11):

\[ i = nr/[((1+r)(1-p_n))] - 1. \quad (12) \]

With the service-ratio as parameter, the interference is plotted in Figure 3 as a function of system size. It can be seen that the interference converges toward a linear asymptote. Also, the rate of convergence increases with the service-ratio. (For \( r = \infty \), the interference and its asymptote are identical.)
Figure 3. Interference in the single-resource system (algorithm S).

Intersection of all asymptotes at (0, -1)
We proceed to discuss a characteristic of the single-resource system which explains this behavior.

When \( n \) approaches infinity, the system states are Poisson distributed with parameter \( 1/r \). This is readily verified by taking the limit of equation (9). Simultaneously, \( P_n \) approaches zero for any positive \( r \). (For \( r = 0 \), however, \( P_n = 1 \) for any value of \( n \).) Thus, equation (12) becomes

\[
1 = nr/(1+r) - 1, \quad P_n \ll 1, \quad (13)
\]

which, in fact, describes the linear asymptotes in Figure 3. For a given \( r \), system throughput cannot be noticeably increased by adding processes once the interference is "close" to its asymptote. To show this, we substitute equation (12) in equation (7) to get

\[
t = (1/C_{opt})[(1+r)/r] (1-P_n).
\]

(14)

For \( P_n \ll 1 \), \( t = (1+r)/rC_{opt} \). In the limit, the throughput becomes independent of \( n \)! Any additional processes add just enough computing power to compensate for the additional interference they create.

From equations (12) and (8) we obtain

\[
e = (1+r)(1-P_n)/rn
\]

(15)

for the relative cost-effectiveness of the single-resource system. Both this measure and the throughput need to be considered in dimensioning a system. Given the service-ratio \( r \) and a minimal cost goal \( e \), equation (15) can be solved graphically for the maximum number of system processes satisfying the cost goal. Equation (14) would then determine whether this number is sufficiently large to provide the necessary throughput.
IV. A RING-STRUCTURED SYSTEM

While single-resource systems can be analyzed as birth-death processes, this is unfortunately not the case when multiple resources permit two or more processes to proceed with their external service concurrently. Since all non-trivial synchronization problems involve multiple resources, the evaluation of their performance has typically been restricted to deadlock and fairness considerations. In view of its interesting synchronization aspect, we choose to model the system known as the Dining Philosophers [4,6] to illustrate a distributed system with requests for multiple resources. In this system, there are \( n \) processes and \( m - n \) resources. Each resource may be requested by two of the processes and each process requires two resources for its external service. The request topology of this system is shown in Figure 4. This model also serves to describe the topology of a ring-structured architecture in which the resources represent communications busses with arbitration logic that prevents two adjacent processors from using the bus concurrently. Usage here refers to sending; receiving is viewed as an interrupt activity and, thus, part of the internal service.

We quote the problem formulation for the Dining Philosophers from Hoare's article [6]:

"Five Benthamite philosophers spend their lives between eating and thinking. To provide them sustenance, a wealthy benefactor has given each of them his own place at a round table, and in the middle is a large and continually replenished bowl of spaghetti, from which they can help themselves when they are seated. The spaghetti is so long and tangled that it requires two forks to be conveyed to the mouth; but unfortunately the wealthy benefactor has provided only five forks in all, one between each philosopher's place. The only forks that a philosopher can pick up are those on his immediate right and his immediate left."

In our model, the five philosophers correspond to \( n \) processes; thinking
Figure 4. Topology of the ring-structured system.
and eating correspond to internal and external service respectively. The two adjacent forks constitute the subset of resources which a process requests. A correct and efficient solution to this problem requires that a process obtains either both resources or none at all. We shall consider three different synchronization algorithms which solve this problem and evaluate their performance.

A Distributed Synchronization Algorithm

Suggested solutions to this synchronization problem [4,6] tend to be distributed in the sense that decisions are based on the states of a few processes only, rather than all n processes. For the implementation of such a distributed algorithm, which we shall call Algorithm A, we assume that processes are numbered 0 through n-1 around the ring. A status variable res[i] will indicate the number of resources in the subset that are available to process i (its values are 0,1,2 with an initial value of 2). When process i makes a request and res[i] = 2, it will decrement the status variables of its two neighbors and enter the external state. If res[i] ≠ 2, the process enters the wait state. Processes leaving the external state increment their neighbors' status variables. A neighbor for which the resulting value is 2 makes the transition from the wait state to the external state. This algorithm makes use of one process queue, queue[i], per process.

Algorithm A

procedure request(i);
begin
  if res(i) ≠ 2 then wait(queue[i]);
  res[(i-1)modn] := res[(i-1)modn]-1;
  res[(i+1)modn] := res[(i+1)modn]-1
end;

procedure release(i);
begin
  res[(i-1)modn] := res[(i-1)modn]+1;
  res[(i+1)modn] := res[(i+1)modn]+1;
  if res[(i-1)modn] = 2 then signal(queue[(i-1)modn]);
  if res[(i+1)modn] = 2 then signal(queue[(i+1)modn])
end;
With three states per process \( (I, W, E) \), an upper limit for the number of system states for \( n \) processes is \( 3^n \). The actual number is somewhat smaller since no two adjacent processes may both be in state \( E \) and no process may be in state \( W \) unless at least one of its neighbors is in state \( E \). Furthermore, symmetric system states and those which are invariant under rotation may be merged. While these reductions do not prevent a rapid growth of the number of system states with \( n \), they do assure the tractability of reasonably small systems.

In the case of two processes, the relevant states are \( II, EI, \) and \( EW \). It is easily verified that the corresponding transition-rate diagram forms a Markov chain as depicted in Figure 2 with \( n=2 \). States \( [0], [1] \) and \( [2] \) are equivalent to states \( EW, EI, \) and \( II \) respectively. It follows that the analysis of the single-resource model applies to this case. This is due to the fact that both processes require the same subset of resources, thus effectively reducing the problem to a single-resource system.

For three processes, the system states can be reduced to the following four: \( III, EII, EWI, EWW \). Again, the transition-rate diagram of Figure 2 applies (here with \( n=3 \)). State \( [k] \) corresponds to a system state with \( k \) processes in state \( I \). In this case, the equivalence to the single-resource model follows because every process requires two out of the three system resources. Consequently, no two processes can execute externally in parallel.

The situation changes drastically for four processes, since it is now possible that two opposite processes use the external resources concurrently. Figure 5 shows the transition-rate diagram for this case. Applying the flow conservation law to every state yields the equations governing the system behavior in equilibrium. With \( P_j \) denoting the equilibrium probability of state \( [j] \), we have
Figure 5. The transition-rate diagram for four processes under algorithm A.
\[ 4r \, p_a = p_b \]
\[ (1+3r) \, p_b = 4r \, p_a + p_c + 2 \, p_e \]
\[ (1+2r) \, p_c = 2r \, p_b + 2 \, p_f \]
\[ (1+r) \, p_d = r \, p_c + 2 \, p_g \]
\[ (2+2r) \, p_e = r \, p_b + p_d \]
\[ (2+r) \, p_f = 2r \, p_e + r \, p_c \]
\[ 2 \, p_g = r \, p_f + r \, p_d \]

To avoid redundancy, one of these equations must be replaced by the probability conservation law:

\[ p_a + p_b + p_c + p_d + p_e + p_f + p_g = 1 \]

It is a tedious, though straightforward, task to obtain the state probabilities as functions of \( r \). We do not present these expressions since little insight can be gained from their forms. Solving such systems of simultaneous linear equations numerically proves to be considerably more effective. To obtain the process wait state probability, we determine the fraction of waiting processes per system state: zero in states \([a], [b], \text{ and } [e]\); one fourth in states \([c] \text{ and } [f]\); one half in states \([d] \text{ and } [g]\). Due to process symmetry, we have

\[ P(W) = (1/4)(p_c+p_f) + (1/2)(p_d+p_g). \]

The interference and the internal and external state probabilities follow from equations (5) and (6). Interference values are plotted in Figure 6, but we precede their discussion by some comments about larger systems.

For five processes, the following ten system states result: IIII1, EIII1, EIII, EIIII, EWEII, EIEII, EIEW1, EIEW1, EWIIW, EWII1, EWEWI, and EWEWW. Note that the external service of at most two processes may overlap.
For six processes, there are 21 system states and up to three processes may be in the external state concurrently. In general, the external service of half the processes may overlap in systems of even size. If \( n \) is odd, at most \((n-1)/2\) processes can be serviced externally. Odd-sized systems should therefore display a higher degree of interference.

Figure 6 summarizes interference characteristics as a function of system size with the service-ratio as a parameter. For a given service-ratio, interference increases with system size for even-sized systems and decreases for odd-sized systems. Unless the service-ratio is rather large \((r > 2)\), the convergence is quite rapid and the interference values for \( n = 6 \) are accurate to within a few percent for arbitrarily large systems. Clearly, the two-process system causes the least interference. This follows from the fact that the two neighbors are actually the same process which assures that their requests occur simultaneously. The three-process system is worst because every process request affects directly all the remaining processes in the system.

The limiting case of \( r \rightarrow \infty \) deserves some attention. In this case, the internal service time is zero and the process states reduce to \( W \) and \( E \). (The possible alternate interpretation of infinite external service has little merit.) The permissible system states are then given by strings of length \( n \) which can be formed by concatenating only two types of substrings: \( WEW \) and \( EW \) (or, by symmetry, \( WE \)). Now, consider the pattern \( \cdot \cdot WEWWEW \cdot \cdot \) and assume that the second process from the left leaves the external state. The synchronization algorithm assures that its two neighbors will now enter the external state, and the zero internal service time puts the central process immediately into the wait state: \( \cdot \cdot EWWEW \cdot \cdot \). Thus, two adjacent \( WEW \) substrings are reduced to three \( EW \) substrings. On the other hand, the pattern \( \cdot \cdot WEWWEW \cdot \cdot \) is transformed into \( \cdot \cdot EWWWEW \cdot \cdot \), i.e., the order of two adjacent \( EWE \) and \( EW \) substrings is reversed. With \( r \rightarrow \infty \), reversals assure all possible reductions of \( WEW \)
Figure 6. Interference as a function of system size (algorithm A).
substrings and in equilibrium there can only be a substring of the form
WEW left if the system size is odd. In that case, the system states are
reduced to those in which \((n+1)/2\) processes are waiting and \((n-1)/2\) pro-
cesses are in the external state. The resulting interference values of
\((n+1)/(n-1)\) are indicated by the broken line in Figure 6. As \(r\) approaches
infinity in even-sized systems, the pattern consisting of EW substrings
will also become dominant. Then, half the processes are waiting and the
interference approaches 100%. For \(r = \infty\), however, half the processes (either
the even-numbered or the odd-numbered ones) will effectively seize the re-
sources. When a process leaves the external state, its immediate neighbors
are waiting on their second neighbors respectively, thus permitting the process
to immediately reenter the external state. In this extreme case, half the
processes experience no interference, while the other half is indefinitely
delayed. The system is in an absorbing state and the overall interference is
infinite. Figure 7 summarizes the interference characteristics as a function
of the service-ratio.

Figure 7 confirms the expectation that increased service-ratios result
in increased interference. It is an interesting anomaly that this need not be
the case over the entire range of \(r\). In Figure 8 we plot the interference
for \(n = 6\) as a function of \(r\) (logarithmic scale). As argued above, the
interference converges toward a value of 100%, but there is a flat maximum of
105.58% at \(r = 14.5\). This anomaly can be explained in terms of the fraction
of waiting processes as a function of \(r\). For small \(r\), few, if any, processes
wait. As \(r\) increases, more processes will wait and the occurrences of WEW
and EW substrings in the system state descriptions increase. Note that WEW
contributes most heavily to interference since two out of three processes are
waiting. With a further increase of \(r\) the state I becomes rare. The fre-
quency of EW substrings will continue to increase, but now at the cost of
WEW substrings which are being transformed. In this range, the slope of the

-19-
Figure 7. Interference in ring-structured systems under algorithm A.
Figure 8. The overshoot anomaly in a system of size six (algorithm A).
interference decreases, but it may still be large enough to overshoot the convergence value, especially if the system size is a multiple of both 2 and 3. Ultimately, of course, the EW substrings dominate and enforce the limiting values previously discussed. Due to the limited substring transformation possibilities, the overshoot anomaly does not occur when $n < 6$.

The detrimental effect of odd system sizes is particularly noticeable for large service-ratios. However, contrary to single-resource systems, interference remains finite. It reaches its maximum at 200% for $n = 3$. In this worst case, the relative costeffectiveness is still 33%. For small service-ratios, system size hardly affects interference as long as it is larger than 3. For $r < 1$ and $n > 3$, Figure 7 shows that $i < r/2$. Substituting this inequality in equation (7) shows that throughput is at least proportional to $n/(1+r/2)$. Furthermore, the relative costeffectiveness is at least $1/(1+r/2)$, and approaches 100% for $r \ll 1$. This value was to be expected, since we assumed that resource costs are distributed over the processes.

A Centralized Synchronization Algorithm

Algorithm A is highly distributed since synchronization decisions are based on the status variables of three processes only (the executing process and its two neighbors). As a consequence, it is reliable and efficient. A possible flaw lies in the occurrences of adjacent EW substrings in the system states with their detrimental effect on interference ($n/3$ processes may keep $2n/3$ processes waiting). Such occurrences can be avoided by a centralized algorithm, Algorithm B, which is given below.

We assume that there is an even number of processes which are numbered 0 through $n-1$ around the ring and capitalize on the fact that all even-numbered processes (or all odd-numbered ones) may receive external service concurrently. Synchronization is determined by the status variable phase. When phase $= 0 (1)$,
all processes in the even (odd) group have priority, i.e. they have their requests granted immediately. Phase will be switched when the last process of a group exits from state E. For efficiency reasons, we also permit a requesting process to adjust phase to its group when all other processes are in the internal state. To recognize this situation, \texttt{count[0]} (\texttt{count[1]}) keeps track of the number of even-numbered (odd-numbered) processes which are not in the internal state. For waiting processes in the even and odd group, two queues, \texttt{queue[0]} and \texttt{queue[1]}, respectively, are maintained.

\begin{algorithm}
\begin{algorithmic}
\Procedure{request}{i};
\State \texttt{if } (\texttt{count[0]}+\texttt{count[1]})=0 \texttt{then } \texttt{phase} := \texttt{imod2};
\State \texttt{count[imod2]} := \texttt{count[imod2]+1};
\State \texttt{if } \texttt{phase} \texttt{mod2} \texttt{then } \texttt{wait(queue[imod2])}
\EndProcedure
\State
\Procedure{release}{i};
\State \texttt{count[imod2]} := \texttt{count[imod2]-1};
\If {\texttt{count[imod2]}=0}
\State \texttt{begin}
\State \texttt{phase} := \texttt{(phase+1)mod2};
\State \texttt{for } \texttt{j} := \texttt{1 until count[(i+1)mod2]} \texttt{do}
\State \texttt{signal(queue[(i+1)mod2])}
\EndProcedure
\EndAlgorithm
\end{algorithmic}
\end{algorithm}

We define the system states \([h; j]\) in terms of the number of processes in state \(I\); \(h\) for those in the current priority group (determined by \texttt{phase}) and \(j\) for those in the non-priority group. Note that processes in the priority (non-priority) group can only be in states \(I\) and \(E\) (\(I\) and \(W\)). State \([n/2; n/2]\) denotes the situation where all processes are in state \(I\). The remaining states are given by \([h; j]\), \(0 \leq h \leq n/2-1\), \(0 \leq j \leq n/2\). Thus, the total number of systems states is \(1 + (n/2+1)n/2\).
For $n = 2$, there are three states: $[1; 1]$, $[1; 0]$, and $[0; 0]$. The corresponding transition-rate diagram is identical to that under algorithm A (c.f. Figure 2 with $n = 2$ in which state $[h; j]$ corresponds to state $[h+j]$). For $n = 2$, the distributed algorithm $A$ is actually identical to algorithm $B$ since the status variables of a process and its (only) neighbor completely describe the system resource state.

For $n = 4$, the seven system states are $[2; 2]$, $[1; 2]$, $[1; 1]$, $[1; 0]$, $[0; 2]$, $[0; 1]$, and $[0; 0]$. Again, the transition-rate diagram equals that of the distributed algorithm in Figure 5 where the states above correspond to states $[a]$ through $[g]$ respectively. While the distributed algorithm does not explicitly consider the state variable of the "opposite" process, the tight coupling of the four processes in the ring assures that two opposite processes can always get external service concurrently. Thus, any process in the external state defines the current priority group in the terminology of algorithm $B$. The two algorithms therefore generate identical service sequences for $n = 4$.

(The equality of the transition-rate diagram showed this already in a probabilistic sense since this diagram may be used to write down the time-dependent differential equations governing the dynamics of the system.)

For $n = 6$, there are 13 system states which are shown in the transition-rate diagram in Figure 9. Note that the transitions out of states $[2; j]$ ($h = 2$ implies that only one priority process is in the external state) which are labeled with rate $\pi$ indicate a phase switch. For comparison, the interference under both algorithms $A$ and $B$ is plotted in Figure 10 (algorithm $C$ will be discussed below). For small $r$, algorithm $A$ causes less interference due to its distributed nature. For typical applications (small values of $r$), it will generally be the superior algorithm. For large values of $r$, algorithm $B$ is shown to converge toward the common asymptote from below. By definition, algorithm $B$ is not susceptible to the overshoot anomaly.
Figure 9. The transition-rate diagram for six processes under algorithm B.
Figure 10. Interference for six processes under algorithms A, B, and C.
For arbitrarily large (even-sized) systems, the interference under algorithm B is quite comparable to that under algorithm A. While it is higher for small $r$ due to the global group notion, the difference is not excessive since the delays of non-priority processes are averaged with the zero delays of priority processes. When $r$ approaches infinity, the two algorithms behave identically: half the processes will be delayed for long periods of time while the other half experiences no delays. This results in the common asymptote $i = 100\%$. The absorbing system state at $r = \infty$ is also common to both algorithms.

A "Fair" Centralized Synchronization Algorithm

Like algorithm A, algorithm B is not fair in the sense that one process may cycle indefinitely while another one is waiting. Sometimes, such fairness considerations may not be important. For instance, if performance in a symmetric distributed system is to be measured in terms of throughput only, it does not matter which processes perform the work. On the other hand, this consideration is crucial when the progress of each process is to be assured. We proceed by presenting an algorithm, Algorithm C, which enforces an even distribution of cycles among competing processes.

Algorithm C makes use of the phase notion in algorithm B, but limits a process to one cycle per phase period, where a phase period is defined as a non-priority phase followed by a priority phase. Once a process exits from the external state, it is not permitted to reenter it until all requesting processes in the other group completed their external service. For synchronization purposes, we introduce the status variable $s[i]$ to reflect the request status of process $i$ during any phase period. Its five values have the following meaning:
<table>
<thead>
<tr>
<th>s[1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>process i has not left state I since the beginning of the phase period,</td>
</tr>
<tr>
<td>1</td>
<td>process i has made a request and is in state W because it is in a non-priority phase,</td>
</tr>
<tr>
<td>2</td>
<td>process i made a request which was honored and put it into state E,</td>
</tr>
<tr>
<td>3</td>
<td>process i left state E and reentered state I,</td>
</tr>
<tr>
<td>4</td>
<td>process i made a second request during the current phase period and is therefore in state W.</td>
</tr>
</tbody>
</table>

During a non-priority phase, the status of a process may change from 0 to 1. During a priority phase, the status may change from 0 to 2 to 3 to 4. As in algorithm B, the phase is switched when the last process of the current priority group leaves state E. (The number of priority processes in state E will be counted by the variable `count[0]`.) At this point, every priority process enters a new phase period and its status changes in the following manner: 0 to 0, 3 to 0, 4 to 1. No priority process can have status 1, and the phase will only be switched if no process has status 2. Waiting non-priority processes now gain priority and are unblocked which will ultimately result in a status change from 1 to 2. Two minor enhancements are added to improve efficiency. First, as in algorithm B, we permit a requesting process to adjust the phase to its group when all other processes are in the internal state (`count[0]` and `count[1]` will again count the number of processes in the even and odd group that are not in the internal state). Second, when the last process of the current priority group exits from state E and no non-priority processes are waiting, the phase is switched twice (or, equivalently, not switched at all). This permits processes with status 4 to proceed with their external service. Note that `count[g]` equals the number of waiting processes when g is the non-priority group. `queue[0]` and `queue[1]` will
again be used for the waiting processes in the even and odd group respectively.

Algorithm 0

procedure request(i);
begin
if (count[0] + count[1])=0 then phase := imod2;
count[imod2] := count[imod2]+1;
s[i] := s[i]+1;
if phase # imod2 or s[i]=4 then
wait(queue[imod2]);
s[i] := 2;
counte := counte+1
end;

procedure release(i);
begin
counte := counte-1;
s[i] := 3;
count[imod2] := count[imod2]-1;
if counte=0 then
begin
for j := imod2 step 2 until n-1 do
if s[j] > 3 then s[j] := s[j]-3;
if count[(i+1)mod2]=0 then
begin
phase := (phase+1)mod2;
for j := 1 step 1 until count[(i+1)mod2]
do signal(queue[(i+1)mod2])
end
else for j := 1 step 1 until count[imod2]
do signal(queue[imod2])
end
end

The assignment s[i] := 2 in procedure request is redundant, since counte already counts the number of processes with status 2. It was included to demonstrate how the program status variables can be used to define the system states in a uniform manner. We define these states in terms of the quadruples \([h_0, h_2, h_3, j_0]\) where \(h_0\), \(h_2\), \(h_3\), and \(j_0\) have the following meaning:

\(h_0\): number of priority processes with status 0,
\(h_2\): number of priority processes with status 2,
\[ h_3: \] number of priority processes with status 3,
\[ j_0: \] number of non-priority processes with status 0.

The number of waiting priority processes (status 4) equals \( n/2 - h_0 - h_2 - h_3 \).
Similarly, the number of waiting non-priority processes is \( n/2 - j_0 \). This
encoding of the system states aids in the construction of the transition-rate
diagrams since exactly \( h_2 \) and \( h_0 + h_3 + j_0 \) processes are in states 1 and
respectively. Thus, the flow rate out of state \([h_0, h_2, h_3; j_0]\) equals
\[ n h_2 + \tau (h_0 + h_3 + j_0) \]. Waiting processes, on the other hand, do not contrib-
tute to the flow rate out of a state. The state \([n/2, 0, 0; n/2]\) describes
the situation where all processes are in the internal state. The remaining
states are characterized by \([h_0, h_2, h_3; j_0]\), \( 1 \leq h_2 \leq n/2 \),
\( 0 \leq h_0 + h_3 \leq n/2 - h_2 \), \( 0 \leq j_0 \leq n/2 \). Summing over these ranges yields a total
of \( 1 + (n/2)(1 + n/2)^2(2 + n/2) / 6 \) system states.

For the two-process system under algorithm C, the state space consists
of the three states \([100; 1]\), \([010; 1]\), and \([010; 0]\). Not surprisingly,
the transition-rate diagram is again identical to the one in Figure 2 with \( n = 2 \).
State \([h_0, h_2, h_3; j_0]\) here corresponds to state \([h_0 + h_3 + j_0]\) there. For
the small configuration of \( n = 2 \), the algorithms A, B, and C result in
identical synchronization sequences.

The 13-state transition-rate diagram for \( n = 4 \) is depicted in Figure 11.
A transition out of a state with \( h_2 = 1 \) and labelled with rate \( r_i \) corresponds
to a phase switch. If, in addition, \( j_0 = 2 \), the transition indicates a
double-switch (no processes in the non-priority group are waiting). Interfer-
ence values for this system (as well as those with \( n = 2 \) and \( n = 6 \) are
plotted in Figure 12 as a function of the service-ratio (logarithmic scale).
For large values of \( r \), the interference approaches 200%. We shall further
discuss this figure below.
Figure 11. The transition-rate diagram for four processes under algorithm C.
For \( n = 6 \), the transition-rate diagram contains \( 41 \) states. It adds little more insight and is therefore not shown. The resulting interference values are plotted in Figure 10 together with those under algorithms A and B. They amply demonstrate the price at which the fair treatment of all processes was obtained. The convergence value for \( r \) approaching infinity is \( 8/3 \) and will be justified below.

The effect of system size on interference under algorithm C is shown in Figure 12. In comparison with \( n = 4 \), the six-process system exhibits less interference for small values of \( r \). To explain this, consider the state where a single process in the priority group keeps all other \( n - 1 \) processes waiting. The effect of this worst case on interference will grow with the fraction of waiting processes, i.e. \((n-1)/n\), but also the fraction of time the system is in this state. For small values of \( r \), the latter decreases much faster than \((n-1)/n\) grows. Except for \( n = 2 \) (where the former effect dominates), the interference for small values of \( r \) will thus decrease with system size.

Quite the opposite holds for large service-ratios. When \( r \) approaches infinity, individual processes can only be in states E or W. Thus, the system state space reduces to states of the form \([0, h, 0; 0]\), \(1 \leq h \leq n/2\). Figure 13 depicts the degenerate transition-rate diagram for \( r = \infty \). Let \( P_h \) be the equilibrium probability of finding the system in state \([0, h, 0; 0]\). From flow conservation we have \( h \cdot P_h = \text{constant}, 1 \leq h \leq n/2\), and solving this set of equations under the probability conservation constraint yields

\[
P_h = \frac{1/h}{\sum_{j=1}^{n/2} (1/j)}, \quad 1 \leq h \leq n/2.
\]

(16)

In state \([0, h, 0; 0]\), there are \( n-h \) waiting processes \((n/2 - h)\) of them in the priority group. Due to symmetry, the marginal process wait state probability is therefore
Figure 12. Interference in ring-structures under algorithm C.
Figure 13. The degenerate transition-rate diagram under algorithm C for infinite service-ratios.
\[ P(H) = \sum_{h=1}^{n/2} \frac{((n-h)/n)}{p_h} = 1 - 1/[2 \sum_{j=1}^{n/2} (1/j)]. \]  

(17)

Substituting this equation in the defining equation (5) we get

\[ i = 2 \left[ \sum_{j=1}^{n/2} (1/j) \right] - 1 \]  

(18)

for the interference when \( r \) approaches infinity. For \( n = 2, 4, 6 \) this evaluates to \( 1, 2, 8/3 \) which confirms the results obtained by solving the set of linear equations numerically (c.f. the convergence values in Figure 12). For large values of \( n \) the series in equation (18) approaches

\[ \sum_{j=1}^{n/2} (1/j) = \gamma + \log(n/2), \quad n \gg 1, \]  

(19)

where \( \gamma \) is Euler's constant (\( \gamma = 0.5772 \)) [8]. The interference for very large systems and service-ratios,

\[ i = 2\gamma - 1 + 2\log(n/2), \quad n \gg 1, \]  

(20)

is thus shown to grow with the logarithm of \( n \). This characteristic of algorithm C which enforces a fair allocation of cycles to competing processes is in sharp contrast to that of algorithm A and B under which the interference approaches a constant value of 100%. Substituting equation (20) in the defining equations (7) and (8) shows that throughput increases proportional to \( n/\log(n) \), rather than \( n \), and that the relative costeffectiveness decreases with \( 1/\log(n) \). This poor performance of algorithm C is not a consequence of insufficient system resources, but follows from the fairness strategy which may cause a process to wait even when the requested resources are available. This is analogous to the performance deterioration which results when the progress of
one process depends on that of other processes as is the case in producer/consumer relations [5].
V. CONCLUSION

The synchronization of asynchronous processes with respect to exclusive accesses of shared resources, like memory and communication facilities, causes a deterioration of system performance. We modeled symmetric distributed systems as collections of cyclic processes which compete for subsets of the resources and defined a measure, the interference, to quantify this deterioration. A process cycle consists of an exponentially distributed internal service time, and an exponentially distributed external service time during which the resources are accessed. The parameters of the model include the number of processes, the number of resources, the service-ratio (the ratio of the average external and internal service time of a process), the request topology which specifies the subsets of resources which processes require for their external service, and the synchronization algorithm. It was shown that system throughput and cost-effectiveness can be expressed in terms of the interference.

Two types of topologies were investigated. The single-resource system, in which all processes compete for the same set of resources, represents a birth-death process for which closed-form expressions of performance measures are readily obtained. Interference characteristics for this system type were plotted and discussed. For more sophisticated topologies, the system behavior is described by a Markov chain whose state space depends on all parameters of the model. Equilibrium solutions are obtained by solving the corresponding system of simultaneous linear equations. For large system sizes, the formulation of these equations tends to become impractical. However, solutions for smaller systems often display a convergence pattern which indicates the characteristics of large systems. In certain cases, these characteristics can be confirmed by direct analysis of the limiting case. We illustrated the general approach for a ring-structured system whose topology corresponds to the scenario of the

-30-
Dining Philosophers [4,6]. For this topology, we defined and analyzed a distributed, a centralized, and a "fair" synchronization algorithm. We plotted the resulting interference as a function of the number of processes, the service-ratio, and the three algorithms. The characteristics of the three algorithms were compared, and the effects of large service-ratios and system sizes were discussed.

The programming of correct and efficient synchronization algorithms is one of the most challenging activities in the design of a distributed system. Although there are now a number of powerful synchronization mechanisms available, correctness proofs are difficult because they depend on the assumptions made about the behavior of the asynchronous processes. However, the technique of using well-defined status variables which formally describe the system state greatly facilitates program verification. In essence, the system is then modeled as a finite-state machine where the transitions correspond to executions of the synchronization procedures. Labeling these transitions with the appropriate rates results in the transition-rate diagrams of the Markov chains presented here. Thus, the technique suggested for program verification can readily be extended to evaluate the performance of a synchronization algorithm.

ACKNOWLEDGEMENTS

The author wishes to thank Giorgio Ingargiola for several motivating discussions concerning the evaluation of synchronization algorithms, the Computer Science Division of the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley for its hospitality during the Spring Quarter of 1979 when most of this work was done, and Bence Gerber for his valuable suggestions regarding the computational efforts.
REFERENCES


