FAST ALLOCATION OF NEARBY RESOURCES IN A DISTRIBUTED SYSTEM*

Nancy A. Lynch
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

1. Introduction

Dijkstra's informally-stated Dining Philosophers problem [D] involves a number n of philosophers sitting in a circle, a single fork between each pair of adjacent philosophers. Any philosopher may decide to eat at any time and requires both of his forks to do so, but he can only "pick up" one fork at a time. Philosophers act asynchronously. The problem is to program the philosophers in ways which guarantee certain conditions of fairness and absence of deadlock. (For instance, if everyone picks up his left fork first and then waits for his right fork to become free, the system can deadlock with a circular chain of waiting philosophers.)

Assuming no means of communication among philosophers other than through information attached to their forks, it is easy to formalize an argument that any solution in which all philosophers are programmed identically must have a possibility of deadlock. Thus, (with this assumption), correct solutions must allow some distinction to be made among the philosophers.

Because of the imprecision of the problem statement, it is not clear how to evaluate and compare various solutions in the literature. For example, Chang [C] presents solutions which distinguish certain philosophers with responsibility for breaking deadlocks. He argues that deadlock is infrequent, so special protocols to avoid deadlock should not be permitted to add running time overhead to normal operation of the system. Thus, his system runs with no special constraints - for example, any philosopher may pick up either fork first. When deadlock occurs, a special recovery protocol is executed. No meaningful analysis is given to support the alleged time efficiency, however.

Intuitively, a difficulty with Chang's solution is that certain time inefficiencies can occur even in the absence of deadlock. Even if a complete circular chain of waiting philosophers is not produced, it can still be the case that a long chain occurs, of philosophers waiting for their neighbors. Such a chain seems to require considerable time to break.

James Burns [B] has suggested the following alternative, simple algorithm (which is also known to other researchers in the area). Alternate philosophers are distinguished as "L" and "R" philosophers (assuming n is even). The L and R philosophers are constrained to pick up their left and right fork first, respectively. It is easy to argue by case-analysis that deadlock does not occur. It is also intuitively plausible that this solution is more time-efficient than Chang's, because waiting chains of length greater than 2 never occur. However, no formal time analysis is presented in [B] to support this intuition.

In this paper, the problem is generalized to a distributed system resource allocation problem which is local in two senses. First, although the system and number of users can be very large, there is a limit to the overlap in resource demands of different users. The second condition can be thought of as a property of the geography of the network - the resources are (or can be) located in the network in such a way that communication between a user and any of its required resources is fast. Both types of locality conditions are satisfied by the Dining Philosophers problem. Under these two conditions, one would hope that waiting chains could be avoided, so that the worst-case time to grant a user's requests is independent of the total size of the network and the total number of users.

In order to prove theorems to this effect, one must state precisely the problem to be solved, and describe an implementation in an unambiguous model. Time complexity analysis should be done using realistic and precisely-defined measures. The simple automata-theoretic model developed in

*This research was supported in part by the National Science Foundation under grants MCS77-15628 and U.S. Army Research Office Contract Number DAAG29-79-C-0155.

[LF] is used in this paper to describe (separately) the generalized problem and one solution. Basically, a particular order is specified for each user to wait for his resources, generalizing the "left-right" idea in a natural way. Then a version of the time measure described in [PF] and [P] is used to perform time analysis. Theorem 8.1, the main result, gives an upper bound (independent of the network size and total number of users) on the worst-case time for a request to be granted. Although Theorem 8.1 gives the needed independence of network size, in many interesting cases the results actually seem sharper than those yielded by Theorem 8.1. A second result, Theorem 10.1, is proved, giving sharper bounds for some special cases.

It can be argued that worst-case time for a request to be granted is not the only valid measure of the time efficiency of an asynchronous resource allocation algorithm. Other useful measures include worst-case "throughput" and worst-case time under various restrictions on the use of the system (e.g. limited number of concurrent active requests). Some arguments about these other measures are sketched in Section 11.

The contributions of this paper are twofold. First, the simple resource-waiting strategy may be of some practical interest. Second (and more importantly), the time complexity analysis methods seem to be tractable and useful tools for analyzing distributed system designs. There is no averaging done over inputs to the system; thus, the probabilistic techniques and the approximations of queuing theory are not used. The analysis of worst-case throughput and other "rates" can be considered to be a type of averaging, but the averaging is done over the course of a single execution, not over alternative executions; it is, therefore, much simpler. The style of complexity analysis is quite similar to the kind of analysis usually performed for ordinary (sequential) complexity theory, with heavy use of recurrence-equation techniques. Problem statements are more complicated in the new setting, and analysis is generally done in terms of many parameters of the system. However, the most obvious difference in the new setting is the need, imposed by the asynchronism of the system, to pinpoint the bottlenecks at each point in the computation. I cannot yet claim that a set of "techniques" for time analysis of asynchronous systems has been developed; rather, analysis has been performed for some particular systems. However, some general principles seem to be emerging and should become crystallized in the process of analyzing a sufficient variety of asynchronous systems.

2. Resource Problems

A resource problem P is a quadruple (R(P), U(P), $\mathcal{R}(P)$, $\mathcal{U}(P)$), where R(P) and U(P) are disjoint, possibly infinite sets (of "resources" and "users" respectively), where $\mathcal{R}(P)$ is a mapping from U(P) to the set of finite nonempty subsets of R(P) (indicating the resources required by each user), where $\mathcal{U}(P)$ is a mapping from R(P) to finite nonempty subsets of U(P) (indicating the users for each resource), and where $r \in \mathcal{R}(P)(u)$ if and only if $u \in \mathcal{U}(P)(r)$.

Let rcommon(P) = $\{(u,u') \in (U(P))^2 : \mathcal{R}(P)(u) \cap \mathcal{R}(P)(u') \neq \phi\}$, and ucommon(P) = $\{(r,r') \in (R(P))^2 : \mathcal{U}(P)(r) \cap \mathcal{U}(P)(r') \neq \phi\}$. Let graph(P) be the graph with R(P) as its node set and ucommon(P) as its edge set. Let contention(P) denote $\max_{r \in R(P)} |\mathcal{U}(P)(r)|$. A coloring of P is a total mapping $c: R(P) \to N$ satisfying the condition "$(r,r') \in$ ucommon(P) implies $c(r) \neq c(r')$". Let $|c|$ denote the largest number in the range of c. Let colors(P) denote the minimum value of $|c|$ for any coloring c of P. (Thus, colors(P) is the chromatic number of graph(P).) Define $first_{P,c}: U(P) \to R(P)$ by

$first_{P,c}(u) = r$ such that $r \in \mathcal{R}(P)(u)$ and $(\forall r' \in \mathcal{R}(P)(u))[c(r') \geq c(r)]$.

Define $next_{P,c} : U(P) \times R(P) \to R(P)$ by

$next_{P,c}(u,r) = r'$ such that $r' \in \mathcal{R}(P)(u)$ and $(\forall r'' \in \mathcal{R}(P)(u))[c(r'') \leq c(r)$ or $c(r'') \geq c(r')]$.

($First_{P,c}$ and $next_{P,c}$ list the resources of each user in increasing order. $First_{P,c}$ is total, while $next_{P,c}$ is partial. They are well-defined since c is injective on $\mathcal{R}(P)(u)$ for each u.)

The first locality condition mentioned in the Introduction, the limit to overlap in resource demands, is captured formally by the bounds contention(P) and colors(P).

Ex. 2.1. Dining Philosophers

$R(P) = \{r_1,\ldots,r_n\}$, $U(P) = \{u_1,\ldots,u_n\}$, $\mathcal{R}(P)(u_i) = \{r_i, r_{i+1 \bmod n}\}$ and $\mathcal{U}(P)(r_i) = \{u_{i-1 \bmod n}, u_i\}$.

Graph(P) is a cycle with vertices r_i, $1 \leq i \leq n$.

Contention(P) = 2.

Let $c(r_i) = i$, and $c'(r_i) =$
- 1 if n is even and i is odd, or if n is odd and $i < n$ is odd,
- 2 if i is even,
- 3 if n is odd and $i = n$.

Thus, c provides a linear order for the resources, while c' provides a partial order of depth ≤ 3. In either case, resources with common users are comparable. However, c' is minimum in the sense that

$|c'| = colors(P) =$
- 2 if n is even,
- 3 if n is odd.

Ex. 2.2. k-Fork Philosophers

Let R(P) and U(P) be as in Ex. 2.1, $\mathcal{R}(P)(u_i) = \{r_i, r_{i+1 \bmod n}, \ldots, r_{i+k-1 \bmod n}\}$ and $\mathcal{U}(P)(r_i) = \{u_{i-k+1 \bmod n}, \ldots, u_{i-1 \bmod n}, u_i\}$.

Contention(P) = k. Colors(P) = k + 1 (if $n \geq k^2$).

Ex. 2.3. 2-Dimensional Philosophers

The resource requirement pattern in a distributed system might not have a 1-dimensional

structure such as those of Ex. 2.1 and Ex. 2.2. As a simplified example of a 2-dimensional pattern, let $R(P) = \{r_i: i = (i_1,i_2) \in Z^2$ and $i_1 + i_2$ is even$\}$, $U(P) = \{u_i: i = (i_1,i_2) \in Z^2$ and $i_1 + i_2$ is odd$\}$, and $R(P)(u_i) = \{r_j: \sum_{\ell=1}^{2} |j_\ell - i_\ell| = 1\}$. Contention(P) = colors(P) = 4. Graph(P) is a diagonal grid.

The remaining two examples will be used later to demonstrate situations in which our algorithm approaches its upper bound; they do not describe "local" resource requirement patterns for which the algorithm is well suited.

Ex. 2.4. k-Tree

Let A be an alphabet of k elements, a any distinguished element of A. Let $R(P) = \{r_i: i \in A^*, |i| < k\}$, $U(P) = \{u_i: i \in A^k\}$, and let $U(P)(r_i)$ consist of all $u_{ia'a^{k-|i|-1}}$ for all $a' \in A$. The resources can be envisioned as forming a tree. For instance, if k = 3, A = {1,2,3} and a = 1, the resources form a 3-ary tree, with users as indicated in Figure 1.

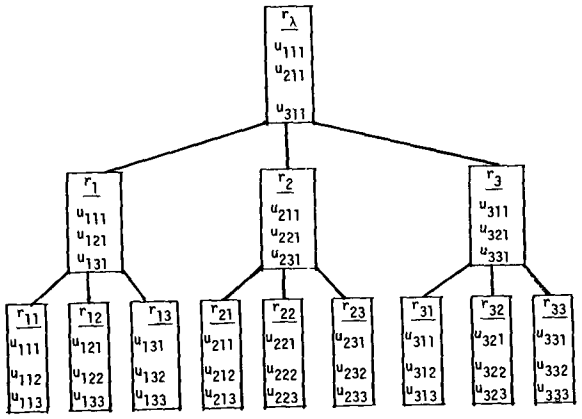


Figure 1

Contention(P) = k and colors(P) = k. (Letting $c(r_i) = |i| + 1$ shows colors(P) $\leq$ k. Colors(P) cannot be less than k because $|R(P)(u_{a^k})| = k$.) Graph(P) includes the tree as a subgraph.

Ex. 2.5. k-Nested Sets

$R(P) = \{r_i: 1 \leq i \leq k\}$, $U(P) = \{u_i: 1 \leq i \leq k\}$, and $U(P)(r_i) = \{u_j: j \leq i\}$. Contention(P) = colors(P) = k, and graph(P) is the complete graph on k vertices.

3. A Model for Distributed Systems and Their Behavior

We use the model of [LF] to describe the problem and our algorithm. The reader is referred to [LF] for a completely formal description of the model. In this paper, we will be less formal; however, it is straightforward to express our informal conditions as precise conditions within the model, following the examples in [LF].

Briefly, the basic entities of the model in [LF] are processes (automata) and variables (for communication). An atomic execution step of a process involves accessing one variable and possibly changing the process' state or the variable's value or both ("test-and-set"). Processes are able to respond in some way to anything that they might find in a variable. A system of processes is a set of processes, with certain of its variables designated as internal and the others as external. Internal variables are to be used only by the given system. External variables are assumed to be accessible to other processes (or other external agents) which can change the values between steps of the given system.

The execution of a system of processes is described by a set of execution sequences. Each sequence is a (finite or infinite) list of steps which the system could perform when interleaved with appropriate actions by the external agent.

For the purpose of describing the external behavior of a system of processes, certain information in the execution sequences is irrelevant. The external behavior of a system of processes is the set of sequences derived from the execution sequences by "erasing" information about process identity, changes of process state and accesses to internal variables. What remains is information about accesses to external variables.

A distributed problem is any set of sequences of accesses to variables. A system is said to solve the problem if its external behavior is any subset of the given problem.

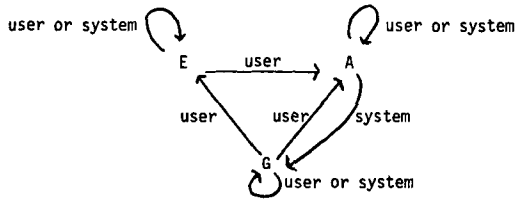
One method for specifying a distributed problem [LF] is to describe first the set of allowable sequences of accesses to the external variables by the user and the system together, tagging each access by the label "user" or "system" as appropriate, second the assumed external behavior of the user (environment) of a system, and third the initialization of the external variables. Then a sequence of system accesses to external variables is acceptable provided when it is interleaved consistently with a correct user sequence, the resulting sequence is correct for the user and system together. The distributed problem is the set of acceptable sequences, and a system solves the problem if all of its external behavior sequences are acceptable.

4. External Behavior Description for Resource Allocation Systems

The model in [LF] is best suited for specification of interface behavior of systems and their components, rather than the "eating and thinking region" behavior described by Dijkstra. Direct formalization of region behavior for the general resource problem does not seem to be particularly natural. Therefore, we formulate the problem in terms of external behavior. (If one wants bounds similar to those in this paper for region behavior, one should proceed as in [FLBB] to construct systems of the required type which "simulate" those described in this paper. This does not appear to pose any serious difficulty, but seems detailed and tedious.)

For any resource problem P, the interface description for the needed system is as follows. (We follow the specification method described in Section 3.)

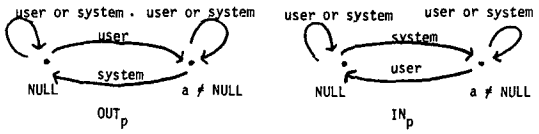
For each $u \in U(P)$, there is an external variable EXT_u, having values 'E'(empty), 'A'(ask) and 'G'(grant). Allowed transitions at any EXT_u are as in the following diagram:



If the value of any EXT_u stops changing, then the final value is 'E'. Finally, if two variables, EXT_u and $EXT_{u'}$, are ever simultaneously equal to 'G', then $(u,u') \notin rcommon(P)$. A "correct user" is one that only makes allowed transitions and does not leave the variable at 'G'. All variables are initialized at 'E'. The set of sequences of system accesses to the external variables which combine consistently with correct user sequences to yield behavior satisfying the interface description comprises the distributed problem to be solved.

5. Geographical Considerations

There are many different solutions (within the [LF] model) to the distributed problem described above. However, in a distributed environment, there are geographical constraints in addition to interface requirements. These constraints involve number and location of processes, connectivity and communication time. For the problem at hand, we constrain the solution to consist of user processes, one accessing each external variable, and (a disjoint set of) resource processes, one for each resource. For notational convenience, these processes are identified with the elements of $U(P) \cup R(P)$. These processes need to communicate, but we do not want to study the means of communication in this paper. Therefore, we assume another interface exists between all of these processes and a message system. Each process p communicates with the message system by two variables IN_p and OUT_p. If M is a message alphabet, then the values of IN_p are $M \cup \{'NULL'\}$, while the values of OUT_p are $M \times 2^{R(P)(p)} \cup \{'NULL'\}$ if $p \in U(P)$ and $M \times (U(P)(p) \cup R(P)(U(P)(p)) \cup \{'NULL"\}$ if $p \in R(P)$. (Intuitively, the second component of OUT values is an address or set of addresses. User processes can simultaneously broadcast the same message to all associated resource processes, while resource processes can only send messages one at a time to associated processes.) Allowed transitions are:



(Here, the tag "system" refers to the message system.) If the value of any variable stops changing, the final value is 'NULL'. Finally, messages "get delivered" - in any execution sequence, the "writes" by the message system to any IN_p variable must be of message values which are some permutation of the message values "read" (i.e. changed to 'NULL') by the system from the OUT variables, addressed to process p. (We do not specify any order for delivery, nor do we care how the message system operates.) A "correct user" of the message system is one that only makes allowed transitions and does not leave its input variable $\neq$ 'NULL'. All variables are initialized at 'NULL'.

The reason for this particular choice of message system interface is that it seems to be the minimum natural interface needed to make our solution work as efficiently as it should. We have made this interface description part of the given conditions on the problem solution. The reader might conceive of an alternate resource allocation strategy which uses a different message interface; such a strategy would not be directly comparable to our solution. In order to compare two solutions with different message interfaces, the two problem-solvers must agree on a common basic interface out of which the two different interfaces can be built, and then complexity analysis must be done relative to the common interface.

The formal correspondence between resources and resource processes is not intended to imply that the resources must be "located at" or "controlled by" the corresponding processes. The space of allowable solutions includes solutions in which control over the granting of a resource is shared by many different resource processes. It also includes solutions in which one resource process controls many resources. (In the solution of this paper, however, the resources are controlled by their respective processes.)

6. Time Measure

The time measure of [PF,P] hypothesizes upper bounds (but no lower bounds) on the time for certain events to occur during execution. (Thus, the set of possible execution sequences is not restricted in any way.) These upper bounds are used to infer upper bounds on the time for certain other events to occur. Let P denote a fixed resource problem. Let *S*, *M*, and *V* denote arbitrary implementations within our model of a correct resource allocation system, a correct message system and a correct user for a resource allocation system, respectively, for P. (Correctness for *M* and *V* involves interface behavior only, while correctness for *S* implies also that its set of processes is U(P) ∪ R(P) with variable access capabilities as described in Section 5.) Let *C* be the system constructed by combining *M*, *S* and *V*. (In the notation of [LF], the combined system is $consist_{Y,f}(S \oplus M \oplus V)$, where

$Y = \{EXT_u: u \in U(P)\} \cup \{IN_p: p \in U(P) \cup R(P)\} \cup \{OUT_p: p \in U(P) \cup R(P), f(EXT_u) = \text{'E' for all } u \in U(P) \text{ and } f(IN_p) = f(OUT_p) = \text{'NULL' for all } p \in U(P) \cup R(P)\}$.) Let e denote an execution sequence of *C*.

Let sent(e,i,a,p,p') denote the number of times message 'a' is placed in OUT_p, addressed to process p' (including broadcasts in which p' is included among the addressees), in execution sequence e up to and including step i. Let collected(e,i,a,p,p') be the number of times changes are made in variable OUT_p from values in which message 'a' is addressed to process p', up to and including step i. Let sentfrom(e,i,a,p) denote $\sum_{p'}$ sent(e,i,a,p,p'), and similarly for collectedfrom(e,i,a,p). Let sentfrom(e,i,p) denote $\sum_{a}$ sentfrom(e,i,a,p), and similarly for collectedfrom(e,i,p). Let sentto(e,i,a,p) denote $\sum_{p'}$ sent(e,i,a,p',p) and similarly for collectedto(e,i,a,p). Let deliveredto(e,i,a,p) be the number of times a transition to 'a' is made in IN_p, up to and including step i. Let sentto(e,i,p) denote $\sum_{a}$ sentto(e,i,a,p), and similarly for collectedto(e,i,p) and deliveredto(e,i,p). Finally, let sent(e,i) denote $\sum_{p}$ sentto(e,i,p), and similarly for collected(e,i) and delivered(e,i).

Let requests(e,i,u) (resp. grants(e,i,u), returns(e,i,u)) denote the number of changes to 'A' (resp. to 'G', from 'G*') at variable EXT_u in execution sequence e, up to and including step i. Let $requests(e,i,V) = \sum_{u \in V} requests(e,i,u)$ for $V \subseteq U(P)$, and similarly for grants(e,i,V) and returns(e,i,V). Let requests(e,i) = requests(e,i,U(P)) and similarly for grants(e,i) and returns(e,i).

Let R^+ represent the nonnegative reals. A timing is a nondecreasing total mapping $t: N \to R^+$. Let $A = (\sigma,\upsilon,\gamma,\delta) \in (R^+)^4$, and let t be a timing. Then t is A-admissible for e provided (a) - (d) hold.

(a) σ is an upper bound on process step time.

Let $p \in U(P) \cup R(P)$ and let the execution steps involving actions of process p be $p_1, p_2, \ldots$. Then $t(p_1) \leq \sigma$ if p_1 exists. Also, $t(p_{i+1}) - t(p_i) \leq \sigma$ for each i for which p_i and p_{i+1} are both defined.

(b) υ is an upper bound on time for a user to return a granted resource.

If grants(e,i,u) = k, if returns(e,j,u) = k and returns(e,j-1,u) < k, then t(j) - t(i) ≤ υ.

(c) γ is an upper bound on message collection time.

If sentfrom(e,i,p) = k, if collectedfrom(e,j,p) = k and collectedfrom(e,j-1,p) < k, then t(j) - t(i) ≤ γ.

(d) δ is an upper bound on message delivery time.

If sentto(e,i,a,p) = k, if deliveredto(e,j,a,p) = k and deliveredto(e,j-1,a,p) < k, then t(j) - t(i) ≤ δ.

The second, "geographical" locality condition mentioned in the Introduction is captured formally by the bound δ. δ is to be thought of as much smaller than the worst-case transmission time for a message system that could send messages between all processes in the entire distributed network.

We now define the "worst-case response time" to be measured. Let $T_C(A,u)$ denote the supremum, for all execution sequences e of combined system C and all A-admissible timings t for e, of the quantity t(j) - t(i), where requests(e,i,u) = k, where grants(e,j,u) = k and grants(e,j-1,u) < k. Let $T_C(A)$ denote $\sup_u T_C(A,u)$.

7. The Solution

We consider solutions in which each resource process maintains a FIFO queue of waiting users. It is easy to see [G] that deadlock is prevented in a distributed resource allocation system if the resources are linearly ordered (say by ≤), if each user waits on queues for all of his resources in increasing order of resources, if he only waits for one resource at a time (i.e. until reaching the front of the associated queue), and if he remains on all queues until he is first on all of them. In fact, if all granted resources are eventually returned, it is clear that each user eventually obtains all of his resources.

On closer examination, we note that a linear ordering of resources is unnecessary. A partial ordering suffices, provided any two

resources required by the same user are comparable. Any coloring c of P specifies such a partial ordering. We use an arbitrary coloring c and the strategy described above in our solution. The complete code for user and resource processes appears at the bottom of the page.

The high-level language used is (almost) the same as that used in [CH,FLBB]. Computation occurring within a lock-unlock pair occurs within a single execution step in the formal model. In the formal model, every step involves access to a variable. The local computation appearing in our language is combined into the previous lock-unlock pair in the formal model. (In [FLBB], this computation was combined into the following lock-unlock pair, an alternative which would change the complexity analysis of our algorithm slightly.) The construct "waitfor (condition);" is an abbreviation for "A:lock; if (¬ condition) then [unlock;goto A];". Subscripts are omitted from EXT, IN and OUT variables.

It is easy to see that deadlock is avoided by this solution, and that each user eventually obtains his resources (provided all granted resources are eventually returned). In addition, if $|c|$ is small, this solution appears to limit the lengths of chains of waiting processes, thereby providing an upper bound on running time. The remaining sections prove results to this effect.

8. Worst-Case Performance

We obtain a general theorem giving an upper bound on performance of our solution, which is not directly dependent on total network size or total number of users. Let C(P,c) denote the combined system composed of our solution for resource problem P using coloring c, and any arbitrary correct message system and correct resource system user. Let $A = (\sigma, \upsilon, \gamma, \delta)$.

Theorem 8.1. $T_{C(P,c)}(A) \leq (\text{contention}(P)^{|c|} - 1)\upsilon + O(|c|\text{contention}(P)^{|c|}(\sigma + \gamma + \delta))$.

Proof. Since processes operate asynchronously, it is generally the case during execution that some parts of the system are waiting for work to be accomplished by other parts. (The waiting processes might be busy-waiting, or might be performing a considerable amount of work.) In the analysis, it is crucial that the key parts of the system be identified at each time during execution.

Classify the resource processes into levels, each resource process r at level c(r). For $1 \leq i \leq |c|$, $1 \leq j \leq$ contention(P), let $G_{i,j}$ denote the supremum, over all execution sequences e and A-admissible timings t, of the time from when any user u reaches position j from the front of a level $\geq i$ resource process QUEUE, until the resources are next granted to u. A system of recurrences is obtained.

First, consider arbitrary i and $j \geq 2$. Let i_1 be an execution step in which u reaches position j from the front of the QUEUE of a level $\geq i$ process, r. Let u' be u's immediate predecessor on r's QUEUE. By induction, within time at most $G_{i,j-1}$ from $t(i_1)$ (as measured by timing t), u' is granted his resources. Then within time at most υ, u' returns the resources (because of A-admissibility), and then within time σ, user process u' detects the return. Also, within time γ of $t(i_1)$, the value $(u', \{first_{P,c}(u')\})$ arising from this u' request is removed from $OUT_{u'}$. Thereafter, within time σ, u' broadcasts a 'RETURN' message, and then within δ the

Code for user process u

local: STATUS init 'E'

```
do forever
   if STATUS = 'E'
   then [lock; if EXT = 'A' then STATUS := 'A'; unlock]
   else [waitfor (OUT = 'NULL'); OUT := (u,{first_P,c(u)}); unlock;
         waitfor (IN = 'G'); IN := 'NULL'; unlock;
         lock; EXT := 'G'; unlock;
         waitfor (EXT ≠ 'G'); STATUS := EXT; unlock;
         waitfor (OUT = 'NULL'); OUT := ('RETURN' , R(P)(u)); unlock];
```

Code for resource process r

local: QUEUE init ∅, MSG init 'NULL', J init 'NULL', STATUS init 'E'

```
do forever
   lock; if IN ≠ 'NULL' then [MSG := IN; IN := 'NULL']; unlock;
   if MSG ε U then append MSG to QUEUE;
   if MSG = 'RETURN' then delete front of QUEUE;
   if MSG ε U and |QUEUE| = 1 or MSG = 'RETURN' and |QUEUE| ≥ 1
   then [STATUS := 'A'; J := front(QUEUE)];
   if STATUS = 'A'
   then [lock; if OUT = 'NULL' then OUT := if next_P,c(J,r) is defined then
      (J,next_P,c(J,r)) else ('G',J); unlock; STATUS := 'E'];
   MSG := 'NULL';
```

'RETURN' reaches IN_r. Within σ, the 'RETURN' is read by r and u' is removed from r's QUEUE, making u first. Then within $G_{i,1}$, u is granted his resources. We see that $G_{i,j} \leq \max(\gamma, G_{i,j-1} + \upsilon + \sigma) + 2\sigma + \delta + G_{i,1}$, for $j \geq 2$.

Next, consider $i \geq 2$ and $j = 1$. Consider an execution step in which u reaches the front of the QUEUE of a level $\geq i$ process, r. Within time $\gamma + 2\sigma$, a value ('G', u) or $(u, next_{P,c}(u,r))$ is placed in OUT_r. If the value is ('G', u), then within $\delta + 2\sigma$, u is granted his resources. If the value is $(u, next_{P,c}(u,r))$, then within $\delta + 2\sigma$, u is appended to the QUEUE of a level $\geq i + 1$ process, r'. At that moment, u is in position $\leq$ contention(P) + 1 on r' 's QUEUE; each contender for resource r' can appear at most once, with the single exception that the first user on r' 's QUEUE might appear twice. (This is because the first user might have his resources granted, return them, and then request them again. The new request might arrive at r' before the 'RETURN' message.) However, within time $\delta + \sigma$, u reaches a position $\leq$ contention(P) on r' 's QUEUE. Then within $G_{i+1,contention(P)}$, u is granted his resources. We see that $G_{i,1} \leq \gamma + 2\sigma + \max(\delta + 2\sigma, \delta + 2\sigma + \delta + \sigma + G_{i+1,contention(P)})$, $i < |c|$. That is, $G_{i,1} \leq \gamma + 2\delta + 5\sigma + G_{i+1,contention(P)}$, for $i < |c|$.

Next, note that $G_{|c|,1} \leq \gamma + 2\sigma + \delta + 2\sigma = \gamma + \delta + 4\sigma$.

Finally, consider $T_{C(P,c)}(A)$. Let i_1 be an execution step in which u makes a request. Within time σ from $t(i_1)$, the request is detected by user process u. Also, within time γ from $t(i_1)$, OUT_u becomes 'NULL'. Thus, within time $\max(\gamma, \sigma) + \sigma$ of $t(i_1)$, the value $(u, \{first_{P,c}(u)\})$ is placed in OUT_u. Then (as above) within $\delta + 2\sigma + \delta + \sigma$, u reaches position $\leq$ contention(P) on some QUEUE. Thus, $T_{C(P,c)}(A) \leq \max(\gamma, \sigma) + \sigma + \delta + 2\sigma + \delta + \sigma + G_{1,contention(P)}$. That is, $T_{C(P,c)}(A) \leq \max(\gamma, \sigma) + 2\delta + 4\sigma + G_{1,contention(P)}$.

To summarize the inequalities, let σ' denote $\sigma + \gamma + \delta$. Then for some constant k, we have $G_{i,j} \leq k\sigma' + \upsilon + G_{i,j-1} + G_{i,1}$ for $j \geq 2$, $G_{i,1} \leq k\sigma' + G_{i+1,contention(P)}$ for $i < |c|$, $G_{|c|,1} \leq k\sigma'$, and $T_{C(P,c)}(A) \leq k\sigma' + G_{1,contention(P)}$. Letting a denote contention(P), we have $G_{i,j} \leq (j - 1)(k\sigma' + \upsilon) + j\, G_{i,1}$ for all i, j, and so $G_{i,a} \leq (2a - 1)(k\sigma') + (a - 1)\upsilon + a\, G_{i+1,a}$ for $i < |c|$. Also, $G_{|c|,a} \leq (2a - 1)(k\sigma') + (a - 1)\upsilon$. Thus, $G_{1,a} \leq (1 + a + a^2 + \ldots + a^{|c|-2})((2a - 1)(k\sigma') + (a - 1)(\upsilon)) + a^{|c|-1} G_{|c|,a} = (1 + a + a^2 + \ldots + a^{|c|-1})((2a - 1)(k\sigma') + (a - 1)(\upsilon))$. Then $T_{C(P,c)}(A) \leq k\sigma' + (1 + a + a^2 + \ldots + a^{|c|-1})((2a - 1)(k\sigma') + (a - 1)(\upsilon)) \leq 2k\,|c|\, a^{|c|}\sigma' + (a^{|c|} - 1)\upsilon$, as required. □

Since we do not hypothesize any lower bounds on time for events to occur, there is no limit on the number of times competing users can get ahead of a particular user. However, Theorem 8.1 shows that the only way large numbers of processes can get ahead is by going fast; there is still a limit on the total time any particular user waits.

Corollary 8.1. For some c, $T_{C(P,c)}(A) \leq (contention(P)^{colors(P)} - 1)\,\upsilon + O(colors(P) \cdot contention(P)^{colors(P)}(\sigma + \gamma + \delta))$.

Ex. 8.1. Dining Philosophers

Recall the colorings c and c' from Ex. 2.1. c yields a worst-case running time of $(2^n - 1)\upsilon + O(n2^n(\sigma + \gamma + \delta))$, while c' yields the much better running time $7\upsilon + O(\sigma + \gamma + \delta)$. Intuitively, the ordering yielded by c allows length n waiting chains to form, but c' does not allow chains of length greater than 3.

Ex. 8.2. k-Fork Philosophers

A worst-case bound of $(k^{k+1} - 1)\upsilon + O(k^{k+2}(\sigma + \gamma + \delta))$ is obtained. If, however, the coloring $c(r_i) = i$ is used, one obtains a bound of $(k^n - 1)\upsilon + O(nk^n(\sigma + \gamma + \delta))$.

Ex. 8.3. 2-Dimensional Philosophers

The bound is $O(\upsilon + \sigma + \gamma + \delta)$.

Ex. 8.4. k-Tree

The bound is $(k^k - 1)\upsilon + O(k^k(\sigma + \gamma + \delta))$.

Ex. 8.5. k-Nested Sets

The bound is $(k^k - 1)\upsilon + O(k^k(\sigma + \gamma + \delta))$.

9. Realizing the Upper Bound

It is not always clear how to produce "bad" execution sequences and "bad" A-admissible timings for which the bound derived in Theorem 8.1 is (approximately) realized. For instance, it does not seem possible to exhibit exponential dependence on n in Ex. 8.1 (Dining Philosophers). In this section, we sketch how to construct bad execution sequences and timings for k-Trees, k-Nested Sets and k-Fork Philosophers. In Section 10, we prove an alternative upper bound theorem which implies that such bad execution sequences and timings cannot be constructed for Ex. 8.1.

Ex. 9.1. k-Tree

Consider an execution sequence for a request from user u_{a^k} in which, whenever a user $u_{ia^{k-|i|}}$ arrives on the QUEUE for a resource r_i, all of the other contenders for r_i have just arrived very shortly before. This execution involves u_{a^k} waiting for $k^k - 1$ other users to obtain (sequentially) their resources. Thus, if a timing is constructed to maximize waiting times, a response time of at least $(k^k - 1)\upsilon$ is realized. Note that whenever the specified users arrive on the specified QUEUEs, it is possible that the other contenders can all arrive as required. This is because these other contenders are only being required to arrive at their first resources, which they can do independently.

Ex. 9.2. k-Nested Sets

Let $c(r_i) = i$, $1 \leq i \leq k$. Construct an execution sequence for a request for user u_1 in which whenever a user u_i arrives on the QUEUE for a resource r_j, $j > i$, it is the case that u_j has just arrived very shortly before. This execution involves u_1 waiting for $2^{k-1} - 1$ other requests to be granted; unlike Ex. 9.1, many of these requests are repeats, however. (For instance, for $k = 5$, the order of granted requests is given by $u_5u_4u_5u_3u_5u_4u_5u_2u_5u_4u_5u_3u_5u_4u_5u_1$.) Thus, if a timing is constructed to maximize waiting times, a response time of at least $(2^k - 1)\upsilon$ is realized. Again, the required arrivals are possible because we are only requiring contenders to arrive at their first resources.

In both of these examples above, a permutation of the values of c will make it impossible to construct execution sequences and timings with exponential dependence on k. (For instance, for Ex. 9.2, simply reversing the order of the resources will make the dependence on k quadratic, as we will show in Section 10.)

Neither of the examples above is of the "local" type for which this algorithm is intended. However, one can easily construct an example with a "local" flavor which approaches the upper bound, by patching together multiple instances of Ex. 9.1 or 9.2.

Ex. 9.3. k-Fork Philosophers

Let $c(r_i) = i$, $1 \leq i \leq n$, as in Ex. 8.3. We construct an execution sequence for a request of u_1, using only $u_1,\ldots,u_{n-k+1}$. Whenever a user u_i, $1 \leq i \leq n-k$, arrives on the QUEUE for a resource r_j, $i < j \leq n-k+1$, it is the case that u_j has just arrived. Then, for example, if $k = 3$ and $n = 7$, the order of granted requests is $u_5u_4u_5u_3u_5u_4u_2u_5u_4u_5u_3u_1$. In general, if $f(k,n)$ is the number of requests for which u_1 waits, then one can calculate $f(k,n)$ as follows.

Consider a slightly modified resource problem P' having $R(P') = \{r_1,\ldots,r_n\}$, $U(P') = \{u_1,\ldots,u_n\}$, and $U(P')(r_i) = \{u_j : i-k+1 \leq j \leq i\}$. (Thus, modular arithmetic is eliminated and so the first few resources have fewer than k users if $k > 1$). Let $c(r_i) = i$. We construct an execution sequence for a request of u_1: whenever u_i, $1 \leq i \leq n-1$, arrives on the QUEUE for a resource r_j, $i \leq j \leq n$, it is the case that u_j has just arrived. If $g(k,n)$ denote the total number of requests granted in this execution up to and including the initial u_1 request, then $f(k,n) = g(k,n-k+1) - 1$ for $n \geq 2k - 1$. Then we can see that $g(k,1) = 1$, $g(k,n) = \sum_{i=1}^{n-1} g(i,i) + 1$ for $n \leq k$, and $g(k,n) = \sum_{i=n-k+1}^{n-1} g(k,i) + 1$ for $n > k$. This Fibonacci-style bound shows that $f(k,n)$ is $\Omega(k^{n/k})$, so that a response time of $\Omega(k^{n/k})(\upsilon)$ is realized.

10. A Special Case

A case analysis for the coloring c of the Dining Philosophers Ex. 2.1 shows (in contrast with Ex. 9.3) that no execution exhibiting exponential dependence on $|c|$ is possible. Also, for example, changing only the ordering of the colors in Ex. 9.2 changes the dependence on $|c|$ from exponential to quadratic. Thus, while Theorem 8.1 yields the required independence of network size, it does not tell the entire story.

Theorem 8.1 allows for the possibility that a user will have to wait for the maximum number of competing processes on each queue. However, if two users contend for two different resources, then neither will ever have to wait for the other for the second of the two resources. Moreover, Theorem 8.1 does not take into account any limitations on the resources needed by any particular user. The second theorem takes these factors into account.

Define a tree waittree(P,c,u) for a resource problem P, a coloring c and $u \in U(P)$ as follows.

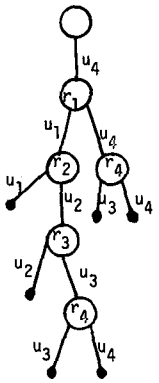
(1) pretree(P,c,u)

The root node has a single son labelled by the resource $first_{P,c}(u)$. The edge joining the the root node to this son is labelled by u.

For any node x labelled by any r, and any $u' \in U(P)(r)$ with $next_{P,c}(u',r)$ defined, there is a son of node x labelled by $next_{P,c}(u',r)$. If $next_{P,c}(u',r)$ is undefined, there is a son of node x which is a dummy node. In either case, the edge joining x to this son is labelled by u'.

Ex. 10.1. Dining Philosophers

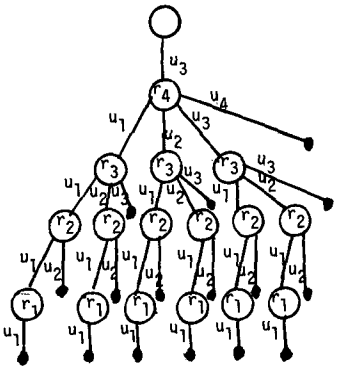
Let n = 4. Pretree(P,c,u_4) is as follows.



Ex. 10.2. k-Nested Sets

Let k = 4 and c'(r_i) = 5 - i.

Pretree(P,c',u_3) is as follows.

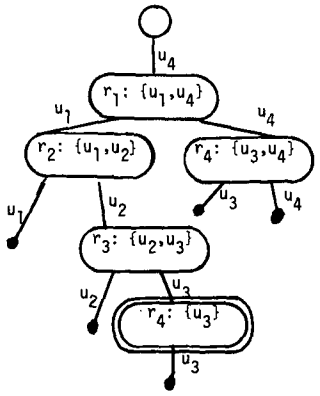


(2) waittree(P,c,u)

The non-dummy nodes of pretree(P,c,u) are assigned tags consisting of sets of processes. If node x has label r, then the tag, A_x, for x is the set of all u' $\in U(P)(r)$ with the following property. For all ancestors y of x, where y is labelled by r', if u' $\in U(P)(r')$ then u' labels the out-edge leaving y in the direction of node x. The resulting tagged tree is then pruned so that the only edges leaving any node are those labelled by processes included in the tag of that node.

Ex. 10.3. Dining Philosophers

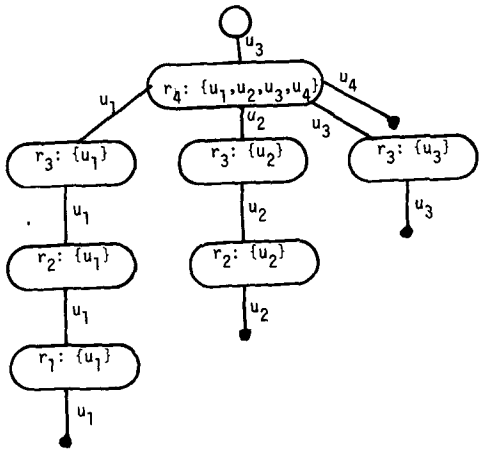
Waittree(P,c,u_4), for Ex. 10.1, is as follows.



The only user omitted from a tag is u_4, omitted from the tag of the double circled node. u_4 is omitted because $u_4 \in U(P)(r_1)$ and u_4 does not label the edge leaving the root in the direction of the double circled node.

Ex. 10.4. k-Nested Sets

Waittree(P,c',u_3) is as follows.



A considerable amount of pruning occurs for this tree.

Let weight(P,c,u,x,B), where B $\subseteq A_x$, denote the number of edges in waittree(P,c,u) below node x, along paths whose first edge below x is labelled by an element of B. Let weight(P,c,u,x) denote weight(P,c,u,x,A_x), and let weight(P,c,u) denote weight(P,c,u,x), where x is the son of the root.

Theorem 10.1. $T_{C(P,c)}(A,u) = O((\text{weight}(P,c,u)) \cdot (\sigma + \upsilon + \gamma + \delta))$.

Proof Sketch. Each request generates a set of QUEUE entries which persist until 'RETURN' messages are received. A QUEUE entry is said to be

active at step i of execution sequence e provided the request which generated it has not yet been granted at (i.e. immediately after) step i. Step i of execution sequence e is consistent with node x of waittree(P,c,u) provided each user which labels an edge above x has an entry which is active on the QUEUE of the resource labelling the lower (son) endpoint of that edge, at step i. (There can be at most one such active entry for each QUEUE.)

Claim 1. At any step i of an execution sequence e, any user u' who has an entry active on the QUEUE for any resource r also has an entry which is first on the QUEUEs for all $r' \in R(P)(u')$ with $c(r') < c(r)$.

Claim 2. Let x be a node of waittree(P,c,u) labelled by resource r. Assume step i of execution sequence e is consistent with x. Let u' be a user with an active entry on r's QUEUE at step i. Then $u' \in A_x$.

Proof of Claim 2. Assume $u' \notin A_x$. Then there is an ancestor y of x, labelled by a resource r', with $u' \in U(P)(r')$ and u" the label of the out-edge leaving y in the direction of node x, $u" \neq u'$. By consistency, u" has an entry active on the QUEUE for some resource with a higher number than c(r'), at step i. By Claim 1, u" has an entry which is first on r' 's QUEUE at step i. However, Claim 1 also implies that u' has an entry which is first on r' 's QUEUE at step i, a contradiction.

Now write $\sigma' = \sigma + \upsilon + \gamma + \delta$.

Claim 3. Let x be a node of waittree(P,c,u) labelled by resource r. Let u' be the label of the edge immediately above x. Assume step i of execution sequence e is consistent with x, and that u" is a user having an active entry a (not necessarily proper) predecessor of u''s active entry on r's QUEUE at step i. Let B be the set of users having active entries which are predecessors of this entry of u" (including u" itself) at step i. (By Claim 2, $B \subseteq A_x$.) Let j be the step at which the request of u" which generated this active entry is granted. Let t be an A-admissible timing for e. Then $t(j) - t(i)$ is $O((\text{weight}(P,c,u,x,B))(\sigma'))$.

Proof of Claim 3. We use induction on the nodes x of waittree(P,c,u), starting at the lowest nodes and working towards the root. For each node x, we use induction on subsets B of A_x, ordered by containment. Assume e, i, x, r, u', u" and B are as above.

There are three cases.

(1) The given active entry of u" is the first active entry on r's QUEUE at step i, and $\text{next}_{P,c}(u",r)$ is undefined.

Then within time $O(\sigma')$, u" 's request is granted, as needed.

(2) The given active entry of u" is the first active entry on r's QUEUE at step i, and $\text{next}_{P,c}(u",r) = r'$.

Then within time $O(\sigma')$, a step i' is reached which is consistent with node y, where y is the son of x reached by following the edge labelled by u". Thereafter, by induction on nodes and by Claim 2, the time until u" 's request is granted is $O((\text{weight}(P,c,u,y))(\sigma'))$. The total time is $O((\text{weight}(P,c,u,y) + 1)(\sigma'))$, as needed.

(3) The first active entry on r's QUEUE at step i is generated by $u''' \neq u"$.

Then, by induction on subsets, within time $O((\text{weight}(P,c,u,x,\{u'''\}))(\sigma'))$, a step i' is reached at which u''' 's request is granted, making the given u''' entry inactive. Step i' is still consistent with x, and $B' = B - \{u'''\}$ is the set of users having active entries which are predecessors of the given entry of u" at step i'. Then, by induction on subsets, u" 's request is granted within time $O((\text{weight}(P,c,u,x,B'))(\sigma'))$. The total time is $O((\text{weight}(P,c,u,x,\{u'''\}) + \text{weight}(P,c,u,x,B'))(\sigma')) = O((\text{weight}(P,c,u,x,B))(\sigma'))$, as needed.

Now consider any request of u. Within time $O(\sigma')$ of initiation, a step i is reached at which u obtains an active entry on the QUEUE for $\text{first}_{P,c}(u)$. Step i is consistent with the son of the root of waittree(P,c,u). Claim 3 yields the result. □

Ex. 10.5. Dining Philosophers

Generalizing Ex. 10.3, we see that coloring c provides a running time of $O(n(\sigma + \upsilon + \gamma + \delta))$, because of the size of the waittrees. This is in contrast to the exponential bound of Ex. 9.3.

Ex. 10.6. k-Nested Sets

Generalizing Ex. 10.4, we see that the coloring c' provides a running time of $O(n^2(\sigma + \upsilon + \gamma + \delta))$. This is in contrast to Ex. 9.2.

There are, of course, cases in which Theorem 10.1 does not provide improvement over Theorem 8.1. For example, for a k-tree, the waittree for coloring c of Ex. 2.4 and user u_{a^k} follows the structure of the k-tree itself. (See the example for k = 3 in Ex. 2.4.) Thus, the waittree has more than k^k edges. We also note that the given upper bound proportional to the size of the waittree cannot always be realized; by some complicated arguments, it is often possible to eliminate still more possible waiting.

11. Throughput and Limited Concurrency Bounds

The material of this section is presented in outline; many details are reserved for a longer version of this paper. One might be interested in measures other than worst-case response time. For example, one can obtain an upper bound on "worst-case throughput" by measuring the rate at which requests are granted, assuming that each user always initiates a new request within some

time ε after the preceding request is returned.

Roughly, if $A = (\sigma, \upsilon, \gamma, \delta, \varepsilon) \in (R^+)^5$, then a timing is A-admissible provided σ, υ, γ, and δ are as before and, in addition, the first request of each user is within ε of the beginning, and each subsequent request is within time ε of the previous return by that user. Then let $T'_C(A)$ denote the supremum of the quantity $\limsup_{i\to\infty} \frac{t(i)}{grants(e,i)}$ for all execution sequences e of C and all A-admissible timings t for e. An easy corollary to Theorem 8.1 says that $T'_{C(P,c)}$ is $O(\frac{|c|contention(P)^{|c|}(\sigma + \upsilon + \gamma + \delta) + \varepsilon}{n})$ if $|U(P)| = n$, This rate seems to compare favorably with other alternatives.

Another interesting measure is worst-case performances under assumptions of limited concurrency. If at most k requests are concurrent with a given request, then worst-case response time for the given request might be better than worst-case with unlimited concurrency, for small values of k. Analysis techniques for deriving such bounds are quite different from those used for Theorems 8.1 and 10.1. For the problem treated in this paper, locality is important. Therefore, we seek a worst-case bound in the presents of at most k concurrent "nearby" requests. For $u \in U(P)$, define pred(u) to be the set of users appearing as edge labels and res(u) to be the set of resources appearing as node labels, in waittree(P,c,u). (Thus, pred(u) represents all users which could delay the granting of a request of u.) Let $A = (\sigma, \upsilon, \gamma, \delta)$ and use the definition of A-admissibility in Section 6. Let $T''_C(a,u,k)$ denote the supremum, for all execution sequences e and A-admissible timings t, of the quantity $t(j) - t(i)$, where u makes a request at step i which is granted at step j, and where $requests(e, j, pred(u)) \leq k + returns(e,i,pred(u))$. It is not difficult to verify the following claim about our system $C(P,c)$: if at any step of any execution sequence there is a request of user u pending, and if there is no request of a user in pred(u) currently granted, then within time $O(|c|(\sigma + \gamma + \delta))$, some request by a user in pred(u) gets granted. Therefore, if there is a bound of k on such requests, the total time to grant the request of u is $(k - 1)\upsilon + O(k|c|(\sigma + \gamma + \delta))$.

A refinement on the analysis outlined above might attempt to use the fact that the message system might also be guaranteed to perform at better than its worst-case performance under the limited usage deducible from the given limit on user requests. The message system is performing a significant part of the work of the entire system, and its improved performance under light usage conditions might be expected to have a significant impact on the calculated bounds. In order to obtain such a sharpened analysis, one needs to include more detailed bounds on the behavior of the message system in the admissibility vector, rather than just γ and δ. Let $A = (\sigma, \upsilon, \gamma, \delta, \mu) \in (R^+)^5$, and redefine a timing t to be A-admissible for an execution sequence e provided σ, υ, γ and δ are as in Section 6, and μ satisfies the following. For all $k \geq 1$ and all p, if $sentto(e,j,p) \leq k + \min(deliveredto(e,i,p), collectedto(e,i,p))$ and if for no ℓ, $i < \ell < j$ is the case that $sentto(e,\ell,p) = collectedto(e,\ell,p) = deliveredto(e,\ell,p)$, then $t(j) - t(i) \leq k\mu$. That is, we bound the length of an interval during which there are at most k messages to one process p, provided that some message to process p is being processed at each intermediate step. For simplicity, we assume a bound of the form $k\mu$, where μ is to be thought of as much smaller than γ and δ. Let $T'''_C(A,u,k)$ be defined to be the same as $T''_C(A,u,k)$, except that the new definition of A-admissibility is used. We require a lemma in order to analyze our system $C(P,c)$.

Lemma 11.1. If $r \in res(u)$ and $u' \in U(P)(r)$, then $u' \in pred(u)$.

Proof. Let x be the highest node on the path from the root of waittree(P,c,u) to any node labelled by r, such that $label(x) \in R(P)(u')$. Then $u' \in A_x$. □

$T'''_{C(P,c)}(A,u,k)$ can be bounded as follows. First, there may be an initial interval of $O(\sigma + \gamma + \delta)$ before all old 'RETURN' messages from non-concurrent requests have been collected, delivered and processed. We analyze the remain interval I until u's request is granted. First imagine that all messages overlapping I which are addressed to users in pred(u) or to resources in res(u) take time zero until collection and delivery. With this assumption, the time for I is at most $(k - 1)\upsilon + O(k|c|\sigma)$. (The analysis is the same as that for $T''_{C(P,c)}(A,u,k)$.) We must add to this bound the total time spent waiting for all the relevant messages to be collected and delivered, which we calculate as follows.

The given requests concurrent with the original request of u produce a set of at most $k(2|c| + 1)$ messages, all addressed either to users in pred(u) or to resources in res(u). Moreover, all messages which overlap interval I and are addressed to these users and resources are among this set of messages. This is because the only messages addressed to a user process are 'G' messages originating from his own requests, and also the only messages addressed to a resource process are 'RETURN' messages and requests from its users and from lower numbered resources of its users. But all of these types of messages must originate from requests by users in pred(u), by Lemma 11.1. Because I does not begin until the initial interval has elapsed, these requests must all be among the given concurrent requests.

Now consider any particular destination process p, and assume ℓ of the messages above are addressed to p. By admissibility, the total time for the ℓ messages to p is at most $\ell\mu$. Summing over all of the relevant processes yields

a total message waiting time of $k(2|c| + 1)\mu$.
Thus, the total time is at most
$(k - 1)\upsilon + O(\sigma + \gamma + \delta) + O(k|c|\ (\sigma + \mu))$.

This work is part of two larger projects - a joint project with Professor Michael J. Fischer on theory of asynchronous systems and a Georgia Tech project for design of distributed computing systems. Thanks for many ideas and discussions go to the members of both projects, especially Mike Fischer, Nancy Griffeth, and Jim Burns.

References

[B] Burns, J.E., PhD Thesis, School of Information and Computer Science, Georgia Institute of Technology, in progress.

[C] Chang, E., n-philosophers: "An Exercise in Distributed Control", University of Toronto, unpublished manuscript, 1978.

[CH] Cremers, A.B. and Hibbard, T.N., "Arbitration and Queueing Under Limited Shared Storage Requirements", University of Dortmund Technical Report, 1979.

[D] Dijkstra, E.W., "Hierarchical Ordering of Sequential Processes", Acta Informatica 1: 115-138, 1971.

[FLBB] Fischer, M., Lynch, N., Burns, J. and Borodin, A., "Resource Allocation with Immunity to Limited Process Failure", 20th Annual Symposium on Foundations of Computer Science, 234-254, 1979.

[G] Garcia-Molina, H., "Performance of Update Algorithms for Replicated Data in a Distributed Database", PhD Thesis, Stanford University, 1979.

[LF] Lynch, N. and Fischer, M., "On Describing the Behavior and Implementation of Distributed Systems", GIT-ICS-79/03.
See also Lecture Notes in Computer Science, Semantics of Concurrent Computation, Proceedings, Evian, France, 147-171, 1979.
Also submitted for publication in Theoretical Computer Science.

[P] Peterson, G., PhD Thesis, Computer Science Department, University of Washington, 1979.

[PF] Peterson, G., and Fischer, M., "Economical Solutions to the Critical Section Problem in a Distributed System", Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, 91-97, 1977.