Time Bounds on Synchronization in a Periodic Distributed System *

Injong Rhee † *Jennifer L. Welch* ‡

September 25, 1997

Abstract

This paper studies the time required to solve the session problem in a new timing model, called the *periodic* model, for shared memory distributed systems. In the periodic model, each process runs at a constant unknown rate and different processes may run at different rates. Nearly matching upper and lower bounds are shown on the time complexity of the session problem in the model. These bounds indicate the inherent cost of synchronizing periodic processes in shared memory distributed systems, and the existence of time complexity gaps among the synchronous, periodic, and asynchronous timing models.

Keywords: Distributed Computing, Time Bounds, Session Problem, Periodic Model

1 Introduction

The (s, n)-*session* problem, first formulated in [2], is an abstraction of the synchronization needed to solve some distributed computing problems. Informally, a *session* is a minimal-length computation fragment that involves at least one "synchronization" step by every process in a distinguished set of n processes. An algorithm that solves the (s, n)-session problem must guarantee that in every computation, there are at least s disjoint sessions, and eventually all the processes become idle.

A direct example of the session problem can be found in a system that solves a set of linear equations by successive relaxation, where each process holds some of the input parameters to the

*This work was supported in part by an IBM Faculty Development Award and NSF PYI Award CCR-9158478. Much of this work was done while the authors were with the Department of Computer Science, University of North Carolina, Chapel Hill.

†Department of Computer Science, North Carolina State University, Raleigh, NC 27606, USA. E-mail: rhee@csc.ncsu.edu

‡Department of Computer Science, Texas A&M University, College Station, TX 77843, USA. E-mail: welch@cs.tamu.edu

linear equations. (cf. [4]). Each process takes one synchronization step when it changes its values. Sufficient interleaving of synchronization steps by different processes ensures that a correct answer is computed, since it implies sufficient interaction among the intermediate values computed by the processes. Solutions for the session problem can be used to solve the successive relaxation problem.

Since the time complexity of the session problem is very sensitive to the timing assumptions of the underlying model, it has been used as a test-case to demonstrate the theoretical differences in the time needed to solve problems in various timing models [2, 3, 11]. In order for our results to be comparable with prior work, we concentrate on shared memory systems with a constant parameter b, which is the maximum number of distinct processes that are ever allowed to access any given shared variable. When b is smaller than the total number of processes in the system, it is not possible for every pair of processes to exchange information in a single step. Instead, information must be propagated from process to process. Thus, as b gets smaller, the amount of propagation required increases. The motivation for this restriction on communication comes from the fact that in a distributed shared memory system, some part of memory is local to a process and can be accessed quickly, while the rest is remote and requires more time for accesses.

The upper and lower bounds on the time required to solve the session problem shown by Arjomandi, Fischer and Lynch [2] demonstrated the first such case where asynchronous systems are less efficient than synchronous systems. In the synchronous model, all processes run in lockstep, while in the asynchronous model, no bounds on process running rates exist. Their result showed an inherent time complexity gap between the synchronous and asynchronous models: s steps are sufficient for the synchronous model, i.e., no interprocess communication is needed, but $(s-1)\lfloor \log_b n \rfloor$ steps are necessary for the asynchronous model. The $\lfloor \log_b n \rfloor$ factor is essentially the cost of communication, since no more than b processes can access any shared variable. Thus, one interprocess communication per session is needed in the asynchronous model.

The session problem has been studied in a semi-synchronous shared memory model as well, in which there are upper and lower bounds on process step time, denoted c_u and c_l respectively. In this model, the time complexity upper bound is $O((s-1) \cdot c_u \cdot \min\{\frac{c_u}{c_l}, \log_b n\})$ [11]. A nearly matching lower bound (within a factor of 2 of the upper bound) appears in [11, 9]. In the semi-synchronous model, the existence of known bounds on the running rate allows processes to determine when enough sessions have elapsed by simply counting the number of local process steps, as in the synchronous model. However, if the difference between the bounds is sufficiently large, then explicit communication per session, as in the asynchronous model, can solve the problem more efficiently. This implies that the efficiency of the semi-synchronous shared memory model lies between those of the synchronous and asynchronous models.

Results concerning the session problem in message passing models can be found in [3, 8, 11].

In this paper, the session problem continues to be used to compare timing models quantitatively for shared memory systems. In particular, we study a new timing model, called the *periodic* model, in which each process takes steps at an unknown constant rate and different processes may run at different rates.

Though the periodic model requires stringent timing guarantees such as constant running rates, a time complexity lower bound for any problem in that model is also applicable to other models with less stringent timing guarantees. In particular, these weaker models include the models with unknown bounds in which the running rates of processes may change over time, but are always bounded by unknown constants [1, 7]. These models provide stronger timing guarantees than the asynchronous model, but weaker guarantees than the periodic models. Thus, the time complexity study in the periodic model can provide indications of the cost to solve problems in these weaker models.

The main result of this paper is a lower bound on the time complexity of solving the session problem in the periodic shared memory model. We show that it requires at least $c_{max} \cdot \max\{s, \lfloor \log_{2b-1}(2n-1) \rfloor\}$ time to solve the session problem in the periodic shared memory model, where c_{max} is the step time of the slowest process. An almost matching upper bound $c_{max} \cdot (s + \Theta(\log_b n))$ is also presented. Intuitively, these bounds imply that to solve the session problem in the periodic model, a total of one interprocess communication delay is necessary and sufficient.

Taken together with the results in [2], our results indicate that, with respect to the session problem, the time complexity of the periodic model falls strictly between those of the synchronous and asynchronous models.

2 The System Model

The system model definition is similar to that defined in [2].

There are finite sets P of processes and V of shared variables. A *process* has a set of internal states, including an initial state. Each *shared variable* has a set of values that it can contain, including an initial value. A *global state* is a tuple of internal states of each process, and values of each shared variable. The *initial* global state contains the initial state for each process and the initial value for each shared variable.

Associated with each variable is a set of at most b processes that are allowed to access that variable. A process can both read and write a shared variable in a single atomic step (i.e., the

variable supports read-modify-write operations); we do not assume any upper bound on the size of the variables. A *step* π consists of simultaneous changes to the state of some process p and the value of some variable x (where p is allowed to access x), depending on the current state of that process and current value of the variable. More formally, we represent the step π with a tuple $((q,p,r),(u,x,v))$, where q and r are old and new states of a process $p \in P$, and u and v are old and new values of a shared variable $x \in V$. We say that step π is *applicable* to a global state if p is in state s and x has value u in the global state.

An *algorithm* consists of P, V, and set Σ of possible steps. For all processes $p \in P$ and all global states g, there must exist some step in Σ involving process p that is applicable to global state g. This condition ensures that p never blocks. A *computation* of a system is a sequence of steps $\pi_1, \pi_2, \ldots$ such that: (1) π_1 is applicable to the initial global state, (2) each subsequent step is applicable to the global state resulting from the previous step, and (3) if the sequence is infinite, then every process takes an infinite number of steps. That is, there is no process failure.

A *timed computation* (α, T) of a system is a computation $\alpha = \pi_1, \pi_2, \ldots$ together with a mapping T from positive integers to nonnegative real numbers that associates a real time with each step in the computation. T must be nondecreasing and, if the computation is infinite, increase without bound. A timed computation is *admissible* if for each process p in P, there is a positive constant c_p such that the time between every pair of consecutive steps of p in the timed computation is c_p and p's first step is taken at time c_p. Since c_p can be different in different timed computations, in essence it is unknown and thus cannot be hard-wired into an algorithm. When the timed computation is clear from context, we use c_{max} to represent the maximum c_p, over all p in P, for that computation.

3 The (s,n)-Session Problem

We now state the conditions that must be satisfied for a system to solve the (s,n)-session problem within a certain amount of time.

There must be a distinguished set Y of n shared variables called *ports*; Y is a subset of V. There must be a unique process in P corresponding to each port, which is called a *port process*, and no two port processes can be assigned to the same port. A *port step* is any step involving a port and its corresponding port process. A port can be accessed by processes in addition to its own port process, but such a step is not a port step. There may be some processes which are not port processes, i.e., it is possible for $|P|$ to be larger than n.[1]

[1] This possibility is implicitly contained in [2], which refers to making the port processes the leaves of a tree network.

Each port process in P must have a subset of special states, called *idle* states. The set Σ of steps of the system must guarantee that once a process is in an idle state, it always remains in an idle state, and after a process enters an idle state, it does not access a port.

A *session* is a minimal sequence of steps containing at least one port step for each port in Y. A computation *performs* s sessions if it can be partitioned into s segments, each of which is one session.

An algorithm *solves the* (s, n)*-session problem within time* X if, in every infinite admissible timed computation of the algorithm, there are at least s sessions and each process is in an idle state by time X.

4 Upper Bound

We present an algorithm A_{per} for the (s, n)-session problem in this model, whose running time is $c_{max} \cdot (s + \Theta(\log_b n))$.

In describing the algorithm, we use a subroutine called *broadcast* as a generic operator for communication in the model. Recall that communication in our system model is constrained by the fact that at most b processes can access any specific shared variable. We conceptually organize the processes and shared variables into a tree. In order for a port process to broadcast information to all other port processes, the information travels up the tree to the root and then down from the root to all the leaves.

In more detail, consider a $(b-1)$-ary tree with n leaves in which each level, except possibly the lowest, has the maximum number of nodes. Note that the number of levels in the tree is $\lceil \log_{b-1} n \rceil + 1$. Associated with each node in the tree are a process and a shared variable. Each port process and its port variable are associated with a leaf node; the processes and variables associated with internal nodes are called *relay* processes and variables. The relay variable associated with a node is accessed by the process associated with the node and the processes associated with that node's children in the tree. Figure 1 illustrates a tree with $b = 4$ and $n = 7$.

Each relay variable has two fields, *up* and *down*. Each process has two local variables, *lup* and *ldown*; initially they are empty except that *lup* at a port process holds the information to be propagated.

Each relay process p other than the root repeats the following two steps. First, p accesses its own shared variable, saving the contents of the *up* field in *lup* and appending the contents of *ldown* to the *down* field. Second, p accesses its parent's variable, appending *lup* to the *up* field and saving the *down* field in *ldown*.

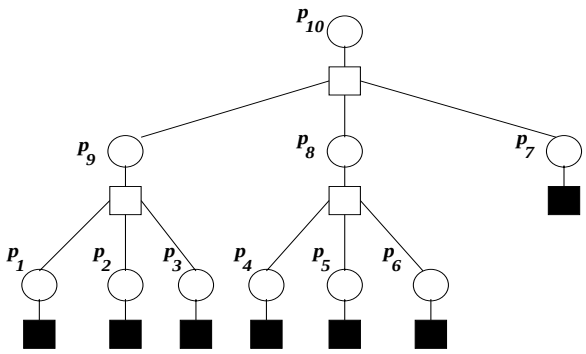

Figure 1: A tree network with $b = 4$ and $n = 7$ where circles represent processes, empty squares relay variables, dark squares port variables, solid lines memory access patterns of processes. Port processes are p_1 through p_7.

The root continuously accesses its own variable; at each access it copies the *up* field to the *down* field.

In the example of Figure 1, a piece of information m is transferred from p_2 to p_6 as follows: p_2 appends m to $v_9.up$; p_9 obtains the contents of $v_9.up$ and appends them to $v_{10}.up$; p_{10} copies the contents of $v_{10}.up$ to $v_{10}.down$; p_8 obtains the contents of $v_{10}.down$ and appends them to $v_8.down$; p_6 obtains the contents of $v_8.down$ and gets m.

It takes at most $c_{max} \cdot \lceil \log_{b-1} n \rceil$ time for a piece of information (or a "message") to be relayed up to the root for the following reason. In each time interval of length c_{max}, every process takes at least one step, and thus every relay process other than the root passes the message up to its parent. Likewise, it takes additional $c_{max} \cdot (\lceil \log_{b-1} n \rceil + 1)$ time for the message at the root to be relayed down to a leaf node in the tree (the one additional c_{max} is for the root to move the message from its *up* to its *down*). Thus, the broadcast is accomplished in $c_{max} \cdot \Theta(\log_{b-1} n)$ time. A similar tree network is mentioned in [2].

> Algorithm A_{per}: Each port process accesses its own port $s-1$ times. After its $(s-1)$-st step, it broadcasts the fact that it has finished its $(s-1)$-st step, and keeps taking port steps until it hears that all other processes have taken $s-1$ steps. Then it takes one more port step and enters an idle state.

Theorem 1 *A_{per} solves the (s,n)-session problem in time $c_{max} \cdot (s + \Theta(\log_b n))$ in the periodic model.*

Proof: Pick any infinite admissible timed computation of A_{per}. Let p be any port process with the maximum step time c_{max}. By the definition of the periodic model, it is clear that s sessions

have occurred and no process is yet idle by the time that p takes its $(s-1)$-st step and broadcasts the fact. It takes at most $s \cdot c_{max}$ time for p to take s steps; every process obtains p's message and becomes idle after an additional $\Theta(\log_b n) \cdot c_{max}$ time has elapsed. ■

5 Lower Bound

Theorem 2 *No algorithm can solve the (s,n)-session problem in the periodic model in time less than $c_{max} \cdot max\{s, \lfloor \log_{2b-1}(2n-1) \rfloor\}$.*

Proof: Suppose that $s \geq \lfloor \log_{2b-1}(2n-1) \rfloor$. Since all processes must take at least s steps to have s sessions, $s \cdot c_{max}$ is obviously the lower bound, which suffices.

Suppose that $s < \lfloor \log_{2b-1}(2n-1) \rfloor$. By way of contradiction we assume that there exists an algorithm A that solves the (s,n)-session problem in the periodic model in time Z strictly less than $c_{max} \cdot \lfloor \log_{2b-1}(2n-1) \rfloor$. Let the set of processes for the algorithm be $P = \{p_1, \ldots, p_m\}$, for some $m \geq n$. We prove that there exists an infinite admissible timed computation of A that contains fewer than s sessions, contradicting the assumed correctness of A.

Let (α, T) be the infinite admissible timed computation in which processes take steps at the same speed in round robin order (p_1 through p_m) and each process's ith step occurs at time $i \cdot c_{max}$. Each consecutive group of steps for p_1 through p_m is a round. (Round i occurs at time $i \cdot c_{max}$ and consists of the i-th step of each process.) Since all port processes should enter idle states by time Z in (α, T) and all the step time periods are equal to c_{max} in (α, T), there are at most $r = \lfloor Z/c_{max} \rfloor$ rounds by time Z in (α, T).

Now we change the step time of p_1 to be $c_{max} \cdot \lfloor \log_{2b-1}(2n-1) \rfloor$, i.e., the steps of p_1 occur every $c_{max} \cdot \lfloor \log_{2b-1}(2n-1) \rfloor$ time units. Note that this step time is bigger than Z. Run A with this modified process and the rest of the original processes to get a new infinite timed admissible computation (α', T').

Since the steps of p_1 occur at different times in (α', T') from those at which they do in (α, T), there will be other steps that are influenced by the steps of p_1. These steps will in turn influence others. We say that these steps are "contaminated" by the steps of p_1. However, we prove that there is at least one process p_j whose steps are not contaminated through time Z. Thus, because all processes are in an idle state at time Z in (α, T), p_j is also in an idle state at time Z in (α', T'). However, there has not been even a single step of p_1 by time Z in (α', T'). This is a contradiction. We now formalize these ideas.

We break α' up into a sequence of disjoint *pseudo-rounds.* Pseudo-round 0 is the empty computation fragment at the beginning of α'. For $1 \leq t \leq r$, pseudo-round t consists of the t-th step in α' of every process except p_1. Thus, every pseudo-round (except for 0) is a minimal computation fragment of α' that consists of one step by every process except p_1.

A variable v is *contaminated* in pseudo-round $t \geq 1$ of α' if there exists $k \leq t$ and process p_j such that v's value in the global state of α' following p_j's step in pseudo-round k is not equal to v's value in the global state of α following p_j's step in round k. We define no variable to be contaminated in pseudo-round 0. A process p_j is *contaminated* in pseudo-round $t \geq 1$ of α' if $p_j \neq p_1$ and there exists $k \leq t$ such that in pseudo-round k of α', p_j accesses a variable that is contaminated in pseudo-round k. We define no processes to be contaminated in pseudo-round 0. Two comments about this definition are in order. First, according to this definition p_1 is not contaminated, which makes subsequent analysis a little easier. Second, the definition provides an over-estimate of the processes that are contaminated by a variable, since it includes cases when the variable does not become contaminated until after the process accesses it.

For $0 \leq t \leq r$, let $P_c(t)$ be the set of processes that are contaminated in pseudo-round t of α', and let $V_c(t)$ be the set of variables that are not contaminated in pseudo-round $t-1$ but are contaminated in pseudo-round t of α'. Note the asymmetry in the definitions: $V_c(t)$ is the number of variables that have just become contaminated in pseudo-round t, while $P_c(t)$ is the total number of processes contaminated up to and including pseudo-round t. The sizes of $P_c(t)$ and $V_c(t)$ depend on the algorithm A, which has been chosen arbitrarily. We now define two other quantities, P_t and V_t. We will show that P_t is an upper bound on $|P_c(t)|$ and V_t is an upper bound on $|V_c(t)|$. Define P_t and V_t to satisfy the recurrence equations:

$$P_0 = 0, \qquad V_0 = 0 \tag{1}$$

$$V_t = 2 \cdot P_{t-1} + 1, \qquad t \geq 1 \tag{2}$$

$$P_t = P_{t-1} + (b-1) \cdot V_t, \qquad t \geq 1 \tag{3}$$

We now show that the number of processes contaminated in pseudo-round r is less than $n-1$.

$$\begin{array}{rcll} |P_c(r)| & \leq & P_r & \text{by Lemma 3 below} \\ & = & \frac{(2b-1)^r - 1}{2} & \text{by Lemma 4 below} \\ & < & \frac{(2b-1)^{\lfloor \log_{2b-1}(2n-1) \rfloor} - 1}{2} & \text{by definition of } r \text{ and assumption on } Z \\ & \leq & \frac{2n-2}{2} & \\ & = & n-1. & \end{array}$$

Since fewer than $n-1$ processes are contaminated in pseudo-round r, at least one port process p_j besides p_1 is not contaminated. Thus, p_j is in the same state at the end of pseudo-round r in α' as

it is at the end of round r in α, namely an idle state. But p_1 has not taken a step yet. Thus, (α', T') is an infinite admissible timed computation that contains fewer than s sessions. Contradiction. ■

Lemma 3 *$|P_c(t)| \leq P_t$ and $|V_c(t)| \leq V_t$ for $0 \leq t \leq r$, where $r = \lfloor Z/c_{max} \rfloor$.*

Proof: The idea behind this lemma is that even if contamination spreads as fast as possible, it cannot reach every process in only r rounds.

We prove the lemma by induction on t.

For the basis $(t = 0)$, no process or variable is contaminated in pseudo-round 0 by definition.

Assume that the lemma is true for $t - 1 \geq 0$. We now show it is true for t. Contamination spreads as fast as possible if the following two conditions hold.

1. Each process contaminates the maximum number of variables in each pseudo-round. This maximum number is two for every process p_j other than p_1, since p_j can fail to access a variable it was supposed to and can access a variable it was not supposed to. This maximum number is one for process p_1, since p_1 fails to access a variable it was supposed to. (Recall that p_1 takes no step until after time Z.)

2. Processes become contaminated as soon as possible, as follows. Suppose a variable v becomes contaminated in a certain pseudo-round due to the action (or lack of action) of some process p_j. Recall that at most b distinct processes can ever access any given shared variable. Since v is already contaminated, due to some previously contaminated process, there are only $b - 1$ remaining processes that can be contaminated. Let the remaining $b - 1$ processes that can access v actually do so in that pseudo-round.

Condition (1) indicates that $|V_c(t)| \leq 2 \cdot P_c(t-1) + 1$. By the inductive hypothesis, $P_c(t-1) \leq P_{t-1}$. Thus, $|V_c(t)| \leq 2 \cdot P_{t-1} + 1 = V_t$.

Condition (2) implies that at most an additional $(b-1) \cdot |V_c(t)|$ processes become contaminated in pseudo-round t. Thus, $|P_c(t)| \leq |P_c(t-1)| + (b-1) \cdot |V_c(t)|$. We just showed that $|V_c(t)| \leq V_t$. Since the inductive hypothesis implies that $P_c(t-1) \leq P_{t-1}$, it follows that $|P_c(t)| \leq P_{t-1} + (b-1) \cdot V_t$.

■

Lemma 4 *For all $t \geq 0$, $P_t = \frac{(2b-1)^t - 1}{2}$.*

Proof: Substituting Eq. 2 into Eq. 3 yields $P_t = (2b-1) \cdot P_{t-1} + b - 1$. A simple induction shows that the solution to this equation with initial condition given by Eq. 1 is the desired expression.

■

Our lower bound proof is similar to some lower bound proofs on the time complexity of computing various functions in the PRAM model of parallel computing (e.g., addition [5] and logical OR [6]). Like ours, these proofs calculate the rate at which a change in one input value or the state of one process affects the states of other processes.

6 Conclusion

We have proposed a new timing model, the periodic model, and proved almost tight upper and lower bounds on the time needed to solve the session problem in the model. The bounds intuitively indicate that to solve the session problem in periodic shared memory systems, a total of one interprocess communication is necessary and sufficient. Since synchronous systems require no communication and asynchronous systems require one communication *per session*, it follows that periodic systems are more efficient than asynchronous systems while less efficient than synchronous systems.

Analogous results have been obtained for the periodic message passing model [10]; the lower bound proof is much simpler than for the shared memory case.

Acknowledgement: We thank the anonymous referees for their critical reading of earlier versions of this paper and many helpful comments.

References

[1] R. Alur, H. Attiya and G. Taubenfeld, "Time-Adaptive Algorithms for Synchronization," *Proc. 26th ACM Symposium on Theory of Computing*, Montreal, Quebec, Canada, May 1994, pp. 800–809.

[2] E. Arjomandi, M. Fischer and N. A. Lynch, "Efficiency of Synchronous versus Asynchronous Distributed Systems," *Journal of the ACM*, vol. 30, no. 3, 1983, pp. 449–456.

[3] H. Attiya and M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Networks," *Mathematical Systems Theory*, vol. 27, 1994, pp. 547–571.

[4] G. Baudet, "Asynchronous Interactive Methods for Multi-Processors," *Journal of the ACM*, vol. 32, no. 4, 1978, pp. 226–244.

[5] P. Beame, "Limits on the Power of Concurrent-Write Parallel Machines," *Proc. 18th ACM Symposium on Theory of Computing,* Berkeley, CA, May 1986, pp. 169–176.

[6] S. Cook, C. Dwork and R. Reischuk, "Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes," *SIAM Journal of Computing*, vol. 15, no. 1, Feb. 1986, pp. 87–97.

[7] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM,* vol. 35, no. 2, 1988, pp. 288-323.

[8] N. Lynch, *Distributed Algorithms*, Morgan-Kaufman Publishers, Inc., San Francisco, CA, 1996.

[9] M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Systems: Atomic Shared Memory," *Computers and Mathematics with Applications*, vol. 25, no. 2, Jan. 1993, pp. 81–91.

[10] I. Rhee, *Efficiency of Partial Synchrony, and Resource Allocation in Distributed Systems,* PhD dissertation, University of North Carolina at Chapel Hill, 1994.

[11] I. Rhee and J. Welch, "The Impact of Time on the Session Problem," *Proc. 11th ACM Symposium on Principles of Distributed Computing,* Vancouver, BC, Canada, August 1992, pp. 191–201.