Time Bounds on Synchronization in a Periodic Distributed System *

(Accepted to Information Processing Letter)

Injong Rhee † Jennifer L. Welch ‡

August 20, 1996

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 inherent 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 [1], 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 Mathematics and Computer Science, Emory University, Atlanta, GA 30322, USA. E-mail: rhee@mathcs.emory.edu

‡Department of Computer Science, Texas A&M University, College Station, TX 77843, USA. E-mail: welch@cs.tamu.edu
linear equations. (cf. [3]). 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 [1, 2, 8]. In order for our results to be comparable with prior work, we concentrate on shared memory systems with a constant $b$, which is the maximum number of distinct processes that are ever allowed to access any given shared variable. 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 while the rest is remote. 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. Thus $b$ reflects the time cost incurred when accessing remote parts of memory.

The upper and lower bounds on the time required to solve the session problem shown by Arjomandi, Fischer and Lynch [1] 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 is upper bounded by $O((s - 1) \cdot c_u \cdot \min\{\frac{\log b}{c_l}, \log_b n\})$ [8]. A nearly matching lower bound (within a factor of 2 of the upper bound) appears in [8, 6]. In words, 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 [2, 8].
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, such as non-periodic running rates. Thus, the time complexity study in the periodic model can provide indications of the cost to solve problems in 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 \( c_{\text{max}} \cdot \max\{s, \lfloor \log_2 n \rfloor, 2n\} \) time to solve the session problem in the periodic shared memory model, where \( c_{\text{max}} \) is the step time of the slowest process. An almost matching upper bound \( c_{\text{max}} \cdot (s + \Theta(\log n)) \) 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 [1], 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 [1].

There are finite sets \( P \) of processes and \( X \) 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.

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 \( \pi \) consists of simultaneous changes to the state of some process \( p \) and the value of some variable \( x \), depending on the current state of the process and current value of the variable. More formally, we represent the step \( \pi \) with a tuple \((s, p, r), (u, x, v)\), where \( s \) 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 \( x \in X \). (**Let’s delete the following definitions because they are not used anywhere in the paper. **) [[We
define $\text{proc}(\pi) = p$ and $\text{var}(\pi) = x$. We say that step $\pi$ 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 $P$, $X$, and set $\Sigma$ of possible steps. We assume there exists a constant $b$ such that $b$ is an upper bound on the number of processes that access any single variable, in all the steps of the system. For all processes $p \in P$ and all global states $g$, there must exist some step in $\Sigma$ 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) $\pi_1$ 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 $(\alpha, T)$ 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 there is a positive constant $c_i$ for each process $p_i$ in $P$ such that the time between every pair of consecutive steps of $p_i$ in the timed computation is $c_i$ and its first step is taken at time $c_i$. Since the $c_i$'s can be different in different timed computations, in essence they are unknown and thus cannot be hard-wired into algorithms. When the timed computation is clear from context, we use $c_{\text{max}}$ to represent the maximum $c_i$ 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.

There must be a distinguished set $Y$ of $n$ shared variables called ports; $Y$ is a subset of $X$. 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. There may be some processes which are not port processes, i.e., it is possible for $|P|$ to be larger than $n$.

Each port process in $P$ must have a subset of special states, called idle states, and any step that changes the state of a process to an idle state must not involve a port. The set $\Sigma$ of steps of the system must guarantee that once a process is in an idle state, it always remains in an idle state.

---

1This possibility is implicitly contained in [1], which refers to making the port processes the leaves of a tree network.
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. Every infinite admissible timed computation must perform at least $s$ sessions and eventually all port processes must be in idle states.

An algorithm solves the $(s,n)$-session problem in time $X$ if, in every infinite admissible timed computation of the algorithm, each process is in an idle state within $X$ time.

4 Upper Bound

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

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 a shared variable. We conceptually organize the processes and shared variables into a $(b - 2)$-ary tree. The port processes are associated with the leaves of the tree while other (relay) processes are associated with the internal nodes. 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. Thus the broadcast is accomplished in $\Theta(\log_{b} n)$ steps. A similar tree network is mentioned in [1].

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 $c_{\text{max}} \cdot (s + \Theta(\log_{b} n))$ in the periodic model.

**Proof:** Pick any infinite admissible timed computation of $A_{per}$. Let $p_{i}$ be any port process with the maximum step time $c_{\text{max}}$. 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 $p_{i}$ takes its $(s-1)$-st step and broadcasts the fact. It takes at most $s \cdot c_{\text{max}}$ time for $p_{i}$ to take $s$ steps; every process reads $p_{i}$'s message and becomes idle after an additional $\Theta(\log_{b} n) \cdot c_{\text{max}}$ time has elapsed.
5 Lower Bound

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

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

Suppose that \(s < \lfloor \log_{2^{k-1}} 2n \rfloor \). 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 \(c_{\text{max}} \cdot \lfloor \log_{2^{k-1}} 2n \rfloor \). 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 \((\alpha, T)\) be the infinite admissible timed computation in which processes take steps at the same speed in round robin order and each process’s \(i\)th step occurs at time \(i \cdot c_{\text{max}}\). Each consecutive group of steps for \(p_1\) through \(p_{|P|}\) is a round. (Round \(i\) occurs at time \(i \cdot c_{\text{max}}\) and consists of the \(i\)-th step of each process.) Since all port processes should enter idle states by time \(Z\) in \((\alpha, T)\) and all the step time periods are equal to \(c_{\text{max}}\) in \((\alpha, T)\), there are at most \(r = \lceil Z/c_{\text{max}} \rceil\) rounds by time \(Z\) in \((\alpha, T)\).

Fix any port process \(p'\) and change the step time of \(p'\) to be \(c_{\text{max}} \cdot \lfloor \log_{2^{k-1}} 2n \rfloor\), i.e., the steps of \(p'\) occur every \(c_{\text{max}} \cdot \lfloor \log_{2^{k-1}} 2n \rfloor\) 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 \((\alpha', T')\).

Since the steps of \(p'\) occur at different times in \((\alpha', T')\) than they do in \((\alpha, T)\), there will be other steps that are influenced by the steps of \(p'\). These steps will in turn influence others. We say that these steps are “contaminated” by the steps of \(p'\). However, we prove that there is at least one process \(p\) whose steps are not contaminated through time \(Z\). Thus, because all processes are in an idle state at time \(Z\) in \((\alpha, T)\), \(p\) is also in an idle state at time \(Z\) in \((\alpha', T')\). \([\text{???}]\) However, there has not been even a single step of \(p'\) by time \(Z\) in \((\alpha', T')\). This is a contradiction. We now formalize these ideas.

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

A variable \(v\) is contaminated in pseudo-round \(t \geq 1\) of \(\alpha'\) if there exists \(j \leq t\) and process \(p \neq p'\) such that \(v\)'s value in the global state of \(\alpha'\) following \(p\)'s step in pseudo-round \(j\) is not equal
to \( v \)'s value in the global state of \( \alpha \) following \( p \)'s step in round \( j \). We define no variable to be contaminated in pseudo-round 0. A process \( p \) is \textit{contaminated} in pseudo-round \( t \geq 1 \) of \( \alpha' \) if \( p \neq p' \) and there exists \( j \leq t \) such that in pseudo-round \( j \) of \( \alpha' \), \( p \) accesses a variable that is contaminated in pseudo-round \( j \). We define no processes to be contaminated in pseudo-round 0.

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

\[
\begin{align*}
P_0 &= 0, \quad V_0 = 0 \\
V_t &= 2 \cdot P_{t-1} + 1, \quad t \geq 1 \\
P_t &= P_{t-1} + (b - 1) \cdot V_t, \quad t \geq 1
\end{align*}
\]

\textbf{Lemma 3} For all \( t \geq 0 \), \( P_t = \frac{(2b-1)^t - 1}{2} \).

\textbf{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.

\textbf{Lemma 4} \( |P(t)| \leq P_t \) and \( |V(t)| \leq V_t \) for \( 0 \leq t \leq r \), where \( r = \lceil Z/c_{\text{max}} \rceil \).

\textbf{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 \( p \) other than \( p' \): \( p \) 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 \( p' \): \( p' \) fails to access a variable it was supposed to. (Recall that \( p' \) is taking any step until after time \( Z \).)
2. Processes become contaminated as soon as possible. This happens if, as soon as a variable $v$ becomes contaminated, which is due to some process $p$ either accessing it when it was not supposed to or failing to do so, the remaining $b - 1$ processes that can access $v$ do so [??] in the same pseudo-round. (Recall that at most $b$ distinct processes can ever access any given shared variable.)

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

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

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

$$|P(r)| \leq \frac{P_r}{2}$$ by Lemma 4

$$= \frac{(2b-1)^{r-1}}{2}$$ by Lemma 3

$$< \frac{(2b-1)^{b \cdot 2n - 1} - 1}{2}$$ by definition of $r$ and assumption on $Z$

$$\leq \frac{2n - 1}{2}$$

$$< n.$$

Since fewer than $n$ processes are contaminated in pseudo-round $r$, at least one port process $p \neq p'$ is in the same state at the end of pseudo-round $r$ in $\alpha'$ as it is at the end of round $r$ in $\alpha$, namely an idle state. But $p'$ has not taken a step yet. Thus $(\alpha', T')$ is an infinite admissible timed computation that contains fewer than $s$ sessions. Contradiction.

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 [4] and logical OR [5]). 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 [7]; the lower
bound proof is much simpler than for the shared memory case.

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

References


vol. 32, no. 4, 1978, pp. 226–244.


Access Machines without Simultaneous Writes,” SIAM Journal of Computing, vol. 15, no. 1,

pp. 81–91.


191–201.