Interdisciplinary Information Sciences, Vol. 8, No. 2, pp. 187–198 (2002)

Analysis of Some Lockout Avoidance Algorithms for the *k*-Exclusion Problem

Michiko OMORI[1]*, Kumiko OBOKATA[2]†, Kazuhiro MOTEGI[3]‡ and Yoshihide IGARASHI[3]‡

[1]*Business Networks Division, NEC Corp., Abiko, Chiba 270-1198, Japan*
[2]*Semiconductor Company, Sanyo Electric Corp., Oizumi, Gunma 370-0596, Japan*
[3]*Department of Computer Science, Gunma University, Kiryu 376-8515, Japan*

Received May 9, 2002; final version accepted September 9, 2002

We analyze two algorithms for the k-exclusion problem on the asynchronous multi-writer/reader shared memory model and show their correctness. The first algorithm is a natural extension of the n-process algorithm by Peterson for the mutual exclusion algorithm to the k-exclusion problem, and the second algorithm is a combination of the first algorithm and the tournament algorithm by Peterson and Fischer for the mutual exclusion problem. These two algorithms satisfy k-exclusion, and can tolerate up to $k-1$ process failures of the stopping type. The running times by the first algorithm and by the second algorithm are bounded by $(n-k)c + O(n(n-k)^2)l$ and $(\frac{n}{k})^k c + O((\frac{n}{k})^{k+1}k)l$, respectively, even if there exist at most $k-1$ process failures of the stopping type, where n is the number of processes, l is an upper bound on the time between successive two atomic steps for any faultless process, and c is an upper bound on the time that any user spends in the critical region.

KEYWORDS: asynchronous processes, concurrent computation, k-exclusion, lockout avoidance, shared memory

1. Introduction

Mutual exclusion is a problem of managing access to a single indivisible resource that can only support one user at a time. An early algorithm for the mutual exclusion problem was proposed by Dijkstra [8]. Since then the problem has been widely studied for its theoretical interests as well as for its practical applications to distributed systems [5–7, 14, 15, 18–20, 26, 27]. The k-exclusion problem is a natural extension of the mutual exclusion problem. In k-exclusion, some number of processes, specified by parameter k, are allowed to be concurrently inside the critical region, where corresponding users can use the resource. This extension was first defined and solved by Fischer et al. on the shared memory model [11].

The solution by Fischer et al.[11] can tolerate the slowdown or even the crash (i.e., stopping failure) of up to $k-1$ processes, but it was based on the use of a powerful primitive, *read-modify-write*. For a *read-modify-write* shared variable, in one atomic step a process is able to access the shared variable, and to use the variable value and the process state to determine a new variable value and a new process state [21]. Such a shared variable is considered to be too powerful to be realized by less powerful shared variables. The algorithm by Fischer et al. is a *first-in, first-enable* solution to the k-exclusion problem, where by ‘‘*enable*’’, we mean that a process no longer needs to wait for action by any other process before it can go into its critical region. Afek et al. [2] gave another solution to the *first-in, first-enable* k-exclusion problem. Unlike the solution by Fischer et al. this solution does not use a powerful *read-modified-write* primitive. A number of k-exclusion algorithms have been also proposed in the asynchronous network setting [17, 28]. The major paradigms for the resource allocation problem on the asynchronous network setting are the permission based ones and the token based ones. For example, the algorithm proposed by Kakugawa et al. [17] and the algorithm proposed by Raymond [28] are permission based.

On the asynchronous shared memory model, processes take steps at arbitrary speeds. The major difficulty of designing an algorithm on such a model stems from the uncertainty in the order of events. A process can observe the situations of other processes only through the shared memory. Hence, in general a process is uncertain whether other processes are faultless. Even if a process observed that the contents of a shared variable concerning another process are the same at the two different points in time, the former process cannot conclude that the latter process is stopping. It may be moving very slowly, or took quickly a number of steps and then sent the original contents to the shared variable during the time period. In other words, in general a process cannot specify which processes are faulty.

In this paper we consider two algorithms for the k-exclusion problem on the asynchronous multi-writer/reader shared memory model. The formal definition of this model is given in [4, 21]. The first algorithm is a natural extension of the n-process algorithm by Peterson [26] or its accelerated versions by Igarashi and Nishitani [15] for the mutual exclusion

* E-mail: m-oomori@dc.jp.nec.com
† E-mail: OBOK021850@swan.sanyo.co.jp
‡ E-mail: {motegi,igarashi}@comp.cs.gunma-u.ac.jp

problem. The algorithm has been presented in [24, 25]. The second algorithm is a combination of the first algorithm and the tournament algorithm by Peterson and Fischer [27] for the mutual exclusion problem or a variation of the algorithm accelerated by Igarashi, et al. [14]. Although the running times of these two algorithms in the worst case are not as good as the best known algorithm using a bounded concurrent time-stamp scheme [4], their structures are simple and they are reasonably fast in the case where the number of competitors wishing to enter the critical region is small. We show the correctness of these algorithms, and give their running times in the worst case. One of the aims of this paper is to introduce some techniques for analyzing time complexity and for proving correctness of asynchronous algorithms.

We only consider the simplest type of process failures: a stopping failure, whereby a process just stops without warning. We do not consider less well-behaved Byzantine failures. Both the algorithms are immune to stopping failures of fewer than k processes, and they are wait-free if the number of processes attempting to get the critical region is at most k, but in general not *first-in, first-enable*. All logarithms in this paper are to the base 2. The shared memory size for the first algorithm is $(n-k)\lceil\log n\rceil + n\lceil\log(n-k+1)\rceil$ bits. It is particularly space economical for a k such that $n-k=O(1)$. The shared memory size for the second algorithm is not much different from the shared memory size for the first algorithm. We describe our algorithms in terms of I/O automata. A full description of I/O automata is given in [21, 23].

In order to estimate upper bounds on the running times of the algorithms, we impose an upper bound of l on the time between successive atomic steps of each faultless process in the trying region and the exit region, and an upper bound of c on the time that any user spends in the critical region. The running time by the first algorithm and the running time by the second algorithm for the trying regions are bounded by $(n-k)c + O(n(n-k)^2)l$ and $(\frac{n}{k})^k c + O((\frac{n}{k})^{k+1}k)l$, respectively. In general, the worst case running time of a process by the second algorithm is worse than the worst case running time by the first algorithm. However, in the case where the number of competitors is small, the second algorithm is faster than the first algorithm. When $k=1$ (i.e., the mutual exclusion case), the worst case running time of the first algorithm is better than the worst case running time $O(n^2)c + O(n^4)l$ of the original n-process algorithm by Peterson [15, 26] and as good as the worst case running times of the accelerated versions by Igarashi and Nishitani [15]. The worst case running time of the second algorithm for $k=1$ is as good as the worst case running time of the original tournament algorithm by Peterson and Fischer [27] for the mutual exclusion problem, but worse than the worst case running time $(n-1)c + O(n)l$ of its accelerated version by Igarashi et al. [14].

The rest of this paper is organized as follows: Sect. 2 contains the computational model in terms of I/O automata and the definition of k-exclusion problem. In Sect. 3, a lockout avoidance algorithm with $n-k$ levels (i.e., our first algorithm) is given, and in Sect. 4 its correctness is shown. The worst case running time of the algorithm is also given in Sect. 4. In Sect. 5, a group tournament algorithm (i.e., our second algorithm) is given. In Sect. 6, the correctness of the group tournament algorithm is shown, and the worst case running time of the algorithm is also given in Sect. 6. In Sect. 7, we compare the time and space efficiencies of the two algorithms as well as an algorithm using a bounded concurrent time-stamp scheme. Concluding remarks are given in Sect. 8.

2. Preliminary

The computational model used in this paper is the asynchronous multi-writer/reader shared memory model. It is a collection of processes and shared variables. Interactions between a process and its corresponding user are by input actions from the user to the process and by output actions from the process to the user. Each process is considered to be a state machine with arrows entering and leaving the process, representing its input and output actions. All communication among the processes is via the shared memory. The model is called an I/O automaton. This model can be considered to be a full description of the asynchronous shared memory model of Lynch and Fischer [22]. We assume that the order of actions by processes is linearizable. For example, even if two different processes try to write on the same shared variable at almost the same time, one process's writing is earlier than the other process's writing. Thus the contents of the shared variable written by the earlier one is changed to the contents written by the later one even if these two events occur very closely. The reader can find a complete and formal description of the model in [21, 22].

A user with access to the resource is modeled as being in the critical region. When a user is not involved in the resource, it is said to be in the remainder region. In order to gain admittance to the critical region, a process executes a trying protocol. The duration from the start of executing the trying protocol to the entrance of the critical region is called the trying region. After the end of the use of the resource by a user, the corresponding process executes an exit protocol. The duration of executing the exit protocol is called the exit region. These procedures can be repeated in cyclic order, from the remainder region to the trying region, then to the critical region, then to the exit region, and then back again to the remainder region. The k-exclusion problem is to devise a protocol for at most k processes to be concurrently in the critical region. When $k=1$, the k-exclusion problem is the mutual exclusion problem.

We assume that the n processes are numbered $0,\ldots,n-1$. Each process i $(0 \le i \le n-1)$ corresponds to user U_i. The inputs to process i from user U_i are try_i which means a request by user U_i for access to the resource, and $exits_i$ which means an announcement of the end of the use of the resource by U_i. The outputs from process i to user U_i are $crit_i$ which means the grant of the resource to U_i, and rem_i which tells U_i that it can continue with the rest of its work not concerning the use of the resource. These are external actions of the shared memory system.

The system to solve the *k*-exclusion problem should satisfy the following conditions.

(1) There is no reachable system state in which more than k processes are in the critical region.
(2) If at least one faultless process is in the trying region and less than k processes are in the critical region, then at some later point some process enters the critical region.
(3) If a faultless process is in the exit region, then at some later point the process enters the remainder region.
(4) If all users always return the resource and if the number of faulty processes of the stopping type is at most $k - 1$, then any faultless process wishing to enter the critical region eventually does so.

Conditions (1), (2), (3) and (4) above are called *k*-exclusion, progress for the trying region, progress for the exit region, and *k*-lockout avoidance, respectively. Note that *k*-lockout avoidance is a property stronger than lockout freedom. That is, *k*-lockout avoidance is not only fair to different users but also immune to at most $k - 1$ stopping failures of processes.

The following procedure (n, k)-*NAIVE* is a naive modification of the *n*-process algorithm by Peterson [26]. When $k = 1$, it is the *n*-process algorithm by Peterson for the mutual exclusion problem.

```
procedure $(n,k)$-NAIVE
shared variables
  for every $s \in \{1,\dots,n-k\}$:
    $turn(s) \in \{0,\dots,n-1\}$, initially arbitrary, writable and readable by all processes;
  for every $i \in \{0,\dots,n-1\}$:
    $level(i) \in \{0,\dots,n-k\}$, initially 0, writable by $i$ and readable by all $j \neq i$;

process $i$
    input actions {inputs to process $i$ from user $U_i$}:
      $try_i$, $exit_i$;
  output actions {outputs from process $i$ to user $U_i$}:
    $crit_i$, $rem_i$;

  ** Remainder region **
  $try_i$:
    for $s = 1$ to $n-k$ do
      begin
        $level(i) := s$;
        $turn(s) := i$;
        waitfor $[\forall j \neq i : level(j) < s]$ or $[turn(s) \neq i]$
      end;
  $crit_i$;
  ** Critical region **
  $exit_i$:
    $level(i) := 0$;
  $rem_i$;
```

By (n, k)-*NAIVE*, a process in the trying region climbs $n - k$ levels until it reaches the critical region. The value of $level(i)$ shows a level which process i reaches. "$turn(s) = i$" shows that process i is the last process which has reached level s in the trying region. Procedure (n, k)-*NAIVE* is *k*-exclusion, and if there are no faulty processes then it is lockout-free. That is, it satisfies condition (1) above, and if there are no faulty processes then any process in the trying region enters eventually the critical region. This means that (n, k)-*NAIVE* is a correct algorithm for the *k*-exclusion problem under the condition that there are no faulty processes. However, (n, k)-*NAIVE* does not provide *k*-lockout avoidance, since if a process crashes at a level, say level s, in the trying region then any process at a level below level s in the trying region is blocked unless some process enters the level and changes the value of shared variable $turn(s)$. The difference between lockout freedom and lockout avoidance may become clear from this explanation. Lockout freedom guarantees the progress of any process in the trying region if there are no faulty processes, but may not guarantee its progress when there are faulty processes. On the other hand, *k*-lockout avoidance guarantees the progress of any faultless process in the trying region if the number of faulty processes is less than k. It is not difficult to show that if there are no faulty processes then the running time by (n, k)-*NAIVE* for the trying region of any process is bounded by $O((n - k)^2)c + O(n(n - k)^3)l$. We can obtain this bound in a similar way to the way given in [15].

3. A Lockout Avoidance Algorithm with $n - k$ Levels

As stated in the previous section, the first condition in the waitfor statement (i.e., $[\forall j \neq i : level(j) < s]$) of the procedure (n, k)-*NAIVE* (as well as the *n*-process algorithm by Peterson [26]) may prevent the move of process i to a higher level when some process at a higher level stops or slow down. We propose a simple lockout avoidance algorithm for the k-exclusion problem on the multi-writer/reader shared memory model. This algorithm is similar to the n-process algorithm by Peterson [26], but the condition in the waitfor statement is modified so that k-lockout avoidance is satisfied. In the n-process algorithm by Peterson [15, 26] or the second accelerated version by Igarashi and Nishitani [15] for the mutual exclusion problem, the condition "$[\forall j \neq i : level(j) < s]$ or $[turn(s) \neq i]$" should be satisfied to move process i from level s to level $s+1$. We cannot use this condition in our algorithm, since k-lockout avoidance is required. The condition "$[\forall j \neq i : level(j) \notin \{s, s+1\}]$ or $[turn(s) \neq i]$" used in the first accelerated version by Igarashi and Nishitani [15] cannot be used either in our algorithm from the same reason. We modify the first part of the condition in the waitfor statement for our purpose. The lockout avoidance algorithm for solving k-exclusion among n processes is described in the following procedure named (n,k)-*EXCL*. In (n,k)-*EXCL*, a process at level s, say process i, tests whether the number of other processes at level s or above level s is at most $n-s-1$, or whether the contents of $turn(s)$ are not i. When the test result is affirmative, the process is allowed to move to level $s+1$ (or granted to enter the critical region in the case where $s = n-k$). In this way, the process moves to higher levels one by one, and eventually reaches the position where it is granted to enter the critical region.

```
procedure $(n,k)$-EXCL
shared variables
  for every $s \in \{1,\dots,n-k\}$:
    $turn(s) \in \{0,\dots,n-1\}$, initially arbitrary, writable and readable by all processes;
  for every $i \in \{0,\dots,n-1\}$:
    $level(i) \in \{0,\dots,n-k\}$, initially 0, writable by $i$ and readable by all $j \neq i$;

process $i$
  input actions {inputs to process $i$ from user $U_i$}:
    $try_i$, $exit_i$;
  output actions {outputs from process $i$ to user $U_i$}:
    $crit_i$, $rem_i$;

  ** Remainder region **
  $try_i$:
    for $s = 1$ to $n-k$ do
      begin
        $level(i) := s$;
        $turn(s) := i$;
        waitfor $[|\{j | j \neq i : level(j) \geq s\}| \leq n-s-1]$ or $[turn(s) \neq i]$
      end;
  $crit_i$;
  ** Critical region **
  $exit_i$:
    $level(i) := 0$;
  $rem_i$;
```

For each level s $(1 \leq s \leq n-k)$, statement "waitfor$[|\{j \mid j \neq i : level(j) \geq s\}| \leq n-s-1]$ or $[turn(s) \neq i]$" in (n,k)-*EXCL* is not atomic step. It consists of a number of atomic steps as given in the following procedure $check_i(n,s)$ for checking the condition by process i at level s. Note that procedure $check_i(n,s)$ is not a snapshot algorithm. The reader may be referred [1, 3, 4, 21] for snapshot algorithms. In other words, for each s, the contents of the local variable *count* is not guaranteed to represent the exact number of other processes that are located at level s or above level s at a particular point in time. Since the shared memory used in this paper is not the *read–modified–write* model, this uncertainty cannot be avoided unless we use a snapshot algorithm. Nevertheless the uncertain information about the number of processes at level s or above level s in the local variable *count* together with the contents of shared variable $turn(s)$ is good enough to guarantee k-exclusion as proved in Sect. 4. The use of a snapshot algorithm may make the space efficiency and the time efficiency worse.

```
procedure $check_i(n,s)$
  shared variables
    $turn(s) \in \{0,\dots,n-1\}$
    for every $j \in \{0,\dots,n-1\}$:
      $level(j) \in \{0,\dots,n-k\}$

  $L$:
    $count := 0$;
    for each $j \in \{0,\dots,i-1,i+1,\dots,n-1\}$ do
```

```
        begin
            $v := level(j)$;
            if $v \geq s$ then $count := count + 1$;
        end;
    $v := turn(s)$;
    if $count \leq n - s - 1$ or $v \neq i$ then return true
        else goto $L$;
```

In an execution by (n, k)-*EXCL*, process i is said to be a winner at level s if it has left the waitfor statement in the sth loop of the **for** statement. Note that if a process is a winner at level s then any $1 \leq t \leq s$, the process is also a winner at level t. For each i $(0 \leq i \leq n-1)$, when process i has entered the exit region, the qualification for the winner of process i is canceled by resetting $level(i) := 0$. The next example may be helpful for the reader to understand the behavior of processes in an execution by (n, k)-*EXCL*.

Example 1. *The following scenario is possible in a fair execution by* (n, k)-*EXCL. Let* $n = 4$ *and* $k = 2$. *Assume that process 0, process 1, process 2 and process 3 entered the trying region in this order. Then process 0, process 1 and process 2 became winners at level 1 by observing that the latter part of the condition in the waitfor statement is satisfied (i.e.,* $turn(1) \neq i$ *for* $i = 0, 1, 2$*). Assume that these winners moved to level 2 in the order of process 2, process 0 and process 1. Then process 0 and process 2 became the winners at level 2 by observing that the latter part of the condition in the waitfor statement is satisfied, and these two winners were granted to enter the critical region. After at least one of these two processes exited the critical region, process 1 became a winner at level 2 by observing that the former part of the condition in the waitfor statement (i.e.,* $|\{j \mid j \neq 1 : level(j) \geq 2\}| \leq 1$*) is satisfied, and then it is granted to enter the critical region.*

4. Correctness for (n, k)-*EXCL*

Without loss of generality, we may assume that any process is initially in the remainder region. However, the starting time points of processors are in general different. For a faultless process it takes either an infinite number of steps or it ends in the remainder region. Throughout this paper, we assume that the number of process failures is always at most $k - 1$.

Lemma 1. *In any reachable system state of* (n, k)-*EXCL, for any* $1 \leq s \leq n - k$ *there are at most* $n - s$ *winners at level* s.

Proof. The proof is an induction on levels. For the sake of the contrary, we assume that there are n winners at level 1 at some point in time τ in an execution. Suppose that $turn(1) = i$ at τ. Since all the processes are winners at level 1 at τ, such process i exists. Then process i has set most recently $turn(1)$ before τ. For any $j \neq i$, process j set $level(j)$ to be 1 before the time when $turn(1)$ was set to be i. Hence, for each $j \neq i$, $level(j) \neq 0$ during the time period from when process i set $turn(1)$ to be i until τ. Then both the former part and the latter part of the condition for process i in the waitfor statement at level 1 in (n, k)-*EXCL* are not satisfied during the time period. Hence, process i cannot be a winner at level 1 at τ. This is contrary to the assumption that all the processes are winners at level 1 at τ.

Let $s \geq 2$. Assume that for any $r < s$, there are at most $n - r$ winners at level r in any reachable system state. For the sake of the contrary, we assume that there are more than $n - s$ winners at level s at a point in time τ. Among the winners at level s at τ, let process i be the process that set $turn(s)$ most recently before τ. Then any process j $(j \neq i)$ in the set of the winners at level s at τ, $level(j) \geq s$ during the time period from when process i set $turn(s)$ to be i until τ. Hence, during this time period the former part of the condition for process i in the waitfor statement at level s in (n, k)-*EXCL* is not satisfied. Since process i is also a winner at level s at τ, the latter part of the condition for process i in the waitfor statement must be satisfied at some point in time during the time period. This means that some process, say process p, not in the set of the winners at level s at τ must set $turn(s)$ to be p at some point in time during the time period. Then process p is a winner at level $s - 1$ at that point in time. Then the number of winners at level $s - 1$ at that point in time is more than $n - s + 1$. This is contrary to the induction hypothesis. Therefore. the assertion of the lemma holds true for any $1 \leq s \leq n - k$. □

The existence of any number of process failures of the stopping type including slowdown processes does not affect the proof of Lemma 1. Hence, the next theorem is from Lemma 1.

Theorem 2. (n, k)-*EXCL guarantees k-exclusion even if any number of process failures of the stopping type exist.*

It is obvious that (n, k)-*EXCL* guarantees progress for the exit region. In order to prove progress for the trying region and k-lockout avoidance, it is enough to give a time bound for the trying region in the case where at most $k - 1$

stopping failures of processes exist.

Theorem 3. *Suppose that the number of stopping failures of processes is always at most $k-1$. In any execution by (n,k)-EXCL, the time from when a faultless process enters the trying region until the process enters the critical region is at most $(n-k)c + O(n(n-k)^2)l$.*

Proof. For each s, $1 \le s \le n-k$, define $F(s)$ to be the maximum time from when a faultless process has entered the competition at level s until it becomes a winner at level s. The worst situation is the case where $k-1$ processes are stopping at the entrance of the critical region or in the critical region. Such $k-1$ faulty processes are winners at every level by the definition of winners, and we may consider these $k-1$ faulty processes are always competitors at every level. Note that in general a process cannot decide whether other processes are moving or stopping. The slowest situation for a faultless process at level $n-k$ in the trying region to reach the critical region is the case where there are $k+1$ winners at level $n-k-1$ and all of them are competitors including the process itself at level $n-k$, and then the process becomes a loser among them at level $n-k$. The time to recognize the existence of k winners at level $n-k$ is at most $(2n+3)l$, where we count 2 atomic steps for setting *level* values and $turn(n-k)$ at level $n-k$, and at most $2n+1$ steps to test the condition in the waitfor statement to be at most $2n+1$ (i.e., computing time for each faultless process to count the number of $level(j)$'s with values not less than $n-k$ and to check the value of $turn(n-k)$). Since the number of stopping failures of processes is at most $k-1$, it is not possible that all the k winners are faulty processes. The time from the end of the competition at level $n-k$ until the time when at least one faultless winner reset its *level* value to 0 in the exit region is at most $3l+c$. Then the loser at level $n-k$ becomes a winner at the level within $(2n+1)l$ time after the reset of the winner's *level* value. Hence, $F(n-k) \le (4n+7)l + c$.

The slowest situation for a faultless process at level s, $1 \le s \le n-k-1$, to become a winner at level s is the case where there are $n-s+1$ competitors at level s, and then the faultless process becomes a loser at level s. Note that some of the winners may stop due to their stopping failures after they become winners at level s. In particular, we may assume the case where $k-1$ processes are stopping at the entrance of the critical region. In this case the time to recognize the existence of the $n-s$ winners at level s is also at most $(2n+3)l$. The faultless loser at level s will become a winner when a new process joins the competition at level s and then $turn(s)$ to be its process name, or when at least one winner at the level eventually resets its *level* value to 0 in the exit region. In the worst case, it is sufficient to consider the latter case. In order that the loser at level s becomes a winner at the level, it must wait until at least one winner at level s exits from the critical region. Since the number of faulty processes is at most $k-1$, there exists a faultless winner at the level. The quickest winner at level s can reach the critical region within $(n-k-s)(2n+3)l$ time. Hence, the loser at level s can move as a winner at level s to level $s+1$ within $F(s) \le (n-k-s+1)(2n+3)l + (2n+1)l + 3l + c = (n-k-s+2)(2n+3)l + l + c$ time.

Thus the time from when a faultless process has entered the trying region until the process enters the critical region is bounded by

$$\sum_{s=1}^{n-k} F(s) \le \sum_{s=1}^{n-k-1} [(n-k-s+2)(2n+3)l + l + c)] + (4n+7)l + c \le (n-k)c + O(n(n-k)^2). \qquad \square$$

The following theorem is immediate from Theorem 2 and Theorem 3.

Theorem 4. *(n,k)-EXCL solves the k-exclusion problem, and it is k-lockout avoidance.*

The running time of (n,k)-*EXCL* in the trying region for a process is bounded by $(n-k)c + O(n(n-k)^2)l$. It is possible to construct a scenario of the behavior of the n processes so that it leads the running time of a process in the trying region as slow as the running time bound given in Theorem 3. Hence, we can claim that the bound given in Theorem 3 is tight. We will compare its efficiency with the efficiencies of other algorithms in Sect. 7.

5. A Group Tournament Algorithm

We denote the null sequence (i.e., the sequence with length 0) by λ. For a set of processes G, the lower half of G means the set of $\lceil |G|/2 \rceil$ lower numbered processes of G, and the upper half of G means the set of $\lfloor |G|/2 \rfloor$ higher numbered processes of G. For example, if $G = \{8, \ldots, 14\}$ then the lower half of G is $\{8, 9, 10, 11\}$ and the upper half of G is $\{12, 13, 14\}$.

Let $T^{(n,k)}$ be the complete binary tree with depth $\lfloor \log(n/(k+1)) \rfloor$. Such a tree is called a tournament tree. Each node of $T^{(n,k)}$ is labeled by the following rule:
(1) The root of the tree is labeled by λ.
(2) The left son and the right son of a node with label x are labeled by $x0$ and $x1$, respectively.

Let $bl(n,k) = \lfloor \log(n/(k+1)) \rfloor$. For each binary sequence x with a length not longer than $bl(n,k)$, G_x is a set of processes defined as follows: $G_\lambda = \{0, \cdots, n-1\}$. If $x = x'0$ then G_x is the lower half of $G_{x'}$, and if $x = x'1$ then G_x is the upper half of $G_{x'}$. For each node x of $T^{(n,k)}$, the node x is associated with the set of processes G_x. For pairs of integers we use the lexicographic order. That is, $(i, j) < (i', j')$ if and only if $i < i'$, or $i = i'$ and $j < j'$. For each pair of $0 \le i \le n-1$ and $0 \le t \le bl(n, k)$, $g(i, t)$ is defined to be x such that $|x| = t$ and i belongs to G_x. From the definitions of G_x and $bl(n, k)$, $g(i, t)$ is a well defined function on $\{(i, t) | 0 \le i \le n-1, 0 \le t \le bl(n, k)\}$.

Example 2. *Let $n = 17$ and $k = 3$. Then $T^{(17,3)}$ is the complete binary tree with depth $bl(17, 3) = 2$. $G_\lambda = \{0, \ldots, 16\}$, $G_0 = \{0, \ldots, 8\}$, $G_1 = \{9, \ldots, 16\}$, $G_{00} = \{0, \ldots, 4\}, G_{01} = \{5, \ldots, 8\}$, $G_{10} = \{9, \ldots, 12\}$, $G_{11} = \{13, \ldots, 16\}$, $g(5, 2) = 01, g(5, 1) = 0$, and $g(5, 0) = \lambda$.*

The next lemma is immediate.

Lemma 5. *For any leaf x of $T^{(n,k)}$, $|G_x|$ is $\lceil n/2^{bl(n,k)} \rceil$ or $\lfloor n/2^{bl(n,k)} \rfloor$, and $k + 1 \le |G_x| \le 2k + 2$.*

The group tournament algorithm works in the following way. For a leaf x of $T^{(n,k)}$, a process i in G_x joins the competition for G_x whenever process i enters the trying region. We apply the technique used in procedure (n, k)-*EXCL* to the competitions. Each process engages in a series of $k \cdot bl(n, k) + |G_{g(i,bl(n,k))}| - k$ competitions. Then the number of final winners of the competitions for G_x is at most k. For an inner node x of $T^{(n,k)}$, final winners from G_{x0} and G_{x1} can join the competitions for G_x. Then for these winners from the competitions for G_{x0} and G_{x1}, we apply again the technique used in procedure (n, k)-*EXCL* to the competitions for G_x. Repeating this way, we can choose at most k final winners of the competition for G_λ. These final winners are allowed to enter the critical region.

Hereafter, for simplicity we assume that $n = 2^t(2k)$ for a nonnegative integer t. Then we may consider only the case where for each leaf x of $T^{(n,k)}$, G_x has $2k$ processes. For a general value for n, we need only minor modifications of the group tournament algorithm and proofs given in the next section. Such modifications are tedious work and leave them for the reader. The following procedure (n, k)-*GTEX* is the group tournament algorithm for the k-exclusion problem, where $n = 2^{bl(n,k)}(2k)$.

```
procedure $(n, k)$-GTEX
shared variables
  for every $(x, s)$ such that $0 \le |x| \le bl(n, k), 1 \le s \le k$ :
    $turn(x, s) \in G_x$, initially arbitrary, writable and readable by all processes in $G_x$;
  for every $i \in G_\lambda$:
    $level(i) \in \{(a, b) | 0 \le a \le bl(n, k), 1 \le b \le k\}$,
    initially $(0, 0)$, writable by $i$ and readable by all $j \ne i$;

process $i$
  input actions {inputs to process $i$ from user $U_i$}:
    $try_i$, $exit_i$;
  output actions {outputs from process $i$ to user $U_i$}:
    $crit_i$, $rem_i$;

  ** Remainder region **
  $try_i$:
    for $p = 0$ to $bl(n, k)$ do
      for $q = 1$ to $k$ do
        begin
          $level(i) := (p, q)$;
          $turn(g(i, bl(n, k) - p), q) := i$;
          waitfor $[|\{j | j \ne i, j \in G_{g(i,bl(n,k)-p)} : level(j) \ge (p, q)\}| \le 2k - q - 1$, or
            $turn(g(i, bl(n, k) - p), q) \ne i]$
        end;
  $crit_i$;
  ** Critical region **
  $exit_i$:
    $level(i) := (0, 0)$;
  $rem_i$;
```

As the waitfor statement in (n, k)-*EXCL*, the waitfor statement in (n, k)-*GTEX* above is not an atomic step. It consists of a number of atomic steps. Testing in the waitfor statement must alternate in some systematic way, for example, it can be done in a cycle where the *level*'s are first tested, and then *turn* is tested as given in the following procedure $gcheck_i(n, k, p, q)$. It checks the condition in the waitfor statement by process i at level q for the competitions in $G_{g(i,bl(n,k)-p)}$. Note again that the contents of local variable *count* does not necessarily represent the exact number of other processes that locate at the same level as process i or above the level of process i. Since the shared memory in this paper is not the *read–modified–write* type, this uncertainty cannot be avoided unless we use a snapshot algorithm. As (n,k)-*EXCL*, such uncertain information about the number of processes with *level* values satisfying the condition is good enough for (n,k)-*GTEX* to guarantee k-exclusion as proved in Sect. 6.

```
procedure $gcheck_i(n,k,p,q)$
  shared variables
    $turn(g(i,bl(n,k)-p),q) \in G_{g(i,bl(n,k)-p)}$;
    for every $j \in G_{g(i,bl(n,k)-p)}$:
      $level(j) \in \{(a,b) | 0 \le a \le bl(n,k), 1 \le b \le k\}$;

  $L$:
    $count := 0$;
    for each $j \neq i, j \in G_{g(i,bl(n,k)-p)}$ do
      begin
        $v := level(j)$;
        if $v \ge (p,q)$ then $count := count + 1$;
      end;
    $v := turn(p,q)$;
    if $count \le 2k - q - 1$ or $v \neq i$ then return true
      else goto $L$;
```

6. Correctness for (n,k)-*GTEX*

In an execution of (n,k)-*GTEX*, process i is said to be a winner at (p,q) for $G_{g(i,bl(n,k)-p)}$ if the process has left the waitfor statement in the qth inner **for** loop of the pth outer **for** loop. If a process i is a winner at (p,q) for $G_{g(i,bl(n,k)-p)}$, then for any $(p',q') \le (p,q)$ the process is also a winner at (p',q') for $G_{g(i,bl(n,k)-p')}$. For each i $(0 \le i \le n-1)$, when process i has entered the exit region, the qualification as a winner for process i is canceled by resetting $level(i) := (0,0)$. In an execution by (n,k)-*GTEX*, if we say a competition at (p,q), it means a competition at level q of a group competition in G_x associated with a node in depth $bl(n,k) - p$ of $T^{(n,k)}$. That is, it is a competition in the qth inner **for** loop of the pth outer **for** loop of (n,k)-*GTEX*.

The competition in each group (i.e., the competition specified by the inner **for** loop in (n,k)-*GTEX*) is essentially the same as an execution by $(2k,k)$-*EXCL*. Hence, the next lemma holds true from Lemma 1.

Lemma 6. *In any reachable system state of* (n,k)-*GTEX, for any* (p,q) $(0 \le p \le bl(n,k), 1 \le q \le k)$ *and any binary sequence* x *of length* $bl(n,k) - p$, *the number of winners in* G_x *at* (p,q) *is at most* $2k - q$.

The existence of any number of stopping process failures does not affect the proof of Lemma 6. The next theorem is therefore immediate from Lemma 6.

Theorem 7. (n,k)-*GTEX guarantees k-exclusion even if any number of process failures of the stopping type exist.*

It is obvious that (n,k)-*GTEX* guarantees progress for the exit region. In order to prove progress for the trying region and k-lockout avoidance, it is enough to give a time bound for the trying region in the case where at most $k-1$ process failures of the stopping type exist.

Let us define a function $f_s(n,k)$ as follows:

$$\begin{aligned} f_s(n,k) &= (2^0 + 2^1 + \cdots + 2^{k-1}) + (2^{k+1} + 2^{k+2} + \cdots + 2^{2k}) \\ &\quad + (2^{2(k+1)} + 2^{2(k+1)+1} + \cdots + 2^{3k+1}) + \cdots \\ &\quad + (2^{bl(n,k)(k+1)} + 2^{bl(n,k)(k+1)+1} + \cdots \\ &\quad + 2^{bl(n,k)(k+1)+k-1}). \end{aligned}$$

Lemma 8. *Let* $n = (2k) \cdot 2^t$ *for some non-negative integer* $t \ge 0$. *Then* $f_s(n,k) = O((\frac{n}{k})^{k+1})$.

Proof. By the definition of $f_s(n,k)$,

$$f_s(n,k) = (2^0 + 2^1 + \cdots + 2^{k-1})(2^0 + 2^{k+1} + \cdots + 2^{bl(n,k)(k+1)}).$$

Since $(2^0 + 2^1 + \cdots + 2^{k-1}) = 2^k - 1$ and $bl(n,k) = t$,

$$f_s(n,k) \le 2^k(2^0 + 2^{k+1} + \cdots + 2^{bl(n,k)(k+1)}) = O(2^{(bl(n,k)+1)(k+1)}) = O(2^{(t+1)(k+1)}).$$

Since $2^{t+1} = (n/k)$, $f_s(n,k) = O((n/k)^{k+1})$. □

Theorem 9. *Suppose that the number of stopping failures of processes is always at most* $k-1$. *In any execution by* (n,k)*-GTEX, the time from when a faultless process enters its trying region until the process enters its critical region is at most* $(\frac{n}{k})^k c + O((\frac{n}{k})^{k+1} k) l$.

Proof. Define $T(p,q)$ to be the maximum time from when a process reaches the entry of a competition at (p,q) until it enters the critical region. Then $T(0,1)$ is the maximum time from when a process enters the trying region until it enters the critical region. We want to bound $T(0,1)$.

We first bound $T(bl(n,k),k)$. Consider the slowest case for process i in a competition at $(bl(n,k),k)$ to enter its critical region. In such a case, $k+1$ processes including process i have entered the competition at $(bl(n,k),k)$ and process i will be the last winner among them at the competition. When one of the winners earlier than process i at the competition enters the exit region after spending in the critical region, process i is made possible to become a winner at the competition. Since we assume that there are at most $k-1$ stopping failures of processes, such an earlier winner exists. The fact that some of earlier winners have entered their exit region can be detected by process i in the test for the condition in the waitfor statement. Hence, we have

$$T(bl(n,k),k) \le (2n + d_1)l + c,$$

where d_1 is an appropriate constant independent of n and k.

For $(bl(n,k),q), 1 \le q \le k-1$, consider again the slowest case for a process i in a competition at $(bl(n,k),q)$ to enter the critical region. In such a case $2k-q+1$ processes including process i have entered the competition at $(bl(n,k),q)$. The quickest process among the competitors at $(bl(n,k),q)$ can reach the critical region within $(n+d_1)l + T(bl(n,k),q+1)$ time. Since the number of stopping failures of processes is at most $k-1$, such a process exists. Any faultless competitor at $(bl(n,k),q)$ will be made possible to become a winner at the competition within further $(n+d_1)l + c$ time. Hence, for each $2 \le q \le k$,

$$T(bl(n,k),q) \le 2T(bl(n,k),q+1) + (2n+2d_1)l + c.$$

The number of *level*'s tested by a process in the waitfor statement in each loop for the competition at $(p,1)$ is $(n/2^{bl(n,k)-p-1}) - 1$. Then by a similar argument, for each $0 \le p \le bl(n,k)$ and each $1 \le q \le k$,

$$T(p,q) \le 2T(p,q+1) + \left(\frac{2n}{2^{bl(n,k)-p}} + 2d_1\right)l + c,$$

and for each $0 \le p \le bl(n,k) - 1$,

$$T(p,k) \le 2T(p+1,1) + \left(\frac{2n}{2^{bl(n,k)-p}} + 2d_1\right)l + c.$$

Let d be an appropriate constant such that $4k + 2d_1 \le d \cdot k$. Then we have the following inequalities.

$$\begin{aligned}
T(0,1) &\le 2T(0,2) + c + d \cdot k \cdot l \\
&\le 2^2 T(0,3) + (2^0 + 2^1)c + (2^0 + 2^1) d \cdot k \cdot l \\
&\le \cdots \\
&\le 2^{k-1} T(0,k) + (2^0 + 2^1 + \cdots + 2^{k-2})c + (2^0 + 2^1 + \cdots + 2^{k-2}) d \cdot k \cdot l \\
&\le 2^k T(1,1) + (2^0 + 2^1 + \cdots + 2^{k-1})c + (2^0 + 2^1 + \cdots + 2^{k-1}) d \cdot k \cdot l \\
&\le 2^{k+1} T(1,2) + (2^0 + 2^1 + \cdots + 2^k)c + (2^0 + 2^1 + \cdots + 2^{k-1} + 2^{k+1}) d \cdot k \cdot l \\
&\le \cdots \\
&\le (2^0 + 2^1 + \cdots + 2^{(bl(n,k)+1)k-1})c + f_s(n,k) d \cdot k \cdot l \\
&\le \left(\frac{n}{k}\right)^k c + O\left(\left(\frac{n}{k}\right)^{k+1} k\right) l.
\end{aligned}$$

□

We have the following theorem from Theorem 7 and Theorem 9.

Theorem 10. (n,k)*-GTEX solves the k-exclusion problem, and it is k-lockout avoidance.*

7. Comparison among Algorithms

We first compare the shared memory sizes between (n, k)-*EXCL* and (n, k)-*GTEX*. The former algorithm uses $n - k$ shared variables taking n distinct values ($turn(s)$, $1 \leq s \leq n - k$) and n shared variables taking $n - k + 1$ distinct values ($level(i)$, $1 \leq i \leq n$). Thus the total number of shared variables for (n, k)-*EXCL* is $2n - k$, and the total shared memory size for (n, k)-*EXCL* is $(n - k)\lceil \log n \rceil + n\lceil \log(n - k + 1) \rceil$ bits. For the latter algorithm, the number of $turn(x, s)$'s (x is a binary sequence with length at most $bl(n, k)$ and $1 \leq s \leq k$) is $k(2^0 + 2^1 + \cdots + 2^{bl(n,k)}) = k(2^{bl(n,k)+1} - 1) = k((n/k) - 1) = n - k$, and the number of $level(i)$'s ($0 \leq i \leq n - 1$) is n. Hence, the total number of shared variables used by (n, k)-*GTEX* is $2n - k$. For (n, k)-*GTEX*, each shared variable $level(i)$ ($0 \leq i \leq n - 1$) takes $k \log(n/k) + 1$ distinct values, and a shared variable $turn(x, q)$ such that $|x| = p$ takes 2^p distinct values. Hence, the total shared memory size (bits) for (n, k)-*GTEX* is not more than

$$n \log\left(k \log\left(\frac{n}{k}\right) + 1\right) + k\left(\log n + 2 \log\left(\frac{n}{2}\right) + 2^2 \log\left(\frac{n}{2^2}\right) + \cdots + 2^{bl(n,k)} \log(2k)\right)$$
$$\leq n \log\left(k \log\left(\frac{n}{k}\right) + 1\right) + (n - k) \log n.$$

Comparing the total shared memory sizes for (n, k)-*EXCL* and (n, k)-*GTEX*, in general the difference is not large. For example, if $k = O(1)$, the total shared memory size for (n, k)-*EXCL* is larger than the total shared memory size for (n, k)-*GTEX*, but the total size for the former is smaller than twice of the total size for the latter. In the case where $n - k = O(1)$, the total shared memory size for (n, k)-*EXCL* is only $O(n)$ bits. In this case, (n, k)-*EXCL* is more space economical than (n, k)-*GTEX*.

We next compare the total shared memory size for (n, k)-*EXCL* with the total shared memory size for a k-exclusion algorithm using a bounded concurrent time-stamp scheme. A time-stamp scheme is like a ticket machine. It enables the representation of temporal relations among system objects. If any process can scan the existing tickets concurrently with other processes and take a new ticket for its own, such a time-stamp scheme is called a concurrent time-stamp scheme. Using a concurrent time-stamp scheme, an algorithm for the k-exclusion problem can be designed [2, 4, 11, 12]. A number of bounded concurrent time-stamp schemes have been proposed [9–11, 13, 16]. When we use the bounded concurrent time-stamp scheme (called the colored ticket algorithm) given in [11] for the *read-modify-write* shared memory model, the number of distinct values of the shared memory is $O(\binom{2k}{k} kn^2)$ [11]. Hence, if we are allowed to use the *read-modify-write* primitive, the total size of the shared memory for the bounded concurrent time-stamp scheme is $O(k \log k + \log n)$. The shared memory used by the bounded concurrent schemes in [9, 10, 16] is the single-writer/multi-reader shared memory type. Since realization of single-writer/reader shared memory is easier than multi-writer/reader shared memory, an algorithm using such a time-stamp scheme for the k-exclusion problem is advantageous over (n, k)-*EXCL* and (n, k)-*GTEX*. However, such a bounded concurrent time-stamp scheme contains complicated construction of precedence graphs. In the case of the single-writer/multi-reader shared memory model, we need n shared variables for bounded concurrent time-stamps, where each shared variable takes $O(kn)$ distinct values [4]. Hence, the total size of shared memory for the k-exclusion algorithm using such a bounded concurrent time-stamp scheme [4] is $O(n \log n)$. In general, the total size of shared memory for the k-exclusion algorithm using a time-stamp scheme on the single-writer/multi-reader shared memory model is not much different from the total size of shared memory for (n, k)-*EXCL* or (n, k)-*GTEX*. In the case where $n - k = O(1)$, (n, k)-*EXCL* needs only $O(n)$ total size of shared memory. In this case (n, k)-*EXCL* is space economical compared with the method using such a bounded concurrent time-stamp scheme for the k-exclusion problem.

Finally we compare running times of the algorithms. When $n - k = O(n)$, the worst case running time of a process by (n, k)-*EXCL* is $(n - k)c + O(n^3)l$. When $n - k = O(1)$, the worst case running time of a process by (n, k)-*EXCL* is $O(1)c + O(n)l$. Hence, when $n - k = O(1)$, (n, k)-*EXCL* is very fast. Comparing Theorem 3 and Theorem 9, the worst case running time of a process by (n, k)-*GTEX* is worse than the worst case running time of a process by (n, k)-*EXCL*. However, this does not simply imply that (n, k)-*GTEX* is less efficient than (n, k)-*EXCL*. The average running time by (n, k)-*GTEX* and the average running time by (n, k)-*EXCL* seem to be not much different when many processes are always seeking the critical regions, although we have not strictly analyze their average case running times. In the case where the number of competitors is always small, (n, k)-*GTEX* is even faster than (n, k)-*EXCL*. For example, consider a time duration where the number of competitors is not more than k. During such a duration, the worst case running time of a process from the entrance of the trying region to the entrance of the critical region by (n, k)-*EXCL* is bounded by $((n - k)(n + 2) + 1)l$. On the other hand, during such a duration, the worst case running time of a process from the entrance of the trying region to the entrance of the critical region by (n, k)-*GTEX* is bounded by

$$((2 + (2k) \cdot 2^0) + (2 + (2k) \cdot 2^1) + \cdots + (2 + (2k) \cdot 2^{bl(n,k)} + 1))l$$
$$\leq (2(bl(n, k) + 1) + 2k(2^{bl(n,k)+1} - 1) + 1)l = \left(2 \log\left(\frac{n}{k}\right) + 2(n - k) + 1\right)l.$$

Hence, in the case where the number of competitors is fairly small, (n,k)-*GTEX* is faster than (n,k)-*EXCL*.

When $k = 1$, the *k*-exclusion problem is the mutual exclusion problem. The worst case running time for the trying region by $(n,1)$-*EXCL* is $(n-1)c + O(n^3)l$ which is better than the worst case running time for the trying region by the *n*-process algorithm by Peterson [15, 26], $O(n^2)c + O(n^4)l$. The worst case running time by $(n,1)$-*EXCL* is the same as within a constant factor as the worst case running time by accelerated version given in [15]. The worst case running time for the trying region by $(n,1)$-*GTEX* is the same within a constant factor as the worst case running time for the trying region by the tournament algorithm for the mutual exclusion problem by Peterson and Fischer [27], but worse than the worst case running time $(n-1)c + O(n)l$ of the accelerated version by Igarashi et al. [14]

The running time of a process in the trying region by an algorithm using a bounded concurrent time-stamp scheme [4] for the *k*-exclusion problem is bounded by $O(\frac{n}{k}(c+nl) + ts(n)l)$, where $ts(n)$ is the running time of the time-stamp scheme to obtain a new time-stamp by a process. The algorithm satisfies *k*-lockout avoidance, too. A bounded concurrent time-stamp scheme in linear time and $O(r \cdot m)$ space was proposed in [16], where r is the number of scanners and m is the sum of the number of scanners and the number of labelers used by the scheme. The shared variables used by the linear time scheme are the single-writer/multi-reader shared memory type. It consists of m processes of two kinds, scanners and labelers, by which a labeling protocol and a scanning protocol are executed. The scheme is described in such a form that labeling activity and scanning activity separated into two kinds of processes [16]. To solve the *k*-exclusion problem using the time-stamp scheme in a distributed system, each process should have both functions as a labeler and a scanner. The running time of a process by (n,k)-*EXCL* or by (n,k)-*GTEX* is not as good as the *k*-exclusion algorithm using the linear time bounded concurrent time-stamp scheme. However, in practice the implementation of a bounded concurrent time-stamp scheme is not easy since the algorithm is very complicated. For k such as $n-k = O(1)$ or $n-k = o(n)$, (n,k)-*EXCL* is fast. For example, when $n-k = O(1)$, the running time in the trying region of a process by (n,k)-*EXCL* is linear in n. In such a case, (n,k)-*EXCL* is as fast as the *k*-exclusion algorithm using the linear time bounded time-stamp scheme.

8. Concluding Remarks

We have proposed two lockout avoidance algorithms, (n,k)-*EXCL* and (n,k)-*GTEX*, for the *k*-exclusion problem on the asynchronous multi-writer/reader shared memory model. We proved the correctness of these algorithms and compared their efficiencies. In an execution by (n,k)-*EXCL* or (n,k)-*GTEX*, each process observes the number of winners at a level, but its observation is somewhat uncertain. We showed that such uncertain information is still good enough to guarantee *k*-exclusion and *k*-lockout avoidance. The structure of each of these algorithms is simple. However, we use multi-writer/reader shared variables, $turn(s)$'s in (n,k)-*EXCL* and $turn(x,s)$'s in (n,k)-*GTEX*. Other shared variables used by these algorithms are the single-writer/multi-reader shared memory type. The main disadvantage of these algorithms is the use of multi-writer/reader shared variables. The functions of $turn(s)$'s in (n,k)-*EXCL* and $turn(x,s)$'s in (n,k)-*GTEX* are essentially the same. That is, the last process arrived at a level can be observed by checking a $turn(s)$ or a $turn(x,s)$. This function is called the turn function. At present we do not know whether the turn function can be simply simulated by single-writer/multi-reader shared variables, although it can be simulated by a general method simulating a multi-writer/reader shared variable by single-writer/multi-reader shared variables. The general method of such simulation is considerably complicated [4]. In other words, at present we do not know whether the simulation of the turn function by single-writer/multi-reader shared variables is simpler than the construction of a bounded concurrent time-stamp scheme on the single-writer/multi-reader shared memory model. If we are successful in designing a simple method of realizing the turn function on the single-writer/multi-reader shared memory model, we can design a simple algorithm for giving the temporal relation among processes in the asynchronous distributed system.

Although the total size of the share memory required by by (n,k)-*EXCL* or (n,k)-*GTEX* is reasonable, we are interested in a problem whether some modifications of these algorithms can reduce the shared memory size. We are also interested in a problem whether we can speed up our algorithms by some modifications for some cases, e.g., the case where both k and $n-k$ are $\Theta(n)$, or the case where k or $n-k$ is $O(1)$. The *k*-assignment problem is closely related to the *k*-exclusion problem [4]. We are tempted to design a simple and efficient algorithm using similar techniques for the *k*-assignment problem on the asynchronous shared memory model. These problems would be worthy of further investigation.

REFERENCES

[1] Afek, Y., Attiya, H., Dolev, D., Gafni, E., and Merritt, M., (1993), Atomic snapshots of shared memory, J. ACM, 40: 873–890.

[2] Afek, Y., Dolev, D., Gafni, E., Merritt, M., and Shavit, N., (1994), A bounded first-in, first-enable solution to the *l*-exclusion problem, ACM Trans. Program. Lang. Syst., 16: 939–953.

[3] Attiya, H., and Rachman, O., (1993), Atomic snapshots in $O(n \log n)$ operations, 12th Annu. ACM Symp. Principles of Distributed Computing, Ithaca, New York, 29–40 .

[4] Attiya, H., and Welch, J., (1998), Distributed Computing: Fundamentals, Simulations and Advanced Topics, McGraw-Hill, New York.

[5] Burns, J. E., Jackson, P., Lynch, N. A., Fischer, M. J., and Peterson, G. L., (1982), Data requirements for implementation of *N*-

process mutual exclusion using a single shared variable, J. ACM, 29: 183–205.

[6] Burns J. E., and Lynch, N. A., (1993), Bounds on shared memory for mutual exclusion, Inf. Comput., 107: 171–184.

[7] Cremers, A. B., and Hibbard, T. N., (1978), Mutual exclusion of *N* processors using an $O(N)$-valued message variable, 5th Int. Colloq. Automata, Languages and Programming, Udine, Italy, Lecture Notes Comput. Sci., 62: 165–176.

[8] Dijkstra, E. W., (1965), Solution of a problem in concurrent programming control, Commun. ACM, 8: 569.

[9] Dolev, D., and Shavit, N., (1989), Bounded concurrent time-stamp systems are constructible, 21st Annu. ACM Symp. Theory of Computing, New York, 454–465.

[10] Dwork, C., and Waarts, O., (1992), Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible, 24th Annu. ACM Symp. Theory of Computing, Victoria, Canada, 655–666.

[11] Fischer, M. J., Lynch, N. A., Burns, J. E., and Borodin, A., (1979), Resource allocation with immunity to limited process failure, 20th Annu. Symp. Foundations of Computer Science, San Juan, Puerto Rico, 234–254.

[12] Fischer M. J., Lynch, N. A., Burns, J. E., and Borodin, A., (1989), Distributed FIFO allocation of identical resources using small shared space, ACM Trans. Program. Lang. Syste. 11: 90–114.

[13] Halder, S., and Vitanyi, P., (2002), Bounded concurrent timestamp systems using vector clocks, J. ACM, 49: 101–126.

[14] Igarashi, Y., Kurumazaki, H., and Nishitani, Y., (1999), Some modifications of the tournament algorithm for the mutual exclusion problem, IEICE Trans. Inf. Syst., E82-D: 368–375.

[15] Igarashi, Y., and Nishitani, Y., (1999), Speedup of the *n*-process mutual exclusion algorithm, Parallel Process. Lett. 9: 475–485.

[16] Israeli, A., and Pinhasov, M., (1992), A concurrent time-stamp scheme which is linear in time and space, 6th Int. Workshop on Distributed Algorithms, Haifa, Israel, Lecture Notes Comput. Sci., 647: 95–109.

[17] Kakugawa, H., Fujita, S., Yamashita, M., and Ae, T., (1994), A distributed *k*-mutual exclusion algorithm using *k*-coterie, Inf. Process. Lett., 49: 213–218.

[18] Lamport, L., (1986), The mutual exclusion problem. Part II: Statement and solutions, J. ACM, 33: 327–348.

[19] Lamport, L., (1974), A new solution of Dijkstra's concurrent programming problem, Commun. ACM, 17: 453–455.

[20] Lamport, L., (1987), A fast mutual exclusion algorithm, ACM Trans. Comput. Syst., 5: 1–11.

[21] Lynch, N. A., (1996), Distributed Algorithms, Morgan Kaufmann, San Francisco, California.

[22] Lynch, N. A., and Fischer, M. J., (1981), On describing the behavior and implementation of distributed systems, Theor. Comput. Sci., 13: 17–43.

[23] Lynch, N. A., and Tuttle, M. R., (1987), Hierarchical correctness proofs for distributed algorithms, 6th Annu. ACM Symp. Principle of Distributed Computing, Vancouver, Canada, 137–151.

[24] Obokata, K., Omori, M., and Igarashi, Y., (2000), A lockout avoidance algorithm for the *k*-exclusion problem, IEICE Tech. Rep., Comp 2000-54, 33–40.

[25] Obokata, K., Omori, M., Motegi, K., and Igarashi, Y., (2001), A lockout avoidance algorithm without using time-stamps for the *k*-exclusion problem, 7th Annu. Int. Computing and Combinatorics Conf., Guilin, China, Lecture Notes Comput. Sci., 2108: 571–575.

[26] Peterson, G. L., (1981), Myths about the mutual exclusion problem, Inf. Process. Lett., 12: 115–116.

[27] Peterson, G. L., and Fischer, M. J., (1977), Economical solutions for the critical section problem in a distributed system, Proc. 9th Annu. ACM Symp. Theory of Computing, Boulder, Colorado, 91–97.

[28] Raymond, K., (1989), A distributed algorithm for multiple entries to a critical section, Inf. Process. Lett. 30: 189–193.