Long-lived, Fast, Waitfree Renaming with Optimal Name Space and High Throughput (Extended Abstract)

Wayne Eberly,* Lisa Higham,** and Jolanta Warpechowska-Gruca***

Department of Computer Science, The University of Calgary, Canada
{eberly | higham | jolanta}@cpsc.ucalgary.ca

Abstract. The (n, k, l)-renaming problem requires that names from the set $\{1, \ldots, l\}$ are assigned to processes from a set of size n, provided that no more than $k \leq l$ processes are simultaneously either holding or trying to acquire a name. A solution to this problem supplies a renaming object supporting both ACQUIRE and RELEASE operations so that no two processes ever simultaneously hold the same name. The protocol is *waitfree* if each participant successfully completes either operation in a bounded number of its own steps regardless of the speed of other processes; it is *long-lived* if it there is no bound on the number of operations that can be applied to the object; it is *fast* if the number of steps taken by any process before it completes an operation is independent of n; and it is *name-space-optimal* if $l = k$.

This paper presents the first renaming algorithm for atomic read/write registers that is waitfree, long-lived, fast, and name-space-optimal. Since optimal name space is impossible for deterministic renaming algorithms, our algorithm is randomized. The maximum number (over schedulers and processes) of the expected number (over coin flips) of accesses to read/write registers required to complete either an ACQUIRE or RELEASE operation is $\Theta(k^2)$.

We also define a notion of amortized expected complexity that measures the throughput of a system. The amortized expected step complexity of the new renaming algorithm is $\Theta(k \log k)$, which is a substantial improvement, in this amortized sense, over any preceding long-lived renaming algorithm for read/write registers (whether name-space-optimal or not). The notion of amortized complexity of waitfree protocols may be of independent interest since it seems to suggest that waitfreedom may not be as impractical as is sometimes suspected.

* Research was supported in part by Natural Sciences and Engineering Research Council of Canada grant OGP0089756.

** Research was supported in part by Natural Sciences and Engineering Research Council of Canada grant OGP0041900.

*** Research was supported in part by Natural Sciences and Engineering Research Council of Canada grant dOGP0041900 and DeVry Institute of Technology, Calgary, Canada.

1 Introduction

In a distributed system, the maximum allowable contention (henceforth, k) for a particular system resource is typically much smaller than the total number of processes (henceforth, n) in the system. However, several protocols for resource management have complexity that depends on the size of the original name space of all processes that use the resource. Thus we are motivated to temporarily rename processes, so that they have distinct names from a smaller name space before they access a resource. This is a primary application of renaming protocols as first introduced by Attiya et. al. [4]. The (n, k, l)*-renaming problem* requires that names from $\{1, \ldots, l\}$ are assigned to processes from a set of size n, provided that no more than $k \leq l$ processes are simultaneously either holding or trying to acquire a name. A solution to this problem supplies a renaming object that supports the operations ACQUIRE and RELEASE and ensures that no two processes ever simultaneously hold the same name.

Several desirable properties have been proposed for renaming protocols. In a multiprocessor system, the likelihood of some machine malfunctioning so that it becomes inactive or very slow increases with the size of the system to the point where, with systems of even moderate size, this likelihood is significant. When such a local slow-down or failure occurs on a process that holds a lock or is otherwise involved in a synchronization event, the system performance will be seriously degraded or come to a standstill. This phenomenon has motivated the development of *waitfree* protocols [10], which guarantee that each process successfully completes any operation in a bounded number of its own steps, regardless of the rate of progress (or stop failures) of other processes. All algorithms mentioned in this paper are waitfree.

Anderson and Moir [3] pointed out that, given the intended use, processes should be able to repeatedly ACQUIRE and RELEASE names, and defined a renaming object to be *long-lived* if there is no bound on the number of operations applied to the object. Long-lived renaming protocols have been designed by Moir and Anderson [2], Buhrman et. al. [7], and Moir and Garay [13].

Moir and Anderson also observed that if a (n, k, l)-renaming protocol has complexity that is dependent on n, then it has the same shortcoming that renaming is intended to circumvent, that is, it has complexity determined by the original size of the name space. They defined a renaming protocol to be *fast* if any process, applying any operation, completes the operation in a number of its own steps that is independent of n. Recently, Attiya and Fouren [5] extended the notion of a fast renaming protocol to one that is *adaptive*, where the worst case step complexity depends only on the number $\hat{k}$ of processes that are active, rather than on the upper bound k for this quantity. They presented an $(n, k, 6k - 1)$ adaptive renaming algorithm with worst-case step complexity in $O(\hat{k} \log \hat{k})$. Unfortunately, their protocol is not long-lived, and it is not currently known whether an adaptive long-lived protocol for renaming exists.

Some applications might benefit from having one resource for each contender, with each resource corresponding to a name. Thus, another goal is that the size of the new name space is exactly the contention bound, and many researchers have focused on reducing the size of the new name space as a function of k. Several renaming algorithms with $l = 2k-1$ have been proposed, including those of Borowski and Gafni [6], Anderson and Moir [2], and Moir and Garay [13]. However, Herlihy and Shavit [11] and, independently, Burns and Peterson [8] proved that no deterministic waitfree solution using only atomic registers can have a name space size below $2k-1$. It remained to overcome this lower bound by using either stronger objects or randomization. Say that an (n, k, l)-renaming protocol is *name-space-optimal* if $l = k$. Anderson and Moir [2] used stronger objects called *set-first-zero* objects to achieve name-space-optimality. Since a version of these objects can be implemented with test-and-set objects, which in turn have a randomized implementation using shared variables [1], their algorithm can be converted into a randomized long-lived (n, k, k)-renaming algorithm for atomic registers, with worst-case expected step complexity $O(k \log n)$. Further massaging, using ideas similar to those in this paper, can be used to make this solution fast, with worst-case expected step complexity $\Theta(k^2)$. However, after these changes, the amortized expected step complexity, as discussed shortly, is also $\Theta(k^2)$. Panconesi et.al. [14] use randomization to solve the related *naming* problem for anonymous systems. That problem differs from renaming because there are no process identifiers to exploit, and no emphasis on the attributes desired of renaming. If their solution is used for name-space-optimal renaming, it is neither fast nor long-lived. A summary of the contribution of the most recent advances in renaming appears in Table 1.

Table 1. Waitfree renaming algorithms for the shared memory atomic registers model.

Deterministic Renaming

Algorithm	**Name Space**	**Worst Case Complexity**	**Amortized Complexity**	**Fast**	**Long Lived**
Borowski, Gafni [6]	$2k-1$	$O(nk^2)$	N/A	No	No
Anderson, Moir [2]	$\frac{k(k+1)}{2}$	$O(k)$	N/A	Yes	No
Anderson, Moir [2]	$2k-1$	$O(k^4)$	N/A	Yes	No
Attiya, Fouren [5]	$6k-1$	$O(\hat{k} \log \hat{k})$	N/A	Yes	No
Anderson, Moir [2]	$\frac{k(k+1)}{2}$	$O(nk)$	$O(nk)$	No	Yes
Moir, Garay [13]	$\frac{k(k+1)}{2}$	$O(k^2)$	$O(k^2)$	Yes	Yes
Moir, Garay [13]	$2k-1$	$O(k^4)$	$O(k^4)$	Yes	Yes

Randomized Renaming

Algorithm	**Name Space**	**Expected Worst Case Complexity**	**Expected Amortized Complexity**	**Fast**	**Long Lived**
Panconesi et. al. [14]	$(1+\epsilon)k$, $\epsilon > 0$	$O(n \log^2 k)$	N/A	No	No
This Paper	$\frac{3k^2+k}{2}$	$O(k^2)$	$O(k)$	Yes	Yes
This Paper	k	$O(k^2)$	$O(k \log k)$	Yes	Yes

This paper presents the first waitfree, long-lived, fast, name-space-optimal renaming algorithm for atomic read/write registers. Since name-space-optimal deterministic renaming is impossible, our algorithm is randomized. In the worst case over all schedulers and processes, the expected number (over coin flips) of accesses to read/write registers required to complete either an ACQUIRE or RELEASE operation is $\Theta(k^2)$. We also define a notion of amortized complexity that measures throughput for a long-lived protocol. The amortized expected step complexity of the new renaming algorithm is $\Theta(k \log k)$, which is a substantial improvement, in this amortized sense, over preceding long-lived renaming algorithms for read/write registers (whether name-space-optimal or not). On the way to our final protocol, we also derive a renaming algorithm that achieves $\Theta(k)$ amortized expected step complexity at the expense of a much large name space (with size approximately $(3/2)k^2$).

This notion of amortized complexity of waitfree protocols may be generally useful. As a correctness condition, waitfreedom can be extremely advantageous for the smooth operation of a distributed system. Yet a waitfree solution may be dismissed because, by the traditional worst case cost measure, it is too expensive. It is possible, however, that there is a waitfree solution such that the worst case cost for a process is realized only when the remaining processes are especially efficient. In this case, throughput is guaranteed to be high even though worst case complexity is unappealing. This is the situation with the renaming protocols in this paper. With high probability, the only way that the scheduler can force one process to use the maximum number of steps to complete an ACQUIRE operation is by allowing many other ACQUIRE operations to complete quickly. In the full version on this paper [9] we prove that none of the other existing long-lived renaming algorithms achieve this feature; all have amortized step complexity asymptotically equal to their worst case complexity. For any application that can tolerate the occasional slow (but bounded) operation provided that many other operations complete during the same interval, the traditional notion of complexity may be inappropriate while amortized complexity can provide a reasonable performance measure.

The model of computation, the proposed complexity measures and the renaming problem are defined in the following section of the paper. Our new algorithm is presented in section 3, proved to be correct in section 4, and analyzed in section 5. It has been necessary to abridge some proofs in this version of the paper; a more complete version of the paper, with full proofs, is available from the authors.

2 Definitions

2.1 Model of Computation

We assume an *(asynchronous) shared memory* system, which is a set of n processes with distinct identifiers, together with a globally shared memory composed of randomly accessible (multi-reader/multi-writer) read/write registers. Each register may be either read or written by a process in one atomic step.

A protocol can be regarded as a shared object. Each process can (repeatedly) access the object by applying an operation to it. A data object is *long-lived* if there is no limit on the number of applications of operations to it.

The execution of a shared memory system is determined by a *scheduler*, which determines the operation to be performed by each process (consistently with the semantics of the object) and decides after each step which process will execute the next step. We assume a *strong, adaptive* scheduler that has full knowledge of the past, but cannot predict or bias future random choices.

A shared object is *waitfree* (respectively, *expected waitfree*) if the number (respectively, expected number) of its own steps required by any process to apply any operation to the object is bounded, under any scheduler.

Let $\mathcal{S}$ denote the set of all schedulers. For any application α of an operation under a scheduler S, let $\mathrm{cost}(\alpha, S)$ be the number of steps taken by the process applying α to complete it. If the protocol is deterministic, then $\mathrm{cost}(\alpha, S)$ is an integer; if the protocol is randomized, then $\mathrm{cost}(\alpha, S)$ is a random variable with the probability space defined by the random choices made by processes.

Definition 1. For any deterministic protocol, the *worst case step complexity* is the supremum, over all schedulers $S \in \mathcal{S}$ and applications α, of $\mathrm{cost}(\alpha, S)$.

Definition 2. For any randomized protocol, the *worst case expected step complexity* is the supremum, over all schedulers $S \in \mathcal{S}$ and applications α, of $\mathrm{E}\left[\mathrm{cost}(\alpha, S)\right]$.

Now let $\alpha_1, \alpha_2, \alpha_3, \ldots$ be the sequence of applications of operations under S, ordered by initiation. (This sequence is a random variable, if the protocol is randomized.)

Definition 3. For any deterministic protocol, the *amortized step complexity* is the supremum over all schedulers $S \in \mathcal{S}$ of

$$\lim_{m \to \infty} \sum_{i=1}^{m} \mathrm{cost}(\alpha_i, S)/m.$$

Definition 4. For any randomized protocol, the *amortized expected step complexity* is the supremum over all schedulers $S \in \mathcal{S}$ of

$$\lim_{m \to \infty} \sum_{i=1}^{m} \mathrm{E}\left[\mathrm{cost}(\alpha_i, S)\right]/m.$$

A deterministic (respectively, randomized) protocol is *fast* if the worst case step complexity (respectively, expected step complexity) is independent of n.

2.2 Renaming

If $l \geq k$ then a (n, k, l)-*renaming object*, Rename, is an abstract data object with name space $\{1, \ldots, l\}$ where each name is in state “used” or “unused”. It can be accessed by any of n processes where each access is either an ACQUIRE or a RELEASE. Each participating process continuously cycles through four phases (in order):

1. It executes ACQUIRE which returns a name, say i.
2. It is in its "use" phase.
3. It executes RELEASE(i).
4. It is in its "remainder" phase.

A process is *active* unless it is in the "remainder" phase. The collection of all ACQUIRE and RELEASE operations performed on this object must be linearizable[4] and must satisfy the following *renaming condition*: Provided that each name in $\{1, \ldots, l\}$ is initially marked "unused" and at most k processes are ever active simultaneously,

- ACQUIRE returns exactly one unused name, say $i \in \{1 \ldots, l\}$, and marks i used.
- RELEASE(i) marks i as unused.

We wish to construct a waitfree, long-lived, fast (n, k, k)-renaming object for an asynchronous shared memory system of n processes. That is, algorithms for ACQUIRE and for RELEASE are required such that the above renaming condition for an (n, k, k)-renaming object is satisfied for any set of ACQUIRE and RELEASE operations under any scheduler that ensures that processes behave as described.

3 The New Renaming Algorithms

We first describe three components that are used in our (n, k, k)-renaming algorithm. In these descriptions, and in the sequel, we say that an operation on a data object is applied *in isolation* if the time interval between the beginning and the end of the operation does not overlap with the time interval for any other operation on the same object.

3.1 Components

The k-Large Object: This is the renaming object of Moir and Garay [13]. They prove that k-Large is a deterministic, waitfree, long-lived $(n, k, k(k+1)/2)$-renaming object, and that the worst case step complexity of any operation supported by k-Large is at most $5k^2$. Denote the ACQUIRE and RELEASE operations of this protocol by ACQ and REL respectively.

The k-Small Object: This object, which we extract from the more complicated object k-Large, is the first component of the triangular array of components that make up k-Large. We state its properties here and give details of its construction elsewhere ([9]). Object k-Small is not a renaming object, since it is not guaranteed to return a name; it may return "failure"

[4] See Herlihy and Wing [12].

instead. Specifically, any k-Small object Q is in state either "used" or "unused" and can be accessed by any of n processes where each access is either ACQUIRE-SMALL or RELEASE-SMALL. The collection of all ACQUIRE-SMALL and RELEASE-SMALL operations performed on Q are linearizable and satisfy the following *small object condition*: Provided that initially Q is in state "unused" and at most k processes simultaneously access Q,

- If Q is unused and ACQUIRE-SMALL is applied to Q in isolation, then this ACQUIRE-SMALL operation returns "success" and Q becomes used.
- If Q is unused and ACQUIRE-SMALL is applied to Q but not in isolation, then either all overlapping ACQUIRE-SMALL operations return "failure" and Q remains unused, or at most one ACQUIRE-SMALL operation among the overlapping ones returns "success," all others return "failure," and Q becomes used.
- If Q is used and ACQUIRE-SMALL is applied to Q then "failure" is returned, and Q remains used.
- RELEASE-SMALL, when executed by the last process that successfully applied ACQUIRE-SMALL to Q, sets the state of Q to "unused".

Objects of type k-Small are deterministic, waitfree and long-lived. The worst case step complexity of ACQUIRE-SMALL and RELEASE-SMALL are each at most $5k$.

The m-T&S Object: Object m-T&S is a test-and-set object, which is either in state "set" or "unset" and supports the operations TEST&SET and UNSET, for processes with distinct names from the name space $\{1, \ldots, m\}$. The collection of all TEST&SET and UNSET operations performed on m-T&S is linearizable where the linearized sequence, say L, of TEST&SET and UNSET operations satisfies the following *test-and-set object condition*:

- If m-T&S is unset, then the next TEST&SET in L returns "success" and changes the state of m-T&S to set,
- If m-T&S is set, then TEST&SET returns "failure".
- UNSET, applied to m-T&S by the process that most recently set it, changes its state to unset.

Although it is impossible to construct a deterministic, waitfree, long-lived m-T&S object from read/write registers, a randomized waitfree, long-lived solution due to Afek et. al. [1], does exist, and has worst case expected complexity of both TEST&SET and UNSET in $\Theta(\log m)$.

Objects of types k-Small, k-Large and m-T&S are used to construct our randomized, long-lived, fast renaming object called Rename-1.

3.2 New Renaming Algorithms

The Rename-1 Object: Rename-1 is constructed from k^2 copies of k-Small, called $k\text{-Small}_1, \ldots, k\text{-Small}_{k^2}$, with associated names $1, \ldots, k^2$ respectively, as well as one copy of k-Large and k copies, numbered 1 through k, of m-T&S objects, for $m = k^2 + k(k+1)/2$. Let ACQUIRE-1 and RELEASE-1 denote the acquire and release operations for Rename-1.

To execute ACQUIRE-1, a process first chooses a number, say j, uniformly, independently and randomly from $\{1, \ldots, k^2\}$, and then executes ACQUIRE-SMALL on k-Small_j. If it succeeds then it temporarily acquires the name j. Otherwise, it executes ACQ on the k-Large object, and this object returns a name $r \in \{1, \ldots, k(k+1)/2\}$ with certainty; the process then temporarily acquires the name $r + k^2$. Denote the name in $\{1, \ldots, m\}$ acquired so far by *tmp*. Next, participating with name *tmp*, and incrementing i (initialized to 1), the process applies TEST&SET to m-$\mathsf{T\&S}_i$ until it achieves success. The final name acquired by the process is the final value of i.

A process with name j, and which had temporary name *tmp*, executes the operation RELEASE-1 by first applying UNSET to m-$\mathsf{T\&S}_j$ and then either by applying RELEASE-SMALL to k-Small_{tmp} if $tmp \leq k^2$, or by applying REL$(tmp - k^2)$ to the k-Large object if $tmp > k^2$.

The Rename-2 Object: Another renaming object, which can be useful if it is critical to minimize the amortized step complexity, can be immediately derived from Rename-1. Denote by Rename-2 the renaming object that consists of only the k-Small objects and the k-Large object in Rename-1, and let ACQUIRE-2 and RELEASE-2 be its acquire and release operations. Operation ACQUIRE-2 is just the first part of the operation ACQUIRE-1, which returns the name *tmp* in the set $\{1, \ldots, k^2 + k(k+1)/2\}$ as the final name. Similarly, RELEASE-2 is just the last part of operation RELEASE-1 consisting of applying RELEASE-SMALL on k-Small_{tmp} if $tmp \leq k^2$, or by applying REL$(tmp - k^2)$ on the k-Large object if $tmp > k^2$.

4 Correctness

Claim 5. *Rename-2 is a waitfree long-lived $(n, k, k(3k+1)/2)$-renaming object.*

Proof. It follows from the properties of k-Large and k-Small that as long as at most k processes are active, a process either acquires a unique name i from k-Small_i for some $1 \leq i \leq k^2$, or it acquires a unique name in $\{1, \ldots, k(k+1)/2\}$ from k-Large. Adding k^2 to the name in this second case ensures that all acquired names are unique and belong to $\{1, \ldots, m\}$ where $m = k^2 + k(k+1)/2 = k(3k+1)/2)$. The waitfree and long-lived properties are both inherited from the k-Large and k-Small objects comprising Rename-2.

Claim 6. *Rename-1 is an expected waitfree long-lived (n, k, k)-renaming object.*

Proof. By Claim 5, ACQUIRE-2 returns a unique name *tmp* in $L = \{1, \ldots, m\}$ where $m = k(3k+1)/2)$. By the properties of m-T&S, each m-$\mathsf{T\&S}_i$ can be set by at most one process with a name from L at any time. Also, by Claim 5 and by the properties of m-T&S, each access has bounded expected time. The long-lived property of object Rename-1 is inherited from its components. In the full paper [9] it is proved, by a straightforward induction on k, that any process that sequentially applies TEST&SET to m-$\mathsf{T\&S}_i$ for increasing i (from 1 to k) necessarily receives success from one of these objects, as long as at most k processes are ever active simultaneously.

5 Analysis

We first give the analysis for Rename-2 and then extend it to Rename-1. Consider an execution consisting of a sequence of accesses to Rename-2 by a collection of n processes such that at most k processes are active at any one time. Since there are no more applications of RELEASE-2 than applications of ACQUIRE-2, and each uses no more steps than the corresponding ACQUIRE-2 application, it suffices to bound the cost of the ACQUIRE-2 applications in the sequence in order to obtain a bound on asymptotic performance. Each ACQUIRE-2 application on Rename-2 either returns the name of a randomly chosen k-Small object or, failing this, a name acquired from the object k-Large. We will overcharge for ACQUIRE-2 applications by assuming that if the chosen k-Small object is not accessed in isolation, then ACQUIRE-SMALL will be unsuccessful and $\Theta(k^2)$ steps will be used by the process to acquire its name. We will show, however, that for the overwhelming majority of accesses, the scheduler cannot prevent a process from applying ACQUIRE-2 to a k-Small object in isolation, in which case the access uses only $\Theta(k)$ steps.

At any time, some *selecting* processes are executing ACQUIRE-SMALL, each on its randomly chosen k-Small object. If there are k selecting processes at some time, then the scheduler must run one or more of the existing selecting processes until a process acquires a name and releases it before another ACQUIRE-2 application can begin. If there are fewer than k processes executing ACQUIRE-2 then the scheduler may choose either to introduce a new selecting process, or to have a process that is already participating in ACQUIRE-2 take its next step.

An application of an operation on a data object is involved in a *collision* if it is not performed in isolation. A scheduler can only cause an application of ACQUIRE-2 to use more than $5k$ steps if this application is involved in a collision on some k-Small object.

We define an "amortized cost" (called $\text{acost}(C_{i-1}, l_i)$ below) for applications of ACQUIRE-SMALL such that for every scheduler's strategy and for every sequence of coin-flip outcomes the (actual) total number of steps taken by processes to complete the first m ACQUIRE-SMALL applications is not more than the sum of the amortized costs associated with them, provided that no additional applications are initiated before these are completed. That is, under these circumstances

$$\sum_{i=1}^{m} \text{cost}(\alpha, S) \leq \sum_{i=1}^{m} \text{acost}(C_{i-l}, l_i) \ .$$

More generally, the total actual cost could exceed the total amortized cost for these applications of operations, but never by more than $5k^2(k-1)$, because at most k of the first m applications are involved in collisions with later applications but with none of the first m.

Consider the i-th application of ACQUIRE-SMALL. If it is not involved in a collision on a k-Small object, we will set its amortized cost to be $5k$. On the other hand, if the application of ACQUIRE-SMALL is involved in a collision on a k-Small object, then (as formalized below), all but the first of the applications involved in the collision are assigned an amortized cost that is at least twice the real cost to acquire a name, while the first may receive an amortized cost that is smaller than the real one. Since at least as many applications of operations are (at least doubly) overcharged as are undercharged, the sum of real costs cannot exceed the sum of the amortized costs.

We represent the state of an execution over time as a sequence of sets $(C_0, C_1, \ldots)$, each of size at most $k-1$, where C_i represents the state of the execution just before the start of the $(i+1)$-st application of ACQUIRE-SMALL: For every i, C_i is a subset of $K = \{1, \ldots, k^2\}$ containing names of the k-Small objects chosen by selecting processes. If $(l_1, l_2, \ldots)$ is a sequence of random selections of names made by our algorithm, then C_0 is the empty set, C_i is a subset of $C_{i-1} \cup \{l_i\}$, and the amortized cost for the i-th application of ACQUIRE-SMALL is

$$\mathrm{acost}(C_{i-1}, l_i) = \begin{cases} 5k & \text{if } l_i \notin C_{i-1} \\ 11k^2 & \text{if } l_i \in C_{i-1} \end{cases}$$

Let $K = \{1, \ldots, k^2\}$, and let Ω_m be the probability space of sequences $F = (l_1, \ldots, l_m)$ such that each $l_i \in F$ is chosen uniformly, independently, and randomly from K. Then $\Pr[l_i = t] = 1/k^2$ for all $t \in \{1, \ldots, k^2\}$ and for all $i \in \{1, \ldots, m\}$.

Theorem 7. *The amortized expected step complexity of* ACQUIRE-2 *is* $\Theta(k)$.

Proof. By the above argument, the definition of amortized expected complexity, and linearity of expectation, the amortized expected step complexity of ACQUIRE-2 is at most

$$\begin{aligned} B &= \lim_{m\to\infty} \left(\mathrm{E}\left[\frac{\sum_{i=1}^{m} \mathrm{acost}(C_{i-1}, l_i)}{m} \right] + 5k^2(k-1) \right) \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \mathrm{E}\left[\mathrm{acost}(C_{i-1}, l_i)\right] + \lim_{m\to\infty} 5k^2(k-1) \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} (5k \cdot \Pr\left[l_i \notin C_{i-1}\right] + 11k^2 \cdot \Pr\left[l_i \in C_{i-1}\right]) + 0 \\ &\leq \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} (5k + 11k^2 \cdot \Pr\left[l_i \in C_{i-1}\right]) \\ &= 5k + 11k^2 \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} (\Pr\left[l_i \in C_{i-1}\right]). \end{aligned}$$

For all i, $\Pr\left[l_i \in C_{i-1}\right] = \left|C_{i-1}\right| / k^2 \leq (k-1)/k^2 < 1/k$. This implies that

$$B \leq 5k + 11k^2 \lim_{m\to\infty} \frac{1}{m} \sum_{1}^{m} \Pr\left[l_i \in C_{i-1}\right] < 5k + 11k^2 \lim_{m\to\infty} \frac{1}{m} \cdot m \cdot 1/k \in \Theta(k). \quad \square$$

Rename-1 achieves (n, k, k)-renaming with an amortized cost that is increased by a logarithmic factor. Again, since there are no more RELEASE-1 applications than ACQUIRE-1 applications and each uses no more steps than the corresponding ACQUIRE-1 application, it suffices to bound the cost of the applications of ACQUIRE-1 in the sequence in order to bound asymptotic performance.

Theorem 8. *The amortized expected step complexity of* ACQUIRE-1 *is* $O(k \log k)$.

Proof. By Theorem 7 the amortized expected step complexity of the first stage of ACQUIRE-1 is $O(k)$. In the second stage, processes attempt to set up to k randomized test-and-set objects in sequence after receiving names from a name space of size $O(k^2)$. Each of these test-and-set objects can be implemented using a tournament tree with $O(k^2)$ leaves and with depth $O(\log k)$ — and the worst case expected number of steps required by a process to climb each level of the tree is bounded by a constant (see Afek et. al. [1]). It follows by linearity of expectation that the worst case expected number of steps required to try to set each one of these objects is in $O(\log k)$ and that the worst case expected number of steps required for a process to perform the entire second stage of the algorithm is in $O(k \log k)$. Since the amortized expected step complexity cannot be more than the worst case expected step complexity, the amortized step complexity of the second stage of ACQUIRE-1, and the amortized step complexity for the entire algorithm, is $O(k \log k)$ as well.

The previous two theorems confirm that both our algorithms achieve a substantial improvement over previous renaming algorithms in terms of amortized complexity. The worst-case expected step complexity of both our algorithms, Rename-1 and Rename-2, is easily shown to be dominated by the worst case complexity of the component k-Large, and hence is $\Theta(k^2)$. The space complexity of both our algorithms is $\Theta(k^3)$, but it exceeds by a constant factor (3 in the case of ACQUIRE-2, and 6 in the case of ACQUIRE-1) the space complexity of the deterministic algorithm of Moir and Garay. Reducing the working space is a subject of further research.

References

1. Y. Afek, E. Gafni, J. Tromp, and P. Vitanyi. Wait-free test-and-set. In *Proceedings of the 6th International Workshop, WDAG'92; LNCS No. 647; Springer-Verlag*, 1992.
2. J. Anderson and M. Moir. Fast, long-lived renaming. In *Proceedings of the 8th International Workshop, WDAG'94; LNCS No. 857; Springer-Verlag*, pages 141–155, 1994.
3. M. Anderson, J. amd Moir. Using k-exclusion to implement resilient, scalable shared objects. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 141–150, 1994.
4. H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the Association for Computing Machinery*, 37(3):524–548, 1990.

5. H. Attiya and A. Fouren. Adaptive wait-free algorithms for lattice agreement and renaming. Technical Report 0931, Computer Science Department, Technion - Israel Institute of Technology, April 1998. Extended Abstract to appear in PODC'98.
6. E. Borowsky and E. Gafni. Immediate atomic snapshots and fast renaming. In *Proceedings of the Twelveth Annual ACM Symposium on Principles of Distributed Computing*, pages 41–52, 1993.
7. H. Buhrman, J. Garay, J.-H. Hoepman, and M. Moir. Long-lived remaning made fast. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, 1995.
8. J. Burns and G. Peterson. The ambiguity of choosing. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, 1989.
9. W. Eberly, L. Higham, and J. Warpechowska-Gruca. Long-lived, fast, waitfree renaming with optimal name space and high throughput. (In preparation), 1998.
10. M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 11(1):124–149, 1991.
11. M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 111–120, 1993.
12. M. Herlihy and J. Wing. Linerizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–491, 1990.
13. M. Moir and J. Garay. Fast, long-lived renaming improved and simplified. In *Proceedings of the 10th International Workshop, WDAG'96*, 1996.
14. A. Panconesi, M. Papatriantafilou, P. Tsigas, and P. Vitanyi. Randomized wait-free naming. In *ISAAC'94, LNCS No. 834*, pages 83–91, 1994.