Online Checkpointing with Improved Worst-Case Guarantees

Karl Bringmann, Benjamin Doerr, Adrian Neumann, and Jakub Sliacan

Max Planck Institute for Informatics

Abstract. In the online checkpointing problem, the task is to continuously maintain a set of k checkpoints that allow to rewind an ongoing computation faster than by a full restart. The only operation allowed is to remove an old checkpoint and to store the current state instead. Our aim are checkpoint placement strategies that minimize rewinding cost, i.e., such that at all times T when requested to rewind to some time $t \leq T$ the number of computation steps that need to be redone to get to t from a checkpoint before t is as small as possible. In particular, we want that the closest checkpoint earlier than t is not further away from t than p_k times the ideal distance $T/(k+1)$, where p_k is a small constant. Improving over earlier work showing $1 + 1/k \leq p_k \leq 2$, we show that p_k can be chosen less than 2 uniformly for all k. More precisely, we show the uniform bound $p_k \leq 1.7$ for all k, and present algorithms with asymptotic performance $p_k \leq 1.59 + o(1)$ valid for all k and $p_k \leq \ln(4) + o(1) \leq 1.39 + o(1)$ valid for k being a power of two. For small values of k, we show how to use a linear programming approach to compute good checkpointing algorithms. This gives performances of less than 1.53 for $k \leq 10$.
One the more theoretical side, we show the first lower bound that is asymptotically more than one, namely $p_k \geq 1.30 - o(1)$. We also show that optimal algorithms (yielding the infimum performance) exist for all k.

1 Introduction

Checkpointing means storing selected intermediate states of a long sequence of computations. This allows reverting the system into an arbitrary previous state much faster, since only the computations from the preceding checkpoint have to be redone. Checkpointing is one of the fundamental techniques in computer science. Classic results date back to the seventies [4], more recent topics are checkpointing in distributed [5], sensor network [8], or cloud [11] architectures.

Checkpointing usually involves doing a careful trade-off between the speed-up of reversions to previous states and the costs incurred by setting checkpoints (time, memory). Much of the classic literature (see [6] and the references therein) studies checkpointing with the focus of gaining fault tolerance against immediately detectable faults. Consequently, only reversions to the most recent checkpoint are needed. On the negative side, setting a checkpoint can be highly time consuming, because the whole system state has to be copied to secondary memory. In such

a scenario, the central question is how often to set a checkpoint such that the expected time spent on setting checkpoints and redoing computations from the last checkpoint is minimized (under a stochastic failure model and further, possibly time-dependent [10], assumptions on the cost of setting a checkpoint).

In this work, we will regard a checkpointing problem of a different nature. If not fault-tolerance of the system is the aim of checkpointing, then often the checkpoints can be kept in main memory. Applications of this type arise in data compression [3] and numerics [7, 9]. In such scenarios, the cost of setting a checkpoint is small compared to the cost of the regular computation. Consequently, the memory used by the *stored* checkpoints is the bottleneck.

The first to provide an abstract framework independent of a particular application in mind were Ahlroth, Pottonen and Schumacher [1]. They do not make assumptions on which reversion to previous states will be requested, but simply investigate how checkpoints can be set in an online fashion such that at all times their distribution is balanced over the total computation history.

They assume that the system is able to store up to k checkpoints (plus a free checkpoint at time 0). At any point in time, a previous checkpoint may be discarded and replaced by the current system state as new checkpoint. Costs incurred by such a change are ignored. However, most good checkpointing algorithms do not set checkpoints very often. For all algorithms discussed in the remainder of this paper, each checkpoint is changed only $O(\log T)$ times up to time T.

The max-ratio performance measure. Each set of checkpoints, together with the current state and the state at time 0, partitions the time from the process start to the current time T into $k+1$ disjoint intervals. Clearly, without further problem-specific information, an ideal set of checkpoints would lead to all these intervals having identical length. Of course, this is not possible at all points in time due to the restriction that new checkpoints can only be set on the current time. As performance measure for a checkpointing algorithm, Ahlroth et al. mainly regard the *maximum gap ratio*, that is, the ratio of the longest vs. the shortest interval (ignoring the last interval, which can be arbitrarily small), maximized over all current times T. They show that there is a simple algorithm achieving a performance of two: Start with all checkpoints placed evenly, e.g., at times $1, \ldots, k$. At an even time T, remove one of the checkpoints at an odd time and place it at T. This will lead to all checkpoints being at the even times $2, 4, \ldots, 2k$ when $T = 2k$ is reached. Since this is a position analogous to the initial one, we can continue in the same fashion forever. It is easy to see that at all times, the intervals formed by neighboring checkpoints have at most two different lengths, the larger being twice the smaller in case that not all lengths are equal. This shows the performance of two.

It seems tempting to believe that one can do better, but, in fact, not much improvement is possible for general k as shown by the lower bound of $2^{1-1/\lceil (k+1)/2 \rceil} = 2(1 - o(1))$. For small values of k, namely $k = 2, 3, 4$, and 5, better upper bounds of approximately $1.414, 1.618, 1.755$, and 1.755, respectively, were shown.

The maximum distance performance measure. In this work, we shall regard a different, and, as we find, more natural performance measure. Recall that the actual cost of reverting to a particular state is basically the cost of redoing the computation from the preceding checkpoint to the desired point in time. Adopting a worst-case view on the reversion time, our aim is to keep the length of the longest interval small (at all times). Note that with time progressing, the interval lengths necessarily grow. Hence a fair point of comparison is the length $T/(k+1)$ of a longest interval in the (at time T) optimal partition of the time frame into equal length intervals. For this reason, we say that a checkpointing algorithm (using k checkpoints) has *maximum distance performance* (or simply performance) p if it sets the checkpoints in such a way that at all times T, the longest interval has length at most $pT/(k+1)$. Denote by p_k the infimum performance among all checkpointing algorithms using k checkpoints.

This maximum distance performance measure was suggested in [1], but not further investigated, apart from the remark that an upper bound of β for the gap-ratio performance implies an upper bound of $\beta(1+\frac{1}{k})$ for the maximum distance performance. In the journal version [2] of [1], for all k an upper bound of 2 and a lower bound of $1+\frac{1}{k}$ is shown for p_k. For $k=2,3,4$, and 5, stronger upper bounds of $1.785, 1.789, 1.624$, and 1.565, respectively, were shown.

Our results. In this work, we show that for all k the optimal performance p_k is bounded away from both one and two by a constant. More precisely, we show that $p_k \leq 1.7$ for all k (cf. Section 6). We present algorithms that achieve an upper bound of $1.59 + O(1/k)$ for all k (cf. Theorem 2), and an upper bound of $\ln(4) + o(1) \leq 1.39 + o(1)$ for k being any power of two (cf. Theorem 3). For small values of k, and this might be an interesting case in applications with memory-consuming states, we show superior bounds by suggesting a class of checkpointing algorithms and optimizing their parameters via a combination of exhaustive search and linear programming (cf. Table 1). We complement these constructive results by a lower bound for p_k of $2-\ln(2)-O(1/k) \geq 1.30-O(1/k)$ (cf. Theorem 6). We round our work off with a natural, but seemingly nontrivial result: We show that for each k there is indeed a checkpointing algorithm having performance p_k (cf. Theorem 4). In other words, the infimum in the definition of p_k can be replaced by a minimum.

2 Notation and Preliminaries

In the checkpointing problem with k checkpoints, we consider a long running computation during which we can choose to save the state at the current time T in a checkpoint, or delete a previously placed one. We assume that our storage can hold at most k checkpoints simultaneously, and that there are an implicit checkpoint at time $t=0$ and the current time.

An *algorithm* for checkpoint placement can be uniquely described by two infinite sequences. First, the time points where new checkpoints are placed, i.e., a non-decreasing infinite sequence of reals $t_1 \leq t_2 \leq \ldots$ such that $\lim_{i\to\infty} t_i = \infty$,

and second, a rule that describes which old checkpoints to delete, that is, an injective function $d : [k+1..\infty) \to \mathbb{N}$ satisfying $d_i < i$ for all $i \geq k+1$.

The algorithm A described by (t, d) will start with $t_1, \ldots, t_k$ as initial checkpoints and then for each $i \geq k+1$, at time t_i remove the checkpoint at t_{d_i} and set a new checkpoint at the current time t_i. We call the act of removing a checkpoint and placing a new one a *step* of A. Note that there is little point in setting the first k checkpoints to zero, so to make the following performance measure meaningful, we shall always require that $t_k > 0$.

We call the set of checkpoints that exist at time T *active*. The active checkpoints, together with the two implicit checkpoints at times 0 and T, define a sequence of $k+1$ interval lengths $\mathcal{L}_T = (\ell_0, \ldots, \ell_k)$. The *quality* $q(A, T)$ of an algorithm A at time $T \geq t_k$ is a measure of how long the maximal interval is, normalized to be one if all intervals have the same length. It is calculated as

$$q(A, T) := (k+1)\bar{\ell}_T / T,$$

where $\bar{\ell}_T = ||\mathcal{L}_T||_\infty$ denotes the length of the longest interval. We also use the term *quality* when we refer to the scaled length of a single interval.

The *performance* $\mathrm{Perf}(A)$ is then the supremum over the quality over all times T, i.e.,

$$\mathrm{Perf}(A) := \sup_{T \geq t_k} q(A, T).$$

Hence the performance of an algorithm would be 1, if it kept its checkpoints evenly distributed at all times. Denote the infimum performance of a checkpointing algorithm using k checkpoints by

$$p_k^* := \inf_A \mathrm{Perf}(A),$$

where A runs over all algorithms using k checkpoints. We will see in Sect. 7 that algorithms achieving this performance actually exist.

Note that we allow checkpointing algorithms to set checkpoints at continuous time points. One can convert any such algorithm to an algorithm with integral checkpoints by rounding all checkpointing times t_i. This introduces only a bounded rounding error. As the original and the rounded algorithm have the same performance for $T \to \infty$, we can restrict our attention to the continuous model.

In the definition of the performance, the supremum is never attained at some T with $t_i < T < t_{i+1}$ for any i, as shown by the following lemma.

Lemma 1. *In the definition of the performance it suffices to look at times $T = t_i$ for all $i \geq k$, i.e., we have*

$$\mathrm{Perf}(A) = \sup_{i \geq k} q(A, t_i).$$

Proof. Consider a time T with $t_i < T < t_{i+1}$ for any $i \geq k$. We show that

$$q(A, T) \leq \max\{q(A, t_i), q(A, t_{i+1})\}.$$

Denote the active checkpoints at time T by $p_1, \ldots, p_k$. Note that $p_k = t_i$, since t_i was the last time we set a checkpoint. Consider the interval $[p_k, T]$. Its quality is exactly

$$(k+1)\frac{T - p_k}{T} \leq (k+1)\frac{t_{i+1} - p_k}{t_{i+1}} \leq q(A, t_{i+1}).$$

Any other interval at time T is of the form $[p_{j-1}, p_j]$ for some $1 \leq j \leq k$ (where we set $p_0 := 0$), whose quality is

$$(k+1)\frac{p_j - p_{j-1}}{T} \leq (k+1)\frac{p_j - p_{j-1}}{t_i} \leq q(A, t_i).$$

Together, this proves the claim. □

To bound the performance of an algorithm we need to bound the largest of the $q(A, t_i)$ over all $i \geq k$. For this purpose, it suffices to look at the two newly created intervals at time t_i for each i, as made explicit by the following lemma.

Lemma 2. *Let $i > k$ and let ℓ_1, ℓ_2 be the lengths of the two newly created intervals at time t_i due to the removal and the insertion of a checkpoint. Then*

$$\max\{q(A, t_{i-1}), q(A, t_i)\} = \max\{q(A, t_{i-1}), (k+1)\ell_1/t_i, (k+1)\ell_2/t_i\}.$$

Proof. If ℓ_1 or ℓ_2 is the longest interval at time t_i the claim holds. Any other interval already existed at time t_{i-1} and had a larger quality at this time, as we divide by the current time to compute the quality. Thus, if any other interval is the longest at time t_i, then we have $q(A, t_{i-1}) \geq q(A, t_i)$ and the claim holds again. □

Often, it will be useful to use a different notation for the checkpoint that is removed in step i. Instead of the global index d, one can also use the index $p : [k+1, \infty) \to [1, k]$ among the active checkpoints, i.e.,

$$p_i = d_i - \sum_{j<i} 1_{d_j < d_i}.$$

We call an algorithm $A = (t, p)$ *cyclic*, if the p_i are periodic with period n, i.e., $p_i = p_{i+n}$ for all i, and after n steps A has transformed the intervals to a scaled version of themselves, that is, $\mathcal{L}_{t_{k+jn}} = \gamma^j \mathcal{L}_{t_k}$ for some $\gamma > 1$ and all $j \in \mathbb{N}$. We call γ the *stretch*, or the scaling factor. For a cyclic algorithm A, it suffices to fix the *pattern* of removals $P = (p_{k+1}, \ldots, p_{k+n})$ and the checkpoint positions $t_1, \ldots, t_k, t_{k+1}, \ldots, t_{k+n}$. We can assume without loss of generality that $t_k = 1$ (and hence $t_{k+n} = \gamma$).

Since cyclic algorithms transform the starting position to a scaled copy of itself, it is easy to see that their performance is given by the maximum over the qualities during one period, i.e., for cyclic algorithms A with period n we have

$$\mathrm{Perf}(A) = \max_{k < i \leq k+n} q(A, t_i).$$

This makes this class of algorithms easy to analyze.

3 Introductory Example – A Simple Bound for $k = 3$

For the case of $k = 3$ there is a very simple algorithm, Simple, with a performance of $4/\phi^2 \approx 1.53$, where $\phi = (\sqrt{5}+1)/2$ is the golden ratio. Because the algorithm is so simple, we use it to familiarize ourselves with the notation we introduced in Sect. 2. The algorithm is cyclic with a pattern of length one. We prove the following theorem.

Theorem 1. *For $k = 3$ there is a cyclic algorithm* Simple *with period length one and*

$$\mathrm{Perf}(\textsc{Simple}) = \frac{4}{\phi^2}.$$

Proof. We fix the pattern to be $P = (1)$, that is, algorithm Simple always removes the oldest checkpoint. For this simple pattern it is easy to calculate the performance depending on the scaling factor γ. Since the intervals need to be a scaled copy of themselves after just one step and we can fix $t_3 = 1$, we know immediately that

$$t_1 = \frac{1}{\gamma^2},\ t_2 = \frac{1}{\gamma},\ t_3 = 1,\ t_4 = \gamma,$$

and hence the performance is determined by

$$4 \cdot \max\left\{\frac{t_1 - 0}{t_3}, \frac{t_2 - t_1}{t_3}, \frac{t_3 - t_2}{t_3}\right\} = 4 \cdot \max\left\{\frac{1}{\gamma^2}, \frac{\gamma - 1}{\gamma^2}, \frac{\gamma - 1}{\gamma}\right\}.$$

Since $\gamma > 1$, the second term is always smaller than the third and can be ignored. As $1/\gamma^2$ is decreasing and $(\gamma - 1)/\gamma$ is increasing, the maximum is minimal when they are equal. Simple calculation shows this to be the case at $\gamma = \phi$.

Hence for $k = 3$ the algorithm with pattern (1) and checkpoint positions $t_1 = 1/\phi^2$, $t_2 = 1/\phi$, $t_3 = 1$, and $t_4 = \phi$ has performance $4/\phi^2 \approx 1.53$. □

The experiments in Sect. 6 indicate that for $k = 3$ this is optimal among all algorithms with a period of length at most 6.

4 A Simple Upper Bound for Large k

In this section we present a simple algorithm, Linear, with a performance of roughly 1.59 for large k. This shows that the asymptotic bound of 2 from [1] does not hold in our setting. Moreover, Linear is easily implemented, works for all k (i.e., it is robust), and yields reasonably good bounds.

Like the algorithm Simple of the previous section, algorithm Linear is cyclic. It has a simple linear pattern of length k. The pattern is just $(1, \ldots, k-1)$, that is, at the i-th step of a period Linear deletes the i-th active checkpoint. Overall, during one period Linear removes all checkpoints at times t_i with odd index i, as shown in Fig. 1. The checkpoints are put on a polynomial. For $i \in [1, 2k]$ we set $t_i = (i/k)^\alpha$, where α is a constant. In the analysis we optimize the choice of α and set $\alpha := 1.302$. For this algorithm we show the following theorem.

Theorem 2. *Algorithm* LINEAR *has a performance of at most*

$$\mathrm{Perf}(\textsc{Linear}) \le 1.586 + O(k^{-1}).$$

Experiments show that the performance of algorithm LINEAR is close to the bound of 1.586 even for moderate sizes of k. Comparisons using the optimization method from Sect. 6 indicate that for the pattern $(1, \ldots, k-1)$ of algorithm LINEAR, different checkpoint placements can yield only improvements of about 4.5% for large k. Experimental results are summarized in Fig. 4.

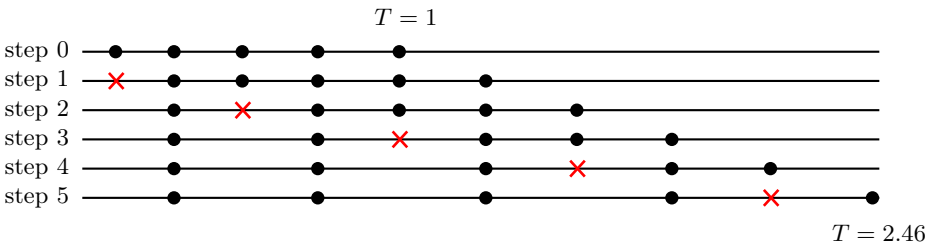

Fig. 1. The moving pattern of algorithm LINEAR from Sect. 4 for $k = 5$. After one period all intervals are scaled by the same factor.

Proof. As algorithm LINEAR is cyclic, we can again compute the performance from the $2k$ checkpoint positions and the pattern,

$$\mathrm{Perf}(\textsc{Linear}) = \max_{k<i\le 2k} (k+1)\bar{\ell}_{t_i}/t_i,$$

where $\bar{\ell}_{t_i}$ is the length of the longest interval at time t_i. By Lemma 2 it suffices to consider newly created intervals at times $t_{k+1}, \ldots, t_{2k}$. Note that at time t_i we create the intervals $[t_{i-1}, t_i]$ (from insertion of a checkpoint at t_i) and $[t_{2(i-k)-2}, t_{2(i-k)}]$ (from deletion of the checkpoint at $t_{2(i-k)-1}$). The quality of the new interval by insertion is, for $k < i \le 2k$,

$$(k+1)\frac{t_i - t_{i-1}}{t_i} = (k+1)\frac{i^\alpha - (i-1)^\alpha}{i^\alpha} \le (k+1)\frac{(k+1)^\alpha - k^\alpha}{k^\alpha}.$$

Using $(x+1)^c - x^c \le c(x+1)^{c-1}$ for any $x \ge 0$ and $c \ge 1$, this simplifies to

$$\le (k+1)\frac{\alpha(k+1)^{\alpha-1}}{k^\alpha} = \alpha(1+1/k)^\alpha = \alpha + O(k^{-1}),$$

for any constant $\alpha \ge 1$.

For the new interval from deleting the checkpoint at $t_{2(i-k)-1}$ we get a quality of

$$\begin{aligned}(k+1)\frac{t_{2(i-k)} - t_{2(i-k)-2}}{t_i} &= (k+1)\frac{(2(i-k))^\alpha - (2(i-k)-2)^\alpha}{i^\alpha} \\ &\le (k+1)2^\alpha \frac{\alpha(i-k)^{\alpha-1}}{i^\alpha},\end{aligned}$$

where we used again $(x+1)^c - x^c \le c(x+1)^c$. An easy computation shows that $(i-k)^{\alpha-1}/i^\alpha$ is maximized at $i = \alpha k$ over $k < i \le 2k$. Hence, we can upper bound this quality by

$$\le \Big(1+\frac{1}{k}\Big)2^\alpha \frac{\alpha(\alpha-1)^{\alpha-1}}{\alpha^\alpha} = 2^\alpha\Big(1-\frac{1}{\alpha}\Big)^{\alpha-1} + O(k^{-1}).$$

We optimize the latter term numerically and obtain for $\alpha = 1.302$ an upper bound of

$$1.586 + O(k^{-1}).$$

Note that this bound is larger than the bound $\alpha + O(k^{-1}) = 1.302 + O(k^{-1})$ from the new intervals from insertion. Hence, overall we get the desired upper bound. □

5 An Improved Upper Bound for Large k

In this section we present algorithm Binary that yields a quality of roughly $\ln(4) \approx 1.39$ for large k. Compared to algorithm Linear from the last section, Binary has a considerably better performance and more involved analysis, but it only works for k being a power of two. Since this is an interesting and realistic condition, we are convinced that this is an interesting result.

Theorem 3. *For $k \ge 8$ being any power of 2, algorithm* Binary *has performance*

$$\mathrm{Perf}(\textsc{Binary}) \le \ln(4) + \frac{0.05}{\lg(k/4)} + O\Big(\frac{1}{k}\Big).$$

Note that the term $O(1/k)$ quickly tends to 0. This is not true for $1/\lg(k/4)$, however, the constant 0.05 is already small. Hence, this quality is close to $\ln(4)$ already for moderate k. Also note that $\ln(4)$ is by less than 0.1 larger than our lower bound from Sect. 8. In fact, for large k our lower bound leaves room for less than a 6% improvement over the upper bound for algorithm Binary. We verified experimentally that algorithm Binary yields very good bounds already for relatively small k. The results are summarized in Fig. 5.

5.1 The Algorithm

The initial checkpoints $t_1, \ldots, t_k$ satisfy the equation

$$t_i = \alpha t_{i/2} \tag{1}$$

for each even $1 \le i \le k$ and some $\alpha = \alpha(k) \ge 2$. Precisely, we set

$$\alpha := 2^{1+\frac{\lg(\sqrt{2}/\ln 4)}{\lg(k/4)}} \approx 2^{1+\frac{0.029}{\lg(k/4)}},$$

where here and in the rest of this paper lg denotes the binary and ln the natural logarithm. However, the usefulness of this expression becomes clear only from the analysis of the algorithm.

During one period we delete all odd checkpoints $t_1, t_3, \ldots, t_{k-1}$ and insert the new checkpoints

$$t_{k+i} := \alpha t_{k/2+i}, \tag{2}$$

for $1 \le i \le k/2$. Then after one period we end up with the checkpoints

$$\begin{aligned} &(t_2, t_4, \ldots, t_{k-2} \ , t_k \ , \quad t_{k+1} \ , t_{k+2} \ , \ldots, t_{k+k/2}) \\ = \alpha \cdot &(t_1, t_2, \ldots, t_{k/2-1}, t_{k/2}, \quad t_{k/2+1}, t_{k/2+2}, \ldots, t_{k/2+k/2}) = \alpha(t_1, t_2, \ldots, t_k), \end{aligned}$$

which proves cyclicity. Note that (1) and (2) allow to compute all t_i from the values $t_{k/2+1}, \ldots, t_k$, however, we still have some freedom to choose the latter values. Without loss of generality we can set $t_k := 1$, then $t_{k/2} = \alpha^{-1}$. In between these two values, we interpolate $\lg t_i$ linearly, i.e., we set for $i \in (k/2, k]$

$$t_i := \alpha^{2i/k-2}, \tag{3}$$

completing the definition of the t_i. Note that this equation also works for $i = k$ and $i = k/2$.

There is one more freedom we have with this algorithm, namely in which order we delete all odd checkpoints during one period, i.e., we need to fix the pattern of removals. In iteration $1 \le i \le k/2$ we insert the checkpoint t_{k+i} and remove the checkpoint $t_{d(i+k)}$, defined as follows. For $m \in \mathbb{N} = \mathbb{N}_{\ge 1}$ let $2^{e(m)}$ be the largest power of 2 that divides m. We define $S\colon \mathbb{N} \to \mathbb{N}, S(m) := m/2^{e(m)}$. Note that $S(m)$ is an odd integer, since it is no longer divisible by any power of 2. Using this definition, we set

$$d(k+i) := S\Big(i + \frac{k}{2}\Big), \tag{4}$$

finishing the definition of algorithm Binary. If we write this down as a pattern, then we have $p_i = 1 + k/(2^{1+e(i)})$ for $1 \le i < k/2$ and $p_{k/2} = 1$. For intuition as to the behavior of this pattern, see the example in Fig. 2. The following lemma implies that the deletion behavior of Binary is indeed well-defined, meaning that during one period we delete all odd checkpoints $t_1, t_3, \ldots, t_{k-1}$ (and no point is deleted twice).

Lemma 3. *The function S induces a bijection between $\{k/2 < i \le k\}$ and $\{1 \le i \le k \mid i \text{ is odd}\}$.*

Proof. Let $A := \{k/2 < i \le k\}$ and $B := \{1 \le i \le k \mid i \text{ is odd}\}$. Since $S(m) \le m$ and $S(m)$ is odd for all $m \in \mathbb{N}$, we have $S(A) \subseteq B$. Moreover, A and B are of the same size. We present an inverse function to finish the proof. Let $x \in B$. Note that there is a unique number $y \in \mathbb{N}$ such that $x2^y \in A$, since A is a range between two consecutive powers of 2 and $x \le k$. Setting $S^{-1}(x) = x2^y$ we have found the inverse. □

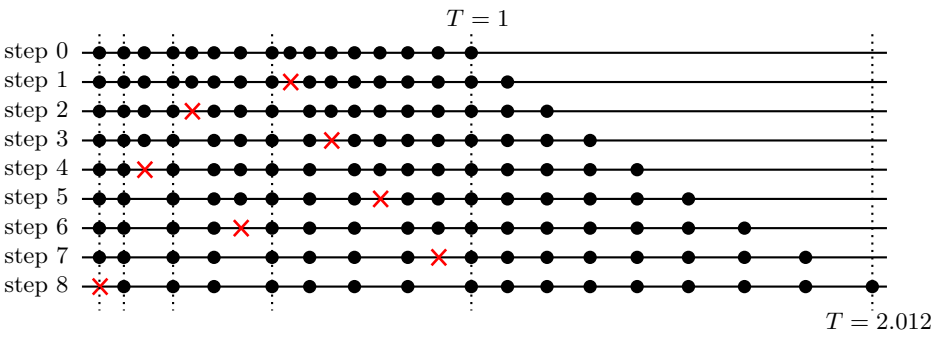

Fig. 2. One run of algorithm BINARY for $k = 16$. Note that, recursively, checkpoints from the right half of the initial setting are removed twice as often (at steps i where $i \bmod 2 = 1$) as checkpoints from the second quarter.

5.2 Quality Analysis

We have to bound the largest quality encountered during one period, i.e.,

$$\mathrm{Perf}(\textsc{Binary}) = \max_{1 \le i \le k/2} q(\textsc{Binary}, t_{i+k}) = (k+1) \max_{1 \le i \le k/2} \overline{\ell}_{t_{i+k}} / t_{i+k}.$$

We first compute the maximum and later multiply with the factor $k+1$. By Lemma 2, we only have to consider intervals newly created by insertion and deletion at any step.

Intervals from Insertion: We first look at the quality of the newly added interval at time t_{i+k}, $1 \le i \le k/2$. Its length is $t_{i+k} - t_{i+k-1}$, so its quality (without the factor $k+1$) is

$$\begin{aligned} \frac{t_{i+k} - t_{i+k-1}}{t_{i+k}} &= 1 - \frac{t_{i+k-1}}{t_{i+k}} \\ &\overset{(2)}{=} 1 - \frac{t_{i+k/2-1}}{t_{i+k/2}} \\ &\overset{(3)}{=} 1 - \alpha^{-2/k}. \end{aligned}$$

Using $e^x \ge 1 + x$ for $x \in \mathbb{R}$ yields a bound on the quality of

$$\frac{t_{i+k} - t_{i+k-1}}{t_{i+k}} \le \ln(\alpha)\frac{2}{k} = \ln(\alpha^2)/k.$$

Deleting t_1: We show similar bounds for the intervals we get from deleting an old checkpoint. We first analyze the deletion of t_1—this case is different from the general one, since t_1 has no predecessor. Note that t_1 is deleted at time $t_{3k/2}$. The deletion of t_1 creates the interval $[0, t_2]$. This interval has quality

$$\frac{t_2}{t_{3k/2}} \overset{(2),(1)}{=} \frac{\alpha t_1}{\alpha t_k} \overset{(1)}{=} \alpha^{-\lg k} \le 1/k,$$

since we choose $\alpha \geq 2$. Hence, this quality is dominated by the one we get from newly inserted intervals.

Other Intervals from Deletion: It remains to analyze the quality of the intervals we get from deletion in the general case, i.e., at some time t_{i+k}, $1 \leq i < k/2$. At this time we delete checkpoint $d(i+k)$, so we create the interval $[t_{d(i+k)-1}, t_{d(i+k)+1}]$ of quality

$$q_i := \frac{t_{d(i+k)+1} - t_{d(i+k)-1}}{t_{i+k}} \overset{(2),(4)}{=} \frac{t_{S(i+k/2)+1} - t_{S(i+k/2)-1}}{\alpha t_{i+k/2}}.$$

Let $h := e(i+k/2)$, so that 2^h is the largest power of 2 dividing $i+k/2$, and $2^h\, S(i+k/2) = i+k/2$. Then $t_{S(i+k/2)+1} = \alpha^{-h} t_{i+k/2+2^h}$ by (1), and a similar statement holds for $t_{S(i+k/2)-1}$, yielding

$$q_i = \alpha^{-1-h} \frac{t_{i+k/2+2^h} - t_{i+k/2-2^h}}{t_{i+k/2}}.$$

Using (3) we get $t_{i+k/2} = \alpha^{2i/k-1}$. Comparing this with the respective terms for $t_{i+k/2+2^h}$ and $t_{i+k/2-2^h}$ yields

$$\begin{aligned} q_i &= \alpha^{-1-h} \left(\alpha^{2^{h+1}/k} - \alpha^{-2^{h+1}/k} \right) \\ &= \alpha^{-1-h} \cdot 2 \sinh\left(\ln\left(\alpha^2\right) 2^h/k \right). \end{aligned}$$

By elementary means one can show that the function $f(x) = x^{-A} \sinh(Bx)$, $A \geq 1, B > 0$, is convex on $\mathbb{R}_{\geq 0}$. Since convex functions have their maxima at the boundaries of their domain, and since by above equation q_i can be expressed using $f(2^h)$ (for $A = \lg \alpha$ and $B = \ln(\alpha^2)/k$), we see that q_i is maximal at (one of) the boundaries of h. Recall that we treated $i = k/2$ separately, and observe that the largest power of 2 dividing $i + k/2$, $1 \leq i < k/2$ is at most $k/4$. Hence, we have $0 \leq 2^h \leq k/4$ and

$$q_i \leq \max\left\{ 2\alpha^{-1} \sinh(\ln(\alpha^2)/k), 2\alpha^{-1}(k/4)^{-\lg \alpha} \sinh(\ln(\alpha)/2) \right\}.$$

We simplify using $\alpha \geq 2$ and $\sinh(x) = x + O(x^2)$ to get

$$q_i \leq \max\left\{ \ln(\alpha^2)/k + O(1/k^2), (k/4)^{-\lg \alpha} \sinh(\ln(\alpha)/2) \right\}. \tag{5}$$

The first term is already of the desired form. For the second one, note that setting $\alpha = 2$ we would get a quality of $4 \sinh(\ln(2)/2)/k = \sqrt{2}/k$. We get a better bound by choosing

$$\alpha := 2^{1 + \frac{c}{\lg(k/4)}},$$

with $c := \lg(\sqrt{2}/\ln(4)) \approx 0.029$. Then the second bound on q_i from above becomes

$$(k/4)^{-\lg \alpha} \sinh(\ln(\alpha)/2) = \frac{4}{k} 2^{-c} \sinh\left(\frac{\ln(2)}{2} \left(1 + \frac{c}{\lg(k/4)} \right) \right).$$

The particular choice of c allows to bound the derivative of $\sinh((1+x)\ln(2)/2)$ for $x \in [0,c]$ from above by

$$\frac{\ln(2)}{2}\cosh((1+c)\ln(2)/2) < 0.39.$$

Hence, we can upper bound

$$\sinh\left(\frac{\ln(2)}{2}\left(1+\frac{c}{\lg(k/4)}\right)\right) \leq \sinh(\ln(2)/2) + \frac{0.39c}{\lg(k/4)}.$$

Thus, in total the second bound on q_i from inequality (5) becomes

$$(k/4)^{-\lg\alpha}\sinh(\ln(\alpha)/2) \leq \frac{4}{k}2^{-c}\sinh(\ln(2)/2) + \frac{4\cdot 2^{-c}\cdot 0.39c}{k\lg(k/4)}.$$

Since $c = \lg(\sqrt{2}/\ln(4)) = \lg(4\sinh(\ln(2)/2)/\ln(4))$, this becomes

$$\leq \ln(4)/k + 0.044/(k\lg(k/4)).$$

Overall Quality: In total, we can bound the performance $p := \mathrm{Perf}(\textsc{Binary})$ of our algorithm (now including the factor of $k+1$) by

$$p \leq (k+1)\max\left\{\ln(\alpha^2)/k + O(1/k^2), \ln(4)/k + 0.044/(k\lg(k/4))\right\}.$$

Using $(k+1)/k = 1 + O(1/k)$ and

$$\ln(\alpha^2) = \ln(4)\left(1+\frac{c}{\lg(k/4)}\right) \leq \ln(4) + \frac{0.040}{\lg(k/4)},$$

this bound can be simplified to

$$p \leq \max\{\ln(4) + 0.040/\lg(k/4) + O(1/k), \ln(4) + 0.044/\lg(k/4) + O(1/k)\},$$

which proves Theorem 3.

6 Upper Bounds via Combinatorial Optimization

In this section we show how to find upper bounds on the optimal performance p_k^* for fixed k. We do so by constructing cyclic algorithms using exhaustive enumeration of all short patterns in the case of very small k or randomized local search on the patterns for larger k, combined with linear programming to optimize the checkpoint positions. This yields good algorithms as summarized in Table 1. In the following we describe our algorithmic approach.

Finding Checkpoint Positions: First we describe how to find a nearly optimal cyclic algorithm given a pattern P and stretch γ, i.e., how to optimize for the checkpoint positions.

Lemma 4. *For a fixed pattern P of length n and scaling factor γ, let $p^* = \inf \mathrm{Perf}(A)$ be the optimal performance among algorithms A using P and γ. Then finding an algorithm with performance at most $p^* + \epsilon$ reduces to solving $O(\log \epsilon^{-1})$ linear feasibility problems.*

Proof. For a fixed pattern and scaling factor, we can tune the performance of the algorithm by cleverly choosing the time points when to remove an old checkpoint and place a new one. By solving a linear feasibility problem we can check whether a cyclic algorithm with stretch γ and pattern P exists that guarantees a performance of at most λ. We can then optimize over λ to find an approximately optimal algorithm.

We construct a linear program with the $k+n$ time points $(t_1, \ldots, t_{k+n})$ as variables (where we can set $t_k = 1$ without loss of generality). It uses three kinds of constraints. The first kind is of the form

$$t_i \leq t_{i+1},$$

for all $i \in [1, k+n)$. These constraints are satisfied if the checkpoint positions have the correct ordering, i.e. checkpoints with larger index are placed at later times.

The second kind of constraints enforces the scaling factor. Since the pattern is fixed, we can readily compute at all steps which checkpoints are active. For $i \in [1, k]$ and $j \in [0, n]$, let τ_i^j be the variable of the i-th active checkpoint in step j and let τ_0^j be 0 for all j. It is easy to see that the algorithm has a stretch of γ if the i-th active checkpoint in the last step is larger by a factor of γ than in the first step. We encode this as constraints of the form

$$\tau_i^n = \gamma \tau_i^0.$$

Lastly we encode an upper bound of λ for the performance. Since the performance of a cyclic algorithm is given by

$$\max_{k < i \leq k+n} (k+1) \bar{\ell}_{t_i} / t_i,$$

and each $\bar{\ell}_{t_i}$ can be expressed by a maximum over k terms, we can encode a performance guarantee of λ with nk constraints of the form

$$\tau_{i+1}^j - \tau_i^j \leq \lambda \tau_k^j / (k+1),$$

for all $i \in [0, k)$ and $j \in [0, n]$.

A feasible solution of these constraints fixes the checkpoint positions and hence, together with the pattern P, provides an algorithm with performance at most λ. Using a simple binary search over $\lambda \in [1, 2]$ we can find an approximately optimal algorithm for this value of γ and the pattern P. □

Finding Stretch Factors: Next we show how to find stretch factors γ for which algorithms with good performance exist. We first show an upper bound for γ.

Lemma 5. *A cyclic algorithm with k checkpoints, performance $\lambda < k$, and a period length of n can have stretch at most*

$$\gamma \leq \left(\frac{1}{1-\lambda/(k+1)}\right)^n.$$

Proof. Consider any checkpointing algorithm $A = (t, d)$ with k checkpoints and performance λ. At any time t_i, $i \geq k$, the largest interval has length $\bar{\ell}_{t_i} \geq t_i - t_{i-1}$, as there is no checkpoint in the time interval $[t_{i-1}, t_i]$. Hence, we have

$$(k+1)\frac{t_i - t_{i-1}}{t_i} \leq \lambda.$$

Rearranging, this yields

$$t_i \leq \frac{1}{1-\lambda/(k+1)} t_{i-1}.$$

Iterating this n times, we get

$$t_{k+n} \leq \left(\frac{1}{1-\lambda/(k+1)}\right)^n t_k.$$

Hence, for any cyclic algorithm (with performance λ, k checkpoints, and a period length of n) we get the desired bound on the stretch $\gamma = t_{k+n}/t_k$. □

Since algorithms with performance 2 are known [1], we can restrict our attention to $\lambda \leq 2$. Hence, for any given pattern length n, Lemma 5 yields a concrete upper bound on γ, while a trivial lower bound is given by $\gamma > 1$. Now, for any given pattern P we optimize over γ using a linear search with a small step size over the possible values for γ. For each tested γ, we optimize over the checkpoint positions using the linear programming approach described above.

Finding Patterns: For small k and n, we can exhaustively enumerate all k^n removal patterns of period length n. Some patterns can be discarded as they obviously can not lead to a good algorithm or are equivalent to some other pattern: No pattern that never removes the first checkpoint can be cyclic. Furthermore, patterns are equivalent under cyclic shifts, so we can assume without loss of generality that all patterns end with removing the first checkpoint. Lastly, it never makes sense to remove the currently last checkpoint. Hence, for k checkpoints there are at most $(k-1)^{n-1}$ interesting patterns of length n. This finishes the description of our combinatorial optimization approach.

Results: We ran experiments that try patterns up to length k for $k \in [3, 7]$. For $k = 8$ we stopped the search after examining patterns of length 7. For even larger k we used a randomized local search to find good patterns. The upper bounds we found are summarized in Table 1 , and for $k \leq 8$ the removal patterns and time points when to place new checkpoints can be found in Fig. 3. Note that for $k = 3$ this procedure re-discovers the golden ratio algorithm of Sect. 3.

Note that we can combine the results presented in Table 1 with algorithm LINEAR (Theorem 2 and Fig. 4) to read off a global upper bound of $p_k \leq 1.7$ for the optimal performance for *any* k.

k	3	4	5	6	7	8	9	10	15	20	30	50	100
Perf.	1.529	1.541	1.472	1.498	1.499	1.499	1.488	1.492	1.466	1.457	1.466	1.481	1.484

Table 1. Upper bounds for different k. For $k < 8$ all patterns up to length k were tried. For $k = 8$ all patterns up to length 7 were tried. For larger k patterns were found via randomized local search.

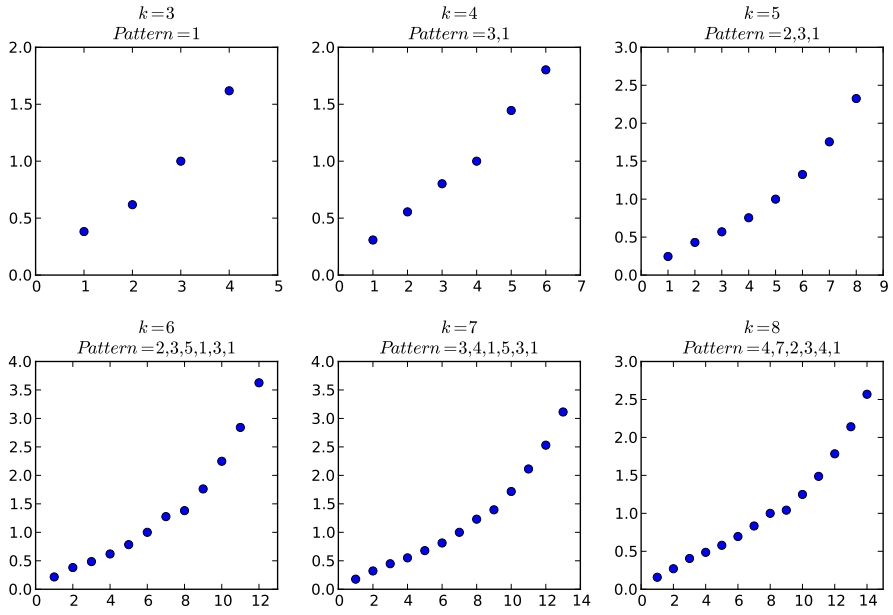

Fig. 3. Time points where the i-th checkpoint is placed to achieve the bounds of Table 1. Time is on the y-Axis, iteration is on the x-Axis.

For a fixed pattern the method is efficient enough to find good checkpoint positions for much larger k. For $k < 1000$ we experimentally compared the algorithm LINEAR of Sect. 4 with algorithms found for its pattern $(1, \ldots, k-1)$. The experiments show that for $k = 1000$ LINEAR is within 4.5% of the optimized bounds. For the algorithm BINARY of Sect. 5, this comparison is even more favorable. For $k = 1024$ the algorithm places its checkpoints so well that the optimization procedure improves performance only by 1.9%. The results are summarized in Fig. 4 and Fig. 5.

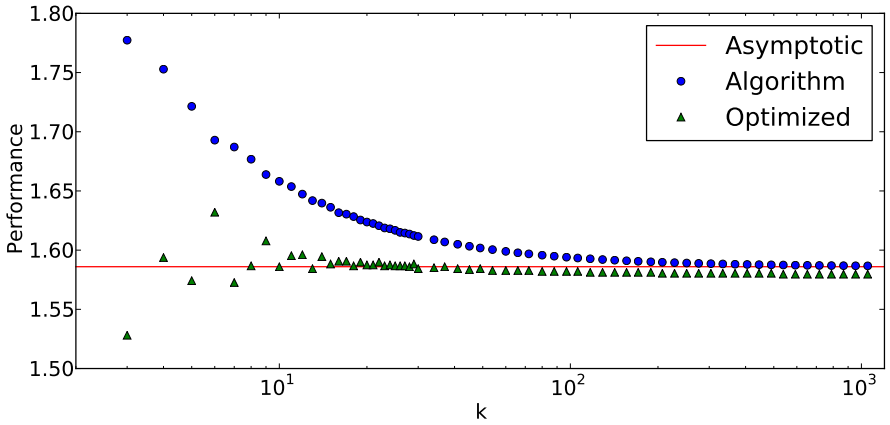

Fig. 4. The performance of algorithm LINEAR from Sect. 4 for different values of k compared with the upper bounds for its pattern found via the combinatorial method from Sect. 6. For large k LINEAR is about 4.5% worse.

Do we find optimal algorithms? One could ask whether the algorithms from Table 1 are optimal, or at least near optimal. There are two steps in above optimization algorithm that prevent this question to be answered positively. First, we are only optimizing over short patterns, and it might be that much larger pattern lengths are necessary for optimal checkpointing algorithms. Second, we do not know how smoothly the optimal performance for fixed pattern P and stretch γ behaves with varying γ, i.e., we do not know whether our linear search for γ yields any approximation on the performance λ. However, in experiments we tried all patterns of length $2k$ for $k \in [3, 4, 5]$ and found no better algorithm than for the shorter patterns of length up to k. Moreover, smaller step sizes in the linear search for γ lead only to small improvements, indicating that the performance is continuous in γ. This suggests that the reported algorithms might be near optimal.

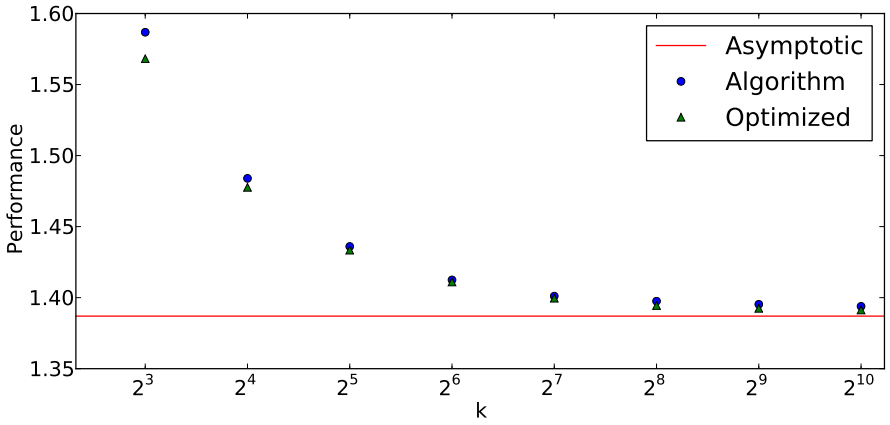

Fig. 5. The performance of the algorithm from Sect. 5 for some values of k, compared with the upper bounds for its pattern found via the combinatorial method from Sect. 6. For $k = 1024$, the optimization procedure finds a checkpoint placement with only 1.9% better performance.

7 Existence of Optimal Algorithms

In this section, we prove that optimal algorithms for the checkpointing problem exist, i.e., that there is an algorithm having performance equal to the infimum performance $p_k^* := \inf_A \mathrm{Perf}(A)$ among all algorithms for k checkpoints.

Theorem 4. *For each k there exists a checkpointing algorithm A for k checkpoints with $\mathrm{Perf}(A) = p_k^*$, i.e., there is an optimal checkpointing algorithm.*

As we will see throughout this section, this a non-trivial statement. From the proof of this statement, we gain additional insight in the behavior of good algorithms. In particular, we show that we can without loss of performance assume that for all i the i-th checkpoint is set by a factor of at least $(1 + 1/k)^{\Theta(i)}$ later than the first checkpoint.

An initial set of checkpoints can be described by a vector $x = (x_1, \ldots, x_k)$, $0 \le x_1 \le \ldots \le x_k$. Since $x = (0, \ldots, 0)$ can never be extended to a checkpointing algorithm of finite performance, we shall always assume $x \neq 0$. Denote by X the set of all initial sets of checkpoints (described by vectors $x \neq 0$ as above), and by X_0 the set of all $x \in X$ with $x_k = 1$.

We say that $A = (t, d)$ is an algorithm for an initial set $x \in X$ of checkpoints if $t_i = x_i$ for all $i \in [k]$. We denote by $p(x) := \inf_A p(A)$, where A runs over all algorithms for x, the *performance* of x. An initial set $x \in X$ is called *optimal* if $p(x) = \inf_{x \in X} p(x) = p_k^*$.

Lemma 6. *Optimal initial sets of checkpoints exist.*

Proof. Since the performance of an initial set of checkpoints is invariant under scaling, that is, $p(x) = p(\lambda x)$ for all $x \in X$ and $\lambda > 0$, we have $\inf_{x \in X} p(x) = \inf_{x \in X_0} p(x)$.

It is not hard to see that $p(.)$ is continuous on X_0: Let $x, x' \in X_0$ with $|x - x'|_\infty \leq \varepsilon$ and consider an algorithm $A = (t, d)$ for x. We construct an algorithm $A' = (t', d)$ for x' by setting $t'_i = t_i$ for $i > k$. Then $|\operatorname{Perf}(A) - \operatorname{Perf}(A')| \leq 2\varepsilon$, since any interval's length is changed by at most 2ε. This implies $|p(x) - p(x')| \leq 2\varepsilon$ and, thus, shows continuity of $p(.)$.

Now, since $p(.)$ is continuous on X_0 and X_0 is compact, there exists an $x \in X_0$ such that $p(x) = \inf_{x \in X_0} p(x) = p^*_k$. □

An easy observation is that if some checkpointing algorithms leads to a vector x of checkpoints at some time, then we may continue from there using any other algorithm for x. The performance of this combined algorithm is at most the maximum of the two performances.

Lemma 7. *Let $A = (t, d)$ be a checkpointing algorithm. Let $i > k$. We call $p_{A,i} = \max_{j \in [k..i]} \bar{\ell}_{t_j}(k+1)/t_j$ the* partial performance *of A observed in the time up to t_i. Assume that when running A, at time t_i the checkpoints $x = (x_1, \ldots, x_k = t_i)$ are active. Let $A' = (t', d')$ be an algorithm for x. Then the checkpointing algorithm obtained from running A until time t_i and then continuing with algorithm A' is a checkpointing algorithm that has performance at most $\max\{p_{A,i}, \operatorname{Perf}(A')\}$. If we run this combined algorithm only until some time t'_j, then the partial performance observed till then is $\max\{p_{A,i}, p_{A',j}\}$.*

Proof. Trivial. □

The above lemma implies that in the following, we may instead of looking at an arbitrary time simply assume that the algorithm just started, that is, that the current set of checkpoints is the initial one.

The following lemma shows that we can, without loss of performance, assume that an algorithm for the checkpointing problem does not set checkpoints too close together. While also of independent interest, among others because it shows how to keep additional costs for setting and removing checkpoints low, we shall need this statement in our proof that optimal checkpointing algorithms exist.

Lemma 8. *Let $A = (t, d)$ be an algorithm for the checkpointing problem with $q(A) < k - 1$. Then there is an algorithm $A' = (t', d')$ with the same starting position such that (i) $\operatorname{Perf}(A') \leq \operatorname{Perf}(A)$ and*

$$(ii)\ t'_{k+3} \geq t'_k \left(1 + \frac{\operatorname{Perf}(A)}{k + 1 - \operatorname{Perf}(A)}\right) \geq t'_k \left(1 + \frac{1}{k}\right).$$

Proof. Let $r = \operatorname{Perf}(A)/(k+1-\operatorname{Perf}(A))$ for convenience. By way of contradiction, assume that the lemma is false. Let A be a counter-example such that $i := \min\{i \in \mathbb{N} \mid t_{k+i} \geq 1 + r\}$ is minimal (the minimum is well-defined, since for any algorithm the sequence $(t_i)_i$ tends to infinity). Note that $i \geq 4$, since A is a counter-example.

Assume that there is a $j \in [1..i-1]$ such that t_{k+j} in the further run of A is removed (and replaced by the then current time t_x) earlier than both t_{k+j-1} and t_{k+j+1}. Consider the Algorithm A' that arises from A by the following modifications. Let t_y be the checkpoint that was removed to install the checkpoint t_j. Let A' be the checkpointing algorithm that proceeds as A except that t_y is not replaced by t_{k+j}, but by t_x, and t_{k+j} is never created. The only interval which could cause this algorithm to have a worse performance than A is $[t_{k+j-1}, t_{k+j+1}]$. However, this interval contributes $(k+1)(t_{k+j+1}-t_{k+j-1})/t_{k+j+1} \leq (k+1)r/(1+r) \leq \mathrm{Perf}(A)$ to the performance of A'. Hence, $\mathrm{Perf}(A') \leq \mathrm{Perf}(A)$ and A' has fewer checkpoints in the interval $[1, 1+r]$ contradicting the minimality of A. Thus, there is no $j \in [1..i-1]$ such that t_{k+j} is removed earlier than both $t_{k+j-1}$and t_{k+j+1} (*).

We consider now separately the two cases that t_{k+1} is removed earlier than t_{k+i-2} and vice versa. Note first that $k+1 < k+i-2$ by assumption that $i \geq 4$.

Assume first that t_{k+1} is removed (at some time t_x) earlier than t_{t+i-2}. Then t_k must have been removed even earlier (at some time t_y), otherwise we found a contradiction to (*). Let A' be an algorithm working identically as A, except that at time t_y the checkpoint t_{k+1} is removed (instead of t_k) and at time t_x the checkpoint t_k is removed (instead of t_{k+1}). Since the checkpoint at t_{t+i-2} is still present, the only interval affected by this exchange, namely the one with t_k as left endpoint, has length at most r. Hence as above, this contributes at most $\mathrm{Perf}(A)$ to the performance of A'. The algorithm A' has the property that there is a checkpoint in between t_k and t_{k+i-2} which is removed before these two points. The earliest such checkpoint, call it t_{k+j}, has the property that t_{k+j} is removed earlier than both t_{k+j-1} and t_{k+j+1}, contradicting earlier arguments.

A symmetric argument shows that also t_{k+i-2} being removed before t_{k+1} leads to a contradiction. Consequently, our initial assumption that $i \geq 4$ cannot hold, proving the claim. □

The following is a global variant of Lemma 8. It shows that any reasonable checkpointing algorithm does not store new checkpoints too often.

Theorem 5. *Let $A = (t, d)$ be a checkpointing algorithm with $\mathrm{Perf}(A) < k-1$. Then there is an algorithm $A' = (t', d')$ with the same starting position such that (i) $\mathrm{Perf}(A') \leq \mathrm{Perf}(A)$ and (ii) $t'_{i+3} \geq (1+1/k) \cdot t'_i$ for all $i \geq k$.*

Proof. Let $j \geq k$ be the smallest index with a small jump, $t_{j+3} < (1+1/k)t_j$. Using Lemma 8 (on the remainder of algorithm A starting at time t_j) we can remove this small jump and get an algorithm $A' = (t', d')$ with $\mathrm{Perf}(A') \leq \mathrm{Perf}(A)$ and $t'_{i+3} \geq (1+1/k) \cdot t'_i$ for all $k \leq i \leq j$, i.e., we patched the earliest small jump. Iterating this patching procedure infinitely often yields the desired algorithm. □

Lemma 9. *For any optimal initial set $x = (x_1, \ldots, x_k)$, there is an algorithm $A = (t, d)$ such that (i) $p_{A,k+3} = \max_{j \in [k..k+3]} \ell_{t_j}(k+1)/t_j \leq p^*_k$, (ii) $t_{k+3} \geq t_k(1+1/k)$, and the set of checkpoints active at time t_{k+3} is again optimal.*

Proof. By the definition of optimality, for each $n \in \mathbb{N}$ there is an algorithm $A^{(n)}$ for x that has performance at most $p^*_k + 1/n$. Let $(t^{(n)}_{k+1}, t^{(n)}_{k+2}, t^{(n)}_{k+3})$ denote

the corresponding next three checkpoints. By Lemma 8, we may assume that $t_{k+3}^{(n)} \geq t_k(1+1/k)$ for all $n \in N$.

Note that (using the same arguments as in Lemma 5) any algorithm having performance at most 2.5 satisfies $t_{k+i} \leq 6^i t_k$ for any $k \geq 2$. Hence, $(t_{k+1}^{(n)}, t_{k+2}^{(n)}, t_{k+3}^{(n)})_{n \in \mathbb{N}_{\geq 2}}$ is a sequence in the compact space $[t_k, 6^3 t_k]^3$. This sequence has a convergent subsequence with limit $(t_{k+1}, t_{k+2}, t_{k+3})$. Also, since there are only finitely many values possible for $(d_{k+1}^{(n)}, d_{k+2}^{(n)}, d_{k+3}^{(n)})$, this subsequence can be chosen such that this d-tuple is constant, say $(d_{k+1}, d_{k+2}, d_{k+3})$. For this subsequence, also all $k+1$ intervals existing at the three times of interest converge. Consequently, the performance caused by each of them also converges to a value upper bounded by p_k^*, showing that $q_{A,k+3} \leq p_k^*$.

Similarly, we observe that the set of checkpoints $x^{(n)}$ active at time $t_{k+3}^{(n)}$ when running algorithm $A^{(n)}$ has performance at most $p_k^* + 1/n$. Consequently, the active checkpoints we get from the limit checkpoints $(t_{k+1}, t_{k+2}, t_{k+3})$ and deletions $(d_{k+1}, d_{k+2}, d_{k+3})$ are again optimal.

Finally, since all $t_{k+3}^{(n)} \geq t_k(1+1/k)$, this also holds for t_{k+3}. □

We are now in position to prove the main result of this section, Theorem 4. For this, we repeatedly apply Lemma 9: We start with an optimal set of checkpoints x. Then we run the algorithm delivered by Lemma 9 for three steps. This creates no partial performance larger than p_k^* and we end up with another optimal set of checkpoints. From this, we continue to apply Lemma 9 and execute three steps of the algorithm obtained. By Lemma 7, the partial performance of the combined algorithm is again at most p_k^*. Iterating infinitely, this yields an optimal algorithm, which proves Theorem 4.

8 Lower Bound

In this section, we prove a non-trivial lower bound on the performance of all checkpointing algorithms. For large k we get a lower bound of roughly 1.30, so we have a lower bound that is asymptotically larger than the trivial bound of 1. Moreover, it shows that algorithm Binary from Sect. 5 is nearly optimal, as for large k the presented lower bound is within 6% of the performance of Binary.

Theorem 6. *All checkpointing algorithms with k checkpoints have a performance of at least*

$$2 - \ln 2 - O(k^{-1}) \geq 1.306 - O(k^{-1}).$$

The remainder of this section is devoted to the proof of the above theorem. Let $A = (t, d)$ be an arbitrary checkpointing algorithm and let $p' := \mathrm{Perf}(A)$ be its performance. For convenience, we define $p = kp'/(k+1)$ and bound p. Since $p < p'$ this suffices to show a lower bound for the performance of A. For technical reasons we add a *gratis checkpoint* at time t_k that must not be removed by A. That is, even after the removal of the original checkpoint at t_k, there still is the

gratis checkpoint active at t_k. Clearly, this can only improve the performance. We analyze the performance of A over the first $k/(2p)$ steps, i.e., up to time $t_{k+k/(2p)}$.

We partition the intervals that exist at time $t_{k+k/(2p)}$ into three types:

1. Intervals existing both at time t_k and $t_{k+k/(2p)}$. These intervals are included in $[0, t_k]$.
2. Intervals that are contained in $[0, t_k]$, but did not exist at time t_k. These intervals were created by the removal of some checkpoint in $[0, t_k]$ after time t_k.
3. Intervals contained in $[t_k, t_{k+k/(2p)}]$.

Note that we need the gratis checkpoint at t_k in order for these definitions to make sense, as otherwise there could be an interval overlapping t_k.

Let $\mathcal{L}_i$ denote the set of intervals of type i for $i \in \{1, 2, 3\}$, and set $k_i := |\mathcal{L}_i|$. Let $\mathcal{L}_2 = \{I_1, \ldots, I_{k_2}\}$, where the intervals are ordered by their creation times $\tau_1 \leq \ldots \leq \tau_{k_2}$. We first bound the length of the intervals in $\mathcal{L}_1$ and $\mathcal{L}_2$.

Lemma 10. *The length of any interval in $\mathcal{L}_1$ is at most pt_k/k.*

Proof. As all intervals in $\mathcal{L}_1$ already are present at time t_k and the algorithm has quality p', we have for any $I \in \mathcal{L}_1$

$$(k+1)|I|/t_k \leq p' = (k+1)p/k.$$

The bound follows. □

The number of checkpoints that are deleted in $[0, t_k]$ up to time $t_{k+k/(2p)}$ is at least k_2 (every interval in $\mathcal{L}_2$ contains at least one removed checkpoint), say their total number is $k_2 + m$. Then m counts the number of deleted checkpoints in $[0, t_k]$ that did not create an interval in $\mathcal{L}_2$, but some strict sub-interval of an interval in $\mathcal{L}_2$. We call these m removed checkpoints *free*.

Lemma 11. *The length of any interval $I_i \in \mathcal{L}_2$ is at most*

$$|I_i| \leq \frac{t_k}{k/p - m - i}.$$

Proof. As the algorithm has performance p', we know

$$|I_i| \leq p\tau_i/k. \tag{6}$$

In the following we bound τ_i, the time of creation of I_i. At time τ_i there are at most $m+i$ intervals in $\mathcal{L}_3$, since at most m free checkpoints and i checkpoints from the creation of $I_1, \ldots, I_i$ are available. Comparing with an equidistant spread of $m + i$ checkpoints in $[t_k, \tau_i]$ and the algorithm's performance, the longest interval L in $[t_k, \tau_i]$ (at time τ_i) has length

$$\frac{\tau_i - t_k}{m+i} \leq |L| \leq \frac{p\tau_i}{k}.$$

Rearranging the outer inequality yields a bound on τ_i of

$$\tau_i \leq \frac{kt_k}{k-(m+i)p}.$$

Substituting this into (6) yields the desired result. □

Furthermore, we need a relation between k_1, k, m, and p.

Lemma 12. *We have*

$$k_1 \leq k + m - k/p + 1.$$

Proof. First, note that in each of the $k/(2p)$ steps of the algorithm from time t_k to $t_{k+k/(2p)}$ at most one point in $[0, t_k]$ is deleted, implying

$$k_2 + m \leq k/(2p). \tag{7}$$

Moreover, as the intervals in $\mathcal{L}_1$ and $\mathcal{L}_2$ partition $[0, t_k]$, there are $k_1 + k_2$ intervals left in $[0, t_k]$ at time $t_{k+k/(2p)}$. Note that each but one such interval has its left endpoint among the k active checkpoints from time t_k (the one exception having as left endpoint 0). Hence, there are $k_1 + k_2 - 1$ checkpoints left in $[0, t_k]$. Comparing with the number $k_2 + m$ of deleted checkpoints and their overall number k yields

$$(k_2 + m) + (k_1 + k_2 - 1) = k.$$

Rearranging this and plugging in inequality (7) yields the desired result. □

Now we use our bounds on the length of intervals from $\mathcal{L}_1$ and $\mathcal{L}_2$ to find a bound on p. Note that the intervals in $\mathcal{L}_1$ and $\mathcal{L}_2$ partition $[0, t_k]$, so that

$$t_k = \sum_{I \in \mathcal{L}_1} |I| + \sum_{I' \in \mathcal{L}_2} |I'|.$$

Using Lemmas 10 and 11, we obtain

$$t_k \leq k_1 \frac{pt_k}{k} + \sum_{i=1}^{k_2} \frac{t_k}{k/p - m - i}.$$

Substituting k_1 using Lemma 12 yields

$$\begin{aligned} t_k &\leq [k + m - k/p + 1]\, pt_k/k + \sum_{i=1}^{k/(2p)-m} \frac{t_k}{k/p - m - i} \\ &= t_k \left(p - 1 + m\frac{p}{k} + O(k^{-1}) + \sum_{i=1}^{k/(2p)-m} \frac{1}{k/p - m - i} \right). \end{aligned} \tag{8}$$

Recall that $H_n = \sum_{1 \leq i \leq n} i^{-1}$ is the n-th harmonic number. Rearranging (8) yields

$$p \geq 2 - m\frac{p}{k} - O(k^{-1}) - H_{k/p-m-1} + H_{k/(2p)-1}.$$

Observe that we have $m\frac{p}{k} + H_{k/p-m-1} \leq H_{k/p-1}$, implying

$$\begin{aligned} p &\geq 2 + H_{k/(2p)-1} - H_{k/p-1} - O(k^{-1}) \\ &\geq 2 + H_{k/(2p)} - H_{k/p} - O(k^{-1}), \end{aligned}$$

since we can hide the last summands of $H_{k/(2p)}$ and $H_{k/p}$ by $O(k^{-1})$. In combination with the asymptotic behavior of $H_n = \ln n + \gamma + O(n^{-1})$, where γ is the Euler-Mascheroni constant, we obtain

$$\begin{aligned} p &\geq 2 + \ln(k/(2p)) - \ln(k/p) - O(k^{-1}) \\ &= 2 - \ln(2) - O(k^{-1}). \end{aligned}$$

This finishes the proof of Theorem 6.

References

1. Lauri Ahlroth, Olli Pottonen, and André Schumacher. Approximately uniform online checkpointing. In Bin Fu and Ding-Zhu Du, editors, *Proceedings of the 17th Annual International Conference for Computing and Combinatorics (COCOON 2011)*, volume 6842 of *Lecture Notes in Computer Science*, pages 297–306. Springer, 2011.
2. Lauri Ahlroth, Olli Pottonen, and André Schumacher. Approximately uniform online checkpointing with bounded memory, 2013. Preprint.
3. M. Bern, D. H. Greene, A. Raghunathan, and M. Sudan. Online algorithms for locating checkpoints. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 359–368. ACM, 1990.
4. K. M. Chandy and C. V. Ramamoorthy. Rollback and recovery strategies for computer programs. *IEEE Transactions on Computers*, C-21:546 –556, 1972.
5. E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34:375–408, 2002.
6. Erol Gelenbe. On the optimum checkpoint interval. *J. ACM*, 26:259–270, 1979.
7. Vincent Heuveline and Andrea Walther. Online checkpointing for parallel adjoint computation in pdes: Application to goal-oriented adaptivity and flow control. In Wolfgang E. Nagel, Wolfgang V. Walter, and Wolfgang Lehner, editors, *Euro-Par 2006 Parallel Processing*, volume 4128 of *Lecture Notes in Computer Science*, pages 689–699. Springer Berlin Heidelberg, 2006.
8. Fredrik Österlind, Adam Dunkels, Thiemo Voigt, Nicolas Tsiftes, Joakim Eriksson, and Niclas Finne. Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulations. In Utz Roedig and Cormac J. Sreenan, editors, *Wireless Sensor Networks*, volume 5432 of *Lecture Notes in Computer Science*, pages 343–357. Springer, 2009.
9. Philipp Stumm and Andrea Walther. New algorithms for optimal online checkpointing. *SIAM Journal on Scientific Computing*, 32(2):836–854, 2010.
10. Sam Toueg and Özalp Babaoglu. On the optimum checkpoint selection problem. *SIAM Journal on Computing*, 13:630–649, 1984.
11. Sangho Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the Amazon elastic compute cloud. In *IEEE 3rd International Conference on Cloud Computing (CLOUD 2010)*, pages 236–243, 2010.