Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A study of mutable checkpointing and related algorithms

Astrid Kiehn [a,*,1], Deepanker Aggarwal [b]

[a] Indian Institute of Technology Mandi, India
[b] University of Wisconsin, Madison, United States

ARTICLE INFO

Article history:
Received 20 February 2016
Received in revised form 26 March 2017
Accepted 29 March 2017
Available online xxxx

Keywords:
Snapshot
Checkpointing
Consistency
Formal verification
Distributed computing

ABSTRACT

Mutable Checkpointing algorithms (MC), [7–9], stand for snapshot algorithms that take checkpoints of processes with causally induced dependencies to the initiating process, only. They classify as coordinated snapshot algorithms with communication induced checkpointing. To specify and verify such algorithms formally, we introduce a formal framework in which their operational semantics can easily be expressed. Within this framework correctness of the algorithms follows from an invariant which explains how snapshots are incrementally built up. For MC and the related blocking queue algorithms of [26,27] some adaptations are required as they take a partial snapshot, only. However, both can be proven in the given framework and in this way we equip them with a direct, constructive consistency proof.

1. Introduction

The on-the-fly computation of a snapshot of a distributed computation is a well known technique for system diagnosis and maintenance. Many snapshot algorithms have been suggested and analyzed leading to classifications like coordinated checkpointing, communication induced checkpointing (CIC) and uncoordinated checkpointing. See [18] for an overview.

New attention has been given to snapshot algorithms with the emerging paradigms of mobile computing and high performance computing. They require an economic use of resources in terms of communication overhead, energy, storage, and scalability.

Mutable checkpointing, [7–9], addresses these requirements by taking checkpoints of only those processes to which there is a causally induced dependency to the initiator. Related are the blocking queue algorithms (BLQ), [26,27,17], which provide an alternative solution to the main problem arising from the partial snapshot request. While there are formal models for classical algorithms, like I/O automata, Petri nets, process algebras, see [28], MC and the blocking queue algorithms have been formulated in terms of pseudo code or verbal explanations, only.

In this paper we set up a general framework for proving correctness of coordinated snapshot algorithms with CIC. We start out with the most prominent algorithm in this class, CL/LY, which is a combination of the seminal papers by Chandy/Lamport [10] and Lai/Yang [20]. The snapshotting is initiated with checkpoint requests sent to all processes, but a checkpoint may also be triggered by the receipt of an application message sent after a checkpoint (the CIC feature). MC and

* Corresponding author.
E-mail address: astrid@iitmandi.ac.in (A. Kiehn).
[1] Adjunct Faculty at Indraprastha Institute of Information Technology Delhi, India.

http://dx.doi.org/10.1016/j.scico.2017.03.009
0167-6423/

BLQ are other instances of such algorithms with the additional feature that only those processes take a checkpoint on which the initiating process directly or indirectly depends on.

Proving correctness of such algorithms amounts to showing consistency of the computed snapshot with the underlying distributed computation. We prove consistency by establishing that a global state and its counterpart in which processes that have taken a checkpoint are frozen, progress in the same way when non-frozen processes are concerned. This approach can be seen as setting up and proving an invariant over the operational semantics of the algorithms, and hence it cannot claim novelty. Surprisingly, however, this technique has not been applied to snapshot algorithms earlier. Proofs that can be found in the literature are either retrospective and reorder events of the completed checkpoint computation (e.g. [10,20]) or are based on contradiction (like for MC). The only two exceptions are the recent work [4] coined as proof by refinement, and [17] a variant of the blocking queue algorithm of [26,27]. The advantage of our direct, constructive proof is twofold: it gives insight into the intermediate behavior of the algorithm, and further opens up the possibility of a computer aided verification (similarly as [4]).

The operational semantics of the algorithms have been extracted from the source papers [9] and [26] where we abstracted (what we consider) the essence of the algorithms by ignoring all details and features not related to the checkpoint taking and its propagation. This allowed us to give a concise description of the algorithms and to identify what we call the principle dilemma of partial snapshotting: how to react when a message is to be received that requires a prior checkpoint in case there is a causally induced dependency between the receiver and the snapshot initiator. As the receiver does not need to be aware of an existing dependency it can either take a tentative checkpoint as in MC or buffer the message in a queue until clarity has been obtained. The latter is the approach of BLQ.

The characterizing invariant of a snapshot algorithm should explain how the snapshot gradually builds up on course of the underlying computation. While the latter is fairly easy to see for algorithms which take checkpoints without discarding them later, it is not obvious for MC where discarding tentative checkpoints is a main feature. The solution we suggest is based on considering simultaneously all possibilities of tentative checkpoints being converted to (permanent) checkpoints later or being discarded. An interesting observation coming out from the analysis is that none of these partially built up snapshots needs to be consistent with the underlying computation. In this respect, the blocking queue algorithms are well-behaved as any such state is consistent with the underlying computation. However, for partial snapshot algorithms the finally computed snapshot is the one with all processes not having taken a checkpoint being reset to their initial states. This final snapshot is consistent with the underlying computation for both algorithms, MC and BLQ.

As a side effect of the formal framework, the algorithms may be compared via the transition rules defining the semantics of the algorithms. Fixing a distributed computation over which a snapshot is to be determined one gets comparative insight into the snapshots computed by CL/LY, MC or BLQ. Though there are limits of this approach, it allows one to establish that MC does not compute "earlier" snapshots than blocking queue algorithms as one might expect.

The paper is structured as follows. The next section introduces the system model and basic definitions of distributed computations. Section 3 explains the framework and the proof technique for coordinated checkpointing with CIC on the basis of CL/LY, the combined algorithm of Chandy and Lamport [10] and Lai and Yang [20]. The principle dilemma of partial snapshotting is explained in Section 4, followed by the analysis of MC, Section 5, and of BLQ, Section 6. Section 7 provides a comparison, and Section 8 concludes with a discussion of related work and final considerations.

The results presented are based on previous work [2,17], in which, separately, mutable checkpointing and a blocking queue algorithm were analyzed. While [2] is subsumed here, [17] analyzes a BLQ algorithm with the additional feature of delayed checkpointing (not present in [26,27]). Delayed checkpointing somewhat blurs the orthogonal approaches of MC and BLQ, so we abstracted away from it in favor of a clearer exposition.

2. System model and basic definitions

We assume a message passing system with reliable FIFO channels. A distributed computation is performed by a set of processes $\{P_1, \ldots, P_n\}$ which communicate solely via FIFO channels C_{ij}, $i, j \leq n$, $i \neq j$. Channel C_{ij} leads from P_i to P_j. Channels are assumed not to lose, reorder or duplicate messages. We further assume that all messages will eventually be read (removed from the channel).

Messages and events are given with *MSG* and *Events*, respectively. For the CL/LY system they are listed in the following table.

cp_taken_1	P_i takes a checkpoint
$send_{ij}(.)$	P_i adds a message to channel C_{ij}
$rcv_{ij}(.)$	P_j receives a message from channel C_{ij}
cpr_1	a checkpoint request message of P_1
$\langle msg, fb\rangle$	a message and attached flagbit

For systems without snapshotting, messages would come without attached flagbits.

With MSG^* and $Events^*$ we consider their free monoids. We use the simple dot to separate letters in a word. For the concatenation of words we use $\circ$ to ease readability. We apply common string operations like replacing in a string w all occurrences of letter a by b, denoted $w[a/b]$, erasing all occurrences of a in w, $w[a/\varepsilon]$, or reversing w to $rev(w)$. More

Please cite this article in press as: A. Kiehn, D. Aggarwal, A study of mutable checkpointing and related algorithms, Sci. Comput. Program. (2017), http://dx.doi.org/10.1016/j.scico.2017.03.009