Int. J. Simulation and Process Modelling, Vol. 5, No. 2, 2009

On the construction of committed consistent global states in optimistic simulation

Francesco Quaglia

Dipartimento di Informatica e Sistemistica,
Sapienza Universita' di Roma,
via Ariosto 25, 00185 Roma, Italy
E-mail: quaglia@dis.uniroma1.it
Website: www.dis.uniroma1.it/~quaglia

Abstract: In this paper, we study how to reuse checkpoints taken in an uncorrelated manner during the forward execution phase in an optimistic simulation system in order to construct global consistent snapshots which are also committed, namely the logical time they refer to is lower than the current Global-Virtual-Time (GVT). Specifically, we present a heuristic-based mechanism relying on update operations applied to local committed checkpoints of the logical processes so to eliminate mutual dependencies among the final achieved state values. The mechanism is lightweight since it does not require any form of (distributed) coordination to determine which are the checkpoint update operations to be locally performed for each logical process. At the same time it is likely to reduce the amount of checkpoint update operations required to realign the consistent global state exactly to the current GVT, taken as the reference time for the snapshot. Our proposal can support, in a performance effective manner, termination detection schemes based on global predicates evaluated on a committed and consistent global snapshot, which represent an alternative as relevant as classical termination check only relying on the current GVT value.

Keywords: optimistic simulation systems; consistent global states; global predicates; rollback-recovery systems.

Reference to this paper should be made as follows: Quaglia, F. (2009) 'On the construction of committed consistent global states in optimistic simulation', *Int. J. Simulation and Process Modelling*, Vol. 5, No. 2, pp.172–181.

Biographical notes: Francesco Quaglia received the Laurea Degree (MS level) in Electronic Engineering in 1995 and the PhD Degree in Computer Engineering in 1999 from the University of Rome 'La Sapienza'. From summer 1999 to summer 2000, he held an appointment as a Researcher at the Italian National Research Council (CNR). Since January 2005, he has worked as an Associate Professor at the School of Engineering of the University of Rome 'La Sapienza', where he previously worked as an Assistant Professor from September 2000 to December 2004. His research interests span from theoretical to practical aspects concerning distributed systems and applications, distributed protocols, middleware platforms, parallel discrete event simulation, federated simulation systems, parallel computing applications, fault-tolerant programming and transactional systems. He has served as Program Co-Chair of ACM PADS 2002, as Program Co-Chair of IEEE NCA 2007, and as General Chair of ACM PADS 2008. Since 2004, he has been an Editorial Board Member of the *International Journal of Simulation and Process Modelling (IJSPM)*.

1 Introduction

Optimistic discrete event simulation systems are based on state saving techniques used to support rollback operations when causality violations occur (Fujimoto, 1990; Jefferson, 1985). Typically, the state logs, also known as checkpoints, are used exclusively for synchronisation purposes (i.e., rollback management) since they are discarded when a new value of the Global Virtual Time (GVT) indicates that they refer to the committed portion of the computation. According to this scheme, termination detection typically relies only on the current GVT value, and on whether it indicates that a specific interval of simulation time has been executed.

However, in general simulation contexts, the termination condition might need to be implemented as a global predicate to be evaluated on a Committed and Consistent Global State (CCGS) (Mattern, 1993).[1] Hence, an optimistic simulation platform should include the facilities for identifying CCGSs in order to support such a general termination detection approach. This is exactly the objective of this paper, which proposes a lightweight mechanism for the identification and construction of CCGSs formed by collections of state values (one from each Logical Process (LP)). The mechanism is lightweight in a twofold sense:

- It does not impose any form of coordination among the state saving activities across different LPs. Hence for each LP we maintain complete autonomy for what concerns the scheme used to determine when to take checkpoints during forward computation. This scheme can be selected according to a spectrum of possibilities (see e.g., Fleischmann and Wilsey, 1995; Quaglia, 2001; Ronngren and Ayani, 1994) in order to optimise the tradeoff between state saving and rollback overheads (Preiss et al., 1994).
- It is based on an update policy of committed checkpoints (supported according to an approach similar to classical 'coasting forward') relying on a heuristic method only exploiting local information available at the LP. This information is used to determine when the update phase of a committed checkpoint can end, while ensuring at the same time the absence of mutual dependencies among the LPs' states forming the global snapshot.

Our proposal can provide advantages also in interactive simulation scenarios, where (aggregate) output data, consistently reflecting changes in the global state of the whole simulated system, need to be continuously provided to, e.g., an interactive end-user.

The proposed heuristic-based mechanism for the construction of the CCGS has been integrated within an operating optimistic simulation platform, and we also report some experimental results demonstrating its limited overhead.

The remainder of this paper is structured as follows. In Section 2 we describe the CCGS construction mechanism. Related work is discussed in Section 3. The experimental study is presented in Section 4.

2 Construction of the CCGS

2.1 Basics and motivations

Our approach to the determination and construction of the CCGS relies on the exploitation of the current GVT value. This value is in fact used to initially determine which checkpoints belong to the committed portion of the simulation, and could form the base for the construction of the CCGS. Also, we want to construct the CCGS formed by local states of different LPs, whose simulation time is at least equal to the simulation time of the latest checkpoints preceding GVT. This is because we want to provide a global snapshot starting from the logs referring to the most part of the committed computation.

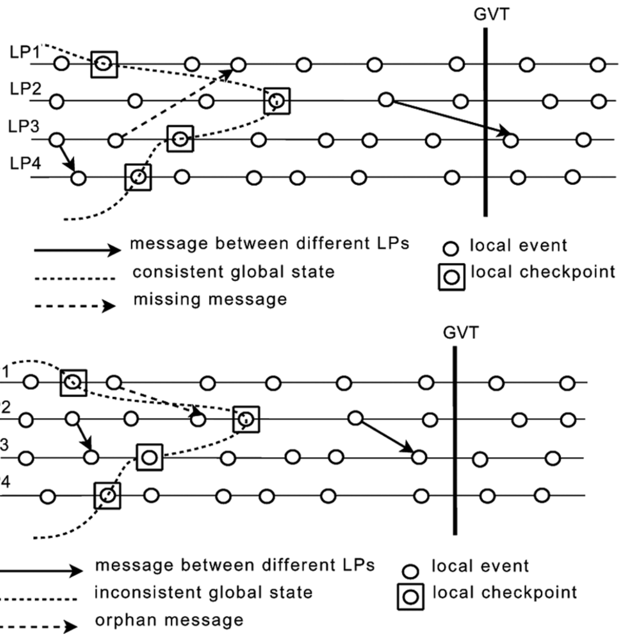
As sketched in Section 1, we consider the typical case in which it does not exist any distributed protocol for taking local checkpoints of the LPs in a coordinated manner. This is done in order to allow the maximal flexibility concerning checkpoint decisions at different LPs in favour of optimising, for each LP, the tradeoff between checkpointing and recovery costs. Hence a simple collection of committed checkpoints, one per LP, does not ensure that the associated global snapshot is consistent.

We base our method on the assumption that each LP can rebuild the value of its own state at any logical time between the latest checkpoint with time prior to GVT (i.e., the latest committed checkpoint) and GVT. As we will show while depicting the operating optimistic simulation platform supporting our CCGS construction mechanism, this can be realised in an effective and application transparent manner via ad-hoc memory management mechanisms allowing no corruption of the current state value while the checkpoint update process takes place. Also, the aforementioned assumption does not contrast with classical optimistic simulation platform design since these platforms always guarantee that, together with the latest checkpoint preceding GVT, all the input events for the same LP with timestamps greater than the logical time of that checkpoint are retained, independently of the schedule of operations performing memory recovery. Hence these events are available for reprocessing actions aimed at updating the latest committed checkpoint up to the target logical time.

As widely known, LPs can create mutual dependencies by scheduling events for each other. Specifically, when the execution of an event at an LP schedules a new event to be executed by a different LP, the former sends a message to the latter. To keep our graphical representation of the dependencies simple in the example pictures, we draw message exchanges in a way that the message send operation exactly coincides with the execution of the former event, while the message receipt operation exactly coincides with the execution of the scheduled event. In other words, we are interested in the dependencies in logical time. Thus intermediate buffering operations (and more in general the message treatment within the operating simulation platform) are not explicitly considered since they do not contribute to dependencies between local states of the LPs in logical time.

Given this premise, Figure 1 (top part) shows a scenario where, once computed the new GVT value, a consistent global state can be obtained by simply collecting each LP latest checkpoint preceding GVT. On the other hand, in Figure 1 (bottom part) we show an example situation where the latest checkpoint of LP_2 preceding GVT records the execution of an event that depends on an event of LP_1, which is instead not recorded as occurred by the latest checkpoint of LP_1 preceding GVT. In the distributed computing community, the message associated with such a dependency is widely known as orphan message (Chandy and Lamport, 1985). Instead, in the reverse situation where a global state records the sending of a message, but does not record its receipt, we have a so called missing message (see again Figure 1). The presence of orphan messages makes a global state not causally consistent, while missing messages are admitted when the reference criterion is limited to the consistency of the global snapshot formed by a collection of local process states, namely more stringent criteria just involving the absence of missing messages, such as those discussed in Hélary et al. (1999), are not of interest.

Figure 1 Some examples



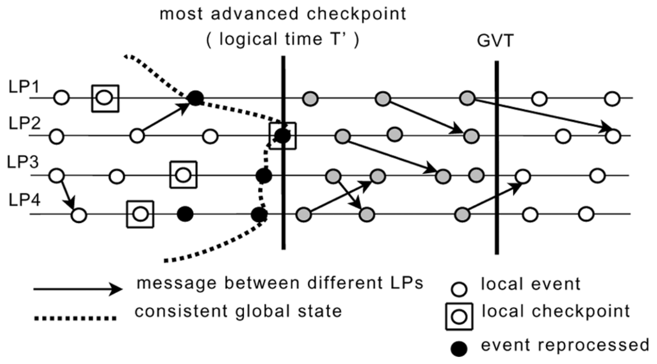
In the case of optimistic simulation systems, it would be sufficient that all LPs rebuild their own states at a same logical time (taken as reference time for the snapshot) in order to assure that the global state is causally consistent. In fact, in this case it is not possible to record the receipt of a message without also recording the corresponding send operation. This is because, when executing whichever event, an LP is allowed to schedule new events either with a future logical time, or at least for the current logical time (this is a classical rule for the consistency of Discrete Event Simulation models). Hence realigning all the LPs' states to a same reference logical time prevents the presence of orphan message dependencies.

Given that the base for the construction of the global snapshot is the preventive identification of the committed portion of the simulation via the calculation of a new GVT value, a straightforward application of the previous scheme would be to arrange that each LP realigns its own state to GVT.[2] In this way, the LPs can immediately rebuild their required local states after they have been notified about the new GVT value. Furthermore, there is even no need for waiting for the completion of any additional (distributed) interaction round among the process instances involved in the computation to determine a committed reference time different from GVT. Moreover we totally exploit the committed computation, by providing the global snapshot exactly at the latest GVT.

However, a drawback of this method is that the number of events each LP has to reprocess when starting from the latest checkpoint preceding GVT is directly proportional to the average checkpoint distance (or checkpoint period), which we denote as χ. In particular, several works have already shown that, if checkpoints are taken in an uncorrelated manner with respect to a specific simulation time value T, then T is expected to fall within a checkpoint interval according to a uniform distribution. This is exactly the case of GVT, since its value is determined a posteriori of both execution and checkpointing activities referring to the committed portion of the simulation. Hence, we have that the average distance (in terms of simulation events) between the latest committed checkpoint and the GVT value is $(\frac{\chi-1}{2})$. Realigning the global state to GVT would hence produce an execution cost that is proportional to $(\frac{\chi-1}{2} \times \#LPs)$.

In order to reduce this cost, a possible solution is to realign the state of each LP to a reference logical time preceding GVT. As an example, consider the set in Figure 2 composed by the checkpoints that the LPs have taken just before GVT. This set is obviously defined only after GVT computation. A good realignment time would be T', namely the maximum logical time of the checkpoints belonging to that set. In this way we could expect, at least in principles, a reduction of the number of events to be reprocessed. Compared to the previous method this one would lead to re-executing only the events represented by black circles, instead of all the events towards GVT. However, such a reduction is likely to occur only in the case the number of LPs is relatively small, since this reduces the likelihood that at least one LP has its latest committed checkpoint very close to GVT. Also, as sketched above, taking T' as the reference time for the snapshot would require an additional (distributed) interaction round, executed just after the calculation of the GVT, to notify T' to all the LPs.

Figure 2 Realignment to a reference time different from GVT



2.2 The heuristic-based mechanism

To address all the issues raised in the previous section concerning the overhead for the collection of the CCGS, we base our solution on a heuristic approach. The basic idea for this approach is that each LP can determine its own realignment time by simply analysing its own data structures, with no need for additional (distributed) interaction rounds with other LPs after GVT calculation. Depending on the current value of these data structures, in the worst case the LP must reprocess all its events till GVT, while in the best case the LP must reprocess no event (hence its latest checkpoint preceding GVT can immediately be used for the construction of the CCGS).

As the base for computing the value of its own realignment time, the LP exploits the fact that the heuristic method will push whichever LP to realign its own state at most to GVT (this is exactly the worst case scenario depicted above). Hence, in order to identify which events must be reprocessed to ensure the absence of orphan message dependencies among the local states eventually forming the global snapshot, each LP must determine the set of its executed events defined by the following conditions:

A The event execution was scheduled in the simulation time interval in between the logical time of the last committed checkpoint and GVT (i.e., the timestamp of the event is within that interval).

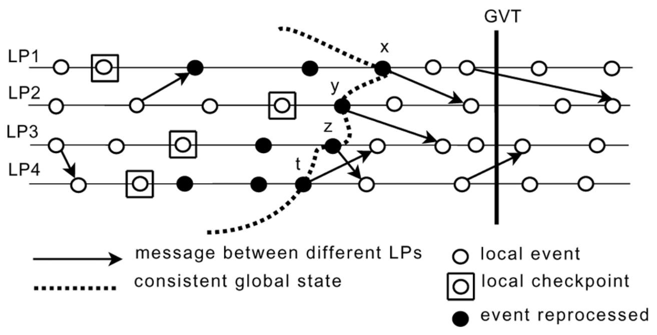
B The event execution has scheduled new events for other LPs at a logical time that precedes GVT.

After having identified the set of all the events, if any, for which both conditions A and B are verified, the LP must determine which event belonging to this set is nearest to the current GVT. The timestamp of that event is exactly the realignment time for the LP. In fact, realigning to that time ensures the absence of orphan dependencies caused by messages sent by this LP with timestamps up to GVT (all the events corresponding to the send operations are recorded as occurred by the realigned LP state). Hence, given that LPs realign their states at most to GVT, the final global state after the realignment of each LP is consistent.

An example of the behaviour of this heuristic method is shown in Figure 3. In this example, the different LPs need to realign their states up to the events x, y, z and t, respectively. This is because, starting from the last taken checkpoints prior to GVT, these are the latest events generating dependencies (via the scheduling of new events for other LPs) with logical time up to GVT. Considering in more details the behaviour of LP_1 allows us to further provide explanations about the tradeoff provided by our heuristic. Specifically, this LP re-executes three events starting from the last taken checkpoint prior to GVT. On the other hand, compared to the case of realignment exactly to GVT, we avoid at LP_1 the reprocessing of other two events. This is achieved with no coordination with other LPs during both checkpointing activities and the realignment phase.

From a methodological point of view, this approach differs from the ones discussed in Section 2.1 since it aims at exploiting (in a lightweight manner) partial information about the structure of the computation locally available at each LP (in terms of real dependencies among the events), for the identification and construction of the CCGS. Instead, the other approaches only exploit a reference simulation time value for realignment operations and elimination of the dependencies. As an example, in case the LPs communicate infrequently and/or the timestamps of new scheduled events are relatively far in the future, our solution exploits these features to reduce the number of events to be reprocessed while realigning the state of each LP (since we can track the absence of dependencies concerning processed events starting from the last committed checkpoint up to GVT).

Figure 3 Heuristic method example



2.3 An implementation

In this section, we describe an implementation of the heuristic-based CCGS construction mechanism within an operating optimistic simulation environment, namely the ROme OpTimistic Simulator (ROOT-Sim). This is an open source, general purpose simulation platform developed using `C` technology, which is based on a simulation kernel layer that ultimately relies on MPI for data exchange across different kernel instances. This software transparently supports all the mechanisms associated with parallelisation (e.g., mapping of the LPs on different kernel instances) and optimistic processing.

In this platform, the interaction between the application software and the simulation kernel mainly occurs via:

- a callback service to be offered by the application level software, namely `ProcessEvent()`, acting as the event handler
- a service offered by the underlying simulation kernel called `ScheduleNewEvent()`.

The `ProcessEvent()` callback has a set of parameters identifying the event to be processed (in the form of an application defined data structure), the global identifier of the scheduled LP (expressed as a numerical code used as an LP indexing information) and the base state address associated with the LP for which the event is occurring. It also has an additional parameter acting as a flag, which indicates whether we are in the presence of the first call to `ProcessEvent()` for a given LP. This flag can be exploited for initialisation purposes, which might entail setup of the main data structure representing the LP state. `ProcessEvent()` can be programmed in `C` technology, with the possibility to use malloc/free compliant memory allocation/deallocation services thanks to the integration of the simulation kernel with a dynamic memory logger and restorer library, named DyMeLoR (Toccaceli and Quaglia, 2008). The `ScheduleNewEvent()` service offered by the platform can be invoked during event processing for injecting new events within the system. This service only needs to receive in input the new event content and a numerical code identifying the destination LP within the whole set of active LPs (whose size is established via an application level defined macro).

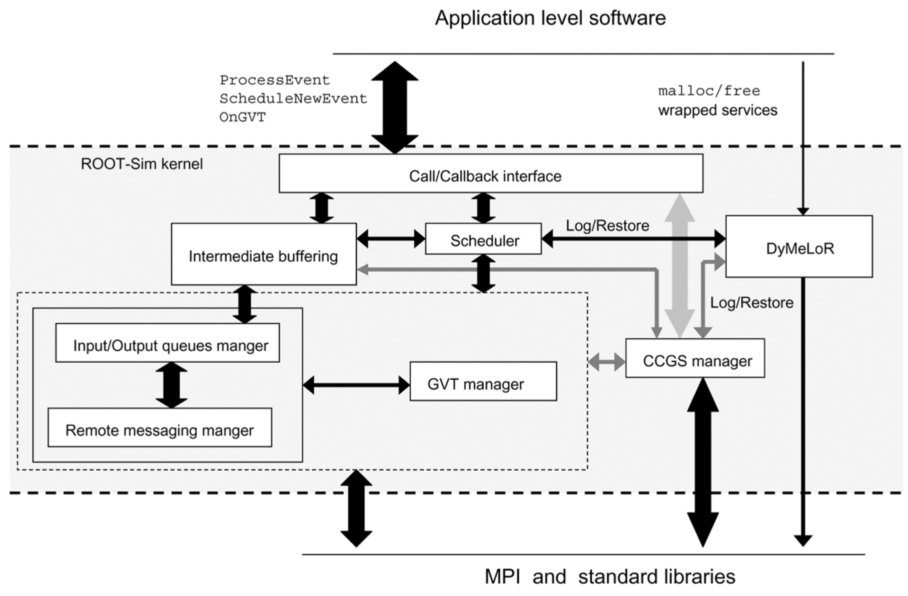
A schematisation of the ROOT-Sim software architecture is provided in Figure 4.

Via DyMeLoR, the simulation kernel layer supports application transparent checkpointing facilities for recording the LP state periodically. The checkpoint operation is mainly implemented as a pack operation of the memory scattered chunks currently belonging to the LP state into a contiguous checkpoint buffer. On the other hand, the state restore operation is implemented via a reverse unpack operation, which puts those chunks back in palace in their correct position within the memory address space. In case `ProcessEvent()` is invoked by the underlying software due to the re-execution of events in a coasting forward phase, the intermediate buffering facilities handle the discarding of the output events in a totally transparent manner to the application programmer. All the other facilities for the management of housekeeping operations (such as CPU scheduling when multiple LPs are hosted by the same instance of the simulation kernel and input/output event queues management) are also handled by the underlying platform in a totally transparent manner.

Integration of the CCGS construction mechanism within this version of the ROOT-Sim platform has been achieved via an approach where the global snapshot is build and then used (e.g., accessed) in a completely distributed manner. In other words, no LP state transfer operation towards a master simulation kernel instance takes place, even after the correct state belonging to the consistent committed snapshot has been reconstructed. This choice has been motivated by the integration with DyMeLoR that, as outlined, provides the opportunity for a flexible programming model entailing the use of ANSI-C services for dynamic memory allocation/release. In such a case, correct maintenance of pointer based linking among the memory chunks belonging to the (reconstructed) state of each LP within the global snapshot would require locating those chunks at predefined memory addresses at the master kernel in case of adoption of centralised collection/access of the CCGS. This would in turn require developing ad-hoc versions of the malloc library, offering the possibility to explicitly define the memory addresses where to reserve buffers at the master kernel, destined to chunks of a given LP state. This would give rise to an evident decrease of software portability, and, likely, to a decrease of efficiency. Also, different LPs might have chunks reserved at the same memory address on different simulation kernel instances, hence giving rise to memory conflicts in case of LP state transfer towards the same (master) kernel instance for supporting centralised access to the CCGS.

The heuristic-based mechanism for the (distributed) identification and construction of the CCGS is implemented in a totally transparent manner to the application software as follows. When the LP state belonging to the global snapshot needs to be reconstructed, the CCGS manager temporarily saves the current LP state via DyMeLoR facilities. Afterwards, it restores the latest committed checkpoint of that LP (again via DyMeLoR), which is then updated according to the heuristic scheme. (The temporary saving of the current state is motivated by the fact that the restore operation of the latest committed checkpoint might entail restoring the content of some dynamically allocated memory chunks still belonging to the current LP state. Hence, the current content of these chunks could get lost in case no temporary copy is performed, to be eventually put back in place when normal event execution gets resumed. On the other hand, with DyMeLoR it is guaranteed that every chunk possibly deallocated during forward computation gets

Figure 4 Schematised ROOT-Sim architecture



really released to the underlying `malloc` library after the deallocation gets committed. Hence that chunk is available for restore and update operations of the latest checkpoint preceding GVT.) The update phase of the restored committed checkpoint is supported by the CCGS manager by simply invoking the `ProcessEvent()` callback passing as input the base state address associated with the restored checkpoint. Also, no output is really produced during the checkpoint update phase since (always transparently to the application level software) the output messages are filtered via the facilities offered by the intermediate buffering subsystem.

The application interface of ROOT-Sim has been also augmented with the callback function `OnGVT()`. This callback receives in input an LP identifier and the base state address of the reconstructed LP state belonging to the CCGS, and can be used (beyond other tasks, such as I/O based on committed state information) to evaluate a predicate indicating whether the computation can end at that LP. The ROOT-Sim programming model requires this callback to return to the underlying CCGS manager a boolean value indicating the result of the evaluation of the predicate. The master instance of the distributed simulation kernel collects these boolean results after GVT has been calculated, each kernel instance has reconstructed the LP states belonging to the CCGS, and has locally invoked the `OnGVT()` callback for each hosted LP. Computation can end when all the collected boolean flags are set to true. The activities of the CCGS manager are triggered as soon as an updated GVT value is available.

Concerning GVT calculation, ROOT-Sim relies on an optimised asynchronous approach based on a message acknowledgment scheme to solve the well-known transient message problem. Within this scheme, each kernel instance keeps track of all messages sent to the other instances. However, to keep the memory consumption limited, this information is retained in an aggregate manner (i.e., via counters). Also, to reduce the communication overhead due to acknowledgment messages, each instance acknowledges received messages periodically, thus sending cumulative acknowledgment messages according to a window-based approach. Finally, to overcome the simultaneous reporting problem (Samadi, 1985), each kernel instance temporarily stops sending acknowledgment messages during the execution of the GVT protocol.

As a final note, in order to manage the heuristic-based mechanism efficiently, we just tweaked the data structures commonly used for event queues by adding the appropriate information. Specifically, for each executed event, we need to identify the minimum logical time of (possibly) scheduled new events, destined to other LPs. The timestamps of events destined to the very same LP are not relevant since the scheduling of these events cannot originate (orphan) messages across different LPs. Hence, during forward execution, the simulation platform must consider the events destined to other LPs and must record the minimum of all their timestamps in an appropriate field (i.e., `min_output_timestamp`) of the data structure which currently buffers the event just executed. This minimum value is determined by analysing the information about new events provided by the intermediate buffering level. In order to assure the consistency of such information, the `min_output_timestamp` field is invalidated when the corresponding event is rolled back.

Let us now consider the actual application of the heuristic-based mechanism. As pointed out before we can determine a consistent global state after the new GVT value is evaluated. By exploiting the previously mentioned information, the simulation kernel must consider for each LP the set of all its executed events that satisfy the below conditions:

- the event timestamp is greater than the logical time of the last checkpoint taken before GVT
- the event timestamp is less than GVT
- the `min_output_timestamp` value associated with the event is less than GVT.

Then the kernel must determine the maximum timestamp across all the events in that set, which represents the final realignment time for the LP local state. In case the set is empty, no realignment takes place, and the latest checkpoint of that LP preceding GVT is selected for the CCGS.

3 Related work

The issue of identifying and recording consistent global states in (distributed) applications involving multiple processes has been thoroughly studied in the context of fault tolerance in order to avoid the so called domino effect, i.e., restoration to the initial state due to the lack of a more recent consistent snapshot (Elnozahy et al., 2002). The two most common approaches for addressing this problem are referred to as coordinated checkpointing (see e.g., Cao and Singhal, 1998), where an explicit coordination scheme is adopted by the processes to take mutually consistent local checkpoints and communication induced checkpointing, where control information is piggybacked on application messages in order to direct forced checkpoints on the recipient process on the basis of local predicates evaluated by exploiting this information (see Alvisi et al., 1999 for a performance analysis of various communication induced protocols). Compared to these approaches, we leave complete autonomy in the checkpointing activities of the LPs, so that they can take checkpoints at their own pace while the execution proceeds.[3] On the other hand, we reconstruct a consistent global snapshot via an update phase of the logged local states. In the latter aspect, our approach shows similarities with fault tolerance techniques based on both uncoordinated checkpointing and message logging (see e.g., Elnozahy et al., 2002; Strom and Yemini, 1985). In fact, these techniques rely on a replay phase for eliminating mutual dependencies among the originally restored state logs. However, compared to these solutions, we do not require the execution of an explicit distributed protocol for the identification and elimination of the dependencies. Instead, we exploit the results of GVT calculation (anyway required for memory recovery purposes) in order to heuristically determine where to realign the local states of the involved LPs while ensuring global snapshot consistency. Similar considerations can be made when comparing our proposal to the global snapshot collection protocols in Chandy and Lamport (1985), Hélary (1989), Kshemkalyani and Wu (2007) and Mattern (1993), some of which have been proposed just in the context of detection of global predicates in distributed systems. These solutions require an explicit distributed algorithm to be executed among the processes participating in the computation for the determination of the consistent global snapshot, which is avoided in our proposal thanks to the exploitation of the results of the GVT protocol.

In the context of Parallel Discrete Event Simulation, the work in Abrams and Richardson (1991) has addressed issues concerning global termination conditions. This is done via categorisation of non-trivial termination predicates, and via the introduction of algorithms suited for detecting predicates in different categories. These algorithms implicitly assume the availability of LPs' state histories for evaluating the termination condition. In this aspect, our proposal is orthogonal to this work since we focus on the lower level mechanisms used for the treatment of state logs in order to provide the input data required by the module implementing the logic for the evaluation of the termination condition. Also, our approach is specialised and optimised for supporting the category of termination conditions relying on stable predicates (since we provide supports for the iterative evaluation of the termination condition on the basis of the periodic calculation of a new GVT value and the identification of an associated CCGS).

To the best of our knowledge, there has been a single proposal based on consistent global checkpoints in the context of optimistic simulation systems (Moreira et al., 2005), which uses control information piggybacked on application messages in order to track the state dependencies and direct forced checkpoints to construct consistent global snapshots to be exploited for synchronisation purposes. However, this has been shown to provide adequate performance only for reduced simulation model sizes. Hence approaches based on independent checkpointing activities of the LPs, as we have assumed in our proposal, remain the most effective ones in general applicative scenarios. Also, compared to the work in Moreira et al. (2005), our solution addresses the issue of exploiting consistency of global states not for synchronisation purposes, but for the evaluation of global predicates in synergy with GVT advancement.

4 Experimental data

4.1 The case study

The application level code we have used in this experimental study is a parameterisable simulation software of a mobile Personal Communication System (PCS). In the used configuration, we explicitly simulate power regulation, by taking into account both fading effects and channel interference, so to perform statistical inferences on the signal strength (or signal quality) based on the Signal-to-Interference Ratio (SIR) (Kandukuri and Boyd, 2002). In the experiments we have simulated a coverage area with 1024 cells, each managing 200 channels, in a peak load configuration where the call duration is exponentially distributed, with average value of 2 min, and the call arrival rate, also exponentially distributed (Boukerche et al., 1999; Carothers et al., 1995), is set to determine an average utilisation factor of 75% for each channel. The supported mobility model is random-walk, with mobile permanence time within each cell exponentially distributed with mean equal to 10 min. Each cell is modelled by a different LP and the adjacency between different cells in the coverage area (determining the possibility of call handoff in case the corresponding mobile switches between neighbour cells) is evaluated by assuming hexagonal shape for the cells. Each time an incoming call arrives, the LP allocates a new record for keeping track of information about, e.g., power assignment, and links it to a list of already active records, each of which is released when the corresponding call ends or is subject to handoff towards a different cell. Finally, SIR calculation and power assignment for a specific call is triggered upon the assignment of a channel for serving the call (hence upon the call arrival or when the handoff event for an ongoing call occurs, which requires channel reassignment at the destination cell). This gives rise to a mixture of both fine grain events (e.g., end call events, with no costly calculation upon channel release) and coarser grain events (e.g., the previously mentioned call arrival events, with costly calculation of the SIR).

4.2 Test settings and measured parameters

All the runs have been carried-out on a Quad-Core machine equipped with four 2.4-GHz/4MB-cache Intel processors and 4 GB of RAM memory, running LINUX (kernel version 2.6.22). Four instances of the ROOT-Sim kernel have been activated on this machine, each managing 256 LPs.

To assess the effectiveness of our CCGS construction mechanism, we have measured the event rate (i.e., the number of committed simulation events per wall-clock time unit), which is a typical parameter representative of the execution speed. The event rate achieved via our optimised mechanism has been compared against the event rate achieved via the following two schemes:

i Realignment of the committed consistent global state to the new computed value of GVT. As mentioned before, this method allows the exploitation of the maximal portion of the committed computation. However, it does not exploit the structure of the computation (in terms of real dependencies among local states) to reduce the realignment overhead.

ii Simple collection of the latest checkpoints preceding the new GVT value, hence providing no guarantee of consistency of the identified global snapshot. This is a baseline configuration showing no realignment overhead, which is used as the reference for the evaluation of the overhead of our mechanism.

For the aforementioned test case, the application level callback function OnGVT() implements a termination predicate based on the total number of calls locally started at the LPs (i.e., calls activated at an LP due to handoff events among different LPs are not counted). This has been done to achieve correct predicate evaluation even for the scheme in point (ii), which does not guarantee consistency of the global snapshot. Hence, we allow comparative analysis of the different schemes under correct computing steps, despite the different guarantees they provide. Correctness is ensured by the fact that the LPs' attributes involved in the selected global predicate are actually independent of each other (i.e., no event scheduled across different LPs changes the value of the counter indicating the number of calls locally activated on the basis of the call generation pattern).

In the experimental study we have considered the checkpoint interval χ of the LPs as an independent parameter. Variation of this parameter allows us to comparatively observe the performance of the considered global snapshot collection schemes at the point in which the best balance between checkpointing and event replay costs is achieved, where the event replay cost is due to both coasting forward operations upon rollback occurrences and realignment (if any) for the global snapshot construction.

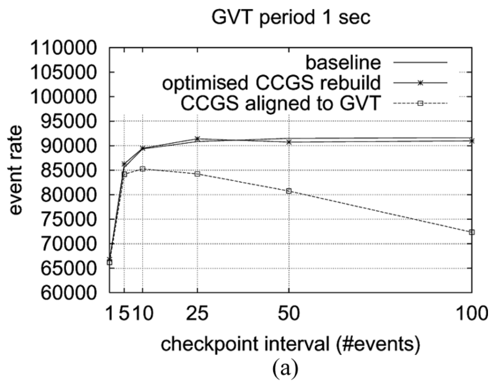
The second independent parameter in the study is the time period for GVT calculation and global snapshot construction. Lower values may tend to represent, e.g., interactive scenarios, where an end-user might require frequent (committed) intermediate outputs on the global state of the simulated system. On the other hand, larger values are representative of situations where, e.g., termination detection via global predicate evaluation is not a time critical operation. Thus it can be triggered on a reduced frequency basis according to GVT calculations whose primary target is the periodic recovery of memory.

4.3 Results

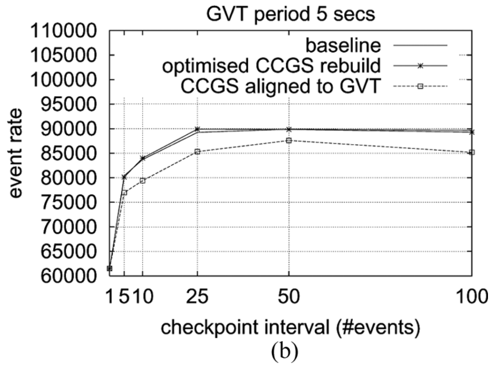
In Figure 5 we report the execution speed, in terms of event rate, while varying the checkpoint interval of the LPs for two different values of the GVT period, namely 1 s and 5 s. Each reported value refers to the event rate evaluated while simulating about 60 virtual time minutes of the previously described peak load scenario for the PCS system. Also, each value results as the average over five samples, obtained in runs executed with different random seeds.

By the results we can draw the following main conclusions. Our proposal shows negligible overhead when compared to the baseline configuration not ensuring consistency of the committed global snapshot. Also, compared to the case of realignment of the global state to GVT, for this specific test-bed application, our mechanism allows improvements of the execution speed at the points where the performance is maximised vs. the checkpoint interval of the LPs. As expected, this is noted especially for the case of more frequent GVT calculation (i.e., the case of GVT calculation each 1 s, where the maximum achieved gain in the execution speed is on the order of 9%). We recall this is the case well representing, e.g., interactive environments requiring frequent output data production based on frequently refreshed (consistent) snapshots of the simulated system state. However, the execution speed gain provided by our mechanism still remains on the order of 4% in the case of GVT calculation each 5 s, namely the typical case where the new GVT value is computed according to a relatively stretched period, mostly tailored to the avoidance of excessive growth of memory usage, so to not incur performance degradation phenomena due to reduced locality and its impact on the underlying memory hierarchy.

Figure 5 Execution speed results: (a) GVT period 1 s and (b) GVT period 5 s



(a)



(b)

We also observe that, compared to CCGS realignment to GVT, our proposal allows much more flat performance vs. variations of the checkpoint interval. This is an indication of better performance guarantees even in the case of suboptimal choice of the checkpoint interval. Also, the top performance achieved with our heuristic based scheme and with the baseline are in practice identical for the two different values of the GVT period. Given that the GVT protocol exhibits negligible overhead due to the tight coupling of the hardware architecture, this is again an indication of negligible overhead for the optimised CCGS reconstruction approach we have presented.

As a final note, the gain from the optimised approach over the scheme with realignment to GVT just comes from the achievement of better balance between state saving and event replay costs (the latter being paid both in classical coasting forward phases when rollbacks occur and in realignment phases of the local states of the LPs when the construction of the CCGS takes place) and not from variations of the rollback pattern in the parallel execution. Specifically, we have noted that the rollback pattern (in terms of both rollback frequency and rollback length) remains the same over all the runs, with a flat efficiency value vs. the checkpoint interval, which is on the order of about 97/98%.[4]

5 Summary

In this paper we have addressed the problem of how to (re)use local checkpoints of the logical processes taken in an uncorrelated manner during the forward execution phase in an optimistic simulation system in order to build committed and consistent global states. We have addressed this problem from both a methodological perspective, by providing a lightweight heuristic approach for the identification of a reduced set of checkpoint update operations aimed at the construction of the committed and consistent global state, and from a pragmatical perspective, via the integration of the proposed approach within an operating optimistic simulation environment. Some experimental data for the assessment of the run-time behaviour of the integrated environment, e.g., in terms of overhead of the global state management sub-system, are also provided.

Acknowledgment

The author would like to thank Diego Cucuzzo, Stefano D'Alessio and Roberto Toccaceli for their help in developing several parts of the ROOT-Sim environment.

References

Abrams, M. and Richardson, D. (1991) 'Implementing a global termination condition and collecting output measures in parallel simulation', *Proceedings of SCS Multi-Conference on Parallel and Distributed Simulation*, pp.86–91.

Alvisi, L., Elnozahy, E.N., Rao, S., Husain, S.A. and Mel, A.D. (1999) 'An analysis of communication induced checkpointing', *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing (FTCS)*, pp.242–249.

Boukerche, A., Das, S.K., Fabbri, A. and Yildz, O. (1999) 'Exploiting model independence for parallel PCS network simulation', *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, IEEE Computer Society, pp.166–173.

Cao, G. and Singhal, M. (1998) 'On coordinated checkpointing in distributed systems', *IEEE Transactions on Parallel Distributed Systems*, Vol. 9, No. 12, pp.1213–1225.

Carothers, C.D., Fujimoto, R.M. and Lin, Y.B. (1995) 'A case study in simulating PCS networks using Time Warp', *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, IEEE Computer Society, pp.87–94.

Chandy, K.M. and Lamport, L. (1985) 'Distributed snapshots: determining global states of distributed systems', *ACM Transactions on Computer Systems*, Vol. 3, No. 1, pp.63–75.

Elnozahy, E.N., Alvisi, L., Wang, Y-M. and Johnson, D.B. (2002) 'A survey of rollback-recovery protocols in message-passing systems', *ACM Computing Surveys*, Vol. 34, No. 3, pp.375–408.

Fleischmann, J. and Wilsey, P. (1995) 'Comparative analysis of periodic state saving techniques in Time Warp simulators', *Proceedings of the 9th Workshop on Parallel and Distributed Simulation, IEEE Computer Society*, pp.50–58.

Fujimoto, R.M. (1990) 'Parallel discrete event simulation', *Communications of the ACM*, Vol. 33, No. 10, pp.30–53.

Hélary, J-M. (1989) 'Observing global states of asynchronous distributed applications', *Proceedings of the 3rd International Workshop on Distributed Algorithms*, Springer-Verlag, LNCS 392.

Hélary, J-M., Netzer, R.H.B. and Raynal, M. (1999) 'Consistency issues in distributed checkpoints', *IEEE Transactions on Software Engineering*, Vol. 25, No. 2, pp.274–281.

Jefferson, D.R. (1985) 'Virtual time', *ACM Transactions on Programming Languages and System*, Vol. 7, No. 3, pp.404–425.

Kandukuri, S. and Boyd, S. (2002) 'Optimal power control in interference-limited fading wireless channels with outage-probability specifications', *IEEE Transactions on Wireless Communications*, Vol. 1, No. 1, pp.46–55.

Kshemkalyani, A. and Wu, B. (2007) 'Detecting arbitrary stable properties using efficient snapshots', *IEEE Transactions on Software Engineering*, Vol. 33, No. 5, pp.330–346.

Mattern, F. (1993) 'Efficient algorithms for distributed snapshots and global virtual time approximation', *Journal of Parallel Distributed Computing*, Vol. 18, No. 4, pp.423–434.

Moreira, E.M., Santana, R.H.C. and Santana, M.J. (2005) 'Using consistent global checkpoints to synchronize processes in distributed simulation', *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*, pp.43–50.

Preiss, B.R., Loucks, W.M. and MacIntyre, D. (1994) 'Effects of the checkpoint interval on time and space in Time Warp', *ACM Transactions on Modeling and Computer Simulation*, Vol. 4, No. 3, pp.223–253.

Quaglia, F. (2001) 'A cost model for selecting checkpoint positions in Time Warp parallel simulation', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 4, pp.346–362.

Ronngren, R. and Ayani, R. (1994) 'Adaptive checkpointing in Time Warp', *Proc. 8th Workshop on Parallel and Distributed Simulation*, Society for Computer Simulation, pp.110–117.

Samadi, B. (1985) *Distributed Simulation Algorithms and Performance Analysis*, PhD Thesis, Computer Science Department, University of California, Los Angeles.

Strom, R.E. and Yemini, S. (1985) 'Optimistic recovery in distributed systems', *ACM Transactions on Computer Systems*, Vol. 3, No. 3, pp.204–226.

Tel, G. (1991) *Topics in Distributed Algorithms*, PhD Thesis, Cambridge Univarsity Press, Cambridge.

Toccaceli, R. and Quaglia, F. (2008) 'DyMeLoR: dynamic memory logger and restorer library for optimistic simulation objects with generic memory layout', *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, IEEE Computer Society, pp.163–172.

Notes

[1]GVT calculation is a form of global predicate, termed *distributed infimum approximation* in Tel (1991), which also relies on information associated with messages in transit. However, this predicate only considers logical time values, and does not take generic LP state information into account, which could play a relevant role for termination detection in specific applicative contexts.

[2]Since GVT is evaluated considering the timestamps of unprocessed events and of events associated with messages still in transit, we intend realignment to GVT as the rebuild of the LP state value at the time of its latest committed event with timestamp less than GVT.

[3]As already hinted, autonomy is fundamental for allowing performance effectiveness since checkpointing in optimistic simulation is a support for synchronisation. Hence, compared to stable storage state logs for fault tolerance purposes, it requires to be executed relatively more frequently in order to cope with the endemic phenomenon of rollback occurrence in the optimistic run.

[4]In an optimistic simulation run, the efficiency is evaluated as the percentage of executed events which are not eventually rolled back.