A Lightweight Heuristic-based Mechanism for Collecting Committed Consistent Global States in Optimistic Simulation

Diego Cucuzzo, Stefano D'Alessio, Francesco Quaglia, Paolo Romano
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma

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 (i.e. the logical time they refer to is lower than the current GVT value). This is done by introducing a heuristic-based mechanism relying on update operations applied to local committed checkpoints of the involved 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 performed. 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 value, 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. Another application concerns interactive simulation environments, where (aggregate) output information about committed and consistent snapshots needs to be frequently provided, hence requiring lightweight mechanisms for the construction of the snapshots.

1 Introduction

Optimistic discrete event simulation systems are based on state saving techniques used to support rollback operations when causality violations occur [9, 11]. Typically, the state logs, also known as checkpoints, are used exclusively for synchronization 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 optimistic 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) [14] ([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 a 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., [8, 16, 17]) in order to optimize the tradeoff between state saving and rollback overheads.
- 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.

[1]GVT calculation is a form of global predicate, termed *distributed infimum approximation* in [20], 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.

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

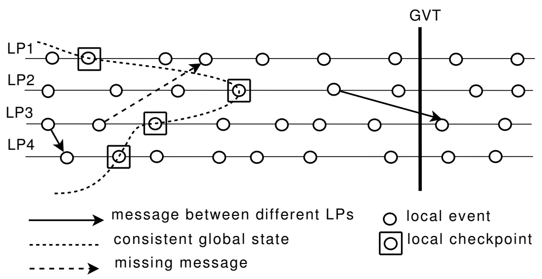
2.1 Basics and Motivations

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.

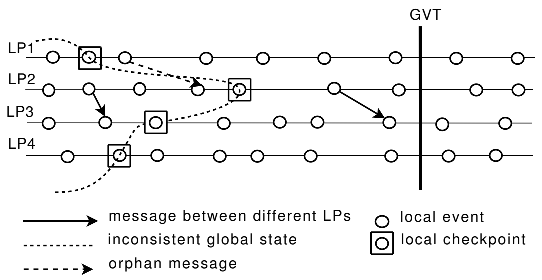
As sketched in the Introduction, we consider the typical case in which it does not exist any distributed protocol for taking local checkpoints of the LPs in a coordinated manner. This is done in order to allow the maximal flexibility concerning checkpoint decisions at different LPs in favor of optimizing, 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 mechanism 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 realized in an effective and application transparent manner by reprocessing events with timestamps in between the checkpoint time and the target logical time, while redirecting the state updates to the checkpoint buffer (instead of the current state buffer of the LP). Also, this 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 also retained, independently of the schedule of operations performing memory recovery.

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 first 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.



(a) Missing message



(b) Orphan message

Figure 1. Some Examples.

Given this premise, Figure 1(a) 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(b) 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 [6]. 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(a)). The presence of orphan messages makes a global state not causally consistent, while missing messages are admitted when the reference criterion is the consistency of the global snapshot formed by a collection of local process states.

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. That is, the resulting global state is the most recent committed snapshot of the simulation.

However, a serious 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 a uniform distribution. This is exactly the case of GVT, since its value is determined a posteriori of the execution and of the 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 leads to

[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.

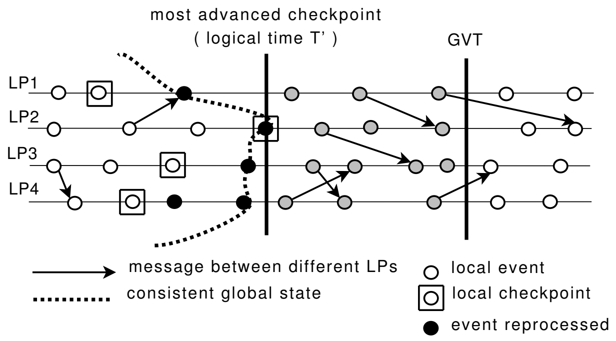


Figure 2. Realignment to a Reference Time Different from GVT.

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.

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 analyzing 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 behavior 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 behavior 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).

2.3 An Implementation

In this section we concisely describe an implementation of the heuristic-based CCGS construction mechanism within an operating optimistic simulation environment. This description allows us to provide practical evidence of the viability of such a mechanism.

One main point our work is based on is the ability to reprocess already committed events (like in a classical coasting forward phase) while applying the state updates to the checkpoint buffer (not to the current LP state buffer). This should be achieved in an application transparent manner.

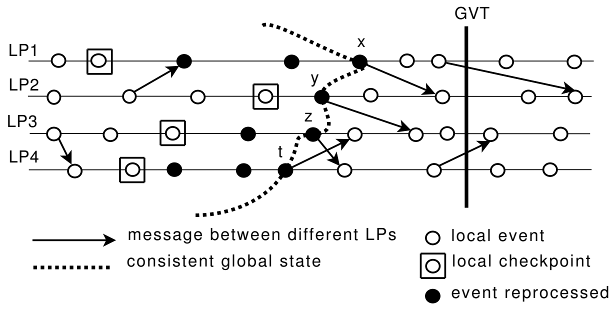


Figure 3. Heuristic Method Example.

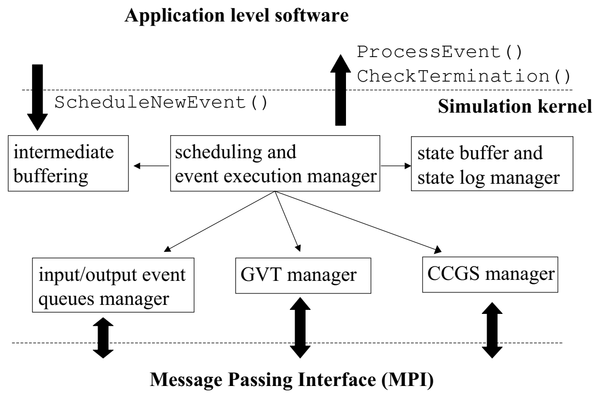

Application level software
ProcessEvent()
CheckTermination()
ScheduleNewEvent()
Simulation kernel
intermediate buffering
scheduling and event execution manager
state buffer and state log manager
input/output event queues manager
GVT manager
CCGS manager
Message Passing Interface (MPI)

Figure 4. Software Architecture of the Operating Simulation Platform.

Figure 4 schematizes the software architecture of our simulation platform, developed using `C` technology. By the scheme, the application level software is completely separated from the underlying housekeeping management software (that we also refer to as simulation kernel), which exploits standard MPI facilities for the management of the distributed message passing environment. The interaction between the application layer and the simulation kernel occurs via:

- Two callback services offered by the application level software, namely `ProcessEvent()` and `CheckTermination()`.
- A service offered by the underlying simulation kernel called `ScheduleNewEvent()`.

In this design, the `ProcessEvent()` callback has a set of parameters identifying the event to be processed and the specific state buffer associated with the LP on which the event is occurring. The underlying intermediate buffering facilities handle, in a transparent manner to the application programmer, the messages associated with new events produced by `ProcessEvent()` and inserted in the system via the invocation of the `ScheduleNewEvent()` service offered by the platform. The latter 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.

The underlying platform also supports checkpointing facilities for recording the state of an LP periodically before invoking the `ProcessEvent()` callback for that same LP. 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.

The callback `CheckTermination()` can be used to evaluate a global predicate indicating whether the computation has ended. The master instance of the distributed simulation kernel invokes this callback after GVT has been calculated and it has collected the CCGS, which is passed as input parameter to the callback in the form of an array of state values (one for each LP). On the other hand, when the new GVT value is known, the master and all the slave instances of the simulation kernel apply the heuristic-based mechanism described in Section 2.2. The slaves eventually communicate to the master the reconstructed local states of the LPs they are handling, which will belong to the CCGS, so that the master is eventually able to correctly invoke the `CheckTermination()` callback passing as input the committed, consistent global state for the evaluation of the global predicate associated with termination detection. To optimize run-time behavior, the master performs the collection of the LPs' states forming the CCGS in an asynchronous manner.

This type of software organization supports the heuristic-based mechanism for the identification and construction of the CCGS in a totally transparent manner to the application software since the underlying platform can drive the execution of the update phase of a committed checkpoint by simply invoking the `ProcessEvent()` callback passing as input the pointer to the checkpoint buffer instead of the pointer to the current LP state buffer. 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.

Given that the CCGS for which the predicate is evaluated is periodically determined in an application transparent manner when the new GVT is computed, the type of predicate supported by `CheckTermination()` is *monotonic* [14], namely it is associated with a monotonic function of the global state of the simulation.

Concerning GVT calculation, we have implemented an optimized 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 [18], 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 needed to tweak 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 analyzing 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) [7]. The two most common approaches for addressing this problem are referred to as coordinated checkpointing (see, e.g., [4]), 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 [2] for a performance analysis of various 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., [7, 19]). 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 [6, 10, 13, 14], 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 [1] has addressed issues concerning global termination conditions. This is done via categorization 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 termination condition. Also, our approach is specialized and optimized 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 [15], 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 synchronization 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 [15], our solution addresses the issue of exploiting consistency of global states not for synchronization 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 parameterizable 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) [12]. In the experiments we have simulated a large 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 minutes, and the call arrival rate (also exponentially distributed [3, 5]) is set to determine an average utilization 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 minutes. Each cell is modeled 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 neighbor cells) is evaluated by assuming hexagonal shape for the cells. Finally, SIR calculation and power assignment for a specific call is triggered upon the assignment of a channel to 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

[3]As already hinted, autonomy is fundamental for allowing performance effectiveness since checkpointing in optimistic simulation is a support for synchronization. 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.

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). Hence, we have an execution scenario with an intermediate average event granularity. Also, the LP state size is about 5 KB.

As a final note, although the employed simulation platform supports parallelization and optimistic synchronization in a totally transparent manner to the application programmer, who is therefore required to structure the application level code with no particular concern towards parallel execution, we have used at the application level a classical pre-scheduling technique for the handoff events spanning across different cells (i.e. different LPs). With this technique, the handoff events for both the source cell and the destination cell are scheduled together just when processing the arrival event of the call that will generate the handoff. This tends to increase the actual lookahead of the simulation, thus typically favoring parallel execution performance.

4.2 Test Settings and Measured Parameters

All the runs have been carried-out on an SMP machine equipped with 4 Xeon CPUs (2.0 GHz) and 4 GB of RAM memory, running LINUX (kernel 2.6). Four instances of the simulation 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 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 at all, 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 `CheckTermination()` implements a termination predicate based on the total number of calls locally started at all 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).

We have varied two parameters in the simulation study. The first one is the checkpoint interval χ of the LPs. 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 collection. 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.

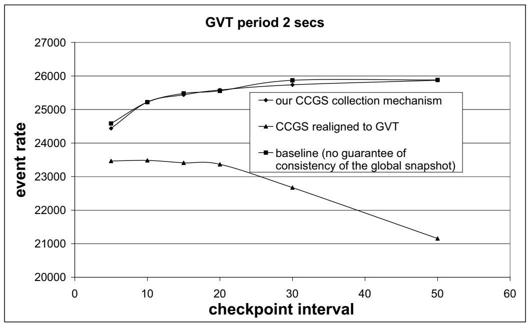
4.3 Results

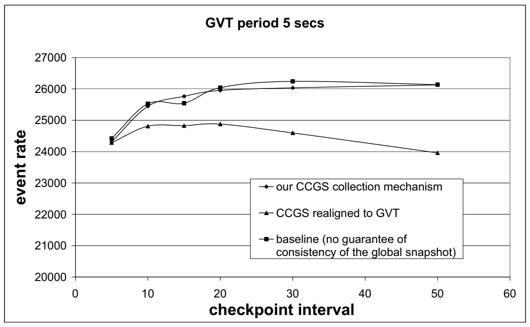
In Figure 5 we report the execution speed, in terms of event rate, while varying the checkpoint interval of the LPs for three different values of the GVT period, namely 2 secs, 5 secs and 10 secs. Each reported value refers to the event rate evaluated while simulating about 30 virtual time minutes of the previously described peak load scenario for the PCS system. Also, each value results as the average over 5 samples, all executed with different random seeds.

By the results we can draw the following main conclusions. Our proposal shows minimal overhead when compared to the baseline configuration not ensuring consistency of the collected 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 point where the performance is maximized vs the checkpoint interval of the LPs. As expected, this is noted especially for the case of more frequent GVT calculation and global snapshot collection (i.e. the case of GVT calculation each 2 secs, where the maximum achieved gain in the execution speed is on the order of 10%). 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 10 secs, 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.

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.

As a final note, the gain from our proposal 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 85% ([4]).





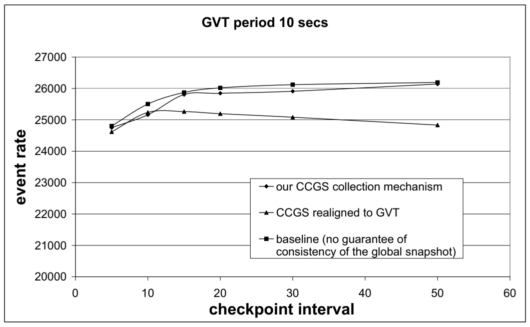


Figure 5. Execution Speed Results.

References

[1] M. Abrams and D. Richardson. Implementing a global termination condition and collecting output measures in parallel simulation. In *Proceedings of SCS Multi-Conference on Parallel and Distributed Simulation*, pages 86–91, Jan. 1991.

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

[3] A. Boukerche, S. K. Das, A. Fabbri, and O. Yildz. Exploiting model independence for parallel PCS network simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, pages 166–173. IEEE Computer Society, May 1999.

[4] G. Cao and M. Singhal. On coordinated checkpointing in distributed systems. *IEEE Transactions on Parallel Distributed Systems*, 9(12):1213–1225, 1998.

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

[6] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[7] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[8] J. Fleischmann and P. Wilsey. Comparative analysis of periodic state saving techniques in Time Warp simulators. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pages 50–58. IEEE Computer Society, June 1995.

[9] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[10] J.-M. Hélary. Observing global states of asynchronous distributed applications. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*. pringer-Verlag, LNCS 392, 1989.

[11] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and System*, 7(3):404–425, July 1985.

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

[12] S. Kandukuri and S. Boyd. Optimal power control in interference-limited fading wireless channels with outage-probability specifications. *IEEE Transactions on Wireless Communications*, 1(1):46–55, 2002.

[13] A. Kshemkalyani and B. Wu. Detecting arbitrary stable properties using efficient snapshots. *IEEE Transactions on Software Engineering*, 33(5):330–346, May 2007.

[14] F. Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel Distributed Computing*, 18(4):423–434, 1993.

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

[16] F. Quaglia. A cost model for selecting checkpoint positions in Time Warp parallel simulation. *IEEE Transactions on Parallel and Distributed Systems*, 12(4):346–362, Feb. 2001.

[17] R. Ronngren and R. Ayani. Adaptive checkpointing in Time Warp. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, pages 110–117. Society for Computer Simulation, July 1994.

[18] B. Samadi. *Distributed Simulation Algorithms and Performance Analysis*. PhD thesis, Computer Science Department, University of California, Los Angeles, 1985.

[19] R. E. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3):204–226, 1985.

[20] G. Tel. *Topics in Distributed Algorithms*. PhD thesis, Cambridge University Press, Cambridge, 1991.