Thesis proposal: Bounded recovery in distributed discrete real-time simulations*

Marcus Brohede
Technical Report HS-IKI-TR-06-008
School of Humanities and Informatics
University of Skövde
P.O. Box 408, SE-541 28 Skövde, Sweden
marcus.brohede@his.se

November 22, 2006

Abstract

This thesis proposal defines the problem of recovery in distributed discrete real-time simulations with external actions; real-time simulations with simulation actions in the real world. A problem that these simulations encounter is that they cannot rely on rollback-based recovery (use of checkpoints) for two reasons. First, some actions in the "real world" cannot be undone, and second, the time allowed for recovery tends to be short and bounded. As a result there is a need for some form of error masking for this category of sim-

*This work was funded by WITAS, CUGS, the University of Skövde, and the Information Fusion Project

ulations. We propose an infrastructure for these simulations with external actions based on an active distributed real-time database that features replication of the distributed simulation. The degree of replication is based on the dependability requirements of the individual nodes in the simulation. A guideline for how to decompose a distributed real-time simulation into parts with different requirements on the replication protocol is also defined as an interesting topic to investigate further. We introduce the simulation infrastructure "Simulation DeeDS" featuring a replication-based recovery strategy for the category of simulations mentioned. We also show that some information fusion applications are indeed examples of applications that need real-time simulation with external actions and as such can benefit from the proposed infrastructure.

Papers

This thesis proposal is based on the following papers:
(Brohede & Andler 2002), (Brohede & Andler 2003), (Brohede, Andler & Son 2005), and (Brohede & Andler 2005).
In (Brohede & Andler 2002) and (Brohede & Andler 2003) we introduce the concept of having a distributed active real-time database as a communication medium in distributed real-time simulations. The majority of content from these papers can be found in section 5.
In (Brohede et al. 2005) we describes a novel simulation synchronization pro-

tocol built on our proposed real-time simulation infrastructure. The results from this paper is found in section 5.4.3.
In (Brohede & Andler 2005)we highlight key requirements for information fusion applications that have need for real-time. The particular requirements are found in 2.1 and how we address them is explained in section 5

1 Introduction

Simulations either use checkpoints to recover from crashes or forward recovery (e.g., extrapolate a new position if a position update is missed) in order to minimize wasted computations. For checkpoint-based distributed simulations this means to regularly create global checkpoints and in case of a simulation failure (e.g., a crashed node) restart all simulation nodes at the latest global checkpoint and run forward to the correct point in the simulation. This type of recovery can be implemented in HLA simulations (Dahmann, Fujimoto & Weatherly 1997). Lüthi & Berchtold (2000) has shown that checkpoint-based recovery for distributed simulations can be improved, by limiting the number of nodes that need to be restarted at these checkpoints.

Simulations that use forward recovery (often used in DIS (DIS Steering Committee 1994)) need to perform some compensating action when it is discovered that the recovery is incorrect, e.g., when new position updates

enters the simulation that show that the extrapolation used to compensate for missed positions is incorrect.

Real-time simulations interact with the environment (e.g., a human operator or some existing machine) and can generate external actions. A problem with recovery in such systems is that external actions may not be possible to undo.

For example, consider precision agriculture, which is an information fusion research area where real-time simulation is needed. More specifically, a tractor about to deploy fertilizer on a field want to fuse information from sensors that give information from soil quality together with historical information such as previous years yield in combination with weather conditions, as well as with simulated crop yields based on different fertilizer amounts and different weather conditions. The results from the simulations must be delivered in real-time otherwise the tractor will have moved too far away from the particular part of the field for which the simulated data were produced. Since, the result from the simulation is used to determine the amount of fertilizer a crashed simulation or a simulation that cannot keep deadlines can have irrevocable results on the fertilizer deployment; deployed fertilizer cannot be retrieved and reversing the tractor to deploy more fertilizer will not be cost effective.

Because of the external actions returning to a previous state may not be possible. Furthermore, simply stoping or delaying execution in order for a recovering simulation node to catch up is not feasible. Hence, for real-time simulations with external actions, current recover approaches are not sufficient. This means that for applications such as the agriculture example there is a need to mask failures, or to bound the recovery time and make sure that this time is short enough to recover before a new real-world action is expected from the simulation.

2 Background

2.1 Information Fusion Applications Requirements

In information fusion applications such as the precision agriculture example, information sources are located at geographically dispersed sites. Hardware and software differences are natural, i.e., heterogeneity is expected and real-time is required. The total amount of information is overwhelming and must, therefore, be controlled to fit the individual information receivers. Any infrastructure supporting the fusion process must be dynamic and adaptive, since information receivers or producers can be added or removed dynamically. In addition, the dynamic nature of the fusion process also means that individual information receivers can change how they want the fused information to be presented, and information sources such as sensors can change how they produce data. All these factors add to the overall complexity of

the information fusion process.

The information fusion process (see Figure. 1) puts requirements on the underlying infrastructure, especially if any part of the fusion process require real-time guarantees. The parts of the information fusion process include information producers, and consumers. An information producer is a source that can provide historical data (past), current state information (present), or try to predict future behavior (future), or combine a number of sources into a fused source. Producers and consumers are distributed over one or more nodes in a network. We here list some of these requirements.

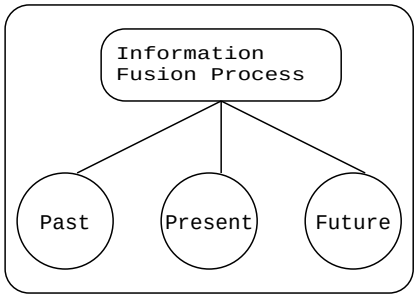

Figure 1: Example of information fusion process.

Three different categories of requirements on information fusion applications that need real-time simulation with external actions are presented: *configuration*, *time*, and *robustness* requirements.

2.2 Configuration requirements

The heterogeneity requirement An infrastructure for information fusion must be capable of handling *heterogeneity*. Given the situation with many independent producers of information it is unlikely that they will all use the same hardware, software, data structures, database schema, standards, degrees of uncertainty etc. Therefore, a useful infrastructure for information fusion must address heterogeneity.

The distribution requirement An infrastructure for information fusion must be capable of handling *distributed* information sources, i.e., distribution is inherent in the concept of information fusion.

The independence requirement There must be *independence* between producers and consumers of information in an information fusion process. This means that producers of information should not be required to know anything about the receiver of the information. Conversely, consumers should only need to specify which information they are interested in without specifying where this information exists. However, an infrastructure must be capable of handling complex correlations between information sources. For example, a consumer could be interested in some information only accessible by combining a number of information sources. The independence requirement is vital since data sources can exist outside organizational boundaries.

The scalability requirement The potential amount of information that can be processed in an information fusion application is constantly increasing resulting in a need for an infrastructure that scales. Scaling both in number of sensors and nodes in the fusion process and in the amount of data handled must be addressed.

The adaptivity requirement Changes in both the node structure and how various nodes interact can change over time must be handled; *adaptivity* in the information fusion infrastructure is needed. For example, information producers or consumers can emerge or disappear. Structure changes cannot always be controlled, since information sources can exist outside the local organization. Changes to individual nodes in the fusion process must also be addressed, e.g., a node producing sensor data could start producing the sensor data in a different format. In addition, new information sources can have different data structures describing the same real world entity as some already known information source. In such a case it is desirable to detect this and benefit from this. Finally, consumers of fused data might change how they want the information presented.

2.3 Time Requirements

The temporal property requirement The information fusion process brings together not only data from distributed data sources, but may also try to use historical data, as well as future predictions, in order to create an

improved (fused) value. We believe that any infrastructure used in information fusion needs to be able to efficiently store and retrieve historical data, as well as determine likely future situations, e.g., through simulations.

Requirements that are added due to time and dependability constraints in real-time and embedded systems include predictability and timeliness.

The predictability requirement In addition to performing their designated tasks, real-time systems must execute them using predictable amounts of resources, e.g., cpu, memory, etc. For an information fusion process, this means that nodes producing, consuming, or merging information must have predictable behavior and resource usage. In other words, usage of resources such as processing time, memory, or network bandwidth must be bounded. One problem that arise in the specific application area of information fusion is that not all parts in the process is controlled by one organization. This means that the design of the information fusion application must be done so that real-time dependent parts do not rely exclusively on un-controlled nodes outside organizational boundaries.

The timeliness requirement All parts in the information fusion process that any real-time application relies on must be timely. A timely system is scheduled such that it meets all its deadlines. This requires predictability and sufficient efficiency. Information sources that cannot be controlled (e.g., they lie outside the local organization) must not be critical to the information fusion process, but rather seen as parts that can improve or optimize

the value of the fused information.

2.4 Robustness requirements

The fault tolerance requirement If the information fusion process is used in systems which affects the real-world it must be made fault tolerant. Fault tolerance increases dependability in a system, for example, by replicating important parts.

The uncertainty management requirement Uncertainty management of information sources introduces two different problems. First, if there are data sources with bad precision, e.g., if a sensor is known to have an uncertainty of ϵ a received value x is actually $x \pm \epsilon$. Second, to what degree can an information source be trusted at all, i.e., if there are arbitrarily failures in the source's information or if the source cannot be trusted for some other reason. This is a form of the Byzantine generals problem (Lamport, Shostak & Pease 1982). For example, a newspaper writes of a lucrative stock, but only gives good information 50% of the time.

2.5 Discrete event simulation

In distributed discrete event simulations ((Misra 1986) and (Fujimoto 1990)) a number of logical processes (LP) simulates a number of physical processes.

The simulation progresses with the LPs processing events in their respective event list. Communication between LPs are carried out through message passing. A fundamental rule in discrete event simulation is the *dependency relation* between all events in the complete simulation. This relation states that no event e can occur unless all the events on which e depends on have occurred. In particular, e must have an associated time of occurrence (timestamp) that is higher than the timestamps of all events it depends on. Misra (1986) states that a simulation is correct if it can predict the sequence of message transmissions in the physical system. The major difference between the physical system and a simulation is that the simulation should be able to operate at a different (usually higher) speed.

Real-time simulation is a special type of simulation where individual events must be completed before predefined *deadlines* even under *worst-case* conditions (Ghosh, Panesar, Fujimoto & Schwan 1994). Real-time simulations should not be confused with *high-performance* simulations, which aim at processing events at a high *average* rate. When constructing real-time systems it may be necessary to simulate parts of the system due to cost or risk. This type of real-time simulation where implemented and simulated parts together interact with an environment is sometimes called a *hybrid simulation* (Ghosh, Fujimoto & Schwan 1993).

Synchronization in distributed simulation can be either *conservative* or

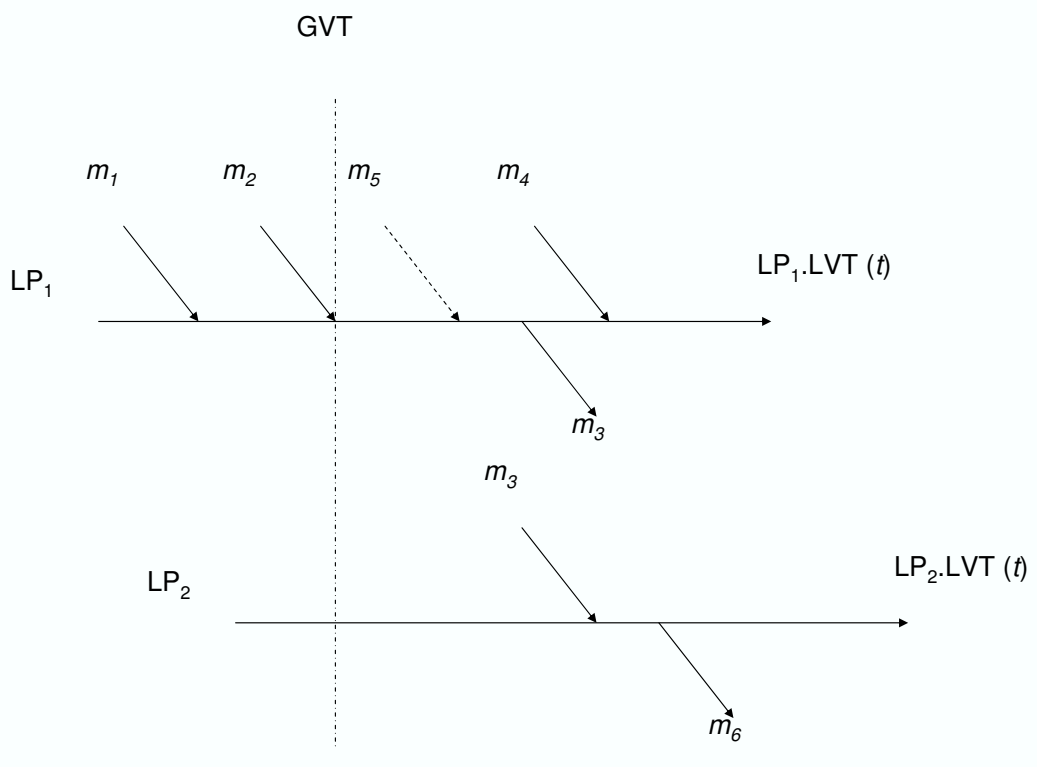

Figure 2: Rollback example in TW.

optimistic (Reynolds 1988). In conservative distributed simulations there is no speculative execution, whereas in optimistic simulation incorrect speculative execution must be handled.

Time Warp (TW) (Jefferson 1985) is a common optimistic synchronization protocol used in discrete event simulations to guarantee that the dependency relation is kept throughout the entire simulation. TW allows independent logical processes (LP) to process events in their own event lists as far

forward as possible, i.e., until they need incoming events to process outgoing events, or until finished. However, if they receive an event with a timestamp t less then their current *local virtual time* (LVT) they must rollback their execution to this point t. A rollback effectively means to set the LVT to time t, reinstall the state just before this time t, and to "unsend" all messages sent during the time that has been rolled back with so called antimessages. Thereafter, the rolled backed node can start to re-execute all events from time t and forward. For the recovery old states as well as sent events must be kept. If no pruning of old states and sent events are done the memory consumption would be uncontrolled. To handle this a *global virtual time* (GVT) defines a common logical time such that every processed message with a timestamp less then GVT is considered stable, i.e., no recovery can go beyond the GVT. In Figure 2 all messages except m_5 arrive in numerical order. LP_1 will send m_3 and process m_4 before receiving m_5. The arrival of m_5 makes the speculative execution on LP_1 invalidated. The state on LP_1, therefore, must be brought back to just before the sending of m_3, which must be un-sent. Then the execution can start over at LP_1 with processing of m_5, re-sending m_3, and re-executing m_4.

Due to possible unbounded rollbacks TW is not suitable for real-time simulations. However, a restricted form of TW called No False Timestamps (NFT) Time Warp, was defined by (Ghosh et al. 1993) to provide a TW variant suitable for real-time simulations. NFT takes in to account overhead such as

state saving, state restoration, sending and receiving messages and antimessages, and can give an upper bound on the execution of a TW simulation given that no false events occur. A *false event* is an event that will be rolled back or canceled. If a simulation can be guaranteed despite it's rollback overhead R to meet all deadlines it is called *R-schedulable*. The R-schedule is generated by adding the overhead to all events in the simulation, i.e., the execution time of each individual event is increased by R. Unfortunately the class of simulations that can conform to the requirements of NFT has been showed to be very limited and of little practical use by Ghosh et al. (1994).

In (Ghosh et al. 1994), Ghosh et al. show that optimistic simulation synchronization protocols that never send incorrect messages (also known as aggressive no-risk simulations (ANR)) together with continuous generation of GVT provides a predictable way to execute optimistic real-time simulation. However, the continuous generation of GVT used in this protocol relies on nodes communicating through a shared memory and it is not useable when nodes only are connected through a network. That is, Ghosh et al. (1994) claim that this protocol is suitable for parallel systems, but not for distributed systems.

A distributed simulation where all nodes progress in unison and one node synchronizes with wall clock time is an example of a conservative way to achieve a discrete event real-time simulation. For example, in figure 3 feder-

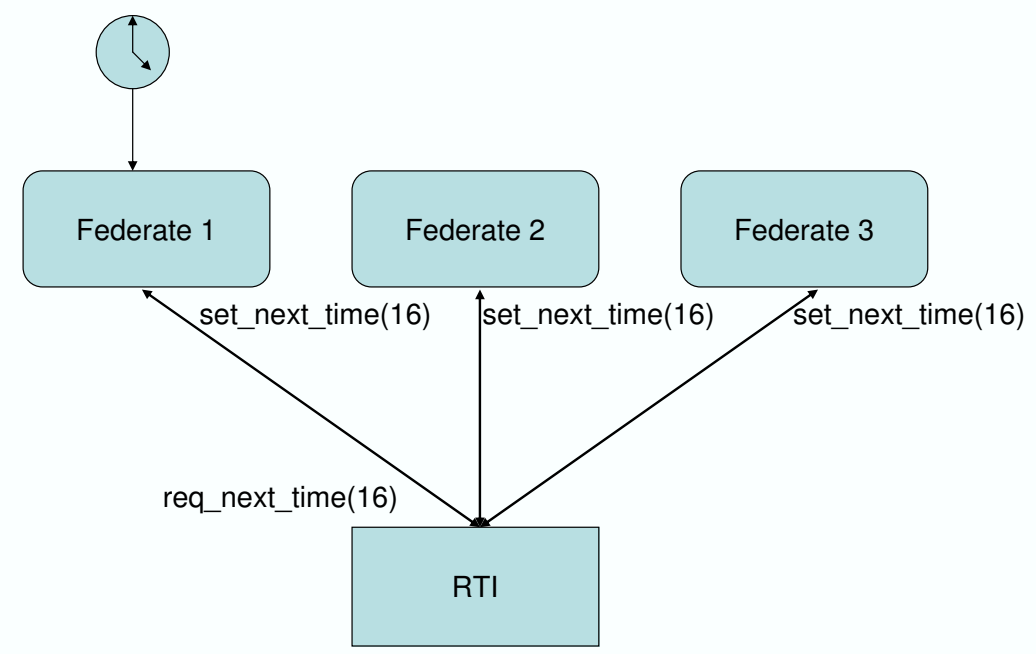

Figure 3: An HLA federation run in conservative "lockstep" execution.

ate 1 requests the time to 16 and the RTI then pushes the new time to all federates. For example, using HLA's time management a federation can be synchronized by using one federate to read a connected local wall clock time and asking the RTI to progress accordingly. However, HLA simulations do not consider node crashes or network partitions, and since there is no explicit fault tolerance specified in HLA that deals with network partitions or node failures some other measures are needed for this type of simulations to be robust.

3 Problem and Motivation

Before stating the problem we need to define *real-time simulation categories* and *degree of fault tolerance*, and discuss how fault tolerance can be introduced into real-time simulation.

3.1 Real-time simulation categories

Here we divide real-time simulations into two categories. The categories differ in how recovery can be done.

The first category of simulations, *real-time simulations without external actions*, are those that must keep track of time and make sure that entities in the simulation behave correctly to timing constraints, i.e., that they keep deadlines. A specific property of these simulations is that they only need to

satisfy the timing constraints when running and they have no dependencies in the real world. If the simulation starts to miss deadlines due to some unexpected increase in CPU load, e.g., garbage collection, one recovery approach is to simply freeze the entire simulation until the load goes back to normal and then continue simulation execution. Environment simulators belong to this category of simulations.

The second category of simulations, *real-time simulations with external actions*, are those that must pace according to wall clock time and keep real-time constraints at all times. These simulations, which are also known as hardware-in-the-loop simulations, have dependencies in the real world and actions taken in the real world may not be possible to undo. A drilled hole or a fired missile are examples of actions that are not easily undone in the real world. The reminder of this document will focus on a fault tolerant infrastructure, including a recovery mechanism, for this category of simulations.

Scaling the simulation time can be useful for simulations. For instance, if we could run every part of a simulation twice as fast as the wall clock time without violating any real-time constraints (e.g., deadlines) then we could do more simulation in less time. Or, if the current hardware is not fast enough to do all the simulation calculations needed in wall clock time the entire simulation could be scaled down by a factor. Both non real-time simulations and real-time simulations without external actions can make use of scaling

the wall clock time. However, scaling a real-time simulation with external events is not possible, since it would require scaling of (external) real world entities.

3.2 Degree of fault tolerance

We choose to define three degrees of fault tolerance for simulations: fault-masking, bounded recovery, and best-effort recovery.

Fault-masking is the highest degree of fault tolerance. The actual down-time due to a failure is zero; failures are masked away. However, this type of fault tolerance is also the most costly (in terms of hardware and software) since it requires redundant hardware and software at all times. Also, to keep replicas consistent there is a need for replica determinism (Powell, Verssimo, Rodrigues & Rufino 1991), which itself is non-trivial problem (Poledna, Burns, Wellings & Barrett 2000).

Bounded recovery is the second highest degree of fault tolerance. The time to detect and recover from a failure has a bound. If the sum of the bounded detection time d and the recovery time r is smaller then time allowed for recovery, this is a less resource intensive fault tolerance degree than fault-masking. Depending the time allowed for recovery, different approaches to recovery can be used. For example, if the recovery time needs to be short, a warm standby replica could be used. A warm standby replica is processing

all the requests that the primary replica receives, but is never used for communication with clients. In this way both replicas will have the same state at all times. As soon as an error is detected, communication can immediately be switched over to the warm standby. The switch over can be done with a very small r time.

Best-effort recovery is the lowest degree of recovery (apart from not having recovery at all). No specific deadline guarantees on how long the system will take to recover is given. However, the value of a recovery is better then doing a full restart of the simulation. The value can be in processor time or some other resource.

3.3 Fault tolerance in simulations

For real-time simulations without external actions checkpoint-based recovery is likely to be a useful approach, i.e., they can simply stop executing during recovery. However, rollback-based approaches cannot be used for real-time simulations with external actions, since rollbacks assume that all state changes can be undone or that there exists time to do a rollback. As stated previously this may not be true for real-time simulations with external actions, therefore, some other approach for recovery is necessary.

If only a part of a distributed simulation is considered to be a real-time simulation with external actions, we might not want to use the relatively

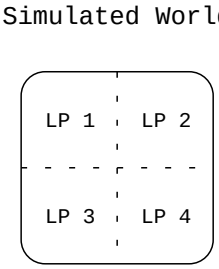

Figure 4: Four logical processes (LP) simulating one part each of "the world".

expensive active replication on the entire simulation. In this type of scenario we would like to use a hybrid approach that combined active replication for the highly dependable parts and some less costly model for the rest of the simulation, e.g., Leader-Follower or bounded time rollback recovery.

Another approach to fault tolerance in simulation is to look at the simulation model. For example, n simulation processes can simulate n different regions of a simulated world (See figure 4). If one of the simulation processes fails (e.g., crashes) only it's region would cease to be updated. However, entities that move between different regions must be handled in some way. One common way of solving the problem of having two simulation processes from different regions doing updates to an entity in transit from one region to another is to incorporate ownership of entities. The simulation process that owns an entity is the only one allowed to update the object. However, other simulation processes can ask the owner of an entity for an update. When an entity moves from one region to another passing of ownership by some

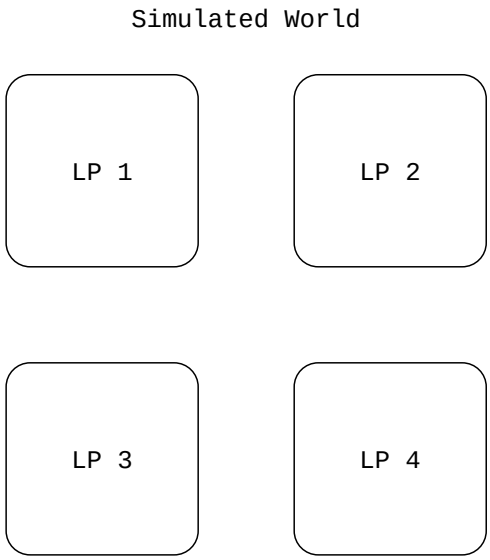

Figure 5: Four LPs simulating "the world".

hand shake protocol is one way of changing updater. In addition, creation of regions is itself a difficult task, .i.e., knowing an optimal (or at least sub-optimal) partitioning. Also, as previously mentioned having regions that are not updated for some time due to recovery may not be acceptable.

On the other hand, one important benefit with this approach is that it scales. If regions are created with respect to anticipated load requirements it gives an evenly distributed total load. Dynamic load balancing can be incorporated by passing object responsibilities from highly loaded nodes to less loaded nodes.

An alternative fault tolerance approach in simulations is to have n repli-

cas of the simulation process (See figure 5). All replicas have there own simulation state and they receive control commands from some common source (this is also known as replication of the simulation model (Knop & Sunderam 1994)). For this approach to work all correct simulation replicas must i) receive the same requests and ii) agree on the order in which to execute simulation requests. This is known as replica coordination or replica agreement. The calculation of simulation states must also be deterministic, i.e., if there are cases where randomness is required they must be based on pseudo random numbers and all replicas must have the same random number seed. This replication approach consume a lot of resources and must be used with care.

To allow bounded recovery, simulators must have the ability to restart simulations in any state, i.e., the simulation must allow to start from arbitrary or designated states. Either a simulation can be started at a specified state or the recovering simulation can be run at a higher pace then real-time in order to eventually catch up with the correct replicas.

4 Problem Statement

Real-time simulations with external actions, which are hard (or impossible) to undo, must have a degree of fault tolerance higher then best effort. No distributed-simulation infrastructures have a built-in fault tolerance degree

higher then best effort. Major distributed simulation infrastructures such as HLA (Dahmann et al. 1997) and DIS(DIS Steering Committee 1994) as well as other approaches that address fault tolerance in simulations do not deal with real-time issues (e.g., (Damani & Garg 1998)), i.e., they only have best-effort degree of fault tolerance.

As previously said, normally the protection against complete restarts in distributed simulations today is to rollback the simulation to a known globally consistent state, often called a checkpoint, or to extrapolate a likely future state. These recovery processes can require unbounded resources (e.g., computation time or memory) although work on bounding the state recovery in interactive soft real-time simulations has been done (Gomes, Unger, Cleary & Franks 1997). For real-time simulations with external actions no such work has been done and since crashed nodes must become operational within some predefined maximum time due to the real-time constraints we need some way of achieving fault tolerance in these types of real-time simulations. Lüthi & Berchtold (2000) also point out that the major concern when moving from central to distributed simulation has been on performance, i.e., improve how the simulation scale. This has often resulted in design where simulation nodes do not fail independently, e.g., in HLA simulations the failure of a single federate often leads to failure of the entire federation. In addition, disconnected nodes due to network failure usually cannot progress separately, i.e., work in autonomy. In HLA, for example, no federate can

progress when the connection to the run-time infrastructure (RTI) is lost, since it is the RTI that paces the entire federation. This can also lead to unbounded delay of the entire simulation.

Simulation infrastructure standards such as HLA and DIS have no built-in support for fault tolerance suitable for real-time simulations with external actions. The only way to avoid complete restart of a crashed simulation is to make use of checkpoints that are unbounded in terms of memory and time. In addition, use of checkpointing in for example HLA is up to each participating federate and not mandatory. Also, even though HLA simplifies distribution and reuse (Dahmann et al. 1997) the architecture has a major flaw when viewed from a fault tolerance perspective, namely the RTI. The RTI is a central unit through which all simulation communication is conducted. This means that if the node running the RTI is unavailable (e.g., due to a crash or network failure) the entire simulation stops.

In the area of information fusion there is often need for real-time simulations with external actions.

4.1 Aim

The aim of the thesis work is to develop a fault tolerant infrastructure for real-time simulations with external actions, i.e., where parts of the simulation state cannot be undone or the recovery time is very limited. That is, the recovery process must have a bound on resources such as CPU time and

memory usage. Moreover, a guideline to which replication strategy to use in the infrastructure depending on the type of real-time requirements that exists will be developed.

Technical challenges in this work include finding a platform to base the infrastructure on, designing a synchronization protocol for distributed discrete simulations that can be used in real-time simulation with external actions, and to define guidelines on how to choose replication strategies depending on the requirements of the real-time simulations.

The optimistic simulation synchronization protocol designed by Ghosh et al. (1994) uses a shared memory architecture when calculating GVT and communication over a network they claim to be too slow and not capable to keep real-time deadlines. One part of this project has investigated if a DARTDB architecture that provides a similar memory sharing facility indeed can keep the deadlines imposed by a continuously updated GVT.

In addition, any simulation synchronization protocol used in the infrastructure must ensure fault tolerance. Here the use of a DARTDB is beneficial since transactions encapsulates all changes to the simulation state and therefore gives (backward and forward) recovery a good starting point. We want to investigate how the two higher fault tolerance degrees (see 3.2) can be incorporated into our simulation synchronization protocol and compare the

protocol with leading simulation infrastructures such as HLA.

4.2 Objectives

1. Define a fault tolerant distributed real-time simulation infrastructure for simulations with external actions. A number of steps must be taken for this to be done.

    - Finding a suitable platform to base the infrastructure on, i.e., what properties are necessary in a infrastructure for real-time simulations with external actions
    - Defining a simulation synchronization protocol that is capable of working with a fault tolerance degree higher the best effort degree.

2. Define a number of real-time simulation scenarios that have external actions in the real world to be used for comparisons between our proposed infrastructure and, for example, HLA or DIS. Evaluate the features needed by the scenarios to see if they conform to the properties found when defining the infrastructure requirements.

3. Run experiments on both a common distributed simulation platform (e.g., HLA) and our infrastructure, to make performance comparisons for example on recovery and simulation overhead. Also, comparisons between different combinations of replication policies to gain deeper

understanding on replication policies influence on each other in real-time simulations with external actions.

5 Simulation DeeDS: A fault tolerant infrastructure for distributed real-time simulation with external actions

We believe that a DARTDB is a suitable platform to base our infrastructure on. The basic concept of having a DARTDB as an infrastructure for distributed real-time simulation has been presented in two papers . In the work presented, a proof-of-concept implementation of our simulation infrastructure, which we now introduce as *Simulation DeeDS*, based on an in-house developed research prototype DARTDB called DeeDS (Andler, Hansson, Eriksson, Mellin, Berndtsson & Eftring 1996) has been conducted. In addition, we show how such architecture can support the requirements imposed by information fusion applications such as the precision agriculture example that need real-time simulation with external actions.

First we show how the requirements described in 2.1 can be met by a DARTDB and continue to show how this can be done in particular with the DeeDS database.

5.1 Supporting Information Fusion Applications

5.1.1 Heterogeneity

A DARTDB could be implemented such that heterogeneity is handled. For example, differences in data types could be addressed by transforming everything stored in the database to a globally defined data structure. This conversion, which should be a part of the information fusion process, should be done without human intervention. One way to make this process automatic is to use active functionality. In addition, the DARTDB must either be implemented on top of a platform independent middleware (e.g., Java VM) or be ported to all the platforms used by nodes in the particular information fusion system.

5.1.2 Distribution

In addition to distributed processing, distribution of node in the information fusion process provides processing close to information sources. Something that is useful in real-time systems, since for example communication latencies are minimized. Since DARTDBs are inherently distributed they could easily be configured to have one node at each information source if needed. For example, sensor data can be collected, trimmed and converted to a globally defined structure before it is sent to other parts in the fusion process.

In a distributed architecture it is also possible to use replicas to increases

fault tolerance. For example, by using semi-active replication such as Leader-Follower (David 1991), data from important information sources can be made fault tolerant. Fault tolerance is particularly interesting for information fusion applications where the fusion process itself must not fail, i.e., the use of fused data is critical. For example, if a decision support system must deliver a suggestion in time for some human (or computer) operator to make a well founded decision. By failing to meet a deadline, e.g., the suggestion was not delivered in time, some severe penalty is issued (financial or some other valuables).

5.1.3 Independence

To achieve independence between producers and consumers of information there is a need for an intermediate part in the fusion process. A database provides a robust way to store information and decouples readers and writers, i.e., by storing and retrieving information in a database producers and consumers are kept independent of each other.

5.1.4 Scalability

Processing of information can be done at any node in a DARTDBS. First, information can be processed directly on the node where data enters the information fusion process, e.g., where sensor data is collection. Second, filtering of data at consumer nodes is also possible, for example, receive only one third of the sensor updates for a specific sensor. Lastly, reformatting or

merging of information can be done by the producer, consumer, or an intermediate node, e.g., combine two independent sensors to get an improved precision on a real world object. Reformatting of data allows information to fit a predefined structure suitable for a stakeholder. For example, displaying sensor readings on a hand held device can be done differently compared to a large desktop display. Merging information means that data from information sources are brought together to form new, fused information with improved value as compared to using the values separately. For example, consider bringing together a radar view, camera view, and infrared camera view into a fused view suitable for a command center.

Finally, a DARTDBS represents a design that can be made to scale if care is take in the design. By adding more processing power, i.e., nodes, when adding producers or consumers the architecture can exploit parallelism in the information fusion process. Also, additional intermediate nodes for merge processing can be added to off-load heavy processing nodes. For example, if node a produced the above mentioned command center view and also collected a number of sensor readings this task could be shared between the two nodes a and b to reduce a's load. However, design decisions such how to store data, i.e., whether the database sould be partitioned, partial, or fully replicated must be decided.

5.1.5 Adaptivity

Active functionality, which is incorporated in DARTDBS, is a core feature of our proposed infrastructure. Based on active rules or triggers in the database this is what makes automatic processing at the different database nodes (producer, consumer, intermediate) possible. Filtering is one type of processing suitable for triggers. For example, allow position updates from cars to enter the database every 400 ms, even if there exists more frequent updates from the sensors. Another example on active behavior is to deliver an alarm if a value in the database exceeds some threshold. In addition, the system can through the triggers react to more complex event combinations such as sequences of events or non-occurrences of events. Finally, triggers can be constructed such that they react on changes from both information sources and processes that use the fused information.

5.1.6 Temporal properties

Temporal properties can be supported directly by DARTDBs by allowing time series to be represented in the database schema. Predictions on future behavior, e.g., event occurrences, or trends, can be included by allowing information sources to be simulators. In this way historical data can be retrieved from the database, as well as current data from sensors, along with predictions generated from simulations.

5.2 DeeDS: A DARTDB prototype

The DeeDS database (Andler et al. 1996) is an in-house developed DARTDB prototype. In this section we discuss how DeeDS can address the requirements described in section 2.1 and how particular issues listed such as data storage model and replication are addressed.

The heterogeneity requirement is partly met by DeeDS through a generic operating systems interface called DeeDS Operating systems Interface (DOI) that provides a simple way to port DeeDS to different platforms, combined with simple marshalling and un-marshalling functions for adapting replicated data to the different platforms. A more general way to support heterogeneity is to replace the communication part of DOI with real-time CORBA (e.g., the TAO (Schmidt, Levine & Mungee 1999) implementation). Another alternative could be to use real-time Java and remote method invocation (RMI).

The need for distribution and independence in information fusion processes are met by the DeeDS database. DeeDS is a distributed database and as such it can have database nodes at distributed locations; information sources can have database nodes locally. Moreover, DeeDS advocates a whiteboard architecture where all communication between applications is performed through the database operations. Bounded-time replication (Lundström 1997) of updates in the database make ensures that real-time requirement are not compromised by the replication protocol.

If all data is not available locally it may be required that other nodes are queried in order to complete a transaction. This could be a problem if the network is non-real-time or unreliable, since this endangers predicability of the database, i.e., real-time guarantees cannot be given. In addition, access times to data in main memory differs in orders of magnitude compared to accesses to secondary storage (i.e., disk). Since worst case access must be considered when designing a real-time database using disk storage as in traditional databases would lead to overly pessimistic worst case execution times (Stankovic, Son & Hansson 1999). Therefore, DeeDS store critical real-time data in main memory. However, since the amount of data in typical information fusion applications is huge it is not practical to have all data in main memory. By segmenting the database, critical data can be stored in segments available in main memory while other segments can be directed to secondary storage such as disk.

The transaction model in DeeDS assumes full replication; all data is replicated to all nodes. However, a database that is fully replicated does not scale (Gray, Helland, O'Neil & Shasha 1996). To resolve this, scalability in terms of memory usage and the number of replication messages needed in DeeDS is achieved by the use of *virtual full replication* through segmentation (Mathiason & F.Andler 2003). Virtual full replication means that every data object that applications on a certain node require is guaranteed to be acces-

sible locally on that node. In essence, no distributed queries to other nodes are required and therefore it appears to the applications as if the nodes is fully replicated.

DeeDS exploits virtual full replication and local commit to avoid unpredictably long timeouts during network partitions. This guarantees predictability when reading and writing to the database. By making sure that all data objects are replicated a sufficient level of fault tolerance can be achieved, with a lower memory consumption compared to a fully replicated database. Even though critical data resides in main memory, a diskless distributed recovery algorithm (Andler, Örn Leifsson & Mellin 1999) makes recovery possible.

Since all transactions commit locally and results are eventually propagated and integrated on all other replicas, the global database state in DeeDS is said to be eventually consistent (Gustavsson & Andler 2005). That is, given that no more transactions enter the system, the database will eventually converge to a globally consistent state. The relaxation of consistency potentially introduces conflicts. These conflicts must be i) detected and ii) resolved. DeeDS makes use of a modified variant of version vectors and log-filters to detect conflicts. Conflict resolution is application dependent and can, for example, be to always take the latest value, highes value, or the value originating from prioritized node.

Adaptivity is achieved by using active behavior in the form of event-

condition-action (ECA) rules in the database. The generation of both simple and complex events, as well as the actual monitoring of events can be done while consuming a predictable amount of resources (Mellin & Andler 2002).

5.3 Summary

To summarize, we argue that using DARTDB as infrastructure gives these key benefits:

- Full support for real-time simulation with external action, in particular we have shown that some information fusion applications are examples of this type of simulation and that it can use our proposed infrastructure.

- Transactional encapsulation of execution. This is useful for recovery, since all operations are contained within transactions and therefore can be rolled back or compensated for.

- Implicit storage of simulation data. Data from simulation runs are implicitly stored in the database during communication, i.e., no explicit action is required for storing.

- Location transparency of nodes. The use of virtual full replication make the database look like a central database from the simulation point of view, i.e., simulation engineers only store and retrieve simulation state

changes in the database. No specific action concerning communication with other nodes is required, as this is handled by the underlying replication protocol.

- Disconnected operation. A feature of the specific database chosen is that local commits are allowed. This means that network partitions does not necessarily jeopardize the predictability of the simulation.

Further work concerning the design of a synchronization protocol for distributed real-time simulations that need to be fault tolerant is needed. Two different approaches can be tried: a conservative or an optimistic synchronization protocol.

5.4 Simulation Synchronization

5.4.1 A fault tolerant conservative simulation synchronization protocol

In a distributed simulation that uses a conservative synchronization protocol nodes progresses in unison, i.e., fast nodes are blocked until all nodes are done executing a specific point in time. If one node would crash, the entire distributed simulation would halt until the crashed node has recovered or a stand-by has been summoned. Depending on the required recovery time different replication policies, e.g., active, hot stand-by, cold stand-by, can be

used. However, as the design of Simulation-DeeDS is based on an optimistic replication protocol we chose not to pursue this conservative approach.

5.4.2 A fault tolerant optimistic simulation synchronization protocol

Rather then blocking nodes that have good performance, optimistic synchronization try to utilize this and execute (speculative) further into the future. This means that nodes can progress independently, i.e., the entire simulation is less vulnerable to network partitions or single node failures. However, computation of GVT needs all participating nodes connected and running, i.e., this computation cannot be done if a node has crashed or there is a network partition. For real-time simulations with real-world actions this means that the simulation state may not be stable fast enough. If an execution of a external action is done that later is found to be incorrect the simulation must handle this.

5.4.3 Implementing an optimistic simulation synchronization protocol

A DARTDB as described in section 5.2 provides a shared memory architecture that can guarantee local hard real-time requirements. By putting the data structures used in TW in the database, i.e., each LP's LVT, message queues, and state as well as the GVT, we can provide a database version of TW.

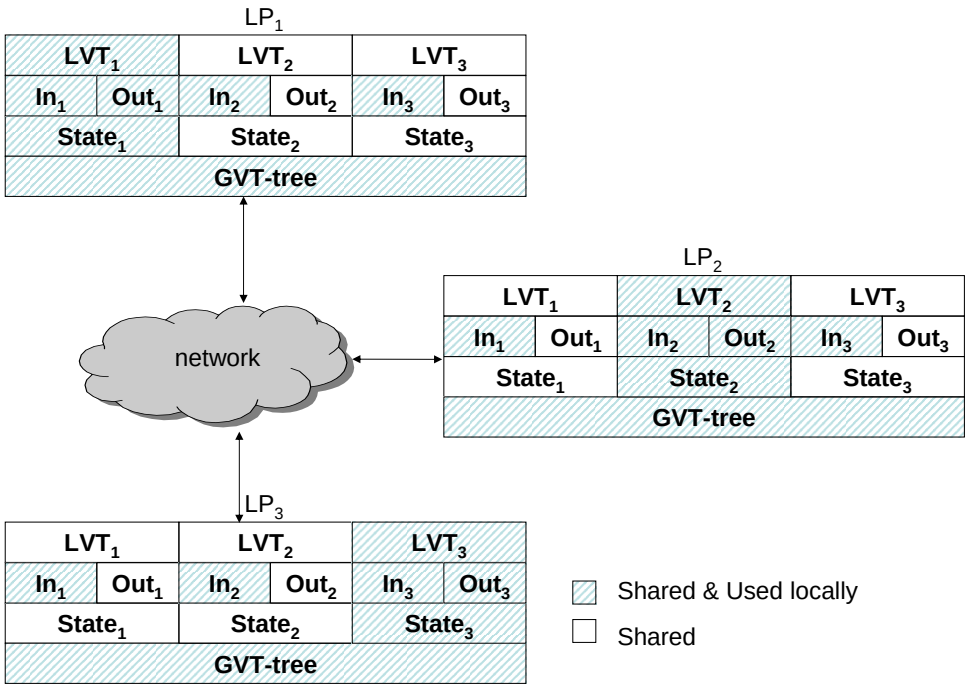

Figure 6: Data structures of TW in the database.

First, we define that each database node is also a database replica. The nodes are fully replicated, i.e., all information is available at all nodes. To ensure local real-time we define transactions to run locally and changes are propagated after commit to all other replicas. A replica is always locally consistent, but inconsistences between replicas can occur. The global state of the database, however, is said to converge to a globally consistent state if no more updating transactions enter the database. This variant of replication policy is called eventual consistency The replication (consisting of propagation and integration of updates) to other replicas can be bounded or unbounded depending on the network capability. For example, if the replicas are connected by a real-time network the replication can be bounded.

Second, each database node serves a LP. This means that each node will hold a local virtual time, and input and output queues for the corresponding LP. Shared between the LPs are the GVT, which is the lowest LVT among the LPs. Figure 6 illustrates a three node simulation. By using the tree structure defined in (Ghosh et al. 1994), where the LVTs of the LPs are leaf nodes and the GVT is the root node. The GVT can be calculated continuously and furthermore, since the database is active, we can specify a rule in the database to automatically update the GVT-tree. For example, the rule could look like this: ON $update(LP_i.LVT)$ IF $LP_i \neq root$ & $LP_i.LVT < LP_i.parent$ THEN $update(LVT_i.parent)$.

The basic operation of the database driven approach follows. Messages in input and output queues are tuples consisting of a time of occurrence (timestamp) and the action(s), e.g., m_1(15,x=1) means that x is set to 1 at LVT 15. The queues are sorted on time of occurrence with the lowest time first. When the simulation starts each LP's LVT is set to ∞ and the GVT is set to 0. The processing on each LP_i consists of 4 steps: 1) take the first message (m_{head}) in LP_i's input queue and if the $m_{head}.timestamp$ is less then LVT_i then we have found a straggler and need to do recovery and start over processing from the recovery point, else set $LVT_i = m_{head}.timestamp$ and perform the actions in m_{head} on LP_i's state. 2) After successfully processing a message we must send the result to all LPs that use the result. This is done by writing the result in LP_i's output queue and the input queue of $\forall LP_j \in LP_j$ uses LP_i's result. 3) Update the GVT by checking if LVT_i is less then LVT_{parent} in the GVT-tree. 4) Check if $GVT = \infty$ and if true end the simulation.

5.5 Improved Database Approach

The database approach in section 5.4.3 does not use many of the features in the DARTDB, it merely uses the database as a message communication system and a storage facility. By adapting the ANR TW to the database an improved database approach can be obtained. For example, memory usage can be reducing by removing the message queues and rely more on the adap-

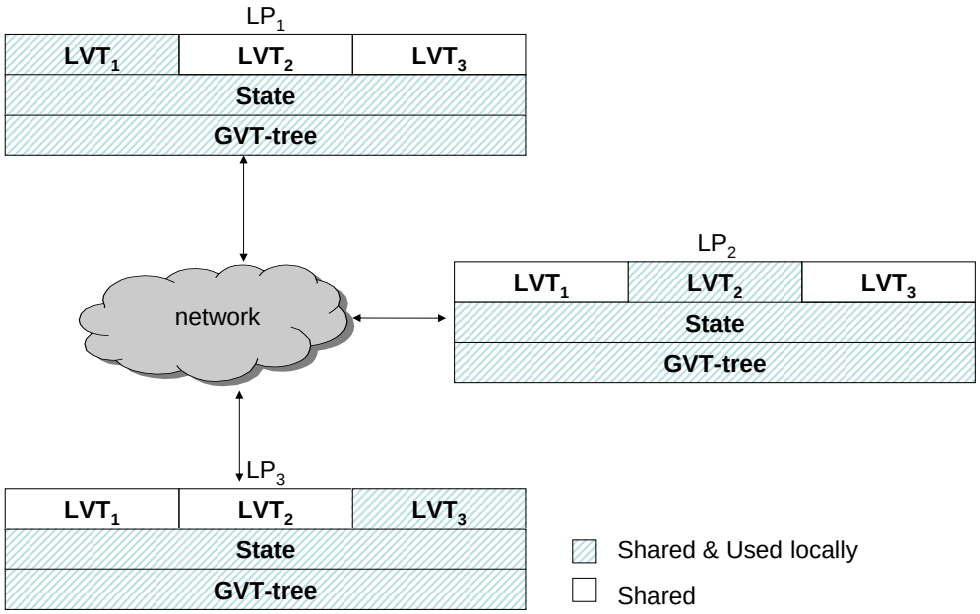

Figure 7: ANR TW in the database using active functionality.

tive functionality.

Due to the fact that the database is fully replicated and active, we do not need to send messages between LP by storing values in the respective input queues. Updates can be done directly on the state variables, and active rules can monitor all these updates. The state variables them selves need to store old values up till the GVT. This means that for a simulation with n LP instead of $2 * n$ message queues and n states we could use 1 state and

no message queues. The new state, however, would be a merge of the n states and each state variable would need to keep track of old values with $timestamps > GVT$. Figure 7 shows a tree LP that share state and have no input or output queues. The "messaging" is provided by the active functionality, which triggers the LP's to act when updates to state variables they use are detected.

In real-time systems all important tasks have deadlines and worst case execution times (wcet). For a task t with wcet t_{wcet} and deadline $t_{deadline}$ we say that the slack time for t is $t_{slack} = t_{deadline} - t_{wcet}$. Assuming that a task has a bounded recovery time $t_{recover}$ we can guarantee that the task t will finish before it's deadline iff $t_{slack} \geq t_{recover} + t_{wcet}$ (see figure 8). This is true under the assumption that a task fails at most once. Checkpoints can be used to avoid restarting a crashed task from the start. The use of checkpoints divides a task into smaller units that, once executed to completion, does not need to be reexecuted in case of a crash.

In figure 9, for example, the shaded part of task t does not have to be reexecuted even if a crash occurs in the later parts of the task. Instead the recovery kicks in and then the execution resumes at the checkpoint prior to the crash. The wcet for a task is then defined as $\sum_1^n wect\_part_i$. If there is a crash in $part_j$ then the following formula must hold: $\sum_1^j wect\_part_i + rt + \sum_j^n wect\_part_i \leq wect + slack$. Factoring leads to $rt + wcet\_part_j \leq slack$,

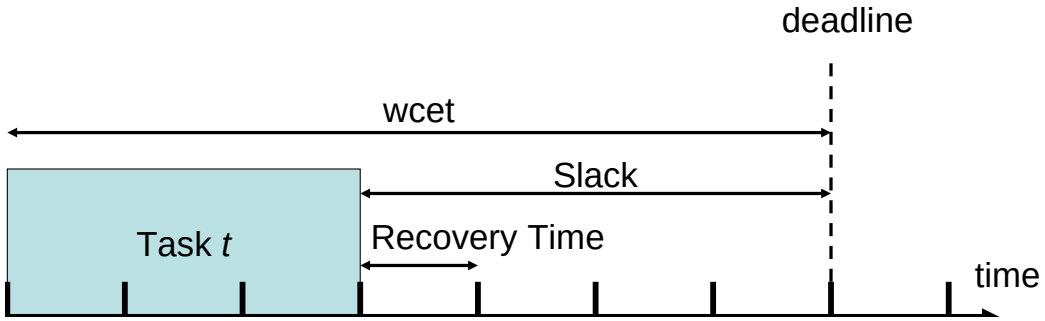

Figure 8: The slack time for a task t.

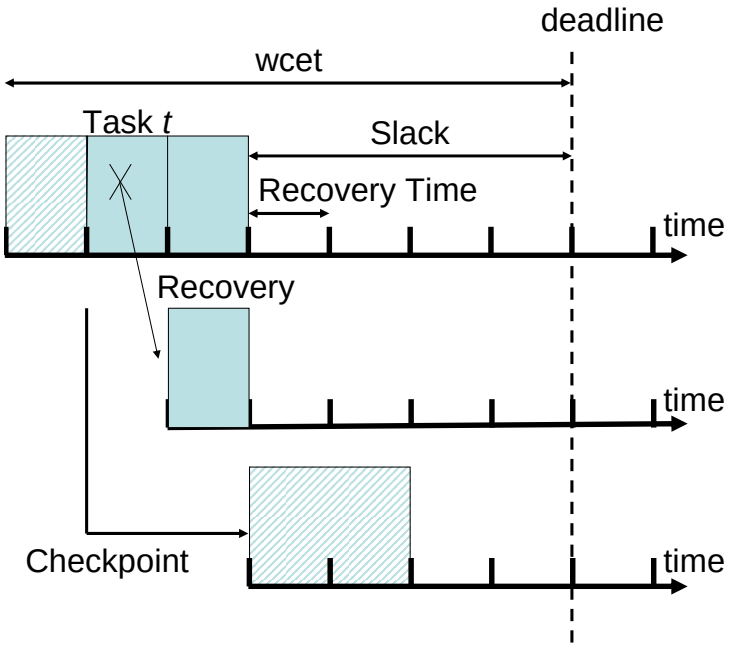

Figure 9: A task t using checkpoints.

i.e., the slack must be greater than the recovery time and the wcet for the crashed part.

A *recovery line* is a checkpoint taken at the same time in all participating nodes in a distributed system. In a distributed simulation, assume that we force a recovery line (just) before interaction with the real-world. Now, if any part of the simulation crashes it would rollback to the recovery line and then start to reexecute. If the next interaction with the real-world occur at

time t_{next}, we must make sure that the recovery time $t_{recover}$ is less or else the entity in the real-world cannot rely on our simulation. For example, if a simulation starts a drill in the real-world. This would require the creation of a recovery line. Than if the simulation crashes and fail to recover in time. The result could be that the drill produces a too deep hole. On the other hand if the detection and recovery can be guaranteed to be shorter than the next time to interact with the real-world, we can tolerate crashes in the simulation parts and still not drill too deep.

The major benefit in using our optimistic concurrency protocol is that real-time simulations with external actions can store all their state variable in a database and even though the simulation is distributed communication issues are hidden from the simulation engineers. Also, adding and removing nodes are potentially less cumbersome since all nodes are decoupled by the database, as opposed to for example distributed simulations with peer-to-peer connections.

As stated, the distributed database can be seen as white board or a shared memory. This simplifies the communication model for simulation engineers since, simulation nodes can communicating by writing and reading to this shared memory and all communication is catered for by the underlying replication mechanism. The database can give local real-time guarantees, since it allows local commits, is fully replicated and, main memory based. However, this design comes with a price: inconsistencies between nodes can exist even

though locally at each node the database is consistent. To bound how long time replicas can be inconsistent due to conflicting updates a replication policy called eventually consistent replication is used. This policy detects and resolves conflicts in the distributed database in a timely and predictable way and it guarantees that the database converges to a globally consistent state (Gustavsson & Andler 2002).

Scaling is another issue that must be addressed when using this database approach. Full replication means that every node in the database hold it's own copy of the data, i.e., every LP's LVT, state, along with the GVT-tree would exist on every database node. This does not scale, i.e., the memory consumption growth is linear with respect to LPs (database nodes) and the number of replication messages growth exponentially. For example, with three nodes it would require three times the storage compared to a single-node database. To allow local commits, however, transactions cannot rely on any data outside of the local node. This is why we need full replication. Transactions, on the other hand, only need the data they read or manipulate (write). This means that data, which never are accessed by transactions on a certain node, are not needed on that particular node for local commit reasons. By removing such superfluous data the important property of full replication, i.e., transactions never have to look elsewhere for data, is kept, but at a lower storage price. This feature is called virtual full replication (Andler et al. 1996). One important point still remains though, data are

replicated for fault-tolerance as well, and this must be considered when creating the database. One way to solve this issue is to segment the database. How to segment the database to achieve virtual full replication, as well as the expected decrease in storage requirements compared to regular full replication is ongoing research (Mathiason & F.Andler 2003).

In addition, using a DARTDBS allows implicit storage of simulation data, i.e., no special attention is needed to save simulation states or results. This can be compared with, for example, HLA where one common way to achieve storage of simulation data is to have one passive federate tap into the federation and collect all simulation data (Kuhl, Weatherly & Dahmann 1999).

6 Scenarios

Three different scenarios that we intend to use when comparing our simulation infrastructure and common off-the-shelf simulation infrastructures are the following: *unmanned rescue mission*, *agriculture optimization*, and *model train*. These scenarios are interesting since they provide examples of real-time simulations with external actions.

6.1 Unmanned rescue mission

One example of a long-running real-time simulation application is continued evaluation of agents used in artificial intelligence. These agents have the long-

running property, since they should evolve over time to make better choices. They often also interact with both simulators and real-world environment, which means that they must obey real-time restrictions. One such agent scenario is described in the WITAS project(Doherty, Granlund, Kuchcinski, Sandewall, Nordberg, Skarman & Wiklund 2000). In this project helicopters (agents), or unmanned aerial vehicles (UAVs), operate in the real-world and communicate with human operators through an operator's assistant. Due to aviation regulations and limited number of helicopters scenarios with more than one helicopter must be simulated. This, however, does not decrease the real-time requirements since some actions performed by the real helicopter cannot be undone, i.e., this becomes a real-time simulation with external actions.

So, consider a traffic accident where more then one helicopter is needed in the rescue operation. One car with up to four passengers have driven off the road and over a step slope. The slope makes it hard to find the car wreck and any survivors. A total of six helicopters are in the nearby area. Two of the helicopters carry heat sensor cameras, i.e., they can detect surviving humans. The remaining four helicopters have first-aid kits onboard. Once a first-aid kit is deployed it cannot be retrieved. Releasing the firs-aid kit can hence be seen as an action that cannot be undone. The only way to replace a dropped first-aid kit is to return to base and have a human operator manually load a new one. For this scenario this is considered to take too much time.

The two helicopters with heat sensing cameras try to find any survivors and when they do they write this to the the distributed database, i.e., informs the other helicopters. The four helicopters that are ready to deploy help wait for a rule to trigger. This rule is trigged when a survivor is found, i.e., when the position of a human in need is present in the database. Once a helicopter is under way to aid a human, it informs other potential helicopters by writing to the database, that the present human will receive help.

Due to the replication policy one or more helicopters can register that they have undertaken to help the same injured person. This conflict needs to be resolved or one person will receive more then one first-aid kit. There exists many ways to solve this conflict. The perhaps best is to have the helicopter closets to the target be the one to do the task. However, finding out which helicopter that is closest is not trivial, since it requires reading the database. By reading the database new, possibly conflicting, transactions can be introduced. A solution that take this into consideration would have a max distance that any helicopter can travel while a position update is inconsistent. If there still exists a helicopter that is closest this one will be selected for the task. If there are more then one helicopter that potentially are closest a predefined preference is used, e.g., if the helicopters are numbered, take the one with lowest number.

Another conflict that can be introduced is that the two searching helicopters can find the same person, i.e., two different reports of the same injured person. A classic problem for autonomous agents is the symbol-grounding problem, i.e., how to tell if two agents see the same or two distinct objects. For this scenario we do not consider this problem, i.e., the helicopters know if they have found the same human. This simplifies the conflict resolution. That is, the consistent database will only contain one person at a given location.

When running a simulation of this scenario different setups can be used. First, the real-world helicopter could have an heat sensor camera. Secondly, the helicopter could carry a first-aid kit.

6.2 Agriculture optimization

Precision Agriculture (PA) is an area of research within information fusion that aim to improve crop yields and limit environmental impact. Specifically, within the PA field one aim is to optimize fertilizing so that the different areas of a field receives the precise amount of fertilizer that can be used. Fertilizing can be improved based on the knowledge of a number of different factors such as the history of fertilizer deployed in previous years, current weather conditions, long-term weather forecasts, soil quality, resulting crop yields from previous years. These factors together with models on how the various factors influence each other help determine the amount of fertilizer to use on a specific part of the field. Some of the needed calculations can be

done off-line but not all, e.g., measuring the soil quality and amount of water in the soil must be done while the tractor is running and deploying fertilizer. Simulations that try to predict the resulting crop based on all the factors mentioned above need to generate an answer before the tractor has moved away from the measured soil, i.e., the simulation need to keep deadlines and therefore is a real-time simulation. Moreover, the action, deploy fertilizer, taken by the tractor cannot be undone, which means that this is in fact an example of a real-time simulation with external actions.

6.3 Controlled lab experiment

Two controlled lab experiments are currently being implemented. In the first, we use a computer controlled model train where some part of the track is simulated. Also, real and simulated trains will run together in this experiment. In the second, a few motes use in sensor fusion will be connected with simulated counterparts. The aim with these experiments, apart from being a proof-of-concept implementation of the Simulation DeeDS, is to give us performance measures for example on the number of simulation objects that can be handled, or the number of real and simulated nodes that can be combined. For the individual experiments we intend to do comparison with a number of different node setups as well as on the number of simulation objects.

7 Contributions

The contributions given by this work will include an infrastructure design along with two new synchronization protocols for distributed real-time simulations with external actions, guidelines on how to achieve fault tolerance for real-time simulations with external actions, and a proof-of-concept implementation to benchmark the performance of our infrastructure.

- We have published an infrastructure, Simulation DeeDS, that supports real-time simulations with external actions and a proof-of-concept has been implemented.
- We have published an optimistic simulation synchronization protocol that is capable to run on our Simulation DeeDS infrastructure.
- We have shown that information fusion applications that require real-time simulation can use Simulation DeeDS.

In the near future a few projects will be carried out:

- We will publish the results from two proof-of-concept experiments that further strengthens our claims that Simulation DeeDS is suitable for real-time simulations with external actions. Also, these experiments will provide performance measures.
- We will begin work on the guidelines concerning choices of replication policies to use depending on the real-time requirements of the simulation.

8 Related work

Ghosh et al. (1994) has developed a prototype infrastructure called PORTS (Parallel Optimistic Real-Time Simulation) that use an optimistic synchronization protocol together with continuous calculation of GVT. PORTS is said to be tightly coupled systems, i.e., distributed simulations that are connected through a network is not suitable.

Wang, Turner, Low & Gan (2004) describes an optimistic synchronization architecture for HLA simulations. It fails to address fault tolerance, i.e., the RTI is still a single point of failure. The architecture provides a way for implicit state saving to simplify for simulation engineers, i.e., they should not need to think of state saving and doing rollbacks.

Goldsman & Withers (1990) describe a way to run replicas with different parameters in order to foresee how a production cell reacts to different control settings. In there approach, rare events with high impact on the simulation, e.g., failure of an important machine, are removed. They argue that occurrence of high impact events probably changes control settings radically and therefore should be treated as input parameters.

Knop & Sunderam (1994) describe a parallel software system called ACES that supports heterogenous network based cluster computing. In particular, with a toolkit called EcliPSe, which is used as an upper layer in this ACES

architecture, target replication-based simulations. They use a checkpoint-rollback mechanism that periodically saves data. However, it is unclear if data is saved to one or several nodes. Also, there is no mentioning of supporting real-time requirements.

Replication in time warp has been done before (Agrawal & Agre 1992), but the replication is done purely on LP level. This means that the entire state of an LP must be replicated along with the actual LP (called object in this work). They are replicated primarily for performance, but also for fault tolerance. The replication protocol assumes a static setup of replicas for each object. There exists two types of messages that can be sent: read and write messages. In case of a read message, the message is sent to the closest (by some metric) replica. The access information is taken from the static object replica information. If a message of write type is sent, then all replicas will receive this message, but any resulting output provided by processing this message is only outputted by a replica determined (in the message) as primary. This guarantees that any message will only generate one output. The replication protocol is a 'read any write all' protocol. Fail-stop replicas are assumed and future work includes ability to handel Byzantine failures. The actual recovery of replicas is not discussed, i.e., they focus on performance increase. Especially in the case where there are few writers and many readers of data.

References

Agrawal, D. & Agre, J. R. (1992), Replicated objects in time warp simulations, *in* J. J. Swain, D. Goldsman, R. C. Crain & J. R. Wilson, eds, '1992 Winter Simulation Conference', pp. 657–664.

Andler, S. F., Örn Leifsson, E. & Mellin, J. (1999), Diskless real-time database recovery in distributed systems., *in* 'Work in Progress at Real-Time Systems Symposium (RTSS'99)'.

Andler, S., Hansson, J., Eriksson, J., Mellin, J., Berndtsson, M. & Eftring, B. (1996), 'DeeDS towards a distributed and active real-time database system', *ACM SIGMOD Record* **25**(1), 38–40.

Brohede, M. & Andler, S. F. (2002), Distributed Simulation Communication through an Active Real-Time Database, *in* '27th Annual NASA Goddard Software Engineering Workshop (IEEE/NASA SEW-27)', IEEE, Greenbelt, MD, pp. 147–155. ISBN 0-7695-1855-9.

Brohede, M. & Andler, S. F. (2003), Distributed Real-Time Database support for HLA (03E-SIW-009), *in* 'European Simulation Interoperability Workshop', Stockholm, Sweden. CD-ROM.

Brohede, M. & Andler, S. F. (2005), Using Distributed Active Real-Time Database Functionality in Information-Fusion Infrastructures, *in* 'Real-Time in Sweden 2005 (RTiS2005)', Skovde, Sweden.

Brohede, M., Andler, S. F. & Son, S. H. (2005), Optimistic Database-Driven Distributed Real-Time Simulation (05F-SIW-031), *in* 'Fall 2005 Simulation Interoperability Workshop (FallSIW 2005)', Orlando, FL, USA.

Dahmann, J. S., Fujimoto, R. M. & Weatherly, R. M. (1997), The Department of Defense High Level Architecture, *in* 'Proceedings of 1997 Winter Simulation Conference', Atalanta, GA, pp. 142–149.

Damani, O. P. & Garg, V. K. (1998), Fault-tolerant distributed simulation, *in* 'Proceedings of 12th Workshop on Parallel and Distributed Simulation'.

David, P., ed. (1991), *DELTA-4 - A Generic Architecture for Dependable Distributed Computing*, Springer Verlag.

DIS Steering Committee (1994), The DIS Vision, A Map to the future of distributed simulation, Technical Report IST-SP-94-01, Institute for Simulation and Training, Orlando, FL. Version 1.

Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E. & Wiklund, J. (2000), The WITAS Unmanned Aerial Vehicle Project, *in* W. Horn, ed., 'Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-00)', IOS Press, pp. 747–755.

Fujimoto, R. M. (1990), 'Parallel discrete event simulation', *Communications of the ACM* **33**(10), 30–53.

Ghosh, K., Fujimoto, R. M. & Schwan, K. (1993), Time warp simulation in time constrained systems, *in* 'Proceedings of the seventh workshop on Parallel and distributed simulation', ACM Press, pp. 163–166.

Ghosh, K., Panesar, K., Fujimoto, R. M. & Schwan, K. (1994), Ports: A parallel, optimistic, real-time simulator, *in* 'Parallel and Distributed Simulation (PADS'94)', ACM.

Goldsman, D. & Withers, D. (1990), Single Replication Simulation, *in* O. Balci, R. P. Sadowski & R. E. Nance, eds, 'Winter Simulation Conference', ACM, pp. 387–391.

Gomes, F., Unger, B., Cleary, J. & Franks, S. (1997), Multiplexed state saving for bounded rollback, *in* 'Proceedings of 1997 Winter Simulation Conference', Atalanta, GA, pp. 460–467.

Gray, J., Helland, P., O'Neil, P. & Shasha, D. (1996), 'The dangers of replication and a solution', *SIGMOD Record* **25**(2), 173–182.

Gustavsson, S. & Andler, S. F. (2002), Self-stabilization and eventual consistency in replicated real-time databases, *in* 'workshop on self-healing systems (WOSS'02)', Charleston, SC, USA.

Gustavsson, S. & Andler, S. F. (2005), Continuous consistency management in distributed real-time databases with multiple writers of replicated data, *in* 'Proc. of the 13th Int'l Workshop on Parallel and Distributed Real-Time Systems 2005 (WPDRTS'05)', Denver, CO, USA.

Jefferson, D. R. (1985), 'Virtual time.', *ACM Transactions on Programming Languages and Systems* **7**(3), 404–425.

Knop, F. & Sunderam, V. S. (1994), An introduction to fault tolerant parallel simulation with EcliPSe, *in* J. D. Tew, S. Manivannan, D. A. Sadowski & A. F. Seila, eds, '1994 Winter Simulation Conference', ACM, pp. 700–707.

Kuhl, F., Weatherly, R. & Dahmann, J. (1999), *Creating computer simulation systems: an introduction to the high level architecture*, Prentice Hall PTR.

Lamport, L., Shostak, R. & Pease, M. (1982), 'The Byzantine Generals Problem', *ACM Transactions on Programming Languages and Systems* **4**(3), 382–401.

Lundström, J. (1997), A conflict detection and resolution mechanism for bounded-delay replication, Master's thesis, University of Skövde.

Lüthi, J. & Berchtold, C. (2000), Concepts for Dependable Distributed Discrete Event Simulation, *in* R. V. Landeghem, ed., 'Proceedings of the Int. European Simulation Multi-Conference', pp. 59–66.

Mathiason, G. & F.Andler, S. (2003), Virtual full replication: Achieving scalability in distributed real-time main-memory systems, *in* 'Proc. of the Work-in-Progress Session of the 15th Euromicro Conf. on Real-Time Systems', Porto, Portugal, pp. 33–36.

Mellin, J. & Andler, S. F. (2002), A formalized schema for event composition., *in* '8th Int'l Conf on Real-Time Computing Systems and Applications (RTCSA 2002)', Tokyo, Japan, pp. 201–210.

Misra, J. (1986), 'Distributed discrete-event simulation', *Computing Surveys* **18**(1), 39–65.

Poledna, S., Burns, A., Wellings, A. & Barrett, P. (2000), 'Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems', *IEEE Transactions on Computers* **49**(2), 100–111.

Powell, D., Verssimo, P., Rodrigues, L. & Rufino, J. (1991), *DELTA-4 - A Generic Architecture for Dependable Distributed Computing*, ESPRIT Research Reports, Springer Verlag.

Reynolds, P. F. (1988), A spectrum of options for parallel simulation, *in* M. Abrams, P. Haigh, & J. Comfort, eds, 'Winter Simulation Conference'.

Schmidt, D. C., Levine, D. L. & Mungee, S. (1999), 'The Design of the TAO Real-Time Object Request Broker', *Computer Communications, Elsivier Science* **1**(4).

Stankovic, J. A., Son, S. H. & Hansson, J. (1999), 'Misconceptions about real-time databases.', *IEEE Computer* .

Wang, X., Turner, S. J., Low, M. Y. H. & Gan, B. P. (2004), Optimistic synchronization in hla based distributed simulation, *in* 'Parallel and Distributed Simulation (PADS'04)', IEEE Computer Society, ACM.