УДК 004.75

ПРЕИМУЩЕСТВА И ОСОБЕННОСТИ РЕАЛИЗАЦИИ РАЗДЕЛЬНОГО СОХРАНЕНИЯ СОСТОЯНИЙ И ЛЕНИВОГО ОТКАТА ДЛЯ ОПТИМИЗАЦИИ ОПТИМИСТИЧЕСКОГО ПРОТОКОЛА СИНХРОНИЗАЦИИ

Попов Ю.В., Воротникова Ю.А.
Донецкий национальный технический университет, Украина

Рассмотрены недостатки использования централизованного стека состояний при оптимистическом протоколе синхронизации в распределенном моделировании. Предложен способ раздельного хранения стека состояний для каждого элемента схемы, что позволяет не удалять состояния, которые не должны быть затронуты процедурой отката. Рассмотрены особенности реализации раздельного сохранения состояний.

Введение

Активное развитие техники и электроники, усложнение и увеличение ее интеллектуальности приводят к росту и усложнению топологии управляющих цифровых схем. Для их тестирования и моделирования работы требуются специализированные системы и алгоритмы. Одним из наиболее перспективных и развитых является распределенное компьютерное моделирование, в котором преимущественно используется оптимистический протокол синхронизации.

Предложено большое количество способов и алгоритмов [1], осуществляющих распределенное моделирование – различные техники отмены неправильных вычислений при оптимистическом протоколе синхронизации, алгоритмы ограничения оптимизма вычислений, вероятностные протоколы сохранения состояний. Однако все они имеют недостатки, связанные со скоростью работы, количеством избыточных перевычислений и количеством потребляемой памяти.

1 Описание основных структур данных, используемых при распределенном логическом моделировании

Для осуществления распределенного моделирования используется ряд сущностей, каждая из которых имеет свою функцию. Наиболее важной является структурно-функциональная модель схемы (СФСМ), которая отвечает за хранение элементов схемы, их функций, значений, связей между ними, а также значений на входных и выходных узлах элементов. Также существуют очереди входных (IQ) и выходных событий (OQ) и список внешних событий (EVL), представленные на рис.1. Обеспечение возможности отката, который неизбежно возникает при оптимистическом протоколе синхронизации, ложится на стек состояний (SS) (рис. 1). Эта структура представляет собой динамический список, который хранит информацию о каждом изменении состояния на каждом из элементов схемы в каждый момент виртуального времени (LVT). Также стек состояний хранит ссылки на список внешних событий, который меняется с продвижением виртуального времени путем исключения из него обработанных событий и добавления

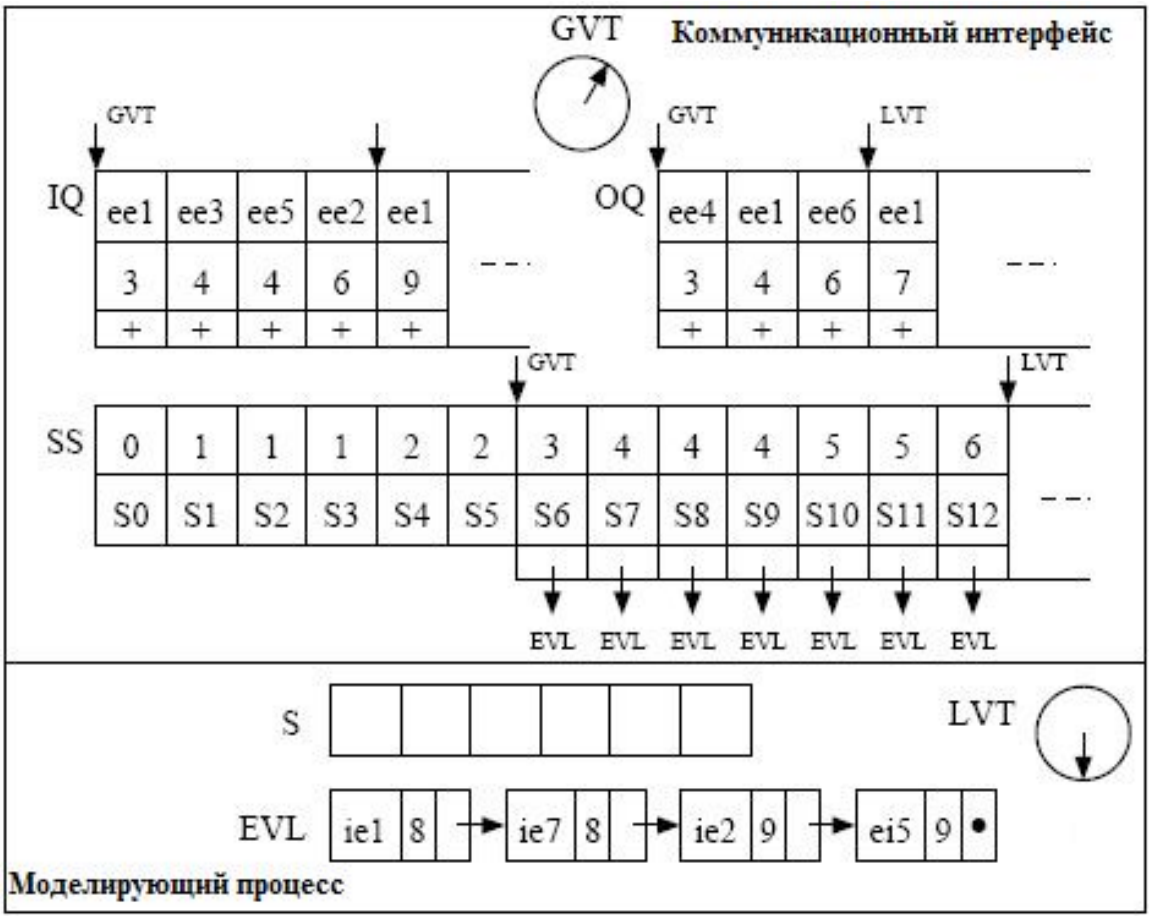

Рисунок 1. Структуры данных

новых. В случае возникновения отката из стека состояний удаляются все состояния до определенного момента времени в прошлом, происходит перемоделирование с учетом произошедших изменений [2]. Списки очищаются при изменении глобального виртуального времени (GVT) в системе, что является подтверждением всех вычислений, произошедших ранее.

Недостаток общего стека состояний в том, что при откате из него удаляются состояния абсолютно всех элементов схемы. Даже тех, которые никак не связаны с элементом, для которого возник откат. Например, для участка схемы, представленного на рис. 2(а), откат на элементе 1 должен привести к откату элементов 4 и 6 (рис. 2(б)), поскольку эти элементы зависят от него входами прямо или косвенно. Однако при использовании общего стека состояний будут удалены состояния всех элементов. Это увеличивает длину и время отката, а также увеличивает количество вычислений, необходимых для повторной обработки изменившегося списка событий.

При больших размерах моделируемой схемы с топологией узлов как на рис. 2.а количество откаченных состояний, которые можно было бы сохранять, равняется 25 - 50 % от всех состояний при равных вероятностях отката на всех узлах. Эта оценка усугубляется необходимостью пересчета состояний, которые были удалены.

Очевидно, что способ хранения последовательности состояний с помощью стека, описанный в [2], является неоптимальным и ведет к излишним накладным расходам как времени, так и процессорных тактов.

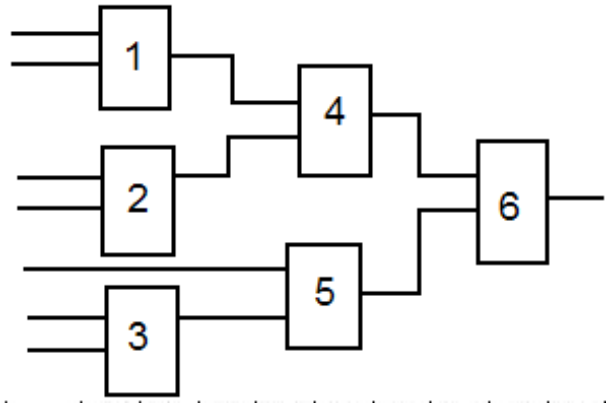

Рисунок 2, а) Пример участка схемы, моделируемого одним логическим процессором

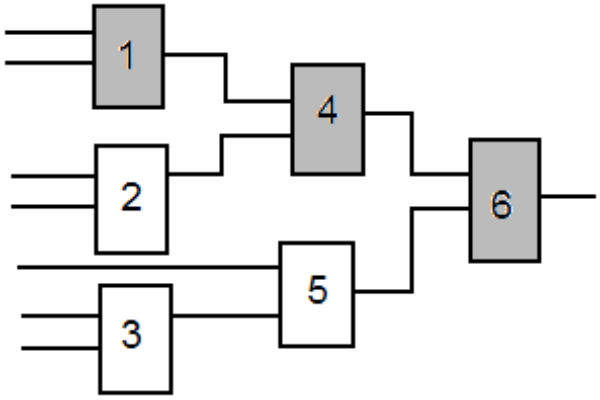

Рисунок 2, б) Элементы, которые должен затронуть откат на элементе 1

2 Раздельное сохранение состояний при моделировании с использованием оптимистического протокола синхронизации

Предлагаемый способ хранения состояний призван решить проблему с неэффективным откатом при использовании стека состояний. Он позволяет:

– не удалять состояния, которые не претерпят изменений в результате отката;
– сократить длину отката;
– сократить количество перевычислений.

Предлагается замена централизованного стека состояний на раздельные списки состояний для каждого элемента схемы (рис. 3).

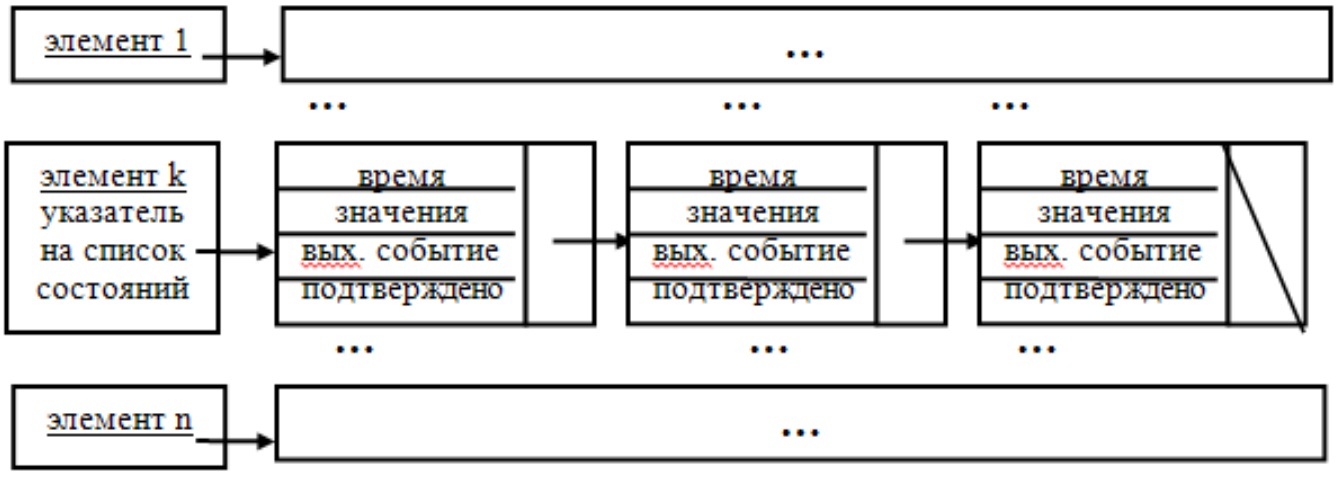

Рисунок 3. Предлагаемая структура для хранения состояний

Каждый моделирующий процессор имеет список ссылок на моделируемые элементы из СФСМ. Также для каждого элемента хранится его текущее локальное виртуальное время – время последнего обработанного состояния. Каждый такой элемент имеет динамический список состояний, который хранит состояния только для данного элемента. Список обновляется после изменения состояния на элементе. Применяется

инкрементное сохранение состояний, при котором сохраняются изменения значений элемента. Одна запись в списке состояний содержит информацию о моменте сохранения состояния, самом значении, состоянии входов и выходов элемента и сгенерированном событии, если оно есть, а также информацию о подтверждении состояния. Эта информация используется при откате и определении виртуального времени, она дает возможность удалять состояния, которые исчезли в результате откатов в системе.

Добавление нового состояния происходит при моделировании, когда меняются значения входов и/или выходов элемента, при этом в СФСМ происходит обновление значений входных и выходных узлов элемента, обновление глобальных входов и выходов, если оно произошло. Все эти изменения заносятся в соответствующую структуру и сохраняются в списке состояний элемента.

Процедура отката значительно отличается от алгоритма отката для стека состояний, описанного в [2]. Если какой-либо из элементов получает «событие из прошлого», т.е. событие с временной меткой, меньшей текущего локального времени, то он должен:

- начиная с конца списка удалить все состояния до момента времени, который указан в «событии из прошлого»;
- установить новое локальное виртуальное время (LVT) для элемента;
- изменить списки входных и выходных событий с учетом откаченных состояний – удалить все события, имеющие отношение к рассматриваемому элементу, время которых больше текущего LVT на элементе;
- добавить новое событие к списку событий и удалить сгенерированные неправильные события;
- выполнить переобработку списка событий.

Повторная обработка списка событий выполняет сразу две полезные функции. Во-первых, происходит логичный процесс моделирования с учетом внесенных изменений. Во-вторых, осуществляется аннигиляция неправильных выходных сообщений, если они есть. Для аннигиляции предлагается использовать технологию ленивого отката, которая подобна технологии ленивой отмены (Lazy Cancellation), описанной в [3]. Ленивый откат применяет исправление выхода только в том случае, когда достоверно известно, что вычисления неправильные, т.е. когда получено «событие из прошлого» или антисообщение на уже обработанное событие. Это осуществляется следующим образом:

1) если ранее в этот момент времени не было выходного события, а теперь оно должно быть, то процесс сгенерирует и отправляет необходимое выходное событие;
2) если ранее было отправлено событие и текущее событие совпадает с ранее отправленным, то процесс не выполняет никаких действий;
3) если ранее было отправлено отличное от текущего событие, то его необходимо аннигилировать, отправив антисообщение, и сгенерировать новое выходное событие.

Алгоритм ленивого отката представлен на рис. 4. При такой процедуре отката исправление состояния происходит на уровне узла, а не логического процесса. Удаляются

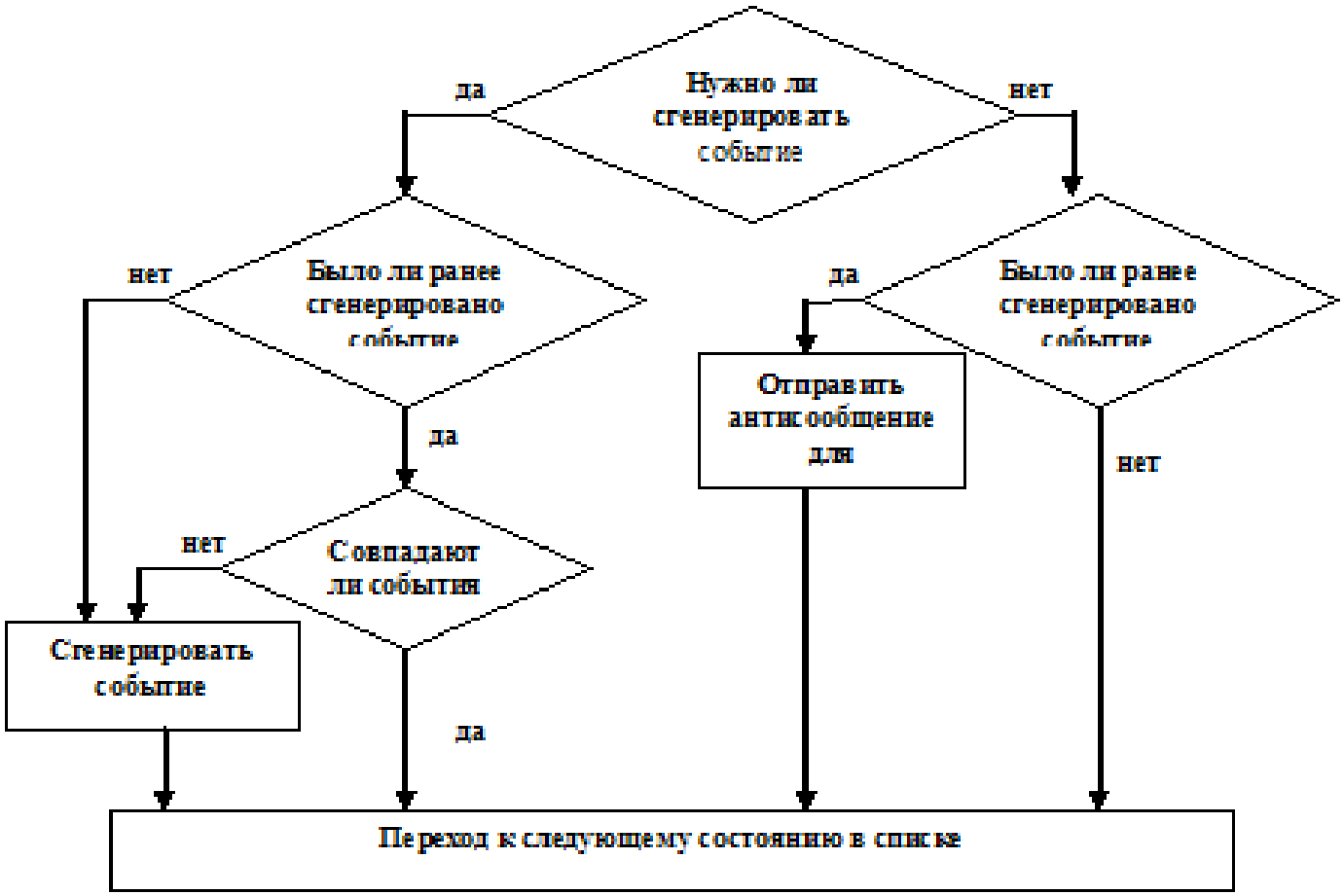

Рисунок 4. Алгоритм ленивого отката

5

состояния на узле, получившем «сообщение из прошлого» или антисообщение. Остальные узлы логического процесса ничего «не знают» об откате до тех пор, пока не получат антисообщение или сообщение с меньшим временем, чем собственное виртуальное время узла.

3 Особенности реализации раздельного сохранения состояний

Для хранения списка состояний предлагается использовать динамическую хеш-таблицу (хеш-список), ключом для которого является время события. Поскольку в каждый момент виртуального времени на каждом элементе может быть запланировано и обработано лишь одно событие, то при раздельном хранении состояний для каждого элемента список состояний будет содержать последовательность состояний с уникальными временными метками. Хеш-таблица позволяет обеспечить быстрый доступ к каждому конкретному состоянию.

Ленивый откат может быть упрощен с помощью использования структурно-функциональной модели схемы, которая позволяет определить список зависимых элементов. Зависимыми элементами считаются элемента, входы которых прямо или косвенно (через другие элементы) связаны с выходами элемента, на котором первоначально произошел откат. В таком случае откат должен сразу затронуть все необходимые элементы, не позволяя распространиться неправильным вычислениям.

Выводы

Предложенный способ сохранения состояний позволяет сократить длину и время отката, уменьшить количество вычислений за счет удаления только тех состояний, которые изменятся в результате откате. Это приведет к ускорению всего процесса моделирования.

Рассмотренный алгоритм ленивого отката позволяет удалять только те состояния элементов, которые должны быть затронуты изменениями. Использование структурно-функциональной модели схемы позволяет упростить реализацию отката, а использование хеш-таблиц для хранения самого стека состояний элемента ускоряет доступ к записям во время отката.

Перечень источников

[1] Benno Jaap Overeinder. Distributed Event-driven Simulation. Scheduling Strategies and Resource Management. Scheduling Strategies and Resource Management // PrintPartners Ipskamp, Enschede, The Netherlands, 2000.

[2] Ferscha A. Parallel and Distributed Simulation of Discrete Event Systems // Handbook of Parallel and Distributed Computing. – McGraw-Hill, 1995.

[3] Voon-yee Vee, Wen-jing Hsu. Parallel Discrete Event Simulation: A Survey // Centre for Advanced Information Systems, SAS; Nanyang Technological University, 1999.