УДК 004.75

ОПТИМИЗАЦИЯ ОТКАТОВ ОПТИМИСТИЧЕСКОГО ПРОТОКОЛА СИНХРОНИЗАЦИИ НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ РАЗДЕЛЬНОГО СОХРАНЕНИЯ СОСТОЯНИЙ И ЛЕНИВЫХ ОТКАТОВ

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

Аннотация

Попов Ю.В., Воротникова Ю.А. Оптимизация откатов оптимистического протокола синхронизации на основе использования раздельного сохранения состояний и ленивых откатов. *Рассмотрен подход сохранения состояний при оптимистическом моделировании, использующий журналирование и общую таблицу для хранения стека состояний всех узлов схемы. Предложен новый способ представления стека состояний и сохранения последовательности состояний для каждого узла в отдельности, что позволяет не удалять состояния, которые не должны быть затронуты процедурой отката.*

Введение

В настоящее время компьютерное моделирование цифровых систем стало популярной отраслью науки. Это связано с тем, что цифровые схемы повсеместно используются в бытовых приборах и на предприятиях, их конструкции достигли большой сложности, а производство требует немалых затрат. Разработано большое количество способов тестирования цифровых схем, наиболее перспективным является моделирование их работы с использованием распределенных компьютерных систем.

Одной из проблем при параллельном и распределенном моделировании схем является наиболее эффективное сохранение стека состояний системы. Предлагаются различные решения. Например, в [1] предлагается инкрементное сохранение состояний в динамической таблице, в [3, 4] рассматривается гибридное сохранение, представляющее собой комбинацию последовательного и инкрементного подходов.

Обзор существующих подходов к моделированию

Сложная топология цифровых логических систем привела к тому, что компьютерное моделирование подобного рода объектов требует большого количества как временных, так и аппаратно-вычислительных ресурсов. В связи с этим активно развиваются классы алгоритмов, которые выполняют моделирование в параллельных и распределенных системах.

Распределенное дискретно-событийное моделирование осуществляется набором логических процессов (ЛП), которые взаимодействуют друг с другом путем отправки сообщений с временными метками. Сообщения представляют моделируемые события. Каждый ЛП имеет свои локальные часы, называемые локальным виртуальным временем (ЛВВ), которое показывает продвижение процесса моделирования. Время увеличивается после обработки события на логическом процессоре.

При распределенном дискретно-событийном моделировании используются три подхода для синхронизации работы логический процессов – консервативный, оптимистический и смешанный. Консервативный протокол дает гарантию, что обработка события не приведет к появлению причинно-следственной ошибки из-за обработки сообщений не по порядку. Оптимистический протокол выполняет события в порядке их поступления, не гарантируя соблюдение причинно-следственных связей. Для исправления ошибки очередности выполнения событий используется механизм отката, в основе которого

– использование антисообщений для аннигиляции отправленных ранее сообщений с неправильными данными. Комбинированный протокол комбинирует два рассмотренных выше подхода синхронизации.

Достоинство оптимистического подхода, по сравнению с консервативным, заключается в том, что его легко распараллелить. Однако и он имеет свои недостатки. Во-первых, такие системы часто страдают от чрезмерно оптимистического выполнения, когда глубина откатов превышает продвижение процесса моделирования. Во-вторых, оптимистические системы требуют большого объема памяти для хранения состояний, чем обеспечивается возможность отката. Проблемой являются и неотменяемые операции, освобождение памяти и эффективное использование процессора.

Описание инкрементного подхода к сохранению состояний

В классической реализации оптимистичного протокола синхронизации состояния предполагается хранить в некотором журнале, который, по сути, является динамически связанной структурой данных. Этот журнал напоминает таблицу, в которую последовательно записываются данные о состоянии каждого узла моделируемой части схемы. Например, для участка схемы, представленного на рис. 1.а, журнал для хранения состояний может быть следующим (рис. 1.б).

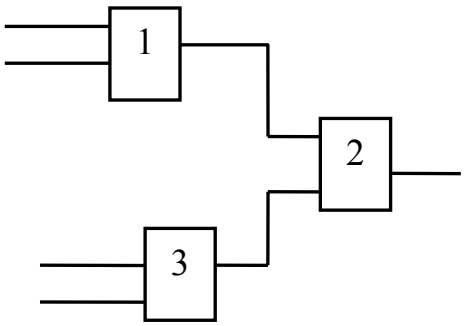

Рисунок 1.а – Пример участка схемы, моделируемого одним логическим процессором

Момент времени	№ узла	Значение
1	1	0
	3	0
2	1	1
	2	1
3	3	0
	1	1
4	3	0
	2	0

Рисунок 1.б – Пример журнала с состояниями узлов

Для такой схемы запись состояния будет состоять из операций:
- выделение памяти под структуру данных;
- изменение указателя на последний элемент таблицы;
- запись текущего значения;
- запись момента времени;
- запись выходного значения;
- запись номера узла-получателя сообщения;

При откате же будут последовательно отменены и удалены все сохраненные состояния вплоть до требуемого момента времени. Основной недостаток такого подхода заключается в том, что удаляются даже те состояния, которые можно не удалять. Например, при откате из момента времени 4 в момент времени 2 по причине того, что узел 3 получил сообщение с меткой времени 1, произойдет откат всех состояний (6 состояний) до момента времени 1. Из них лишних будет два, поскольку входы узла 1 никак не связаны с выходами узла 3. То есть, приблизительно треть состояний удаляется и пересчитывается после отката совершенно напрасно.

В целом для большой системы с топологией узлов как на рис. 1.а количество откаченных состояний, которые можно было бы сохранять, приблизительно равно трети всех откаченных состояний при равных вероятностях отката на всех узлах. Эта оценка усугубляется необходимостью пересчета состояний, которые были удалены.

Таким образом, очевидно, что способ хранения последовательности состояний системы, описанный в [1], является неоптимальным и ведет к излишним накладным расходам как времени, так и процессорных тактов.

Описание предлагаемой структуры данных для хранения состояний

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

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

Предлагается хранение состояний для каждого узла в отдельности (рис. 2).

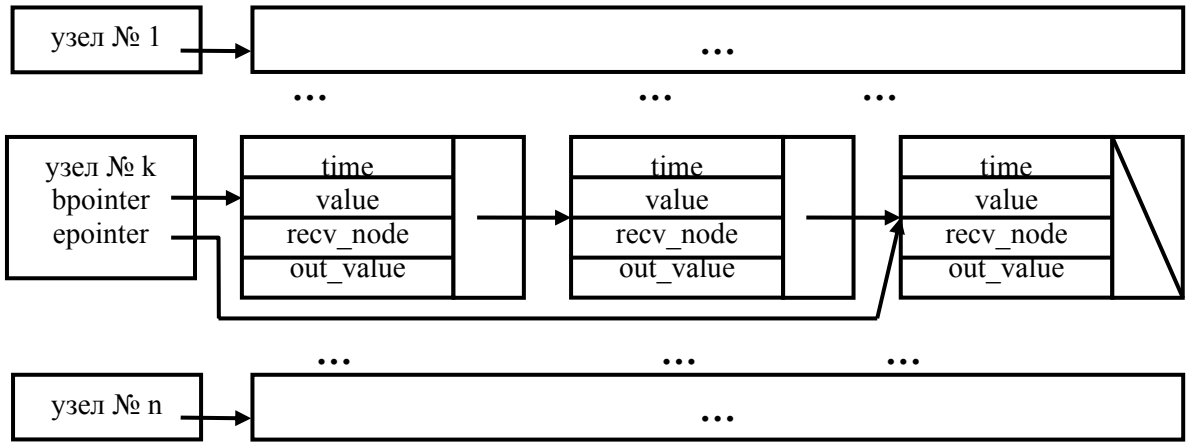

Рисунок 2 – Предлагаемая структура для хранения состояний

Каждый узел имеет собственный динамический список, который обновляется после изменения состояния на узле. Применяется инкрементное сохранение состояний, при котором сохраняются изменения значений узла. Запись содержит информацию о моменте сохранения состояния, самом значении и отправленном сообщении, если оно есть. Сам узел имеет два указателя – на начало и на конец списка для более удобного доступа к нему. Набор операций для добавлений нового состояния заключается в следующем:

- выделение памяти под структуру данных;
- изменение указателя на последний элемент списка в самом списке;
- изменение указателя на последний элемент списка в узле;
- запись текущего значения;
- запись момента времени;
- запись выходного значения;
- запись номера узла-получателя сообщения.

Процедура отката существенно отличается описанной в [1]. Если какой-либо из узлов получает «сообщение из прошлого», то он должен:

- найти момент времени в списке состояний, до которого необходимо выполнить откат;
- изменить указатель на конец списка в узле;
- записать новое состояния с учетом обработки «сообщения из прошлого»;
- выполнить проход по откаченным состояниям.

Проход по откаченным состояниям 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 используется технология ленивого отката, которая подобна технологии ленивой отмены (Lazy Cancellation), описанной в [2]. Ленивый откат применяет исправление выхода только в том случае, когда достоверно известно, что вычисления неправильные. Это осуществляется следующим образом:

1) если ранее в этот момент времени не было выходного сообщения, то процесс должен сгенерировать и отправить необходимое выходное сообщение;

2) если ранее было отправлено сообщение и текущее сообщение совпадает с ранее отправленным, то процессу не нужно выполнять никаких действий;

3) если ранее было отправлено отличное от текущего сообщение, то его необходимо аннигилировать, отправив антисообщение, и сгенерировать новое выходное сообщение.

Аналогично происходит и откат при получении узлом антисообщения от другого узла. При такой процедуре отката исправление состояния происходит на уровне узла, а не логического процесса. Удаляются 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.

Выводы

Предложенный способ хранения состояний позволяет сократить число операций при откате за счет уменьшения количества удаляемых состояний. Также сокращается процессорная нагрузка по причине того, что нет необходимости повторно пересчитывать ранее вычисленные состояния. Новый подход использует такое же количество памяти, какое необходимо при журналировании, поскольку не меняется структура хранимых данных. Запись состояния также не требует больше накладных расходов. Недостатком предложенной структуры можно считать необходимость поиска по списку состояний момента времени, до которого происходит откат.

Список литературы

1. Ferscha A., Thripathi S.K. Parallel and Distributed Simulation of Discrete Event Systems // Hardbound of Parallel and Distributed Computing. – McGraw-Hill, 1995.

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

3. Francesco Quaglia , Vittorio Cortellessa. Rollback-Based Parallel Discrete Event Simulation By Using Hybrid State Saving // proc. of 9-th European Simulation Symposium, 1997. – p. 275 – 279.

4. Benno J. Overeinder. Distributed Event-driven Simulation. Scheduling Strategies and Resource Management // Krakow, Faculteit: Natuurwetenschappen, Wiskunde en Informatica, 2000.