АЛГОРИТМЫ СИНХРОНИЗАЦИИ, ОСНОВАННЫЕ НА ЗНАНИЯХ О МОДЕЛИ, В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ ИМИТАЦИИ

Е.Б. Замятина[1], А.А. Митраков[2]
[1]НИУ ВШЭ(Пермский филиал), [2]ПГНИУ
e_zamyatina@mail.ru, mitrakov-artem@yandex.ru

Аннотация – Рассматриваются алгоритмы синхронизации в распределенных системах имитации и способы их оптимизации. Приводятся описания двух типов систем имитации: традиционной (событийно-ориентированная парадигма) Triad.Net и агентно-ориентированной. Предлагается использовать метод оптимизации алгоритмов синхронизации, основанный на знаниях. Приводятся результаты. подтверждающие важность использования знаний о конкретной имитационной модели для оптимизации алгоритмов синхронизации

Введение

Метод имитационного моделирования широко применяется для исследования сложных динамических систем, каковыми являются, в частности, компьютерные сети, параллельные и распределенные вычислительные системы (РВС), к которым можно отнести сетевые организации, виртуальные предприятия, социальные сети и т.д.. Еще одной интересной задачей имитационного моделирования является исследование масштабируемых алгоритмов, выполняющихся на базе параллллельных и распределенных систем. Исследователи, применяя метод имитационного моделирования, пытаются определить адекватность поведения параллельных и распределенных алгоритмов при значительном изменении количества вычислительных узлов (проиграть их функционирование в вычислительной среде, которой пока не существует). Очень часто количество вычислительных узлов в РВС таково, что имитационный эксперимент является затратным по времени.

По этой причине возникает необходимость в сокращении времени выполнения имитационного эксперимента. Одним из решений этой проблемы является использование высокопроизводительной техники. Система имитации, в этом случае, использует ресурсы нескольких вычислительных узлов для прогона распределенной имитационной модели. Распределенная (параллельная) имитационная модель представляет собой совокупность логических процессов, выполняющихся на различных вычислительных узлах и обменивающихся сообщениями друг с другом. Управляют функционированием распределенной имитационной модели алгоритмы синхронизации. Для того, чтобы распределенный имитационный эксперимент был эффективным, необходимо оптимизировать алгоритм синхронизации. Вторым источником оптимизации является балансировка логических процессов распределенной имитационной системы. В системе моделирования Triad.Net реализована мультиагентная система балансировки, описание и результаты можно посмотреть в [1].

1. Два класса алгоритмов синхронизации и способы их оптимизации

Алгоритмы синхронизации логических процессов распределенной имитационной модели обычно разделяют на два класса: консервативные и оптимистические[2].

Консервативные алгоритмы обычно «сдерживают» продвижение системного времени, очередное событие выполняется, если можно убедиться, что другой логический процесс не передаст текущему сообщение, помеченное меньшим штампом времени, чем рассматриваемое (текущее) событие. Для того чтобы расширить временные горизонты процессов и не попасть в тупиковую ситуацию, каждый логический процесс по всем выходным связям при изменении своего времени посылает сообщение – либо реальное, запрограммированное в модели, либо фиктивное или «нулевое».

Классическим *оптимистическим алгоритмом* является алгоритм «Time Warp»[3]. Алгоритм позволяет обрабатывать любые события (не только безопасные), но при поступлении сообщения с временной меткой T' (используют еще термин «штамп времени», которая меньше текущего времени T, происходит откат на момент времени T'. Для реализации отката необходимо восстановить состояние текущего логического процесса на момент времени T'. Кроме того, необходимо отменить сообщения, разосланные другим логическим процессам. Для этого используют механизм «анти-сообщений»: для каждого обработанного сообщения хранится противоположное ему. Для отмены сообщения посылают «анти-копию». При получении анти-сообщения процесс либо удаляет положительное сообщение из очереди, если оно не было еще обработано, либо так же производит откат до момента времени, которым помечено анти-сообщение.

2. Два подхода к организации распределенных систем имитационного моделирования

В настоящее время создаваемые распределенные системы имитации разделяются на два класса: традиционные, соответствующие основным парадигмам имитационного моделирования: -процессо-ориентированные, событийно-ориентированные, объектно-ориентированные; и агентно-ориентированнные. Если проектирование имитационной модели в первом случае ведется сверху-вниз, то в агентно-ориентированных взаимодействующие агенты совместно отображают реально-существующую систему, ситуацию, процесс (проектирование снизу-вверх). Агентно-ориентированные системы изначально представляют собой распределенную систему.

3. Распределенная система моделирования Triad.Net и алгоритм синхронизации TriadRule

Распределенная система имитации Triad.Net[4] была разработана на базе системы автоматизированного проектирования и моделирования Triad[5] и предназначалась для разработки и исследования вычислительных систем. Система Triad имеет отличительные характеристики, такие, как трехслойное представление модели, включающее слой структур, слой рутин и слой сообщений. Описание структуры вычислительной системы, обычно представлено графом P =(V,W,E), где V-множество вершин, соответствующих объектам моделируемой системы, W –множество входных и выходных полюсов, E –множество дуг, соединяющих эти вершины. Каждая вершина может быть описана составляющими ее компонентами (графом более низкого уровня). Слой рутин состоит из последовательности событий, планирующих друг друга. Эта последовательность событий описывает поведение каждой из вершин. Слой сообщений

необходим для описания сообщений сложной структуры. В качестве сообщения может быть рассмотрена и описана программа, которая выполняется на описанной модели вычислительной системы. Описание структуры, рутин, сообщения представляет собой параметризованные процедуры. Управляющая программа (симулятор) продвигает время от события к событию.

В распределенной модели отдельные вершины Р-графа могут располагаться на различных вычислительных узлах. Во время имитационного эксперимента распределенная модель представляет собой совокупность логических процессов (описанных рутинами), которые обмениваются сообщениями друг с другом. Для продвижения времени в распределенной системе имитации реализован оптимистический алгоритм Time Warp. Time Warp не ограничивает выполнение логических процессов, что приводит к большому количеству откатов, которые сигнализируют о значительных тратах вычислительных ресурсов впустую. Модельное время из-за откатов тоже продвигается медленно. Поэтому возникает необходимость в их оптимизации. Существует ряд работ [2], в которых используются знания об имитационной модели для сокращения времени выполнения распределенного имитационного эксперимента. Примерами являются классические алгоритмы: *lookahead* используют в консервативных алгоритмах, *lookback* в оптимистических, циклы в алгоритме Work Flow и т.д.

При реализации алгоритма синхронизации в Triad.Net [6] также были использованы знания о модели. Проблема алгоритма состоит в том, что необходимо на каждом шаге определить, является ли обработка следующего локального события безопасной, т.е можно ли гарантировать отсутствие отката. Анализ текущей последовательности событий (в рамках одного прогона) или результаты предыдущих выполнений системы, а также, знания эксперта позволяют выявить зависимости между событиями и, следовательно, определить безопасное событие. Наиболее адекватным решением является применение продукционной экспертной системы. Правила определяются не только знаниями эксперта о конкретной модели, но и генерируются автоматически: в ходе статического анализа кода модели перед выполнением, а так же в ходе выполнения: при откатах необходимо создавать правила, предотвращающие появление откатов в будущем. Например, если мы обработали событие покупки билета вторым клиентом (E2), а затем произошел откат из-за события покупки билета первым клиентом (E1), в системе генерируется связь ($E1 \rightarrow E2$).

Моделирование проводилось на кластере, с использованием нескольких простых моделей. Для сравнения производительности были выбраны классический консервативный и оптимистический алгоритмы. Наиболее показательные результаты были получены на модели вычислителя IllIAC.

По графику видно, что алгоритм TriadRule показывает лучшую производительность, чем оба традиционных алгоритма, причем при увеличении количества вычислительных узлов разница в производительности становится более заметна. Происходит это потому, что увеличивается количество логических процессов, а, следовательно, и общее количество внешних связей в системе. Для консервативного алгоритма это приводит к уменьшению быстродействия, т.к. временной горизонт

продвижения становится меньше, а для оптимистического приводит к увеличению количества откатов.

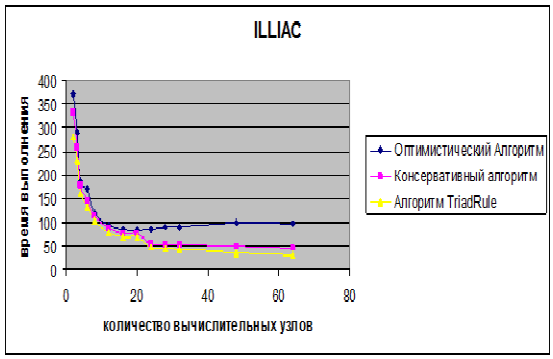

Рис. 1. Время выполнения имитационного эксперимента в зависимости от количества вычислительных узлов

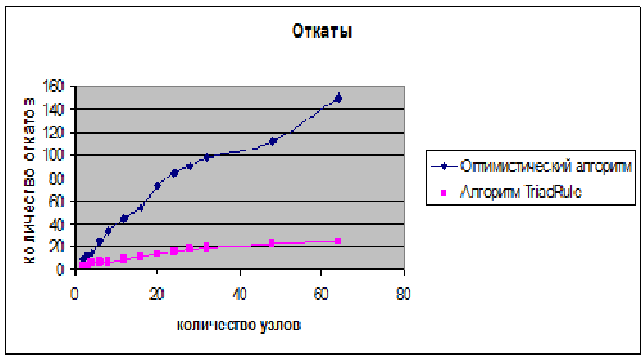

Рис. 2. Сравнение количества откатов

Итак, использование знаний о конкретной модели позволило улучшить характеристики алгоритма синхронизации в традиционных системах имитационного моделирования. Посмотрим, можно ли применить аналогичный подход в мультиагентных системах имитационного моделирования.

4. Мультиагентная система имитационного моделирования и алгоритм синхронизации, основанный на знаниях о модели

Настоящая версия мультиагентной системы моделирования выполнена на языке *Scala*[7]. Язык *Scala* был выбрана в качестве языка программирования для реализации агентной платформы моделирования. Язык *Scala* представляет собой кросспарадигменный объектно-функциональный язык программирования, совмещающий в себе ООП и функциональное программирование и специализирующийся на создании легкомасштабируемого компонентного программного обеспечения (*Scala* была создана в 2004г. под руководством Мартина Одерски в Университете *EPFL* (Lausanne, Switzerland)). Среди ключевых особенностей языка можно отметить: единую объектную модель, наличие примесей (*traits*), технику сопоставления с образцом, лямбда-исчисление, виды (*type views bounds*), линеаризацию типов (*type linearization*), параметрический и функциональный полиморфизм, вариантность типов, кейс-классы (*case classes*), вывод типов, поддержка хвостовой рекурсии, наличие многочисленных инструментов по созданию новых языковых конструкций и обработку списков. Обобщённая структурная схема симулятора изображена на рис. 3.

Таким образом, архитектура симулятора является не многокомпонентной, а многослойной. Достигается это за счёт того, что к модулю, реализующему некоторую базовую функциональность, «примешиваются» другие компоненты, расширяющие структуру, поведение и семантику. Трейт Simulator является базовым модулем, к которому подмешиваются другие элементы, уточняя его поведение. Фактически, Simulator, – это каркас для логического процесса. Трейт Communicator представляет собой модуль для работы с акторской системой. Он содержит в себе экземпляр актора, который занимается посылкой/приёмом сообщений. Трейт OptimisticSynchronizator является ключевым звеном для организации PDES – он реализует оптимистический

алгоритм синхронизации Time Warp. Трейт ModelObservable является реализацией концепции информационных процедур, принятых в области имитационного моделирования для организации сбора статистики (информационные процедуры – программные компоненты, накладываемые на модель с целью сбора статистики в ходе имитационного прогона). Трейт Analyzer добавляет «интеллектуальность» алгоритмам синхронизации, используемых для реализации распределенного моделирования.

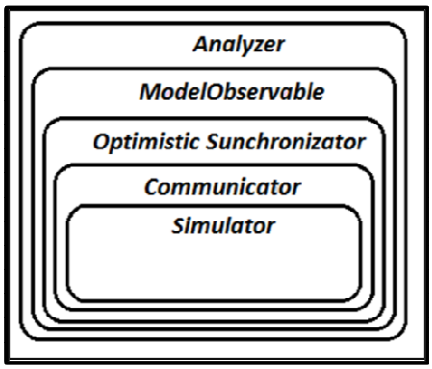

Рис. 3. Многослойная архитектура симулятора

В системе реализован оптимистический алгоритм синхронизации, основанный на знаниях об имитационной модели. Использование знаний об агентной имитационной модели, извлекаемые из онтологий, дали хорошие результаты при реализации алгоритмов синхронизации в агентной системе имитации[8]. Были проведены эксперименты с одной и той же моделью многократно при одних и тех же параметрах. Общая длительность моделирования составляла 300 (±15) ед. модельного времени.

Эксперименты показали, что метрики, характеризующие скорость проведения распределенного эксперимента, управляемого модифицированным алгоритмом синхронизации (алгоритм KBASA, основанный на знаниях о модели) существенно сократились по сравнениями с метриками классического оптимистического алгоритма Time Warp. Действительно, (а) Количество откатов сократилось с 71.2 до 3.8(рис.2.). (б) Общее число неглубоких откатов сократилось практически до нуля (в среднем 0.1-0.2 против 7-11). (в) Число глубоких откатов также сократилось с 19.4 до 3.2. (г) Количество отправленных антисообщений сократилось с 108.4 до 12.8 (рис.).

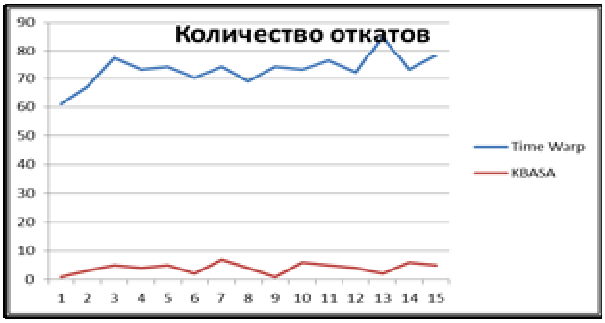

Рис.4.Количество откатов за 300 ед. модельного времени

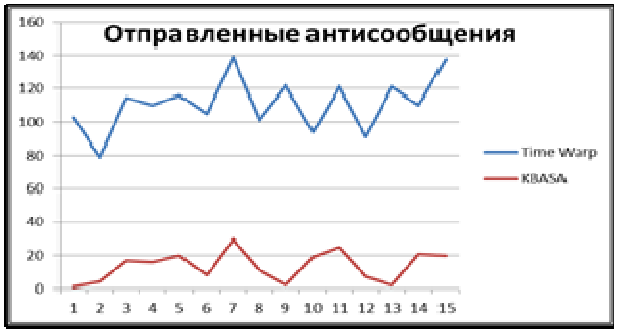

Рис.5.Отправлено антисообщений за 300 ед. модельного времени

Заключение

Итак, был рассмотрен подход, который использует знания о модели для сокращения времени выполнения имитационного эксперимента для распределенной как традиционной, так и агентной имитационной модели. Очень часто, при оптимизации алгоритмов синхронизации разработчики стремятся к универсализации алгоритма, а результаты исследований, приведенные выше, показали, что целесообразно еще и извлекать знания о конкретной модели. Этот же подход целесообразно применять при проведении балансировки вычислительных узлов, на которых проводится распределенный имитационный эксперимент.

Благодарности: Работа была выполнена при финансовой поддержке гранта РФФИ №12-07-00302- а, гранта РФФИ №13-07-96506_юг_а

Литература

1. Миков А.И., Замятина Е.Б., Козлов А.А. Мультиагентный подход к решению проблемы равномерного распределения вычислительной нагрузки. Natural and Artificial Intelligence, ITHEA, Sofia, Bulgaria, 2010, pp.173-180.
2. Fujimoto R.M. Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. The 2003 Winter Simulation Conference 7-10 December 2003. The Fairmont New Orleans, New Orleans, LA, pp. 124-134
3. Jefferson D.R. Virtual Time. ACM Transactions on Programming Languages and Systems, 1985. 7(3): p.404-425.
4. Замятина Е.Б., Миков А.И. Программные средства системы имитации Triad.Net для обеспечения ее адаптируемости и открытости. Информатизация и связь. №5, 2012, АНО «Редакция журнала «Информатизация и связь», ISSN 2078-8320, С.130-133.
5. Mikov A.I. Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages. Las Vegas, USA, 1995. P. 15 20.
6. Замятина Е., Ермаков С. Алгоритм синхронизации объктов распределенной имитационной модели в TRIAD.Net. Applicable Information Models. ITHEA, Sofia, Bulgaria, 2011, ISBN: 978-954-16-0050-4, стр.211-220
7. Odersky M. The Scala Programming Language [Электронный ресурс]. URL: http://www.scala-lang.org/node/25 (дата обращения: 06.06.2013)
8. *Замятина Е.Б., Каримов Д.Ф., Митраков А.А.*Архитектура агентно-ориентированной системы имитации с агентами, основанными на нейронных сетях. Информатизация и связь. 2014. № 2. С. 89-97.