СИНХРОНИЗАЦИЯ ОБЪЕКТОВ РАСПРЕДЕЛЕННОЙ ИМИТАЦИОННОЙ МОДЕЛИ

С.А. Ермаков, Е.Б. Замятина

Введение

В последние десятилетия появились программные разработки, связанные с поддержкой распределенного имитационного моделирования. Еще в 90-ые годы прошлого столетия появились статьи Chandy, Mizra[1] и Bagrodia[2] с описанием алгоритмов распределенного имитационного моделирования. В настоящее время распределенное имитационное моделирование стало востребованно. Это объясняется ростом сложности вычислительных задач, с которыми сталкивается имитационное моделирование, с одной стороны, и достаточно стремительным развитием парка высокопроизводительных ЭВМ.

Специально разработанные программные средства обеспечивают использование ресурсов гетерогенных или гомогенных многопроцессорных или мультикомпьютерных вычислительных систем (ВС). Использование ресурсов нескольких вычислительных узлов во время имитационного эксперимента сокращает время его проведения. Однако алгоритм управления распределенными по вычислительным узлам объектами (в дальнейшем будем называть этот алгоритм алгоритмом синхронизации) является гораздо более сложным, чем алгоритм продвижения времени в последовательном моделировании. Это направление в развитии распределенного митационного моделирования называют PDES (Parallel Discrete Event Simulation). Для создания новых высокоэффективных систем моделирования в этом случае используют спецпроцессоры для параллельного моделирования, сопутствующие языки, библиотеки и инструментальные средства. Выигрыш во времени получают при запуске сразу нескольких репликаций имитационного экспенимента (статистическое имитационное моделирование) на различных вычислительных узлах .

Еще одна из причин развития и распространения распределенного моделирования заключается в необходимости объединения уже готовых имитаций при помощи специального программного обеспечения. Примером может служить объединение нескольких тренажёров, имитирующих, управление танком, самолётом и других имитаторов (военные приложения)[3]. Их объединяют с целью создать распределённую виртуальную среду для обучения человека действиям по возможным сценариям в нештатных ситуациях. Другим примером может служить необходимость в объединении нескольких изучаемых подсистем (их моделей) в одну с той целью, чтобы можно было проанализировать их взаимодействие. Так, например, моделирование транспортной инфраструктуры города следует рассматривать неразрывно с подсистемой его электроснабжения и с другими коммуникационными инфраструктурами. В данном случае речь идёт о том, что гораздо более экономичным является подход, когда имитационные модели (в военных приложениях или при моделировании инфраструктур) объединяются с помощью дополнительных программных средств (HLA, High Level Architecture)[3]. В противном случае возникает необходимость в разработке новой имитационной модели. Распределенное имитационное моделирование целесообразно использовать при работе нескольких исследователей над одним и тем же имитационным проектом через Интернет. Проект может выполняться на суперкомпьютере или на вычислительной системе с кластерной архитектурой. Существуют и отечественные разработки в области распределенного имитационного моделирования. Здесь можно перечислить такие разработки как DYANA[4], MERA[5]. В настоящей работе речь пойдет о распределенной системе имитационного моделирования Triad.Net (Пермский государственный университет)[6]. Triad.Net является «монолитной распределенной системой», а это означает, что система должна обеспечить синхронизацию объектов распределенной имитационной модели. В работе приводится краткий обзор классических алгоритмов синхронизации объектов в распределенных системах имитации, рассматривается распределенная система имитации Triad.Net и особенности реализации алгоритма синхронизации объектов в этой системе.

При реализации алгоритма синхронизации авторы использовали *знания о модели* (знания пользователей о модели, знания, извлекаемые из структуры модели и знания, извлекаемые в процессе функционирования модели). Знания о модели представлены в виде правил. Алгоритм основан на классическом оптимистическом алгоритме. В работе приводятся результаты экспериментов, которые показали эффективность предложенного алгоритма синхронизации. Сначала расскажем об особенностях организации имитационной модели в Triad.Net.

Имитационная модель Triad.Net

Описание имитационной модели в Triad состоит из трех слоёв: слоя структур (STR), слоя рутин (ROUT) и слоя сообщений (MES). Таким образом, модель в системе Triad можно определить как M={STR,ROUT,MES}. Слой структур представляет собой совокупность объектов, взаимодействующих друг с другом посредством посылки сообщений. Слой структур можно представить графом. В качестве вершин графа следует рассматривать отдельные объекты. Дуги графа определяют связи между объектами.

Объекты действуют по определённому алгоритму поведения, который описывают с помощью рутины (rout). Рутина представляет собой последовательность событий e_i, планирующих друг друга. События e_i ($i=1\div n$) образуют множество E, множество событий рутины является частично упорядоченным в модельном времени.

Выполнение события сопровождается изменением состояния q_k объекта. Состояние объекта определяется значениями локальных переменных рутины. Таким образом, система имитации является событийно-ориентированной. Для передачи сообщения служит специальный оператор *out* (*out* <сообщение> *through* <имя полюса>). Совокупность рутин определяет слой рутин ROUT.

Слой сообщений (MES) предназначен для описания сообщений сложной структуры. Система Triad реализована таким образом, что пользователь может описать только один слой. Так, если возникает необходимость в исследовании структурных особенностей модели, то можно описать в модели только слой структур.

Алгоритмом имитации назовём объекты имитационной модели, функционирующие по определённым сценариям, и синхронизирующий их алгоритм. В распределенном моделировании эти объекты или группы объектов распределены по вычислительным узлам или процессорам. Во время функционирования модели объекты выполняют события за событием по определенному сценарию (последовательность упорядоченных по времени событий образует логический процесс) и обмениваются сообщениями с другими объектами посредством сообщений. Сообщения и события отмечены *«штампом времени».* Сообщения по сути дела также являются событиями посылки и приема сообщений. Задача распределенного алгоритма имитации-поддерживать каузальность событий. Предварительное размещение объектов по вычислительным узлам выполняется в Triad.Net специальным приложением подсистемы балансировки TriadBalance. Алгоритм статической балансировки в Triad.Net выполняет размещение объектов по вычслительным узлам (mapping), учитывая структурные особенности имитационной модели, связи между объектами (напомним, что модель представлена графом).

Сбор и обработка информации о функционировании модели

Для сбора, обработки и анализа имитационных моделей в системе Triad.Net существуют специальные объекты – информационные процедуры и условия моделирования. Информационные процедуры и условия моделирования реализуют алгоритм исследования.

Информационные процедуры ведут наблюдение только за теми элементами модели (событиями, переменными, входными и выходными полюсами), которые указаны пользователем. Если в какой-нибудь момент времени имитационного эксперимента пользователь решит, что следует установить наблюдение за другими элементами или выполнять иную обработку собираемой информации, он может сделать соответствующие указания, подключив к модели другой набор информационных процедур. Именно с помощью информационных процедур пользователь может осуществить взаимодействие с объектами модели во время имитации.Условия моделирования анализируют результат работы информационных процедур и определяют, выполнены ли условия завершения моделирования.

Система имитации Triad.Net располагает языковыми средствами для описания алгоритмов работы информационных процедур. Таким образом, подсистема анализа модели обеспечивает получение информации по заранее сформулированному запросу, а не ограничивает пользователя строго регламентированным набором собираемых данных. Такой подход к сбору информации позволяет избежать избыточности собранной информации или того, что она окажется недостаточной. Информационные процедуры и условия моделирования используют и для сбора информации о поведении модели, о ее характеристиках для распределенного алгоритма синхронизации.

Запуск модели

Модель может быть представлена графически или на встроенном языке TRIAD[7]. Далее выполняется трансляция описания в объектный код. По окончании трансляции выполняется этап статической балансировки: размещение объектов имитационной модели по узлам ВС. Далее происходит запуск модели на выполнение.

Время выигрыша от использования ресурсов нескольких вычислительных узлов может быть увеличено за счет эффективного алгоритма синхронизации. В докладе авторов предпринята попытка ускорить время выполнения алгоритмов синхронизации за счет его адаптации к конкретной имитационной модели, за счет знаний о ее поведении, которые используются при проведении распределенного имитационного эксперимента.

Распределенные алгоритмы синхронизации

Традиционно используются два подхода к реализации алгоритма синхронизации объектов моделирования, распределенных по разным вычислительным узлам: консервативный и оптимистический. Основной целью этих алгоритмов является обеспечение выполнения каждым логическим процессом событий в порядке не убывания их временных меток, чтобы сохранить причинно-следственные связи.

Принципиальная задача консервативного алгоритма – определить время, когда обработка очередного события из списка необработанных событий является «безопасным». Событие является безопасным, если можно гарантировать, что процесс в дальнейшем не получит от других процессов сообщение с меньшей временной меткой. Консервативный подход не позволяет обрабатывать событие до тех пор, пока не убедится в его безопасности. Подробное описание консервативных алгоритмов можно найти у R.Fujimoto, Chandy, Mizra и др.

В отличие от консервативных алгоритмов, не допускающих нарушения ограничения локальной каузальности, оптимистические методы не накладывают такого ограничения. Они обеспечивают

восстановление правильно порядка при его нарушении. Самый известный алгоритм - Time Warp, который был разработан Jefferson[8]. Когда логический процесс получает событие, имеющее временную отметку меньшую, чем уже обработанные события, он выполняет откат и обрабатывает эти события повторно в хронологическом порядке. Откатываясь назад, процесс восстанавливает состояние, которое было до обработки событий (все состояния системы сохраняются) и отказывается от сообщений, отправленных в результате обработки событий, которые необходимо отменить в результате отката. Для отката от этих сообщений разработан механизм антисообщений.

Консервативные и оптимистические алгоритмы представляют собой две крайности. Для некоторых специфических моделей консервативные алгоритмы показывают производительность, превосходящую оптимистические в несколько раз, обратное, вообще говоря, тоже верно. Однако, проанализировав все выше перечисленные алгоритмы синхронизации, можно четко проследить тенденцию возрастания эффективности алгоритма при увеличении количества информации, которое известно о модели (lookahead, lookback, временная метка следующего события и т.п.). Эту информацию используют как консервативные, так и оптимистические алгоритмы: консервативные, таким образом, позволяют увеличить горизонт безопасных событий, а оптимистические наоборот, сдержать чрезмерное продвижение модельного времени.

Рассмотрим алгоритм, основанный на применении такой информации, созданный на базе оптимистического алгоритма. С точки зрения моделирования, главным является соблюдение правильного порядка выполнения модели во времени - каузальная связь. Кроме того, между событиями должна сохраняться дистанция по времени. Эти два типа связи применимы к событиям и сообщениям.

Пользовательская информация

Как правило, пользователь системы моделирования имеет некоторое приблизительное неточное представление о модели, которое он может описать в виде правила, например: «Прежде, чем получить товар, его нужно оплатить» - получаем представление этого знания в виде правила: «IF Оплатить Товар THEN Получить Товар». Как видно из примера, информация, задаваемая пользователем, является достаточно простой, однако для алгоритма синхронизации она позволит сократить количество откатов, и, следовательно, уменьшить время выполнения модели.

Очевидно, что знания эксперта являются нечеткими и ненадежными, поэтому каждому правилу приписывается коэффициент доверия CF.

Казуальная зависимость между событиями в общем виде представляется в виде

IF E1 AND E2 AND E3 AND … AND En-1 THEN En CF <0..100>. Здесь En зависит от E1,E2,E3 … En-1. CF - коэффициент доверия. 0 – нет доверия, 100 – максимальное доверие.

Информация, генерируемая самой системой

Системная информация также представлена в виде правил, которые формируются на двух стадиях обработки рутин (рутины представляют собой последовательность событий, планирующих друг друга): на стадии трансляции из описания рутин; на стадии выполнения.

В общем случае, при увеличении количества различных событий и сообщений в системе, количество правил может расти экспоненциально. Поэтому генерировать все возможные зависимости трудоемко для системы. Более того, скорее всего большинство из сгенерированных правил могут ни разу не сработать (например, если правило описывает отношения объектов в рамках одного и того же локального процесса). По этой причине в информации о связи событий будем придерживаться следующих утверждений: (а) генерация правил происходит только для объектов системы находящихся в разных локальных процессах; (б) наиболее значимыми правилами являются те правила, которые помогают предотвратить временной парадокс (они генерируются из тех событий, которые в прошлом вызвали временной парадокс).

Сбор, анализ и распространение информации

Опишем общую схему реализации сбора информации. Сам по себе логический процесс не должен тратить свои вычислительные ресурсы на обработку информации. Он должен использовать ее уже в готовом виде. Поэтому предлагается создание специального агента, который выполняет следующие функции: (а) сбор информации; (б) удаление избыточной информации; (в) анализ информации на непротиворечивость; (г) преобразование информации в удобный для использования вид (сворачивание мелких правил в одно более общее, сортировка правил в соответствии с их точностью (CF) и релевантностью); (д) обмен информацией между локальными процессами.

Кроме агентов, расположенных на локальных вычислительных узлах, вся информация агрегируется в центральную базу знаний, расположенную на отдельно выделенном сервере. Главный управляющий процесс выполняется на этом же сервере. В качестве агентов используют информационные процедуры.

Использование информации при моделировании

Будем строить алгоритм на основе обычного оптимистического алгоритма с посылкой антисообщений и откатами (алгоритм Time Warp). Главная задача алгоритма – минимизировать количество откатов в системе, которые тормозят работу системы, при максимальном сохранении параллелизма в модели. Чтобы откаты не происходили, необходимо, чтобы обрабатывались только безопасные события. Определение безопасности события происходит по следующему алгоритму: (а) выполняется поиск конкретного события в заключениях

всех правил в базе знаний системы; (б) если таких правил не обнаружено, то происходит выполнение по обычному сценарию.

Далее алгоритм выполняет логический вывод по продукционной базе знаний. В результате возможно 2 результата: событие B выведено с некоторой вероятностью CA или событие B не выведено. При вычислении доверия к заключению доверия к сработавшим правилам комбинируются. Алгоритм решает, следует ли обрабатывать данное событие исходя из доверия к заключению. Если доверие максимально, то событие обрабатывается со 100% вероятностью.

Вычислительный эксперимент

Для проведения вычислительного эксперимента были выбраны несколько моделей, которые тестировались с применением различных алгоритмов на разных конфигурациях ВС с целью выявления наиболее оптимального алгоритма. Основной показатель оптимального алгоритм – время выполнения, а для оптимистического алгоритма – это еще и количество откатов. Будем считать, что время выполнения всей модели – максимальное из времен выполнения каждого локального процесса.

Модель вычислителя ILLIAC

Модель представляет собой в значительной степени упрощенную схему распределенных вычислений. Представлена мультипроцессорная вычислительная система с 64 вычислительными узлами – процессороми, где каждый процессор связан с четырьмя соседними. Процессоры выполняют вычисления и обмениваются данными между собой. Схема вычислительной системы изображена на рис.1. Распределенный алгоритм синхронизации характеризуется в этом случае большим количеством откатов вследствие большого количества связей и асинхронного характера обмена информацией между локальными процессами выислительных узлов.

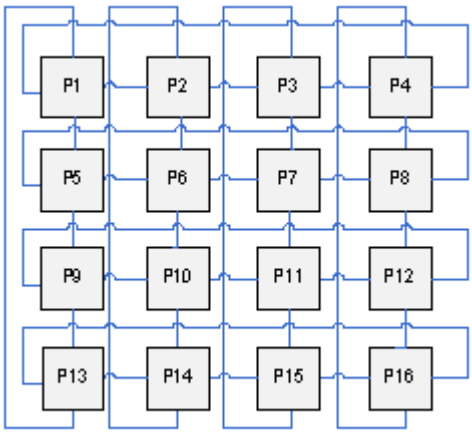

Рис. 1. Схема взаимодействия узлов в вычислительной системе ILLIAC

Проведем вычислительный эксперимент с моделью, для уменьшения объема результатов, будем полагать, что модельное время = 5000 ед.

Таблица 1. Результаты эксперимента с моделью "ILLIAC"

Количество узлов	Время выполнения		
	оптимистический	консервативный	TriadRule
2	370,5	330,5	280,73
3	290,1	260,07	230,75
4	185	177,6	162,8
6	170,6	146,3	133,275
8	120	115	103,75
12	96,37	87,57	78,77
16	85	76,7	68,3
20	84,3	76,3	68,3
24	85,7	55,4	47,8
28	90	53,1	44,1
32	91,2	52,1	42,3
48	100	49,7	36,7

64	97,3	46	31,3

График показывает, что время выполнения всех алгоритмов за исключением оптимистичекого уменьшается с увеличением количества узлов. При увеличении количества узлов, количество внешних связей растет, а это приводит к увеличению количества откатов, следовательно, время выполнения оптимистического алгоритма, начиная с некоторой точки (в данном эксперименте это 23 вычислительных узла), растет.

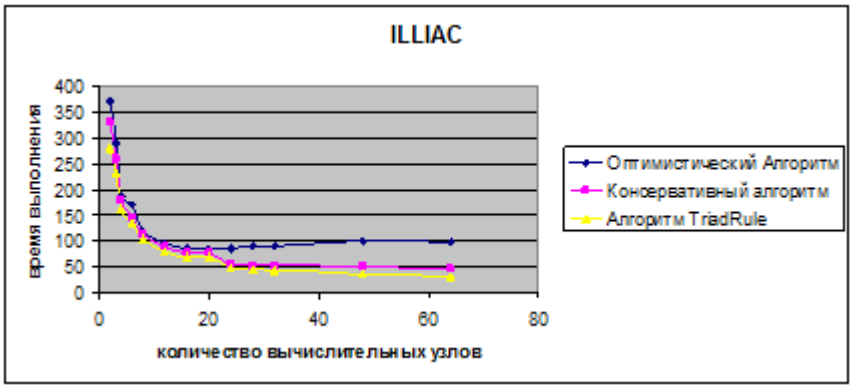

Рис. 2. Диаграмма Зависимости времени выполнения от количества узлов для модели ILLIAC

Консервативный алгоритм постепенно сокращает время выполнения, однако в силу ограниченности параллелизма, сокращение времени его выполнения происходит медленно. Новый алгоритм показывает самые лучшие результаты, поскольку позволяет сохранить параллелизм, уменьшив количество откатов в системе. Количество откатов в оптимистическом алгоритме возрастает прямо пропорционально увеличению количества вычислительных узлов, а, значит, и связей между локальными процессами. Поэтому, не смотря на ускорение вычислений, общее время выполнения увеличивается. Разработанный алгоритм позволяет свести возрастание количества откатов к минимуму, по графику видно, рост количества откатов происходит медленно.

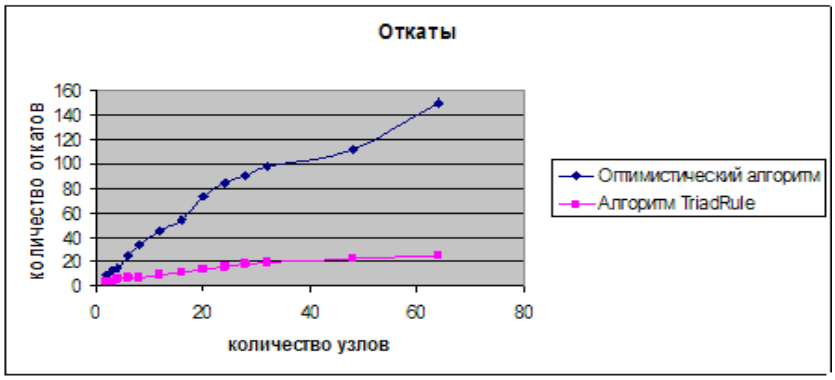

Рис. 3. Диаграмма зависимости количества откатов от количества узлов для модели ILLIAC

Проанализировав полученные результаты можно заметить следующее: консервативный алгоритм сдерживает выполнение, уменьшая скорость выполнения, оптимистический алгоритм наоборот, слишком забегает вперед, что приводит к откатам, которые сильно тормозят продвижение времени. Алгоритм TriadRule – это оптимистический алгоритм, однако для сдерживания чрезмерного забегания логического процесса вперед он использует правила. Эксперименты показали, что сдерживание получается минимальным. Однако для моделей, которые по своей сути являются параллельными, а именно, временные парадоксы в которых возникают редко или вообще не возникают, данный алгоритм оказывается хуже оптимистического. Рассмотрим этапы выполнения алгоритма TriadRule и их характеристику:

Таблица 2. Этапы выполнения алгоритма синхронизации, основанного на знаниях

Этап	Скорость выполнения	Количество правил	Происходящие процессы
Начальный	Как у оптимистического	Правил нет, или очень мало	Накопление правил
Средний	Скорость приближается к консервативному	Правила есть, однако они срабатывают редко, что приводит к замедлению выполнения	Накопление правил и коррекция коэффициентов доверия
Последний	Скорость значительно выше и консервативного и оптимистического алгоритма	База знаний правил почти полностью сформирована	Происходит лишь слабое пополнение БЗ и небольшая корректировка коэффициентов

Таким образом, алгоритм начинает приносить выгоду только на последнем этапе. Для каждой модели данный этап наступает в разное время. Главная задача повышения эффективности вычислений в том, чтобы последний этап наступил как можно скорее. Для того, чтобы приблизить этот этап, необходимо изначально наполнять базу знаний пользовательскими правилами.

Заключение

В работе рассматривается проблема разработки алгоритма синхронизации объектов распределенной имитационной модели, основанного на знаниях. Авторы предлагают модернизировать классический оптимистичекий алгоритм с откатами Time Warp. Выполнение классического оптимистического алгоритма корректируется с помощью продукционных правил, которые построены на знаниях пользователя о модели. Правила синхронизации включают также знания о модели, извлеченные во время функционирования модели. Результаты эксперимента подтверждают, что разработанный алгоритм TriadRule позволяет частично сократить временные затраты на моделирование. Сбор информации о модели выполняется с помощью информационных процедур. Для реализации алгоритма TriadRule были использованы встроенные информационные процедуры. Однако пользователь может внести в процедуру сбора информации коррективы, воспользовавшись языком Triad и лингвистическими средствами описания информационных процедур. Таким обрзом, алгоритму придается дополнительная гибкость.

ЛИТЕРАТУРА:

1. Chandy K.M., Misra J. Distributed simulation: a case study in design and verification of distributed programs IEEE Transactions on Software Engineering. 1978. Vol. SE–5(5).P. 440-452.
2. Meyer R.A., Bagrodia R. Parsec User Manual. Release 1.1., UCLA Parallel Computing Laboratory, 1998 Доступно на сайте: pcl.cs.ucla.edu/projects/parsec
3. 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
4. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата. Труды Всероссийской науно-практической конференции «Методы и средства обработки информации», МСО-2005, М.: Издательский отдел факультета ВМиК МГУ, М.:МГУ,2005,-стр. 59-76.
5. ОкольнишниковВ.В. Распределённая система имитационного моделирования. Труды Всероссийской науно-практической конференции «Методы и средства обработки информации», МСО-2003, М.: Издательский отдел факультета ВМиК МГУ, М.:МГУ,2005,-стр. 468-4
6. Миков А.И., Замятина Е.Б. Технология имитационного моделирования больших систем // Труды Всероссийской научной конференции «Научный сервис в сети Интернет» – М.: Изд-во МГУ, 2008. С.199-204.
7. Mikov A.I. Formal Method for Design of Dynamic Objects and Its Implementation in CAD Systems // Gero J.S. and F.Sudweeks F.(eds), Advances in Formal Design Methods for CAD, Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design, Mexico, Mexico, 1995. pp.105-127.
8. Jefferson D.R., Virtual time II: storage management in distributed simulation Proc. of the Ninth Annual ACM Symposium on Principles of Distributed Computing.