АЛГОРИТМ СИНХРОНИЗАЦИИ ОБЪЕКТОВ РАСПРЕДЕЛЕННОЙ ИМИТАЦИОННОЙ МОДЕЛИ В TRIIAD.NET

Елена Замятина, Сергей Ермаков

Аннотация: *В работе приводится краткий обзор классических алгоритмов синхронизации процессов в распределенных системах имитации, рассматривается распределенная система имитации Triad.Net и особенности реализации алгоритма синхронизации процессов 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.*

Keywords*: имитационное моделирование; экспертные сстемы, распределенное имитационное моделирование, консервативный алгоритм синхронизации, оптимистический алгоритм синхронизации, база знаний.*

ACM Classification Keywords*: I.6 Simulation and Modelling – I.6.2 Simulation Languages; I.2 Artificial Intelligence – I.2.5 Programming Languages and Software – Expert system tools and techniques.*

Введение

В настоящее время актуален переход к созданию распределенных систем имитационного моделирования (СИМ). Как известно, имитационное моделирование является часто используемым, а иногда и единственным методом исследования сложных систем. Сложность задач, решаемых методом имитационного моделирования, явилось причиной, по которой наряду с последовательными системами имитационного моделирования появились распределенные/параллельные системы имитации[Fujimoto, 2003; Meyer, 1998;Окольнишников,2005]. Разработка распределенных систем имитации сопровождается необходимостью создания специального программного обеспечения, которое позволило бы использовать ресурсы 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 объектами, распределенными по вычислительным узлам (в дальнейшем будем называть его алгоритмом синхронизации), является гораздо более сложным, чем алгоритм продвижения времени в последовательном моделировании. 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.

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 Р.Фуджимото, Чанди, Мизры и др. [Bryant,1977; Chandy, 1978].

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 Джефферсоном[Jefferson, 1990] . 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 утверждение, вообще говоря, тоже верно. Однако, проанализировав все перечисленные выше 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. Отсюда можно сделать вывод, что лучший алгоритм должен сочетать в себе оба подхода. Сформулируем требования, которым должна удовлетворять информация:

- Информация помогает максимально повысить эффективность алгоритма имитации.
- Информацию может вводить пользователь (создатель модели, эксперт).
- Информацию можно получать от самой системы моделирования в процессе её выполнения.
- Информацию легко хранить и обрабатывать.
- Информация адекватно представляет знания о модели.

В настоящей реализации информация о модели представлена в виде продукционных правил.

Рассмотрим алгоритм, использующий такую информацию во время имитационного прогона и созданный на базе оптимистического алгоритма. С точки зрения моделирования, главным является соблюдение правильного порядка выполнения модели во времени. Следовательно, получаем два основных типа связи, который сами по себе связаны друг с другом:

- Каузальная связь - связь последовательности выполнения событий.
- Дистанция по времени (временное расстояние) между событиями.

Данные двух типов связей применимы к объектам, имеющим временные метки, то есть 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 (например, в данном конкретном случае, товар может быть украден, или наоборот, после оплаты оказалось, что товар некачественный и подлежит возврату), поэтому каждому правилу приписывается коэффициент доверия CF.
Казуальная зависимость между событиями в общем виде представляется в виде

IF E_1 AND E_2 AND E_3 AND … AND E_n THEN E_k CF [0..100].

Здесь E_k зависит от E_1, E_2, E_3,…, E_n. CF - коэффициент доверия. 0 – нет доверия, 100 – максимальное доверие.

Информация, генерируемая самой системой

Пользовательская (экспертная) информация является самой важной, так как эти знания мы получаем от эксперта в своей предметной области. Однако необходимость детального описания поведения системы пользователем затруднительна для самого пользователя (извлечение знаний – это трудоемкий процесс). К тому же при имеющемся всеохватывающем описании системы теряется главный смысл проведения моделирования – зачем его проводить, если мы и заранее знаем поведение системы. Исходя из этого, система должна уметь сама обеспечивать себя необходимой информацией. Для упрощения описания информации и для интегрирования двух типов знаний, системная информация, так же как и пользовательская будет представлена в виде правил. Системные правила формируются на различных стадиях:

- *На стадии трансляции*:
  рутина представляет собой пользовательские процедуры обработки событий и сообщений, поэтому она по определению содержит информацию о каузальной связности событий и сообщений.
- *На стадии выполнения*:
  во время выполнения модели последовательность обработки событий и сообщений анализируется, и на основе этой информации генерируются правила.

В общем случае, при увеличении количества различных событий и сообщений в системе, количество правил может расти экспоненциально. Поэтому генерировать все возможные зависимости очень трудоемко для системы. Более того, большинство из сгенерированных правил могут ни разу не сработать (например, если правило описывает отношения объектов в рамках одного и того же локального процесса).

Поэтому, исходя из потребностей алгоритма моделирования в информации о связи событий, будем придерживаться следующих утверждений:

- Генерация правил происходит только для объектов системы находящихся в разных локальных процессах.
- 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:

- Сбор информации.
- Удаление избыточной информации.
- Анализ информации на непротиворечивость.
- Преобразование информации в удобный для использования вид (сворачивание мелких правил в одно более общее, сортировка правил в соответствии с их точностью (CF) и релевантностью).
- 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 на локальных вычислительных узлах используют информационные процедуры. Речь о них пойдет несколько позже. Информационные процедуры – это средства сбора информации об имитационной модели в Triad.

Использование информации во время имитационного прогона

Ключевой момент алгоритма – использование полученной информации о модели для синхронизации объектов имитационной модели во время прогона. Поскольку консервативный алгоритм синхронизации не допускает временных парадоксов, а применение правил не дает уверенности в данном случае неприемлемо. Алгоритм в системе Triad.Net будет строиться 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.

Рабочая память системы содержит факты – события и сообщения, которые были обработаны в течение некоторого промежутка времени ΔT_{act} это время, в течение которого данные объекты считаются актуальными. Каждый раз при увеличении времени локального процесса, рабочая память актуализируется.

Определение безопасности события происходит по следующему алгоритму:

- Выполняется поиск данного события в заключениях всех правил в базе знаний системы.
- Если таких правил не обнаружено, то происходит выполнение по обычному сценарию.

Если же такие правила обнаружены, то выполняется логический вывод по продукционной базе знаний. В результате возможно 2 результата: событие B выведено с некоторой вероятностью CF или событие B не выведено. При вычислении доверия к заключению доверия к сработавшим правилам комбинируются.

Важно заметить, что изначально база знаний может быть не заполнена до такой степени, чтобы обеспечить эффективное выполнение алгоритма: если событие B не было получено в результате логического вывода, то это не значит, что его не надо обрабатывать в данный момент. Это может значить, что еще не было добавлено правило, в результате которого можно получить событие B в текущем состоянии системы. Поэтому необходимо установить временную границу начального обучения алгоритма, когда его база знаний (БЗ) наполняется. Если система находится в стадии заполнения БЗ, то событие B так или иначе будет обработано, в противном случае, не будет.

Система решает, следует ли обрабатывать данное событие, исходя из доверия к заключению. Возможны две стратегии решения обрабатывать событие или нет:

- *Пороговая*: если CF больше заданного порога (для каждого объекта может быть свой порог: статический или динамически вычисляемым (например, с помощью генетического алгоритма)).
- *Вероятностная:* чем больше CF, тем больше вероятность, что событие будет обработано. В реализации этого подхода используют генератор случайных чисел.

В случае, если сработавшее правило привело к откату системы, его CF снижается. В противном случае, CF повышается.

В посылке и заключении правила можно указывать событие конкретного объекта (в случае, если в модели есть несколько одинаковых объектов): IF O1.E1 THEN O2.E2. Так же в посылке правила можно задать условие на состояние объекта: IF O1.Attribute1 = value1 AND O1.E1 THEN O2.E2. Это необходимо, в случае, если последовательность обработки событий зависит от внутреннего состояния объекта.

Таким образом, система позволяет максимально учитывать зависимости между объектами на стадии выполнения.

Описанный выше алгоритм, основанный на знаниях, реализован в системе Triad.Net[7]. Несколько слов о системе моделирования Triad.Net.

Распределенная система имитации Triad.Net

На кафедре математического обеспечения ВС разрабатывается распределенная версия системы автоматизированного проектирования и моделирования Triad.Net. Распределённая версия реализуется с использование технологии .Net. Использование технологии .Net дают возможность реализовать гибкую компонентно-ориентированную систему.

Triad.Net является распределенной версией системы моделирования Triad [Миков, 2009;Миков,2008]. Особенности системы моделирования:

- Входной язык описания моделей содержит переменные типа «модель». Над моделями определены операции. Операции определены как для моделей в целом, так и для каждого слоя (в Triad модель представлена слоем структур, рутин и сообщений.). Имитационная модель может быть описана средствами языка Triad или построена в результате исполнения некоторого алгоритма преобразования модели. При выполнении операций над моделью (или операций внутри каждого из слоев) перетрансляции модели не требуется.
- Модель является иерархической, т.е. каждая вершина в слое структур может быть расшифрована подструктурой.
- Подсистема анализа модели должна обеспечить получение информации по заранее сформулированному запросу, а не ограничивать пользователя строго регламентированным набором собираемых данных. Такой подход к сбору информации позволяет избежать избыточности собранной информации или того, что она окажется недостаточной.
- Алгоритм имитации должен быть эффективным и масштабируемым, т.е. объекты должны быть таким образом распределены по компьютерам, чтобы снизить потери, связанные с передачей сообщений по каналам связи от компьютера (процессора) к компьютеру (процессору) и учитывать загрузку компьютеров (процессоров) (В СИМ Triad разрабатывается подсистема управляемой динамической балансировки).
- Система должна быть объектно-ориентированной (должно поддерживаться наследование, переиспользование кода).

Описание имитационной модели

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. Каждый объект имеет полюса (входные Pin и выходные Pout), которые служат соответственно для приёма и передачи 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 ($e_i \in E, i = \overline{1,n}$, E-множество событий, множество событий рутины является частично упорядоченным в модельном времени). Выполнение события сопровождается изменением состояния qk объекта.

Состояние объекта определяется значениями переменных var_j рутины ($var_j \in Var, j = \overline{1,m}$, Var – множество переменных рутины). Таким образом, система имитации является событийно-ориентированной. Рутина так же, как и объект, имеет входные ($Prin$ и выходные $Prout$) полюса. Входные полюса служат соответственно для приёма сообщений, выходные полюса – для их передачи. В множестве событий рутины выделено входное событие ein. Все входные полюса рутины обрабатываются входным событием. Обработка выходных полюсов осуществляется остальными событиями рутины. 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, функционирующие по определённым сценариям, и синхронизирующий их алгоритм.

Итак, выше были приведены сведения об имитационной модели в Triad.Net для выполнения ее на одном вычислительном узле. В распределенном варианте (выполнение на нескольких вычислительных узлах) имитационная модель представляет собой совокупность объектов, распределенных по разным вычислительным узлам. На одном узле могут располагаться один или несколько объектов. Во время имитационного выполнения объекты обмениваются инфорацией друг с другом, посылая сообщения.размещение объектов по вычислительным узлам происходит во время этапа статической балансировки. Критерием размещения объектов по узлам являются структурные особенности имитационной модели: сильносвязанные объекты целесообразно разместить на одном узле.

Информационные процедуры

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. Информационные процедуры являются единственным средством системы для одновременного доступа к элементам модели, принадлежащим разным объектам. 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. В описание информационных процедур входят: описание начальной части (выполняется до начала имитационного эксперимента), описание заключительной части (эта часть выполняется по окончании эксперимента и служит для окончательной обработки интегральных характеристик, например, подсчет среднего значения), описания основной части информационных процедур, настроечных параметров и параметров интерфейса. При изменении значения переменной, за которой ведётся наблюдение, при выполнении события, указанного пользователем, или после прихода (передачи) сообщения на входной полюс происходит подключение информационной процедуры (ее основной части) к конкретному элементу модели (посредством параметров интерфейса) и данные обрабатываются по заданному в информационной процедуре алгоритму.

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 организации алгоритма синхронизации.

Вычислительный эксперимент

Модель может быть представлена графически или на встроенном языке TRIAD [Миков, 1995]. Далее выполняется трансляция описания модели в объектный код. По окончании трансляции выполняется этап статической балансировки: размещение объектов имитационной модели по узлам вычислительной системы (ВС), ну а потом - запуск модели на выполнение.

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 алгоритма – время его выполнения, а для оптимистического алгоритма – это еще и количество откатов. Стоит отметить, что исследования выполнялись для достаточно простых (по вычислительной сложности) моделей. Время выполнения всей модели – максимальное из времен выполнения каждого локального процесса. Ниже приводится описание модели и результаты вычислительного эксперимента.

Модель вычислителя ILLIAC

Описание модели:

Модель представляет собой сильно упрощенную схему распределенных вычислений. Представлена сультипроцессорная 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.

Анализ модели:

На каждый вычислительный узел попадает несколько процессоров. В самом лучшем варианте число внешних связей в системе будет $¾ * N$, где N – количество процессоров. Поскольку в системе сообщения передаются асинхронно, и количество связей велико, следовательно, количество возможных временных парадоксов и, как следствие, откатов (для оптимистического алгоритма) будет достаточно большим.

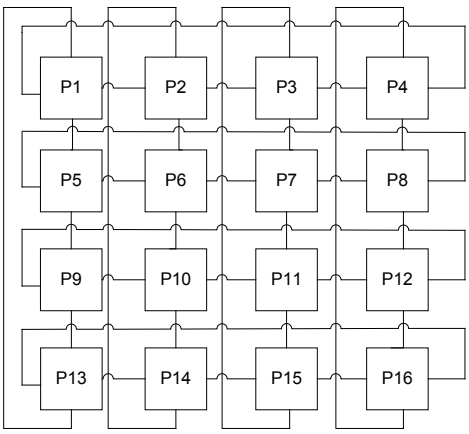

Рис. 1. Схема взаимодействия узлов в вычислительной системе ILLIAC

Эксперимент с моделью:

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

Результаты эксперимента с моделью "ILLIAC"

Таблица 1.

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

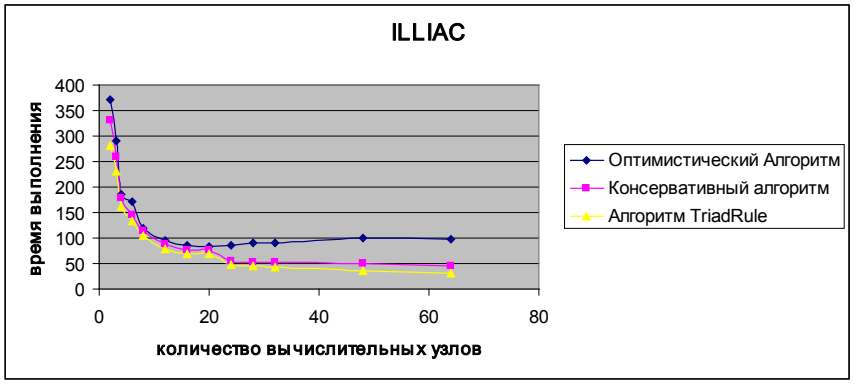

Рис. 1. Диаграмма Зависимости времени выполнения от количества узлов для модели ILLIAC

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. 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.

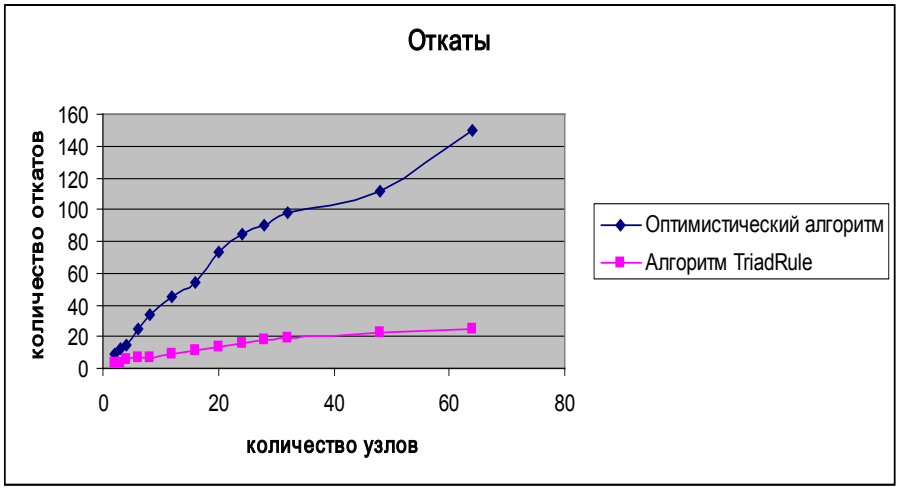

Рис. 2. Диаграмма зависимости количества откатов от количества узлов для модели ILLIACK.

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.

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:

Этапы выполнения алгоритма синхронизации, основанного на знаниях

Таблица 2.

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.

В работе подробно описывается организация подсистемы синхронизации объектов имитационной модели, проблемы сбора информации о модели во время имитационного прогона (с использованием механизма информационных процедур), описание вычислительного эксперимента с алгоритмом синхронизации TriadRule и приводятся результаты этого эксперимента. Во время эксперимента определяется время имитационного прогона. 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.

Библиографический список

[Fujimoto, 2003] 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

[Meyer, 1998] 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

[Окольнишников,2005] Окольнишников В.В. Представление времени в имитационном моделировании. Вычислительные технологии. Т. 10, №5, Сибирское отделение РАН, 2005, стр. 57-77

[Bryant,1977] Bryant R.E., Simulation of Packet Communications Architecture Computer Systems. MIT-LCSTR–188. 1977.

[Chandy, 1978] 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.

[Jefferson, 1990] Jefferson D.R., Virtual time II: storage management in distributed simulation Proc. of the Ninth Annual ACM Symposium on Principles of Distributed Computing. 1990. P. 75–89.

[Миков, 2008]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.

[Миков, 2009]Миков А.И., Замятина Е.Б., Козлов А.А. Оптимизация параллельных вычислений с применением мультиагентной балансировки // Труды международной научной конференции «Параллельные Вычислительные Технологии». Нижний Новгород – Челябинск, Изд. ЮУрГУ, 2009. С. 599-604.

[Mikov, 1995] 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. pp. 15-20.

Сведения об авторах

Елена Замятина *– Пермский государственный университет, доцент кафедры математического обеспечения вычислительных систем; Россия, г. Пермь, 614017, ул. Тургенева, 33–40; e-mail: e_zamyatina@mail.ru.*

Сергей Ермаков *– Пермский государственный университет, аспирант кафедры математического обеспечения вычислительных систем; Россия, г. Пермь, ул. Стахановская 7-52; e-mail: choufler@gmail.com.*