6. Zhang, Y. Fast tridiagonal solvers on the GPU / Y. Zhang, J. Cohen, J.D. Owens // Principles and Practice of Parallel Programming, 2010. – P.127–136.

7. Специализированный курс «Архитектура и программирование массивно-параллельных вычислительных систем» на основе технологии CUDA в МГУ им. Ломоносова. – URL: http://cuda.cs.msu.su.

8. NVIDIA GPU Computing Developer. – URL: http://developer.nvidia.com/object/gpucomputing.html.

С.А. Ермаков, Е.Б. Замятина, А.А. Козлов

Пермский государственный университет

ОПТИМИЗАЦИЯ РАСПРЕДЕЛЕННОГО ИМИТАЦИОННОГО ЭКСПЕРИМЕНТА ПО ВРЕМЕНИ И ПО НАДЕЖНОСТИ

Метод имитационного моделирования широко используется в качестве метода исследования сложных динамических систем в самых разных областях знаний: в бизнесе, производстве, здравоохранении, при проектировании компьютерных систем, летательных аппаратов и т.д. Системы имитационного моделирования часто используются в качестве ключевого компонента в системах поддержки принятия решения. Именно поэтому необходимо, чтобы результаты моделирования были достоверны, а следовательно, и процесс проведения имитационного эксперимента должен быть *надежным*, защищенным от сбоев, которые могут привести к потере важной информации. Поскольку объектом моделирования являются сложные системы, то и время моделирования может быть весьма и весьма значительным. По этой причине возникает настоятельная необходимость в *сокращении времени* имитационного прогона.

Сокращение времени имитационного прогона возможно, если при его проведении существует доступ к ресурсам сразу

нескольких вычислительных узлов. В этом случае имитационную модель разбивают на логические процессы, которые обмениваются информацией, посылая сообщения друг другу. Тем не менее при проведении распределенного имитационного эксперимента возникает ряд проблем, которые становятся причиной временных потерь. Первой причиной является организация *алгоритма синхронизации* распределенных по вычислительным узлам объектов имитационной модели, второй - возникновение *дисбаланса*.

В настоящей работе авторы предлагают решения, которые в той или иной мере позволяют снизить потери, связанные с проблемами синхронизации объектов модели и неравномерным распределением нагрузки на узлы вычислительной системы. Решения и в том, и в другом случаях связаны с использованием знаний исследователя о поведении имитационной модели. При разработке оригинальных алгоритмов синхронизации и динамического равномерного распределения нагрузки на вычислительные узлы авторы используют методы искусственного интеллекта: мультиагентный подход, онтологии, экспертные системы.

Разработка перечисленных выше алгоритмов выполнялась для системы имитационного моделирования Triad.Net, которая первоначально предназначалась для автоматизированного проектирования и моделирования вычислительных систем (ВС). По этой причине целесообразно предварительно рассмотреть особенности представления имитационной модели в Triad [1], тем более что организация процесса моделирования в Triad. Net допускает динамическое изменение структуры имитационной модели, разрешая исследователю выполнять операции над структурой модели и над моделью в целом.

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 полюсы (входные P_{in} и выходные P_{out}), 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 ($e_i \in E, i=1 \div n$, E – множество событий, множество событий рутины является частично упорядоченным в модельном времени). Выполнение события сопровождается изменением состояния объекта. Состояние объекта определяется значениями переменных рутины. Таким образом, система имитации является событийно-ориентированной. Рутина так же, как и объект, имеет входные (Pr_{in} и выходные Pr_{out}) полюсы. Входные полюсы служат соответственно для приёма сообщений, выходные полюсы – для их передачи. Совокупность рутин определяет слой рутин ROUT. Слой сообщений (MES) предназначен для описания сообщений сложной структуры.

Трехслойное представление модели позволяет исследователю изучать моделируемый объект по частям (например, только структуру), постепенно усложняя и детализируя модель. Кроме того, допускается изменение структуры модели во время имитационного прогона: добавление или удаление вершин, полюсов, дуг и ребер. Допускается также операция наложения рутины на вершину и другие операции над моделью в целом. В случае распределенного выполнения процесса имитации возможность *выполнения операций* над моделями являются дополнительными аргументами для проведения балансировки нагрузки на вычислительные узлы, поскольку вычислительная нагрузка на узлах будет произвольно изменяться.

Алгоритмом имитации называют совокупность объектов, функционирующих по определённым сценариям, и синхронизи-

рующий их алгоритм. Последовательный алгоритм является событийно ориентированным. Каждая рутина имеет свой локальный календарь событий ($tloc_i$). При выполнении очередного события выполняется поиск события с минимальным временем (поиск в календарях событий всех рутин) и управление передается рутине, которая включает это событие. Для сбора, обработки и анализа имитационных моделей в системе Triad.Net существуют специальные объекты – информационные процедуры и условия моделирования, реализующие *алгоритм исследования*. Информационные процедуры ведут наблюдение только за теми элементами модели (событиями, переменными, входными и выходными полюсами), которые указаны пользователем. Условия моделирования задают список информационных процедур, выполняющих наблюдение за элементами модели, анализируют результат работы информационных процедур и определяют, выполнены ли условия завершения моделирования.

Имитационная модель, описанная на языке Triad, может быть запущена на моделирование с помощью специального оператора *simulate*: *simulate* <список элементов моделей, за которыми установлено наблюдение> *on conditions of simulation* <наименование> (список фактических настроечных параметров>)[<список входных и выходных фактических параметров интерфейса>].

Следует отметить, что исследователь может одновременно запустить на моделирование сразу несколько моделей, указав элементы, за которыми будут следить информационные процедуры, перечисленные в условиях моделирования. Кроме того, можно указать сразу несколько условий моделирования. В этом случае каждая модель или ее копия могут выполняться на отдельном вычислительном узле, как на ВС с кластерной архитектурой, так и на ВС с GRID-архитектурой, параллельно.

Алгоритм 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.

Задача консервативного алгоритма – определить время, когда обработка очередного события из списка необработанных событий является «безопасной». Событие является безопасным, если можно гарантировать, что процесс в дальнейшем не получит от других процессов сообщение с меньшей временной меткой. В отличие от консервативных алгоритмов, не допускающих нарушения ограничения локальной каузальности, оптимистические методы не накладывают такого ограничения. При нарушении локальной каузальности (логический процесс получает событие, имеющее временную отметку меньшую, чем уже обработанные события) происходит восстановление правильного порядка событий [2, 3]. Самый известный алгоритм – Time Warp. Для восстановления порядка выполняется откат, последующие события обрабатываются повторно в хронологическом порядке. Откатываясь назад, процесс восстанавливает состояние имитационной модели, которое было до обработки событий (все состояния модели сохраняются), и отказывается от сообщений, отправленных в результате обработки событий, которые необходимо отменить в результате отката. Для отказа от этих сообщений разработан механизм антисообщений.

Для некоторых специфических моделей консервативные алгоритмы показывают производительность, превосходящую оптимистические в несколько раз, для других моделей выигрыш во времени дает оптимистический алгоритм. Анализ консервативного и классического алгоритмов показывает, что для увеличения производительности консервативного алгоритма необходимо расширить горизонт безопасных событий, а для оптимистического – сократить количество откатов, сдержав чрезмерное продвижение модельного времени. Существует большое количество модификаций алгоритмов синхронизации [4]. Все они предназначены для оптимизации алгоритма по времени.

Так или иначе, если алгоритм синхронизации располагает информацией о модели (lookahead, lookback, временная метка следующего события и т.п.), то производительность его повышается. Таким образом, для оптимизации времени выполнения распределенного имитационного эксперимента целесообразно использовать информацию о модели, в том числе и знания исследователя о конкретной модели, что позволит адаптировать алгоритм синхронизации в каждом конкретном случае. Авторы предлагают использовать для хранения информации исследователя о модели продукционные правила. Кроме того, знания должны извлекаться из самой модели как во время ее трансляции, так и во время ее функционирования.

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. Итак, пользователь обычно имеет представление о поведении модели. В результате могут быть построены продукционные правила 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_n. CF – коэффициент доверия. 0 – нет доверия, 100 – максимальное доверие. Правила отражают каузальную зависимость между событиями, а поскольку знания не являются точными, то каждому правилу приписывают коэффициент доверия.

Однако детальное описание поведения модели в виде правил достаточно затруднительно. Необходимо извлекать информацию о поведении модели автоматически. Эту информацию также будем хранить в виде правил (системные правила). Системные правила формируются на стадии трансляции (процедуры обработки событий и сообщений) и на стадии выполнения модели (анализ последовательности обработки событий). Для сбора, обработки и распространения информации алгоритма синхронизации целесообразно реализовать специальных агентов на вычислительных узлах. Информация (центральная база знаний) и главный управляющий процесс нахо-

дятся на выделенном сервере. Агент сбора и обработки информации определяет безопасное событие следующим образом: происходит поиск события в заключениях всех правил в базе знаний, а далее, если событие обнаружено, то выполняется логический вывод по правилам. Далее выбирают событие с максимальным коэффициентом доверия.

Проведенные испытания (для разных моделей и на ВС с различными конфигурациями) показали, что алгоритм дает выигрыш во времени. Особенно удачные результаты были получены для модели клиент-сервер (15 %).

Балансировка нагрузки на вычислительные узлы. Распределенная имитационная модель представляет собой совокупность объектов Triad-модели, которые располагаются на различных вычислительным узлах ВС и обмениваются информацией друг с другом. Распределение объектов выполняется на этапе статической балансировки исходя из структурных особенностей модели (клики располагаются на одном узле). При функционировании модели может возникнуть дисбаланс (гетерогенность вычислительных узлов и линий связи, гетерогенность имитационной модели). Восстановление равномерного распределения нагрузки выполняется динамической системой балансировки.

Существует большое количество алгоритмов, решающих эту задачу, но многие из них является «жесткими», при изменении модели или условий моделирования, алгоритма синхронизации, алгоритмы работают менее эффективно или вообще не дают никакого эффекта [5]. Авторы настоящей работы предлагают учитывать и особенности имитационной модели, и условий моделирования, и особенности вычислительной среды. Динамическая система балансировки TriadBalance является *мультиагентной*, она состоит из совокупности агентов разных типов: агента-датчика вычислительного узла; агента-датчика имитационной модели; агента анализа; агента миграции; агента распределения. Агенты каждого типа действуют по своему сценарию для достижения цели, а вместе они реализуют балансировку распределенной имитационной модели.

Алгоритм балансировки является децентрализованным и представляет собой последовательность шагов. 1. Агенты-датчики ВС (они располагаются на каждом вычислительном узле) регулярно на всем протяжении имитационного эксперимента собирают статистику о загруженности узлов, о состоянии линий связи, о наличии свободной памяти и размещают ее в базе данных (доска объявлений). Каждый вычислительный узел располагает своей доской объявлений. 2. Агент анализа, сканируя доски объявлений, с помощью правил из базы знаний определяет, необходимо ли выполнять балансировку. Например, при превышении показателя загрузки вычислительного узла агент анализа принимает решение о необходимости выполнения балансировки. В этом случае агент анализа обращается к агенту распределения. 3. Агент распределения извлекает информацию из базы данных. Информация в базе данных появляется в результате работы агента-датчика имитационной модели. Для извлечения информации о модели используют механизм информационных процедур. В частности, агенты-датчики собирают информацию о частоте выполнения событий имитационных объектов, частоте обменов между ними (частота появления сообщений на входных и выходных полюсах рутин). Таким образом, агент распределения на основании анализа данных на доске объявлений может сделать вывод о том имитационном объекте, который следует перенести на другой узел. Кроме того, он общается с агентами распределения, расположенными на соседних вычислительных узлах. Он сообщает о перегрузке, о том, что ему необходимо освободиться от некоторых своих имитационных объектов. Агенты распределения других вычислительных узлов извещают о своей нагрузке, пополняя базу данных агента распределения, корректируя правила в базе знаний этого агента. Во время сеанса взаимодействия агенты распределения других узлов могут сообщить о том, что они незагружены. Далее агент распределения действует по правилам, которые хранятся в базе знаний. Для принятия решения об адресе целевого узла агент распределения, наряду с другими правилами, использует, в ча-

стности, правила топологических характеристик ИМ и ВС. При выборе целевого узла сообщение направляется агенту миграции. 4. Агент миграции, получив запрос от агента распределения, выполняет непосредственно перенос выбранных объектов имитационной модели на выбранные целевые узлы сети.

Для повышения гибкости мультиагентной системы балансировки, адаптируемости когнитивных агентов к изменяющимся условиям используют метаправила, более точно – два типа метаправил: метаправила, определяющие структуру правил (их используют агенты распределения и анализа для распознавания нужных правил), и метаправила, определяющие, как можно изменить правила.

Настоящая версия мультиагентной подсистемы динамической балансировки реализована на кластере из 8 вычислительных узлов, работающих под управлением операционной системы Windows HPC Server 2008, которая является специализированной кластерной операционной системы Windows Server 2008. Подсистема балансировки разработана с использованием Net Framework 3.5 и пакета сборок HPC Pack 2008, позволяющего управлять распределённым выполнением приложений. Эксперименты показали (для модели «Клиент-сервер»), что время, затраченное на имитационный эксперимент при использовании подсистемы мультиагентной балансировки, действительно снижается, эффективность применения балансировки увеличивается с увеличением длительности эксперимента.

Надежность. Для обеспечения отказоустойчивости распределенной системы имитации (византийская модель сбоев) также была разработана специальная мультиагентная подсистема. Византийская модель сбоев (или модель произвольных сбоев) является наиболее общей моделью. Она позволяет моделировать такие типы сбоев, как отказы оборудования, потеря связи, неисправность оборудования, программные сбои, ошибки операторов и действия злоумышленников. Мультиагентная подсистема отказоустойчивости состоит из агента-детектора, агента анализа, агента репликаций и агента переноса кода (располага-

ются на каждом вычислительном узле). Аналогично подсистеме балансировки подсистема отказоустойчивости использует знания о модели, знания о состоянии вычислительных узлов и т.д. для выбора наилучшей стратегии в условиях сбоя (в отличие от многих существующих систем отказоустойчивости, которые предлагают ряд оптимизаций, но не указывают области, где их применение наиболее эффективно). Подсистема отказоустойчивости тесно взаимодействует с подсистемой балансировки и подсистемой визуализации.

Итак, в работе представлены программные решения, основанные на мультиагентном подходе и применении экспертных систем, которые позволяют оптимизировать распределенный имитационный алгоритм. Проведенные эксперименты подтвердили эффективность разработанных программных средств.

Список литературы

1. 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.

2. Fujimoto R.M. Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference Chick S., Sánchez P.J., Ferrin D., Morrice D. J. eds., pp. 124–134.

3. Окольнишников В.В. Представление времени в имитационном моделировании // Вычислительные технологии. Т. 10, № 5; Сибирское отделение РАН. – 2005. – С. 57–77.

4. Вознесенская Т.В. Математическая модель алгоритмов синхронизации времени для распределённого имитационного моделирования // Программные системы и инструменты: темат. сб. факультета ВМиК МГУ им. Ломоносова. № 1. – М.: Изд-во МГУ, С. 56–66.

5. Wilson L.F. and Shen W. Experiments In Load Migration And Dynamic Load Balancing In Speedes // Proceedings of the 1998 Winter Simulation Conference. D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds. – Washington, 1998. – P. 483–490.