Построение распределенных дискретно-событийных моделей в среде AnyLogic

М.А. Кондратьев, М.В. Гарифуллин

В настоящий момент широко используется компьютерное имитационное моделирование, зачастую требующее значительного количества вычислительных ресурсов и времени. Для уменьшения времени моделирования можно попытаться распараллелить выполнение модели. Данная работа посвящена разработке программных средств, позволяющих строить распределенные дискретно-событийные модели (в первую очередь агентные модели) в отечественной среде AnyLogic на основе технологии Java RMI, а также исследованию различных методов синхронизации распределенных моделей. В ней представлена техника разбиения модели на несколько частей и синхронизация их работы с помощью оптимистического алгоритма логического времени. Получено значительное увеличение скорости выполнения агентной модели за счет разнесения моделирования групп агентов, не имеющих постоянного взаимодействия, по различным вычислительным потокам.

1. Введение

Имитационное моделирование в наши дни широко применяется для решения разнообразного круга задач. Компьютерное имитационное моделирование - это имитация поведения объектов реального мира с помощью компьютерной программы. Существует множество программ для описания и выполнения моделей. Часть этих программ создана для определенных предметных областей, часть – для создания моделей общего назначения. Независимо от выбора программы, выполнение или имитация модели требует определенных вычислительных ресурсов и занимает время. Требования к объему, точности и функционалу современных имитационных моделей непрерывно возрастают и часто обгоняют технические возможности инструментов для создания моделей. Для таких моделей невозможно выполнить имитацию в разумные сроки, поэтому специалисты используют подходы, позволяющие уменьшить время моделирования. Одним из подходов является уменьшение точности модели, что приводит и к уменьшению точности результатов. В случае, когда уменьшение точности неприемлемо, можно воспользоваться подходом распараллеливания выполнения модели.

При распределении отдельных частей модели по сети компьютеров части модели выполняются отдельными компьютерами. Теоретически, при условии независимости компонент модели, выигрыш в производительности прямо пропорционален количеству рабочих станций. Если же компоненты модели обмениваются информацией, то ко времени моделирования прибавляется время на осуществление взаимодействия.

Все наиболее часто используемые инструменты для имитационного моделирования, такие как Arena, Extend, VenSim и др. являются однопоточными приложениями. Поэтому для создания распределенной модели специалисту приходится переносить модель из удобного инструмента разработки моделей в неудобную и чуждую среду программирования. Перенос модели заключается в переносе описания объекта, а также в обеспечении взаимодействия распределенных компонент. Для специалистов по построению имитационных моделей эти задачи стоят далеко от их профессиональной области.

Целью данного проекта является преодоление трудностей, связанных с распараллеливанием моделей, за счет добавления средств создания распределенных моделей непосредственно в инструмент имитационного моделирования. Это позволит специалисту сфокусировать внимание собственно на создании модели, а не на программировании. От него потребуется только логически разделить модель на части с небольшим объемом взаимодействий. Далее, путем использования библиотечных элементов и простой настройки, такая модель может быть запущена на множестве рабочих станций.

2. Выбор инструмента имитационного моделирования

Большинство инструментов имитационного моделирования созданы для конкретных областей применения: производство, управление финансами, оптимизация складской логистики. Существует несколько гораздо более гибких инструментов общего назначения, с помощью которых можно описать любые модели. Мы бы не хотели ограничивать множество распределяемых моделей какой-то определенной предметной областью, поэтому выбор инструмента имитационного моделирования осуществлялся из систем общего назначения.

Вторым критерием для выбора инструмента является гибкость. Для создания библиотеки средств распараллеливания моделей необходимо иметь в самом инструменте собственно возможность создания библиотек, а также достаточные средства для описания библиотечных элементов. Во многих инструментах разработчик ограничен небольшим множеством функций. А для функции распределения модели в инструменте как минимум должны быть средства обмена данными по сети.

В результате исследования инструментов имитационного моделирования по двум критериям (инструмент общего назначения и гибкость) была выбрана программа AnyLogic. AnyLogic основан на языке Java и является инструментом общего назначения.

3. Синхронизация распределенных частей модели

Важнейшим аспектом при построении распределенной (параллельной) модели является синхронизация отдельных ее частей. Пакет AnyLogic – среда для создания преимущественно дискретно-событийных моделей. В данном проекте будет рассматриваться только распределение частей дискретно-событийных моделей, так как при попытке синхронизации моделей, оперирующих с непрерывными сущностями, возникают совершенно иные проблемы, выходящие за рамки данного исследования.

Исполнение компьютерной дискретно-событийной модели предполагает наличие некоторого множества переменных состояния, характеризующих модель в текущий момент модельного времени, а также списка запланированных событий, которые должны произойти в модели в будущем. В подавляющем большинстве случаев, для дискретно-событийных моделей, разрабатываемых в среде AnyLogic, имеет значение, в каком порядке будут обрабатываться события, происходящие в модельном времени. То есть в том случае, если какое-либо событие произойдет раньше или позже, чем это было запланировано создателем модели, результаты моделирования могут оказаться неверны. Кроме того, как правило, среда моделирования должна обеспечивать такое свойство, как «повторяемость» результатов моделирования (т.е. результаты «прогонов» одной и той же модели с одинаковыми начальными данными должны совпадать).

В случае последовательного (однопоточного) моделирования в пакете AnyLogic, среда моделирования позаботится о том, чтобы происходящие события были упорядочены правильным образом. Когда же мы прибегаем к распределенному (параллельному) моделированию, мы имеем несколько подмоделей, и у каждой из них будет свое текущее модельное время и список событий. Одна подмодель может общаться с другой, например, помощью передачи сообщений. Результатом получения такого сообщения будет возникновение в модели некоторого события. Сложности возникают, если к моменту получения сообщения, предполагающего возникновение события в момент времени t, модельное время уже «ушло» вперед до некоторого момента $t + \Delta t$. Будем называть такие ситуации «парадоксом времени» [1].

Для того чтобы избежать «парадоксов времени», необходима реализация специальных алгоритмов синхронизации. Существуют два основных класса алгоритмов синхронизации модельного времени: консервативные и оптимистические [2, 3, 4].

Задачей консервативных алгоритмов является предотвращение парадоксов времени. Консервативные алгоритмы предполагают, что между подмоделями используются FIFO каналы передачи данных, то есть адресат получает сообщения в том порядке, в котором их посылал отправитель. Это означает, что если подмоделью в момент времени t получено сообщение,

отправленное адресантом в момент модельного времени $t + \Delta t$, она не получит в интервале Δt другого сообщения от данного отправителя и может продолжать вычисления в интервале времени Δt не «опасаясь» возникновения «парадокса времени». В том случае, если от какого-либо адресанта нет сообщений, то для предотвращения «парадокса времени» вычисления следует временно остановить до их получения.

Основным недостатком консервативных алгоритмов является то, что приостановка вычислений серьезно снижает эффективность использования параллелизма. Кроме того, возникают тупиковые ситуации (deadlocks), когда подмодели, ожидающие друг от друга сообщения, образуют замкнутый круг и исполнение всей распределенной модели останавливается. Разрешение тупиковых ситуаций требует дополнительных усложнений алгоритма.

Оптимистические (агрессивные) алгоритмы основаны на механизме «откатов» (rollbacks). В то время как консервативные алгоритмы предотвращают возникновение «парадоксов времени», оптимистические не придают значения возможности их образования до их возникновения. В том случае, если в процессе исполнения модели обнаруживается подобная ситуация, оптимистические алгоритмы запускают некоторый алгоритм компенсации. Наиболее употребляемыми является алгоритмы, периодически сохраняющие переменные состояния модели. В случае возникновения «парадокса времени» это позволяет вернуться к некоторому моменту вычислительного эксперимента в прошлом с целью повторного исполнения последующих событий с учетом вновь пришедшего.

Проблема оптимистического алгоритма заключается в том, что при «откате» следует каким-то образом отменить уже посланные другим подмоделям сообщения (для этого используется отправка «антисообщений»), которые, в свою очередь, могут вызвать откаты в других подмоделях. При появлении таких ситуаций возможны значительные вычислительные потери.

Следует особо рассмотреть такой прием, как нестрогое упорядочение событий в модели (relaxed synchronization). Он уместен в том случае, если для модели не имеет особого значения порядок возникновения событий. При использовании такого приема мы задаем некоторый промежуток времени, считая, что события, возникающие в течение этого промежутка, «достаточно близки», и поэтому не имеет значения, в какой последовательности их обрабатывать. Хотя использование этого приема не позволит обеспечить «повторяемость» экспериментов с моделью, однако в некоторых случаях это позволит серьезно улучшить эффективность оптимистических алгоритмов.

Таким образом, существует множество различных алгоритмов синхронизации. Для каждой конкретной модели максимальной эффективности использования параллелизма позволяет достичь свой алгоритм, и именно от него зависит тот выигрыш, который можно получить в результате замены последовательной модели распределенной.

4. Пример распределения агентной модели

Таким образом, целью настоящего исследования является создание библиотеки программных средств для построения распределенных моделей, синхронизируемых различными способами. Это позволит разработчику модели подобрать для ее синхронизации наиболее эффективный метод.

В нашем проекте создание библиотеки для распределения частей модели мы начали для такого подкласса дискретно-событийных моделей, как агентные. Общепризнанного определения агентных моделей не существует; до сих пор спорят о том, какими же качествами должен обладать объект, чтобы «заслужить» называться агентом. Однако есть нечто, объединяющее все агентные модели: они существенно децентрализованы. В них нет такого места, где централизованно определялось бы поведение (динамика) системы в целом. Вместо этого аналитик определяет поведение на индивидуальном уровне, а глобальное поведение возникает как результат деятельности многих (десятков, сотен, тысяч, миллионов) агентов, каждый из которых следует своим собственным правилам, живёт в общей среде и взаимодействует со средой и с другими агентами. Такой подход к имитационному моделированию хорошо подходит для создания, например, моделей сложных экономических

или социальных систем. В то же время, именно агентные модели нуждаются в наибольшем количестве вычислительных ресурсов (нередко при агентном подходе для построения адекватной модели требуется использование огромного количества агентов), а значит, разрабатываемая библиотека потребуется в первую очередь для создания распределенных агентных моделей.

Выполнена работа по созданию распределенных агентных моделей с помощью Java RMI в среде AnyLogic 6. Создание распределенной модели рассмотрим на примере классической для применения агентного подхода к имитационному моделированию задачи – модели распространения заболеваний. Предположим, что существуют три города, часть населения которых (агенты) совершает перелеты из города в город через фиксированные промежутки времени. Агенты в каждом городе находятся в некотором метрическом пространстве (Рис. 1) и могут быть здоровы («Susceptible»), больны («Infectious») или идти на поправку («Recovered»). Находясь на определенном расстоянии друг от друга, больной агент может заразить здорового (переход «Contact»).

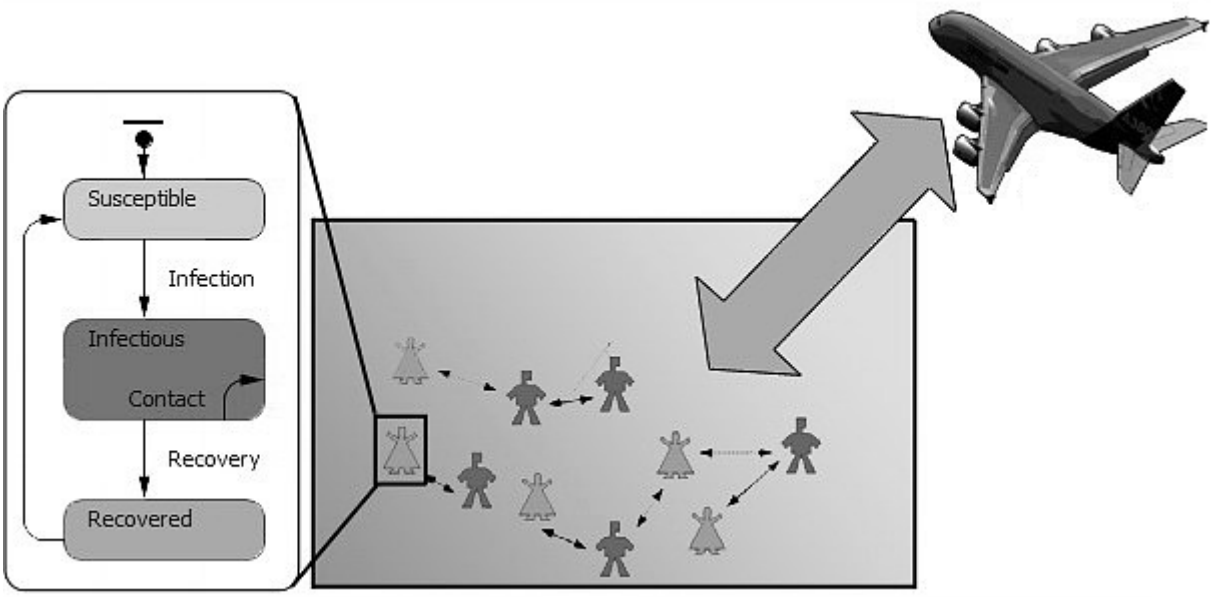

Рис. 1. Модель поведения агентов в одном городе

Имея такую модель, можно ставить различные задачи. В данной работе мы наблюдали за распространением инфекции, которой изначально был заражен один агент в одном городе. Критерием завершения вычислений было достижение определенного значения модельного времени.

К сожалению, моделирование поведения большого числа агентов в метрическом пространстве требует серьезных вычислительных затрат, и уже начиная с нескольких тысяч агентов, вызывает затруднения. Количество вычислений, необходимое для таких моделей, растет полиномиально с увеличением числа агентов. Если же мы хотим промоделировать поведение реального города (пусть даже маленького), агентов потребуются сотни тысяч.

Поэтому, обычную последовательную модель такого процесса, реализованную в среде AnyLogic 6, следует для уменьшения времени «прогона» попытаться распараллелить. В данной работе модель была разбита на три части (каждая часть включала один город с агентами), каждая подмодель выполнялась в своем вычислительном потоке. Перелет самолета с агентами из города в город был представлен с помощью посылки соответствующего сообщения (используя Java RMI), для синхронизации подмоделей был использован алгоритм «деформации времени» (Time Warp) [5].

Рассмотрим полученные результаты. На рисунке 2 изображена зависимость времени, потребовавшегося на проведение вычислений для описанной модели, от числа агентов, населяющих каждый город, в случае ее однопоточного варианта и параллельного выполнения (в трех потоках). Видно, что использование параллелизма позволило получить выигрыш для описанной модели в 7-9 раз. Его величина объясняется особенностями обработки

взаимодействия агентов в среде AnyLogic, требующая, например, обновления состояний всех агентов в случае взаимодействия нескольких из них. Все вычисления проводились на двуядерном процессоре, поэтому можно ожидать даже лучших результатов в случае применения процессора с большим числом ядер.

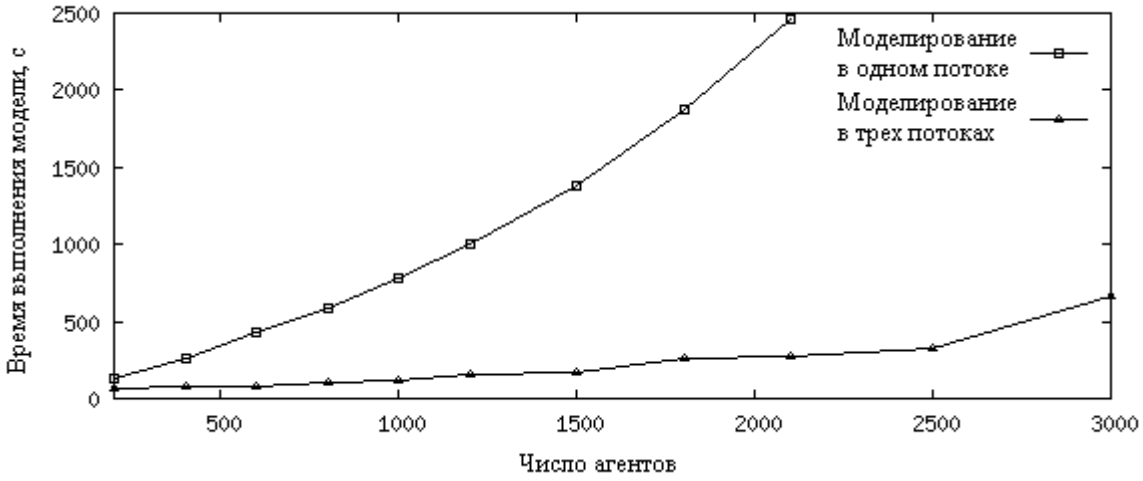

Рис. 2. Зависимость времени выполнения модели от числа агентов

Данная модель является хорошим примером эффективного применения оптимистического алгоритма синхронизации. Но результаты, показанные распараллеливанием в данном случае, будут настолько хороши не всегда. Использование оптимистического алгоритма вынуждает периодически сохранять все переменные состояния (в данной модели это происходило при отправке очередного сообщения – отлете самолета), в том числе для всех агентов. При росте числа агентов будут расти затраты на хранение этих переменных (памяти), а также вычислительные затраты на восстановление их значений в случае «отката». И, хотя даже в таком случае выгоднее использовать распределенные вычисления, нежели однопоточные, в какой-то момент (при определенном числе агентов) следует заменить алгоритм синхронизации с оптимистического на консервативный.

Обобщая, можно говорить о том, что модели, требующие большого количества вычислений, но ограниченного количества памяти, можно эффективно синхронизировать с помощью оптимистических алгоритмов, в то время как модели, занимающие значительное количество памяти по сравнению с вычислительными затратами на обработку происходящих в них событий, следует синхронизировать с помощью консервативных алгоритмов.

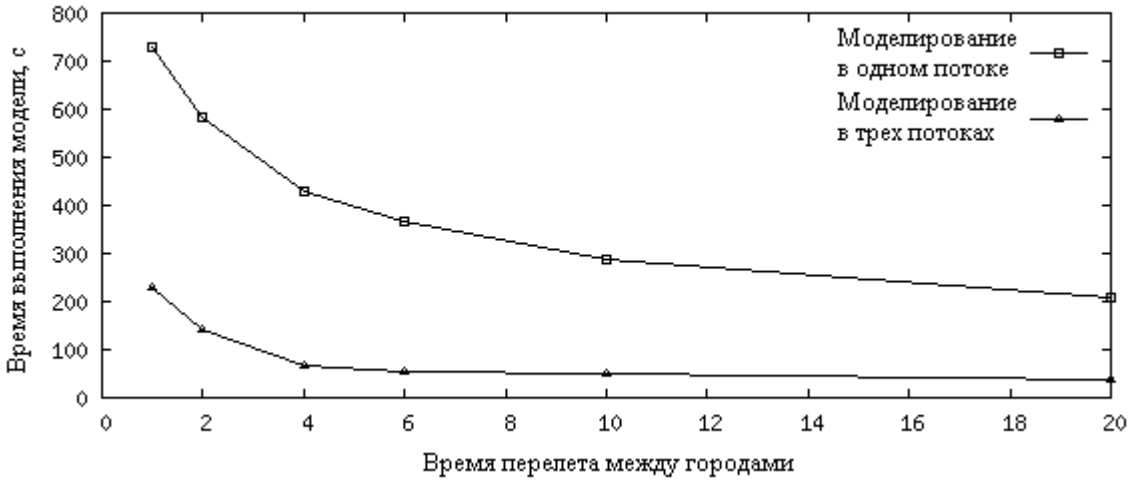

Рис. 3. Зависимость времени выполнения модели от времени полета между городами

Другой причиной хороших результатов, показанных на этой модели, стала такая естественная особенность модели, как учет времени перелета самолетов между городами, то есть, если самолет вылетал в момент времени t, события, связанные с его прилетом должны были бы произойти в момент времени $t + \Delta t$. Величина Δt позволила избежать частого возникновения «откатов» из-за небольших рассогласований в модельном времени разных городов. Кстати, эта особенность модели позволяет также эффективно использовать для ее синхронизации консервативные алгоритмы. График зависимости времени «прогона» модели от величины Δt изображен на рисунке 3. Предполагалось, что расстояние, а значит и время перелета, между городами равное. Значительное улучшение времени, затрачиваемого на моделирование, с увеличением этой величины объясняется не только улучшением работы алгоритма синхронизации, но скорее тем, что в модели становится гораздо меньше событий, связанных с прилетом/улетом самолетов (количество самолетов – фиксированное, не прилетевший самолет не может улететь).

5. Заключение

Итак, распараллеливание рассмотренной модели показало хорошие результаты, однако их можно улучшить. Например, можно использовать нестрогое упорядочение событий (relaxed synchronization). Предположим, что событие прибытия самолета в город получено слишком поздно и требует «отката». Но ведь в том случае, если разница между временем прибытия этого самолета и модельным временем города не очень велика, можно, не обращая внимания на возникновение «парадокса времени», обработать событие прибытия нового самолета. Для данной модели такой подход вполне приемлемо объяснить задержкой самолета в пути, например, из-за погодных условий. В то же время, это позволит избежать вычислительно дорогой операции «отката».

Не для всех агентных моделей эффективным будет использование оптимистических алгоритмов и сохранение переменных состояния только при отправке сообщений. Например, может оказаться, что сообщений между подмоделями слишком мало и сохранение состояния происходит слишком редко. В этом случае при возникновении «откатов» будет теряться значительная часть вычислений, которые можно было бы и не повторять. В таких ситуациях удачным выходом станет дополнительное сохранение переменных состояния в промежутках между отсылкой сообщений.

Таким образом, требуется дальнейшее развитие полученных результатов, включающее в себя как исследование применения консервативных алгоритмов синхронизации к рассмотренной модели, так и доработку уже реализованных алгоритмов (например, использование нестрогого упорядочения событий), а также проверку полученных результатов на других моделях.

Создана часть библиотеки элементов (блоков), обладающей графическим интерфейсом, позволяющей создавать в среде AnyLogic распределенные модели и применять некоторые из описанных выше методов синхронизации.

Рассмотренные методы построения распределенных моделей позволяют добиться серьезного увеличения производительности не только при запуске модели на нескольких рабочих станциях, но, как это продемонстрировала приведенная модель, также и при использовании многоядерной архитектуры современных процессоров.

Литература

1. Окольнишников В.В. Представление времени в имитационном моделировании // Вычислительные технологии. –2005. –Т. 10, № 5. –С. 57–80.
2. Fujimoto R.M. Parallel discrete event simulation // Proc. of the Winter Simulation Conf. –1989. – P. 19–28.
3. Fujimoto R.M. Distributed simulation systems // Proc. of the Winter Simulation Conf. –2003. – P. 124–134.

4. Perumalla K.S. Parallel and distributed simulation: traditional techniques and recent advances // Proc. of the Winter Simulation Conf. –2006. –P. 84–95.
5. Jefferson D.R. Virtual time // ACM Transactions on Programming Languages and Systems. – 1985. –Vol. 7, N. 3. –P. 404–425.