УДК 519.688

А. Н. Мягков[1], *Ю. И. Бродский*[2]

[1]Московский физико-технический институт (государственный университет)
[2]Вычислительный центр им. А.А. Дородницына РАН

Об управлении временем в распределённых имитационных моделях

Обсуждаются вопросы, касающиеся алгоритмов синхронизации модельного времени в таких сложных распределённых имитационных моделях, для компонент (или отдельных агентов) которых известны определённые правила их взаимодействия между собой, позволяющие в некоторых случаях оптимизировать алгоритмы управления временем. Рассматриваются также вопросы общей эффективности таких алгоритмов.

Ключевые слова: управление временем, декларативная парадигма программирования, компонента, событие.

1. Проблема времени в распределённых имитационных системах

Вначале приведём общие сведения об управлении временем в распределённых моделях. В большинстве работ, касающихся имитационного моделирования, различают три категории времени:

• физическое время, т.е. реальное время, характеризующее процессы в реальной (физической) системе, которая описывается моделью. Как правило, это либо обычное мировое время, либо любое другое время, состоящее из интервалов, характерных для исследуемой системы;

• модельное время, т.е. отображение физического времени в виртуальное, использующееся имитационной системой. Модельное время может течь как быстрее, так и медленнее физического времени, в зависимости от назначения модели и требований, предъявляемых к имитационному процессу;

• процессорное время, т.е. фактическое время выполнения модели в вычислительной машине (на компьютере). Модельное и процессорное время могут быть синхронизированы (например, для моделей-тренажёров реального времени), но в большинстве случаев модельное время течёт значительно быстрее процессорного, поскольку сам процесс моделирования должен выполняться как можно быстрее.

В случае последовательного (нераспределённого) моделирования все три категории времени однозначно соотносятся между собой, и задачей имитационной системы является просто продвижение модельного времени. В случае распределённого моделирования этот процесс оказывается гораздо более сложным из-за возникновения так называемых парадоксов времени.

Рассмотрим компонентно-событийную распределённую модель, описываемую в [1, 2]. Каждая компонента представляет собой логический процесс, который обладает собственной функциональностью (методами) и взаимодействует с другими компонентами путём передачи сообщений о некоторых событиях, происходящих в системе. События определяются как функции значений внутренних и внешних переменных компоненты в начале шага моделирования. Программно событие реализуется как метод, входными параметрами которого является подмножество внутренних и внешних характеристик компоненты, а выходной параметр один: прогнозируемое время до наступления этого события. Если это прогнозируемое время равно нулю — значит, событие уже наступило. Таким образом, каждому событию в соответствие ставится время его возникновения.

Для того чтобы выполнение имитационной модели в распределённой системе давало такие же результаты, как и в случае выполнения в последовательной системе, а также

для того чтобы результаты модели были повторяемы (т.е. чтобы имитационные эксперименты давали одинаковые результаты при последовательных прогонах с одним и тем же набором исходных данных), необходимо, чтобы все события всех компонент выполнялись в хронологическом порядке (в терминах хронологии модельного времени).

Таким образом, работа управляющей программы системы распределённого имитационного моделирования заключается в том, чтобы выбрать из списка необработанных событий событие с минимальной отметкой времени и обработать его. При этом изменяются значения характеристик одной или нескольких компонент, что в свою очередь может привести к возникновению новых событий. Именно из-за этого в условиях, когда различные компоненты выполняются на разных узлах распределённой системы моделирования, задача выполнения событий в хронологическом порядке не является тривиальной.

Рассмотрим простейший пример конкурентного использования ресурсов: транзакционное проведение валютных операций. Пусть несколько компонент последовательно производят снятие средств с валютного счёта, после чего сообщают валютной организации (банку) о произведённой операции. Для каждой компоненты банк производит проверку: если средств достаточно, то операция выполняется, в противном случае банк отклоняет её.

В распределённом варианте описанной модели, когда все компоненты выполняются независимо, может возникнуть такая ситуация, что сообщение от одной из них поступит в банк позже, чем другая компонента проведёт собственную операцию, в результате чего проверка наличия достаточного количества средств на счёте будет неверной. Схематично такое поведение изображено на рис. 1.

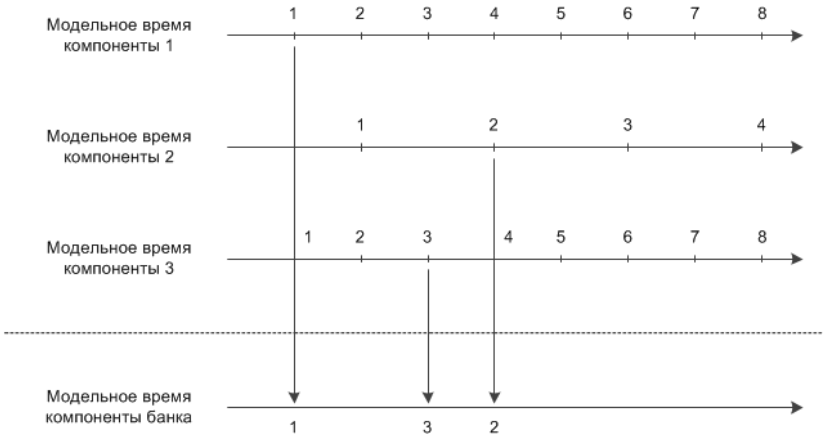

Рис. 1. Возникновение «парадокса времени»

Технически такое может возникнуть из-за разной скорости течения модельного времени для разных компонент, например, в случае, если вычисления для одной из компонент производятся на медленном или загруженном процессоре, в то время как другие компоненты «уходят» вперёд. Именно такие ситуации и называются парадоксами времени, и их преодоление является важной задачей распределённого имитационного моделирования.

На сегодняшний день разработаны и широко применяются два класса алгоритмов синхронизации модельного времени: консервативный и оптимистический. В следующих разделах будут описаны оба этих алгоритма. Также обсуждаются положительные и отрицательные стороны каждого из них и возможные пути их оптимизации. Но прежде введём понятия императивной и декларативной парадигм программирования. Под парадигмой программирования в соответствии с [3] понимается набор представлений о некотором классе программных систем, допускающих реализацию с помощью этой парадигмы. Императивное программирование — это самый распространенный подход к написанию программ на языках программирования типа C или Java, при котором программа описывается как последовательность элементарных инструкций, которые должна выполнить вычислительная машина. Таким образом, если применяется императивное программирование, то описывается последовательность действий, которой должно быть достаточно для получения результа-

та исследования. Сам результат при этом не предполагается известным заранее, наоборот, скорее всего, целью императивной программы и является получение этого результата.

При декларативном программировании, наоборот, описывается то, каким является известный заранее результат (или каким должен являться желаемый результат). При этом выбор последовательности действий, приводящей к описанному в системе декларативного программирования результату, как правило, оставляется на усмотрение соответствующего компилятора или интерпретатора.

2. Консервативные алгоритмы управления временем

Алгоритмы управления временем основаны на концепции списка (календаря) запланированных событий. Как было упомянуто выше, каждое возникающее в модели событие имеет выходную характеристику — прогнозируемое время до наступления данного события. На основании этих временных меток управляющая программа имитационной модели может в любой момент построить упорядоченный по времени список событий, которые возникают (т.е. планируется, что возникнут) при дальнейшем выполнении модели.

Задачей консервативного алгоритма является определение времени, когда обработка очередного запланированного события будет безопасной, т.е. когда можно быть уверенным в том, что ни одна компонента не запланирует события с меньшей меткой времени. Реализация алгоритма основана на вычислении LBTS (Lower Bound of the Time Stamp — нижней границы временных меток) всех прогнозируемых событий. За счёт сравнения текущего модельного времени с временем LBTS управляющая программа может определить, обработка каких событий является безопасной. Обработка всех небезопасных событий приостанавливается до тех пор, пока они не станут безопасными.

Очевидно, такой подход приводит к снижению эффективности параллельных (распределённых) вычислений, ведь в данном случае скорость выполнения всей модели будет ограничена скоростью вычисления самой медленной компоненты в каждый момент времени. Кроме этого, временная приостановка выполнения компонент может привести к тупиковым ситуациям (аналогичным deadlock в реляционных СУБД), когда ожидающие компоненты образуют замкнутую цепь и выполнение модели останавливается. В данной статье мы не будем подробно останавливаться на проблеме тупиковых ситуаций; подробно они исследуются в [4, 5]. Будем также считать, что существует конечное разбиение временного отрезка моделирования событиями. Вместо этого рассмотрим возможности оптимизации алгоритма управления временем в целях увеличения общей эффективности распределённых систем имитационного моделирования.

Большинство существующих способов оптимизации алгоритмов управления временем (например, алгоритм с нулевыми сообщениями, использование дополнительной информации о временной метке следующего события и т.д., см. [4, 5]) строится на общих принципах таких алгоритмов и не использует особенностей той или иной модели. Эти способы являются наиболее универсальными, однако их эффективность зачастую ограничена.

Для многих моделей более существенной оптимизации можно добиться, если использовать дополнительную заранее известную информацию о компонентах, возникающих в них событиях и их возможных взаимосвязях. Здесь будет полезно обратиться к декларативной парадигме программирования и понятию «поведения» компоненты для того, чтобы собрать такую информацию. Под поведением компоненты будем понимать её способность давать стандартные ответы на стандартные запросы окружающей среды и других компонент.

В истории развития информатики известно несколько подходов к программированию, при котором объекты обладают поведением. Подходы эти зародились в среде исследователей, занимавшихся проблемами искусственного интеллекта, и известны как модель акторов [6] и агентное программирование [7, 8]. Проблема здесь в том, что, согласившись на наличие поведения у компонент системы, нужно уметь описывать и моделировать это поведение. С полной достоверностью заранее оно не может быть известно, поскольку может зависеть от внешних по отношению к рассматриваемой компоненте условий, на которые она должна

уметь реагировать должным образом. Тем не менее кое-что о поведении компоненты мы всегда знаем заранее, так как она в силу исходных предположений работы имеет в предметной области моделирования достаточно хорошо изученный прообраз. Поэтому обычно мы заранее можем сказать многое о её возможных «умениях», т.е. действиях, которые она может совершать, событиях, которые она может порождать, возможном взаимодействии компонент друг с другом (т.е. об их взаимозависимостях) и возможных реакциях при таком взаимодействии — в силу того, что так устроен прообраз этой компоненты в предметной области моделирования. Именно такое знание, коррелирующее с пониманием устройства в предметной области как прообраза отдельной компоненты, может быть выражено на декларативном языке программирования и может существенно помочь при реализации алгоритмов управления временем в таких моделях.

В общем случае, без использования такой информации, все происходящие в модели события считаются глобальными, т.е. затрагивающими все компоненты модели. В результате используется единый для всей модели список запланированных событий и единственное время LBTS. Однако в большинстве сложных моделей компоненты взаимодействуют друг с другом направленно, т.е. одна компонента посылает сообщения, так или иначе касающиеся лишь некоторых других компонент. Это позволяет объединить компоненты в логические группы G_i, внутри которых посылаются сообщения. В этом случае построение списка запланированных событий и вычисление LBTS можно осуществлять в рамках каждой отдельной группы. Компоненты, находящиеся вне группы G_i, можно не принимать во внимание, поскольку они не могут породить ситуацию парадокса времени для компонент из группы G_i. Разумеется, такое разделение компонент на группы является динамическим, т.е. должно перестраиваться при наступлении каждого нового события.

В определённых случаях более удобным будет построение групп обратных зависимостей компонент, т.е. для каждой компоненты k определение множества C_k компонент, которые могут посылать сообщения компоненте k (по крайней в течение ближайшего промежутка модельного времени Δt). В этом случае при выполнении компоненты k можно не принимать во внимание события, запланированные компонентами, не входящими во множество C_k.

Ещё одним важным понятием, которое позволит оптимизировать консервативный алгоритм управления временем, является чувствительность компоненты к событиям. Под данным термином здесь понимается набор возможных реакций компоненты на событие. Зачастую может случиться так, что, хотя одна компонента отправляет другой компоненте сообщение, поведение второй компоненты от этого не изменится либо, по крайней мере, не изменится её влияние на ход всей модели в целом. Это позволяет не блокировать выполнение второй компоненты, несмотря на то, что для неё могут появиться события с метками времени в прошлом относительно её текущего модельного времени (т.е. то, что как раз и называется «парадоксом времени»). Например, в описанной выше модели валютных операций в случае, если счёт кредитный, то выполнение операции одной компонентой не повлияет на решение банка о выполнении операции другой компонентой. Таким образом, вычисления, производимые для одних компонент, можно не приостанавливать, ожидая получение сообщений от других компонент.

Расширением этого варианта оптимизации служит так называемый принцип нестрогого упорядочения событий. Хотя он и не позволяет добиться полной повторяемости результатов моделирования, тем не менее в ряде случаев этот приём может быть полезен. Наиболее часто он применяется при использовании другого класса алгоритмов управления временем (оптимистических) о которых мы поговорим ниже.

Напоследок рассмотрим противоположный вариант: когда валютный счёт дебетовый, и при этом остаток средств на нём насколько мал, что ни одна из валютных операций не может быть выполнена. В такой ситуации перечисленные события заведомо не произойдут (во всяком случае в течение определённого промежутка времени), а значит, заранее можно сказать, что список событий до определённого момента не пополнится новыми событиями. Для анализа подобных ситуаций используется понятие «забегание вперёд» (lookahead), основанное в свою очередь на понятии «расстояния» между компонентами. Расстояние —

это количество модельного времени, которое должно пройти, прежде чем одна компонента вызовет событие, которое повлияет на выполнение другой компоненты. Забегание вперёд — это возможность каждой компоненты предсказать, что в течение определённого времени в будущем она не вызовет событий, которые повлияют на одну или несколько других компонент. Такая технология особенно эффективна в случае, если компоненты слабо связаны между собой (т.е. если в них редко возникают события, затрагивающие другие компоненты). В более общих терминах технология забегания вперёд описана также в [4, 5].

Перейдём теперь к описанию другого класса алгоритмов управления временем и поищем возможные пути их оптимизации.

3. Оптимистические алгоритмы управления временем

Как следует из сказанного выше, задачей консервативных алгоритмов управления временем является полное исключение возможности возникновения парадоксов времени. При этом такие алгоритмы достаточно сильно снижают эффект, который может быть достигнут за счёт использования распределённой модели, особенно если речь идёт о моделях, в которых компоненты сильно связаны, а возникновение событий не детерминировано.

Для устранение указанных негативных эффектов был разработан другой алгоритм управления временем, который не запрещает возникновение парадоксов времени, однако откатывает результаты расчётов модели в случае, если какая-либо из компонент запланировала событие «в прошлом». В данном случае опять будет уместной аналогия с реляционными базами данных, в которых результаты выполнения транзакций могут быть откачены (rollback) в случае её неудачного выполнения. Откат вычислений подразумевает не только отмену результатов моделирования конкретной компоненты, но также отмену событий, возникших с момента возникновения парадокса времени. Именно этот момент и является самым сложным при реализации оптимистических алгоритмов, поскольку отмена одного события может повлечь цепную реакцию отмены других событий, возникших за это время, и в конечном счёте может отразиться на всех компонентах модели. Технически откаты могут выполняться различными способами, однако наиболее распространён способ, сохраняющий промежуточные результаты моделирования для каждой компоненты в момент возникновения каждого нового события. Такой способ реализации технически достаточно прост, он позволяет быстро из без больших затрат выполнить откат вычислений, однако может потребовать значительных ресурсов (прежде всего памяти) для хранения промежуточных результатов в случае, если модель состоит из очень большого числа компонент. Другие способы реализации алгоритма хотя и уменьшают нагрузку на память, но перекладывают её на вычислительный блок, тем самым замедляя выполнение алгоритма. Наиболее простым, но эффективным способом оптимизации количества ресурсов, затрачиваемых оптимистическим алгоритмом управления временем, является вычисление нижней границы временных меток любого будущего отката — GVT (Global Virtual Time). Процедура его вычисления аналогична процедуре вычисления LBTS в консервативных алгоритмах, а смысл этого показателя состоит в том, что все сохранённые промежуточные результаты моделирования, относящиеся к моментам времени, предшествующим GVT, могут быть очищены из памяти, поскольку откат к таким состояниям невозможен. Такая техника позволяет периодически очищать память при продвижении вычислительного эксперимента, таким образом существенно экономя ресурсы.

Другие общие способы оптимизации алгоритма могут быть найдены в [4, 5], здесь же мы опишем вариант оптимизации под названием «принцип нестрогого упорядочения событий», уже упомянутый выше. Этот принцип позволяет не выполнять откат вычислений, даже если какое-либо сообщение получено слишком поздно, в тех случаях, когда разница между временем сообщения и текущим модельным временем получившей сообщение компоненты невелика, либо если результат вычислений для этой компоненты в результате позднего получения сообщений значительно не изменится. Это может быть характерно для не детерминированных моделей (т.е. моделей, в которых присутствует случайный фактор),

а также для моделей, состоящих из очень большого числа компонент (агентов), динамика изменения характеристик которых невелика (например, модели распространения заболеваний, загрязнений, модели динамики популяций и т.п.). Поскольку в таких моделях парадоксы времени случаются достаточно часто, а количество вычислений, требующееся для осуществления отката и повторного прогона модели, велико, то для таких моделей принцип нестрогого упорядочения событий показывает весьма хорошие результаты.

4. Выбор алгоритма управления временем

В большинстве случаев нельзя точно сказать, алгоритм какого класса — консервативный или оптимистический — будет предпочтительнее для той или иной распределённой модели. Это зависит от многих факторов: динамики выполнения модели, количества компонент модели и степени их взаимосвязи, а также используемых средств оптимизации. Однако во многих случаях оптимистические алгоритмы дают больший выигрыш во времени выполнения модели за счёт более эффективного использования распределённых вычислительных средств. Кроме того, базовый вариант оптимистического алгоритма оказывается проще для реализации, чем консервативный.

Тем не менее, для того чтобы получить серьёзный выигрыш от распределённого моделирования, следует уделять значительное внимание оптимизации алгоритмов управления временем. Помимо использования общих способов оптимизации значительный выигрыш можно получить, основываясь на особенностях поведения тех или других моделируемых объектов.

Все описанные в данной статье способы оптимизации одинаково эффективны как для консервативных, так и для оптимистических алгоритмов. Следует ещё раз отметить, что все они исходят из неких предопределённых знаний о возможном поведении компонент модели, т.е. основаны на их декларативных характеристиках. Это не позволяет сделать их универсальными, однако в определённых случаях их эффективность оказывается значительной.

Работа выполнена при финансовой поддержке РФФИ, грант №10-07-00176-а.

Литература

1. *Бродский Ю. И.* Распределенное имитационное моделирование сложных систем. — М.: ВЦ РАН, 2010. — 156 с.
2. *Бродский Ю. И., Павловский Ю. Н.* Разработка инструментальной системы распределенного имитационного моделирования // Информационные технологии и вычислительные системы. — 2009. — № 4. — С. 9–21.
3. *Zave P.* A compositional approach to multiparadigm programming // IEEE Software. — 1989. — V. 6, I. 5. — P. 15–25.
4. *Окольнишников В. В.* Представление времени в имитационном моделировании // Вычислительные технологии. Сибирское отделение РАН. — 2005. — № 5. — С. 57–80.
5. *Замятина Е. Б.* Современные теории имитационного моделирования: специальный курс. — Пермь: ПГУ, 2007. — 119 с.
6. *Hewitt C.* Viewing control structures as patterns of passing messages // Artificial Intelligence. — 1977. — V. 8, I. 3. — P. 323–364.
7. *Shoham Y.* Agent-oriented programming // Artificial Intelligence. — 1993. — V. 60, I. 1. — P. 51–92.
8. *Shoham Y.* Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. — Cambridge.: University Press, 2009. — 483 p.

Поступила в редакцию 05.05.2012.