ОПТИМИЗАЦИЯ РАСПРЕДЕЛЕННЫХ АЛГОРИТМОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

С. А. Ермаков, Е. Б. Замятина (Пермь)

В настоящее время имитационное моделирование (ИМ) широко применяется в различных сферах науки и производства. Развитие науки и технологий, и, следовательно, рост сложности исследуемых систем ставит перед ИМ все более сложные задачи. Последовательные симуляторы не справляются с таким объемом вычислений, поэтому наиболее актуальными становятся распределенные системы моделирования. В настоящее время существуют как иностранные (TeD/GTW [1], SPEEDES [2], Task-Kit [3] и др.), так и отечественные (Мера[4], Диана [5], СИРИУС [6], OpenGPSS [7]) разработки в области распределенного моделирования. Разработка распределенных систем моделирования требует создания специального программного обеспечения, обеспечивающего корректное выполнение модели. В основе таких симуляторов лежит алгоритм синхронизации (АС) модельного времени объектов, который обеспечивает соблюдение причинно-следственных связей между объектами модели, находящимися на различных вычислительных узлах распределенной системы. Выигрыш при распределенном выполнении можно получить за счет эффективности АС. В работе авторы рассматривают особенности существующих алгоритмов, а также предлагают свой алгоритм, использующий явные и неявные знания о модели.

Алгоритм синхронизации

Целью АС является соблюдение правильной последовательности выполнения событий модели – в порядке неубывания их временных меток, для того, чтобы сохранить причинно-следственные связи. В последовательном моделировании правильный порядок обеспечивается за счет выбора на каждом шаге события с минимальной меткой из общего календаря событий всей системы, текущее модельное время одинаково для всех объектов. В распределенном варианте общий календарь отсутствует, что приводит к необходимости использования АС. Каждая часть распределенного симулятора является последовательным симулятором, содержащим часть общей модели. Традиционно такая часть называется логическим процессом (ЛП). Каждый ЛП не может заранее знать о том, на какое время будет запланировано событие, которое он получает от другого логического процесса.

По главному принципу работы, алгоритмы принято разделять на два класса: консервативные и оптимистические. Консервативные АС запрещают обработку событий со временем *T*, пока не будет гарантии, что ЛП не получит сообщений с временной меткой меньше *T*. Оптимистические АС не накладывают столь жестких ограничений, позволяя обрабатывать такие события, однако при поступлении события с временной меткой, меньшей текущего времени, организуют откат до этого момента.

Классическим консервативным АС является алгоритм «нулевых сообщений» [8]. Суть подхода состоит в том, что каждый логический процесс подсчитывает $\min(T_{inp}^{i})i=\overline{1,n}$, где T_{inp}^{i} – временная метка последнего поступившего сообщения от $ЛП_t$, может продвигать время до этого минимума. Для того чтобы его расширить временные горизонты процессов и не попасть в тупиковую ситуацию, каждый ЛП по всем выходным связям при изменении своего времени посылает сообщение – либо реальное, запрограммированное в модели, либо фиктивное или «нулевое».

Классическим оптимистическим АС является алгоритм «Time Warp» [9]. Алгоритм позволяет обрабатывать любые события, но при поступлении сообщения с временной меткой *T'*, меньшей текущего времени *T*, происходит откат на время *T'*. Для

реализации отката необходимо восстановить состояние текущего ЛП на момент времени *T*'. А также отменить сообщения, разосланные другим логическим процессам. Для этого используется механизм «антисообщений»: для каждого обработанного сообщения хранится противоположное. Для отмены сообщения посылается соответствующая «антикопия». При получении антисообщения процесс либо удаляет положительное сообщение из очереди, если оно не было еще обработано, либо также производит откат до времени антисообщения.

Оптимизация алгоритмов синхронизации

Рассмотренные выше базовые алгоритмы имеют серьезные недостатки: алгоритм «нулевых сообщений» генерирует огромное количество пустых сообщений, а также сильно ограничивает параллельное выполнение процессов. Time Warp – напротив, никак не ограничивает выполнение симуляторов, что приводит к большому количеству откатов, которые сигнализируют о значительных тратах вычислительных ресурсов впустую. Модельное время из-за откатов так же движется медленно.В настоящее время существует большое количество работ по оптимизации распределенных АС.

Алгоритм нулевых сообщений оптимизируется различными путями, рассмотрим основные из них:

1. Оптимизация количества посылаемых нулевых сообщений [10]:
   а) по запросу для расширения горизонта;
   б) при возникновении тупика
2. Расширение временного горизонта за счет использования:

а) параметра, называемого LookAhead [11]. В этом случае, процесс гарантирует, что после посылки сообщения с временной меткой *T*, следующее сообщение будет не ранее, чем через LookAhead единиц времени. Далее возможна более тонкая оптимизация за счет различных LookAhead для разных связей между логическими процессами. В основном предполагается, что данный параметр будет задаваться вручную, однако есть работы по его автоматическому вычислению [12].

б) информации о временной метке следующего события. Это еще более тонкая оптимизация, чем LookAhead, однако требуется специальный способ описания модели, в общем случае мы не можем получить точную информацию о временной метке будущего события.

Для Time Warp так же существуют способы оптимизации. В основном они борются с чрезвычайным забеганием вперед времени логических процессов, для этого вычисляется нижняя временная метка всех логических процессов (GVT), ниже которой откат произойти не может, а далее используются следующие методы:

1. «Временное окно» – это параметр *T*, ограничивающий продвижение времени в промежутке от [GVT, GVT+*T*]. Временное окно может задаваться жестко, а может вычисляться автоматически, с помощью генетических алгоритмов [13], техники машинного обучения [14] или по другим методикам. Такие методы хороши для моделей, в которых допустимая разница между временем логических процессов фиксирована, тогда за сравнительно короткий промежуток времени находится его оптимальное значение, и далее модель функционирует эффективно. Однако разница может варьироваться в процессе выполнения модели, и данный алгоритм показывает слабые результаты.

2. Использование LookBack [15]. Методика очень похожа на консервативный алгоритм с LookAhead и служит для того, чтобы избежать глобальных откатов (откатов, затрагивающих несколько локальных процессов). Если текущее время ЛП-*T*, то (*T*-LookBack) это – время, за которое процесс может выполнить исключительно локальный откат, таким образом, если ограничивать продвижение времени с помощью LookBack, можно избежать глобальных откатов, что значительно экономит ресурсы

вычислительных узлов, однако, сильно ограничивает продвижение времени, приближая алгоритм к консервативному.

3. Посылка сообщений между ЛП, только в том случае, если гарантируется, что они не будут аннулированы в случае отката. Данный алгоритм напоминает LookBack, но накладывает еще более жесткие условия на продвижение времени в модели.

Мы проанализировали лишь основные направления оптимизации алгоритмов, исключив технические усовершенствования, такие как выполнение вспомогательных вычислений на отдельных узлах, оптимизация хранения состояния модели, или использование обратных вычислений для восстановления состояния. Идеальный алгоритм должен максимально использовать параллелизм модели, при этом не выполняя лишних вычислений (вычисления не должны откатываться, а расходы на реализацию алгоритма должны быть минимальны). Большинство алгоритмов для достижения этой цели использует вспомогательные параметры (LookAhead, LookBack, временное окно), которые представляют собой неявную информацию, заложенную внутри модели, причем заметна тенденция увеличения эффективности алгоритма с увеличением количества используемой информации о модели.

Алгоритм, использующий информацию о модели

Проанализированные АС в основном используют зависимость процессов по времени, хотя на самом деле такая зависимость может варьироваться. Рассмотрим пример: пусть имеются 2 кассы, продающие билеты на один поезд. Поток клиентов к первой кассе приходит раз в 15±30 мин, клиенты ко второй кассе приходят раз в 60±15 мин. Пусть при некотором прогоне получилась такая последовательность: 1, 25, 40, 120; 2, 66, 135, 200. Оба процесса выполняются с одинаковой скоростью, однако модельное время движется по-разному. Если рассматривать зависимость по времени, то получается, что временное расстояние изменяется, что затрудняет подсчет, например, корректного временного окна. Однако если рассмотреть суть зависимости, то видно, что вторая очередь должна генерировать событие только после первой, т.е. налицо зависимость между событиями.

В сущности, результат моделирования представляет собой последовательность изменения состояний, связанную с последовательностью обработки событий. Проблема алгоритма на каждом шаге – определить, является ли обработка следующего локального события безопасной, т.е. гарантия отсутствия отката. Сама природа модели содержит не только зависимости между объектами по времени, которую традиционно используют алгоритмы, но и зависимость (событие, состояние) → событие.

Если заранее иметь полное множество зависимостей между состояниями и событиями модели во времени, то принимать решение об обработке очень легко, нужно лишь сверить текущее состояние системы с нужным состоянием из множества – если оно ему соответствует, то событие можно обработать, иначе приостановить выполнение до получения соответствия. Однако, построение такого множества зависимостей – это и есть процесс моделирования, поэтому мы не можем получить его заранее. Можно лишь анализировать текущую накопленную последовательность (в рамках одного прогона) или результаты предыдущих выполнений системы. Также к построению зависимостей можно привлекать эксперта, создающего модель. Эти зависимости являются причинно-следственными, т.е. A→B. Так как такая связь может быть не строгой, а зависеть от генерации случайных величин во время прогона, или в случае задания экспертом может содержать неточность или нечеткость, поэтому необходимо использовать понятие доверия к зависимости.

Адекватное представление такой информации о модели дает продукционная парадигма, кроме того, в качестве дополнительного плюса ее использования можно отметить понятность и наглядность ее для пользователя.

Мы не можем целиком полагаться на знания пользователя, так как в основном явные знания уже запрограммированы в модели, а неявные сложно извлекать, поэтому необходимо уметь генерировать продукционные правила автоматически: в ходе статического анализа кода модели перед выполнением, а так же в ходе выполнения: при откатах создавать правила, предотвращающие появление откатов в будущем. Если мы обработали событие покупки билета вторым клиентом (E2), а затем произошел откат из-за события покупки билета первым клиентом (E1), в системе генерируется связь (E1→E2). Разработанный алгоритм авторы назвали TriadRule. Подробно теоретические основы и практическая реализация изложены в работе [16].

Эксперименты с алгоритмом TriadRule

Выше описанный подход использован для реализации синхронизации распределенной системы Triad.NET[16]. Алгоритм в данный момент оптимизируется, но уже получены некоторые результаты. Моделирование проводилось на кластере с использованием нескольких простых моделей. Для сравнения производительности были выбраны классический консервативный и оптимистический алгоритмы. Наиболее показательные результаты были получены на модели вычислителя ILLIAC (рис. 1).

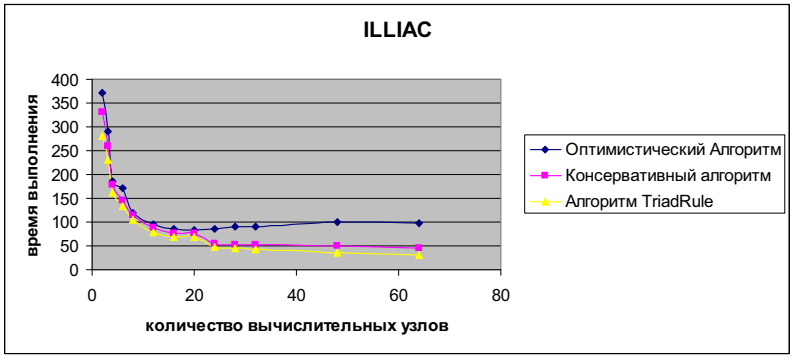

Рис. 1. Эксперименты с моделью ILLIAC

По графику (рис. 2) видно, что алгоритм показывает лучшую производительность, чем оба алгоритма традиционных алгоритма, причем, что важно, при увеличении количества вычислительных узлов, разница становится более заметна. Происходит это потому, что растет количество логических процессов, и, следовательно, и общее количество внешних связей в системе, что для консервативного алгоритма приводит к уменьшению быстродействия, т.к. временной горизонт продвижения становится меньше, а для оптимистического приводит к увеличению количества откатов.

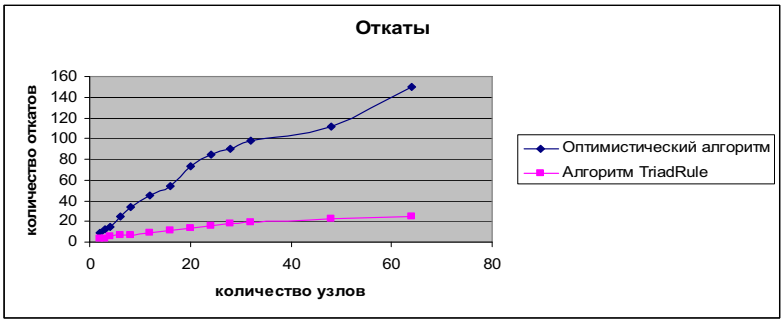

Рис. 2 Сравнение количества откатов

Выводы

Для реализации эффективных алгоритмов синхронизации необходимо извлечение максимальных знаний о модели. Традиционно используются знания только о временных зависимостях между логическими процессами, при этом теряется информация о зависимости событий и состояний между собой. Авторы предложили подход, позволяющий учесть причинно-следственные связи для продвижения времени. Чтобы достичь оптимальности, нужно скомбинировать эти подходы, а также можно задействовать средства анализа результатов моделирования, например Data-Mining, с целью выявления более скрытых зависимостей.

Литература

1. **Bhatt S**. Parallel Simulation Techniques for Large-Scale Networks.// IEEE Communications, 1998. 36(8). P. 42–47.
2. **Steinman J. S.** SPEEDES: A Multiple-Synchronization Environment for Parallel // Discrete Event Simulation. International Journal on Computer Simulation, 1992. P. 251–286.
3. **Unger B.** Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation // Proceedings of the Workshop on Parallel and Distributed Simulation, 1999.
4. **Окольнишников В. В.** Разработка системы распределенного имитационного моделирования для различных операционных сред //Имитационное моделирование. Теория и практика (ИММОД 2005), Санкт-Петербург, 2005. Т. 1. С. 256–260.
5. **Грибов Д. И., Смелянский Р. Л.** Комплексное моделирование бортового оборудования летательного аппарата //Труды конференции «Методы и средства обработки информации», МСО-2005. М., 2005. С. 59–76.
6. **Феоктистов А. Г.** Организация распределенного имитационного моделирования в GRID-системе //Параллельные вычислительные технологии: Тр. I Междунар. науч.-практ.конф. Челябинск, 2007. Т. 2. С. 28–36.
7. **Диденко Д. Г.** От последовательного моделирования в системе GPSS/World к распределённому моделированию в OpenGPSS. // Имитационное моделирование. Т. 1. Теория и практика (ИММОД 2009). СПб.: 2009. С. 251–256.
8. **Chandy K. M., Misra J.** "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." IEEE Transactions on Software Engineering, 1978 SE-5(5). P.440–452.
9. **Jefferson D., Sowizral H.** Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control // Rand Note N-1906AF, Rand Corp., California, 1982.
10. **Rizvi S. S., Elleithy K. M., Riasat A.** Minimizing the null messageexchange in conservative distributed simulation // Proceedings of International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, 2006.
11. **Fujimoto R. M.** Lookahead in parallel discrete event simulation//Proceedings of the 1988 International Conference on Parallel Processing, 1988. P. 34–41.
12. **Zarei Behrouz, Pidd Michael.** Performance analysis of automatic lookahead generation by control flow graph: some experiments // Simul. Pr. Theory, 2001. P.511–527.
13. **Wang J., Tropper C.** Using Genetic Algorithms to Limit the Optimism in Time Warp //Winter Simulation Conference (WSC), Austin, Texas, December 13-December 16, 2009.
14. **Meraji Sina, Tropper Carl**. A Multi-State Q-learning Approach for the Dynamic Load Balancing of Time Warp //Workshop on Advanced and Distributed Simulation (PADS'10), May 16-May 19, Atlanta, Ga, 2010.
15. **Chen G., Szymanski B.K**. Lookback: a new way of exploiting parallelism in discrete event simulation // Proc. of the 16th Workshop on PDES, 2002. P. 153–162.
16. **Замятина Е. Б., Ермаков С. А.** Алгоритм синхронизации объектов распределенной имитационной моделт в Triad.Net //ITHEA, Болгария, 2011. С. 211–221.