Метод локальных каскадных откатов и организация списков событий для распределенного моделирования компьютерных систем

Ладыженский Ю.В., Мирецкий А.В.
Донецкий национальный технический университет
a.miretsky@gmail.com

Abstract

Ladyzhensky Y., Miretsky A. Method of local cascade rollback and distribution of event lists between components. *New method of local cascade rollbacks is described in this paper. This method allows decreasing depth of the rollbacks in time and in space of components of simulation. Event lists distributed between components are used in this method. The algorithm for inserting events into them is described in this paper too. Time of inserting events is estimated for it. The distribution of events allows increasing speed of inserting events into the event-list.*

Введение

Имитационное моделирование широко применяется для исследования свойств компьютерных систем и сетей [1]. Эти системы часто представляются дискретными моделями, поэтому для них используется метод дискретно-событийного моделирования.

Моделирование компьютерных систем и сетей на одном процессоре обычно требует много времени. Ускорить процесс моделирования можно, если выполнить декомпозицию исследуемой модели и выполнять моделирование полученных подмоделей на разных процессорах. Однако в этом случае возникает задача синхронизации логических процессов по времени. Существует два базовых алгоритма синхронизации моделирующих процессов для метода дискретно-событийного моделирования: консервативный и оптимистический [2]. Среди недостатков консервативного алгоритма можно выделить то, что могут возникать ситуации, когда даже слабосвязанные моделирующие процессы циклически ожидают друг друга с целью синхронизации, превращая тем самым параллельное моделирование в последовательное. У оптимистического алгоритма есть ряд преимуществ по сравнению с консервативным алгоритмом. Однако следует выделить один из самых значимых недостатков оптимистического алгоритма – это выполнение каскадных откатов, которые приводят к возврату в прошлое состояние всех логических процессов, обменивавшихся между собой сообщениями. Выполнение откатов влечет за собой временные издержки. Чем глубже откат, тем больше времени требуется на его выполнение.

Наличие перечисленных недостатков обусловило поиск решений по их устранению. Разработаны модификации базовых алгоритмов, их комбинации и адаптивные алгоритмы, меняющие свои свойства от консервативного к оптимистическому. Однако, в большинстве случаев, эти алгоритмы не учитывают особенностей поведения моделируемых систем [3].

Актуальными являются задачи разработки методов ускорения распределенного моделирования компьютерных систем и сетей, учитывающих особенности структуры и поведения этих систем.

Цель настоящей работы – разработка метода уменьшения откатов в оптимистическом алгоритме для моделирования компьютерных систем и сетей. Метод должен учитывать особенности этих систем.

В работе предлагается метод локальных каскадных откатов. Он позволяет ускорить процесс оптимистического распределенного дискретно-событийного моделирования за счет уменьшения глубины откатов в пространстве моделирующих компонентов. Для уменьшения глубины откатов, учитывается обмен сообщениями между компонентами системы, моделируемыми в рамках логических процессов. Метод предполагает распределение всех списков событий, используемых в оптимистическом алгоритме, между моделирующими компонентами. Рассмотрена используемая в методе особая организация списков событий. Общий список событий для логического процесса и распределенные списки событий для компонентов объединены в одну общую структуру данных. Предложен алгоритм вставки новых событий в такой список. Показано аналитически и экспериментально, что распределение списков событий между компонентами позволяет уменьшить время вставки новых событий в список.

1 Оптимистический алгоритм

Рассмотрим оптимистический алгоритм в сравнении с консервативным.

Консервативный алгоритм предполагает, что выполнение каждого события может привести к нарушению ограничения причинности [2]. Каждый логический процесс выполняет событие тогда, когда уверен в том, что другие процессы не породят для него событий, предшествующих выполняемому событию. Мерой уверенности является характеристика моделируемой системы lookahead, определяющая интервал времени, в течение которого процесс точно не будет порождать событий для других процессов. От этой характеристики зависит скорость выполнения моделирования.

В оптимистическом алгоритме выполнение всех событий считается безопасным [2]. Моделирующие процессы работают параллельно без синхронизации, пока один из процессов не получит отставшее во времени сообщение от другого процесса. Как только это происходит, выполняется процедура отката. Состояние процесса, принявшего отставшее сообщение, возвращается назад, откатывается до временной метки этого сообщения. Если на этом интервале времени, процесс отправлял сообщения другим процессам, то выполняется каскадный откат этих процессов.

Для выполнения отката состояния логического процесса в оптимистическом алгоритме ведется журнал изменения состояния и журнал выполненных событий. Для всех сообщений, которые отправляются другим процессам, создаются анти-сообщения, которые помещаются в журнал анти-сообщений. Они необходимы для осуществления каскадных откатов.

Есть два основных метода ведения журнала состояния: сохранение снимка состояния логического процесса; инкрементное сохранение состояния (сохраняется только информация об изменениях в состоянии). Первый способ позволяет очень быстро откатить состояние логического процесса, однако требует больших объемов памяти для хранения снимков. Второй способ использует меньше памяти, но откат выполняется дольше, т.к. требуется проход по всем записям журнала для возврата к состоянию в заданный момент времени. На практике чаще всего используется второй способ ведения журнала.

Откат состояния логического процесса в целом и каскадные откаты – это основные факторы, снижающие скорость моделирования при использовании оптимистического алгоритма.

2 Метод локальных каскадных откатов

Метод локальных каскадных откатов (ЛКО) использует компонентное представление моделируемой системы. Основная идея метода локальных каскадных откатов заключается в том, что можно выполнять откат только части компонентов логического процесса с учетом того, как они обменивались сообщениями между собой, потому что откат состояния логического процесса в целом не всегда оправдан [4].

Формально алгоритм ЛКО представим следующим образом (см. рис. 1).

```
Откат компонента c до временной метки rt - Rollback(c, rt)
Вход c, rt
Если LVT(c) ≥ rt тогда
 Откат состояния c до отметки rt
 LVT(c) := rt
 LC := LinkedComponents(c)
 ДляКаждого lc из LC цикл
  ts := MinTimeStamp(AS(c, lc), rt)
  Если ts ≠ ∅ тогда
   Rollback(lc, ts)
  КонецЕсли
  DeleteAllGreater(AS(c, lc), rt)
 КонецЦикла
 ДляКаждого e=(σ`, τ`, c`, h`) из HE(c) цикл
  Если σ` ≥ LVT(c`) тогда
   DeleteEvent(e, HE(c))
  Иначе
   Если τ` ≥ rt тогда
    ScheduleEvent(e)
   КонецЕсли
  КонецЕсли
 КонецЦикла
 ДляКаждого e=(σ`, τ`, c`, h`) из SE(c) цикл
  Если σ` ≥ LVT(c`) тогда
   DeleteEvent(e, SE(c))
   DeleteEvent(e, SE)
  КонецЕсли
 КонецЦикла
КонецЕсли
Выход
```

Рисунок 1 – Алгоритм локальных каскадных откатов

Пусть граф $M = (C, L)$ - это имитационная модель вычислительной системы, состоящей из множества компонентов C и множества линий связи между компонентами – L. Каждый компонент характеризуется локальным виртуальным временем $LVT^{(c)}$. Пусть $e=(\sigma, \tau, c, h)$ – сообщение, созданное компонентом c в момент времени σ для компонента h, который должен обработать это сообщение в момент времени τ. Для каждого логического процесса имеются следующие списки: SE – общий список событий; $SE(c)$ – список событий, запланированных для компонента c; $HE(c)$ – список событий, обработанных компонентом c; $AS(c_1, c_2)$ – список событий, отправленных компонентом c_1 компоненту c_2 (список антисообщений). Введем

следующие программные функции: *LinkedComponents(c)* – возвращает множество компонентов, связанных с *c* линиями связи; *MinTimeStamp(A,t)* – ищет сообщения $e=(\sigma,\tau,c,h)$ во множестве сообщений *A*, такие, что $\sigma \geq t$, и возвращает минимальную временную метку σ среди найденных сообщений; *DeleteAllGreater(A,t)* – удаляет все сообщения из множества *A* с временными метками $\sigma \geq t$; *DeleteEvent(e, A)* – удаляет сообщение *e* из множества сообщений *A*; *ScheduleEvent(e)* – запланировать событие *e*, т.е. поместить его в списки *SE* и *SE(h)*.

Покажем работу алгоритма на примере процесса моделирования сети на рисунке 2.

Пусть логический процесс LP_3 передает логическому процессу LP_1 сообщение с временной меткой $ts < T_1$, где T_1 – локальное модельное время процесса LP_1. Это сообщение является отставшим и приведет к откату процесса LP_1 до момента времени *ts*. Пусть на интервале времени $[ts, T_1]$ между компонентами происходил обмен сообщениями, показанный на рисунке 2 стрелками. Направления стрелок показывают, какие компоненты отправляли сообщения, и какие их принимали.

Согласно алгоритму Time Warp, будет выполнен откат всего логического процесса LP_1. Это, в свою очередь вызовет каскадный откат всех логических процессов, кроме LP_3, показанных на рисунке. Однако, есть некоторая вероятность того, что обработка отставшего события и повтор отмененных во время отката событий не вызовут создание новых сообщений, отправляемых узлами 1, 2, 3 в узлы 4, 5, 6, соответственно, на интервале $[ts, ts + \Delta t]$, где $\Delta t > 0$ (в общем случае). Т.к. только сообщения меняют состояние компонентов и способны порождать новые сообщения, то из вышесказанного следует, что в откате всего логического процесса нет необходимости. Достаточно выполнить откат только для компонентов 1, 2, 3 и для логического процесса LP_4.

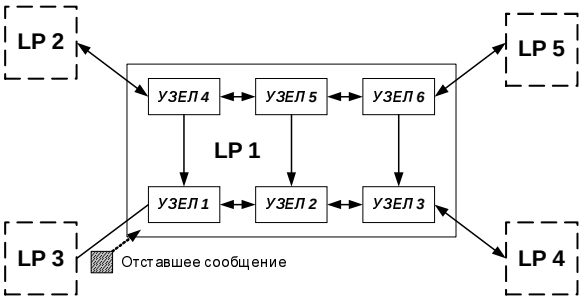

Рисунок 2 – Модель сети. Поступление отставшего сообщения.

На рисунке 3 показан пример процесса каскадной отмены событий произошедших в узлах 1 и 2, выполняемого в соответствии с алгоритмом локальных каскадных откатов.

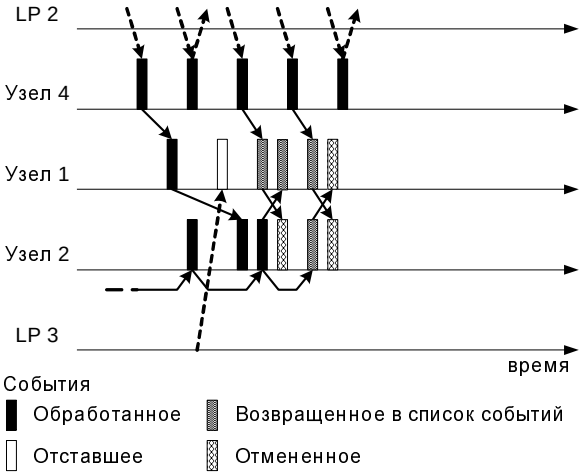

Рисунок 3 – Порядок отмены событий методом локальных каскадных откатов.

Стрелками показаны причинно-следственные связи между событиями. Пунктирными стрелками показано, что событие пришло из внешнего логического процесса. Прямоугольниками показаны события, обработанные компонентами до выполнения отката.

На рисунке 3 видно, что применение метода локальных каскадных откатов позволяет избежать отмены всех событий логического процесса. Отменяются только те события, которые были обработаны компонентом «Узел 1» и все следствия этих событий.

Пусть в момент $ts + \Delta t$ узел 1 создает сообщение для узла 4 и пусть $ts + \Delta t < T_1$. Тогда для компонента 4 тоже должен быть выполнен откат, т.к. для него не был выполнен откат. Но откат уже будет выполнен до отметки $ts + \Delta t$. Откат компонентов смежных узлу 4 может быть выполнен только в том случае, если узел 4 отправлял им сообщения на интервале $[ts + \Delta t, T_1]$. И так далее.

Метод локальных каскадных откатов отличается от метода каскадных откатов, используемых в алгоритме Time Warp, в том, что каскадные откаты переносятся с уровня логических процессов на уровень компонентов, т.е. имеет место детализация откатов. Эта детализация уменьшает глубину отката в пространстве и во времени, т.к. учитывает обмен сообщениями не только между логическими процессами, но и между их компонентами.

3 Распределение списков событий по компонентам

Метод локальных каскадных откатов использует два типа списков запланированных событий:

1) общий список событий;

2) внутрикомпонентные списки.

Общий список событий – это список всех событий, запланированных для логического процесса.

Внутрикомпонентный список событий – это список событий, ассоциированный с одним компонентом логического процесса и содержащий только те события, обработчиком которых назначен этот компонент. Внутрикомпонентные списки не пересекаются, а объединение всех внутрикомпонентных списков компонентов одного логического процесса есть общий список событий этого процесса. Элементы внутрикомпонентных списков отображаются на элементы общего списка событий. Можно сказать, что общий список событий одного логического процесса распределен между компонентами.

Известно, что структура элементов двунаправленного списка описывается через три составляющих: информационная часть (полезная информация, сохраненная в списке), указатель на следующий элемент списка и указатель на предыдущий элемент списка. Для упрощения отображения элементов внутрикомпонентных списков на элементы общего списка и для уменьшения затрат памяти предлагается объединить общий список и все внутрикомпонентные списки в одну общую структуру данных. Для этого структура элемента списка должна быть расширена двумя дополнительными указателями на следующий и предыдущий элементы.

Оценим объем памяти, необходимой для хранения списка событий, распределенного между компонентами. Пусть v_1 – объем памяти, необходимый для хранения одного указателя; v_2 – объем памяти, необходимый для хранения одного события. Один элемент двунаправленного списка событий требует выделения памяти объемом $2v_1+v_2$ байт. А список, содержащий N элементов – $N\cdot(2v_1+v_2)+2v_1$, где последнее слагаемое $2v_1$ – это объем памяти для хранения указателя на голову списка и на хвост списка.

Для хранения элемента списка с предложенной структурой необходимо $2v_1+2v_1+v_2$ байт памяти. Так как список распределен между компонентами, то каждый компонент должен поддерживать свои два указателя на голову и хвост своего списка. Следовательно, общий объем памяти, требуемый для организации списка из N элементов, распределенного между K компонентами равен: $N\cdot(2v_1+2v_1+v_2)+2Kv_1+2v_1$. На рисунке 4 приведен график зависимости длины списка событий от количества компонентов в системе при следующих параметрах: N=100000 событий, v_1=4 байт, v_2=24 байт (указатель на компонент-обработчик – 4 байт, указатель на компонент-создатель – 4 байта и два вещественных числа для хранения временных меток по 8 байт).

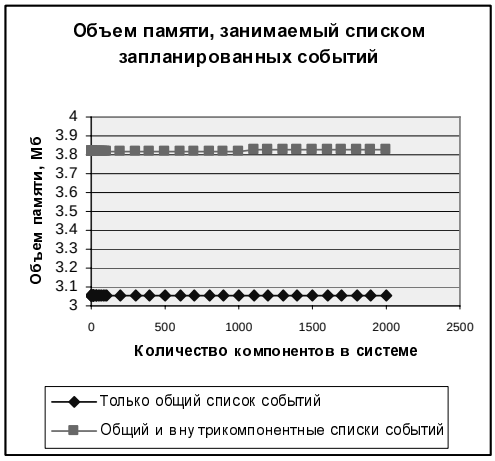

Рисунок 4 - Зависимость длины списка событий от количества компонентов в системе.

Из графика на рисунке 4 видно, что увеличение требуемой памяти составляет около 25%, при увеличении количества компонентов и постоянной длине списка эта величина практически не изменяется.

Помимо практической необходимости для реализации алгоритма локальных каскадных откатов, предложенная организация списков событий позволяет сократить время планирования (вставки в список событий) нового события.

Идея алгоритма вставки нового события в список событий состоит в том, что сначала производится поиск двух событий во внутрикомпонентном списке, между которыми должно быть вставлено новое. Затем для нового события ищется позиция в общем списке, причем просматриваются только те события, которые расположены между двумя найденными.

Формально алгоритм вставки нового события в список представим следующим образом. Пусть $C=\{c^{(k)} \mid k=1..K\}$ – множество компонентов модели, где K – количество компонентов. $E=\{e_i=(\tau_i, \sigma_i, c_i, h_i) \mid i=1..N, \tau_{i-1}\le\tau_i \ \forall i>1\}$ – общий список событий, где N – количество событий в списке. $E^k=\{e^{(k)}_j=(\tau^{(k)}_j, \sigma^{(k)}_j, c^{(k)}_j, h^{(k)}_j) \mid j=1..N^{(k)}, \ \tau^{(k)}_{j-1}\le\tau^{(k)}_j \ \forall j>1, \ h^{(k)}_j=c^{(k)}\}$ – внутрикомпонентный список событий, принадлежащий компоненту $c^{(k)}$, где $N^{(k)}$ – количество событий во внутрикомпонентном списке событий. На списки событий наложены ограничения: $E=\bigcup_k E^{(k)}$, $E^{(k1)}\cap E^{(k2)}=\varnothing \ \forall k_1, k_2: k_1\neq k_2$. $G(k, j)=(i: e_i=e^{(k)}_j)$ – функция отображения события из внутрикомпонентного списка на событие в общем списке, где k – номер компонента; j – номер события во внутрикомпонентном списке; результат функции – номер события в общем списке. Пусть $e=(\tau, \sigma, c, h)$ – событие, которое нужно вставить. Алгоритм представлен на рисунке 5.

Вход E, N, $E^{(k)}$ (k=1..K), C, e=(σ,τ,c,h)

{Определяем номер внутрикомпонентного списка событий, в который вставляем новое событие}
0: Lk:=(k|$c^{(k)}$=c);
{Ищем пару событий во внутрикомпонентном списке, между которыми будет вставлено новое событие}
SE:=$N^{(Lk)}$; *{Инициализируем номер левого события}*
EE:=0; *{Инициализируем номер правого события}*
1: **Пока** SE>0 **цикл**
Если $\tau_{SE}^{(Lk)} \le \tau$ **то перейти к** 2;
КонецЕсли;
EE:=SE;
SE:=SE-1;
КонецЦикла;

{Отображение найденной пары событий на общий список событий}
2: CSE:=0;
CEE:=N;
Если SE>0 **тогда** CSE:=G(Lk, SE);
КонецЕсли;
Если EE>0 **тогда** CEE:=G(Lk, EE);
КонецЕсли;

{Поиск позиции для нового события в общем списке}
3: **Пока** CEE>SEE **цикл**
Если $\tau_{CEE} \le \tau$ **то перейти к** 4;
КонецЕсли;
CEE:=CEE-1;
КонецЦикла;

{Вставка события}
4: CSE := CEE;
CEE:=CEE+1;
insert(e, CEE, CSE); *{Вставка нового элемента e в список между парой соседних элементов с номерами CSE и CEE}*
insert_k(Lk, e, EE, SE); *{Вставка нового элемента e во внутрикомпонентный список компонента $c^{(Lk)}$ между парой соседних элементов с номерами SE и EE}*

Рисунок 5 – Алгоритм вставки нового события в список событий.

Оценим время вставки нового события в список, содержащий N событий, для модели с K компонентами:

$$T_{INSERT} = b_1 \frac{N}{2 \cdot K} + b_3 \frac{K}{2} + (b_0 + b_2 + b_4),$$

где b_0, b_1, b_2, b_3, b_4 – константы, показывающие время выполнения операторов (или тел циклов), на соответствующих шагах алгоритма.

Средняя временная сложность алгоритма вставки нового события в обычный список событий:

$$T'_{INSERT} = b_5 \frac{N}{2} + b_6,$$

где b_5 и b_6 – некоторые константы.

Видно, что при N>K и K>1, среднее время вставки с использованием внутрикомпонентных списков в общем случае ниже времени вставки без использования этих списков.

Приведенная аналитическая оценка подтверждена экспериментально. В экспериментах измерялось время вставки 10^5 событий в список, распределенный между K=1..2000 компонентами. Такое решение обусловлено тем, что практически невозможно измерить время вставки одного события. Аналитическая оценка среднего времени заполнения распределенного списка n событиями равна:

$$T_{INSERT}(n) = \sum_{i=0}^{n} \left(b_1 \frac{i}{2 \cdot K} + b_3 \frac{K}{2} + (b_0 + b_2 + b_4) \right) =$$

$$b_1 \frac{1}{2 \cdot K} \cdot \sum_{i=0}^{n} (i) + b_3 \frac{K}{2} n + (b_0 + b_2 + b_4) n =$$

$$= b_1 \frac{n(n+1)}{4 \cdot K} + b_3 \frac{K}{2} n + (b_0 + b_2 + b_4) n =$$

$$= b_1 \frac{n^2}{4 \cdot K} + b_1 \frac{n}{4 \cdot K} + b_3 \frac{K}{2} n + (b_0 + b_2 + b_4) n$$

На рисунке 6 приведена зависимость среднего времени заполнения списка $n=10^5$ событиями от количества компонентов, между которыми список распределен. При построении графика коэффициенты равны: $b_1=1$, $b_3=1$, $b_0+b_2+b_4=1$.

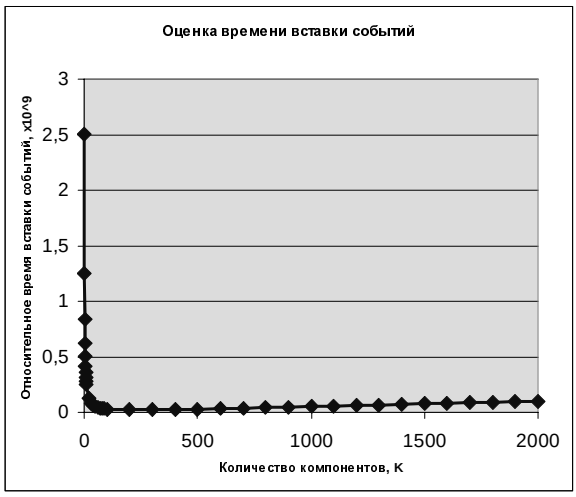

Рисунок 6 – Аналитическая оценка среднего времени заполнения распределенного списка событий

На рисунке 7 показано экспериментально полученное время заполнения. В эксперименте использовался компьютер со следующими характеристиками: процессор – Turion MK36, RAM – 512Мб, операционная система – MS Windows XP SP2. Программная система для выполнения эксперимента написана на языке C++.

Время заполнения распределенного списка событий сравнивается со временем заполнения нераспределенного списка событий. Для

последнего использовался метод прямого перебора при вставке новых событий, т.к. для вставки в распределенный список тоже используется метод прямого перебора.

Видно, что при увеличении количества компонентов, время вставки в список значительно сокращается, однако, начиная примерно с K=200, время вставки начинает линейно возрастать. Это объясняется увеличением расстояния в общем списке между парой событий, найденных во внутрикомпонентном списке.

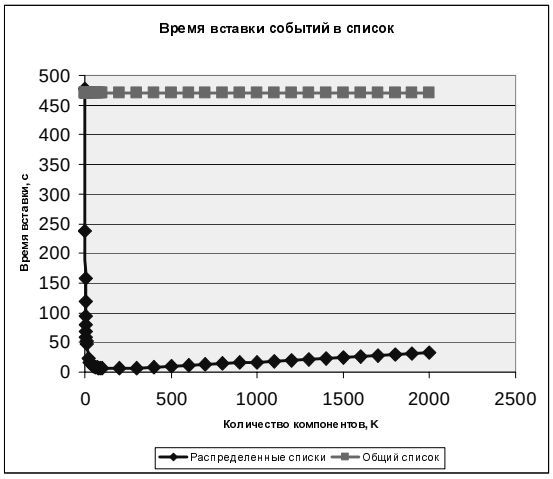

Рисунок 7 – Экспериментально полученное время заполнения распределенного списка событий

Заключение

Разработан новый метод локальных каскадных откатов. Он позволяет ускорить распределенное моделирование за счет уменьшения глубины откатов. Глубина откатов уменьшается путем детализации каскадных откатов, предусмотренных алгоритмом Time Warp.

Метод локальных каскадных откатов использует списки событий, распределенные между моделирующими компонентами. Разработана особая организация общего и внутрикомпонентных списков событий. Элементы общего списка и распределенных списков совмещены в одну общую структуру данных. Разработан алгоритм вставки событий в список с этой организацией. Показано аналитически и подтверждено экспериментально, что использование распределенных списков позволяет сократить время вставки.

Метод локальных каскадных откатов используется в распределенной системе моделирования, основанной на разработанной авторами системе моделирования алгоритмов динамической маршрутизации [5], [6].

Литература

1. Кельтон В., Лоу А. Имитационное моделирование. Классика CS. 3-е изд. – СПб.: Питер; Киев: Издательская группа BHV, 2004. – 847 с.: ил.

2. Fujimoto R.M. Parallel and Distributed Simulation Systems, Wiley Interscience, 2000

3. Chen Gilbert (Gang). New methods for parallel discrete event simulation. // A thesis submitted to the graduate faculty of Rensselaer Polytechnic Institute in partial fulfillment of the requirements for the degree of Doctor of Philosophy Major subject: Computer science. – Troy, New York.: May 2003.

4. Ладыженский Ю.В., Мирецкий А.В. Оптимистический алгоритм синхронизации с внутрикомпонентными откатами для распределенного моделирования // Моделирование и компьютерная графика - 2007: Материалы второй международной научно-технической конференции, г. Донецк, 10-12 октября 2007 года – Донецк, ДонНТУ, Министерство образования и науки Украины, 2007. – 358с. // стр.187 – 192

5. Ладыженский Ю.В., Мирецкий А.В. Моделирование алгоритмов динамической маршрутизации // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: Тези доповідей Міжнародної науково-практичної конференції м. Запоріжжя, 13-15 квітня 2006 року / Під заг. ред. Д.М. Пізи. – Запоріжжя: ЗНТУ. 2006. С. 165-167.

6. Ладыженский Ю.В., Мирецкий А.В. программная система для моделирования алгоритмов динамической маршрутизации // Інтернет-Освіта-Наука-2006, п’ята міжнародна конференція ІОН-2006, 10-14 жовтня, 2006. Збірник матеріалів конференції. Том 2. – Вінниця: Універсум-Вінниця. 2006. С. 372-374.