ПРОГРАММНЫЕ СРЕДСТВА ОПТИМИЗАЦИИ РАСПРЕДЕЛЕННОГО ИМИТАЦИОННОГО ЭКСПЕРИМЕНТА

А.И. Миков, Е.Б. Замятина, А.А. Козлов

Введение

Метод имитационного моделирования является известным, признанным, а иногда, и единственным методом исследования сложных систем. Зачастую имитационный эксперимент требует больших вычислительных ресурсов. В этом случае целесообразно проводить **распределенный** имитационный эксперимент, используя ресурсы нескольких узлов вычислительной системы (ВС) (компьютерной сети, многопроцессорной ЭВМ и т.д.)[1]. Использование вычислительных ресурсов нескольких узлов вычислительной системы позволяет оптимизировать имитационный эксперимент по времени и по надежности (при выходе из строя одного или нескольких вычислительных узлов их функции берут на себя другие узлы). Распределенная имитационная модель (ИМ) представляет собой совокупность логических процессов LPi, где i=1..n, которые выполняются параллельно на различных узлах вычислительной системы (ВС) и обмениваются друг с другом сообщениями. Для синхронизации логических процессов в системах распределенного имитационного моделирования применяют специальные алгоритмы: консервативный, оптимистический[1]. Оба этих алгоритма следят за тем, чтобы не возник парадокс времени - ситуация, когда процесс посылает сообщение со штампом времени, меньшим, чем локальное время процесса, которому отсылают это сообщение.

Однако выигрыш от параллельного выполнения распределенной имитационной модели могут быть сведены к нулю в том случае, если возникнет дисбаланс загрузки вычислительных узлов. Причина дисбаланса: **гетерогенность вычислительной системы** (узлы вычислительной системы имеют разную производительность, а линии связи - разную пропускную способность), **гетерогенность имитационной модели** (некоторые процессы имеют гораздо большую интенсивность обменов, нежели другие, часть процессов оказывают большую нагрузку на вычислительные узлы, выполняя одно событие за другим , в то время как другие могут быть приостановлены, ожидая прихода того или иного сообщения), **нагрузка**, которую оказывают на вычислительные узлы сторонние приложения. Таким образом, целесообразно корректировать возникающий дисбаланс. С этой целью обычно разрабатывают специальное программное обеспечение, которое следит за нагрузкой вычислительных узлов и восстанавливает равномерное распределение приложений по вычислительным узлам, выполняя перенос части компонентов приложений на другие, менее загруженные узлы. При этом программное обеспечение, выполняющее балансировку, следит за передачей сообщений по линиям связи. Нагрузка линий связи также должна быть сбалансирована.

Восстановление баланса нагрузки является хорошо известной задачей, которая получила множество решений. Разработано большое количество алгоритмов. Однако очень часто эти алгоритмы применимы только для конкретного приложения, для решения конкретной задачи. Существуют и решения, которые применимы для оптимизации распределенного имитационного эксперимента[2,3]. Однако, несмотря на то, что разработчики подсистемы балансировки SPEEDES и Charm++ пытались разработать алгоритмы балансировки, которые могли бы адаптироваться к изменяющейся обстановке, к характерным особенностям той или иной имитационной модели, результаты экспериментов показали, что эффективности от применения этих алгоритмов им удалось добиться лишь в частных случаях (описание адаптивного алгоритма SPEEDES рассмотрим ниже). Действительно, найти универсальный алгоритм, который мог бы быть эффективным (сокращение времени выполнения имитационного эксперимента) для любой имитационной модели практически невозможно.

Авторы доклада предложили хотя бы частично разрешить эту проблему, применив **управляемую балансировку**[4]. Управляемая балансировка предполагает, что программное обеспечение, применяемое для восстановления равномерной загрузки должно включать экспертный компонент, который на основании логического вывода предлагает решения для переноса избыточной нагрузки с перегруженного вычислительного узла на менее загруженный. Программные средства балансировки предназначены для системы распределенного имитационного моделирования Triad.Net. (Triad - система автоматизированного проектирования и имитационнного моделирования вычислительных систем[5], Triad.Net[6,7] - ее распределенная версия, для описания имитационной модели используют язык Triad).

Кроме того, для разработки оригинального алгоритма балансировки, учитывающие все особенности конкретного параллельного (распределенного) приложения, авторы предлагают специальные языковые средства. Эти языковые конструкции написаны на входном языке системы моделирования Triad.Net (язык Triad).

Для того, чтобы можно было сократить временные затраты на саму балансировку, авторы предлагают использовать мультиагентный подход, который позволяет применить **децентрализованный** алгоритм балансировки и тем самым сократить время на обмены между компонентами самой подсистемы балансировки. Архитектура мультиагентной подсистемы балансировки, протоколы и алгориты взаимодействия агентов будут описаны ниже.

Задача балансировки

Задача балансировки – это задача отображения неизоморфных связных графов B: TM -> NG, где TM – множество графов моделей, NG – множество графов – конфигураций компьютерной сети. Граф G из NG , G = {C, Ed }, определяется множеством вычислительных узлов C и множеством ребер Ed, обозначающих линии связи. Граф M из TM, задает имитационную модель. Имитационную модель M можно представить и как совокупность логических процессов MP = {LPj}, j=1..n, взаимодействующих между собой путем передачи сообщений.

Система имитации

Описание имитационной модели

Описание модели в системе Triad можно определить как M = {STR, ROUT, MES}, где STR – слой структур, ROUT – слой рутин, MES – слой сообщений. Слой структур представляет собой совокуп-ность объектов, взаимодействующих друг с другом посредством посылки сообщений. Каждый объект имеет полюсы (входные и выходные), которые служат соответственно для приёма и передачи сообщений. Основа представления слоя структур – графы. 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. Рутина представляет собой последовательность событий ei , планирующих друг друга (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 и выходные полюса. Входные полюса служат соответственно для приёма сообщений, выходные полюса – для их передачи. В множестве событий рутины выделено входное событие ein. Все входные полюса рутины обрабатываются входным событием. Обработка выходных полюсов осуществляется остальными событиями рутины. Для передачи сообщения служит специальный оператор out (out <сообщение> through <имя полюса>). Совокупность рутин определяет слой рутин ROUT. Слой сообщений (MES) предназначен для описания сообщений сложной структуры. Система Triad реализована таким образом, что пользователю необязательно описывать все слои. Так, если возникает необходимость в исследовании структурных особенностей модели, то можно описать в модели только слой структур. Алгоритмом имитации называют совокупность объектов, функционирующих по определённым сценариям, и синхронизирующий их алгоритм.

Алгоритм исследования

Для сбора, обработки и анализа имитационных моделей в системе Triad.Net существуют специальные объекты – информационные процедуры и условия моделирования. Информационные процедуры и условия моделирования реализуют алгоритм исследования модели. Информационные процедуры ведут наблюдение за теми элементами модели (событиями, переменными, входными и выходными полюсами), которые указаны пользователем. Если в какой-либо момент времени имитационного эксперимента пользователь решит, что следует установить наблюдение за другими элементами или выполнять иную обработку собираемой информации, он может сделать соответствующие указания, подключив к модели другой набор информационных процедур. Условия моделирования анализируют результат работы информационных процедур и определяют, выполнены ли условия завершения моделирования. Информационные процедуры и условия моделирования используют и для сбора информации о поведении модели, о ее характеристиках и в системе балансировки для сбора информации о модели. Для имитационной модели в Triad возможно использование операций над моделью (добавление, удаление вершины, добавление, удаление дуг и ребер и др.). Операции изменяют структуру имитационной модели во время имитационного прогона, а это в свою очередь требует перераспределения нагрузки на вычислительных узлах. Кроме того, авторы разрабатывают систему имитации с удаленным доступом. Это предполагает балансировку запросов при обращении удаленных пользователей к имитационной модели. В настоящее время реализуется распределен-ная версия Triad.Net. Синхронизация объектов модели, располагающихся на разных вычислительных узлах, выполняется оптимистическим алгоритмом.

Разработка и реализация подсистемы управляемой балансировки (централизованный алгоритм)

Как уже говорилось ранее, управляемая балансировка Triad.Net предполагает наличие экспертного компонента. База знаний экспертного компонента включает знания исследователя о конкретной имитационной модели. К примеру, исследователь знает, что через 100 единиц модельного времени с начала моделирования два объекта имитационной модели, представленные логическими процессами LPi и LPj, должны интенсивно обмениваться информацией в течение некоторого временного интервала. В этом случае целесообразно к моменту времени t = "начало моделирования" + 100 ед.модельного времени перенести эти два процесса на один вычислительный узел или расположить их на соседних вычислительных узлах, если линия связи между ними имеет хорошую пропускную способность. При переносе нагрузки экспертный компонент используют правила, учитывающие знания о топологии ВС и ИМ, знания о нагрузке узлов, линий связи и о функционировании имитационной модели.

Таким образом, система балансировки в Triad.Net включает следующие компоненты (рис.1.): подсистему анализа и принятия решения, подсистему миграции, подсистему мониторинга имитационной модели, подсистему мониторинга вычислительной системы, базу знаний с правилами перераспределения

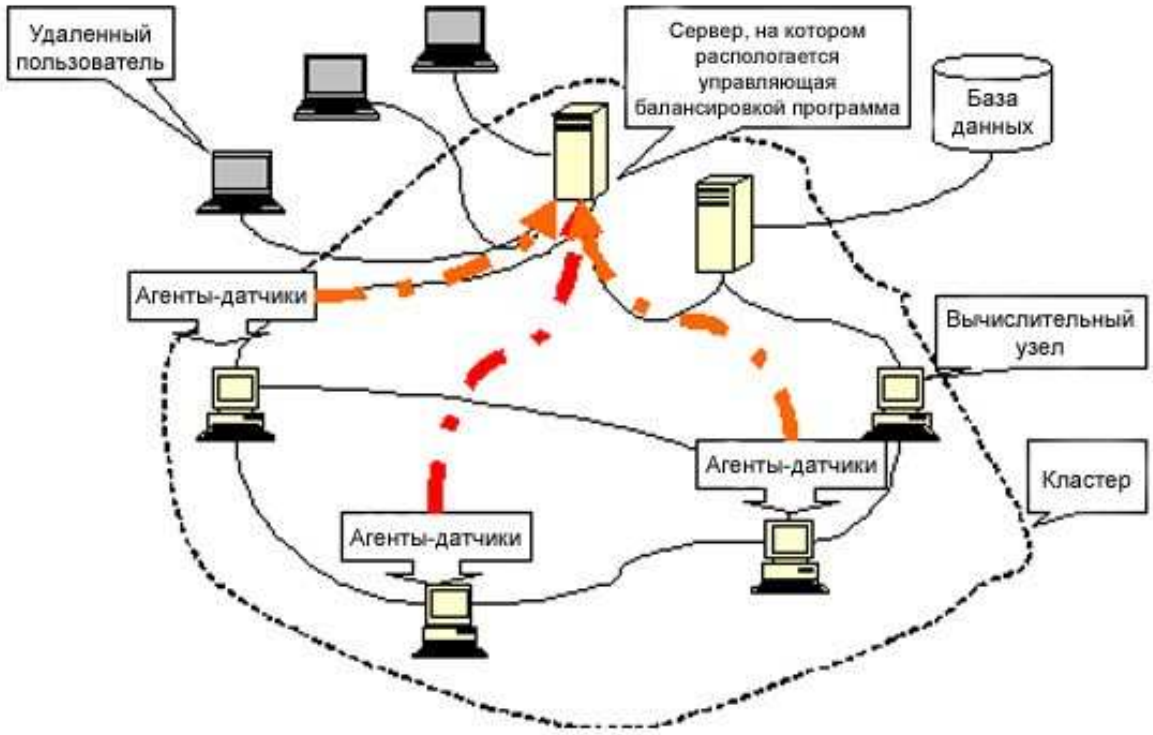

Рис.1.Централизованный алгоритм

нагрузки, редактор правил, механизм вывода и подсистему оценки качества оптимизации.

Подсистема мониторинга ВС собирает информацию о текущем состоянии вычислительной системы, на которой выполняется имитационный эксперимент. Подсистема мониторинга имитационной модели собирает информацию о текущем состоянии имитационной модели (использует механизм информационных процедур). Подсистема анализа получает информацию от подсистем мониторинга и принимает решение о перераспределении нагрузки, после чего обращается к экспертной системе и получает от нее рекомендации о том, какие объекты имитационной модели следует перенести и на какие вычислительные узлы. Далее управление передают подсистеме миграции, которая и осуществляет перенос объектов. Подсистема оценки качества оптимизации определяет, насколько изменилось время имитационного прогона и условия этого прогона.

Однако управление является централизованным, все правила хранятся в единой базе знаний, подсистема мониторинга вычислительной системы также располагается на одном из выделенных вычислительных узлов, взаимодействуя с остальными узлами. Централизованный алгоритм характеризуется временными затратами на коммуникацию.

Мультиагентная подсистема балансировки (децентрализованный алгоритм)

Динамическая система балансировки PSU.HPC.Balance 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 (см.рис.2).

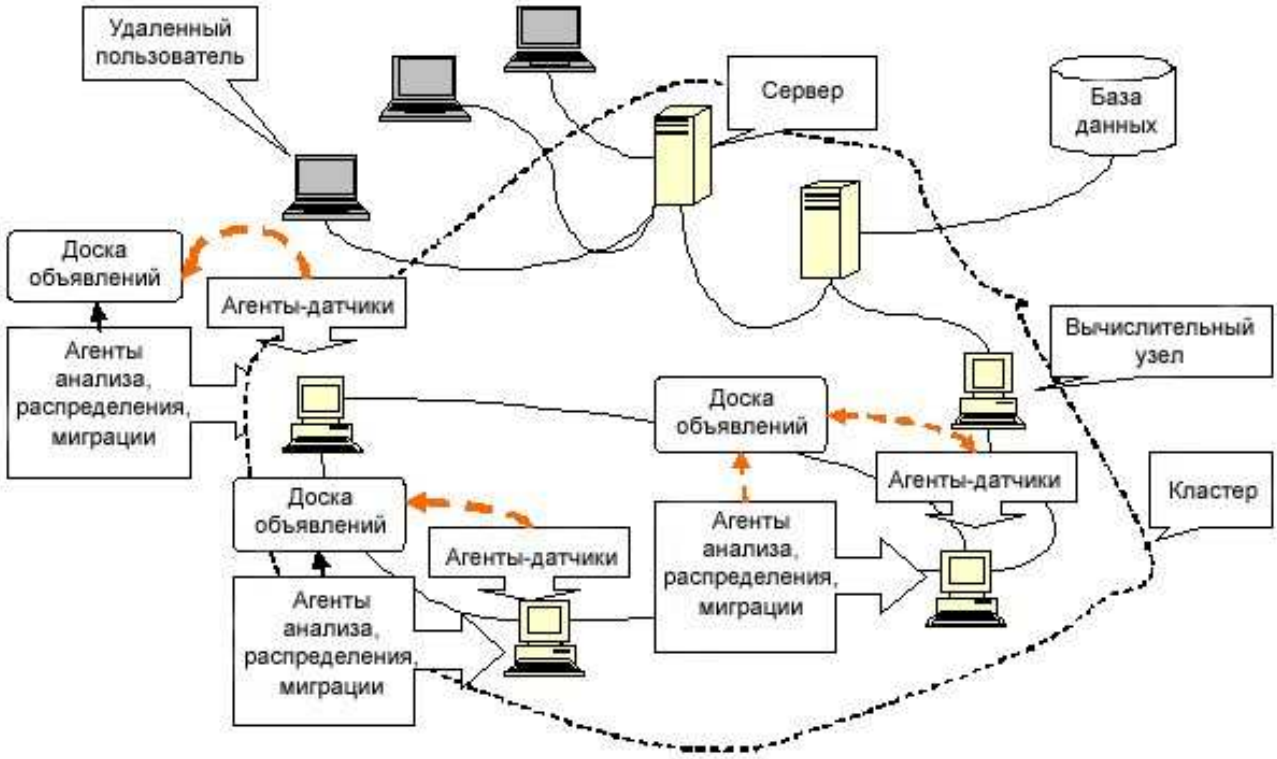

Рис.2.Децентрализованный алгоритм

Агент-датчик вычислительного узла постоянно собирает информацию о нагрузке вычислительного узла, о состоянии линий связи, о наличии свободной памяти. Этот агент взаимодействует с агентом анализа, используя доску объявлений.

Агент-датчик имитационной модели ведет наблюдение за объектами имитационной модели во время имитационного прогона, регистрируя интенсивность обмена между объектами, частоту выполнения тех или иных событий, частоту изменения переменных и т.д. Агент-датчик имитационной модели использует механизм информационных процедур. Он взаимодействует с агентом распределения (опять же используя доску объявлений).

Агент анализа, взаимодействуя с агентом-датчиком вычислительного узла (эти агенты являются реактивными), принимает решение о необходимости перераспределения нагрузки. Он является когнитивным агентом и, принимая решения, использует правила из базы знаний агента. В этих правилах анализируются данные о работе аппаратуры: общее время мониторинга; загруженность процессора (в процентах); общее количество дескрипторов в системе; общее количество потоков в системе и т.д.

Рис. 3. Взаимодействие агента анализа и агента-датчика вычислительной системы

Агент распределения выполняет следующую роль: агент распределения должен решить, который из объектов, если их несколько на одном узле, следует перенести на другой узел. Для решения этой проблемы агент должен располагать информацией, полученной от агента-датчика имитационной модели. Эту информацию агент распределения извлекает с доски объявлений. Агент распределения определяет, на какой вычислительный узел стоит перенести объект имитационной модели. С этой целью он обращается к агентам распределения вычислительных узлов-соседей. Взаимодействие агентов-датчиков, агента-анализа и агента

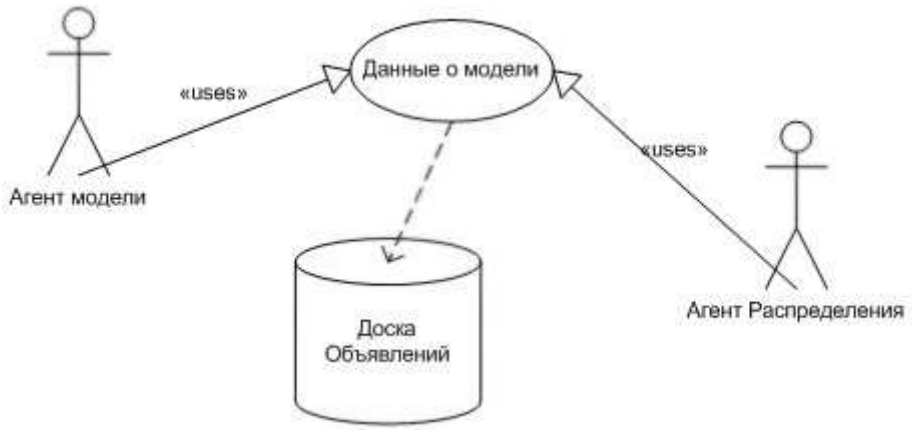

Рис. 4. Взаимодействие агента-датчика имитационной модели и агента распределения

распределения представлены на рис.3. и рис.4.

Итак, алгоритм балансировки является децентрализованным и представляет собой последовательность шагов :

Агент-датчик вычислительной системы (располагаются на каждом вычислительном узле) регулярно на всем протяжении имитационного эксперимента собирает статистику о загруженности узла и размещает ее в базе данных (доска объявлений). Каждый вычислительный узел располагает своей доской объявлений.

Агент анализа, сканируя доску объявлений, с помощью правил из базы знаний определяет, необходимо ли выполнять балансировку. Например, при превышении показателя загрузки вычислительного узла (Node.Pwr > PwrLimit) агент анализа принимает решение о необходимости выполнения балансировки. В этом случае агент анализа обращается к агенту распределения.

Агент распределения извлекает информацию из базы данных. Информация в базе данных появляется в результате работы агента имитационной модели.Для извлечения информации о модели используют механизм информационных процедур. В частности, агенты-датчики собирают информацию о частоте выполения событий имитационных объектов, о 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 объектах. 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, расположенными в базе знаний. Для принятия решения об адресе целевого узла агент распределения, наряду с другими правилами, использует, в частности, правила о топологических характеристик

имитационной модели и вычислительной системы. При выборе целевого узла сообщение напрвляется агенту-миграции.

Агент миграции, получив запрос от агента распределения, выполняет непосредственно перенос выбранных объектов имитационной модели на выбранные целевые узлы сети.

Таким образом, выполняется децентрализованный алгоритм балансировки, так как нет единого управления данным процессом, а каждый вычислительный узел управляет балансировкой на собственном узле, учитывая информацию только соседних вычислительных узлов.

Реализация подсистемы мультиагентной балансировки и результаты экспериментов

Настоящая версия мультиагентной подсистемы динамической балансировки "PSU.HPC.Balance" распределённой имитационной модели реализована на кластере из 8 вычислительных узлов под управлением операционной системы Windows HPC Server 2008 (далее WinHPC), которая является специализированной кластерной (High Performance Computing) версией операционной системы Windows Server 2008. Подсистема разработана с использованием .Net Framework 3.5 и пакета сборок HPC Pack 2008, позволяющего управлять распределённым выполнением приложений в используемой авторами операционной системе. Вычислительным узлом кластера может считаться как отдельный компьютер, так и серверная стойка. Выделяются узлы трех типов: головной узел (Head Node), вычислительный узел (Compute Node) и узел брокера WCF (WCF Broker Node).

Головной узел может также выполнять и другие функции, т.е. быть вычислительным узлом или узлом брокера WCF. Узлы кластера должны быть объединены в локальную сеть по одной из пяти топологий: вычислительные узлы изолированы в частной сети, все узлы присоединены к корпоративной и частной сетям и т.д.. Управление кластерными вычислениями производится через «HPC Job Manager». Для управления задачами системы было использовано программное средство«HPC Pack 2008». Перед запуском системы выполнялось предварительная статическая балансировка модели (начальное размещение объектов модели на вычислительных узлах сети.) Следующий шаг: настройка системы. Пользователь, основываясь на знаниях о модели (он знает, как должна работать модель), модифицирует правила балансировки. На основании указанных правил агенты и принимают решение о переносе объектов модели с одного вычислительного узла сети на другой.

Система балансировки включает редакторы правил и метаправил (для включения новых правил, формируемых когнитивными агентами). Правила формируются в виде XML-файла. В систему балансировки включена подсистема визуализации, с помощью которой можно определить, на каких узлах располагаются объекты имитационной модели. Текущее расположение объектов имитационной модели на вычислительных узлах можно получить в любой момент времени имитационного эксперимента по запросу пользователя. Кроме того, подсистема визуализации позволяет оценить качество оптимизации, отображая в удобных формах загрузку процессоров (и других показателей о функционировании модели и вычислительных узлов во время имитационного прогона).

```
//Клиент
routine Клиент( input ПРИЕМ; output ВЫДАЧА )[ real deltaT
        initial
                boolean запросПослан;
                запросПослан := false;

                schedule ЗАПРОС in 0;
                Print "Инциализация клиента";
        endi

        event ЗАПРОС;
                out "Запрос на обслуживание" through ВЫДА
                Print "Клиент послал запрос серверу";
                schedule ЗАПРОС in deltaT;
        ende
endrout
```

Рис. 5. Описание клиента.

Для проведения экспериментов была разработана имитационная модель "Клиент-Сервер", фрагмент которой приведен на рис.5 и рис. 6.

```
(*
Сервер с очередью заявок
максДлина – максимальная длина очереди
T – время обслуживания
*)
routine Сервер [ integer максДлина; real deltaT ]( input ПРИЕМ; output ВЫДАЧА )
        initial
                boolean серверЗанят;
                integer длина;

                серверЗанят:= false;
                длина := 0;

                Print "Инициализация сервера";
        endi

        event;
                if длина < максДлина then
                        длина := длина + 1;
                        if !серверЗанят then
                                schedule ОКОНЧАНИЕ_ОБСЛУЖИВАНИЯ in deltaT;
                        endif;
                        серверЗанят:= true;
                        Print "Постановка запроса в очередь";
                else
                        Print "Сервер отказал в обслуживании";
                endif
        ende

        event ОКОНЧАНИЕ_ОБСЛУЖИВАНИЯ;
                if длина > 0 then
                        длина := длина – 1;
                        schedule ОКОНЧАНИЕ_ОБСЛУЖИВАНИЯ in deltaT;
                else
                        серверЗанят:= false;
                endif;
                Print "Окончание обслуживания. Текущая длина очереди:";
                Print длина;
        ende
endrout
```

Рис. 6. Описание сервера.

В результате экспериментов удалось получить данные, которые представлены на рис.6 и рис.7. Результаты показали, что время, затраченное на имитационный эксперимент при использовании подсистемы мультиагентной балансировки действительно снижается. Применение подсистемы становится более эффективным при высоких цифрах системного времени моделирования.

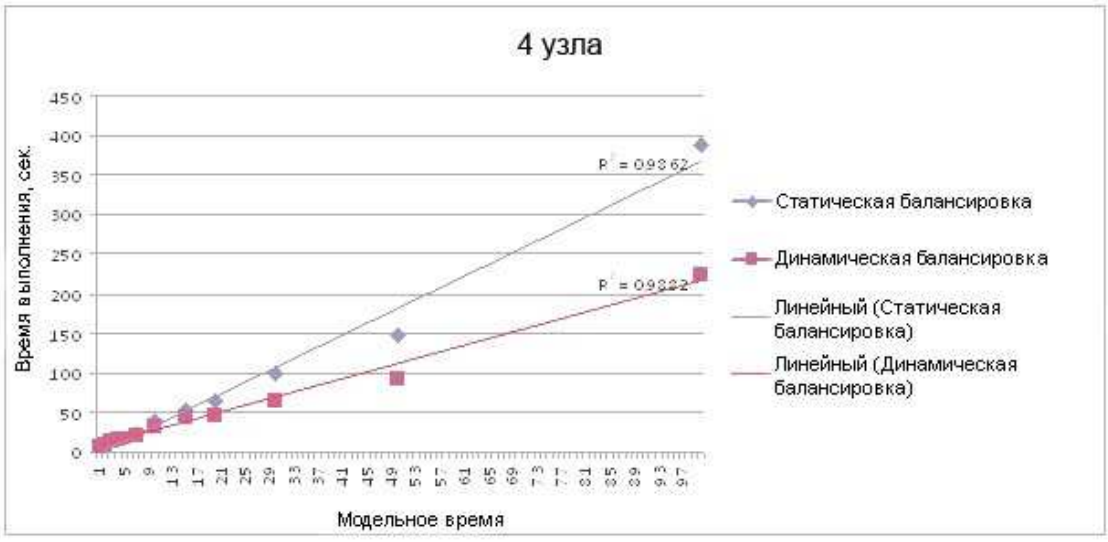

Рис. 7. Результаты экспериментов (4 узла)

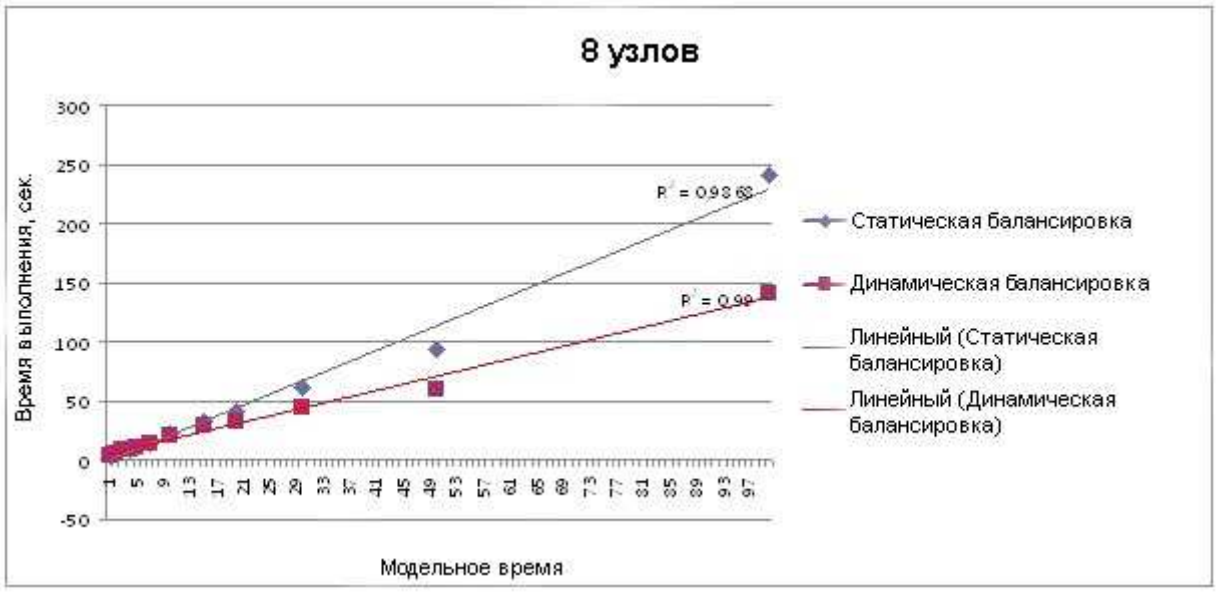

Рис. 8. Результаты экспериментов (8 узлов)

Работа выполнена при финансовой поддержке гранта РФФИ 08-07-90005 Бел_ф

ЛИТЕРАТУРА:

1. Fujimoto R.M. Distributed Simulation Sys-tems. In Proceedings of the 2003 Winter Simu-lation Conference S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, eds. pp. 124-34
2. Wilson L.F. and Wei Shen. Experiments In Load Migration And Dynamic Load Balancing In Speedes. Proceedings of the 1998 Winter Simulation Conference.
3. D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds. Washington. 1998. pp.483-490
4. Zheng G. Achieving High Performance on Extremely Large Parallel Machines: Perform-ance Prediction and Load Balancing; in Ph.D. Thesis, Department of Computer Science. University of Illinois at Urbana-Champaign, 2005,165p. Доступно на сайте: http://charm.cs.uiuc.edu/
5. Миков А.И., Замятина Е.Б., Осмехин К.А. Динамическое распределение объектов имитационной модели, основанное на зна-ниях //Proceedings of XIII International Con-ference “Knowedge-Dialogue-Solution” KDS), ITHEA, Sofia, 2007.Vol.2.,P.618-624
6. 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.
7. Миков А.И., Замятина Е.Б. Технология имитационного моделирования больших систем. Труды Всероссийской научной конференции «Научный сервис в сети Интернет»-М.Изд-во МГУ, 2008,сс.199-204