Оптимизация параллельных вычислений с применением мультиагентной балансировки*

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

Эффективность применения высокопроизводительных вычислительных средств может быть значительно снижена в связи с возникновением дисбаланса нагрузки на вычислительных узлах. Для восстановления баланса используют специальное программное обеспечение, реализующее алгоритмы равномерного распределения нагрузки. В работе приводится архитектура мультиагентной системы балансировки параллельного приложения. Правила, которые используют агенты, восстанавливающие баланс, наряду со знаниями о нагрузке вычислительных узлов, пропускной способности и загруженности линий связи, вычислительной и коммуникационной сложности параллельного приложения, используют знания пользователя о ходе вычислений. В качестве приложения рассматривается параллельная система имитационного моделирования, и приводятся особенности реализации её подсистемы балансировки.

1. Введение

Высокопроизводительные вычисления завоевывают все более прочные позиции при решении разного рода задач (в том числе задач, в которых применяются методы имитационного моделирования), используя ресурсы нескольких исполнителей для выполнения вычислений. Основная цель использования этих средств – оптимизация времени вычислений. Однако гетерогенность исполнителей (вычислительные узлы имеют разную производительность, линии связи между узлами имеют разную пропускную способность), гетерогенность самого параллельного приложения (приложение представляет собой совокупность логических процессов, расположенных на разных вычислительных узлах и взаимодействующих посредством посылки сообщений друг другу) приводит к возникновению дисбаланса нагрузки на вычислительных узлах. В результате выигрыш от использования нескольких исполнителей при выполнении вычислений сводится к нулю.

Для того, чтобы избежать нежелательных последствий дисбаланса используют специальное программное обеспечение, реализующее алгоритм балансировки. Алгоритм балансировки предназначен для равномерного распределения нагрузки на вычислительные узлы. Если на каком-нибудь вычислительном узле нагрузка превышает допустимую, то, следуя алгоритму балансировки, часть нагрузки переносят на другой, менее загруженный узел. При этом следует учитывать затраты приложения на коммуникацию между вычислительными узлами.

В настоящей работе в качестве приложения рассматривается распределенная система имитации (Triad.Net)[1,2] (в дальнейшем будем рассматривать балансировку применительно к распределенной имитационной модели). Распределенная модель представляет собой совокупность объектов, функционирующих на различных вычислительных узлах и взаимодействующих путем посылки сообщений друг другу (реализована с применением технологии .Net).

Существует большое количество различных алгоритмов восстановления равномерного распределения нагрузки на вычислительные узлы во время выполнения распределенного/параллельного приложения[4,5]. Достаточно большое количество работ посвящено и балансировке распределенных имитационных моделей. Авторы работ [6,7] выполнили анализ ряда алгоритмов балансировки для распределенных имитационных моделей. Проведенные ими исследования показали, что чаще всего эти алгоритмы либо требуют изменения кода имитационной модели, либо применимы для очень узкого класса задач.

Следует различать статическую балансировку, которая выполняется до имитационного прогона и динамическую балансировку, выполняемую во время имитационного прогона, не прерывая его. Статическая балансировка предполагает предварительное распределение объек-

* Работа выполнена при финансовой поддержке грантов РФФИ 08-07-90005 Бел_а, 08-07-90006 Бел_а и гранта РФФИ 07-урал_а.

тов имитационной модели по вычислительным узлам сети (или многопроцессорной ЭВМ). В SPEEDES[6] размещение объектов имитационной модели выполняется в зависимости от результатов предыдущих имитационных прогонов.

Однако во время имитационного прогона структура имитационной модели может измениться, что зачастую обусловлено природой моделируемых объектов (мультиагентных систем, например, где динамически изменяются среда, характеристики и поведение агентов, а также и взаимодействие среды и агентов). Для системы имитации Triad.Net (как и в ее последовательной версии Triad [3]) характерно динамическое изменение структуры имитационной модели во время вычислительного эксперимента и, соответственно, изменение нагрузки на вычислительных узлах. Таким образом, необходимость в динамической балансировке еще более возрастает.

Авторы SPEEDES[6] предложили адаптивный алгоритм, который эффективно применим для более широкого класса задач. Ими была разработана RCL-стратегия - стратегия выбора порции нагрузки на перегруженном узле с целью переноса ее на менее загруженный узел. R (Random) - стратегия представляет собой алгоритм выбора порции нагрузки на вычислительном узле случайным образом с тем, чтобы впоследствии выполнить перемещение ее на другой узел с помощью процедуры Migrate(). C (Communication) - стратегия позволяет выбрать объекты, размещенные на различных вычислительных узлах и наиболее интенсивно обменивающиеся сообщениями, и перенести их на другие узлы или узел с целью снижения временных затрат на обмен информацией. L (Load) – стратегия реализует алгоритм, основанный на выборе объектов, которые дают наибольшую нагрузку на вычислительный узел. Каждому объекту на вычислительном узле предварительно приписывается число, определяющее эту нагрузку. Число вычисляют аналитически до выполнения балансировки. Авторы провели многочисленные эксперименты, которые показали, что, используя различные стратегии в конкретных ситуациях, можно добиться сокращения времени выполнения имитационного эксперимента.

Таким образом, прослеживается тенденция найти *эффективный* алгоритм, который может быть применен для имитационной модели *любой конфигурации*, для *любого алгоритма* синхронизации распределенных объектов модели (известно, что для синхронизации объектов используют различные модификации консервативного и оптимистического алгоритмов).

Однако задачи, которые решают с применением методов имитационного моделирования, и соответственно имитационные модели, отличаются большим разнообразием. Следовательно, найти универсальный алгоритм для балансировки весьма сложно.

Авторы настоящей работы предлагают использовать *управляемую* балансировку, основанную на знаниях пользователя о конкретной модели[2]. Алгоритм балансировки использует правила, которые хранятся в базе знаний. Правила могут быть отредактированы пользователем перед проведением распределенного имитационного эксперимента. Именно автор имитационной модели знает особенности ее поведения. В настоящей работе авторы предлагают применить мультиагентный подход для реализации управляемой динамической балансировки.

2. Подсистема балансировки в системе имитации Triad.Net

2.1 Постановка задачи балансировки

Задачу балансировки можно определить как задачу отображения неизоморфных связных графов, B: TM $\rightarrow$ NG, где TM – множество графов моделей, NG – множество графов – конфигураций компьютерной сети.

Граф G $\in$ NG, где G = {C, E_d}, определяется множеством вычислительных узлов C и множеством ребер E_d, обозначающих линии связи.

Граф NG можно рассматривать как суперграф, содержащий все возможные (допустимые) графы G в качестве подграфов.

Граф M $\in$ TM , M = задает имитационную модель, U – множество входных и выходных полюсов вершины, V – множество вершин, W – множество ребер.

Прежде, чем приступить к описанию подсистемы балансировки, рассмотрим архитектуру имитационной модели.

Имитационная модель в Triad представлена тремя слоями: M = {STR, ROUT, MES}, где STR – слой структур (совокупность объектов и связей между ними), ROUT – слой рутин, MES – слой сообщений.

Слой структур представляет собой совокупность 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 e_i , планирующих друг друга ($e_i \in E$, $i=1 \div n$, E - множество событий, множество событий рутины является частично упорядоченным в модельном времени). Выполнение события сопровождается изменением состояния объекта. Состояние объекта определяется значениями переменных рутины. Модель в Triad является иерархической. На каждом уровне иерархии она представлена графом P={U, V, W}.

Для сбора информации о поведении модели используют информационные процедуры, которые следят за изменением локальных переменных рутины, полюсов и событий. Для управления имитационным прогоном – условия моделирования.

2.2 Подсистема балансировки

В Triad.Net рассматриваются три разновидности задачи балансировки[2]: статическая B_s, динамическая (автоматическая) B_a и динамическая (управляемая) B_c.

В алгоритме статической балансировке B_s наилучшим результатом считается отыскание подграфа $G \subset NG$, изоморфного графу – модели M. Однако такой подграф существует далеко не всегда, поэтому предлагается метод отыскания в некотором смысле «близкого» подграфа.

В алгоритме автоматической динамической балансировки B_a графы G и M рассматриваются как нагруженные. Вершинам первого графа приписывается параметр – производительность, а его ребрам – скорость передачи данных. Во втором графе вершины характеризуются временной сложностью вычислений, а ребра – интенсивностью потоков сообщений (выходных событий).

Веса вершин и ребер графа NG (и, значит, любых его подграфов) считаются известными. Соответствующие параметры графа M должны определяться во время имитационного прогона. В соответствии с некоторым алгоритмом происходит определение «узких» мест ВС и имитационной модели и выполняется перенос объектов на менее загруженные узлы, не прерывая процесса моделирования. Алгоритм автоматической балансировки можно описать на языке Triad[8].

На основании собранной статистики, используя генетические алгоритмы, для последующих имитационных прогонов конкретной модели можно найти лучшее распределение объектов по вершинам графа G.

И, наконец, B_c - алгоритм динамической балансировки, основанной на знаниях.

2.2.1 Архитектура управляемой балансировки с централизованной базой знаний

Первоначально для реализации управляемой динамической балансировки было разработано программное обеспечение, которое включало:

- Подсистему анализа и принятия решения.
- Подсистему миграции.
- Подсистему мониторинга имитационной модели.
- Подсистему мониторинга вычислительной системы.
- Базу знаний с правилами перераспределения нагрузки.
- Редактор правил.
- Механизм вывода.

В базе знаний содержатся правила миграции, которые используют данные о текущем состоянии модели (частота обменов сообщениями между логическими процессами, частота выполнения тех или иных событий и т.д.) и текущем состоянии ВС (загрузка процессоров, загруз-

ка линий связи). Наряду с ними в БЗ содержатся данные о поведении конкретной имитационной модели.

Например, пользователь знает, что через определенный промежуток времени (100 единиц модельного времени) интенсивность обмена между двумя логическими процессами значительно увеличится. В этом случае целесообразно принять решение о переносе интенсивно взаимодействующих процессов на соседние узлы с соответствующей пропускной способностью линий связи (или расположить их на одном узле).

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.

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, а это увеличивает время на коммуникации в подсистеме балансировки.

В настоящее время разрабатывается мультиагентная версия балансировки.

2.2.2 Мультиагентная система балансировки нагрузки

Динамическая система балансировки TriadBalance 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. Агенты – аппаратно-программные сущности, они могут действовать автономно. Агенты взаимодействуют друг с другом и с внешней средой. В TriadBalance используется сетевая модель взаимодействия агентов.

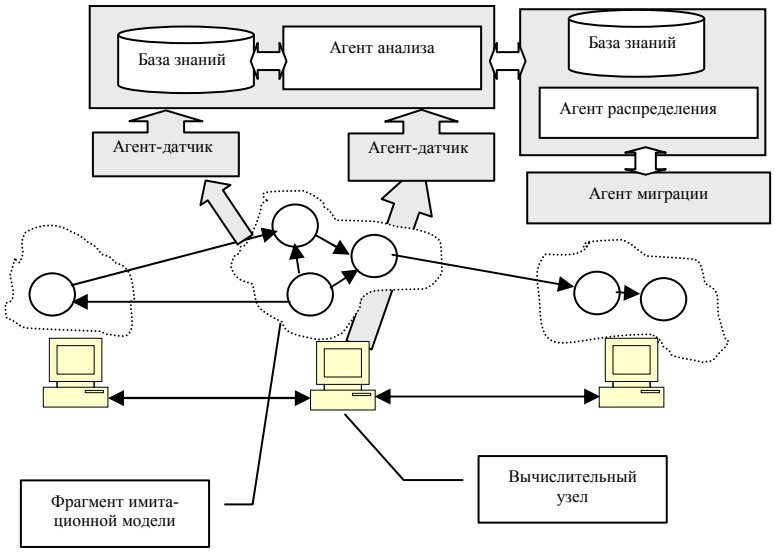

Рис.1. Архитектура мультиагентной подсистемы балансировки

Агенты могут быть определены следующим образом: A=<Sens, Eff, Prog, I,O, R, MR, E, G>, где Sens – сенсоры агента, функции, с помощью которых агент получает информацию о внешней среде, Eff – эффекторы, функции, с помощью которых агент воздействует на внешнюю

среду, Prog : P→O – преобразование входной информации (I) в выходную (O), MR- метаправила,R – правила, по которым действует агент (метаправила и правила характерны для когнитивных агентов), E – внешняя среда (вычислительная среда: сеть, многопроцессорная ЭВМ, кластер, GRID), G – цель, которую пытается достичь агент.

Агенты анализа и распределения определены как когнитивные. Агенты датчики и агент миграции являются реактивными. На рис. 1. представлена архитектура системы балансировки на каждом узле.

Пользователь, основываясь на знаниях о модели (он знает, как должна работать модель), модифицирует правила балансировки. На основании данных правил, агенты будут принимать решение о переносе объектов модели с одного вычислительного узла сети на другой.

На каждом вычислительном узле располагаются агенты 5 типов. Рассмотрим подробнее назначение каждого из них.

Агент-датчик вычислительной системы собирает данные о состоянии вычислительного узла. В числе этих данных: нагрузка на вычислительные узлы, загруженность линий связи. При сборе информации используют счетчики производительности[9].

Агент-датчик имитационной модели ведет наблюдение за изменением состояния объектов имитационной модели, расположенных на узле. В качестве данных, которые агент-датчик предоставляет в качестве выходной информации частоту совершения события, частоту принятия и посылки сообщений с полюсов, частоту изменения состояния и т.д. В качестве агента-датчика в TriadBalance используют информационные процедуры [10].

Агент анализа опрашивает агенты анализа через определенные промежутки времени, решает (используя правила), есть ли необходимость в балансировке. Если это так, то происходит обращение к агенту распределения. Агент распределения является источником «знаний» об окружающей среде (объекты-соседи, расположенные на соседних вычислительных узлах, статистическая информация о соседних узлах) для агента анализа. Эти знания предназначены для уточнения правил, руководствуясь которыми агент принимает решение о необходимости проведения балансировки.

Агент распределения на основании правил (множество R) принимает решение о том, какие именно объекты модели необходимо переносить, а также выбирает целевые вычислительные узлы сети.

Агент распределения, выбрав объекты для переноса на другие вычислительные узлы, обращается к соседним агентам распределения. Далее выполняется синхронизация агентов распределения, в результате которой определяют агент-лидер (будем считать, что к началу выполнения алгоритма синхронизации, все процессы находятся в одном состоянии - агенты распределения готовы выполнить остановку системы).

После выполнения алгоритма синхронизации агент-лидер останавливает процесс моделирования, отправив системе имитации соответствующее сообщение. Далее агенты запрашивают необходимые объекты у системы (в виде потоков сериализованных данных). Перенос объектов на другие узлы выполняется *агентом миграции.* Агент миграции передает объекты имитационной модели агентам распределения целевых узлов. После завершения процедуры миграции опять происходит синхронизация агентов распределения и запуск системы имитации. Процесс моделирования продолжается. При этом продолжается наблюдение за состоянием вычислительных узлов и перераспределение нагрузки.

Реализованное на настоящее время программное обеспечение было протестировано на моделях “ILLIACK IV” (матрица узлов 1024×1024) и «Клиент сервер». Тестирование выполнялось на кластере механико-математического факультета ПГУ. Время выполнения имитационного прогона при использовании балансировки сократилось примерно на 10%.

3. Заключение

В работе рассматривается архитектура подсистеме балансировки. При реализации был использован мультиагентный подход. Мультиагентный подход позволил:

- Равномерно распределить нагрузку, которая появляется при функционировании подсистемы балансировки по вычислительным узлам.

- Ускорить выполнение балансировки, так как агенты-датчики находятся на тех вычислительных узлах, для которых они должны получить статистические данные об функционировании этих узлов.
- Унифицировать систему балансировки для различных систем имитации.

В настоящее время авторы продолжают проводить эксперименты с мультиагенной системой балансировки, используя различные имитационные модели, алгоритмы продвижения времени (консервативный, оптимистический), уточняя правила когнитивных агентов и применяя агентов, обученных на нейронных сетях.

Литература

1. Миков А.И., Замятина Е.Б., Фатыхов А.Х. Система оперирования распределёнными имитационными моделями сетей телекоммуникаций. Труды Второй Всероссийской научной конференции «Методы и средства обработки информации». М.: Изд-во МГУ, 2003 г.
2. Миков А.И., Замятина Е.Б., Осмехин К.А. Динамическое распределение объектов имитационной модели, основанное на знаниях. //Proceedings of XIII International Conference “Knowedge-Dialogue-Solution” KDS, ITHEA, Sofia, 2007.Vol.2.,pp.618-624
3. Mikov A.I. Formal Method for Design of Dynamic Objects and Its Implementation in CAD Systems // Gero J.S. and F.Sudweeks F.(eds), Advances in Formal Design Methods for CAD, Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design, Mexico, Mexico, 1995. pp.105-127.
4. Копысов С.П. Динамическая балансировка нагрузки для параллельного распределённого МДО // Труды Первой Всероссийской научной конференции «Методы и средства обработки информации» / МГУ. Москва, 2003. С. 222-228.
5. Курилов Л.С. Прогностическая стратегия балансировки загрузки для невыделенных кластерных систем // Труды Первой Всероссийской научной конференции «Методы и средства обработки информации» / МГУ. Москва, 2003. С. 413-418.
6. Wilson L.F., Shen W. Experiments In Load Migration And Dynamic Load Balancing In Speedes // Proceedings of the 1998 Winter Simulation Conference. D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds, . pp.590-596
7. Zheng G. Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing; in Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005. 165 p. Доступно на сайте: http:// charm. cs.uiuc. edu/
8. Миков А.И.Языковые средства управления балансировкой распределенной имитационной модели // Математика программных систем: Межвуз. Сб. науч. тр./Перм.ун-т.-Пермь. 2007. 4-15 с.
9. Замятина Е.Б., Усынин А.С. Компонент мониторинга функционирования вычислительной системы для динамической балансировки нагрузки вычислительных узлов. //Математика программных систем: Межвуз. Сб. науч. тр./Перм.ун-т.-Пермь.2007. 28-39с.
10. Замятина Е.Б. Компонент сбора информации о модели в подсистеме балансировки //Математика программных систем: Межвуз. Сб. науч. тр./Перм.ун-т.-Пермь.2007. 40-49с.