УДК 004.89:004.48

В.И. Шинкаренко, Д.В. Олейник

ПРОГРАММНЫЕ АГЕНТЫ В ОРГАНИЗАЦИИ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ АДАПТИВНОГО СИНТЕЗА И ОБУЧЕНИЯ НЕЙРОСЕТЕЙ

Разработана среда для выполнения распределенных вычислений в гетерогенных и много платформенных компьютерных сетях. Представлена организация вычислительного процесса, управляемого агентами. Выполнена детализация организационных решений для адаптивного формирования и обучения многослойных прямоточных нейросетей на основе программных агентов. Приведена методика оценки временной эффективности предложенной организации вычислительного процесса.

Введение

Постоянно возрастающие объемы информации и интенсивность информационных потоков в автоматизированных системах повышают нагрузки на СУБД. Эффективность СУБД снижается и в некоторых случаях время выполнения запросов прекращает удовлетворять пользователей.

Одним из предлагаемых решений для устранения этой проблемы является адаптация СУБД к изменяющимся условиям функционирования. Предполагается изменение некоторых управляющих параметров СУБД и структуры БД при изменении параметров рабочей нагрузки. Изучение возможностей такой адаптации, эффективности подбора управляющих параметров предполагается на соответствующих моделях, в том числе и моделях рабочей нагрузки СУБД, представимых в виде временных рядов.

В результате изучения существующих подходов и средств моделирования временных рядов, поставлена задача нейросетевого моделирования на основе прямоточных многослойных сетей с гибридным алгоритмом обучения на основе генетического алгоритма и алгоритма обратного распространения ошибки [1].

При решении задач нейросетевого моделирования возникает необходимость поиска эффективных структур нейросетей (НС) и параметров алгоритма обучения [2, 3]. Специфика синтеза и адаптации структур НС на основе анализа качества моделирования обученных НС заключается в потребности значительных временных и технических ресурсов.

Особенности метода адаптивного формирования (синтеза) нейросетей для моделирования рабочей нагрузки СУБД приведены в [4]. В данной работе рассматриваются вопросы, связанные с организацией вычислений, обеспечивающих высокую временную эффективность вычислительных процессов.

Основным решением, позволяющим повысить производительность вычислительных систем средствами организации вычислительного процесса, является распараллеливание вычислений. На основе опыта решения таких задач, как астрономические расчёты, прогнозирование погодно-климатических условий, моделирование ядерных испытаний, дешифрование, эволюционные вычисления построена программно-техническая база на основе суперкомпьютеров [5], кластерных систем [6 – 10], систем распределённых вычислений [11 – 13].

Так сложилось исторически, что сначала параллельные вычисления появились на базе суперкомпьютеров, поэтому их достоинства и недостатки наиболее очевидны. Главное достоинство суперкомпьютеров (после производительности) – высокая скорость обмена данными между вычислительными устройствами ввиду размещения данных в общей памяти. Основной недостаток – стоимость, так как каждый суперкомпьютер, по сути, искусственное изделие сложной архитектуры.

Кластер – набор компьютеров, рассматриваемый ОС, системным программным обеспечением, программными приложениями и пользователями как единая система [12]. Использование кластера позволяет построить высокопроизводительные, отказоустойчивые вычислительные системы. Основные области применения кластеров – системы управления базами данных, одновременная работа большого количества пользователей, научные приложения.

Особенностью кластерных систем является гомогенность их узлов и наличие на каждом узле установленной специализированной операционной системы. Компьютеры, входящие в кластер, соединяются локальной сетью, что может стать узким местом вычислительной системы при активном обмене данными между узлами кластера [10]. Тем не менее, кластерные системы занимают достаточно большой сегмент суперкомпьютерного рынка, предлагая конкурентоспособную производительность за существенно более низкую стоимость.

Распределённые вычисления (distributed computing, grid computing, volunteer computing) – способ решения трудоёмких вычислительных задач с использованием двух и более компьютеров объединённых в сеть [14]. Особенность систем распределённых вычислений: аппаратная и программная гетерогенность узлов. Задачи, решаемые такими системами, в основном, вычислительного характера, причём важным аспектом является возможность сегментации общей задачи на независимые подзадачи.

С ростом популярности Интернета появилась возможность существенно увеличить количество узлов, входящих в систему. Так, проект распределённых вычислений в интернете SETI@Home [15] по вычислительной мощности сравним с современными суперкомпьютерами.

В данной работе рассматривается организация вычислительного процесса в гетерогенной среде распределённых вычислений с использованием агентных технологий [16, 17], а также применение интеллектуальных программных агентов в задачах адаптивного формирования и обучения НС. Преимущество такого подхода проявляется в использовании для вычислений скрытых ресурсов неспециализированных технических и программных средств.

Насколько эффективны предложенные решения по организации вычислительного процесса, предлагается оценивать по показателю степени превосходства временной эффективности алгоритмов [18].

1. Организация вычислительного процесса

Необходимо выделить предпосылки, предопределившие особенности организации рассматриваемого вычислительного процесса. Эти предпосылки связанны с текущим состоянием информатизации организаций, задействованных в сфере производства, торговли, оказания услуг, науки и образования.

В настоящее время на предприятиях имеется обширный парк компьютеров, объединённых в локальные или распределенные корпоративные сети.

В торгово-производственных организациях использование компьютеров сводится к работе в различных офисных программах и одной или нескольких учётных системах, кроме того, на предприятиях внедряются серверы приложений и терминальные серверы (типа RDP или Citrix MetaFrame), которые снимают с локальных компьютеров вычислительные нагрузки. В учебных заведениях на компьютерах функционирует математическое программное обеспечение, программы для проектирования и моделирования, среды программной разработки, офисные пакеты.

Загруженность компьютеров в организациях подчинена определённым временным закономерностям. В торговых и учебных заведениях активность использования парка компьютеров приходится на дневное время – остальную часть суток вычислительные ресурсы не используются. На производстве с круглосуточной загруженностью можно выделить периоды простоя или значительного снижения нагрузки.

В подавляющем большинстве случаев вычислительные ресурсы компьютера: процессора, оперативной памяти, винчестера и т.п. загружены незначительно, часто на 1 – 5 %.

В настоящей работе организация вычислительного процесса направлена на то, чтобы использовать невостребованные вычислительные мощности. Фактически это решается путём создания среды распределённых вычислений, в рамках которой будет выполняться вычислительный процесс.

Для организации вычислительного процесса реализовано два независимых уровня: среда распределённых вычислений (СРВ) и исполнители вычислений. СРВ может функционировать независимо от того, работают исполнители вычислений или нет, а исполнители вычислений, в свою очередь, могут быть запущены и без использования среды. Такой подход делает СРВ достаточно гибким механизмом, позволяющим выполнять различные вычислительные процессы без привязки к среде их реализации.

1.1 Среда распределённых вычислений

СРВ состоит из центра управления (ЦУ) на выделенном компьютере и множества исполнительных узлов (ИУ) на отдельных ЭВМ сети. Назначение СРВ заключается в предоставлении возможности запуска с ЦУ необходимых программ на многих ИУ, предоставлении центру управления информации о работоспособности исполнительных узлов, а также передаче полученных результатов с ИУ на ЦУ.

ИУ должен быть проинсталлирован на каждом компьютере, включаемом в состав СРВ. Обеспечивается восстановление работы исполнительного узла при перезагрузке компьютера (с использованием системного реестра). В момент запуска ИУ регистрируется на ЦУ и ждёт его команд. Центр управления при запуске регистрирует запущенные ИУ и периодически опрашивает их, контролируя работоспособность.

ЦУ предоставляет пользователю консоль для ввода команд, отправляемых на выполнение исполнительным узлам, и визуализации отклика исполнительных узлов. При этом центр управления обрабатывает как единичные команды, так и пакеты команд, которые могут быть созданы предварительно (в текстовом файле) или интерактивно вводом с консоли. Общий формат команд, посылаемых от ЦУ к ИУ, выглядит так:

<ip>@<имя команды>[#<параметры команды>],

где <ip> – ip-адрес ЭВМ ИУ,

<параметры команды>:=[show|hide] <имя процесса> [<параметры процесса>],

где show/hide – признаки визуализации/скрытия окна процесса, <имя процесса> – название исполнимого или пакетного файла, <параметры процесса> – параметры, передаваемые процессу через командную строку.

Полный набор команд приведён в табл. 1.

Для работы исполнителей вычислений необходимо наличие на ИУ определённых исполнимых файлов, библиотек, а также файлов ресурсов разного рода. Эти файлы могут время от времени изменяться (исправление ошибок, изменение требований к вычислительному процессу). Изменения должны актуализироваться на каждом исполнительном узле.

СРВ предоставляет механизм обновления файлов для ИУ. Для этого на компьютере ЦУ создаётся открытая по сети папка !UPDATE, в которую выкладываются обновлённые файлы. При передаче исполнительному узлу соответствующей команды, он копирует себе все файлы из этой папки и запускает процесс самообновления, если в пакет обновления входит откорректированный исполнимый файл ИУ.

Следует подчеркнуть, что сама по себе СРВ никак не связана с агентными технологиями. С её помощью можно решать совершенно разные задачи. Например, можно использовать СРВ для параллельного решения нескольких независимых систем дифференциальных уравнений с помощью математического пакета

Таблица 1. Команды взаимодействия центра управления и исполнительных узлов СРВ

Команды	Назначение	Ответ клиента
<ip>@GetSystemIdle	Определить загруженность процессора ИУ	<ip>@GetSystemIdle: % загрузки
<ip>@GetProcList	Получить список запущенных процессов на компьютере ИУ	Список процессов клиента
<ip>@UpdateFiles	Обновить файлы, предназначенные для работы исполнителей вычислений. Файлы находятся в открытой по сети папке !UPDATE на ЦУ	<ip>@OK – обновление прошло успешно
<ip>@KillProcByName# <имя процесса>	Снять с выполнения процесс на ИУ	<ip>@OK – действие выполнено успешно
<ip>@ShellExecute# [show|hide] <имя процесса> <параметры>	Запустить на выполнение процесс на ИУ с указанием параметров запуска	<ip>@OK – действие выполнено успешно
<ip>@GetLogs	Скопировать результаты вычислительного процесса (файлы *.log) с ИУ на ЦУ	<ip>@OK – действие выполнено успешно

MAPLE. Пусть центр управления СРВ расположен на компьютере с ip-адресом 192.168.1.1, а исполнительные узлы СРВ загружены на трёх компьютерах сети ip-адресами: 192.168.1.2, 192.168.1.3, 192.168.1.4. Три программы решения разных систем дифференциальных уравнений на языке MAPLE сохранены в файлах prog1.mws, prog2.mws, prog3.mws. Порядок действий для организации такого вычислительного процесса с использованием СРВ:

– поместить файлы (cwmaple.exe, *.dll, prog1.mws, prog2.mws, prog3.mws) в папку \\192.168.1.1\!UPDATE на ЦУ;

– запустить файл с пакетом команд обновления для всех ИУ:

192.168.1.2@UpdateFiles
192.168.1.3@UpdateFiles
192.168.1.4@UpdateFiles

– для запуска параллельного решения систем дифференциальных уравнений необходимо запустить файл с таким пакетом команд:

192.168.1.2@ShellExecute#show cwmaple.exe prog1.mws
192.168.1.3@ShellExecute#show cwmaple.exe prog2.mws
192.168.1.4@ShellExecute#show cwmaple.exe prog3.mws

– результаты расчёта сохраняются в файлы result.log на каждом ИУ. Для сбора результатов в ЦУ необходимо выполнить такой пакет команд:

192.168.1.2@GetLogs
192.168.1.3@GetLogs
192.168.1.4@GetLogs

– после чего результаты будут скопированы в папки ЦУ, название которых совпадает с ip-адресами ИУ.

Рассматриваемая далее работа агентов (исполнителей вычислений) инициализируется на исполнительных узлах СРВ средствами центра управления.

1.2. Исполнители вычислений

Вычислительный процесс адаптивного формирования и обучения нейронных сетей реализован с использованием агентно-ориентированного подхода. Исполнителями вычислений являются агенты, объединённые в группы: постановщики задач, эксперты, диспетчеры и исполнители. Будем обозначать агента определённой группы префиксом А-, например, А-постановщик.

Основной обязанностью А-постановщиков задач является формирование и адаптация эффективных структур НС. Они взаимодействуют: с А-экспертами для получения рекомендуемых параметров алгоритма обучения; с А-диспетчерами, которым передаются

структуры НС и параметры алгоритма обучения с целью получения оценок структур НС.

В обязанности А-диспетчеров входит эффективное распределение вычислений среди доступных А-исполнителей и сбор оценок качества обучения НС.

Главная обязанность А-исполнителей – по заданной структуре НС и параметрах алгоритма обучения выполнить обучение НС и получить оценку качества её обучения. У А-исполнителей пассивная роль. С ними взаимодействуют А-диспетчера, которые предоставляют задание и требуют результатов.

А-эксперты должны найти рациональные параметры алгоритма обучения НС. Поэтому А-эксперты взаимодействуют, во-первых, с А-постановщиками, которым они передают параметры алгоритма обучения и получают достигнутые критерии качества НС при этих параметрах, во-вторых, между собой – для принятия компромиссных решений.

1.3. Управление вычислительным процессом

Важным аспектом устойчивости вычислительного эксперимента является планирование распределения пакетов заданий для выполнения на исполнительных узлах СРВ. Ответственность за это лежит на А-диспетчерах. Сформированный пакет задания включает в себя описание структуры нейросети и параметры алгоритма обучения для однократного обучения НС.

А-диспетчер раздаёт пакеты заданий свободным А-исполнителям. После того, как А-исполнитель закончил расчёт и вернул результат, ему передаётся следующий пакет задания. Когда все пакеты заданий розданы, но имеются в наличии свободные А-исполнители, тогда А-диспетчер раздаёт уже выданные, но недосчитанные, пакеты заданий свободным А-исполнителям. Таким образом, один и тот же пакет задания может обрабатываться двумя и более агентами-исполнителями.

Такой подход эффективен, если в СРВ участвуют компьютеры с разной производительностью, т. е. свободные А-исполнители пытаются помочь отстающему. Как только один из А-исполнителей решает продублированный пакет задания и сообщает об этом А-диспетчеру, тот посылает остальным А-исполнителям команду на прекращение обработки.

Кроме того, А-диспетчер контролирует работоспособность А-исполнителей, выполняющих пакет задания. А-исполнитель может находиться в трёх состояниях: свободен, занят решением, хранение результата. Если состояние А-исполнителя «свободен», тогда А-диспетчер поручает ему выполнение нового задания. Если А-исполнитель находится в состоянии «занят решением», тогда А-диспетчер просто фиксирует его доступность. Если А-исполнитель находится в состоянии «хранит результат», тогда А-диспетчер запрашивает у него результат, после чего А-исполнитель переходит в состояние «свободен». Если А-исполнитель становится недоступным (перегрузили исполнительный узел агента, пропала связь по сети или сняли процесс агента), тогда А-диспетчер перераспределяет пакет задания другому свободному А-исполнителю.

Такие действия А-диспетчера позволяют гарантировать выполнение вычислительного процесса до тех пор, пока доступен хотя бы один А-исполнитель. Дополнительно А-диспетчер отслеживает появление новых А-исполнителей и динамически подключает их к вычислительному процессу.

Если вычислительный процесс прекратился (перегрузили компьютер с А-диспетчером или А-постановщиком, пропала связь со всеми агентами исполнителями), то его можно возобновить с той точки, на которой произошла остановка.

Для возобновления формирования структур НС необходимо инициализировать А-постановщика последними доступными результатами предыдущего процесса и возобновить процесс.

При остановке вычислительного процесса знания А-экспертов о параметрах алгоритма обучения также будут сохранены. А-эксперты хранят факты о параметрах алгоритма обучения и результатах обучения НС при этих параметрах в тек-

стовых файлах, имеющих теговую структуру. Периодически (первый раз при старте агента) из этих фактов А-эксперты извлекают знания о том, какие параметры наиболее эффективны.

Таким образом, при остановке вычислительного процесса и последующем его возобновлении А-эксперты просто повторно извлекут знания из имеющихся фактов.

Разработанная организация вычислительного процесса адаптивного формирования и обучения НС позволяет выполнять вычисления в гетерогенной сети с возможностью управления и контроля за ходом вычислений из единого центра.

2. Общая модель взаимодействия агентов

Задача адаптивного формирования и обучения НС состоит из двух отдельных подзадач: адаптивного формирования структур НС и адаптивного обучения НС.

Первая подзадача решается с использованием генетического алгоритма параметрической адаптации (ГАПА). С помощью этого алгоритма А-постановщик задач формирует структуры НС, которые попадут в пакет заданий.

Вторая подзадача решается адаптацией параметров гибридного генетического алгоритма с градиентным дообучением лидеров (ГГАО) [1], используемого для получения оценок НС. ГГАО реализован в А-исполнителе. Параметрическая адаптация данного алгоритма к обучающей выборке осуществляется А-экспертами методом кластеризации.

Согласно подзадачам выделяются две группы агентов: агенты, занимающиеся адаптивным формированием структур НС (А-постановщик, А-диспетчер, А-исполнитель) и агенты, адаптирующие параметры алгоритма обучения (А-эксперты).

На рис. 1 (в нотации UML [19, 20]) представлена общая схема взаимодействия агентов, реализующих вычислительный процесс.

Агент А-постановщик задач (aNNarchitector) является связующим зве-

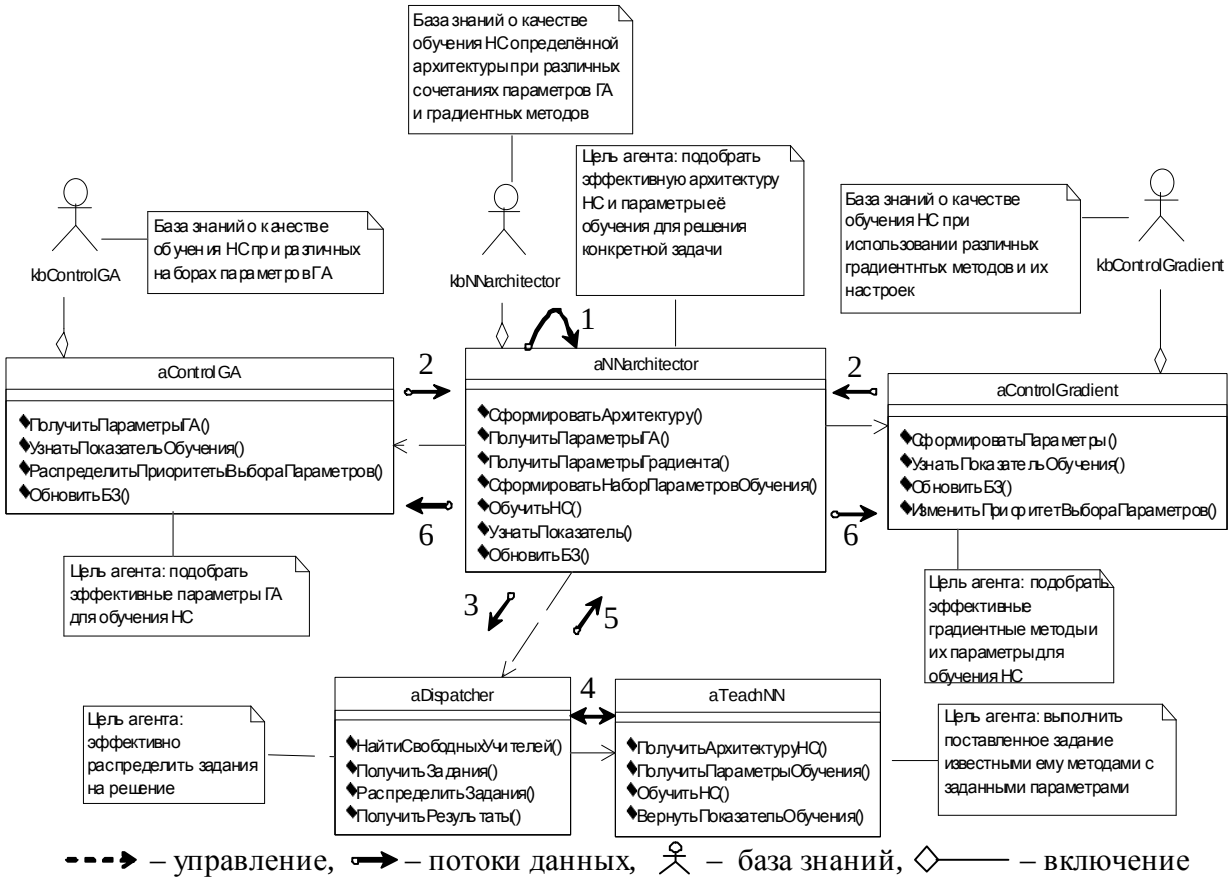

Рис. 1. Общая модель взаимодействия агентов

ном между агентами адаптации структуры НС и агентами адаптации параметров алгоритма обучения. Именно А-постановщик задач формирует пакеты заданий, состоящие из структур НС и параметров алгоритма обучения.

Таким образом, взаимодействие всех представителей агентной среды происходит в таком порядке:

– постановщик задач формирует набор структур НС;

– запрашивает у экспертов параметры алгоритма обучения для каждого набора;

– передаёт пакеты заданий диспетчеру;

– диспетчер раздаёт пакеты заданий исполнителям и получает оценки структур НС;

– диспетчер возвращает оценки всей популяции агенту постановщику задач;

– постановщик задач оповещает экспертов о результатах обучения;

– если в течении некоторого периода (10-15 итераций) структура НС не упрощается или исчерпан выделенный лимит итераций, то расчет закончен, иначе переход вначало.

3. Особенности разработки среды распределенных вычислений и агентных исполнителей вычислений

Качество разработанного программного обеспечения и необходимые для разработки ресурсы, в том числе и временные, в значительной мере зависят от адекватности выбранного инструментария разработки, его возможностей. В нашем случае инструментарию разработки уделено достаточно серьёзное внимание.

Среда распределённых вычислений реализована в MS Visual Studio 2005 на языке C#. Таким образом, для её работы необходимо на каждом компьютере, входящем в СРВ установить платформу Microsoft.Net версии не ниже 2.0.

Существует ряд программных средств разработки интеллектуальных программных агентов [16, 17]. Некоторые из них специализированы для проектирования мультиагентных систем. Основные характеристики средств разработки: базовый язык программирования, платформа реализации, соответствие стандартам.

В данной работе использовалась среда разработки JADE (Java Agent DEvelopment Framework) [21, 22]. Эта среда состоит из набора базовых классов Java, на основании которых разрабатываются специализированные агенты, а также классов, реализующих платформу для функционирования созданных агентов. Выбор Java в качестве языка программирования объясняется кросс-платформенностью его виртуальной машины, которая переносится и на агентов, созданных на этом языке программирования. Среда JADE удовлетворяет спецификациям FIPA97 (Foundation for Intelligent Physical Agents), используемый в работе язык взаимодействия агентов – FIPA ACL.

Платформа JADE на этапе функционирования состоит из двух агентов: AMS (Agent Management System) – агент, создающий среду функционирования агентов и предоставляющий агентам разнообразные сервисы; DF (Directory Facilitator) – агент, который ответствен за регистрацию агентов в платформе и предоставление информации об их местонахождении.

Для взаимодействия агентов в пределах одной платформы использовался протокол RMI (Remote Method Invocation).

Функционирование агентов JADE задаётся с помощью наследников класса Behavior (поведение), объекты которых агент декларирует в качестве своего поведения. Наследники класса Behavior позволяют реализовать одноразовые, многократно и периодически повторяющиеся действия. Существует возможность создать сложное поведение, состоящее из нескольких простых действий, выполняемых последовательно или параллельно. Наиболее сложным с точки зрения взаимодействия является поведение А-диспетчера. Оно задаётся тремя достаточно сложными поведениями для взаимодействия с А-исполнителями (опрос состояния, раздача заданий, получение результатов) и одним более простым поведением взаимодействия с А-постановщиком задач.

Взаимодействие агентов осуществляется с помощью сообщений на языке

ACL (Agent Communication Language) [23]. Язык ACL предназначен для совершения коммуникативных актов между агентами. Сообщения на этом языке представляют собой скомпонованный набор пар: название параметра – значение. Параметры, которые могут входить в состав ACL-сообщения, стандартизованы. Некоторые из используемых параметров представлены в табл. 2.

Таблица 2. Параметры ACL-сообщений

Параметр	Категория параметра
performative	Тип коммуникативного акта
sender	Отправитель сообщения
receiver	Получатель сообщения
reply-to	Получатель ответа на сообщение
content	Содержимое сообщения
language	Язык сообщения
encoding	Кодировка сообщения
ontology	Онтология сообщения

Перформатив (тип коммуникативного акта) может принимать следующие значения: INFORM (передаётся информация), REQUEST (запрос выполнения действия), QUERY_REF (запрос на получение информации), QUERY_IF (запрос на получение информации по условию).

В общем случае считается, что сообщение содержит текст на одном из языков передачи знаний (FIPA Semantic Language, Prolog и др.), но в рамках решаемой задачи в этом не было необходимости, поэтому параметры «Language» и «Ontology» использовались в качестве уточняющих параметров сообщения. Например, сообщение, которым A-постановщик задач информирует A-диспетчера о размере популяции, выглядит таким образом:

```
(inform
  :sender aNNarchitector
  :receiver aDispatcher
  :content "50"
  :language Fill
  :ontology Count),
```

где значение языка Fill говорит о том, что сейчас речь идёт о заполнении A-диспетчера пакетами заданий, а онтология Count уточняет, что содержание будет передавать количество пакетов.

В качестве IDE (Integrated Development Environment) использовалась среда разработки Eclipse. Данная среда обладает мощнейшим инструментарием, позволяющим значительно ускорить разработку. Особо следует отметить многопоточный отладчик, позволивший одновременно запускать несколько агентов и отлаживать их взаимодействие в одном окне.

Применение выбранного инструментария разработки позволило сократить сроки разработки, упростить проведение численных экспериментов.

4. Оценка эффективности организации вычислений

Организацией параллельных вычислений, в общем случае, занимается диспетчер или планировщик (scheduler) процессов, который реализует некоторый алгоритм диспетчеризации (дисциплину обслуживания [24]).

Основным назначением алгоритма диспетчеризации является повышение общей временной эффективности совместно выполняемых процессов. Таким образом, функциональная эффективность [25] алгоритма диспетчеризации, выражается временной эффективностью совокупно выполняемых параллельных процессов (диспетчер работает настолько хорошо насколько ускоряется выполнение диспетчеризации процессов).

Особенности программ, подлежащих диспетчеризации, проявляются в том, что:

– задача разбивается на несколько частей, которые могут быть выполнены параллельно;

– время выполнения отдельных частей может существенно различаться;

– общее время решения задачи определяется по окончанию выполнения последней части.

Это отличает их от задач, диспетчируемых в операционной системе и вычислений на кластерах.

На временную эффективность разработанной системы существенное влияние оказали: распараллеливание вычислений и выбор алгоритма диспетчеризации (п. 1.3).

Эти решения являются в некоторой степени универсальными. Они применимы для разработки различных программных средств, требующих значительных вычислительных ресурсов и имеющих способность к распараллеливанию вычислений неравномерными частями.

Так как данная работа предназначена для отражения аспекта организации вычислений в задаче адаптивного синтеза и обучения НС выполним оценку эффективности алгоритмов организации вычислений согласно методикам [18, 25].

Оценим эффективность распараллеливания вычислений.

Пусть, вычисления при решении некоторой задачи можно разбить на n независимых, одинаковых по количеству и последовательности вычислений, частей. Для вычислений может быть использовано m одинаковых вычислительных устройств (ЭВМ). Время вычислений для каждой части постоянно и равно t_i^* для некоторой i-й реализации.

Тогда время вычислений i-й реализации на одной ЭВМ составит $T_i(\Re_1) = n \cdot t_i^*$, на m ЭВМ – $T_i(\Re_m) = v(n,m) \cdot t_i^*$,

где $v(n,m) = \left[\frac{n}{m}\right] + \begin{cases} 0, \text{ при } \left\{\frac{n}{m}\right\} = 0 \\ 1, \text{ в противном случае} \end{cases}$,

и $[\ ], \{\ \}$ – целая и дробная часть целочисленного деления.

Степень превосходства алгоритма, с параллельным выполнением над таким же алгоритмом с последовательным выполнением составляет [18]:

$$S_1(\Re_m, \Re_1) = \frac{1}{N}\sum_{i=1}^{N} \frac{T_i(\Re_1) - T_i(\Re_m)}{\max(T_i(\Re_1), T_i(\Re_m)} \cdot 100\% =$$

$$= \frac{1}{N}\sum_{i=1}^{N} \frac{n \cdot t_i^* - v(n,m) \cdot t_i^*}{n \cdot t_i^*} \cdot 100\% = \quad (1)$$

$$= \left(1 - \frac{v(n,m)}{n}\right) \cdot 100\% \ .$$

В формуле (1) время на диспетчеризацию параллельных процессов и обмен данными не учитывается, так как в нашем случае объемы обмениваемых данных небольшие и диспетчеризация простейшая.

При разбиении задачи на 100 частей и выполнении на 30 ЭВМ, например, степень превосходства составит 96 %, при 1000 … 10000 и 7 соответственно – 85 … 86 %.

Пусть вычисления выполняются на различных по производительности ЭВМ. Время вычислений на наиболее производительном компьютере t_i^*, наименее производительном – t_i^{**} для некоторой i-й реализации. Определим максимальную и минимальную степень превосходства алгоритмов выполняемых параллельно по отношению к последовательному.

Время вычислений i-й реализации на наиболее производительной из имеющихся ЭВМ составит $\overline{T}_i(\Re_1) = n \cdot t_i^*$, худшее время на m ЭВМ – $\underline{T}_i(\Re_m) = v(n,m) \cdot t_i^{**}$. Максимальная степень превосходства:

$$\overline{S}_2(\Re_m, \Re_1) = \frac{1}{N}\sum_{i=1}^{N} \frac{\overline{T}_i(\Re_1) - \underline{T}_i(\Re_m)}{\max(\overline{T}_i(\Re_1), \underline{T}_i(\Re_m)} \cdot 100\% =$$

$$= \frac{1}{N}\sum_{i=1}^{N} \frac{n \cdot t_i^* - v(n,m) \cdot t_i^{**}}{n \cdot t_i^*} \cdot 100\% = \quad (2)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \left(1 - \frac{v(n,m) \cdot t_i^{**}}{n \cdot t_i^*}\right) \cdot 100\% .$$

Аналогично минимальная степень превосходства –

$$\underline{S}_2(\Re_m,\Re_1)=\frac{1}{N}\sum_{i=1}^{N}\left(1-\frac{v(n,m)\cdot t_i^{*}}{n\cdot t_i^{**}}\right)\cdot 100\%\ . \quad (3)$$

В случае, когда части, на которые разбиваются вычисления, различны и они выполняются на различных по производительности ЭВМ, степень превосходства согласно (1) будет:

$$S_3(\Re_m,\Re_1)=$$
$$=\frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{n}\tau_{ijk}-\max\limits_{k=1..m}\left(\sum\limits_{j=f(n,m,k-1)+1}^{f(n,m,k)}\tau_{ijk}\right)}{\sum_{j=1}^{n}\tau_{ijk}}\cdot 100\%, \quad (4)$$

где τ_{ijk} – время выполнения j – части вычислений на k-м компьютере в i-й реализации алгоритма,

$$f(n,m,k)=\left[\frac{n}{m}\right]\cdot k+$$
$$+\begin{cases}k-(m-\left\{\frac{n}{m}\right\}), \text{ если } k>m-\left\{\frac{n}{m}\right\}\cdot\\ 0, \text{ в противном случае}\end{cases}$$

Для оценки эффективности диспетчеризации следует сравнивать время выполнения с диспетчеризацией и без неё. В последнем случае распределения вычислений по компьютерам выполняется только по предварительному планированию. При этом, не имея априорной информации о времени выполнения каждой части на каждом компьютере, планирование может быть сведено только к примерно одинаковому (±1) по количеству частей распределению. Этот факт, а также алгоритмическое распределение вычислений с диспетчеризацией позволяют определять показатели степени превосходства (3) и (4) только экспериментальным путем.

$$S_4(D,P)=\frac{1}{N}\sum_{i=1}^{N}\frac{T_i(P)-T_i(D)}{\max(T_i(P),T_i(D)}\cdot 100\%, \quad (5)$$

где D – вычислительные алгоритмы под управлением алгоритма диспетчеризации, P – те же алгоритмы, выполняемые согласно предварительному планированию.

Для определения (5) не обязательно выполнять дополнительные численные вычисления с предварительным планированием. Достаточно выполнять протоколирование процесса вычислений при расчетах с диспетчером, при котором отмечается время начала и окончания выполнения расчетов каждой части, с указанием исполнителя.

Выполнены численные эксперименты по адаптивному синтезу и обучению НС на экспериментальной базе, которая с технической стороны состояла из компьютеров различной конфигурации.

Компьютеры экспериментальной базы различались объемом кэш-памяти, частотой процессора и шины, другими техническими характеристиками. Некоторые из них приведены далее в таком порядке: процессор / чипсет системной платы – кэш L1 кода / L1 данных / L2, Кб - тактовая частота/частота системной шины, МГц:

– Intel Pentium 4 / DFI PS35–BC/BL - 12/8/512 – 2400 (12 x 200) / 200;

– DualCore Intel Pentium E2180 / Gigabyte GA-945GM-S2 v3 – 32/32/1024 – 2000 (10 x 200) / 200;

– Intel Pentium 4 / DFI NB73-BC/BL/8/512 – 2400 (18 x 133) / 133;

– DualCore Intel Core 2 Duo E4500 / Asus P5G-MX- 32/32/2048 – 2200 (11 x 200) / 200 (расчет выполнялся на двух ядрах);

– DualCore Intel Core 2 Duo/ Gigabyte GA–G31MX-S2 - 12/8/512 – 2400 (12 x 200) / 200 ;

– Intel Pentium 4 / Transcend TS-ABR4 – 12/8/256 – 1700 (17 x 100) / 800 / 100.

На основании экспериментально полученной выборки случайного времени выполнения одной части вычислений для каждой ЭВМ построили интегральную функцию распределения (пример на рис. 2). Для каждого экспериментально полученного τ_{ijk} определена вероятность $P(\xi_k<\tau_{ijk})=n_{ijk}/N_k$, где ξ_k – случайное время выполнения одной части на k-й ЭВМ, n_{ijk} – количество вычислений из обучающей выборки для k-й ЭВМ, с временем выполнения не превосходящим τ_{ijk}, N_k – объем обучающей выборки для k-й ЭВМ. Функция интегрального распределения построена как линейная интерпо-

ляция между определенными таким образом точками.

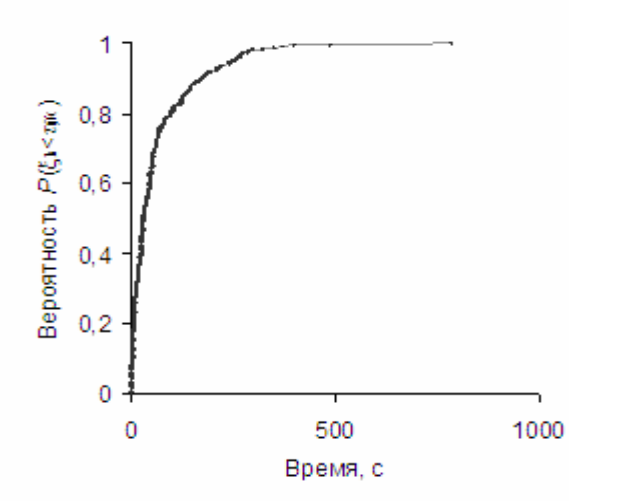

Рис. 2. Интегральная функция распределения времени обучения НС

Генератор случайного времени выполнения одной части вычислений $\bar{\tau}_{ijk}$ ставил в соответствие равномерно распределенной в интервале [0,1] случайной величине, полученной стандартными средствами, соответствующее функции интегрального распределения значение.

Время решения задачи при планировании моделировалось как $\max_{k=1..m}\left(\sum_{j=f(n,m,k-1)+1}^{f(n,m,k)}\bar{\tau}_{ijk}\right)$. Алгоритм диспетчирования моделировался средствами MS Excel и определялось общее время решения задачи. Согласно (5) определяем степень превосходства при различном количестве частей. Для каждого N_k решения задачи адаптивного синтеза и обучения НС моделировалось 15 раз.

На рис. 3 показаны результаты для компьютеров экспериментальной базы, на которых проводилась адаптация и обучение НС. Там же показаны доверительные интервалы определенные согласно методики приведенной в [18] и соответствующие регрессионные квадратичные кривые согласно критерия наименьшего среднеквадратичного отклонения.

Пессимистическая оценка (в области между средним значением и нижней границей доверительного интервала) говорит о 20 … 30% повышения временной эффективности за счет описанной в п. 3.2 диспетчеризации.

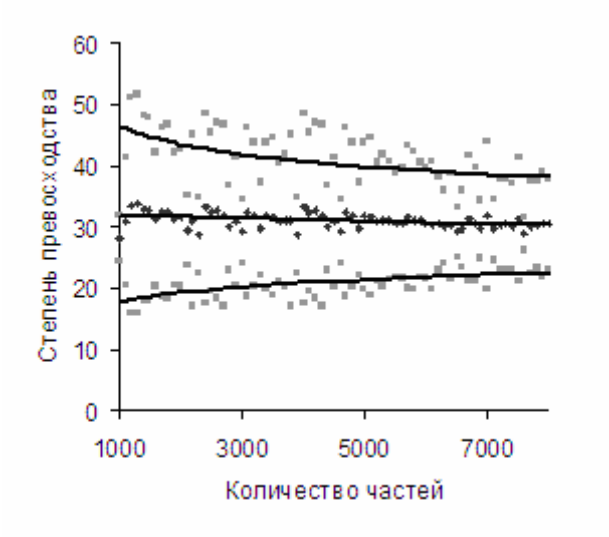

Рис. 3. Степень превосходства алгоритма диспетчеризации над алгоритмом простого планирования

Заключение

Применение агентного подхода позволило повысить эффективность вычислительного процесса путем использования невостребованных ресурсов, имеющихся в наличии компьютеров корпоративной сети.

Получен положительный опыт при проведении вычислений во время проведения занятий в учебных аудиториях. Вычисления проводились под ОС Windows XP в контексте недоступного студентам пользователя. При проведении занятий, не требующих значительных вычислительных мощностей, повышение времени отклика осталось практически незамеченным и не мешало проведению занятий.

Использование в вычислительном процессе современных рабочих станций с многоядерными процессорами, только повышает их КПД, не влияя на другую работу, выполняемую на них.

Большинство локальных сетей гетерогенно, в силу того, что формировались постепенно. Неисправные и морально устаревшие компьютеры постепенно заменялись новыми. В силу этого достоинством предлагаемого подхода и разработанного ПО является его перено-

симость, как по отношению к различной конфигурации, так и к различной архитектуре компьютеров.

Применение агентов позволило максимально упростить архитектуру разрабатываемого ПО, достаточно просто и эффективно организовать вычислительный процесс.

Агентный подход нашел естественное применение и при решении задачи адаптивного формирования и обучения НС. Интеллектуальные агенты позволили повысить эффективность направленного случайного поиска параметров НС.

Сравнивая степень превосходства алгоритма быстрой сортировки над алгоритмами с временной сложностью $O(n^2)$ в исследуемой области [18], которая составила 98 … 99,9 % и степень превосходства параллельно выполняемых алгоритмов в лучшем случае (при одинаковых частях) над последовательно выполняемыми, что для 30 ЭВМ (процессоров) составляет 96 %, очередной раз убеждаемся в приоритетной необходимости хорошей алгоритмизации. Что, в свою очередь, предполагает наличие соответствующих критериев, показателей и оценок эффективности, чему в некоторой степени и посвящена данная работа.

1. *Паклин Н.Б., Сенилов М.А., Тененев В.А.* Интеллектуальные модели на основе гибридного генетического алгоритма с градиентным обучением лидера // Искусственный интеллект. – 2004. – № 4. – С. 159 –168.
2. *Осовский С.* Нейронные сети для обработки информации. – М.: Финансы и статистика, 2002. – 343 с.
3. *Шинкаренко В.И., Олейник Д.В.* Нейросетевое моделирование рабочей нагрузки СУБД // Искусственный интеллект. – 2007. – № 4. – С. 657–664.
4. *Олейник Д.В., Шинкаренко В.И.* Мультиагентная адаптация гибридного генетического алгоритма для обучения нейросетей // Искусственный интеллект. – 2008. – № 4. – С. 463–470.
5. *Лацис А.О.* Как настроить и использовать суперкомпьютер. – М.: Бестселлер, 2003. – 274 с.
6. *Воеводин Вл.В., Жуматий С.А.* Вычислительное дело и кластерные системы. – М.: Изд–во МГУ, 2007. – 150 с.
7. *Залещанский Б.Д., Чернихов Д.Я.* Кластерная технология и живучесть глобальных автоматизированных систем. – М.: Финансы и статистика, 2005. – 384 с.
8. *Gregory F. Pfister* In search of clusters , 2nd Edition. – Prentice Hall, 1998. – 578 с.
9. *Дорошенко А.Е., Рухлис К.А., Мохница А.С.* Распределённая платформа для управления ресурсами гетерогенного кластера // Проблемы программирования. – 2008. – № 2-3. – С. 150–156.
10. *Перевозчикова О.Л., Тульчинский В.Г., Ющенко Р.А.* Построение и оптимизация параллельных компьютеров для обработки больших объёмов данных // Кибернетика и системный анализ. – 2006. – № 4. – С. 117–129.
11. *Гофф Макс К.* Сетевые распределенные вычисления. Достижения и проблемы.– М.: Кудиц-образ, 2005. – 320 с.
12. *Топорков В.В.* Модели распределённых вычислений. – М.: ФИЗМАТЛИТ, 2004. – 320 с.
13. *Дорошенко А.Е., Розенблат А.П., Рухлис К.А., Тырчак Ю.М.* Проблемы и средства программирования Грид–систем // Проблемы программирования. – 2007. – № 3. – С. 16–31.
14. http://ru.wikipedia.org/wiki/Распределенные_вычисления.
15. http://ru.wikipedia.org/wiki/SETI@Home
16. *Плескач В.Л., Рогушина Ю.В.* Агентні технології: Монографія. – К.: – Нац. торг.-екон. ун-т, 2005. – 338 с.
17. *Атанасова Т.* Агентная технология: концепции, модели, приложения. – Варна, 2000. – 155 с.
18. *Шинкаренко В.И.* Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов // Проблемы программирования. – 2001 – № 3-4. – С. 31–39.
19. *Стефан Бергстрём, Лотта Роберг.* Rational Unified Process – путь к успеху. Руководство по внедрению RUP. – М.: Кудиц - образ, 2004. – 256 с.

20. *Мартин Фаулер.* UML. Основы. – М.: Символ, 2005. – 192 с.
21. http://jade.tilab.com
22. *Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood.* Developing Multi–Agent Systems with JADE. – Wiley, 2007. – 300 с.
23. http://www.fipa.org/specs/fipa00061/SC00061G.html
24. Словарь по кибернетике / Под ред. В.С. Михалевича. – Киев.: Гл. ред. УСЭ им. М.П. Бажана, 1989. – 751 с.
25. *Шинкаренко В.И.* Функциональная эффективность нечетко специфицированных алгоритмов // Проблемы программирования. – 2006 – № 1. – С. 31–39.

Получено 10.09.2008

Об авторах:

Шинкаренко Виктор Иванович,
кандидат технических наук,
доцент кафедры “Компьютерных информационных технологий”,

Олейник Денис Викторович,
ассистент,
кафедра “Компьютерных информационных технологий”.

Место работы авторов:

Днепропетровский национальный университет железнодорожного транспорта,
рабочий адресс: 49010 каф. «КИТ»,
ДНУЖТ, ул. ак. Лазаряна 2,
Днепропетровск,
Тел.: 8 (056) 373 1535

Днепропетровский национальный университет железнодорожного транспорта,
49010 каф. «КИТ», ДНУЖТ, ул. ак. Лазаряна 2, г. Днепропетровск,