ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА СОЗДАНИЯ ВЫЧИСЛИТЕЛЬНОЙ ИНФРАСТРУКТУРЫ В СЕТИ ИНТЕРНЕТ*

С. Н. Васильев, Г. А. Опарин, А. Г. Феоктистов, И. А. Сидоров
Институт динамики систем и теории управления СО РАН, Иркутск, Россия
e-mail: snv@icc.ru, oparin@icc.ru

In this work we consider a methodology of construction, architecture and the basic features of the distributed computing environment. The distributed computing environment is created on the basis of a modular programming system. An example of the application of this environment for solving a problem of imitational modeling is demonstrated.

Введение

Одной из концепций поддержки высокопроизводительных распределенных вычислений научно-технического характера является технология grid. На сегодняшний день эта технология обеспечивает возможность удаленного доступа к узлам вычислительной сети и позволяет определить вычислительные возможности конкретного узла (количество процессоров, объем оперативной памяти и т. п.) и степень его работоспособности, выполнить на этом узле некоторое независимое задание. Между тем при проведении фундаментальных и прикладных исследований на основе метода математического и компьютерного моделирования необходимы высокоуровневые инструментальные средства, которые позволят автоматически определять возможные варианты и способы взаимодействия независимых узлов вычислительной сети, а также порядок (в общем случае построить параллельный план) использования вычислительных ресурсов этих узлов, необходимый для достижения цели исследования.

Перспективным подходом к построению инструментальных сред такого рода является создание проблемно-ориентированных высокоуровневых языков и систем распределенного модульного программирования [1, 2], которые позволяют накапливать в памяти центральной (управляющей) ЭВМ знания о вычислительных ресурсах предметной области (ПО) и использовать эти знания при автоматическом решении задач заданного класса. Под вычислительным ресурсом понимается интегральная характеристика узла, включающая спецификацию возможностей как вычислительного оборудования, так и установленного на нем

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 04-07-90358).

прикладного программного обеспечения (вычислительных модулей). Инструментальные средства на основе базы знаний открывают возможности в проблемно-ориентированных терминах как конструировать распределенные параллельные вычислительные процессы, так и автоматически их синтезировать на основе непроцедурных постановок задач типа "дано — требуется". Такие средства предназначены для пользователей, желающих эффективно использовать вычислительную сеть, не вдаваясь в подробности распределенного параллельного программирования.

В данной работе представлены интеллектные инструментальные средства для организации распределенной вычислительной среды (РВС) в сети Интернет. В частности, рассматриваются: протокол сетевого взаимодействия вычислительных узлов, основанный на технологии "клиент-сервер"; проблемно-ориентированный язык описания предметной области в виде расширения языка XML; программные средства удаленного запуска вычислительных модулей; способы и алгоритмы обработки параллельных списков данных; способы и алгоритмы планирования загрузки вычислительных ресурсов.

1. Средства реализации

На наш взгляд, можно выделить следующие основные критерии оценки программных средств реализации РВС: возможности графического инструментария, многоплатформенность (переносимость на уровне исходного кода), эффективность использования памяти, наличие удобной и мощной среды для разработки приложений и развитые сетевые средства. Исходя из перечисленных выше критериев проведен сравнительный анализ инструментальных сред, таких как QT (Q Toolkit), Java Toolkit, GTK (Gimp ToolKit) и MFC (Microsoft Foundation Class). В качестве программного средства реализации РВС выбрана инструментальная среда QT [3], включающая библиотеку классов C++ и набор инструментальных программных средств, предназначенных для построения многоплатформенных приложений с графическим интерфейсом. QT представляет собой единую платформу для приложений, которые могут работать под управлением операционных систем Windows 95/98/Me/2000/XP, Mac OS X, Linux, Solaris, HP-UX и других версий Unix. В качестве языка спецификации ПО решено использовать расширение языка разметки XML.

2. Планирование распределенных вычислений

Обобщенная схема базы знаний РВС может быть представлена в виде системы $KB = \langle P, T, F, M, Y, IP, \text{in}, \text{out}, IF, R\rangle$, где P — множество имен величин (параметров) предметной области; T — множество допустимых типов параметров; F — множество имен абстрактных операций, выполняемых над этими величинами; M — множество имен модулей, реализующих абстрактные операции из F; Y — множество имен (адресов) узлов вычислительной сети, на которых установлен тот или иной модуль из M. Содержательно операция $f_i \in F$ базы знаний KB предполагает возможность вычисления параметров $\text{out}(f_i)$ по параметрам $\text{in}(f_i)$ с помощью некоторого модуля m из библиотеки модулей M, установленного в узле $y \in Y$. Связи между элементами множеств P, T, F, M и Y заданы в базе знаний отношениями $IP \subset P \times T$, $\text{in}, \text{out} \subset P \times F$, $IF \subset F \times M$, $R \subset M \times Y$ (в общем случае типа многие-ко-многим).

Множество T допустимых типов параметров наряду с традиционными типами языков программирования включает специальный тип par — "параллельный список". Интерпрета-

ция операции $f : x \to y$, где x, y — параметры типа par, выполняется следующим образом: 1) в базе знаний KB осуществляется поиск всех вычислительных ресурсов (модуль+узел), реализующих операцию f; 2) организуется параллельное применение i-го параллельного ресурса к i-му элементу списка x; 3) результат применения присваивается i-му элементу списка y.

Структуру $H = (A_0; B_0)$, где $A_0, B_0 \subset Z$, далее будем называть постановкой задачи для модели M или просто задачей, а множества A_0, B_0 соответственно входом и выходом задачи. Модули f_1 и f_2 из множества F фактически моделируют условия постановки задачи планирования $H = (A_0; B_0)$, а именно модель распределенного вычислителя M содержит модули $f_1(; A_0)$ и $f_2(B_0;)$, где $A_0, B_0 \subset Z$. Отсутствие атрибута до или после точки с запятой означает пустоту соответствующего множества. Модуль f_1 считается модулем начальных данных, а модуль f_2 — целевым модулем.

Предполагается, что модель M распределенного вычислителя является в общем случае избыточной в том смысле, что для решения задачи H используется только часть модулей из F и/или задача T имеет несколько альтернативных планов решения.

Задача планировщика состоит в том, чтобы, имея описание модели KB, определить, какие модули из множества F и в какой последовательности должны быть исполнены для вычисления требуемого множества целевых параметров B_0 по заданному множеству параметров — исходных данных A_0. Цель — получить параллельный план решения задачи $H = (A_0, B_0)$, представляющий собой последовательность ярусов, каждый из которых включает определенное число модулей из F. Модули каждого яруса могут выполняться параллельно. Параллельные вычисления осуществляются по схеме FORK/JOIN, т. е. модуль $(i + 1)$-го яруса может быть выполнен, если завершилось выполнение всех модулей i-го яруса.

3. Язык спецификации плана решения задачи

Для построения параллельного плана решения задачи используется проблемно-ориентированное расширение языка разметки XML. Файл спецификации плана решения задачи plan.xml состоит из двух основных секций: описания всех параметров, участвующих в процессе вычислений (<parameters>), и описания процесса вычислений (<process>). Ниже приведены фрагменты файла plan.xml:

Фрагмент 1. Описание параметров:

```
<parameters>
   <param>
      <id>1</id>
      <name>cm</name>
      <type>file</type>
      <filename>cm.txt</filename>
      <comment>Computing model</comment>
   </param>
   <param>
      <id>2</id>
      <name>vdgm</name>
      <type>filelist</type>
      <filepattern>vdgm%d.txt</filepattern>
```

```
      <comment>Variants of data of GPSS model</comment>
   </param>
</parameters>
```

В данном примере рассмотрены два основных типа параметров: файл и параллельный список. Каждый из параметров должен обладать уникальным идентификатором, именем, типом и комментарием. Остальные свойства зависят от конкретного типа параметра. Для типа "файл" необходимо явно указывать имя файла, которое будет передано модулю при запуске. Для параллельного списка указывается шаблон имени файла, в данном примере "vdgm%d.txt", где %d будет заменено на числовое значение порядкового номера элемента в списке.

Фрагмент 2. Описание процесса вычислений, включающее инициализацию значений входных параметров и исполнение модулей:

```
<?xml version="1.0" encoding="UTF-8"?>
   <plan>
      <process>
         <!-- init input parameters -->
         <stage type='init'>
            <parameters action='load'>
               <param>
                  <id>1</id>
                  <file>in/cm.txt</file>
               </param>
               <param>
                  <id>5</id>
                  <file>in/gm.gps</file>
               </param>
            </parameters>
         </stage>
         ...
         <stage type='modules'>
            <module id='1'/>
            <module id='3'/>
            <module id='4'/>
         </stage>
         ...
      </process>
```

В данном примере инициализируются значения двух входных параметров с идентификаторами 1 и 5, значения которых считываются из файлов in/cm.txt и in/gm.gps соответственно. Запуск модулей 1, 3 и 4 будет осуществлен по мере освобождения вычислительных ресурсов.

4. Архитектура распределенной вычислительной среды

Основными составляющими архитектуры РВС являются визуальный пользовательский интерфейс, управляющий компьютер РВС и вычислительные узлы.

Пользовательский web-интерфейс. Решение задачи многоуровневой обработки данных с помощью распределенных и многократно используемых специализированных прикладных программ предполагает создание специального web-интерфейса к распределенным программным средствам. При этом пользователь задает свои входные данные (заполняет определенные формы), не заботясь о том, где и как реально осуществится обработка данных. Примерами таких систем являются UNICORE, PowerBuilder, Javelin и др. Проведенный анализ архитектуры клиентского интерфейса для распределенных систем показал, что наиболее оптимальным вариантом при реализации пользовательского интерфейса является связка web-сервер+шлюз+DataBase. Наиболее оптимальный вариант для реализации данной архитектуры — Apache+XML/XSLT+Perl+MySQL. Связка XML/XSLT позволяет отказаться от жесткой привязки к дизайну интерфейса и манипулировать более гибкими структурами данных.

Управляющий компьютер РВС выполняет координирующую роль. Условно в нем можно выделить три основных компонента:

— менеджер вычислительных ресурсов, владеющий полной информацией о всех ресурсах РВС и выполняющий функции подключения, отключения и аутентификации узлов. В составе менеджера ресурсов реализованы основные функции протокола передачи данных для взаимодействия сервера с удаленными клиентами;

— менеджер вычислительных процессов. Вычислительный процесс (ВП) — это обособленная единица, представляющая экземпляр реализации плана решения задачи в РВС. Основной функцией данного компонента является поддержка многозадачности, т. е. одновременное выполнение нескольких ВП, управление пространством глобальных параметров, инициализация входных параметров ВП и возврат результатов счета для выходных параметров ВП;

— менеджер очереди задач ОС, осуществляющий функции планирования по выделению ресурсов тем или иным вычислительным процессам.

Вычислительные узлы подключаются к РВС при помощи специальных программ-клиентов, позволяющих организовать запуск модулей, размещенных на удаленном узле. Клиентская программа получает значения входных параметров, организует выполнение модуля на удаленном узле и после завершения отправляет значения выходных параметров на сервер. Клиентское приложение разработано с соблюдением всех принципов многоплатформенности и может выполняться под управлением операционных систем Windows, Linux и Macintosh.

5. Централизованная очередь процессов

Как отмечено в [4], основное назначение очереди задач — обеспечение максимально интенсивного использования вычислительной системы путем поддержки высокой степени параллелизма функционирования ее отдельных элементов. В большинстве распределенных сред принципы, полагающиеся в основу очереди задач, заимствуются из операционных систем с той разницей, что в распределенных средах преимущественно используются алгоритмы пакетной обработки заданий.

Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, т. е. решение максимального числа задач в единицу времени. Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое

задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, т. е. множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие разные требования к ресурсам, так, чтобы обеспечивалась сбалансированная загрузка всех узлов вычислительной сети. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, иными словами, выбирается "выгодное" задание [5].

Выделение ресурсов тому или иному процессу осуществляется в результате планирования и диспетчеризации. Планируется выделение ресурсов на основе информации, хранящейся в описании вычислительных процессов и подзадач, которые требуют вычислительных ресурсов в момент опроса очереди задач. При планировании могут приниматься во внимание приоритеты задач, время их ожидания в очереди, накопленное время выполнения и другие факторы. К основным алгоритмам планирования загрузки ресурсов следует отнести следующие.

1. Алгоритм планирования, основанный на приоритетах. Данный алгоритм подразумевает наличие у каждого процесса дополнительной характеристики — приоритета привилегированности процесса по отношению к другим процессам.

2. Алгоритм планирования, основанный на информации о затребованных вычислительных ресурсах. При выделении ресурсов планирование осуществляется с учетом примерной оценки времени выполнения задачи на вычислительном узле с тактовой частотой процессора G.

3. Смешанные алгоритмы планирования. Планирование основывается на концепции как приоритетов задач, так и запрашиваемых вычислительных ресурсов.

Процесс планирования загрузки ресурсов в рассматриваемой РВС организован по принципу "одношагового" планирования, при котором опрос очереди ВП запускается в ответ на происходящие события, из которых наиболее важны освобождение ресурсов и появление новых заданий.

Циклическая процедура опроса очереди задач начинается с поиска свободных ресурсов. При наличии простаивающих узлов РВС сети менеджер опрашивает процессы на предмет наличия заданий для свободного узла. Данная процедура заключается в сравнении всех модулей узла с перечнем не выполненных модулей ВП на текущей стадии. После того как для свободного вычислительного ресурса выделяется задача, начинается процедура подготовки к запуску модуля на удаленном узле.

Ниже приведена циклическая процедура опроса очереди задач.

```
void GridServer::doProcess()
{
   logMessage("Trying to do something");
   //search not busy node
   for((int)node_num=0;node_num < (int)nodes.size();node_num++)
   {
      if ( !nodes[node_num]->busy )
      {  //if node not busy we give this node to all processes which have jobs
         //GridProcess check if this node can execute module  on current stage
         for ( int process_num=0; process_num<(int)processes.size(); process_num++ )
         {
            if ( processes[process_num]->haveJobForNode(nodes[node_num] ) )
            {
               int moduleId = processes[process_num]->giveJobToNode( nodes[node_num] );
```

```
                if ( moduleId )
                {
                   //this node have module which process need to execute
                   //TODO: in the gridProcess
                   nodes[node_num]->busy = true;
                   nodes[node_num]->process_id = processes[process_num]->id;
                   nodes[node_num]->prepareToStartModule( moduleId );
                };
             } else if ( processes[process_num]->completed )
             {
                logMessage( QString("Process(id:%1) is removed from processes list").
                                                    arg(processes[process_num]->id) );
                //TODO:remove this process from processes list
             };
          };
       };
    };
};
```

Настоящая реализация процесса планирования загрузки вычислительных узлов не учитывает их производительность и не включает отработанный механизм изменения приоритетов вычислительного процесса. Приоритеты задач назначаются при постановке в очередь либо в соответствии с порядком следования задач в очереди.

6. Запуск модуля на удаленном узле вычислительной сети

Запуск модулей, размещенных на удаленных узлах, представляется наиболее сложным для любой РВС. Обычно ситуация усложняется, когда помимо команды на запуск вычислительного модуля на удаленный узел необходимо передавать значения входных параметров и получать их в виде файлов.

При подключении нового узла к распределенной сети клиентская программа отправляет информацию о списке модулей, которые способен запускать подключаемый узел. Данная информация запоминается и используется при опросе очереди ВП, если удаленный узел успешно прошел авторизацию и сервер включил его в РВС. Допустим, что процесс p_1 определяет, что узел n_1 способен запустить модуль m_1, тогда сервер переходит в стадию подготовки к процедуре выполнения модуля на удаленном узле. В этом случае процесс инициализации, запуска и завершения выполнения модуля включает следующие шаги.

1. На сервере проверяется наличие значений всех входных параметров для запуска модуля m_1. Если данное условие соблюдается, то на удаленный узел отправляются значения входных параметров.

2. При корректном получении значений всех параметров удаленный узел передает на сервер информацию о том, что все параметры получены успешно и клиент готов к запуску модуля.

3. Сервер отправляет команду на запуск удаленного модуля.

4. На вычислительном узле в соответствии со спецификацией модуля значения всех входных параметров сохраняются на диск и производится попытка запуска модуля. В случае успеха клиент отправляет сообщение о том, что модуль успешно запущен.

5. Сервер меняет состояние для данного узла и ожидает завершения выполнения модуля.

6. После того как модуль завершил вычисления, клиент считывает значения выходных параметров и передает их на сервер.

7. На сервере значения выходных параметров копируются в глобальное пространство параметров.

8. На завершающем этапе выполняется очистка состояния вычислительного узла и он помечается свободным.

Если какой-либо из этапов выполнен некорректно, то после последнего шага процедура запуска модуля на удаленном узле прекращается.

7. Протокол передачи данных

При реализации передачи данных по сети необходимо применять низкоуровневые библиотеки для работы с протоколом TCP. В QT реализован класс QSocket, использующийся при разработке приложений серверов и клиентов, работающих по протоколу TCP. Выбор данного класса позволяет избежать проблем при переносе программы на операционные системы класса Windows.

Сообщения, передаваемые по сети, представляют собой XML-документы. Использование XML в качестве формата предаваемых данных существенно облегчает разработку и модернизацию протокола, так как основное свойство XML — это расширяемость. Таким образом, нет необходимости создавать жесткую структуру передаваемых сообщений, которую придется перерабатывать при необходимости добавления дополнительных полей в передаваемом сообщении.

Ниже приведен пример взаимодействия клиента с сервером.

Сервер отправляет запрос на передачу параметров модулю № 4:

```
<?xml version='1.0' encoding='UTF-8'?>
   <request id='120' name='parameter'>
      <module id='4'>
         <parameter id='1'><![CDATA[01010101010]]></parameter>
      </module>
   </request>
```

Клиент обрабатывает запрос и в случае успешного получения всех параметров отправляет серверу ответ:

```
<?xml version='1.0' encoding='UTF-8'?>
   <answer id='120' status='1' msg='success'>
      <module id='4'/>
   </answer>
```

В случае сбоя серверу возвращается сообщение об ошибке:

```
<?xml version='1.0' encoding='UTF-8'?>
   <answer id='120' status='2'msg='failed'>
      <module id='4'/>
   </answer>
```

Если на предыдущем этапе клиент вернул положительный ответ, сервер обновляет состояние для данного узла и посылает команду для запуска модуля:

```
<?xml version='1.0' encoding='UTF-8'?>
   <request id='130' msg='start'>
      <module id='4'/>
   </request>
```

Данный пример хорошо иллюстрирует структуру передаваемых сообщений. Каждый запрос и ответ снабжается универсальным идентификатором, коротким строковым сообщением для пояснений текущего запроса и телом сообщения.

Следует отметить, что реализация сетевого интерфейса поддерживает двухстороннюю инициацию передачи, т. е. клиент имеет возможность посылать запросы серверу и получать на них ответы. Это позволяет не производить постоянный опрос клиентов с целью получения результатов о выполнении модулей. В случае, если инициатором является сервер, клиенту нет необходимости периодически опрашивать сервер на предмет наличия новых заданий для него.

На уровне кода взаимодействие узлов в сети организуется с помощью двух основных функций — receiveDataFromFromRemoteNode() и sendDataToRemoteNode(). Функция получения данных является слотом (в терминах QT) и вызывается при генерации сигнала readyRead() класса QSocket. Полученное сообщение анализируется и в зависимости от идентификатора назначается функция, выполняющая дальнейшую обработку сообщения. Функция отправки сообщений на удаленную машину принимает два входных параметра — тело сообщения в формате xml, и если тип передаваемого сообщения — запрос, то в качестве второго параметра передается ссылка на функцию, которая будет осуществлять обработку ответа от удаленной машины.

8. Пример решения задачи

Рассмотрим задачу моделирования процесса погрузочно-разгрузочных работ (ПРР) для ООО “Иркутский хладокомбинат”.

Система моделирования реализована в виде распределенного пакета имитационного моделирования. Инструментальная среда и пакет имитационного моделирования установлены на “горизонтальном” вычислительном кластере Института динамики систем и теории управления СО РАН, который состоит из одного центрального (управляющего) и четырех вычислительных узлов. Конфигурация узла: процессор Pentium 4 2.8 ГГц, оперативная память 512 Мбайт, жесткий диск IDE 80 Гбайт, сетевая плата Gigabit Ethernet.

Модель зоны ПРР хладокомбината в общем виде может быть представлена в виде структуры $M = \langle Z, W, G, F, Cl, T, Co \rangle$, где Z — множество товарных зон; W — множество помещений для хранения товаров; G — множество товарных групп; F — множество товаров; Cl — множество клиентов; T — множество температурных режимов; Co — множество, включающее ограничения на пропускную систему склада, такие как K — количество каров, Lo — количество грузчиков, S — количество кладовщиков, L — количество лифтов и др.

Множества Z, W, G, F, Cl и T связаны между собой следующими отношениями (в общем случае типа многие-ко-многим): $R_1 \subset F \times Cl$ — разгружаемые товары; $R_2 \subset F \times Cl$ — отгружаемые товары; $R_3 \subset F \times Cl \times W$ — хранимые товары; $R_4 \subset F \times T$ — температурный

режим товаров; $R_5 \subset W \times T$ — температурный режим помещений для хранения товаров; $R_6 \subset F \times G$ — товары, составляющие товарные группы; $R_7 \subset W \times G$ — помещения, соответствующие товарным группам; $R_8 \subset W \times Z$ — размещение помещений для хранения товаров по товарным зонам.

Имитационная модель зоны ПРР автоматически генерируется на основе его описания, хранящегося в базе данных. Эту функцию выполняет модуль GenMod.exe. Этот же модуль генерирует множество файлов с различными вариантами количественных и вероятностно-временных характеристик процесса работы склада (параллельный список данных). Модуль GPSSW.exe (свободно распространяемая студенческая версия системы имитационного моделирования — Minuteman Software GPSS World for Windows) предназначен для прогона имитационной модели зоны ПРР с одним из вариантов исходных данных. Копии модуля GPSSW.exe размещены на вычислительных узлах кластера. Модуль CMchoice.exe предназначен для анализа результатов расчетов на всех вариантах исходных данных и выбора наиболее оптимального из них.

Решение задачи моделирования зоны ПРР (имитация работы склада хладокомбината в течение одного месяца) в РВС на различных наборах входных данных показало ускорение процесса решения задачи приблизительно в 3.3 раза по сравнению со временем решения аналогичной задачи на локальном персональном компьютере.

Заключение

Представленная в данной работе инструментальная среда разработана на основе принципов создания многоплатформенных приложений, обеспечивающих переносимость на уровне исходного кода и использование в качестве модулей приложений, разработанных под операционные системы Windows, Unix like systems, MAC OS X.

В плане дальнейшего ее развития планируется доработать отдельные подсистемы с целью улучшения заложенных в них алгоритмов и исправления возникших на данный момент проблем. Во-первых, очередь процессов планируется реализовать в соответствии с алгоритмом BackFill [6], в котором используется принцип обратного заполнения: вначале размещаются задания строго по приоритетам, а в образовавшиеся “дырки” помещаются менее приоритетные задания. Во-вторых, требуется решение проблем с отказоустойчивостью, так как на данный момент отключение вычислительных узлов от распределенной среды может приводить к нарушению выполнения вычислительного процесса. Также планируется доработать протокол передачи данных по сети и включить в инструментальную среду наиболее развитые протоколы передачи файлов: FTP, NFS, NetBIOS (Samba).

Список литературы

[1] Опарин Г.А., Феоктистов А.Г. Инструментальная распределенная вычислительная САТУРН-среда // Программные продукты и системы. 2002. № 2. С. 27–30.

[2] Васильев С.Н., Опарин Г.А., Феоктистов А.Г. Интеллектный подход к автоматизации моделирования сложных управляемых систем // Современные проблемы прикладной математики и механики: Тр. Междунар. конф. RDAMM-2001. Новосибирск: ИВТ СО РАН, 2001. Т. 6, ч. 2. С. 159–168.

[3] Бланшет Ж., Симмерфилд М. QT3: Программирование GUI на C++. М.: Кудиц-образ, 2005. 448 с.

[4] Лорин Г., Дейтел Х.М. Операционные системы. М.: Финансы и статистика, 1984. 392 с.

[5] Олифер В.Г., Олифер Н.А. Сетевые операционные системы. СПб.: Питер, 2003. 544 с.

[6] Feitelson D.G., Weil A.M. Utilization and predictability in scheduling the IBM SP2 with backfilling // Proc. of IPPS/SPDP. IEEE Computer Society. 1998. P. 542–546.

Поступила в редакцию 26 сентября 2006 г.