УДК 6.83.1

Ю.М. Руденко, Л.В. Мусина

УНИВЕРСАЛЬНАЯ СИСТЕМА РЕШЕНИЯ ПАРАЛЛЕЛЬНЫХ ЗАДАЧ НА ОДНОРОДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Рассмотрена универсальная система решения параллельных задач на однородных вычислительных системах. Непрерывное совершенствование техники в настоящее время обусловило появление многопроцессорных комплексов со сложной архитектурой межпроцессорных соединений – вычислительных систем.

Email: Rudenko-yuriy@inbox.ru, Liya@demx.ru

Ключевые слова: *вычислительная система, последовательный алгоритм, параллельный алгоритм, граф-схема, матрица следования, параллельное задание, кластер, планировщик.*

Структура системы. В вычислительных системах обеспечивается одновременное параллельное выполнение программных модулей, которые могут принадлежать одной или нескольким программам. Структурная схема такой системы представлена на рис. 1

Блок 1 на рис. 1 отвечает за построение последовательного алгоритма, представляемого по ГОСТ 19.003 – 80 ЕСПД и ГОСТ 19.701 – 90 ЕСПД. Следует отметить, что может быть использован любой последовательный язык: FORTRAN, C, C++, Pascal и др. В блоке 2 выполняется преобразование последовательного алгоритма в параллельный, подробно описанное в работе [1]. Сущность такого преобразования заключается в том, что последовательный алгоритм разбивается на участки. Границей раздела служат логические операторы. На каждом участке анализируются связи между блоками алгоритма по выходным параметрам. Если выходной параметр стоящего выше блока не используется стоящим ниже блоком, то эти блоки могут выполняться параллельно и т. д.

Блок 3 осуществляет преобразование параллельного алгоритма в его граф-схему [2]. Использование граф-схем дает возможность применять теорию графов для анализа особенностей решаемой задачи: каждый блок параллельного алгоритма представляется вершиной графа, а связи между блоками – дугами между соответствующими вершинами. Таким образом формируется ориентированный граф. Этот граф представляется в виде сверток и разверток граф-схемы [3], каждая из которых может быть логической функцией, если считать наличие информации на дуге за логическую единицу, а отсутствие – за логический нуль.

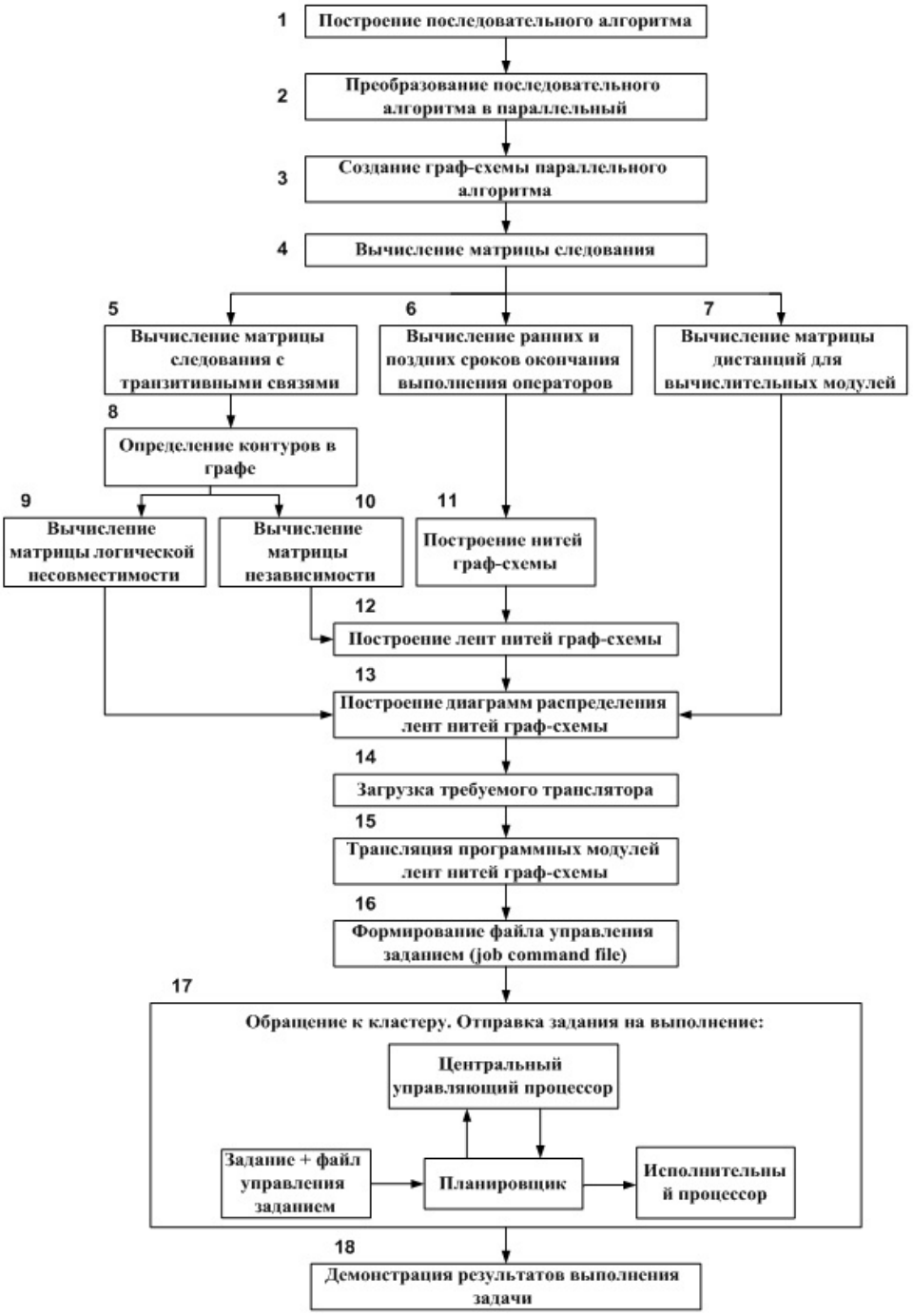

Рис. 1. Структурная схема универсальной системы решения параллельных задач на однородных вычислительных системах

Для представления современных алгоритмов в виде граф-схем с логическими функциями достаточно функций, И и Исключающее ИЛИ [4]. Введение остальных логических функций может значитель-

но расширить возможности представления решаемых задач с помощью граф-схем.

Для удобства исследования и преобразования графов вводится матрица следования – транспонированная матрица смежности из теории графов [5], которая вычисляется в блоке 4. Именно матрица следования является основой для последующего построения рассматриваемой системы. По графу без контуров может быть построена треугольная матрица следования с нулевыми элементами на главной диагонали.

Для анализа матрицы следования необходимо отразить в ней неявные связи между операторами, называемые транзитивными, которые реально существуют и определяются задающими связями, которые между ними в графе отсутствуют. Их введение необходимо для выявления ветвей связанных операторов, которые не могут выполняться параллельно, и построения матрицы следования с транзитивными связями [6], реализованного в блоке 5.

При исследовании граф-схем алгоритмов одними из их основных характеристик являются ранние и поздние сроки окончания выполнения операторов [7], используя которые, можно построить планы решения параллельной задачи на однородных вычислительных системах произвольной конфигурации. Эти величины определяются в блоке 6 структурной схемы системы. Диаграммы выполнения операторов для ранних и поздних сроков – наглядный способ представления многопроцессорной обработки.

Для удобства размещения операторов алгоритма на вычислительных модулях системы ее структура представляется в виде матрицы дистанций. Она определяет подходящие вычислительные модули, которые имеют минимальные расстояния со всеми остальными модулям, для размещения нитей решаемой задачи [7]. Построение матрицы дистанций реализовано в блоке 7.

При построении алгоритмов распараллеливания часто приходится использовать матрицы следования общего не треугольного вида. В этом случае просмотр строк матрицы производится неоднократно до установления факта ее неизменности. В результате указанных преобразований в блоке 8 устанавливается факт наличия контура в графе, о котором свидетельствуют ненулевые диагональные элементы. Весь последующий предлагаемый метод должен исключать наличие контуров. В работе [7] рассмотрен алгоритм определения контуров в граф-схеме.

Матрица следования с транзитивными связями позволяет определить матрицу логической несовместимости и матрицу независимости, за формирование которых отвечают блоки 9 и 10 соответственно. Алгоритм нахождения матрицы несовместимости, подробно описанный в статье [8], предусматривает формирование

множества логически несовместимых операторов – операторов, принадлежащих различным логическим ветвям и способных выполняться в какой-то из этих ветвей при однократном исполнении алгоритма. Матрица независимости служит инструментом анализа независимости операторов по данным и управлению. Алгоритм ее нахождения описан в работе [9].

Диаграммы выполнения операторов в ранние и поздние сроки используются для построения нитей решаемой задачи [10] – последовательности программных модулей, предназначенных для выполнения на одном вычислительном модуле. Построение нитей граф-схемы реализовано в блоке 11. Они используются для вычисления перспективных планов загрузок программных модулей для планировщика заданий соответствующей вычислительной системы.

На основе матриц несовместимости и независимости строится множество взаимо-независимых операторов [11]. Если образовать множество операторов $\{a, b, c, d\}$ взаимо-независимых операторов, можно образовать ленты нитей L_a, L_b, L_c, L_d, которые никак друг с другом не связаны. Используя это свойство, в блоке 12 рассматриваемой системы формируются ленты нитей решаемой задачи.

Ленты нитей при необходимости можно решать параллельно на различных вычислительных системах. В случае нехватки ресурсов для решаемой задачи ленты нитей можно выполнить на одной вычислительной системе последовательно (см. блок 13). Для размещения лент нитей решаемой задачи удобно использовать упомянутую выше матрицу дистанций.

В блоке 14 предусмотрена загрузка требуемого транслятора. Предполагается, что до рассматриваемого момента программные модули представлены в виде граф-схемы решаемой задачи. Поскольку граф-схема представляет собой аналог схемы алгоритма задачи, то необходимо произвести обработку с помощью задаваемых в технических требованиях трансляторов. В соответствие с этим, в блоке 15 рассматриваемой структурной схемы выполняется трансляция лент нитей граф-схемы.

После трансляции программных модулей лент нитей в блоке 16 формируется файл управления параллельным заданием, который подробно описывает задание, отправленное на выполнение. В блоке 17 файл поступает на выполнение в кластер непосредственно вместе с самим заданием.

Рассмотрим подробнее процесс выполнения параллельного задания в условиях функционирования программного продукта IBM Tivoli Workload Scheduler LoadLeveler.

IBM Tivoli Workload Scheduler LoadLeveler. IBM TWS LoadLeveler (далее LoadLeveler) – система управления заданиями, которая позволяет пользователям выполнять большее число заданий за

меньшее время, сопоставляя необходимые заданию вычислительные мощности с доступными ресурсами. Это многофункциональная система управления рабочей нагрузкой для последовательных и параллельных пакетных заданий [12]. LoadLeveler обеспечивает планирование заданий и предоставляет функции для построения, передачи, а также быстрой и эффективной обработки заданий в динамической среде. LoadLeveler отслеживает все ресурсы, используемые каждым последовательным или параллельным заданием, и предлагает несколько вариантов отчетов для отслеживания заданий и использования ресурсов по пользователям, группам, учетным записям или типу за указанный период времени.

Пример среды, в которой LoadLeveler способен планировать выполнение задач, демонстрирует рис. 2. Эта среда представляет собой LoadLeveler кластер – совокупность различных машин или процессоров, используемых LoadLeveler. Для возможности планирования выполнения заданий в LoadLeveler процессор должен входить в состав кластера.

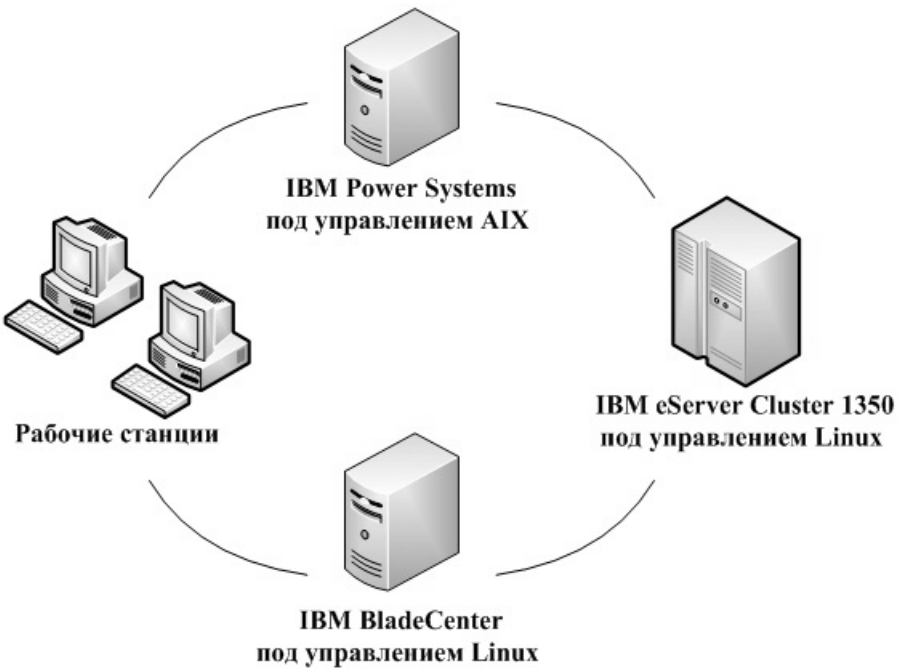

Рис. 2. Пример LoadLeveler кластера

Прежде чем послать задание на выполнение или выполнить другие запросы, связанные с исполнением задания, необходимо создать файл управления заданием (job command file), который описывает задание, отправленное на выполнение, и может включать операторы LoadLeveler. Подробная структура файла управления заданием описана в работе [12].

LoadLeveler планирует выполнение заданий на одном и более процессорах. В данном контексте задание представляет собой совокупность шагов. Для каждого шага можно задать собственное выполнение (в каждый конкретный момент выполняется конкретная часть задания). LoadLeveler можно использовать для выполнения заданий, состоящих из одного и более шагов, каждый из которых зависит от результатов выполнения предыдущего шага задания.

Поток шагов задания представлен на рис. 3. Каждый из шагов описан в файле управления заданием, который определяет имя задания, выполняемый шаг. В файле также могут содержаться другие данные для LoadLeveler [12].

Рис. 3. Шаги задания в LoadLeveler

LoadLeveler пытается выполнить каждый шаг на процессоре, который обладает необходимыми для этого ресурсами, и останавливается после выполнения шага. Если файл управления заданием состоит из нескольких шагов, они могут выполняться на разных процессорах, если явно не определено иное.

Для выполнения параллельных заданий (размещения узлов, назначения заданий узлам и их запуска) LoadLeveler взаимодействует со средой параллельной работы Parallel Operating Environment (POE).

Каждый процессор в кластере LoadLeveler выполняет одну или несколько ролей при планировании заданий:

• планировочный процессор (scheduling machine) или планировщик. Когда задание приходит на выполнение, оно поступает в очередь, управляемую планировщиком, который связывается с процессором, осуществляющем центральное управление всего кластера. Планировщик запрашивает у управляющего процессора свободную для выполнения задания машину, а также хранит неизменную информацию о задании. Некоторые планировочные процессоры являются открытыми – любой пользователь имеет доступ к ним. Такие процессоры планируют выполнение заданий, пришедших от подчиненных процессоров, описанных ниже;

• центральный управляющий процессор (central manager machine). Его задача – проверить необходимые для задания требования и найти один или более процессоров в кластере, которые его выполнят. После того, как процессор (процессоры) будет найден, управляющий процессор извещает об этом планировщика;

• исполнительный процессор (executing machine). Этот процессор непосредственно выполняет задания;

• подчиненный процессор (submitting machine). Подчиненный процессор, известный также как submit-only, представлен в кластере в ограниченном виде. Хотя название подразумевает, что пользователи этих процессоров могут только оправлять задания на выполнение, они также могут вызывать и отменять задания. Пользователи имеют собственный графический пользовательский интерфейс Graphical User Interface (GUI), обеспечивающий работу со множеством функций. Особенность подчиненных (submit-only) процессоров позволяет рабочим станциям, которые не входят в состав кластера, отправлять задания на выполнение.

Один и тот же процессор может выполнять несколько функций, как показано на рис. 4.

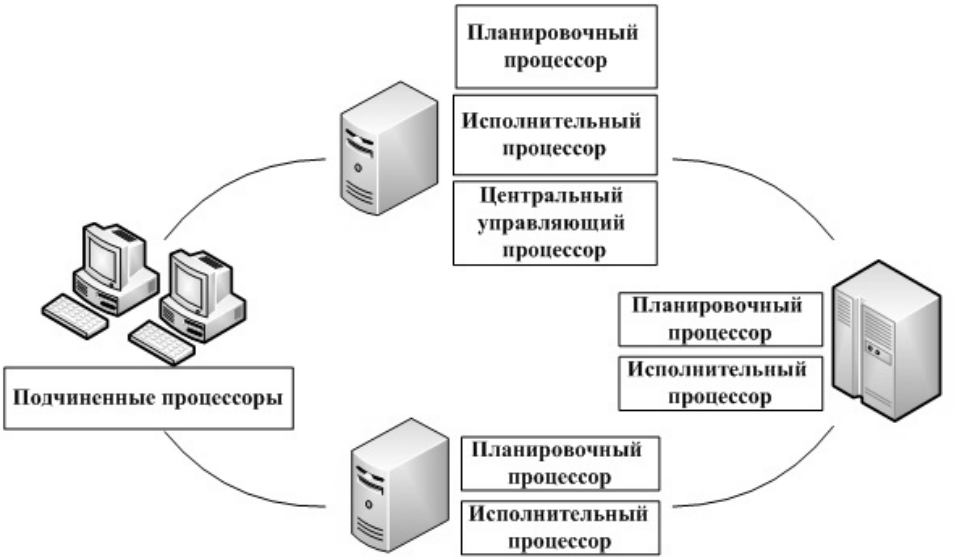

Рис. 4. Функции, выполняемые процессорами

После того, как пользователь отправит задание на выполнение, LoadLeveler проверяет файл управления заданием с целью определения требуемых для его выполнения ресурсов и устанавливает, какой процессор или группа процессоров лучше всего подходит для предоставления этих ресурсов, после чего отправляет задание на соответствующие процессоры. Для этого LoadLeveler использует очереди. Очередь заданий (job queue) – список заданий, которые

ждут своего выполнения. Если LoadLeveler не готов отправить пришедшее задание на выполнение на другом процессоре, оно поступает во внутреннюю базу данных, находящуюся на одном из процессоров кластера.

Процесс управления выполнением задач, называемый диспетчеризацией, заключается в автоматической обработке множества заданий и включает:

- определение времени запуска для каждого задания и доступных исполнительных ресурсов – планирование;
- доставку входных файлов на исполнительный узел;
- мониторинг выполнения задания и доставку результата его выполнения.

Одним из основных требований к управлению заданиями является гарантирование запуска, т. е. на момент запуска в системе должны существовать подходящие для задания свободные ресурсы.

По сравнению с обычными однопроцессорными заданиями, управление параллельными заданиями значительно более сложно: необходимо подбирать исполнительные ресурсы так, чтобы все его процессы стартовали и завершались одновременно – задача коаллокации ресурсов.

К настоящему времени для многопроцессорных компьютеров и кластеров разработан ряд методов, с помощью которых решается задача планирования параллельных заданий. LoadLeveler включает в себя планировщик BACKFILL, определяющий необходимые ресурсы для выполнения задания и рассчитывающий ближайшее время освобождения этих ресурсов.

BACKFILL. BACKFILL – метод обратного заполнения, при котором пользователь дает оценку времени выполнения своих заданий, что позволяет выделять ресурсы заданиям не в момент освобождения, а заблаговременно [13]. Для этого строится план распределения ресурсов – расписание запусков заданий. При построении расписания ресурсы выделяются заданиям в порядке их приоритетов, причем задание может получить некоторые ресурсы только при условии, что они уже не отведены более приоритетным заданиям (в консервативном варианте метода) или самому приоритетному заданию (в агрессивном варианте метода). С помощью механизма предварительного резервирования метод гарантирует получение ресурсов высокоприоритетными заданиями и, вместе с тем, допускает нарушение порядка очереди, что способствует повышению коэффициента общей загрузки ресурсов.

Пример прохождения потока информации о задании в LoadLeveler кластере представлен на рис. 5.

Стрелки на рис. 5 указывают следующее:

- стрелка 1 – задание было отправлено пользователем в LoadLeveler кластер;

• стрелка 2 – планировщик сообщает центральному управляющему процессору, что задание было отправлено, и выясняет, есть ли свободный процессор, удовлетворяющий требованиям задания;

• стрелка 3 – управляющий процессор проверяет, есть ли процессоры, способные выполнить задание. Когда такой процессор найден, управляющий процессор сообщает планировщику, что процессор доступен;

• стрелка 4 – планировщик обращается к исполнительному процессору и передает ему всю информацию о задании.

Рис. 5. Пример прохождения задания в LoadLeveler кластере

Задание может выполняться любым процессором или, в случае параллельного задания, несколькими. Когда задание достигнет исполнительного процессора, оно выполнится.

Таким образом, разработана система решения параллельных задач на однородных вычислительных системах, реализующая распараллеливание алгоритма решаемой задачи, построение соответствующей граф-схемы и средств ее анализа с целью формирования параллельного задания и его дальнейшего выполнения на вычислительных узлах кластера под управлением системы IBM Tivoli Workload Scheduler LoadLeveler.

СПИСОК ЛИТЕРАТУРЫ

1. Руденко Ю. М., Мельдианов П. В. Алгоритм преобразования последовательного алгоритма в параллельный // Информатика и системы управления в XXI веке. Сб. тр. молодых ученых, аспирантов и студентов. – М., 2007. – Вып. 5. – С. 127–131.
2. Руденко Ю. М. Представление параллельных алгоритмов в виде граф-схем // Аэрокосмические технологии. Научные материалы международной научн.-техн. конференции МНТК. – М., 2009. – С. 179–181.

3. Мусина Л. В., Руденко Ю. М. Временная задержка на вычислительных модулях при реализации граф-схем // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. Спец. выпуск «Информационные технологии и компьютерные системы». – 2011. – С. 70–74.
4. Руденко Ю. М. Построение плана выполнения параллельных алгоритмов на базе граф-схем // Аэрокосмические технологии. Научн. материалы международной научно-технической конференции МНТК. – 2009. – С. 179–181.
5. Касьянов Е. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. – Санкт-Петербург, БХВ-Петербург, 2003. – 1104 с.
6. Руденко Ю. М. Учет зависимостей программных модулей по данным и последовательностям их выполнения при параллельных вычислениях // Известия высших технических заведений. Технические науки. – Поволжский регион, 2009. – Вып. 3. – С. 67–75.
7. Руденко Ю. М., Волкова Е. А. Вычислительные системы. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2010. – 211 с.
8. Руденко Ю. М., Кукса П. П., Шмаков Е. В. Алгоритм нахождения матрицы логической несовместимости // Сб. науч. тр. – М., 2002. – С. 195–197.
9. Руденко Ю. М., Шмаков Е. В. Уменьшение количества транзитивных связей в матрице независимости при планировании параллельных вычислений // Тезисы доклада на международной научно-технической конференции, посвященной 80-летию гражданской авиации России, «Гражданская авиация на современном этапе развития науки, техники и общества». – М., 2003. – С. 167.
10. Руденко Ю. М. Построение нитей в информационном графе, представляющем решаемую задачу // Современные информационные технологии. Сб. тр. кафедры МГТУ им. Н.Э. Баумана. Факультет «Информатика и системы управления», кафедра ИУ6 «Компьютерные системы и сети». – М., 2008. – Вып. 3. – С. 142–146.
11. Руденко Ю. М., Неустроева М. С, Жежель С. С. Сравнительная оценка алгоритмов построения полных множеств взаимонезависимых операторов // Современные информационные технологии. Юбилейный сб. тр. кафедры ИУ6, посвященный 180-летию МГТУ им. Н.Э. Баумана. – М., 2011.
12. Tivoli Workload Scheduler LoadLeveler. Using and Administering. Version 3, Realease 4 NY: IBM, 2006. – 698 p.
13. Коваленко В. Н., Коваленко Е. И., Корягин Д. А., Семячкин Д. А. Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами. – М.: ИПМ им. М.В. Келдыша РАН, 2007. – 28 с.

Статья поступила в редакцию 14.05.2012