УДК 519.6

СИСТЕМА ПАКЕТНОЙ ОБРАБОТКИ ЗАДАНИЙ JAM

А. Б. Киселёв, С. Н. Киселёв
(РФЯЦ-ВНИИЭФ)

Описываются возможности системы пакетной обработки заданий JAM, дается подробная схема взаимодействия составляющих систему программных компонентов. Приводятся описания алгоритмов планирования заданий (метод изменения очередности выполнения задания backfill, алгоритмы *лучшего* и *худшего заполнения* — best-fit и worst-fit), а также программных решений, которые реализованы в этой системе.

Ключевые слова: многопроцессорная вычислительная система, пакетная обработка заданий, планирование пакетного задания, программный метод изменения очередности выполнения задания (backfill).

Введение

Система пакетной обработки заданий JAM (СПО JAM) ориентирована на решение ряда важных проблем крупного вычислительного центра, среди которых эффективное использование вычислительных ресурсов многопроцессорной вычислительной системы, прохождение коротких по времени выполнения заданий на фоне обработки длительных и высокоприоритетных заданий в условиях массового счета задач коллективом пользователей.

В планировщике заданий СПО JAM для кластера с большим числом процессоров реализован программный метод заполнения свободных участков вычислительного поля (backfill [1—6]), его комбинации с алгоритмами *худшее заполнение* и *лучшее заполнение* (worst-fit и best-fit), а также алгоритм распределения заданий по узлам кластера с учетом обнаруженных дефектов аппаратно-программного состояния компонентов и структуры коммуникационной среды. Реализация этих алгоритмов планирования заданий и распределения ресурсов кластера позволила большому коллективу пользователей сочетать разработку и отладку прикладных программ, а также счет производственных задач, соблюдая установленный порядок.

СПО JAM функционирует в ОС Unix/Linux. Система интегрирована в единую систему управления заданиями (ЕСУЗ) [7] неоднородного вычислительного комплекса кластеров, где она играет роль системы управления заданиями отдельного кластера.

Программные компоненты СПО JAM

В состав программного обеспечения СПО JAM входят следующие компоненты:

- пользовательский интерфейс;
- сервер обработки команд и пакетных заданий[1];
- планировщик заданий;
- исполнительный демон;
- сервер сбора стандартного вывода и диагностики.

[1]Пакетный режим не допускает интерактивного взаимодействия пользователя со своей программой, поскольку система пакетной обработки выполнит задание тогда, когда это позволит административная политика и будет доступно необходимое количество вычислительных ресурсов. Пакетное задание — это обычно командный файл интерпретатора shell, который содержит некоторую информацию управления и резервирования вычислительных ресурсов (называемую атрибутами, опциями или флажками); в нем обычно указывается сценарий запуска исполняемой программы (например, mpirun) и название файла, который ее содержит, количество вычислительных ресурсов, которые необходимы для ее нормального выполнения, и время, в течение которого программа будет их использовать. Переданное в систему пакетной обработки задание помещается в очередь, которая соответствует указанным в задании атрибутам и определяет политику его обработки (планирования).

Взаимодействие серверных и исполнительных компонентов реализовано с применением технологии JINI [8] с сервисом JavaSpace [9]. Вся связанная с обработанными заданиями информация хранится в базе данных СПО JAM, управляемой СУБД Apache Derby [10].

Пользовательский интерфейс (утилиты, позволяющие с помощью командной строки вводить в СПО JAM задание, управлять его прохождением, осуществлять настройку системы) устанавливается на пользовательские инструментальные серверы кластера. Сервер обработки пакетных заданий, планировщик, сервер сбора стандартного вывода и диагностики устанавливаются на служебные инструментальные серверы.

Программный сервис JavaSpace

Программная технология JINI с сервисом JavaSpace [9] позволяет создавать распределенные приложения. В программную модель JavaSpace входит контейнер, предназначенный для хранения Java-объектов и обмена ими между процессами, распределенными по разным ЭВМ. Благодаря JavaSpace такие процессы могут осуществлять координацию своих действий. JavaSpace используется для обмена служебными сообщениями между планировщиком и исполнительными демонами, сбора стандартного вывода и диагностики каждого процесса параллельной программы. СПО JAM имеет два экземпляра JavaSpace: один — для передачи служебных сообщений между исполнительными демонами и планировщиком, другой — для временного хранения данных из стандартных потоков вывода и диагностики.

СПО JAM использует такие возможности JavaSpace, как поиск/чтение информации по шаблону и синхронизация данных.

Описание работы СПО JAM

Схема работы СПО JAM представлена на рис. 1.

С помощью пользовательского интерфейса содержимое файла задания передается серверу обработки команд и заданий. Сервер выполняет проверку полученной информации, присваивает заданию уникальный идентификатор, преобразует его во внутренний формат СПО JAM и фиксирует в очереди системы.

После постановки задания в очередь его обрабатывает планировщик, который учитывает некоторую административную политику в отношении пользователя или вычислительного ресурса, приоритет задания, его ресурсные требования. Планировщик выделяет заданию нужное

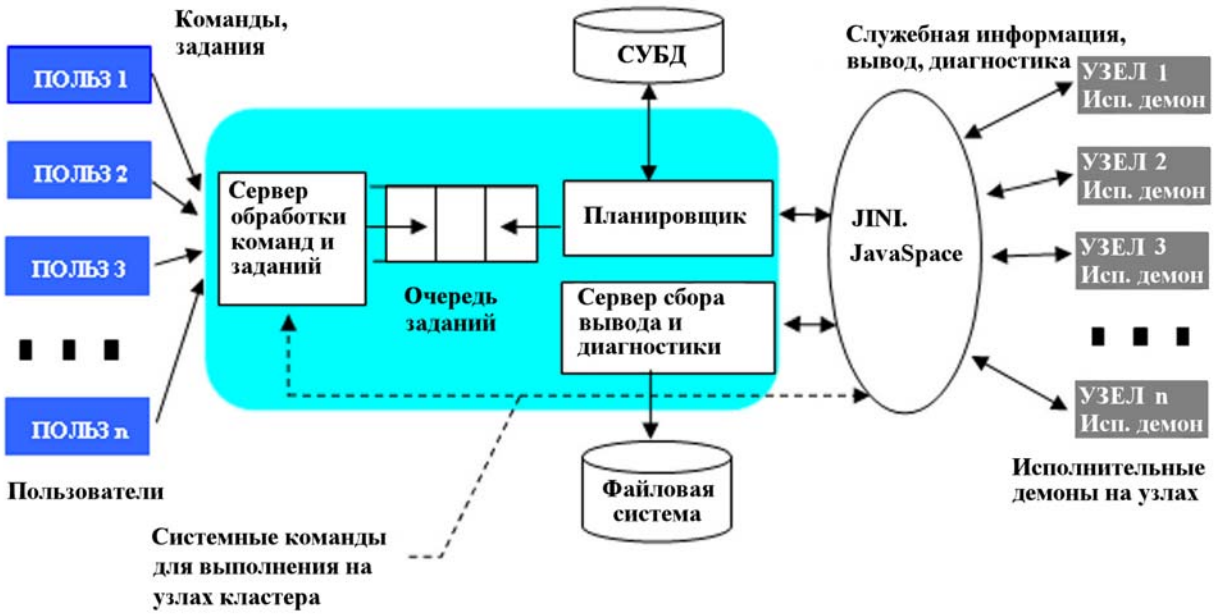

Рис. 1. Схема работы СПО JAM

количество узлов на время, которое указал пользователь[2], и запускает задание на выполнение.

Первый шаг выполнения задания — это запуск командного файла интерпретатора shell или программы, указанной в прологе задания. При ненулевом коде возврата программы, означающем ее аварийное завершение, задание удаляется из очереди. В случае нормального завершения программы (код возврата нулевой) планировщик продолжает обработку задания. Информация, характеризующая задание (команда запуска, идентификатор, атрибуты, названия узлов и т. д.), в некотором структурированном виде записывается в JavaSpace для каждого исполнительного демона на вычислительном узле, выделенном для задания. Если пользователь заказал сбор потоков стандартного вывода и диагностики в один файл, планировщик запускает экземпляр сервера сбора стандартного вывода и диагностики, который будет обслуживать только данное задание. В противном случае стандартный вывод и диагностика каждого процесса программы будут направлены в отдельные файлы непосредственно на вычислительном узле кластера.

Исполнительные демоны считывают из JavaSpace предназначенную для них служебную информацию, выполняют узловой (системный) программный пролог[3] и указанный в задании сценарий запуска пользовательской программы. В течение всего времени выполнения пользовательской программы исполнительные демоны посредством JavaSpace сообщают планировщику ее статус. Если в задании указаны атрибуты, управляющие выводом информации, то исполнительный демон направляет потоки стандартного вывода и диагностики посредством JavaSpace серверу сбора стандартного вывода и диагностики.

Исполнительный демон периодически выполняет сценарий мониторинга состояния аппаратуры и системных программных сервисов узла. Информация о статусе узла посредством JavaSpace передается управляющему серверу.

Статус исполняемой программы анализируется планировщиком. Если появилось сообщение, содержащее информацию о завершении процесса программы, планировщик останавливает обработку задания. Посредством JavaSpace планировщик посылает сообщение на удаление всех процессов задания. Получив сообщение, исполнительные демоны удаляют контролируемые ими процессы, выполняют системный эпилог[4] и с помощью JavaSpace сообщают планировщику о завершении процессов задания на узле.

Последние шаги в обработке задания — это запуск программы, указанной в эпилоге задания, принудительное завершение сервера сбора вывода и диагностики, удаление задания из очереди.

Возможности системы

Мониторинг состояния кластера. Сценарий мониторинга аппаратуры и системных программных сервисов (командный файл интерпретатора shell, выполняемый каждым исполнительным демоном) позволяет контролировать общее состояние кластера. Записав в этот файл необходимые команды, администратор может организовать мониторинг аппаратно-программных компонентов вычислительного узла.

Функционирование узла контролируется не только по отдельным его параметрам, но и по времени связи узла с планировщиком СПО JAM. Если в течение установленного администратором периода времени планировщик не получил служебного сообщения от исполнительного демона, то узел переводится в статус нерабочего. Такой узел исключается из списка вычислительных ресурсов, доступных для выполнения заданий. Если планировщик через какое-то время все-таки получит служебное сообщение, которое не будет содержать информации о сбое, то узел автоматически будет переведен в разряд рабочих. Планировщик не завершает задания, которое исполняется на проблемном узле.

Администратор системы может узнать причину сбоя узла с помощью соответствующей ути-

[2] Планировщик может использовать историю выполнения задания, хранящуюся в базе данных СПО JAM, для замены переоцененного пользователем времени счета на реальное значение, взятое из истории.

[3] Пролог, исполняемый узловым демоном, — это командный файл интерпретатора shell. Он может содержать любые системные команды. В последней версии СПО JAM с помощью пролога узловой демон удаляет из оперативной памяти любые программы, которые системными средствами идентифицированы как пользовательские.

[4] Эпилог, который запускает исполнительный демон, — это командный файл интерпретатора shell. Файл может содержать любые системные команды. С помощью эпилога демон выполняет удаление пользовательских процессов, которые по каким-то причинам не смогли завершиться самостоятельно.

литы. Кроме того, информация о состоянии обнаруженных подсистемой мониторинга проблемных узлов может быть автоматически послана администратору системы по электронной почте.

Мониторинг состояния задания. В СПО JAM создан механизм, позволяющий контролировать *зависание* выполняющейся пользовательской программы. Данный механизм активизируется с помощью некоторого атрибута, который указывается в задании. Система контролирует обновление файла стандартного вывода задания на предмет его изменения программой в течение указанного атрибутом периода времени. *Зависшее* задание автоматически удаляется, а создавшему задание пользователю по электронной почте посылается уведомление (эта операция должна быть им заказана).

Синхронизация работы исполнительных демонов, обслуживающих задание. Постоянная работа системных процессов на узле (в частности, исполнительного демона СПО JAM) может помешать синхронизованному выполнению процессов пользовательского задания. Чтобы снизить это негативное воздействие, введен программный механизм синхронизации работы узловых демонов, который действует в течение всего времени выполнения на узле процессов пользовательской программы. Синхронизация работы исполнительных демонов выполняется по времени, поэтому принципиально важным моментом является синхронизация времени на всех узлах кластера.

Операция инициирования синхронизованной работы исполнительных демонов начинается в момент старта сценария запуска пользовательской программы на узле благодаря присутствию в служебной информации параметра, содержащего календарное время начала синхронизованной работы узловых демонов. С указанного в параметре времени все демоны выполняют операцию контроля аппаратно-программной среды узла и запись служебных сообщений в JavaSpace через временной интервал, числовое значение которого находится в конфигурационном файле исполнительного демона.

В синхронизованном режиме исполнительный демон выполняет свою работу, руководствуясь циклической последовательностью: мониторинг аппаратно-программной среды, запись служебного сообщения в JavaSpace, бездействие.

Сбор информации из стандартных потоков задачи. Сервер вывода. В СПО JAM существуют два режима сбора потоков стандартного вывода и диагностики *параллельной* программы. Первый режим — это запись в файлы непосредственно на вычислительном узле, второй — перенаправление серверу вывода.

Если в задании используется не адаптированный для СПО JAM сценарий, то потоки стандартного вывода и диагностики не контролируются СПО, управление ими осуществляется согласно этому сценарию.

Если в задании указан адаптированный для СПО JAM сценарий, то пользователь имеет возможность управлять потоками стандартного вывода и диагностики. Потоки могут быть направлены в файлы непосредственно на узле или переданы серверу вывода, который посредством JavaSpace принимает данные и записывает их в два файла.

Удаленное выполнение системных утилит. В СПО JAM созданы программные средства, позволяющие удаленно конфигурировать кластер или управлять его ресурсами. С помощью исполнительных демонов можно выполнить на вычислительных узлах любые команды интерпретатора shell (монтирование раздела файловой системы, создание каталога и т. д.). Поддерживается последовательный и параллельный режимы выполнения. При наличии ошибки исполнительный демон передает серверу обработки команд диагностическое сообщение.

Функция экономии электроэнергии. В СПО JAM создан программный механизм выключения простаивающих вычислительных узлов, например, в ночное или нерабочее время. Данный механизм основан на использовании утилиты из пользовательского интерфейса, которая инициирует включение/выключение электропитания компьютеров, укомплектованных BMC-картами (Board Management Controller). Данная функция обеспечивается планировщиком, который при выполнении операций выключения электропитания использует составленное на будущее время расписание выполнения заданий и файл с описанием команд включения и выключения каждого узла кластера. При выполнении операции включения вычислительных узлов планировщик использует список выключенных им узлов и файл с соответствующими командами.

Особенности планирования заданий

Размещение процессов задания на смежных вычислительных узлах. Планировщик СПО JAM поддерживает размещение задания на смежных узлах. Для этого используется специальный файл, в котором все вычислительные узлы распределены по группам. Каждая группа — это узлы, подключенные к одному коммутатору коммуникационной сети кластера.

Операцию размещения задания на смежных узлах будем называть *проецированием*. Суть проецирования заключается в следующем. Вычислительные ресурсы выделяются заданию таким образом, чтобы процессы указанной в задании программы выполнялись на смежных узлах, связанных с одним коммутатором высокопроизводительной сети.

Переходя от узлов к смежным коммутаторам, можно утверждать, что процессы могут обмениваться сообщениями внутри множества соседствующих друг с другом коммутаторов, при этом дистанция между процессами будет минимальной. Использование проецирования теоретически может ускорить счет параллельного приложения.

Если к некоторому множеству коммутаторов подсоединено по одному узлу, то в группы можно объединить узлы, маршрут между которыми содержит минимальное количество переходов между коммутаторами.

Поддержка пластичных заданий. Некоторые задания позволяют более эффективно использовать ресурсы кластера, поскольку содержат заказ *интервала* процессоров и времени. Далее в отношении таких заданий будет употребляться термин *пластичные задания.* Задания, которые не обладают этим свойством, будут называться *монолитными*, или *твердыми.*

Пластичность по времени выполнения. Пластичность задания по времени заключается в том, что в нем пользователь указывает время счета в виде интервала $[\text{ВРЕМЯ}_{\min}; \text{ВРЕМЯ}_{\max}]$. $\text{ВРЕМЯ}_{\min}$ выполняет еще и роль шага, поэтому выделяемое для такого задания время будет равно $N \cdot \text{ВРЕМЯ}_{\min} \leq \text{ВРЕМЯ}_{\max}$, где N — натуральное число, вычисляемое планировщиком.

Свойство пластичности по времени выполнения позволяет планировщику заполнять такими заданиями временные пустоты. Выполняясь в период времени, входящий в заданный интервал, пластичное задание должно постепенно истратить все заказанное максимальное время.

В направлении поддержки свойства пластичности по времени СПО JAM тесно связана с ЕСУЗ [7], на которую возложены функции вычисления остатка времени счета и продолжения обработки пластичного задания, если остаток времени ненулевой.

Поддержка пластичности по количеству процессоров. Масштабируемость по процессорам означает возможность выполнения задания на любом количестве процессоров из указанного в задании диапазона. Если в задании указан интервал процессоров $[\text{ПРОЦ}_{\min}; \text{ПРОЦ}_{\max}]$ и шаг, то ресурсы для него выделяются, исходя из следующей формулы: $\text{ПРОЦ}_{\min} + N \times \times \text{ШАГ} \leq \text{ПРОЦ}_{\max}$, где N — натуральное число, вычисляемое планировщиком.

Обработка многошаговых заданий. Для удобства пользователей в СПО JAM автоматизирована операция создания цепочки связанных заданий. Пользователь предоставляет системе файл с описанием зависимых друг от друга заданий. Каждому звену-заданию пользователь присваивает некоторое уникальное название. Обрабатывая файл, СПО JAM меняет названия звеньев цепочки на собственные идентификаторы.

Планировщик автоматически удаляет звенья многошагового задания, если условия выполнения не могут быть удовлетворены, например, код завершения предыдущего шага ненулевой.

Административная политика распределения вычислительных ресурсов и планирования заданий. СПО JAM позволяет группировать вычислительные узлы кластера в разделы, которые могут использоваться для выполнения заданий отдельных пользователей или их групп. Раздел создается на основе обработки файла административной политики, в котором могут быть указаны следующие характеристики:

- владелец раздела (пользователь или группа пользователей);
- количество или список названий вычислительных узлов;
- время начала и завершения действия политики;
- алгоритм планирования заданий, для выполнения которых создан раздел (backfill, backfill с worst-fit, backfill с best-fit);

– алгоритм распределения вычислительных ресурсов заданиям;
– ограничение времени выполнения;
– ограничение количества заданий одного пользователя.

В алгоритме распределения вычислительных ресурсов реализованы три метода. Первый выделяет заданиям узлы, принадлежащие только данному разделу. Второй метод обеспечивает задание, которое больше ширины раздела, дополнительными вычислительными ресурсами. Такое задание займет все вычислительные узлы раздела и часть не принадлежащих ему узлов. Третий метод позволяет размещать задание вне раздела.

Алгоритм планирования заданий

Основной алгоритм. Планировщик СПО JAM планирует выполнение заданий в соответствии с реализованным в нем алгоритмом backfill и его комбинациями с алгоритмами worst-fit и best-fit.

Алгоритм планировщика backfill модифицирован для выполнения следующих требований:
– обработка заданий в соответствии с планом производственного счета;
– распределение вычислительных ресурсов в соответствии с позицией задания в очереди;
– автоматическое планирование заданий;
– достижение максимальной пропускной способности кластера при отсутствии операторского управления;
– работа под общим управлением ЕСУЗ.

Комбинации backfill с алгоритмами worst-fit и best-fit созданы для обеспечения режима, при котором производительность системы является единственной задачей и не важен порядок прохождения заданий на кластере. Распределение заданий по алгоритму backfill с best-fit или worst-fit реализуется с помощью формирования списка заданий одного приоритета и его сортировки по убыванию величины, получаемой перемножением заказанного времени счета на количество выделенных для задания узлов. Первое запланированное задание в списке лучше заполнит вычислительные ресурсы (best-fit), а последнее соответственно хуже (worst-fit).

Оптимизация. Эксплуатация СПО JAM выявила прогрессирующее замедление при обработке нескольких десятков заданий. Планировщик справлялся со своей работой благодаря небольшому числовому значению конфигурационного параметра, ограничивающему количество обрабатываемых заданий. Если указано большое значение параметра, планировщик может составить более качественное расписание — это повышает пропускную способность системы и загрузку кластера, но требует довольно значительного времени из-за сложности алгоритма backfill. Если значение слишком мало, то некоторая часть заданий будет ожидать своей очереди даже тогда, когда для их выполнения есть свободные ресурсы. Логично найти разумное соотношение между количеством обрабатываемых заданий и качественной характеристикой алгоритма планировщика — скоростью его работы.

В код планировщика были внесены алгоритмические изменения, позволившие значительно повысить число обрабатываемых заданий без значительного увеличения времени планирования. В результате модификаций сложность алгоритма планирования составила $O(n^3)$, в то время как немодифицированная версия имела сложность $O(n^4)$.

На рис. 2 представлены графики работы последней версии планировщика при специально организованной симуляции обработки 100 заданий с заказом 1 600 узлов по 8 процессоров и с заказом 12 800 процессоров.

Из графиков видно, что обработка заданий, для которых требуются процессоры, производится быстрее. Поэтому, чтобы ускорить работу

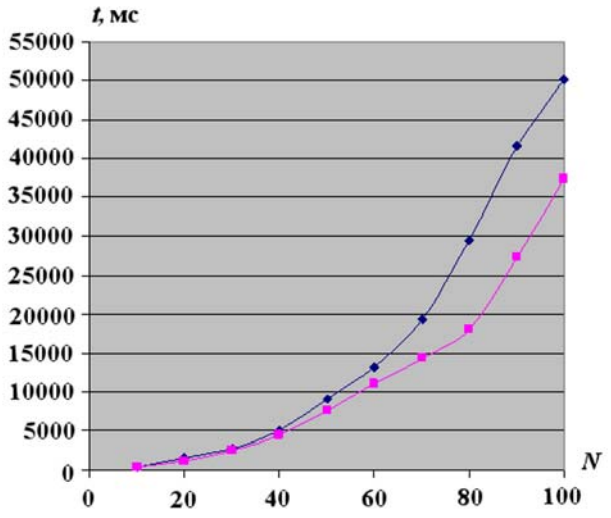

Рис. 2. Время, потраченное планировщиком со сложностью алгоритма $O(n^3)$ на обработку заданий с заказом узлов (–♦–) и процессоров (–■–)

планировщика, СПО JAM принудительно преобразует (если это можно сделать) заказ узлов к заказу процессоров.

Использование алгоритмов worst-fit и best-fit почти не влияет на общее время работы планировщика, поскольку реализованный программный код выполняет обработку 1 000 заданий по каждому алгоритму не более чем за 60 миллисекунд.

Заключение

Система прошла длительный этап тестирования, модернизации и использования на разных аппаратно-программных платформах. При разработке СПО JAM учтен опыт эксплуатации таких программных продуктов, как Open PBS [11] и Torque [12]. В системе реализованы программные решения, которых нет ни в SLURM [13], ни в LoadLeveler [14], такие как:

- взаимодействие серверных и исполнительных компонентов с применением технологии JINI с сервисом JavaSpace;
- направление программному серверу стандартных потоков вывода и диагностик процессов программы пользователя;
- настройка исполнительных демонов на синхронизованную работу перед началом обслуживания задания;
- размещение задания на смежных узлах, подключенных к смежным коммутаторам коммуникационной сети;
- настраиваемый программный механизм контроля состояния задания, позволяющий автоматически завершать зависшее задание.

В списке программного обеспечения системы есть конверторы, которые трансформируют задания для Open PBS (Torque), SLURM в формат СПО JAM. Система укомплектована руководствами администратора, оператора и пользователя. В состав дистрибутива входят адаптированные для СПО JAM сценарии запуска параллельных приложений и их исходные тексты.

Список литературы

1. *Mu'alem A. W., Feitelson D. G.* Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling // IEEE Trasns. Parallel & Distributed Syst. 2001. Vol. 12 (6). P. 529—543.
2. *Snell Q. O., Clement M. J.* Preemption Based BackFill. Provo, Utah.: Brigham Young University, 2002.
3. *Lowson B. G.* Self-adapting Backfilling Scheduling for Parallel Systems. Richmond, 2001.
4. *Fietelson D. G., Rudolph L., Wong P.* Theory and practice on parallel job scheduling. Lecture notes in computer science. Vol. 1291. Springer Berlin, 1997. P. 1—34.
5. Maui Cluster Scheduler. http://www.supercluster.org/maui.
6. Maui Molokini Edition Scheduler. http://sourceforge.net/scheduler.
7. *Киселёв А. Б., Бартенев Ю. Г., Вархин А. М. и др.* Единая система управления заданиями на ЭВМ неоднородного вычислительного комплекса // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2008. Вып. 1. С. 60—66.
8. JINI Starter Kit. http://www.jini.org.
9. *Freeman E., Hupfer S., Arnold K.* JavaSpaces^{TM} Principles, Patterns, and Practice. Addison Wesley, 1999.
10. Apache Derby RDBMS. http://www.apache.org/db/derby.
11. Open PBS. http://www.openpbs.org.
12. Torque batch system. http://www.clusterresources.org/torque.
13. SLURM. http://www.llnl.gov/linux/slurm.
14. LoadLeveler. http://www.mhpcc.edu/training/workshop/loadleveler/MAIN.html.

Статья поступила в редакцию 12.03.09.