УДК 681.3

А.Е. Дорошенко, А.П. Розенблат, К.А. Рухлис, Ю.М. Тырчак

МОДЕЛИ И СРЕДСТВА ПРОГРАММИРОВАНИЯ ГРИД-СИСТЕМ

Рассмотрены проблемы программирования Грид-систем и приведена некоторая классификация моделей и средства программирования для Грид. Намечены тенденции развития методов программирования Грид-систем, проиллюстрированные современными примерами их реализации.

Введение

Современные Грид-системы представляет собой инфраструктуру, построенную на основе Интернет и Всемирной Паутины (World Wide Web), которая обеспечивает масштабируемые, безопасные и быстродействующие механизмы для обнаружения и доступа к удаленным вычислительным и информационным ресурсам [1].

Главная цель программирования Грид-систем – построение моделей программирования, инструментальных средств, методов, которые поддерживают эффективную разработку переносимых приложений и позволяют получать высокоэффективные программы для Грид-систем. Программирование Грид часто требует использования возможностей и свойств, лежащих за пределами простого последовательного программирования и даже параллельного или распределенного программирования. Помимо управления операциями над распределенными структурам данных программист Грид должен управлять вычислениями в среде, которая является обычно открытой, разнородной и динамичной в сочетании с многоуровневой памятью, иерархией пропускной способности и латентности системы связи. Поэтому кроме обычных вычислений над структурами данных программист Грид также должен проектировать взаимодействие между удаленными сервисами, источниками данных и ресурсами аппаратных средств. Отсюда возникает все более глубокое понимание того, что современные инструментальные средства и языки программирования недостаточны для поддержки эффективной разработки высокопроизводительных программ для Грид-систем [2].

Приложения Грид являются разнородными и динамическими по своей природе, они выполняются на различных типах ресурсов, конфигурация которых может изменяться во время выполнения приложений. Такие динамические конфигурации могут определяться изменениями в среде, например, изменениями производительности, или отказами аппаратуры, а также потребностью формирования виртуальных организаций [3] из доступных ресурсов Грид. Но независимо от этих причины, модель или инструмент программирования должны давать программисту общее представление о разнородных ресурсах Грид-систем, скрывая их различия и в то же время обеспечивая контроль над каждым типом ресурса в случае необходимости. Главной задачей при этом является нахождение подходящих абстракций программирования, которые могут обеспечивать ”прозрачность” системы при обнаружении ресурсов и сокрытие этих действий во время выполнения программ.

Грид-системы часто используются для крупномасштабных и высокоэффективных вычислений, поэтому получение высокой эффективности вычислений всегда требует определенного равновесия объемов вычислений и коммуникаций для всех вовлекаемых ресурсов. В настоящее время это достигается путем управления вычислениями, обменом и расположением данных, используя передачу сообщений или методы удаленного вызова, а они требуют от программиста знания путей передачи аргументов от источника до назначения. Чтобы достигать высокой производительности вычислений на сильно- или слабосвязанных в Грид-сетях и мультипроцессорных кластерах прикладным системам потребуется обеспечивать большую “зернистость” вычислений, а также использовать большое количество узлов сети с высоким показателем латентности обме-

нов между ними. Высокой степени параллелизма в разнородной среде невозможно будет добиться ручным кодированием приложений, в общем случае крупномасштабное распараллеливание возможно только при условии высокого уровня автоматизации этого процесса.

Современный термин модели программирования включает не только языки программирования, но и многие различные формы поддержки процесса программирования, например, спецификации, библиотеки API, инструментальные комплексы и др. В последнее время модели программирования присутствуют в каркасах, порталах, средах решения задач, хотя обычно они не играют в них главной роли. Наиболее успешные модели программирования позволяют достигать высокой эффективности, гибкой композиции в управлении ресурсами. Модель программирования также влияет на полный жизненный цикл программного обеспечения от проектирования до сопровождения. Следовательно, успешные модели программирования должны также облегчить эффективное использование средств разработки, например, компиляторов, отладчиков, средств контроля производительности и т.д.

В работе представлен краткий обзор общих проблем и моделей программирования для Грид-систем. Рассматривается техника программирования и подходы к усовершенствованию инструментальных средств для решения этих проблем на основе опыта, накопленного в ИПС НАН Украины.

1. Проблемы программирования Грид-систем

В настоящее время существует несколько общих свойств, которые являются желательными для всех моделей программирования.

1.1. Мобильность, интероперабельность и адаптивность. Современные языки высокого уровня позволяют получать программный код, независимый от процессора, на котором он должен выполняться. Модели программирования для Грид также должны давать возможность такой мобильности кода. Это может означать независимость от архитектуры (в смысле универсальной интерпретируемой виртуальной машины), а также использование различных предварительно заготовленных программ (или сервисов) в различных узлах сети, которые обеспечивают эквивалентную функциональность. Такая мобильность кода является необходимым условием для работы с динамическими разнородными конфигурациями.

Использование различных, но эквивалентных, кодов и сервисов подразумевает возможность взаимодействия (*интероперабельность*) реализаций моделей программирования. Понятие открытой и расширяемой архитектуры Грид подразумевает наличие распределенной среды, которая может поддерживать протоколы, сервисы, интерфейсы прикладного программирования и среды программирования, в которых такое взаимодействие является возможным [3]. Впрочем, мобильность и интероперабельность стимулируют такое качество современных Грид-систем как *адаптивность*. Программа для Грид должна уметь адаптировать себя к различным конфигурациям, основанным на доступных ресурсах. Это может происходить во время запуска, или во время выполнения в силу изменяющихся условий или требований или в силу устранения неисправностей. Такая адаптивность может включать, например, простой рестарт на других ресурсах или фактическую миграцию вычислений и перенос данных.

1.2. Обнаружение ресурсов – неотъемлемая часть Грид-вычислений Программы для Грид должны ясно ”находить” соответствующие хост-машины (главные компьютеры), на которых они смогут выполняться. Поскольку в Грид-сети может находится много постоянных (persistent) сервисов, то программы должны уметь обнаруживать эти сервисы и интерфейсы, которые они поддерживают. Построение сервисов должно быть композиционным и программируемым некоторым однородным способом. Поэтому среды программирования и инструментальные средства должны знать о возможностях обнаружения сервисов и предлагать пользователю

явные или неявные механизмы для использования этих сервисов при разработке и развертывании приложений Грид.

1.3. Производительность. Для многих приложений Грид производительность является одной из главных целей их создания. Грид-системы представляют собой набор разнородных ресурсов, включая иерархии вычислительных узлов различной прозводительности, каналы связи различной пропускной способности и системы памяти различной латентности, которые могут затруднить достижение высокой производительности и эффективного использования ресурсов. Соотношение объемов обменов и вычислений, которое может быть поддержано в типичной среде Грид, может особенно затруднять разработку приложений для тесно-связанных систем. Однако для многих приложений не менее важной проблемой является надежная производительность. Динамическая, разнородная среда может давать большой разброс оценок производительности, которые могут иногда быть недопустимыми. Следовательно, в условиях общедоступной среды, качество обслуживания становится все более и более необходимым для достижения надежной производительности для данных конструкций программирования на данной конфигурации ресурсов. Некоторые потребители могут требовать фактически детерминированной модели производительности, поэтому более приемлемым может быть получение надежной производительности в пределах некоторой статистической границы.

1.4. Отказоустойчивость. Динамическая природа Грид-вычислений предполагает обеспечение некоторого уровня отказоустойчивости. Это особенно справедливо для высокораспределенных программ (например, таких как в методе Монте-Карло), когда могут быть инициализированы тысячи подобных друг другу независимых заданий на тысячах компьютеров. Ясно, что с увеличением количества ресурсов повышается вероятность отказа некоторого ресурса. Приложения Грид должны уметь проверять исправность коммуникаций и/или вычислительных ресурсов во время выполнения программ и обеспечивать на программном уровне реакции на ошибки или действия по восстановлению вычислений. При этом в условиях неисправностей инструментальные средства могут обеспечивать минимальный уровень достоверных вычислений, которые реализуются, в основном, механизмами времени выполнения.

1.5. Безопасность. Грид-программы при выполнении обычно пересекают границы многих административных областей, используя общедоступные ресурсы, такие как компьютерные сети. Если важным является обеспечение строгой аутентификации между двумя узлами, то в Грид-системе не редки случаи, когда приложение включает большое количество программно-управляемых узлов. Фактически имеется необходимость отслеживать деревья запросов произвольной глубины, в которых отбор ресурсов решается динамически. Поэтому механизмы защиты, которые обеспечивают удостоверение подлинности и секретность, должны быть неотъемлемой частью модели программирования Грид.

1.6. Метамодели программ. Кроме простого обнаружения интерфейса полная модель программирования Грид требует также описания самих программ. Традиционное программирование на языках высокого уровня при трансляции между двумя моделями программирования основано на технике компиляции и системе команд машины, которые выполняют применение функций к данным, хранящимся в памяти. Часть процесса трансляции – построение ряда семантических моделей программ и их расширений (таких как оптимизация, сборка "мусора", проверки диапазонов значений переменных), которые относятся к уровню метамоделей программирования. Другие, аналогичные метамодели требуются для программирования Грид, где применение расширений усложняется распределенной и разнородной природой Грид-систем.

2. Средства программирования Грид-систем

Пути решения вышеозначенных проблем определяет практика программирования и особенности Грид. За последние двадцать лет в области параллельного и распределенного программирования сформировался некоторый корпус знаний, на которые наибольшее влияние оказали успешные архитектуры аппаратных средств, а также желание разрабатывать программные системы с такими ”хорошими” свойствами, как улучшенная сопровождаемость и возможность повторного использования. Далее в кратком обзоре инструментальных средств, языков, и сред для программирования Грид многие из них имеют корни в ”обычных” параллельных или распределенных системах и находят применение в средах Грид благодаря установившейся методологии программирования.

2.1. Модели с общедоступным состоянием. Модели программирования с общедоступным состоянием (централизованные) обычно ассоциируются с тесносвязанными системами, синхронными языками и моделями реализации, предназначенными для машин с общей (совместно используемой) или гибридной (физически распределенной, но логически общей) памятью. Такие модели имеют место также и для мультипроцессорных систем с распределенной памятью, которые используют специализированные схемы соединений, обеспечивающие очень высокую пропускную способность и низкую латентность каналов связи между процессорами. Несмотря на то, что при невысоких значениях этих показателей в большинстве распределенных Грид-систем, такие модели и основанные на них инструментальные средства неэффективны, тем не менее существует класс моделей программирования, методологически очень близких к вышеупомянутым, которые основаны на общедоступном состоянии, где поставщики и потребители данных непосредственно не связаны, но могут взаимодействовать посредством операций на общедоступных состояниях программной системы.

2.1.1. JavaSpaces – это реализация понятия пространства кортежей координационного языка Linda на основе Java, в которой кортежи представлены сериализуемыми объектами [4]. Использование Java позволяет взаимодействовать разнородным клиентам и серверам независимо от операционных систем архитектуры процессоров. Модель JavaSpaces рассматривает приложение как совокупность процессов, сообщающихся между собой передачи и получения объектов в/из одного из пространств. Пространство кортежей – это общедоступный и постоянный (persistent) объектный репозиторий, который является доступным процессом через сеть. Процессы используют репозиторий как механизм обмена для координации вместо прямой связи друг с другом. Главные действия, которые процессы могут осуществлять с пространством объектов, это *put* (включение), *take* (изъятие), and *read* (чтение, т.е. копирование без изъятия). При выполнении операций изъятия и чтения, получаемый объект определяется ассоциативным соответствием по типу и количеству операндов объектов, помещенных в пространство. Программист, который хочет построить приложение на основе такого координационного пространства, должен проектировать распределенные структуры данных как наборы объектов, которые хранятся в одном или более пространств. Подход, который JavaSpaces предлагает программисту, намного упрощает создание распределенных приложений даже в условиях таких динамических сред как Грид-системы. Реализации JavaSpaces в Грид-системах на основе инструментария Jini/Jxta и Globus предложены в [5, 6].

2.1.2. Механизмы опубликования/подписки. Ассоциативное соответствие, кроме того, что оно является базисной операцией, лежащей в основе JavaSpaces, является также фундаментальным понятием, через которое реализуется ряд важных возможностей, невыполнимых любым другим способом. Эти возможности включает создание коммуникационных подсистем на основе распределения по содержанию (content-based routing), службы событий (event services) и механизма опубликования/подписки. Как вышеописано, это по-

зволяет поставщикам и потребителям данных координировать свои действия, не будучи при этом непосредственно связанными и даже ”знающими” друг друга по имени.

Ассоциативное соответствие – это дорогой способ реализации, особенно в географических широкомасштабных средах. Учитывая большое значение механизма опубликования/подписки для базисных сервисов Грид, таких как службы событий, которые играют важную роль в обеспечении отказоустойчивых вычислений, такая возможность должна быть доступной в некоторой законченной форме. Для этого возможны три различных подхода:

- *сеть серверов* – традиционный подход для многих существующих распределенных систем обслуживания. Здесь один из первых примеров – сервис событий в архитектуре CORBA, в котором осуществляется принцип ”развязанной связи” между поставщиками и потребителями на основе иерархии клиентов и серверов;
- *промежуточное программное обеспечение* – слой программного обеспечения, в котором, в частности, содержится расширенный сервис связи. В качестве примера можно указать системы одноранговых сервисов на уровне приложений (например, Jxta [6]). Алгоритмы распределения (маршрутизации) определяются на определенном приложением иерархическом пространстве имен, в котором наборы ресурсов и объектов могут быть отображены в модификациях маршрутов;
- *наложения сетей (оверлейные структуры)* – используются для выделения проблемы построения топологии распределения и имеют свойства абстрагирования. Благодаря абстрагированию сетевые оверлейные структуры могут превращать изолированные ресурсы в фактически непрерывные с определенной топологией.

2.2. Модели передачи сообщений. В моделях передачи сообщений процессы работают в непересекающихся адресных пространствах и информация передается в виде сообщений между ними. Несмотря на то, что явное распараллеливание с передачей сообщений может быть громоздким, оно дает пользователю полный контроль и таким образом применимо к случаям, где более удобные полуавтоматические методы программирования могут терпеть неудачу. Кроме того, это вынуждает программистов точно учитывать, где может происходить потенциально дорогостоящий обмен сообщениями. Эти два момента важны как для одиночных параллельных компьютеров, так и, даже в большей степени, для сред Грид-сетей.

Интерфейс передачи сообщений MPI [7] – широко принятый стандарт, который определяет библиотеку двусторонних передач сообщений, т. е. таких, где согласованы операции передачи и приема, и который является вполне подходящим для Грид. Существует много реализаций и вариантов MPI, наиболее известна в контексте Грид-вычислений является MPICH-G2 [8].

MPICH-G2 – это реализация MPI с поддержкой Грид, которая использует сервисы Globus [3] (например, запуск задания, обеспечение безопасности) и позволяет программисту соединить много компьютеров, в общем случае, различной архитектуры для выполнения приложений MPI. MPICH-G2 автоматически преобразовывает данные в сообщениях между компьютерами различной архитектуры и поддерживает многопротокольную связь, автоматически выбирая TCP для межмашинной передачи сообщений и поставляемый продавцом MPI протокол для внутримашинной передачи сообщений. Он же освобождает пользователя от продолжительной (и потому нежелательной) работы по изучению подробностей, специфичных для конкретных машин и позволяет пользователю запускать многомашинное приложение одной командой mpirun. MPICH-G2 требует, однако, чтобы сервисы Globus были доступны на всех участвующих компьютерах, чтобы можно было войти в контакт с каждой удаленной машиной, подтвердить подлинность пользователя на каждой из них и начать выполнение (например, выполнив команду ветвления *fork*, поставив задачу в очереди, и т.д.).

Популярность MPI породила ряд вариантов, которые решают проблемы Грид, такие как управление динамическими процессами и более эффективные коллективные операции. Библиотека MagPIe [8], например, реализует коллективные операции MPI, такие как широковещание, барьер и редукция, с оптимизацией для широкомасштабных систем типа Грид. Используя эту библиотеку, существующие параллельные приложения MPI могут быть выполнены на платформах Грид. Библиотека имеет простой API, через который вычислительная платформа Грид обеспечивает информацию о том, сколько имеется доступных кластеров, и какие процессы на каких кластерах зафиксированы.

2.3. Модели RPC и RMI. Модели передачи сообщений, будь она двухточечной, широковещательной или ассоциативно адресованной, все имеют существенный атрибут явно выстраиваемых аргументов, которые передаются согласованному оператору приема, который их разбирает и разрешает обработку, обычно основанную на типе сообщения. Семантика, связанная с каждым типом сообщения, обычно определяется статически проектировщиками приложений. Односторонние модели передачи сообщений изменяют эту парадигму, не требуя соответствия приема и разрешая отправителю определять тип удаленной обработки. Модели для удаленного вызова процедур (RPC) и удаленного вызова методов (RMI) обеспечивают те же возможности, но структурируют взаимодействие между отправителем и получателем больше в виде конструкции языка, чем вызова библиотечной функции. Модели RPC и RMI обеспечивают простой и понятый механизм для управления удаленными вычислениями. Будучи механизмом для управления потоками команд и данных, RPC и RMI также допускают некоторые проверки типов аргументов и их количества. RPC и RMI могут также использоваться для построения моделей высшего уровня для программирования Грид, таких как компоненты, каркасы и сервисы.

2.3.1. RPC с поддержкой Грид. GridRPC – это модель RPC и API для Грид-систем [9]. Помимо стандартной семантики RPC с асинхронным крупнозернистым параллельным выполнением заданий, эта модель обеспечивает удобную высокоуровневую абстракцию, посредством которой многие детали взаимодействия со средой Грид могут быть скрыты. Далее указаны три очень важных возможности для Грид, которыми GridRPC может прозрачно управлять:

- *динамическое обнаружение ресурсов и планирование*: сервис RPC может быть расположен где угодно в Грид-сети. Обнаружение, отбор и планирование удаленного выполнения должны выполняться на основе ограничений пользователя;
- *безопасность*: защита Грид через сертификаты GSI и X.509 является существенной для функционирования в открытой среде;
- *отказоустойчивость*: обеспечивается через автоматические контрольные точки, откаты или повторные выполнения; становится все более существенным по увеличению количества вовлеченных ресурсов.

Администрирование интерфейсов – важная проблема для всех моделей RPC. Обычно она решается на основе языка описания игтерфейсов (IDL). В этом отношении средства GridRPC разрабатываются с рядом других свойств для улучшения применимости, легкости реализации и развертывания, а именно:

- *поддержка "научного" IDL*: в которую включаются большие матрицы как операнды, общая память для матричных операндов, файловые операнды и вызов по ссылке. Полосы и блоки матриц могут быть определены так, чтобы сократить затраты на обмены данными;
- *управление IDL только на стороне сервера:* только серверы GridRPC могут управлять заглушками (stubs) RPC и отслеживать ход выполнения заданий. Следовательно, клиентское взаимодействие простое и требует небольшой поддержки на стороне клиента.

Фундаментальными объектами в модели GridRPC являются функциональные дескрипторы (function handles) и идентификаторы сеанса (session IDs).

Имена функций GridRPC отображаются на сервер, который может вычислить эти функции. Такое отображение впоследствии обозначается функциональным дескриптором. Модель GridRPC не определяет механизм обнаружения ресурса, позволяя таким образом различным реализациям использовать различные методы и протоколы. Все запросы RPC, использующие функциональный дескриптор, выполняются на сервере, указанном дескриптором. Специальный (неблокирующий) вызов RPC обозначается идентификатором сеанса, который может использоваться для проверки состояния вызова, ожидания завершения, отмены вызова или проверки возвращаемого кода ошибки.

Поэтому, GridRPC является прямым расширением понятия сетевого сервиса. Существуют прототипные реализации, построенные на основе систем типа NetSolve [10]. Тот факт, что используется управлением IDL только на стороне сервера означает, что развертывание и поддержка здесь проще, чем при других подходах к организации распределенных вычислений, таких как CORBA, где клиент должен быть изменен, если изменяется сервер. Заметим, что другие механизмы RPC для Грид также возможны. Они включают протоколы SOAP и XML-RPC, которые используют XML поверх гипертекстового транспортного протокола HTTP. В то время как XML обеспечивает большую гибкость, так же он имеет ограниченные возможности для поддержки научных данных и существенную стоимость кодирования. Эти проблемы могут быть решены с введением, например, типов матрицы с двойной точностью и полей двоичной информации. GridRPC можно реализовать на основе открытой архитектуры сервисов Грид OGSA [11].

2.3.2. RMI на Java. Удаленный вызов – известное понятие, которое подкрепило разработку первоначального RPC и затем RMI на Java. Удаленный вызов метода Java дает возможность программисту создавать такие распределенные приложения на Java, в которых методы удаленных объектов Java могут быть вызваны из других виртуальных машин Java, находящихся на различных хост-машинах. RMI наследует базисную конструкцию RPC, но также имеет свои отличительные признаки, которые расширяют возможности базисного RPC. Программа с RMI, выполняющаяся на одной из виртуальных машин (JVM), может вызывать методы других объектов, постоянно находящихся в различных виртуальных машинах (JVM). Главные преимущества RMI заключаются в том, что он является чисто объектно-ориентированным, поддерживает все типы данных Java-программы и поддерживает ”сборку мусора”. Эти данные позволяют четко отличать вызывающий оператор от вызываемого, а разработка и поддержание распределенных систем становятся проще. Таким образом RMI на Java обеспечивает высокоуровневый интерфейс программирования, который может успешно использоваться и для Грид-вычислений.

2.4. Гибридные модели. Существенной чертой Грид-вычислений является стремление сделать доступными для приложений Грид как можно больше компьютеров (хост-машин), находящихся в сети. Следовательно, некоторым приложениям может понадобиться выполнение как в пределах, так вне непосредственно доступных адресных пространств, т. е. они, возможно, будут использовать многопоточный режим в общей памяти, а также, передачу данных и управления между машинами в распределенной среде. Такая ситуация происходит в кламперах (clumps) (кластерах симметричных многопроцессорных систем) и в Грид-системах. Для решения вопросов, связанных с возникающими проблемами, разработано несколько моделей программирования.

2.4.1. OpenMP и MPI – это библиотека, которая поддерживает параллельное программирование в многопроцессорных системах с общей памятью [12]. Она разработана одноименным консорциумом с целью получения стандартного интерфейса программирования для параллельных компьютеров с общей памятью, который может использоваться основными языками программирования, такими как ФОРТРАН, C, и C++. OpenMP позволяет парал-

лельное выполнение кода (параллельные опрераторы цикла), определение общих данных (тип *SHARED*) и синхронизацию процессов.

Комбинация, OpenMP и MPI в пределах одного приложения для получения клампа и среды Грид рассмотривались многими исследовательскими группами ранее. Первый вопрос в этих прикладных системах – вопрос о том, кто управляет. OpenMP – по существу многопоточная модель программирования. Следовательно, OpenMP поверх MPI потребует от MPI безопасности потоков или явного управления доступом к библиотеке MPI, которая поверх OpenMP может требовать дополнительной синхронизации и ограничивать степень параллелизма, которую OpenMP может реализовать. Какой подход реально окажется лучше обычно зависит от приложения.

2.4.2. MPJ. Все эти программные концепции могут быть объединены в один пакет, как это сделано в Java с передачей сообщений, или MPJ [13]. Соображением в пользу MPJ было то, что много приложений естественно требуют скорее симметричной модели передачи сообщений, чем асимметричной модели RPC/RMI. Поэтому MPJ предлагает разработчику прикладных систем многопоточный режим, RMI и передачу сообщений. Такой подход, однако, представляет проблему для реализации. Реализация MPJ поверх собственной библиотеки MPI дает хорошую производительность, но нарушает модель безопасности Java и не позволяет выполнять аплеты. Реализация MPJ на Java обычно дает более низкую производительность. Дополнительная поддержка со стороны компилятора может улучшить общую производительность и сделать этот подход единого языка более реальным.

2.5. Одноранговые модели. Одноранговые вычисления (P2P) [14] представляют собой разделение компьютерных ресурсов и сервисов путем прямого обмена между системами, которые пользуются преимуществом существующих вычислительных возможностей настольных систем и их связностью работы с сетями, позволяя недорогим способом усилить их коллективную мощность и извлечь выгоду в целом для организаций. В архитектуре P2P компьютеры, которые традиционно использовались исключительно как клиенты, могут связываться прямо между собой и действовать как клиенты и как серверы, выполняя эти роли наиболее эффективным способом для сети. Это уменьшает нагрузку на серверах и позволяет выполнять специализированные сервисы (такие как порождение списка рассылки, выставления счета, и т.д.) более эффективно. По мере того, как компьютеры становятся повсеместными, идеи по реализации и использованию вычислений P2P получают все большее значение. Одноранговые вычисления и Грид-технологии концентрируются на гибком разделении и использовании разнородных вычислительных и сетевых ресурсов.

2.5.1. JXTA. Семейство протоколов, специально разработанных для вычисления P2P получило название JXTA [15]. Само происхождение термина JXTA (от ”juxtapose”) означает приближение вычислений P2P к клиент/серверной модели и вычислением на Веб-сети. JXTA представляет собой набор открытых, обобщенных протоколов P2P, определенные в XML-сообщениях, которые позволяют любому подсоединенному к сети устройству, от сотовых телефонов и беспроводных вычислительных устройств до персональных компьютеров и серверов, взаимодействовать в сети способом P2P. При использовании протоколов JXTA, узлы могут сотрудничать для формирования самоорганизующихся и самоконфигурирующихся групп, независимо от их позиций в сети и без необходимости для инфраструктуры централизованного управления. Узлы могут использовать JXTA протоколы для объявления своих ресурсов и обнаружения сетевых ресурсов (сервисы, каналы, и т.д.), доступных от других узлов, которые объединяются в группы для создания специальных отношений между ними. Узлы также сотрудничают для маршрутизации сообщений и формирования полнодоступной связности узлов. Протоколы JXTA позволяют узлам связываться без необходимости понимать или управлять сложными

и динамическими сетевыми топологиями, которые становятся общими. Эти черты превращают JXTA в модель для реализации Грид-сервисов и приложений средствами P2P [6].

2.6. Каркасы, компонентные модели и порталы. Помимо библиотечно-ориентированных подходов и языковых средств для облегчения проектирования и развертывания распределенных приложений разрабатываются полнофункциональные среды программирования. Широкая классификация данных подходов включает каркасы, компонентные модели и порталы. Здесь приводится обзор нескольких важных примеров.

2.6.1. Cactus. Программный код и вычислительный инструментарий Cactus обеспечивает модульный каркас для вычислительной физики [16]. Cactus предоставляет прикладным программистам высокоуровневый API для набора сервисов, приспособленных для вычислительной науки. Кроме поддержки сервисов типа параллельного ввода/вывода и параллельной расстановки контрольных точек и рестарта, имеются сервисы для управления вычислениями (динамическое изменение параметров во время выполнения) и удаленной визуализации. Для построения приложений в системе Cactus пользователь формирует модули, называемые ”thorns” (тернии), которые подключаются к основе (flesh) каркаса.

2.6.2. Объектные и компонентные архитектуры. Общая объектная архитектура брокера запроса (CORBA) – это стандартный инструмент, в котором используется интерфейс метаязыка для управления интероперабельностью объектов. Доступ к элементу объекта определяется, используя язык описания интерфейса IDL. Объектный брокер запроса (ORB) используется для обнаружения ресурсов среди клиентских объектов. Несмотря на то, что часто CORBA рассматривается как промежуточное программное обеспечение, его главная цель состояла, прежде всего в управлении интерфейсами между объектами. Поэтому главное внимание было уделено клиент-серверным взаимодействиям в пределах относительно статической ресурсной среды. Акцент на гибкости управления интерфейсами ведет к введению дополнительных уровней программного обеспечения, что в свою очередь приводит к ухудшению производительности. Для улучшения производительности таких приложений была предложена разработка High-Performance CORBA [17], которая представляет собой попытку улучшить производительность CORBA не только за счет улучшения эффективности ORB, но также и за счет агрегирования обработки в кластерах или параллельных компьютерах. Часть этой работы включает поддержку параллельных объектов, которые ”понимают”, как связываться в распределенной среде.

Предпринимаются также усилия, чтобы сделать сервисы CORBA непосредственно доступными для Грид-вычислений. Для этого в проекте CoG Kit ('Commodity Grids) [18] осуществляется путем создания интерфейсного программного слоя, который отображает сервисы Globus на интерфейсы API CORBA.

Система Legion, обеспечивает объекты глобально уникальными (и непрозрачными) идентификаторами [19]. Используя такой идентификатор, объект и его элементы, могут быть видны отовсюду. Возможность порождать и манипулировать глобально уникальными идентификаторами требует существенной распределенной инфраструктуры.

Компоненты расширяют объектно-ориентированную парадигму, позволяя объектам управлять интерфейсами, которыми они себя представляют и обнаруживать интерфейсы, представленные другими объектами. Это также позволяет выполнять реализации, которые полностью отделены от определений и версий. От компонентов требуются, чтобы они имели набор известных портов, который включает порт инспекции. Это позволяет одному компоненту опрашивать другой и обнаруживать, какие интерфейсы поддержаны и какие их точные спецификации. Такой подход означает, что компонент должен уметь обеспечивать метаданные относительно своего интерфейса, а также, возможно, своей

функциональности и производительности. Эта возможность также поддерживает повторное использование и композиционность программного обеспечения.

Наиболее известны среди компонентных и компоненто-подобных систем – это COM/DCOM, CORBA 3 Component Model, Enterprise Java Beans и Jini, и Common Component Architecture [20]. Из них модель Common Component Architecture включает определенные характеристики для высокопроизводительных вычислений, такие как коллективные порты и прямые соединения.

2.6.3. Порталы могут рассматриваться как обеспечение Веб-интерфейса для распределенных систем. Обычно порталы влекут за собой трехярусную архитектуру, состоящую из: 1) клиентов (первый уровень); 2) брокеров или серверов (средний уровень); 3) репозитариев объектов (вычислительные серверы, базы данных или любые другие ресурсы или сервисы). Используя данную общую архитектуру, можно строить порталы, которые поддерживают широкое многообразие прикладных областей, например, порталы областей науки, вычислительные порталы, порталы покупок, порталы образования и так далее. Однако, чтобы сделать это эффективно, требуется набор портальных инструментальных средств, которые могут быть специально подготовлены для каждой прикладной области.

Известными примерами здесь является инструментарий порталов Грид GridPort [21], состоящий из двух частей: 1) клиентских инструментальных средств интерфейсов;
2) модуля сервисов Веб-портала. Клиентские инструментальные средства интерфейсов позволяют настраивать разработку интерфейса портала и не требуют от пользователя знаний технологии портала. Модуль сервисов Веб-портала выполняется на коммерческих Веб-серверах и обеспечивает аутентифицированное использование ресурсов Грид.

Следующий важный пример – научный портал XCAT [22]. В этом проекте порталы разработаны с использованием *сборника* типичных Веб-страниц, текста, графики, форм и выполнимых сценариев. Сборники имеют интерактивный редактор сценариев/форм, основанный на языке JPython, который позволяет иметь доступ к другим наборам инструментов, таких как реализации CoG Kit и XCAT для архитектуры CCA. Такая связь порталов и компонентов облегчает работу пользователя и динамическую композицию кодов и сервисов Грид.

2.7. Модели Веб-сервисов

2.7.1. OGSA. Развитие Грид-технологий привело к созданию открытой архитектуры сервисов Грид (OGSA) [23], в которой Грид обеспечивает расширяемый набор сервисов для осуществления взаимодействий виртуальных организаций. OGSA определяет однородную семантику сервисов (так называемые Грид-сервисы), основанную на понятиях и технологиях, разработанных в сообществах Грид- и Веб-систем, они же определяют стандартные механизмы для создания, именования и обнаружения временных экземпляров Грид-сервисов, обеспечивают прозрачность размещения и многопротокольные связывания для экземпляров сервисов и поддерживают интеграцию с находящимися в основе средствами платформ.

OGSA имеет цель определять общую модель ресурсов, которая является абстрактным представлением как реальных ресурсов, таких как узлы, процессы, диски, файловые системы, так и логических ресурсов. Она обеспечивает некоторые общие операции и поддерживает множество лежащих в основе моделей ресурсов, представляющих ресурсы как экземпляры сервиса. Абстракции OGSA и сервисы обеспечивают стандартные блоки, которые разработчики могут использовать для реализации множества высокоуровневых Грид-сервисов, но сервисы OGSA являются, в принципе, независимыми от языка программирования и модели программирования. OGSA стремится определять семантику экземпляра Грид-сервиса: как он создается, как называется, как определена его продолжительность жизни, как с ним связаться и так далее.

OGSA не рассматривает проблемы реализации программной модели, языка программирования, инструментальных средств или среды выполнения. Определение и реализация OGSA окажет существенное влияние на модели программирования Грид, потому, что они могут использоваться для поддержки сервисов OGSA, а высокоуровневые модели могут включать механизмы программирования, необходимые для приложений Grid. Проект Globus выполняется как разработка с открытым кодом по реализации OGSA путем развития Globus Toolkit до совместимой с OGSA реализации этого инструментария.

2.8. Координационные модели, их цель – обеспечение способов интегрирования разнородных компонентов путем их связывания с помощью интерфейсов для формирования единого приложения, которое сможет выполняться на параллельной или распределенной системе [24]. Координационные модели могут использоваться для разделения вычислительных аспектов распределенного или параллельного приложения и аспектов сотрудничества, позволяя выполнять их отдельную разработку, но также осуществляя и возможное слияние из этих двух стадий разработки.

Понятие координации близко связано с понятием неоднородности. Так как интерфейс координационной модели рассматривается отдельно от вычислительной, реальные языки программирования, используемые для записи вычислительной части алгоритмов не играют важной роли в установке режимов координационных механизмов. Кроме того, так как компонент координации предлагает однородный путь для межпроцессорной связи и абстрагируется от деталей архитектуры, координационные модели поощряют использование разнородных ансамблей компьютеров.

Координационный язык предлагает композиционный механизм создания и накладывает некоторые ограничения на формы параллелизма и на используемые механизмах связи с помощью интерфейса. Координационные языки для Грид-вычислений вообще являются ортогональными к последовательному или параллельному коду, они обеспечивают модель для композиции программ и должны реализовывать межмодульную оптимизацию, которая принимает во внимание архитектурные свойства для обеспечения эффективного выполнения приложений на Грид-системе. Недавние исследовательские работы в этой области используют XML или скелетонные модели для программирования Грид [24]. Другая потенциальная прикладная область для инструментальных средств координации Грид – это средства потоков работ (workflow) [25], модели управления работы на предприятии, в которой модули (блоки) работы передаются между пунктами обработки на основе процедурных правил.

3. Перспективные методы программного обеспечения Грид

Вышерассмотренные модели программирования находят широкое применение для Грид-вычислений, однако они не могут быть достаточными в достижении высокой производительности и гибкости. Они требуют значительных усилий в ручном программировании и использовании низкоуровневых сервисов Грид. Цель перспективных методов создания программного обеспечения Грид – это автоматизация и упрощение данных методов, насколько это возможно. Тесно связанные приложения, которые обычно предназначены для выполнения на архитектурах мультипроцессорных машин с общей памятью, не могут быть эффективными в системах Грид. Для этого существует ряд традиционных методов обеспечения производительности, которые могут быть применены для программ на Грид:

• *перекрытие вычислений и обменов данными* – это требует знания плана обменов в Грид, чтобы в моменты обменов данными можно было осуществлять вычисления на не зависящей от обменов памяти процессов;

• *репликация данных (теневые массивы)* – использование дополнительной памяти теневых массивов позволяет разрывать связи операторов программ по дан-

ным и таким образом нивелировать высокую латентность обменов с медленными уровнями памяти;

• *агрегация коммуникаций* – эффективность обменов может быть улучшена путем объединения многих малых сообщений в меньшее количество больших сообщений;

• *сжатие данных* – уменьшение объема обменов за счет сжатия данных;

• *настройка протокола* – настройкой параметров протокола, таких как размер окна TCP, приложения могут повысить производительность передачи данных.

С данными методами достигнуто масштабирование 88 % и 63 % соответственно по размеру приложений и количеству использованных ресурсов. Однако остается проблемой – как эти методы могут быть встроены в модели программирования и инструментальные средства так, чтобы они могли прозрачно применяться в приложениях Грид.

3.1. Управляемые данными (data flow) методы. Помимо улучшения производительности передачи данных, традиционные методы также ориентированы на обеспечение условия слабосвязанности выполнения кода MPI. Как можно реализовать более асинхронный слабосвязанный режим выполнения в модели программирования? Управляемые данными методы программирования могут облегчить данную задачу. При этом такие модели могут страдать от чрезмерных проверок соответствия операндов и накладных расходов на планирование, в ограниченных формах крупноблочного распараллеливания – эти методы могут давать существенные выгоды. Потоки работ (workflow) и программирование потоков (stream programming) – хорошие примеры для такой модели. Как показано на опыте каркаса DataCutter [26], программирование потоков может использоваться для управления доступом к большим хранилищам данных и привязкой обработки и обменов данными в распределенной среде. К наборам данных, которые являются слишком большими для их легкого копирования или перемещения, можно обращаться через сквозные фильтры, которые могут выполнять пространственную фильтрацию, цифровое прореживание (уменьшение размера изображения), угол поворота или кэширование копии изображения.

3.2. Распределенные методы. Еще один метод – распределение обработки по данным. В глубокой иерархии латентностей Грид-сети использование синхронных по данным языков параллельного программирования очевидно является неадекватным. Если синхронизация и связь параллельных процессов требуется не слишком часто, распределенные методы могут достигать очень высоких показателей совокупной производительности локальной обработки и пропускной способности обмена данными. Цель состоит в том, чтобы скрыть всю латентность памяти на фоне вычислений, но в совершенно другом масштабе. Например, архитектура Grid Datafarm разработана в [27] для использования локальности доступа путем планирования программ на крупномасштабном распределенном дисковом хранилище, которые осуществляли обработку близко к памяти хранения.

3.3. Расширенные сервисы связи. Модели программирования могут зависеть от поддержки со стороны инфраструктуры, и расширенные сервисы связи являются частью этой инфраструктуры. То, что под этим понимается, является по существу некоторым типом семантики связи, отличной от простой, достоверной передачи данных (unicast) от точки A до точки B или групповой передачи (multicast) данных ”один – всем”. Следовательно, что составляет расширенную службу связи, может быть определено широко и может мотивироваться различными факторами.

Роль и использование топологии сети в Грид-вычислениях будет возрастать, так как производительность Грид-сети в целом все более будет зависеть от задержек при передаче данных. Ожидается, что в последующие несколько лет магистральные каналы станут производительнее и реактивнее из-за ограничений латентности. Поэтому для управления производитель-

ностью средства программирования типа MPI должны стать ”знающими топологию”. Они смогут минимизировать трафик обмена данными при групповых операциях на медленных каналах. Вместо сложности передач *O(nlogn)* в зависимости от диаметра сети *n*, как это обычно бывает, ориентированные на знание топологи групповые операции могут давать линейную зависимость этой сложности от среднего диаметра сети. Другой мотивирующий фактор для расширенных сервисов связи – это необходимость в существенно различных свойствах связи. Речь идет о контенто-ориентированной (основанной на содержании) и политико-ориентированной маршрутизации (основанной на политиках, составленных из регулируемых правил). Традиционно реализация групповых операций для распространения информации основана на физической топологии сети. Контенто-ориентированная маршрутизация дает возможность приложениям управлять планированием сообщений, маршрутизацией и фильтрацией больше на основе динамических требований приложений для данной группы пользователей, чем из-за необходимости всегда использовать связь точка-точка. Естественно, это потребует достаточного понимания топологии на некотором уровне.

Таким образом, расширенные службы сервисы могут быть классифицированы по нескольким широким категориям.

• *Обогащенная семантика связи*: в противовес изменениям основных алгоритмов маршрутизации, большинство операций связи может просто быть оснащено дополнительными функциональными возможностями. Известные примеры такого подхода включают кэширование, фильтрацию, сжатие, шифрование, качество обслуживания и др.

• *Групповые операции*: приложения могут требовать синхронных режимов выполнения операций, таких как барьерные, префиксные или/и операции редукции. Эти операции обычно реализуются на коммуникационной топологии, в основе которой лежат обмены точка-точка.

• *Контенто-ориентированная и политико-ориентированная маршрутизация*: это существенно различные парадигмы, первая из которых позволяет определять маршрутизацию на основе определения параметров при передаче данных, что предоставляет возможности типа опубликования/подписки для управления интересами; вторая включает такие примеры, как распределение сообщений в соответствии с требованиями управления качеством (QoS).

• *Масштаб связи*: некоторые сервисы связи могут быть слишком дороги для реализации, особенно в большом масштабе, поэтому если приложения могут определить свои собственные масштабы для сервисов, тогда они могли бы свести размеры приложений к минимуму.

3.4. Безопасность и отказоустойчивость. Приложения Грид должны уметь проводить удостоверение подлинности (аутентификацию), проверку полномочий (авторизацию), проверку целостности и секретность. В контексте модели программирования это влечет за собой дополнительные уточнения. Базисная двухточечная защита может быть выполнена путем интеграции механизма защиты с конструкциями программирования. Пример этому интеграция SOAP с GSI [28]. В широком контексте, однако, такие вызовы RMI или RPC могут существовать в дереве вызовов, а поддержка безопасности вдоль дерева вызовов требует использования понятия делегирования доверия. Принятие и проверка сертификатов на RPC также представляет собой накладные расходы, которые должны быть сбалансированы с количеством работы, представленной RPC. Накладные расходы по защите могут управляться путем установления безопасного доверительного домена.

Вопросы надежности и отказоустойчивости в моделях и инструментальных средствах для программирования Грид в значительной степени еще не исследованы, кроме простых механизмов контрольных точек и рестарта. Проблема состоит в том, как сделать модели и инструментальные средства программирования Грид по существу более достоверными и отказоустойчивыми. Ясно, что существует

различие между надежностью и отказоустойчивостью приложения, этими же характеристиками в модели или инструменте программирования, или же в самой инфраструктуре Грид. Чтобы обеспечить надежность и отказоустойчивость на верхнем уровне, они должны быть доступными на всех нижних уровнях.

Дальнейшее различие может быть сделано между обнаружением неисправностей, извещением о неисправности и устранением неисправности. В распределенной среде Грид важна способность обнаружения неисправности. Распространение извещения о неисправности на соответствующие узлы также является критическим. В следствии эти узлы должны быть способными принять меры для восстановления или ограничения последствий неисправности. Эти возможности требуют, чтобы в модели программирования Грид и инструментальные средства были интегрированы модели событий.

3.5. Метамодели программ и Грид-системы времени выполнения. Еще одна серьезная проблема для моделей программирования Грид – понятие метамоделей программ и их использования ”знающими-Грид” системами времени выполнения. Независимо от того, как в конечном счете развертываются Грид-системы, они будут состоять из компонентов и сервисов, которые являются или постоянными (persistent), или могут быть установлены (instantiated). Некоторые из этих компонентов и сервисов уже широко используются. Поэтому многие приложения можно построить, частично или в целом, с помощью композиции компонентов и сервисов. Как такая композиция может быть выполнена автоматически, и при этом удовлетворялись такие характеристики, как производительность, безопасность и отказоустойчивость, может быть обеспечено только с использованием метамоделей, которые определяют характеристики компонента и их свойства. Метамодели могут быть составлены вручную, но их можно также произвести автоматически.

На компиляторы и инструментальные средства композиции можно возложить ответственность за генерирование метамоделей и их использование для идентификации композиций и их приведение к правильному (допустимому) виду. В контексте Грид термин ”правильный” может означать намного больше, чем то, являются ли параметры интерфейса совместимыми. Например, правильность может означать сохранение характеристик производительности, безопасности и отказоустойчивости. На основе высокоуровневого описания программы (например, как в сценарном языке порталов), компилятор может отображать семантику высшего уровня на компоненты низшего уровня и сервисы. Это дает возможность определить цель компилятора Грид не в традиционном смысле генерирования машинных команд, а в смысле набора всюду доступных сервисов.

Примером такой работы в этой области можно назвать разработку инструментальных программных средств для приложений Грид (GrADS) [29]. Подход GrADS основан на: 1) системе подготовки программ; 2) системе выполнения программ. Система подготовки программ воспринимает на вход программу пользователя вместе с повторно используемыми компонентами и библиотеками и создает конфигурируемую объектную программу. Объекты этой программы аннотируются требованиями ресурсов и прогнозируемой производительностью. Среда выполнения использует данную информацию для выбора соответствующих ресурсов для выполнения, и также для контроля производительности приложения согласно с ”контрактом”.

Заключение

Какие модели программирования Грид станут успешными, во многом будут определяться такими аспектами, как мобильность, интероперабельность, адаптивность, а также способность поддерживать обнаружение, безопасность и отказоустойчивость при сохранении производительности. Модели и инструментальные средства программирования в значительной степени зависят от доминирующей парадигмы программирования. Имеется также различие

между возможностями общей инфраструктуры и возможностями моделей программирования и инструментальных средств, построенных на основе данных инфраструктур.

В отношении общей инфраструктуры, огромный интерес к развитию Вебсервисов означает, что по всей вероятности открытая архитектура сервисов Грид будет разрабатываться и далее. Что касается моделей и инструментальных средств программирования, то очевидно будет продолжена разработка стандарта MPI и его потенциальных расширений с минимальными изменениями API. Другие модели и инструментальные средства, такие как каркасы, предоставляющие богатый набор сервисов на основе общих Гридсервисов типа Cactus и XCAT, также имеют шансы на дальнейшее развитие.

Эволюция вычислительных платформ от одиночных компьютеров до параллельных вычислительных систем и Грид-сетей должна ускорить и соответствующее развитие методов программирования для решения вычислительных проблем. Программисты всегда адаптируют свои программные коды и стиль программирования к соответствующим инфраструктурам. Они будут стремиться делать свои коды более слабосвязанными. Проблемно-ориентированные архитектуры будут разрабатываться наиболее подходящим способом для среды Грид.

1. *Системы* Grid-вычислений – перспектива для научных исследований / А.Е. Дорошенко, О.В. Алистратов, Ю.М. Тырчак, и др. // Проблемы программирования.– 2005.– № 1.– С. 14–38.
2. *Lee C., Talia D.* Grid programming models: current tools, issues and directions // in "Grid Computing — Making the Global Infrastructure a Reality"(F. Berman, A. Hey and G. Fox, eds.), Wiley, 2003. – P. 555 – 578.
3. *Foster I., Kesselman C. and Tuecke S.* The anatomy of the grid: enabling scalable virtual organizations. International J. of Supercomputer Applications.– 2001.– N 15.– P. 200–222, www.globus.org/research/ papers/anatomy.pdf .
4. *Freeman E., Hupfer S. and Arnold K.* JavaSpaces: Principles, Patterns, and Practice. Reading, MA: Addison-Wesley, 1999, http://java.sun.com/docs/books/ jini/javaspaces/ .
5. *Saelee D. and Rana O. F.* Implementing services in a computational Grid with Jini and Globus. First EuroGlobus Workshop, 2001, www.euroglobus.unile.it .
6. *Developing* Grid Services with Jini and JXTA / O.F. Rana, V.S. Getov, E. Sharakan, S. Newhouse, and R. Allan.– 2004, http://westminsterresearch.wmin.ac. uk/847/ .
7. *Message* Passing Interface Forum, MPI-2: Extensions to the Message Passing Interface, July, 1997, www.mpi-forum.org/ .
8. *Foster I. and Karonis N.T.* A grid-enabled MPI: message passing in heterogeneous distributed computing systems. Supercomputing. IEEE, November 1998, www.supercomp.org/sc98 .
9. *Nakada H., Matsuoka S., Seymour K.* Overview of GridRPC: A Remote Procedure Call API for Grid Computing. 3rd International Workshop on Grid Computing, LNCS, November 2002. – Vol. 2536.– P. 274 – 278.
10. *Arnold D. et al.* Users' Guide to NetSolve V1.4, Innovative Computing Laboratory, Department of Computer Science, University of Tennessee, Knoxville, TN, 2001, http://www.netlib.org/netsolve/UG/ .
11. *Foster I., Kesselman C., Nick J. and Tuecke S.* The Physiology of the Grid: Open Grid Services Architecture for Distributed Systems Integration, presented at GGF4, Feb. 2002 http://www.globus.org/research/ papers/ogsa.pdf.
12. *Getov V., Laszewski G., Philippsen M. and Foster I.* Multi-paradigm communi cations in java for grid computing // Commun. ACM. – 2001. – Vol.44, N 10. – P. 118 – 125.
13. *Carpenter B.* MPJ: MPI-like message-passing for Java // Concurrency: Practice and Experience. – 2000. – Vol. 12, N 11. – P. 1019 – 1038.
14. *Gong L.* Peer-to-Peer Networks in Action // IEEE Internet Computing. – 2002. – Vol. 6, N1. – http://dsonline.computer.org/ 0201/ic/w102gei.htm .
15. *Gong L.* JXTA: A Network Programming Environment // IEEE Internet Computing, 2001. – Vol. 5, N 3. – P. 88 – 95.

16. *Cactus*, www.CactusCode.org , 2000.
17. *Jacobsen H.A.* High Performance CORBA Working Group, www.omg.org/ realtime/workinggroups/highperformancecorba.html .
18. *Verma S., Gawor J., Laszewski G. and Parashar M.* A CORBA commodity grid kit. Grid 2001.– November 2001.– P. 2–13.
19. *Grimshaw A., Wulf W.* "The Legion Vision of a Worldwide Virtual Computer". Communications of the ACM, January 1997. – Vol. 40(1).– P. 39 – 45.
20. *Gannon D.* CCAT: The Common Component Architecture Toolkit, www.extreme.indiana.edu/ccat .
21. *The* Grid Portal Toolkit, www.gridport.npaci.edu .
22. *Krishnan S.* The XCAT science portal. Supercomputing, November 2001, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1592792 .
23. *Foster I., Kesselman C., Nick J. M. and Tuecke S.* Grid services for distributed system integration. IEEE Computer. – June 2002. – P. 37 – 46.
24. *Furmento N., Mayer A., McGough S., Newhouse S., Field T. and Darlington J.* An integrated grid environment for component applications. Second International Workshop on Grid Computing (Grid 2001), LNCS. – 2001. – Vol. 2242. – P. 26 – 37.
25. *Fischer L.* The Workflow Handbook 2003. Lighthouse Point, FL: Future Strategies, Inc. – 2003, http://www.amazon.com/Workflow-Handbook-2003-Layna-Fischer/dp/0970350929
26. *Beynon M., Kurc T., Sussman A. and Saltz J.* Optimizing execution of component-based applications using group instances // IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, May 2001. – P. 56 – 63.
27. *Tatebe O., Morita Y., Matsuoka S., Soda N., and Sekiguchi S.* Grid datafarm archi tecture for petascale data intensive computing // Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002).– 2002.– P. 102 – 110.
28. *ActiveGrid*, http://www.activegrid.com/ .
29. *Дорошенко А.Е., Рухлис К.А.* Средства оптимизации Grid-вычислений на основе globus toolkit // Проблемы программирования.– 2006.– № 2–3.– С. 156 – 164.

Получено 05.04.2007

Об авторах:

Дорошенко Анатолий Ефимович,
доктор физико-математических наук,
профессор, заведующий отделом теории компьютерных вычислений,
e-mail: dor@isofts.kiev.ua,

Розенблат Александр Петрович,
аспирант Института программных систем НАН Украины,

Рухлис Константин Александрович,
младший научный сотрудник Института программных систем НАН Украины,

Тырчак Юрий Марьянович,
аспирант Института программных систем НАН Украины.

Место работы авторов:

Институт программных систем
НАН Украины,
03680, Киев-187,
проспект Академика Глушкова, 40.
тел. (044) 526 1538