# DESKTOP GRID ДЛЯ ПАРАЛЛЕЛЬНЫХ ЗАДАЧ С ПРОИЗВОЛЬНЫМ КОММУНИКАЦИОНЫМ ПРОФИЛЕМ[1]

**Ю.А. Жолудев**[1]

[1] *Факультет Вычислительной математики и кибернетики Московского Государственного Университета имени М.В. Ломоносова*

*119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова*

e-mail:diemenator@gmail.com

**Аннотация**

Целью данной работы является создание инструментария для развертывания распределенных вычислительных сред, позволяющей эффективно и надежно решать большие задачи, допускающие обмены данными между любыми параллельно выполняющимися процессами. В рамках данной статьи представляются результаты первого этапа разработки, основной задачей которого является создание прототипной реализации для надежного решения параллельных задач в рамках изолированной физической сети.

Введение данной работы содержит краткий обзор технологий распределенных вычислительных сред, предлагающих различные методы решения актуальных для традиционных клиент-серверных архитектур проблем.

В основной части данной работы рассматривается архитектура разрабатываемого программного инструментария. Подробно рассматриваются ключевые механизмы обеспечения надежности работы распределенной вычислительной среды для решения параллельных задач и поддержки привычной модели программирования. Также описываются сценарии работы вычислительной среды таких механизмов для задач в рамках традиционной модели параллельного программирования.

В заключительной части работы приводятся технические детали реализации рассмотренных подходов и механизмов. Рассматриваются дальнейшие пути развития прототипа до полноценного легко используемого инструментария.

## Введение

Способы распределенного решения больших задач не менялись уже очень давно: с момента появления Distributed.net[1], SETI@home[2], и X-Com[3] прошло уже более 10 лет. В этих проектах был заложен основной принцип объединения распределенных вычислительных ресурсов. Организация распределенной вычислительной среды требует выделить центральный инфраструктурный компонент, задачей которого является координация участвующих в расчете узлов. В ответственности этого компонента входит разбиение большой задачи на множество независимых подзадач для включенных в расчет компьютеров и сборка результатов решения этих задач.

1 Работа выполняется при поддержке гранта Президента Российской Федерации для молодых российских ученых-кандидатов наук МК-5104.2011.9.

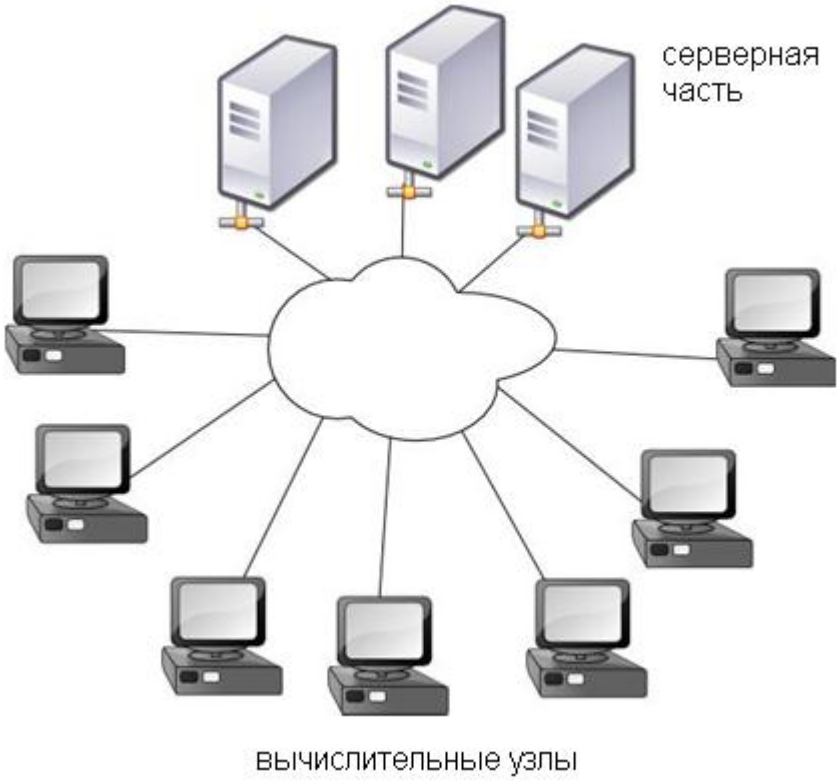


Рис. 1. Традиционная организация распределенных вычислительных сред.

Устоявшаяся клиент-серверная архитектура (рис.1) распределенных вычислительных сред позволяет объединить в одном расчете десятки и сотни тысяч компьютеров с минимальными затратами на развертывание необходимого программного обеспечения. Это также помогает избежать множества проблем, связанных с ненадежностью каналов связи и динамичностью состава вычислительных узлов, участвующих в решении задачи. Однако за эти и другие достоинства традиционных распределенных вычислительных сред приходится платить довольно высокую цену: все задачи для этих сред должны входить в класс Embarrassingly Parallel (EP), т.е. исходную задачу требуется представить в виде множества независимых подзадач, исключая взаимодействие между частями задачи. Кроме того, рост числа вычислительных узлов вычислительной среды ведет к другой проблеме: увеличивающаяся вместе с количеством узлов нагрузка на серверную часть среды приводит к перегрузке каналов связи и серверного оборудования, понижая эффективность распределенного расчета.

Альтернативный способ объединения компьютеров для организации распределенных заключается в отказе от клиент-серверной архитектуры и использовании технологий, подразумевающих децентрализованное взаимодействие, а именно технологий одноранговых или peer-to-peer сетей (p2p). Одноранговые сети обычно реализуются в виде виртуальных сетей, строящихся поверх неоднородных физических сетей с помощью специального программного обеспечения. В таких сетях все узлы равноправны: все узлы сети могут как открывать, так и принимать соединения от других узлов, при этом выход из строя какого-либо из узлов не приводит к полному выходу из строя всей сети (рис.2). Каждый узел содержит информацию о подмножестве подключенных к сети узлах и физических маршрутах, необходимых для передачи данных между ними, реализуя таким образом

распределенное динамическое хранилище информации, которое используется для эффективной передачи данных между узлами. При этом способ организации передачи данных p2p-сети поддерживает динамичность в составе узлов и облегчают нагрузку на физические каналы связи, способствуя масштабируемости таких сетей.

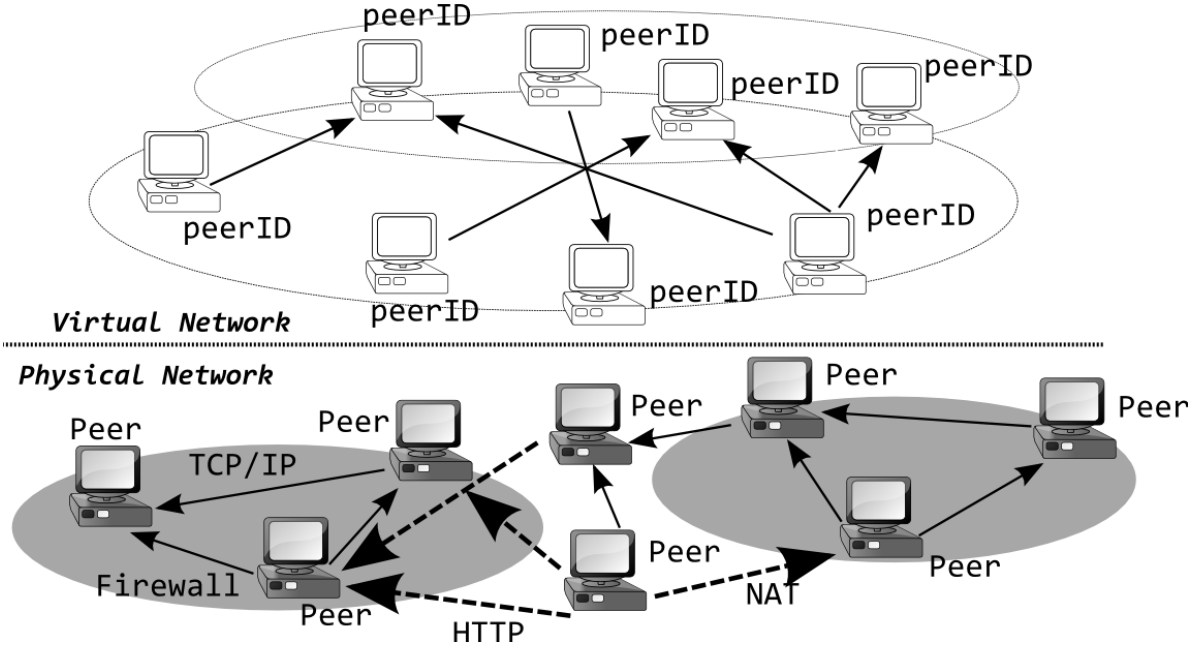


Рис. 2. Организация p2p-сетей.

Несмотря на кажущуюся простоту, масштабное применение p2p-сетей имеет важные особенности.

Во-первых, структура физической сети влияет на эффективность работы p2p-сети: при наличии в сети искусственных барьеров (NAT, Firewall'ы) появляются накладные расходы, связанные с построением сложных обходных маршрутов для передачи данных и, как следствие, многократного увеличения латентности.

Во-вторых, ни один узел p2p-сети не может одновременно получить актуальную информацию обо всех остальных подключенных узлах, т.к. подобная функциональность требует постоянной пересылки большого количества служебной информации между всеми узлами сети, что при увеличении масштабов приводит к переполнению пропускной способности каналов связи.

Несмотря на эти ограничения, p2p-технологии нашли свое применение в области распределенных вычислений. Ниже приводится несколько проектов вычислительных сред, так или иначе использующих достоинства p2p-сетей.

Первым таким проектом стал P3 (Personal Power Plant)[4], реализующий программную модель MPI на Java для фиксированного набора распределенных узлов, но не поддерживающий свойственную распределенным средам динамичность.

Тем не менее, идеи P3 нашли свое развитие в проекте P2P-MPI[5], где был реализован механизм репликации параллельных процессов, позволяющий производить расчеты на вычислительных узлах, характер участия которых в расчете нестабилен. Ограничения P2P-MPI заключаются в неполном равноправии вычислительных узлов: для запуска параллельного приложения в среде требуется узел, играющий роль координатора, в ответственности которого входит хранение и обновление информации обо всех узлах сети и

распределение процессов параллельной задачи на подключенные компьютеры. Кроме того, уровень передачи сообщений в реализации P2P-MPI реализован в рудиментарном варианте, не поддерживающим коммуникации в физических сетях с искусственными барьерами.

Другой вариант использования p2p-технологий был предложен в проекте Cohesion Orbweb[6], в котором вычислительная среда реализуется в виде распределенного динамического хранилища задач, подходящего для ряда приложений из класса Irregularly Structured (IS). Каждый узел среды при решении основной задачи может получить из хранилища задачу, решить её каким-либо образом или разбить её на множество подзадач и отправить результаты работы обратно в хранилище. В частности, такая схемы работы показала высокую эффективность при решении задачи выполнимости булевых формул: каждая подзадача занималась тем, что фиксировала значения очередной переменной, соответственно упрощала исходную формулу, порождая 2 новых подзадачи с новыми формулами [7].

Третий вариант использования p2p-технологий заключался в использовании повышенных возможностей масштабирования и толерантности p2p-сети к отключению узлов для решения задач из класса EP [8].

Разрабатываемой в рамках данной работы инструментарий x-p2p [9] призван объединить достоинства рассмотренных p2p-систем и предоставить легкий способ развертывания вычислительных сред для решения параллельных задач произвольным коммуникационным профилем с использованием привычных моделей программирования.

**Требования**

Определим основной перечень требований к разрабатываемой системе:

- Независимость среды от топологии физической сети и её искусственных ограничений;
- Независимость среды от программно-аппаратной платформы участвующих в среде узлов;
- Параллельная задача может быть поставлена на выполнение с любого узла среды;
- Параллельные процессы, выполняющиеся на разных узлах среды, должны иметь
- возможность обмениваться сообщениями;
- Среда должна поддерживать одновременную работу нескольких параллельных задач;
- Результаты решения задач должны быть доступны с любого узла среды;
- Состав узлов, участвующих в решении задачи, может меняться в ходе расчета, при этом решение задачи не должно прерываться.

Зафиксировав требования к самой среде, необходимо определить модель программирования. Традиционная модель параллельного программирования подразумевает, что на нескольких вычислителях параллельно выполняются процессы, которые посылают друг другу сообщения и производят вычисления. Программируя на MPI, мы всегда предполагаем, что если процесс A выполнил операцию Recv, то обязательно должен быть процесс B, который в какой-то момент времени посылает процессу A сообщение и наоборот. В противном случае коммуникационные операции утрачивают смысл и скорее всего программа написана неверно.

Теперь, учитывая возможность отключения вычислительных узлов в ходе расчета, рассмотрим случай, когда узел процесса B прекратил свою работу, не выполнив ожидаемой от процесса A коммуникационной операции. Тогда необходимо обеспечить, чтобы процесс B возобновил свою работу на каком-либо другом узле и выполнил требуемые операции. Для этого необходимо сохранять состояние процесса в ходе работы, чтобы была возможность передавать состояние процесса между узлами и дублировать процесс на других узлах. Поскольку постоянно сохранять и передавать состояние процессов между узлами означает неоправданные накладные расходы, введем понятие контрольной точки: момент в жизни процесса, который допускает сохранение миграцию процесса на другие узлы. Так как корректное выполнение процесса напрямую зависит от внешних зависимостей (это входящие сообщения), то будем считать, что в контрольных точках все запрошенные коммуникационные операции должны быть завершены.

Исходя из данных соображений, формально определим модель программирования следующим образом:

- Параллельная задача — набор взаимодействующих параллельных процессов;
- Каждый процесс может передавать сообщения любому другому процессу путем передачи сообщений;
- Исполняемый код всех процессов разбивается контрольными точками на фрагменты;
- Коммуникационные операции, запрошенные в ходе фрагмента, могут считаться выполненными по достижению из контрольной точки;
- Процессы задачи не имеют информации о топологии и характеристиках физической сети; адресация в рамках задачи основана на нумерации процессов.

Понятие контрольных точек в данной модели позволяет выполнить и другие требования к среде. Решение задачи может начинаться с одного узла, независимо от количества процессов в задаче: процессы могут выполняться по очереди по фрагментам, сохраняя отправленные сообщения вместе со своим состоянием в файлы и получая необходимые сообщения из других сохраненных процессов. В дальнейшем, при включении нового узла в расчет, часть процессов может мигрировать и продолжить свое выполнение на новом узле.

Исходя из этого, опишем основной цикл работы вычислительного узла:

1. Получить сохраненный процесс (локально или с другого узла)
2. Получить входящие сообщения для процесса
3. Восстановить работу процесса с контрольной точки и продолжить выполнение до следующей контрольной точки; при этом сохранять локально и одновременно отправлять исходящие сообщения текущего фрагмента другим процессам
4. Сохранить состояние процесса

Сохраненные процессы могут быть скопированы на другие узлы в качестве реплик, с помощью которых можно восстановить состояние параллельной задачи в случае частичного отключения узлов из расчета. Для этого часть контрольных точек процессов должны иметь глобальный смысл: если доступно состояние процесса в глобальной контрольной точке K, то

для корректного возобновления работы параллельной задачи в этой точке необходимо, чтобы состояния остальных процессов в соответствующих K контрольных точках были также доступны. Тогда глобальные контрольные точки могут быть пронумерованы и состояния процессов в этих точках могут разделяться между узлами в фоновом режиме.

Например, в итерационных задачах глобальные контрольные точки могут быть определены в конце каждой итераций (или итераций с кратным определенному числу номером), а локальные – только в местах взаимодействия между процессами в рамках одной итерации.

**Архитектура**

Архитектура приложения вычислительного узла может быть представлена в виде 3 слоев:

1. Слой p2p-сети:
   1.1. Поиск узлов и маршрутов
   1.2. Поиск и публикация объектов
   1.3. Абстракция групп узлов
   1.4. Передачи данных
2. Инфраструктурный слой:
   2.1. Поиск и публикация сохраненных процессов и сообщений
   2.2. Передача сообщений
   2.3. Передача файлов
3. Слой взаимодействия с процессом
   3.1. Интерфейс передачи сообщений
   3.2. Интерфейс публикации контрольных точек
   3.3. Интерфейс публикации файлов

Слой p2p отвечает за включение и идентификацию вычислительных узлов в p2p-сеть. На этом слое происходит фоновый обмен и распределенное хранение состояния узлов в вычислительный сети, что позволяет выполнять распределенный поиск любых объектов, включая маршруты к определенным узлам, группы узлов и открытые виртуальные каналы передачи данных.

Инфраструктурный слой отвечает за адресацию и передачу сообщений процессов соответствующим узлам, а также за хранение, разделение и публикацию файлов процессов, их состояний и сохраненных сообщений в рамках группы узлов, участвующих в одной задаче.

Слой взаимодействия с процессом предоставляет работающему процессу функциональность нижних слоев архитектуры, скрывая детали, касающиеся хранения и передачи сообщений и контрольных точек и способов разделения файлов в группах узлов. Именно этот слой доступен в качестве API для прикладной задачи.

**Реализация**

Инструментарий разрабатывается на кросс-платформенном языке Java. Технологической основой для создания и организации работы p2p-сети является набор

реализации p2p-протоколов JXTA JXSE[10] 2.6. Данная реализация лежит в основе нижнего слоя архитектуры приложения узла и предоставляет базовую функциональность p2p-сети инфраструктурному слою архитектуры. В частности, протокол JXTA определяет следующие объекты:

- узлы (peers)
- маршруты (routes)
- группы узлов (peer groups)
- разделяемые данные (content)
- каналы передачи сообщений (pipes and sockets)
- сервисы

Каждый физический узел JXTA-сети определяет набор поддерживаемых (с учетом административных ограничений физической сети) транспортных или прикладных протоколов (http, https, tcp, udp) и возможность принимать входящие соединения. В рамках JXTA такой узел включается в сеть путем прямого подключения к какому-либо узлу сети, после чего узел создает и периодически публикует свои объекты (сервисы, разделяемые данные и т.д.) и получает необходимые для запрошенных с верхнего уровня архитектуры маршруты к другим узлам.

Информация об объектах узла доступна другим узлам с помощью механизма публикаций (Advertisements), который реализует распределенное динамическое хранилище метаданных для каждого объекта в сети. Каждая публикация имеет свое время жизни (обычно составляет несколько минут), что используется при хранении полученных с других узлов публикаций в локальном временном хранилище. Просроченные публикации удаляются и описываемые в них объекты считаются несуществующими до тех пор, пока по соответствующему запросу не будет найдена новая публикация.

Механизм публикации интенсивно используется как в самом слое сети-p2p, так и в инфраструктурном слое для поиска сохраненных контрольных точек процессов и их расположения на узлах.

Инфраструктурный слой x-p2p строится поверх сетевого слоя, обращения к которому производят несколько исполняемых потоков:

- сервис публикации группы узлов задачи
- сервис публикации сохраненных состояний процессов и журналов сообщений в группе узлов
- сервис передачи файлов группы узлов
- сервис разрешения состояний процессов (реализует стратегии поиска, хранения и разделения состояний процессов в контрольных точках) для группы узлов

- поток-сервер для прикладного процесса, обеспечивающий работу слоя взаимодействия с процессом задачи
- поток основного цикла вычислительного узла

Сервисы публикации с заданной периодичностью снимают показатели текущих объектов узла и отправляют в сетевой слой архитектуры соответствующие данные, которые становятся доступны в группе узлов (или в рамках всей p2p-сети, если это группа узлов самой прикладной задачи).

Сервис разрешения состояний процессов отвечает за механизм миграции и репликации состояний процессов, а также принимает решение о перезапуске задачи с последней глобальной контрольной точки. Такое решение принимается в случае, когда обнаружен простой выполняющихся на узле процесса (в ожидании входящих сообщений) и актуальная информация (публикация) о каком-либо из других процессов в любой локальной контрольной точке текущего глобального фрагмента хода задачи недоступна. В таком случае сервис выбирает узел, у которого имеется доступное состояние глобальной контрольной точки, после чего на этом узле выполнение текущего процесса откладывается и инициализируется выполнение «потерянного» процесса. Внешние взаимодействия восстанавливаемого процесса воспроизводятся из сохраненных для текущего глобального фрагмента журналов сообщений от других процессов, поэтому перезапуск остальных процессов с контрольной точки не нужен.

Поток-сервер прикладного процесса представляет собой локальный tcp-сервер, предоставляющий интерфейс для уровня прикладного процесса.

Прикладной процесс задачи – это пользовательский Java-код и набор внешних программных библиотек. В данном коде пользователь определяет начальное порождение всех процессов задачи, процесс загрузки внешних библиотек, и собственно прикладной код на API x-p2p. Внешние операции прикладного процесса в API x-p2p производятся посредством обращений к интерфейсному серверу инфраструктурного слоя. Прикладной процесс задачи запускается как отдельный процесс операционной системы в потоке основного цикла вычислительного узла. Хотя прикладной API еще до конца не устоялся, представление о прикладном коде можно получить на рис. 3.

```
package xp2p.job.sample;

import java.io.File;
import xp2p.job.JobBase;

public class MyJob extends JobBase
{
    private long helloMessageId;

    public void main(String[] args) {
        int size = getSize(); // начальное количество процессов задачи
        int rank = getRank(); // номер процесса [0..size-1]
        File resultsFile = new File("results.dat");
        if (rank == 0)  {
            if (wasCheckpoint()) {  // будет пропущено если была достигнута контрольная точка, описанная ниже
                MsgSvc.send("hello", 1); // послать "hello" 1-му процессу
                checkPoint(); // сохранение состояния процесса и блокировка до получения сообщений
            }
            if (wasCheckpoint()) {
                // ...
            }
        }
        if (rank == 1)  {
            if (wasCheckpoint()) {
                helloMessageId = MsgSvc.startRecv(0); // получить сообщение от 0го процесса
                checkPoint(); // сохранение состояния процесса и блокировка до получения результата MsgSvc.startRecv(0);
            }
            if (wasCheckpoint()) {
                String hello = (String) MsgSvc.completeRecv(helloMessageId); // завершение получения сообщения
                // ...
            }
        }
        FileSvc.Publish(resultsFile, "results" + rank + ".dat"); // публикация файла с результатами
    }
}
```

Рис.3. Пример вызовов API x-p2p в коде процесса прикладной задачи

**Заключение**

На текущем этапе разработки x-p2p еще не до конца определены оптимальные стратегии распределения узлов p2p-сети по задачам при работе в многопользовательском режиме и миграции процессов на новые узлы, без чего нельзя поставить состоятельные эксперименты для исследования эффективности разработанных технологий на различных классах задач.

Кроме того, используемый слой p2p-сети JXTA предоставляет мало информации о построенных маршрутах для изучения эффективности передачи данных в неоднородных физических сетях. Реализация такой функциональности и исследование адаптивных к физической структуре сети стратегий распределения процессов по узлам будет являться предметом отдельного исследования.

ЛИТЕРАТУРА


1. SETI@home // [HTML]http://setiathome.berkeley.edu/ (доступ 30.10.2012)
2. Distributed.net // [HTML]http://www.distributed.net (доступ 30.10.2012)
3. Воеводин Вл.В., Жолудев Ю.А., Соболев С.И., Стефанов К.С. Эволюция системы метакомпьютинга X-Com // Вестник Нижегородского государственного университета им. Н.И. Лобачевского. №4. 2009. С 157-164.
4. Kazuyuki Shudo, Yoshio Tanaka and Satoshi Sekiguchi. P3: P2P-based Middleware Enabling Transfer and Aggregation of Computational Resources. Proc. Cluster Computing and Grid 2005 (CCGrid 2005, Fifth Int'l Workshop on Global and Peer-to-Peer Computing), May 9, 2005
5. Stephane Genaud and Choopan Rattanapoka. A Peer-to-Peer Framework for Message Passing Parallel Programs, // Parallel Programming, Models and Applications in Grid and P2P Systems, vol. 17, pages 118--147, Ed. Fatos Xhafa, Advances in Parallel Computing,IOS Press, June 2009.
6. S. Schulz, W. Blochinger, and M. Poths. Orbweb - A Network Substrate for Peer-to-Peer Grid Computing Platforms based on Open Standards. // Journal of Grid Computing, 8(1):77-107, Springer, 2010.
7. S. Schulz and W. Blochinger. Cooperate and Compete! A Hybrid Solving Strategy for Task-Parallel SAT Solving on Peer-to-Peer Desktop Grids. // In Proc. of the International Conference on High Performance Computing & Simulation (HPCS 2010), Workshop on Parallel Satisfiability Solving, Pages 314-323, Caen, France, June 2010, IEEE Computer Society.
8. Marco Ferrante - The JXTA way to Grid: a dead end? // Dottorato in Infromatica, XXII ciclo, 2008.
9. x-p2p computing platform // [HTML]http://code.google.com/p/xp2p/ (доступ 30.10.2012)
10. Gabriel Antoniu, Mathieu Jan and David Noblet. Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. // In Proc. of the first Intl. Conf. on High Performance Computing and Communications (HPCC'05), (3726):429-439, Springer-Verlag, Sorrento, Italy, September 2005.