УДК 681.3:658

Гороховський С. С, Римарчук В. К.

ТЕХНОЛОГІЯ СІТКИ (GRID) І ВИКОРИСТАННЯ АГЕНТНИХ ПЛАТФОРМ ДЛЯ ЗАДАЧ ПЛАНУВАННЯ

Працю присвячено поєднанню двох супертехнологій — Grid, яка пропонує створення нової інфраструктури розподілених обчислень шляхом координованого доступу до різноманітних ресурсів, об'єднаних мережею, і агентної технології, що забезпечує інтелектуальні, автономні й соціальні здатності. Основне завдання Сітки — це координування колекцій ресурсів. Застосування агентних систем для планування задач у Grid дасть змогу розв 'язати дві основні проблеми — масштабованості та адаптивності. Методи й прийоми, що використовуються сьогодні, не забезпечують розв'язання цих складних проблем.

Вступ

Технологія Grid (Сітка), що швидко поширюється , пропонує створення нової і нфраструктури розподілених обчислень шляхом координованого доступу до різноманітних ресурсів, об'єднаних мережею. Основне завдання Сітки - це координування колекцій ресурсів. Застосування агентних систем для планування задач у Grid дасть змогу розв'язати дві основні проблеми - масштабованості та адаптивності. Методи й прийоми, що використовуються сьогодні, не забезпечують розв'язання цих складних проблем. Метою цієї роботи є проект планувальника задач для Grid на основі агентів [1]. Агенти надають ресурси та здійснюють пошук відповідного ресурсу Зроблено спробу контролювати параметри якості, хоча б час виконання задачі, що визначається користувачем. Аналіз зазначених технологій дає змогу знайти спільні точки для їх взаємодії. Архітектура агентів для задач планування має забезпечувати ефективність розв'язання задач, що створюють певний виклик для розробників. Метою керування ресурсами є ефективне планування виконання задач на наявних ресурсах гетерогенного середовища. Координування колекцій ресурсів - це завдання вищого рівня, і більшість процесів (моніторинг, діагностика, планування) вимагає інтелектуальних, автономних, соціальних здатностей — усе це є вже звичними характеристиками агентів. У статті запропоновано підхіддо створення планувальника задач Grid на основі інтелектуальних агентів. Агенти здійснюють пошук відповідного ресурсу й водночас можуть його надавати. Ієрархія гомогенних агентів побудована таким чином, що кожен агент «знає» лише сусідів, серед яких і здійснює пошук та анонсування сервісів. Така структура дає змогу розширювати систему до масштабів Grid. У процесі планування беруться до уваги параметри якості, QoS (Quality of Service), а саме бажаний час виконання задачі, що визначається користувачем. Відсутність централізованого контролю дає можливість уникнути потенційних вузьких місць (bottlenecks). Для реалізації проекту з огляду на платформну незалежність використовується IBM Java Aglet.

У першій частині статті подано відомості про Grid, архітектуру задач, які ця технологія розв'язує , а також про програмне забезпечення та проблему планування. У другій частині розглянуто аспекти використанняагентіву Grid, основнірозробки; третю частину присвячено проекту планувальника для Grid, що базується на агентах.

Grid

У середині 1990-х рр. виникла ідея створення нової інфраструктури розподілених обчислень для розв'язання передових наукових та інженерних проблем. її було названо Grid — за аналогією з електричною мережею. Тобто основна ідея, покладена в основу технології Grid,- поєднання обчислювальних ресурсів, можливостей сховищ даних, мережних ресурсів, сенсорів тощо, що робить купівлю різноманітних сервісів та ресурсів так само простою, якою сьогодні для нас є купівля електроенергії. Вмикаючи прилад у розетку, ми не знаємо, яким чином виробляється електроенергія, як вона до нас надходить, але використовуємо її для наших потреб. Аналогічно, коли нам потрібні оброблені певним чином дані, ми не знаємо, де Grid бере ці дані та які ресурси використовує для їх обробки, але отримуємо результат. Основ-

© *Гороховський С С, Римарчук В. К.,* 2005

ною метою є створення ілюзії простого, але водночас великого та потужного самокерованого віртуального комп'ютера з великого набору поєднаних гетерогенних систем, що поділяють різні комбінації ресурсів [2].

Технологія Grid виникла як нова важлива сфера, що відрізняється від звичайних розподілених обчислень своїм акцентом на розділенні масштабних ресурсів, новаторських застосуваннях та орієнтації на високу ефективність. Розділення (sharing) ресурсів, про яке тут ідеться,- це не просто обмін файлами, а скоріше прямий доступ до комп'ютерів, програмного забезпечення, даних та інших ресурсів, що необхідні в процесі пошуку стратегій розв'язання проблеми та обміну ресурсами, які виникають у виробництві, науці та інженерії. Таке розділення жорстко контролюється постачальниками ресурсів та споживачами, які чітко визначають, що поділяється, кому дозволено поділяти та за яких умов.

Ознаки Grid. Сьогодні можна багато почути про Compute Grids, Data Grids, Science Grids, Access Grids, Knowledge Grids, Bio Grids, Sensor Grids, Cluster Grids, Campus Grids, Tera Grids та Commodity Grids.

Визначимо ключові характеристики Grid [3]. Це система, що:

- координує ресурси, які не є об'єктами централізованого контролю;
- використовує стандартні, відкриті, універсальні протоколи та інтерфейси;
- забезпечує нетривіальну якість, QoS.

Основні можливості Grid:

- експлуатація ресурсів; паралельне використання процесорів;
- створення й супровід віртуальних організацій;
- створення додаткових ресурсів;
- батансування навантаження.

Grid об'єднує велику кількість ресурсів, наданих окремими комп'ютерами у великий спільний віртуальний ресурс |4]. Для Grid-застосувань використовується планування задач і розподіл їх по машинах із меншою завантаженістю, таким чином двома способами досягається ефективне використання ресурсів та уникнення несподіваних піків перевантаження в роботі:

- неочікуваний пік може бути перерозподілений на машини, що фактично простоюють;
- якщо ж Grid вже повністю завантажена, виконання задач із меншим пріоритетом може бути тимчасово призупинено або відмінено й виконано пізніше, щоб звільнити місце для задач вищого пріоритету.

Без інфраструктури Grid таке батансування навантаження дуже важко досягти.

Архітектура Grid. Ефективна робота Grid вимагає можливості встановлення зв'язків, що поділяються, між усіма потенційними учасниками. Таким чином, інтероперабельність стає центральним результатом, на досягнення якого все спрямовується. У середовищі мережі інтероперабельність означає спільні протоколи. Отже, архітектура Grid — це перш за все архітектура протоколів (протокол є основним механізмом, за допомогою якого користувачі та ресурси домовляються про розділені зв'язки, встановлюють їх, управляють ними та експлуатують їх).

Ця архітектура організована у шари компонентів. Компоненти всередині кожного шару поділяють спільні характеристики, але можуть надбудовувати можливості та поведінки, що забезпечуються компонентами будь-якого нижчого рівня.

Частину проблем Grid вже успішно розв'язано. Для деяких запропоновано локальні рішення специфічних задач, тому важливою є розробка універсальних та стандартизованих рішень для забезпечення масштабованості та інтероперабельності.

Задачі планування в Grid. Планування та балансування навантаження належать до найважливіших функцій Grid. Нині користувачі вільно віддають свої ресурси, а дослідники, приєднані до мережі, їх використовують. По суті, доступність ресурсів є непередбачуваною, і завдання виконуються на основі принципу «першим прийшов - першим виконався». Всередині групи політики планування можуть відрізнятися, але часто вони керуються адміністратором, який виконує задачі за тим самим принципом або встановлюючи пріоритети.

Деякі платформи Grid мають власні можливості планування. Окремі програмні продукти можуть бути інтегровані з існуючими планувальниками. Наприклад, Globus Toolkit може бути інтегрований з багатьма планувальниками, у тому числі PBS, CondorTa LoadLeveler [2]. Однак ці планувальники стосуються кластерного рівня, й для балансування між багатьма кластерами потрібен планувальник вищого рівня.

Було розроблено багато планувальників для Grid, але виділити можна лише кілька (Nimrod-G, GRaDS та Condor-G) як такі, що визначаються високим ступенем розробки і досить широко поширені [9]:

- Nimrod-G (http://www.gridbus.org/);
- GRaDS (http://hipersoft.cs.rice.edu/);
- Condor-G (http://www.cs.wisc.edu/condor/).

Ці три платформи мають такі спільні характеристики:

- всі вони є однією вхідною точкою до Grid;
- всі вони передбачають, що мають контроль політик планування для всіх вузлів;
- всі вони передбачають, що вузли, приєднані до Grid, тобто частина планувальника, що розміщує задачі по вузлах, будуть також відповідальні за планування задач усередині вузла.

Отже, всі платформи мають низку обмежень. Координування колекцій ресурсів - це завдання вищого рівня, і більшість процесів (моніторинг, діагностика, планування) вимагає інтелектуальних, автономних, соціальних здатностей — усе це є характеристиками агентів. Більшість сервісів такого рівня може бути ефективно спрощено внаслідок застосування мультиагентнихсистем (МАС) [6].

Агенти в Grid

Кожна нова технологія, щоб бути корисною, має запропонувати одну з двох речей: можливість вирішити проблеми, розв'язання яких досі не було автоматизовано, або можливість розв'язувати проблеми, що вже розв'язані, але у кращий спосіб (дешевше, природніше, простіше, ефективніше, швидше).

На сьогодні агенти використовуються в Grid по-різному. Однак часто ці застосування не є окремою галуззю й не беруться до уваги дослідниками агентів. Можна відзначити такі проекти [5]: AppLeS, Condor, DPSS, GARA, NetLogger, NetSolve.

Розробки агентних систем. Хоча агенти завдяки здатності до переговорів можуть значно поліпшити планування в таких масштабних системах, як Grid, сьогодні ще небагато зроблено у сфері використання агентів для задач планування. Фактично, в стадії початкових розробок нині перебувають дві системи — AgentScape [6] таА4[7].

AgentScape. Проект AgentScape [6] забезпечує мультиагентну інфраструктуру, що може використовуватися для інтеграції та координування розподілених ресурсів у середовищі Grid. Мета системи - забезпечити «мінімальне, але достатнє» середовище для агентних застосувань. Модель AgentScape визначає *агентів* та *об'єкти* як головні сутності. Крім агентів, об'єктів та розташування, модель AgentScape визначає також і *сервіси,* які забезпечують інформацію або дії від імені агентів чи проміжного програмного забезпечення (middleware) AgentScape.

AgentScape надає низку компонентів, а саме ядро, сервіси каталогів та пошуку ресурсів тощо. Ця широкомасштабна розподілена агентна система спроектована для підтримки гетерогенності та інтероперабельності, полегшує розширюваність: достатньо легко будувати агентні середовища «над» AgentScape. Також AgentScape порівняно просто адаптується до різних операційних систем та мережних інфраструктур. По суті, AgentScape може бути досить просто інтегрована з іншими середовищами та підтримувати підхід до керування Grid-ресурсами, базований на агентах. Робота в цій галузі є незавершеною, й AgentScape лишається прототипом агентної платформи.

Л4 (Agile Architecture and Autonomous Agents) та ARMS. Проектом А4 було розроблено каркас для керування ресурсами в Grid, що базується на агентах. А4 — проектна методологія, що фокусується на розв'язанні проблем масштабованості та динаміки у великих мультиагентних системах. Методологія А4 відповідно до назви (Жвава Архітектура та Автономні Агенти) бере до уваги такі проблеми [8]:

- оскільки багато агентів можуть бути використані як масштабована мультиагентна система, вони не є визначеними наперед для спільної роботи; вони мають власну мотивацію, ресурси тасередовище;
- автономність використовується для опису характеру агента стосовно його інтелектуальності та соціальних здатностей; агент може виконувати складні завдання прямо або через кооперацію з іншими агентами;
- архітектура використовується для забезпечення каркаса для взаємодії між агентами, наприклад, агенти можуть бути організовані в ієрархію;
- жвавість (Agility) використовується для опису характеру архітектури й означає швидку адаптацію до змін середовища; якщо автономність надає системі адаптивність компонентного рівня, то швидкість надає адаптивність на рівні архітектури.

Проект ARMS - це система планування для мереж Grid, що використовує методологію А4 та її цілі. У системі ARMS агенти є водночас продавцями та покупцями обчислювальних сервісів. Сервіс планування працює на «сервісному рівні» з агентами, що анонсують свої ре-

сурси як «сервіси». Як тільки агент продав власний ресурс, починається внутрішнє планування на фактичному ресурсі, використовуючи інструментарій передбачення продуктивності PACE. Використання агентів спрямоване в основному на пошук сервісів, а планувальника PACE - на планування на машинному рівні.

Проект планувальника

Основні вимоги та огляд існуючих розробок. У процесі розробки застосувань для такої розподіленої системи, як Grid, потрібно брати до уваги ключові вимоги, які вони мають задовольняти. Grid вимагає насамперед розв'язання таких проблем [101:

- *масштабованість:* Grid може потенційно включати багато високопродуктивних обчислювальних ресурсів. Цей компонент Grid матиме власні функції, ресурси та середовище. Вони можуть бути фізично розподілені в різних організаціях і не «знати» одне про одного;
- *адаптивність:* Grid — це динамічне середовище, де розміщення, тип та продуктивність компонентів постійно змінюється. Наприклад, ресурсний компонент може бути доданий або вилучений з Grid у будь-який час. Ресурс може бути неповністю виділений для користування в Grid, його можливості можуть варіюватися з часом.

Техніки планування та керування ресурсами, що існують нині, не задовольняють цих двох вимог одночасно. Ми пропонуємо використання ієрархії гомогенних агентів, що є представниками локального ресурсу на вищому рівні, здійснюють пошук відповідного ресурсу та враховують показники продуктивності. Агенти співпрацюють одне з одним для пошуку відповідного ресурсу для задачі, використовуючи техніку анонсування сервісів та пошуку.

Є декілька запропонованих підходів до планування та керування ресурсами в Grid. Проте вони мають свої недоліки і відрізняються від даноїроботи.

- Condor використовує структуру зіставника (matchmaker), що є одночасно і провайдером, і шукачем, де зіставник стає вузьким місцем системи. Досягнення масштабованості ускладнюється. Керування ресурсами можливе лише всередині локальної Grid.
- Globus використовує Metacomputing Directory Service (MDS), що приймає репрезентацію даних та API, визначені LDAP-сервісом. Globus спроектований для досягнення вимог масштабованості та динамічності ресурсів. Питання продуктивності не беруться до уваги.
- Hector використовує підхід майстер/підлеглий (master/slave) для керування динамічними ресурсами, але масштабованості не досягається.
- Legion використовує Resource Management Infrastructure (RMI), де колекції інформації дуже подібні до інформаційних сервісів Globus. Об'єкти використовуються як головна абстракція системи. У цій роботі використовуються агенти як абстракція вищого рівня.
- NetSolve використовує агентів як базу даних та брокер ресурсів для керування ресурсами.
- Ninf - метасерверний компонент, що здійснює моніторинг багатьох серверів мережі та виконує планування і балансування навантаження клієнтських запитів (за принципом, подібним до NetSolve).

УроботахKao [8; 10; 11] в процесі визначення параметрів ресурсу та застосування для зіставлення використовується специфічний інструментарій PACE Toolkit, що порушує критерій універсальності. У застосуваннях, що базуються на системах типу Challenger [12; 13], цього вдалося уникнути, але планування відбувається лише на обмеженій кількості машин, тобто порушується вимога масштабованості.

Агенти-планувальники

Основним компонентом планувальника є агент. Кожен агент розглядається як представник Grid-ресурсу на метарівні — тобто можна розглядати агента як провайдера сервісу. Агенти ієрархічно організовані. Кожен агент (крім «найвищого») має одного «вищого» агента і може мати або кілька «нижчих» агентів, або не мати жодного. Інформація про локальний ресурс анонсується агентом як угору, так і вниз. Також агенти взаємодіють одне з одним для пошуку відповідного ресурсу.

Кожен агент має інформацію про сервіси сусідніх («вищих» та «нижчих») агентів, що записана у відповідні файли, які агент обробляє в процесі пошуку залежно від джерела інформації. Наприклад, файл local містить дані про локальний ресурс; lowl, low2 і т. д. містять інформацію, отриману від «нижчих», і high - від «вищого» агентів відповідно. Інформація оновлюється при отриманні анон-

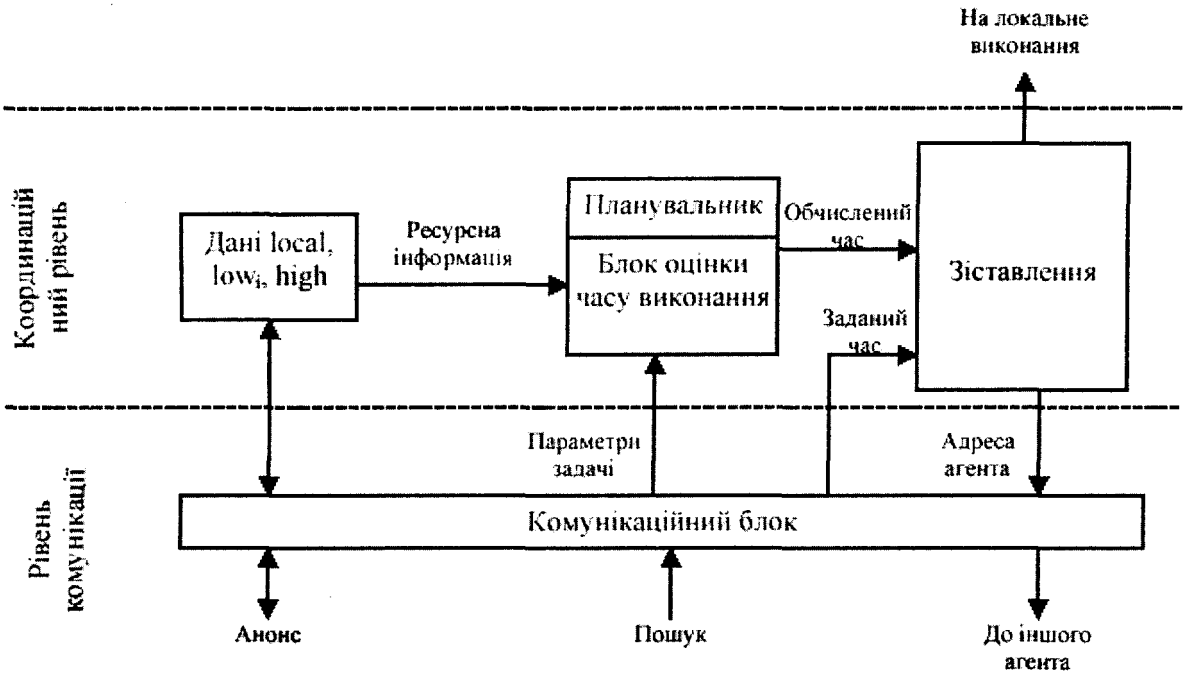

Рис. 1. Структура агента

су від відповідного агента. При зміні в локальному ресурсі агент відправляє свій анонс до сусідніх агентів. У процесі пошуку агент переглядає цю базу в такому порядку: local, low$_i$, high. Якщо інформацію вичерпано, а ресурсу не знайдено, задача пересилається до «вищого» агента. Коли досягнуто верхівки ієрархії, пошук припиняється з причин невдалого результату, тобто для задачі немає відповідного ресурсу.

Структуру агента [4] показано на рис. 1. Агент має два рівні – комунікаційний та координаційний. Через комунікаційний рівень агент взаємодіє з іншими агентами, відсилаючи до них або отримуючи від них анонси та задачі на виконання. На координаційному рівні відбувається власне процес планування.

Визначення часу виконання. У файлі локальних ресурсів агента міститься інформація про процесори кластера, який представляє цей агент, а саме *id* процесора та кількість операцій з плаваючою точкою, які він може виконувати за секунду, *op*. Відповідна інформація міститься також у файлах щодо «вищого» та «нижчих» агентів. Своєю чергою, для задачі визначається її складність *cmplxt* (у даній реалізації – це константна, лінійна, ступенева або експоненційна складність) та порядок розміру вхідних даних *deg*.

З параметрів *cmplxt* та *deg* визначається необхідна кількість операцій, що вимагає виконання задачі, *num_op*. Час (у секундах) виконання задачі j на певному процесорі i визначається таким чином:

$$t_{comp} = (num\_op_j) / op_i . \qquad (1)$$

Отриманий час потім використовується в процесі зіставлення.

Планування. Окрім складності та ступеня вхідних даних, параметром задачі є бажаний час виконання (у секундах) t_{desir}. Коли до агента надходить нова задача (від користувача, тобто локально, або від сусіднього агента), вона стає у чергу з пріоритетом. Чим менший бажаний час виконання, тим вищий пріоритет у задачі і тим вона ближче до початку черги. Процес планування починається з того, що з черги береться задача найвищого пріоритету і для неї обчислюється t_{comp} послідовно для процесорів спершу локально, потім серед «нижчих» агентів і, нарешті, у «вищого» агента. Як тільки перший t_{comp} обчислено, його значення порівнюється з t_{desir}. Якщо $t_{comp} \leq t_{desir}$, це означає, що відповідний процесор задовольняє вимогам виконання задачі. Задача стає спланованою і вилучається з черги, подальший пошук призупиняється. В іншому ж випадку процес пошуку триває. Якщо відповідний процесор було знайдено в сусіднього агента, задача пересилається до відповідного агента. Якщо задачу було отримано від одного з «нижчих» агентів, то це означає, що або відповідний ресурс було знайдено «нижчим» агентом у поточного агента, або ресурс не було

знайдено. Тобто в обох випадках ресурс не було знайдено у «нижчого» агента, тому в процесі пошуку ресурсу для цієї задачі відповідний файл агента-відправника не переглядатиметься.

Реалізація. Для розробки планувальника в цій роботі було використано **IBM** Java Aglet з огляду на платформну незалежність.

Основним класом є *My Agent.* При створенні агент зчитує свій конфігураційний файл такої структури:

- власна адреса;
- кількість «нижчих» агентів;
- адреса «вищого» агента;
- адреси «нижчих» агентів.

Задача від користувача представляється у вигляді текстового рядка, в якому параметри розділяються проміжком. Наприклад, рядок «0 123 є 2 1000» означає, що задачу було отримано від локального користувача («0», якщо задачу отримано від іншого агента, то першим параметром буде його адреса), її номер «123», складність експоненційна («є»), порядок обсягу вхідних даних = 2, і вона має бути виконана за 1000 **с** Такий рядок розбивається на окремі параметри, що записуються в одновимірний масив. Такі масиви стають у чергу, що реалізована за допомогою динамічного масиву (клас Vector). Оскільки задачі мають різний пріоритет, то місце в черзі визначається відповідно до останнього параметра (час виконання); чим він менший, тим ближче до початку черги стає задача.

Метод *sched* реалізує пошук ресурсу, основною функцією є *search,* що має такі параметри: складність задачі, порядок розміру вхідних даних, заданий час виконання та файл (повний шлях), серед інформації якого і шукається ресурс (спочатку локально, потім у «нижчих» агентів і, нарешті, у «вищого»). Структура цих файлів така:

- id_1 процесора;
- кількість операцій з плаваючою точкою за секунду для процесора *id',*
- id_2 процесора;
- кількість операцій з плаваючою точкою в секунду для процесора *id.*

Функція *countPar* обчислює кількість операцій, якої потребує задача певної складності й порядку обсягу вхідних даних. За формулою (1) обчислюється час, за який задача може бути виконана даним процесором.

Якщо ресурс знайдено у сусіднього агента, створюється «підлеглий» (slave) агент (клас *MyAgentSlave),* шо відсилається за адресою відповідного сусіднього агента. «Головний» (master) агент відсилає своєму «підлеглому» повідомлення типу «job» (клас *com.ibm.aglet. Message)* і аргумент - текстовий рядок задачі, але першим параметром стане власна адреса агента-відправника, а «підлеглий», своєю чергою, використовуючи локальний обмін повідомленнями (local messaging), передає задачу цільовому агентові. Такий варіант є зручним, оскільки передача повідомлення є локальною, немає зайвого завантаження мережі.

Коли параметри локального ресурсу змінюються, агент відсилає анонс про ці зміни сусіднім агентам. Інформація файла *local* формується у текстовий рядок, першим параметром є власна адреса агента. За таким самим механізмом, як і пересилання задачі, відбувається пересилання анонсу. Відмінність полягає в тому, що анонс пересилається до усіх сусідніх агентів і тип повідомлення — «announce». Агент-отримувач аналізує отриманий рядок, визначає за адресою, хто є відправником, і змінює файл (low або high) відповідно до отриманих даних (метод *RewriteFile).*

Висновки

Результатом проведених досліджень стала розробка планувальника, основаного на агентах, для Grid. Побудований планувальник відповідає основним критеріям масштабованості та адаптивності. Така реалізація дає змогу використати переваги агентних технологій, уникнути центршіізованого контролю, а отже, і вузьких місць, зменшити навантаження на мережу. Пошук ресурсу відбувається з урахуванням вимог користувача, а саме часу виконання задачі. Пошук призупиняється, як тільки знайдено ресурс, що задовольняє цій вимозі. Описана стратегія пошуку ресурсів не забезпечує оптимального балансування навантаження, але метою є віднайдення відповідного ресурсу якнайшвидше та якнайближче, що, як правило, є вигідним для користувача, тобто планування є орієнтованим на користувача, а не на балансування навантаження в мережі. Використання такого планувальника уможливить розв'язання частини проблем координування ресурсів у Grid. Поєднання двох перспективних технологій — Grid та агентів - є виграшним, оскільки переваги агентів дають змогу природніше та ефективніше розв'язати ключові задачі Grid, причому не лише проблеми планування та координування ресурсів.

1. *Гороховськиі С. С* Агентні технології: спроба критичного огляду // Наукові записки НаУКМА. Комп'ютерні науки.- 2001.- Том 19.- Частина П.- С. 391-395.
2. *Foster I., Kesse/man C.* The GRID: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.
3. *Foster I., Kesselman C, Tuecke S.* The Anatomy of the Grid. Enabling Scalable Virtual Organizations, 2001.- http://www.globus.org/research/papers/
4. *Ferreira L., Berstis V., Armstrong J.* at al. Introduction to Grid Computing with Globus, 2002.- http://ibm.com/redbooks/
5. Grid Today. Daily news and information for the Global Grid Community.- July 22.- 2002.- Vol. 1, № 6.- http://www.gridtoday.com/
6. *Cao J., Kerbyson D. J., Nudd G R.* Using Software Agents in Computational Grids, 1999.
7. *Wijngaards N. J. E., Overeinder B. J., Steen M. van. Brazier F. M. T.* Supporting internet-scale multi-agent systems, 2002.- http://www.iids.org/publications/bnaic2002_dke.pdf
8. A4 Project.- http://wwwxcrl-nece.de/~cao/A4/
9. *Cao J., Kerbyson D. J., Nudd G. R.* ARMS: an agent-base resource management infrastructure for grid computing // Proc. 1st IEEE International Symposium on Cluster Computing. Scientific Programming. Special Issue on Grid Computing 10, 2002.
10. *Gradwell P.* Grid Scheduling with Agents,- University of Bath, BATH, 2003.
11. *Cao J., Kerbyson D. J., Nudd G. R.* Performance evaluation of an agent-based resource management infrastructure for grid computing *И* Proc. 1st IEEE International Symposium on Cluster Computing and the Grid.— Brisbane, Australia, 2001.
12. *Cao J., Spooner D. P., Jarvis S. A., Saini S., Nudd G. R.* Agent-base load balancing using performance-driven task scheduling /*I* Proc. 17th IEEE International Parallel & Distributed Processing Symposium, Nice.- France, 2003.
13. *Chavez A. M. A., Maes P.* Challenger: a multi-agent system for distributed resource allocation // Proc. 1st International Conference on Autonomous Agents, 1997.
14. *Greenshields I. R., Kerasha M., Pandya D.* A multi-agent distributed resource allocator in untrustworthy grid environment, 2001.

S. S. Gorokhovsky, V. K. Rymarchuk

GRID TECHNOLOGY AND SCHEDULING TASKS WITH INTELLIGENT AGENT

The paper is devoted to association of two super technologies — to technology Grid which offers creation of a new infrastructure of the distributed computations by the coordinated access to the various resources, incorporated by a network, and agent technology which provides intelligent, autonomous, and social abilities. The primary goal of Grid is a coordination of collections of resources. Application agent systems for planning tasks in Grid will allow solving two basic problems — scalability and adaptability. Methods and techniques which are used at present do not provide the decision of these difficult problems.