УДК 004.057.4+004.724

Ю. О. Тимошенко, А. С. Краєвий

ОГЛЯД ТА КЛАСИФІКАЦІЯ МЕХАНІЗМІВ ПОБУДОВИ РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Вступ

Сучасні обчислювальні середовища значно складніші тих, що використовувалися у недавньому минулому, у часи дорогих обчислювальних потужностей, обмежених ресурсів для збереження інформації і статичної структури обчислювальних мереж. Зараз, окремі елементи даних та сервіси є мобільними і дублюються для збільшення доступності, продуктивності і надійності всієї системи.

У традиційних моделях клієнт-сервер, інформація сконцентрована на серверах і розподіляється через мережу клієнтським комп'ютерам, що діють як пристрої для її відображення. У централізованих системах легше керувати правами доступу і безпекою, однак централізація неминуче приводить появи критичних компонент системи і неефективності використання ресурсів.

Зазначимо, що при зростанні запитів до ефективності і потужності інформаційних систем зростає важливість та популярність розподілених рішень, які використовують окремі обчислювальні машини об’єднані мережею [1]. В основному це пов'язано з тим що ціна використання і підтримки агрегованого мережевого обчислювального ресурсу виявляється на порядки нижчою в порівнянні зі створенням і монопольним використанням централізованого [2].

Існує досить велика кількість різних типів розподілених обчислювальних систем (РОС), починаючи від спеціалізованих багатопроцесорних платформ і закінчуючи різними видами систем, що поєднують кінцеві станції обчислювальних мереж [3, 4].

Постановка задачі

У даній роботі розглядаються симетричні розподілені обчислювальні системи (СРОС), що також відомі в джерелах як РОС класу peer-to-peer. Серед задач, які реалізуються за допомогою СРОС, можна вказати розподілені обчислення, децентралізоване збереження, індексування й обробку великих обсягів даних, підтримку комунікації та спільної роботи користувачів системи [2, стор. 8; 5].

Їх основною відмінною рисою в порівнянні з узагальненими РОС є найбільший ступінь децентралізації та динамічності. Динамічна природа таких обчислювальних середовищ значно ускладнює традиційні методи рішення задач знаходження й іменування об'єктів, маршрутизації повідомлень і забезпечення цілісності даних інформаційної системи.

Як наслідок, дослідники пропонують спеціалізовані рішення цих задач, що використовують нові методи організації і взаємодії компонент СРОС [6, 7, 8]. В статті надається узагальнений огляд актуальних вимог до роботи СРОС та методів їх забезпечення. Основною задачею даної роботи є аналіз існуючих прикладів СРОС, визначення базових понять і характеристик, необхідних для їх порівняння й оцінки.

Загальна модель

Існує декілька визначень СРОС, які використовуються серед їх дослідників. Дослідницька група Intel використовує визначення "обчислювальні системі, які базуються на розподіленні ресурсів за допомогою прямого обміну". Крім цього, можна навести наступне визначення: розподіленні системи, які використовують кінцеві робочі станції мережі як сервера [9]. В роботі [10] наводиться схоже визначення: "РОС класу peer-to-peer використовують ресурси кінцевих робочих станцій обчислювальної мережі". Розподілюваними ресурсами можуть бути обчислювальні потужності, дані або ж мережеві пропускні здібності [2].

Проте, слід зазначити, що у цілком децентралізованій моделі СРОС не може бути централізованого сервера з глобальним представленням усієї мережі, а це ускладнює використання системи. Тому більшість систем використовують гібридні моделі, з деяким ступенем централізації. Таким чином, не можна сказати, що модель будь-якої СРОС обов'язково вільна від централізованих обчислювальних компонентів - у багатьох практичних прикладах, як буде показано далі, деякі з функцій системи реалізовуються централізовано [2, 4].

Для організації взаємодії вузлів, в кожній СРОС задається спеціалізований протокол. Взагалі такий протокол слід віднести до прикладного рівня моделі OSI [11]. Проте, як буде показано далі, цей протокол часто може враховувати деталі нижчих рівнів.

Часто вузли мережі СРОС можуть знаходитися в різних обчислювальних мережах з точки зору транспортних протоколів, при цьому вони об’єднуються в одну мережу з точки зору протоколу СРОС [2, 3]. Така логічна мережа, побудована «понад» готовою фізичною інфраструктурою називається оверлейною мережею. При цьому вузли оверлейної мережі з’єднуються не апаратними, а логічними каналами - повідомлення, яке передається напряму по логічному каналу може насправді ретранслюватися декілька разів у апаратній мережі. Як наслідок, в оверлейній мережі з’являється власний абстрагований механізм маршрутизації, який, наприклад, не покладається на протокол IP та IP-адреси вузлів [2].

Таким чином, протокол СРОС, який загалом відноситься до прикладного рівня, може передбачувати механізми маршрутизації. Зазначимо, що такі механізми маршрутизації також виявляються абстрагованими від фізичного середовища передачі повідомлень, і, взагалі кажучи, можуть відноситися до мережевого рівня стеку протоколів OSI тільки для оверлейної мережі. Для маршрутизації оверлейна мережа використовує власні адреси вузлів, які виявляються специфічними для кожної СРОС.

При цьому, якщо ми розглянемо адресний простір, який використовує система, та відповідність цих адрес до вузлів системи у адресному просторі то ми прийдемо до розгляду віртуальної топології СРОС. Віртуальна топологія виявляється корисною абстракцією при аналізі оптимальності роботи механізму маршрутизації повідомлень СРОС та загальній оцінці організації оверлейної мережі.

Основні вимоги до СРОС

Основними вимогами до СРОС є надійність, правильність і безпека її роботи, проте стандартні методи забезпечення таких вимог ускладнюються динамічністю обчислювального середовища і малою надійністю компонент системи.

Надійність роботи системи виражається у впливі наступних показників на виконання функцій системи: одночасні збої деякої частки компонент (живучість), пікові часові навантаження (розширюваність), неоднорідності навантаження на компоненти (балансування навантаження) [2, 4].

Правильність роботи системи зв'язана з такими фундаментальними характеристиками як цілісність моделі даних та детермінованість результатів виконання операцій.

Безпека роботи зв'язана з безпекою виконання обчислень і збереження даних у розподіленому середовищі. Важливо виділити наступні значимі аспекти: безпека локального доступу до розподілених ресурсів; безпека доступу розподіленого коду до локальних ресурсів; прозора спільна робота декількох користувачів в одній системі; об'єднання обчислювальних ресурсів декількох організацій в одну віртуальну обчислювальну систему [12].

Для забезпечення цих аспектів використовуються наступні методи: криптографічний захист локального постійного запам'ятовуючого пристрою і даних в оперативній пам'яті, віртуальні машини і інтерпретовані мови програмування, стратегії розподілу навантаження і пріоритети задач користувачів, агреговані механізми контролю доступу до обчислювальних ресурсів [2, 12].

Живучість: Для забезпечення живучості системи можна застосувати декілька рішень, одним із найочевидніших є часткова централізація системи, з метою забезпечення більшої надійності виконання критичних задач. Це рішення не є однозначним, тому що працездатність усієї системи стає залежною від працездатності центрального компонента. Однак, централізація використовується досить часто, тому що виявляється вигідною або ж і зовсім необхідною з інших точок зору.

Найбільш розповсюдженим рішенням, що не впливає на структуру СРОС, варто вважати надлишкове копіювання прикладних компонентів серед апаратних, з метою збільшення доступності даних при збоях вузлів, тобто використання різних схем реплікації та кешування. Підкреслимо, що реплікація є процесом активного дублювання елементів, в той час як кешування – процес, залежний від операцій, які виконує РОС [13, 14].

Крім того, можливе визначення ступеня критичності даних і статистики збоїв вузлів, з використанням цих показників при розподілі даних серед вузлів [15]. Аналогічним методом можна розподіляти обчислення серед учасників СРОС або використовувати не критичність даних, а частоту їх використання.

Розширюваність: Розширюваність системи в основному залежить від методів її організації і ступеня залежності виконуваного коду і даних від окремих підзадач. Точніше, якщо код і дані, необхідні для виконання окремих підзадач, слабко передбачувані – система буде слабко розширюваною [16, 17].

Таким чином, можливі гібридні схеми, у яких визначений набір задач системи виконується централізовано - з метою забезпечення надійності збереження даних і роботи трансакцій (за рахунок зменшення розширюваності і спрощення реалізації). Наприклад, такий підхід був використаний у проекті SETI@home [3].

Розподілення навантаження: Існує досить широкий набір різних стратегій балансування навантаження в розподілених обчислювальних середовищах, починаючи від методів прямого і випадкового обміну і закінчуючи розподіленими стратегіями які базуються на статистичних підходах для аналізу локального представлення системи й адаптації схеми розподілу усієї системи [8, 14].

Зупинимося докладніше на можливих застосуваннях алгоритмів балансування навантаження і найбільш розповсюджених методах вирішення для кожної області:

1. неоднорідність обсягів даних, що зберігаються вузлами, приводить до появи невеликої кількості вузлів, що зберігають велику частину даних системи при малій завантаженості інших, і, отже, до різкого зменшення живучості системи;
2. неоднорідність запитів на одержання і зміну даних, приводить до появи перевантажених вузлів, що сповільнять роботу всієї системи в цілому і погіршать розширюваність системи;
3. неоднорідність у пропорції між доступними і використовуваними локальними ресурсами апаратних компонентів системи приводить до зниження загальної ефективності роботи системи і знижує її адаптивність;
4. неоднорідність віртуальних шляхів передачі запитів між прикладними компонентами приводить до насичення апаратних комунікаційних каналів і зниження розширюваності системи;

5. неоднорідність у показниках критичності прикладних компонентів і надійності відповідного апаратного компонента приводить до зниження загальної надійності і живучості системи.

Для вирішення задач 1 і 2 використовують методи обміну (або більш складні адаптивні алгоритми) і стратегії реплікації даних [14]. Для рішення задачі 3 використовуються ті ж алгоритми, але з використанням не абсолютного, а відносного показника завантаження апаратного компонента системи [8]. Для рішення задачі 4 використовуються різні методи адаптації віртуальної структури мережі до апаратної і статистичні підходи, що використовують мережні затримки при передачі повідомлень [18]. Задача 5 вирішується за допомогою додаткових стратегій з нечіткими станами, що використовують статистику збоїв апаратних компонент [15].

Цілісність даних, детермінованість результатів: Помітимо що взагалі СРОС не надають тих гарантій, що забезпечуються централізованими системами щодо цілісності даних і детермінованості результатів виконання обчислень. В основному це зв'язано з меншою надійністю апаратних компонентів розподіленої системи і значною гетерогенністю використовуваних ресурсів.

Взагалі СРОС доцільно застосовувати у обчислювальних задачах які допускають існування значних затримок доступу до гарантовано цілісних даних, або можуть функціонувати з використанням частково застарілих даних. Таким чином, СРОС використовує так звані «частково узгоджені» (loosely consistent) структури даних, які, крім того, не зберігаються цілком на жодному з вузлів обчислювальної мережі а виявляються розподіленими по мережі [8].

Крім такого розподілу даних, резервне копіювання функціональних компонентів і елементів даних ускладнює підтримку їх погодженості. Як наслідок, багато систем спираються на централізовані компоненти для рішення деякої підмножини задач, що вимагають високого ступеня погодженості даних та надійної ізоляції трансакцій.

Для покращення цих характеристик в розподіленому середовищі використовують паралельне виконання однієї задачі на декількох вузлах з наступною перевіркою результатів, цифрові підписи повідомлень, використання черг повідомлень з нечітким статусом доставки.

Приклади СРОС

У цьому розділі ми наведемо загальні характеристики різних прикладів СРОС, з погляду їх призначення та походження. При цьому відзначимо, що деякі з розглянутих прикладів є програмними платформами для реалізації і виконання розподілених програм (Avaki, JXTA), у той час інші є протоколами для взаємодії компонент(вузлів) СРОС (Tapestry, FreeNet, Gnutella).

Програмна платформа, призначена для реалізації СРОС, повинна представляти протокол для взаємодії вузлів. У випадку комерційної реалізації платформи, публічний доступ до протоколу часто пов'язаний з деякою ліцензійною угодою, у той час як публічний протокол може підтримуватися різними, повністю незалежними, програмами.

Програмна платформа Avaki: ця платформа надає можливість розглядати і використовувати сукупність гетерогенних обчислювальних середовищ як один віртуальний комп'ютер, за допомогою реалізації основних ідей метакомп'ютинга для інформаційно-обчислювальних мереж, розмір яких варіюється від корпоративних до глобальних.

Проект Avaki є спадкоємцем проекту Legion, що був ініційований Ендрю Грімшоу в університеті Вірджинії у 1993 році [19]. Проект Legion, у свою чергу, є розширенням проекту Mentat, об'єктно орієнтованої системи для паралельних обчислень [20]. Висока продуктивність, розширюваність і приховання складності в умовах гетерогенності, були головними критеріями при розробці системи Avaki.

Головною метою проекту Legion є одержання уніфікованого представлення обчислювальних ресурсів, розподілених по країні, у вигляді віртуального комп'ютера. По завершені стадії розробки і першого показу технології Legion, цей дослідницький проект у 1998 році був комерціалізований Applied MetaComputing. У середині 2001 року, ця компанія-власник була перейменована в Avaki Corp.. У даний момент проект фокусується на наданні розподілених обчислювальних рішень рівня підприємства.

Крім рішень для підприємств, Avaki надає значні обчислювальні потужності для рішення наукових задач, що полягають у дослідженні просторів параметрів, прикладами можуть бути задачі біохімічного моделювання: complib - порівняння послідовностей протеїнів і ДНК і CHARMM - дослідження хімічних сполук (Chemistry at HARvard Molecular Mechanics).

Подальший розвиток проекту передбачається в реалізації спеціалізованих порталів для доступу до розподілених систем, поліпшенні моделі безпеки і подальшій роботі по інтеграції гетерогенних розподілених систем. Також можливе розширення програмного інтерфейсу користувачів - планувальники задач, інформаційні сервіси і спеціалізовані інтерфейси для конкретних обчислювальних задач.

Проект SETI@home: Програма SETI (Search for Extraterrestrial Intelligence) – сукупність дослідницьких проектів, спрямованих на виявлення чужорідних цивілізацій. Один з цих проектів, SETI@home, аналізує карти радіовипромінювання космосу, отримані за допомогою телескопа Arecibo.

Проект SETI@home можна віднести до групи науково-дослідних, його метою є створення величезного віртуального комп'ютера, заснованого на вільній обчислювальній потужності робочих станцій, підключених до Internet. Цей проект був широко підтриманий і одержав значну обчислювальну потужність у кілька десятків teraFlop у результаті об'єднання більш трьох мільйонів комп'ютерів.

Основною цінністю цього проекту є його вік, достатній для досягнення дуже високої якості прикладних засобів і підтвердження застосовності неструктурованих СРОС для задач, які раніше розв'язувалися централізованим способом на спеціалізованих суперкомп'ютерах.

Розподілена система Groove: Система Groove використовується для комунікації, спільної діяльності й обміну даними: голосовий зв'язок по Internet, обмін текстовими повідомленнями, тематична організація дискусій; обмін файлами, документами, зображеннями і контактами; планування групової діяльності, спільне редагування документів і діаграм. Вона призначена для підтримки спільної роботи користувачів локальних обчислювальних мереж і Internet і може використовуватися на мобільних пристроях [21].

Groove був заснований у 1997 Реєм Оззі, одним з ведучих розробників Lotus Notes. Перша версія продукту була опублікована на початку 2001 року. Головною задачею Groove було забезпечення прямого зв'язку користувачів без використання сервера [22]. У березні 2005 року були опубліковані результати успішно завершених переговорів про придбання корпорацією Microsoft компанії Groove Networks, з метою розширення можливостей аналогічних продуктів Microsoft: Microsoft SharePoint Portal Server і Microsoft Office Live Communications Server [23].

З точки зору користувачів, Groove являє собою сукупність спільно використовуваних просторів, що мають наступні властивості: спонтанність - користувачі можуть діяти без адміністратора, безпека - простори функціонують як віртуальна приватна мережа для визначеної групи користувачів, контекстна орієнтованість - простір визначає і зберігає контекст діяльності користувачів, синхронність - простір синхронізується з усіма пристроями і користувачами, фрагментарність - синхронізація реалізується за допомогою обміну фрагментами документів.

Протокол FreeNet: Концепція протоколу FreeNet була запропонована Іаном Кларке в 1999 році в університеті Единбурга, як засіб забезпечення свободи преси в глобальній обчислювальній мережі. Публічна розробка зразка FreeNet почалася в 2000 році [24, 25].

Головною метою FreeNet є анонімність збереження і доступу до інформації при забезпеченні надійності, простоти використання і ефективності: користувач може робити будь-які запити, при цьому для інших учасників немає можливості встановити джерело запитів; при збереженні файлу у системі немає можливості встановити хто саме зберігає файл; оператор будь-якого вузла системи не може знати які саме файли зберігаються на його вузлі.

Ці форми анонімності дозволяють надавати простір для збереження файлів у системі і користуватися системою з упевненістю, що користувач залишиться невідомим і не може бути притягнутий до відповідальності.

FreeNet доступний у виді реалізації з відкритим кодом. Внутрішній протокол системи також добре формалізований, що при необхідності дозволяє створювати сторонні реалізації.

Протокол Tapestry: Протокол Tapestry призначений для пошуку і маршрутизації у СРОС. Головними відмінними рисами цього проекту є використання статистики збоїв і продуктивності на рівні окремих вузлів для досягнення кращих показників роботи всієї системи в цілому [15].

У той час як традиційні підходи до пошуку ресурсів в обчислювальному середовищі використовують централізований підхід, Tapestry використовує тільки прямі з'єднання між вузлами мережі і надає власну інфраструктуру для пошуку, засновану на переадресації локальних повідомлень до найближчої копії ресурсу або сервісу.

З часу початку розробки Tapestry у березні 2000 року, були опубліковані декілька схожих дослідницьких проектів - самими важливими з них є Pastry під керівництвом Microsoft Research і техаського університету Райса, Chord, що спільно розроблюється MIT та університетом Berkeley і Content-Addressable Networks (CAN), розроблюваний Центром інтернет-досліджень AT&T.

Незважаючи на те, що всі ці проекти в основному концентруються на механізмах пошуку даних і маршрутизації повідомлень, основною відмінною рисою Tapestry є використання даних про топологію обчислювальної мережі для оптимізації роботи системи. Основні версії систем Chord і CAN не беруть до уваги мережеві затримки під час побудови початкової віртуальної структури мережі, в той час як Tapestry і Pastry обмежують кількість апаратних ретрансляцій необхідних для однієї переадресації у віртуальній мережі [26].

Протокол Gnutella: Головною метою протоколу Gnutella є надання цілком розподіленого рішення для обміну файлами. Програми, що використовують його, дозволяють користувачам шукати і завантажувати файли інших користувачів. Головною перевагою Gnutella є можливість доступу до великої кількості інформаційних ресурсів, однак ця перевага нівелюється нестабільністю пошуку ресурсів, що представлені малою кількістю примірників. Децентралізована природа Gnutella забезпечує користувачам визначений рівень анонімності, в основному за рахунок ослаблення детермінованості результатів.

Технологія обміну файлами Gnutella була представлена в березні 2000 року двома працівниками підрозділу Nullsoft компанії AOL у вигляді програми з відкритим кодом і функціональністю, схожої з програмою Napster [3]. Програма Gnutella була відключена на наступний же день після запуску, у зв'язку з можливою погрозою для Warner Music і EMI. Незважаючи на малий час роботи, ця програма з відкритим кодом була доступна достатній час для того щоб з'явилися її копії, що використовують той же протокол [2].

Через досить короткий час, версії оригінальної програми почали зв'язуватися з модифікованими версіями. З часу появи першої реалізації Nullsoft багато компаній надали свої реалізації, намагаючись

виправити недоліки оригінальної програми. Серед найбільш популярних можна назвати Limeweire, ToadNode і BearShare [2].

В результаті СРОС, що використовує цей протокол, досі функціонує, об'єднуючи різні програми різних авторів та версій.

Платформа JXTA: Проектування платформи JXTA розпочалося в 1999 році як внутрішнє дослідження корпорації Sun Microsystems під керівництвом Біла Джоя та Майка Клері. Головним завданням проекту було вивчення можливостей розподілених обчислень при використанні симетричних моделей, і створення базових елементів та сервісів для СРОС. Основна ідея проекту - надати відкриту платформу для широкого спектру розподілених програм і апаратних реалізацій. В 2001 році були опубліковані протоколи та їх перша реалізація, проект набув значної популярності та пізніше перейшов у стадію публічного і за останні роки зазнав значного покращення і публікації наступної версії [27].

Головними цільовими характеристиками JXTA можна вважати наступні:

- легкість взаємодії: можливість вузлів мережі робити пошук один одного, надавати публічні сервіси і користуватися ними;
- незалежність від платформи: JXTA не залежить від мови програмування, системної платформи і мережевого середовища;
- легкість використання: JXTA можна реалізувати і використовувати на будь-якому пристрої - починаючи від серверних систем обробки інформації і закінчуючи мобільними пристроями.

Ця платформа розділена на три рівні, нижній рівень платформи містить базові механізми і концепції, рівень сервісів базується на нижньому рівні для розширення його можливостей. Прикладний рівень містить набір програм, що демонструють широку застосовність платформи [28].

Характеристики СРОС

У цьому розділі ми проведемо порівняння різних прикладів СРОС, з метою огляду методів вирішення задач актуальних для СРОС та ефективності цих методів в кожному конкретному випадкові.

Програмна платформа Avaki: Платформа Avaki являє собою віртуальну машину з легко розширюваним розподіленим керуванням, розділену поміж декількома незалежними областями адміністрування, що виявляється доцільним з точки зору забезпечення вимог безпеки.

Розміри мереж та задач, підтримуваних Avaki, пов'язані зі значною надмірністю обчислювальних ресурсів, проте визначення і корекція помилок є актуальною задачею. При розробці системи було прийнято рішення збільшити швидкість виконання за рахунок зменшення ступеня стійкості: живучість системи визначається стійкістю при відмовах апаратних компонентів системи і не враховує прикладних помилок [19].

При збої на рівні вузла мережі, Avaki динамічно змінює конфігурацію обчислювальної мережі і мігрує відповідні роботи на інші вузли. У випадку внутрішніх помилок при виконанні прикладної програми користувач попереджається і може вплинути на подальші кроки системи щодо програмних збоїв.

Проект SETI@home: Головними компонентами в цьому проекті можна вважати сервер бази даних і клієнт. Вузьким місцем при розширенні системи може бути сервер, що використовується для розподілу робіт кожній робочій станції-учаснику проекту, збереження результатів виконання робіт. Однак, велика кількість користувачів дає підставу стверджувати що сервер може витримати таке навантаження.

Модуль клієнта не зв'язаний з будь-якою технологією або програмною платформою, його реалізації доступні для найбільш поширених операційних систем; хоча така незалежна реалізація здається дорогою, у даному випадку вона полегшила вирішення проблем безпеки (ізоляція локального середовища виконання, підписування даних).

Стійкість виконання є головною цінністю в цьому проекті, якщо ми врахуємо, що один ізольований етап обчислень займає біля десяти годин. Отже, необхідно гарантувати прозоре збереження і наступне відновлення результатів процесу обчислень при вимиканні комп'ютера або початку виконання задач користувача.

Механізм забезпечення живучості, використаний в цьому проекті полягає в збереженні проміжних результатів роботи в локальному постійному сховищі кожні 10 хвилин і називається checkpointing. При перериванні виконання обчислення продовжається з останнього збереженого набору даних. Цей механізм додає малий відсоток обчислень, досить простий у реалізації і забезпечує значну стабільність виконання програми.

Також помітимо, що основною проблемою, яка вимагала підписування даних, були не атаки зловмисних користувачів, а апаратні збої обчислень при збільшенні частот процесорів локальних комп'ютерів з метою підвищення продуктивності й одержання кращих місць у рейтингу учасників проекту.

Розподілена система Groove: В системі Groove для збереження і передачі командних запитів і даних використовуються об'єкти XML, що дозволяє зберігати послідовності команд і передавати їх для відновлення даних у випадку відключення користувача. В основному вказаний механізм призначений для забезпечення стабільності роботи системи через збереження повідомлень у черзі при втраті зв'язку з цільовим вузлом мережі або його тимчасовому збої. Крім того, це дає користувачам можливість працювати з локальною версією даних, відкладаючи синхронізацію у випадку недоступності мережі.

Таким чином, в CPOC Groove з метою підвищення живучості використовуються механізми кешування повідомлень та реплікації об'єктів з можливістю відкладеної синхронізації, що виявляється достатнім для гарантування стабільності роботи цієї сильно централізованої CPOC [22].

Протокол FreeNet: З використанням простої стратегії кешування даних, FreeNet оптимізує обробку запитів на знаходження необхідного вузла:

- при успішному знаходженні файлу, адреса вузла і файл зберігається в локальних маршрутних таблицях вузлів, що обробляли запит, таким чином, часто запитувані файли частіше реплікуються і при реплікації наближаються до запитувачів;
- при одержанні наступного запиту, запит транслюється до того вузла з маршрутної таблиці, у якого ключ найбільш схожий із запитуваним, це призводить до того що запит швидше наближається до цільового вузла-учасника мережі.

Розширюваність FreeNet досліджувалася з використанням моделювання [24]. Ці дослідження дозволяють затверджувати, що довжини маршрутів ростуть логарифмічно при збільшенні числа користувачів. В експериментах використовувалися мережі з 1000 вузлів з кільцеподібною топологією. Під час виконання експерименту випадкові ключі додаються і видаляються із системи і довжини маршрутів повідомлень зменшуються від 500 до 10, що відповідає очікуваній логарифмічній залежності.

Одним з недоліків зв'язаних з анонімністю використання FreeNet є великі труднощі спостереження за поводженням усієї системи в цілому. Відсутність ідентичності і стохастичні вибори роблять виміри неусталеними і статистично недостовірними. Єдине реальне застосування FreeNet - системи збереження і пошуку інформації [2].

Часте використання криптографічних методів захисту й анонімізації запитів може виявитися корисним в інших ситуаціях. Наприклад, цей протокол застосовний для резервного копіювання даних, з гарантією конфіденційності збереженої інформації. Проте відсутність гарантій збереження даних та мала функціональність протоколу значно зменшують його корисність.

Протокол Tapestry: Вузли мережі Tapestry пристосовуються до збоїв вузлів, використовуючи нечітку логіку для керування вмістом локального сховища даних і маршрутних таблиць. Порушення цілісності маршрутних таблиць, збої вузлів і з'єднань є штатними подіями для системи, а не виключними ситуаціями.

При цьому для виявлення збоїв використовуються тайм-аути протоколу TCP, порушення цілісності маршрутних таблиць виправляються за допомогою пересилання службового повідомлення, що фіксує ідентифікатори пройдених вузлів.

Кожен вузол мережі зберігає список покажчиків на найближчі вузли мережі (backpointers), до яких періодично відправляються повідомлення, які підтверджують робочий стан вузла і можливість маршрутизації повідомлень через нього [15].

Додатково, маршрутна таблиця кожного вузла містить список найближчих вузлів для відповідної позиції. При збої основного вузла позиції маршрутизація буде виконуватися через вузли з відповідного списку найближчих. При цьому сам вузол одразу не видаляється з маршрутної таблиці, а маркірується як тимчасово недоступний.

При використанні рівноправних виборів цільових вузлів серед списку найближчих, перевага віддається ближчим та менше завантаженим, що значно підвищує адаптацію маршрутів і призводить до реалізації базових алгоритмів розподілу навантаження [18].

Таким чином, протокол Tapestry реагує на мережні збої виключенням вузлів, виводить з обслуговування вузли, які часто виявляються в збійному стані, і швидко адаптує віртуальну топологію мережі до умов дійсного моменту.

Протокол Gnutella: У той час як модель Gnutella виявилася досить успішною з погляду підтримки і поширення, він не є такою з погляду розширюваності.

Кількість запитів і потенційних відповідей експоненційно зростає з кожним пересиланням запиту. Наприклад, якщо кожен вузол знає адреси ще двох вузлів мережі й обмеження кількості пересилань TTL дорівнює 7 (стандартне значення), то число відправлених запитів дорівнює 128, а кількість відповідей може бути набагато більше, у залежності від популярності шуканого файлу [2, 3].

Протокол Gnutella не надає механізмів для поліпшення живучості системи, головним припущенням у цій ситуації є те що до мережі буде підключена достатня кількість робочих станцій, так що запит буде розповсюджений досить далеко і шуканий файл буде знайдений. Однак, розподілена природа протоколу не надає таких гарантій.

Насправді, дослідники [29] показали що, тільки невелика частина користувачів Gnutella знаходяться в мережі достатній час для того щоб відповідати на запити інших користувачів. Тобто, проблемою в даному випадку є широка відомість системи, яка призводить до появи великої кількості користувачів, що намагаються знайти інформацію, не відповідаючи на запити інших користувачів. В такому випадку, вузол системи буде учасником тільки той час коли виконується його запит. В наступних версіях та аналогах ця проблема вирішується за допомогою введення системи рейтингів та статистики користування системою [3].

Відомо [30], що наприкінці 2000 року тільки 10% завантажень завершувалися вдало а в березні 2001 цей відсоток вже досягав 25%. Крім того, у тому ж джерелі зазначено, що найбільша кількість користувачів мережі Gnutella досягала 11000. Однак, цей факт не можна вважати доказом розширюваності або продуктивності мережі Gnutella при збільшенні числа учасників до сотень тисяч або мільйонів.

Платформа JXTA: Відзначимо, що доцільно розглянути ті покращення які були внесені в систему в новій версії та їх вплив на характеристики системи.

Перша версія функціонувала неефективно через ті вузли що епізодично знаходяться в системі – при додаванні та видаленні такого вузла генерується багато службових повідомлень, необхідних для зміни маршрутних таблиць інших вузлів. Для вирішення цієї проблеми, в мережі була виділена підмножина вузлів, які значний час знаходяться в системі. Ця підмножина формує опорну мережу, вузли якої виконують більший обсяг функцій та зберігають дані о топології мережі [27].

Як наслідок, живучість та розширюваність системи значно збільшуються, через зменшення кількості службових повідомлень які потрібні для підтримки маршрутних таблиць у актуальному стані і їх збереження на більш стабільних вузлах.

Крім того, задачі збільшення живучості також вирішуються за допомогою підтримки вузлами опорної мережі розподіленого індексу усіх елементів мережі. Це також значно підвищує розширюваність системи через меншу кількість ретрансляцій пошукового запиту, який буде оброблятися тільки опорною мережею.

Додатково, вузли опорної мережі можуть виконувати декілька спеціалізованих функцій – трансляція мережевого адресу, яка потрібна для об'єднання захищених корпоративних мереж та маршрутизація повідомлень.

Методи організації

У цьому розділі ми приведемо короткий огляд методів організації кожного з прикладів СРОС, розглянутих вище, з метою подальшої їх класифікації.

Програмна платформа Avaki: Система Avaki використовує для адресації два типи адрес: структуру даних, що містить тип адреси і безпосередньо дані адреси (OA, Object Address) та специфічну структуру даних, використовувану прикладними програмами (LOI, Legion Object ID).

При цьому, OA є залежним від типу об'єкта (типовим прикладом буде адреса IP і номер порту відповідного вузла мережі), LOI генерується самою платформою і відповідає заздалегідь визначеній угоді, дійсній для всієї системи в цілому [31].

При цьому на прикладному рівні виконуються виклики функцій об'єктів, що ідентифікуються за допомогою LOI, а відповідні об'єктні адреси (OA) генеруються самою системою Avaki та зберігаються в локальному реєстрі кожного вузла мережі.

Якщо ж у локальному реєстрі вузла немає відповідного зв'язування LOI-OA, тоді вузол формує запит до агента зв'язування (binding agent), для відповідного класу об'єкта. Якщо агент не може знайти відповідне зв'язування у своєму локальному реєстрі, агентом виконується відповідний запит для метакласса шуканого об'єкта. Так як будь-який об'єкт системи успадковується від спільного метакласса-предка, такий механізм виявлення необхідного зв'язування завершиться за кінцеву кількість переадресацій.

Таким чином, віртуальна структура Avaki є ієрархічною, вузли верхніх рівнів (агенти зв'язування) зберігають інформацію про структуру мережі. При цьому, зв'язування дублюються на кожному вузлі мережі і локальний реєстр являє собою локальну таблицю маршрутизації. Варто помітити, що переадресації запитів не відбувається - пошук відповідального вузла відбувається незалежно від доставки запиту.

Проект SETI@home: Ця система використовує централізоване серверне рішення для збереження даних і розподілу робіт. Отже, SETI@home не є розподіленою системою з точки зору розподілу даних, розподіляються тільки обчислення, при цьому між підзадачами немає залежностей, і, отже, немає необхідності в запитах і передачі даних між вузлами мережі. Така архітектура є застосовною тільки до задач у яких невеликі обсяги даних зв'язані зі значним обсягом цілком незалежних обчислень.

Розподілена система Groove: У Groove використовується гібридна топологія: симетричний розподіл певних функцій сполучається з використанням централізованих серверів для інших функцій. Для інтеграції симетричних служб із серверами часто використовуються агенти - робочі станції, основними функціями яких є резервування і синхронізація даних і ретрансляція з'єднань у випадку перебування клієнта в захищеній мережі.

При цьому слід зазначити, що централізовані сервіси Groove успадковують більшість недоліків властивих для класичних клієнт-серверних рішень, що призводить до малої розширюваності системи. У результаті Groove підтримує порівняно невеликі групи користувачів [2].

Протокол FreeNet: Головним принципом організації системи FreeNet є повна децентралізація. Користувачам потрібно самостійно знаходити інших учасників мережі. Єдиними операціями, підтримуваними системою FreeNet є додавання і пошук файлів у системі.

В обох випадках потрібно знайти вузол, що зберігає файл. У загальному випадку, вузли FreeNet формують мережу, у якій повідомлення передаються з метою знаходження такого вузла. Для маршрутизації таких повідомлень використовуються ключі файлів - FreeNet збирає файли зі схожими ключами на одному вузлі і переадресовує запити до вузлів адреси яких найбільш відповідають запитові.

Новий вузол інтегрується з мережею за допомогою виконання пошукового запиту, який представляє його іншим вузлам мережі. Перед тим як відіслати пошуковий запит, вузол має знайти як мінімум один інший вузол мережі, що може виконати такий запит. У цьому випадку немає визначених рішень цієї задачі - кожне рішення призводить до використання централізованих ресурсів [2].

Операції додавання працюють схожим чином - при цьому виконується пошук по ключу файлу і кожен вузол на шляху до необхідного вузла зберігає ключ, уміст файлу й адресу відповідального вузла. Це додає ефективності реплікації і зберігає правильність маршрутних таблиць інших учасників мережі.

Отже, FreeNet використовує чисту модель СРОС і базові механізми реплікації для підвищення розширюваності системи.

Протокол Tapestry: Цей протокол ґрунтується на методах, даних у роботі [32], присвяченій механізмові знаходження найближчих копій об'єктів у розподіленому обчислювальному середовищі за назвою Plaxton. Механізм маршрутизації Plaxton використовує локальні маршрутні таблиці, для ретрансляції повідомлень, що поступово наближає повідомлення до цільового вузла.

При цьому маршрутні таблиці розділяються на кілька рівнів, що відповідають різним ступеням зсуву щодо адреси поточного вузла. Для того, щоб знайти наступний вузол, використовується відповідний рівень маршрутної таблиці з найбільшим ступенем зсуву - це забезпечує логарифмічний ріст кількості ретрансляцій повідомлення в залежності від кількості вузлів мережі.

У цілому, підхід Plaxton надає базові механізми для нейтралізації збоїв вузлів і використовує дані про фізичну структуру мережі (локальність), що забезпечує пропорційність маршрутів (кількість ретрансляцій повідомлення відповідає відстані у фізичній мережі). Проте, існують і значні недоліки - Plaxton спирається на глобальні дані про всю мережу, використовує центральний вузол і не адаптується до змін запитів. [32, 33]

Так само як і в протоколі Plaxton, Tapestry використовує локальні маршрутні таблиці аналогічної структури. Але в той же час, для кожного елемента маршрутної таблиці Tapestry зберігається не одна адреса найближчого вузла, а список реплік даної адреси.

Крім цього, в Tapestry з'являється відповідність збережених вузлом елементів адресі самого вузла. Така угода дозволяє зменшити розміри локальних маршрутних таблиць і збільшити надійність механізму маршрутизації. Таким чином, у даній розподіленій системі визначена віртуальна топологія - вузли обчислювальної мережі відповідають області адресного простору.

Протокол Gnutella: Цей протокол об'єднує велику кількість різних програм, при цьому протокол не передбачає механізмів початкового знаходження інших вузлів мережі. Тому для початку роботи користувачеві потрібно взнати адресу іншого вузла, що підтримує протокол Gnutella, це можна зробити на веб-сторінці, на якій зберігається список користувачів Gnutella.

Якщо користувач хоче знайти файл, він відсилає запит тим користувачам, чиї адреси він знає. Ці користувачі можуть надати потрібний файл або ж переслати запит іншим користувачам. Запит можна пересилати до тих пір поки кількість пересилань не перевищує поле запита TTL [2].

Основна версія протоколу не передбачує механізмів розподілення навантаження чи адаптації віртуальної структури мережі і використовує повністю симетричну модель.

Платформа JXTA: У першій версії платформи обчислювальна мережа функціонально була повністю симетричною, хоча припускалося що кожен вузол може підтримувати тільки деякий набір операцій чи протоколів [34]. Друга версія платформи більш точно описує можливі функціональні відмінності вузлів та використовує ці відмінності для подальшої структуризації мережі та покращення характеристик її роботи.

Таким чином, був запропонований ієрархічний метод організації СРОС, який виділяє в симетричній обчислювальній мережі опорні вузли, що утворюють симетричну мережу вищого порядку. Така опорна мережа виконує функції, зв'язані зі глобальним представленням мережі. При цьому надійність роботи підвищується до рівня централізованих рішень, у той час як розширюваність системи залишається на рівні децентралізованих систем.

Порівняльний аналіз

Виділимо методи організації СРОС, та проаналізуємо їх взаємозв'язок з властивостями СРОС.

Головною характеристикою можна вважати наближеність до моделі, яка розглядає РОС як сукупність динамічних множин рівноправних вузлів з однаковим набором функцій. Якщо використовувана модель повністю відповідає такому визначенню, то модель називається чистою. При використанні централізованих серверів модель стає гібридною. По розгляді прикладів ми можемо зробити висновок, що централізація системи зменшує її розширюваність, у той час як повна децентралізація ускладнює погодження даних у масштабі усієї СРОС.

Іншою важливою характеристикою можна вважати структурованість системи, яка полягає в наявність різниці в підтримуваних функціях вузлів-учасників. Підкреслимо різницю між централізацією та структурованістю СРОС на прикладі платформи JXTA: множина виділених вузлів, що підтримують більше функцій, залишається динамічною і децентралізованою.

В випадку, коли структуризація СРОС призводить до виникнення множини супервузлів, СРОС прийнято називати ієрархічною. Зазначимо також, що в ієрархічних СРОС зустрічається розподілення вузлів нижнього рівня серед вузлів вищого рівня. При такій організації СРОС значно полегшується реалізація задач балансування навантаження, які вирішуються окремо на кожному супервузлі.

Слід зазначити, що використання гібридної моделі не заперечує структуризації: наприклад, СРОС Groove використовує додаткові вузли для зв'язку децентралізованої частини системи з серверами – агенти, які можна вважати вузлами що підтримують додатковий набір функцій. Проте, при використанні декількох

ієрархічних рівнів СРОС, які спираються на рівень, що складається з одного вузла, таку систему слід рахувати частково централізованою і, як наслідок, такою, що використовує гібридну модель.

Варто врахувати необхідність централізованого контролю системи, що необхідний компанії-власникові у випадку комерційного використання системи. Типовим прикладом можна вважати комерційну систему Groove, яка залежить від присутності сервера керування ліцензіями.

Крім того, важливим виявляється наявність в системі віртуальної топології, та її використання при маршрутизації повідомлень, що обмежує кількість ретрансляцій повідомлення по логічних каналах і значно підвищує ефективність роботи системи. Крім того, досить важливим є можливість оптимізації віртуальної топології з точки зору відповідності до апаратної структури мережі, як це зроблено в протоколі СРОС Tapestry.

Нижче представлена таблиця 1, в якій наводиться короткий опис прикладів СРОС, розглянутих в даній статті.

Таблиця 1. Основні характеристики розглянутих прикладів СРОС

Назва	Метод організації	Наявність віртуальної топології	Механізм маршрутизації повідомлень	Примітки
Avaki	ієрархічна, гібридна – ієрархія спирається на центральний вузол	відсутня	відсутній	мала динамічність структури мережі
SETI@HOME	гібридна модель, неструктурована	відсутня	відсутній	повна ізольованість розподілюваних задач
Groove	гібридна, із використанням агентів	відсутня	відсутній	мала розширюваність
FreeNet	чиста, неструктурована	присутня	стохастичний	важкість дослідження
Tapestry	чиста, неструктурована	присутня, адаптується до апаратної топології	з використанням віртуальної топології	нечітка обробка збоїв
Gnutella	чиста, неструктурована	відсутня	передача усім зв'язаним вузлам	мала ефективність
JXTA	чиста, ієрархічна, з використанням опорної мережі	присутня	з використанням віртуальної топології	інтегровані індексація та пошук

Виконавши аналіз вищенаведених систем, ми пропонуємо наступну модель їх розвитку.

Спочатку були створені системи з централізованим індексуванням даних або робіт та розподіленим і повністю незалежним їх збереженням або виконанням, прикладами таких систем являються SETI@Home та Napster.

Після цього, актуальними стали чисті моделі, які робили СРОС застосовними для задач в взаємними залежностями та більшими запитами з точки зору розширюваності, прикладами таких систем є Gnutella та FreeNet.

Як наслідок підвищення ступеня децентралізації, стали актуальними проблеми покращення живучості – це призвело до появи спеціалізованих алгоритмів пошуку та маршрутизації повідомлень та їх використання в ієрархічних СРОС, таких як JXTA, а також в системах з віртуальною топологією, таких, як Chord, Tapestry або P-Grid.

Паралельно відбувалась еволюція розподілених, так званих Grid-систем, які характеризуються менш динамічною структурою мережі і схожою архітектурою, в сторону абстрагування від апаратної структури мережі та автоматичної адаптації до змін її структури. Прикладами систем, що знаходяться на межі класифікації peer-to-peer та Grid систем, можна вважати Avaki та SETI@home.

Висновки

В роботі визначені поняття оверлейної мережі та віртуальної топології для СРОС, які використовуються для опису логічної структури обчислювальної мережі, визначення ієрархії вузлів та схем розподілу ресурсів. На основі проведеного в роботі аналізу та класифікації сучасних розподілених обчислювальних та інформаційних систем відмітимо, що найбільш актуальними характеристиками є використання віртуальної топології та можливість ієрархічної організації. Використання віртуальної топології при маршрутизації повідомлень зменшує кількість їх ретрансляцій по логічних каналах оверлейної мережі та значно підвищує ефективність роботи основних механізмів та розподіленої системи в цілому.

Найбільш актуальним напрямком подальших досліджень автори вважають дослідження та моделювання різних ієрархій СРОС. Першорядними задачами в цьому напрямку виступає моделювання роботи СРОС з різними віртуальними топологіями та механізмами утворення ієрархії вузлів, а також експериментальне дослідження впливу вищезазначених характеристик на показники роботи розподіленої системи.

Література

1. *K. G. Cofman, A. M. Odlyzko*, “Growth of the Internet” Section 7 “Disruptive Innovations”, pp. 27-31. // Academic Press, 2002.
2. *Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins ,Zhichen Xu*, “Peer-to-peer computing” // HP Laboratories, Palo Alto, 2002.
3. *Andrew Oram (editor), Nelson Minar, and Clay Shirky*, “Peer-to-peer: Harnessing the power of disruptive technologies” // O'Reilly Press, 2001.
4. *Ian Foster, Carl Kessekman, and Steven Tuecke,* “The anatomy of the Grid: Enabling scalable virtual organizations.” // The International Journal of Supercomputer Applications, 2001.
5. *Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz,* “DataCutter: Middleware for filtering very large scientific datasets on archival storage systems.” // In Proceedings of the 2000 Mass Storage Systems Conference, College Park, MD, March 2000.
6. *Ossama Othman, Douglas C. Schmidt,* “Optimizing Distributed system Performance via Adaptive Middleware Load Balancing.” // In Proceedings of the Workshop on Optimization of Middleware and Distributed Systems, Snowbird, Utah, June 2001. ACM SIGPLAN.
7. *Zhao, B. Y., Huang, L., Stribling, J., Joseph, A. D., and Kubiatowicz, J. D.,* “Exploiting routing redundancy via structured peer-to-peer overlays” // In Proc. of ICNP (Atlanta, GA, Nov 2003), IEEE, pp. 246--257.
8. *K. Aberer, A. Datta, M. Hauswirth,* “Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins.” // Self-* Properties in Complex Information Systems, "Hot Topics" series, LNCS, 2005.
9. *A. Veystel,* “There is no P-2-P market… But there is a market for P-2-P.” // Aberdeen Group Presentation at the P2PWG, May 2001.
10. *C. Shirky*, “What is P2P… and what isn’t.” // O’Reily network, 2001.
11. “Information Technology – Open Systems Interconnection – Basic Reference Model” // International Standard ISO/IEC 7498-1, 1994
12. *I. Foster, C. Kesselman, J. Nick, S. Tuecke.* “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration” // Technical report, The Globus Project, Jan. 2002.
13. *K. Ranganathan and I. Foster*, "Identifying Dynamic Replication Strategies for a High Performance Data Grid" // Presented at International Workshop on Grid Computing, Denver, CO, 2001.
14. *M. Mitzenmacher*, “On the analysis of randomized load balancing schemes” // In Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 292--301, Newport, Rhode Island, June 22--25, 1997.
15. *B. Zhao, J. Koubiatowicz, and A. Joseph*, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing" // Technical Report UCB/CSD-01-1141, C.S. Dept. U.C. Berkeley, Apr. 2001.
16. *D. Eppstein and Z. Galil,* “Parallel algorithmic techniques for combinatorial computation” // Ann. Rev. Computer Science, 988.
17. *Blumofe, R. D., and Park, D. S.,* “Scheduling large-scale parallel computations on network of workstations” // 3rd IEEE International Symposium on High-Performance Distributed Computing. Aug. 1994.
18. *B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz*, “Exploiting routing redundancy using a wide-area overlay” // Technical Report UCB/CSD-02-1215, University of California, Berkeley, 2002.
19. *Grimshaw, A. S.,Wulf, W. A., French, J. C.,Weaver, A. C., Reynolds JR., P. F.*, “Legion: The Next Logical Step Toward a Nation-wide Virtual Computer.” // UVa CS Technical Report CS-94-21, June 8, 1994.
20. *Grimshaw, A.*, “Easy to use Object-oriented Parallel Programming with mentat” // IEEE Computer, pp39-51, May 1993.
21. *Leigh, P. Benyola, P.,* 2001. “Future Developments in Peer Networking. Equity Research” // White Paper, Raymond James & Associates, INC.
22. *Suthar, P. Ozzie, J.,* “The Groove Platform Architecture” // Groove Networks Presentation, 2000.
23. “Microsoft, Groove Networks to Combine Forces to Create Anytime, Anywhere Collaboration” // Microsoft Press, May 2005, available at http://www.microsoft.com/presspass/features/2005/mar05/03-10GrooveQA.mspx
24. *Clarke, I., Sandberg, O, Wiley, B., Hong, T.W.*, “Freenet: A Distributed Anonymous Information Storage and Retrieval System.” // Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, ed. by H. Federrath. Springer: New York. 2001.
25. *Clarke, I.I., Miller, S.G. Wong, T.W., Sandberg, O., Wiley, B.*, “Protecting free expression online with FreeNet” // IEEE Internet Computing 6(1):40-49. January-February, 2002.
26. “Tapestry: Infrastructure for Fault-resilient, Decentralized Location and Routing” // Available at http://p2p.cs.ucsb.edu/chimera/, 2005.
27. *L.Gong.*, “Project JXTA: A Technology Overview” // Sun Microsystems, Inc. whitepaper, Apr. 2001.
28. *Sing Li*, “JXTA 2: A high-performance, massively scalable P2P network” // IBM developerWorks articles, published on 11 Nov 2003.
29. *Adar, E., Huberman, B.*, “Free Riding on Gnutella” // First Monday, vol 5, no 10, October 2000. Available at http://www.firstmonday.dk/issues/issue5_10/adar.

30. *Kotzen, M.*, "Dramatic Improvements in the Gnutella Network Since Late 2000" // Available at www.limewire.com/index.jsp/net_improvements, 2001.
31. "Legion Developer Manual" // Available at http://legion.virginia.edu/documentation/Developer_1.8.pdf, 2005.
32. C. Greg. Plaxton, Rajmohan Rajaraman, and Andrea W. Richa., "Accessing nearby copies of replicated objects." // In proceedings of ACM SPAA. ACM, June 1997.
33. *Rory Bland, Darren Caulfield, Emmet Clarke, Alan Hanley, Eamon Kelleher*, "P2P networks" // Networks and Telecommunications Research Group, Department of Computer Science, Trinity College Dublin, 2005.
34. "JXTA protocol specification" // Available at http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html#proto-prp