ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ УПРАВЛІННЯ

УДК 504.064

[1]І.М. Доманецька, [2]М.С. Петрушенко, [1]О.В. Федусенко, [1]А.О. Федусенко

[1] *Київський національний університет будівництва і архітектури, Київ*
[2] *Компанія EPAM, Київ*

ІМІТАЦІЙНА МОДЕЛЬ ФУНКЦІОНУВАННЯ ВУЗЛА ПІРИНГОВОЇ МЕРЕЖІ

Висвітлено питання застосування теорії масового обслуговування для моделювання роботи вузла пірингової мережі з метою визначення параметрів, які будуть використані для створення програми-файлообмінного клієнта для протоколу Кадемлія.

Ключові слова: пірингові мережі, P2P, комунікаційні протоколи децентралізованих файлообмінних мереж, протокол Кадемлія, розподілена хеш-таблиця, відстань між вузлами, СМО

ИМИТАЦИОННАЯ МОДЕЛЬ ФУНКЦИОНИРОВАНИЯ УЗЛА ПИРИНГОВОЙ СЕТИ

Посвящено вопросам применения теории массового обслуживания для моделирования работы узла пиринговой сети с целью определения параметров, которые будут использованы для создания программы-файлообменного клиента для протокола Кадемлия.

SIMULATION MODEL OPERATION UNIT PEERING NETWORKS

The work is devoted to the application of queuing theory to model the work node peer to peer network in order to determine the parameters that will be used to create software for file-sharing client protocol Kademliya.

Постановка проблеми

Сьогодні пірингові мережі настільки розвинулися, що WWW вже не є найбільшою інформаційною мережею за ресурсами і породжуваним інтернет-трафіком. Відомо, що трафік, обсяг інформаційних ресурсів (у байтах), кількість вузлів пірингових мереж, якщо їх розглядати в сукупності, нічим не поступаються мережі WWW. Більш того, трафік пірингових мереж складає 70% всього інтернет-трафіку! При цьому можна відзначити два дуже важливих аспекти: по-перше, про пірингові мережі дуже мало пишуть у науковій літературі, а по-друге, проблеми пошуку та вразливості пірінгових мереж, як найбільшої «білої плями» сучасних комунікацій, поки залишаються відкритими.

Тому однією з актуальних задач сьогодення є створення файлообмінного клієнта – програми для пошуку і завантаження файлів будь-якої природи, що базується на протоколі Кадемлія.

Аналіз останніх досліджень і публікацій

Комп'ютерні мережі типу peer-to-peer (або P2P) засновані на принципі рівноправності учасників і характеризуються тим, що їх елементи можуть зв'язуватися між собою, на відміну від традиційної архітектури, коли лише окрема категорія учасників, яка називається серверами, може надавати певні сервіси іншим[1].

В чистій «peer-to-peer» мережі не існує поняття “клієнтів” або “серверів”, лише рівні вузли, які одночасно функціонують як клієнти та сервери по відношенню до інших вузлів мережі. Ця модель мережевої взаємодії відрізняється від клієнт-серверної архітектури, в якій зв'язок відбувається лише між клієнтами та центральним сервером (рис.1). Така організація дозволяє зберігати працездатність мережі за будь-якої конфігурації доступних її учасників. Проте практикується використання P2P мереж, які все ж таки мають сервери, але їх роль полягає вже не у наданні сервісів, а у підтримці інформації з приводу сервісів, що надаються клієнтами мережі.

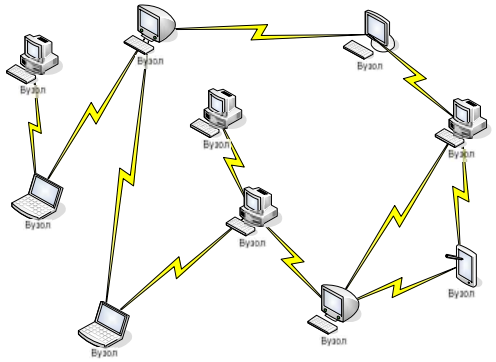

Рис. 1. Архітектура P2P

В P2P системі автономні вузли взаємодіють з іншими автономними вузлами. Вузли є автономними в тому сенсі, що не існує загальної влади, яка може контролювати їх. В результаті автономії вузлів, вони не можуть довіряти один одному та покладатися на поведінку інших вузлів, тому проблеми масштабування та надмірності стають більш важливими ніж у випадку традиційної архітектури.

Сучасні P2P-мережи набули розвитку завдяки ідеям, пов'язаним з обміном інформацією, які формувалися у руслі того, що кожен вузол може надавати та отримувати ресурси, які надаються будь-якими іншими учасниками. У випадку мережі Napster, це був обмін музикою, в інших випадках це може бути надання процесорного часу для пошуку інопланетних цивілізації або ліків від раку.

На практиці пірингові мережі складаються з вузлів, кожен з яких взаємодіє лише з деякою підмножиною інших вузлів (через обмеженість ресурсів). На відміну від архітектури «клієнт-сервер» така організація дозволяє зберігати працездатність мережі за будь-якої кількості і будь-якого поєднання доступних вузлів.

В роботі розглядається робота пірингової мережі, що базується на протоколі Кадемлія [2; 3].

Кадемлія – це комунікаційний протокол для мереж P2P. Він є однією з багатьох версій розподілених хеш-таблиць (DHT).

Мережа Кадемлія характеризується трьома константами, які називають *alpha*, *B*, та *k*. Перша та остання є стандартними. Друга була введена тому, що деякі реалізації мережі Кадемлія використовують різні довжини ключів.

- *Alpha* – це мале число, що характеризує рівень паралелізму мережевих викликів, звичайно дорівнює трьом.
- *B* – це розмір у бітах ключа, що використовується для ідентифікації вузлів та для того, щоб зберігати та отримувати дані. У базовому варіанті Кадемлії це число рівне 160, що дорівнює довжині дайджесту SHA-1 (хешу).
- *k* – це максимальна кількість контактів, що можуть бути збережені у комірці, звичайно ця кількість дорівнює 20.

Також зручно ввести деякі інші константи, що не вказані у офіційній специфікації.

- *tExpire = 86400s* – це час, після проходження якого пара ключ-значення застаріває, це так званий період життя (TTL) з моменту публікації.
- *tRefresh = 3600s* – якщо протягом даного часу до деякої комірки доступ не здійснювався, то необхідно її оновити.
- *tReplicate = 3600s* – це інтервал між послідовними викликами процедури реплікації, коли вузол повинен повторно опублікувати всю базу даних ключів, що у нас є.
- *tRepublish = 86400s* – після проходження цього часу оригінальний публікатор повинен повторно опублікувати пару ключ-значення, що у ній він зацікавлений.

Той факт, що *tExpire* і *tRepublish* є рівними, вводить, так звану, умову гонки. Повідомлення STORE для тих даних, що були опубліковані, може надійти як раз у той момент, коли ці ж дані вже застаріли, тому насправді необхідно виконувати пересилку даних. Нормальна реалізація могла б використовувати такі значення *tExpire,* що були б набагато більшими ніж *tRepublish*.

Мережа Кадемлія складається із деякої кількості взаємодіючих вузлів, що спілкуються один з одним і зберігають один для одного інформацію. Кожен вузол має ідентифікатор *nodeID,* псевдо-унікальне двійкове число, що ідентифікує даний вузол у мережі. У мережі, деякий блок даних, так зване *значення (value),* може бути ототожнене з двійковим числом однакової фіксованої довжини, тобто *ключем (key)* для даного значення. Таким чином, вузол, якому потрібне *значення,* виконує пошук цього значення на вузлах, яких він вважає найближчими до *ключа*, асоційованого з цими даними.

Ідентифікатори вузлів є двійковими даними довжиною *B*=160 біт. У базовій версії Кадемлії кожен вузол обирає свій власний ідентифікатор за допомогою деякої невизначеної псевдовипадкової процедури. Дуже важливим є момент, щоб ідентифікатор вузла був рівномірно розподілений, оскільки від цього залежить надійність функціонування мережі. У той час, як протокол не вимагає цього, але цілком можливо, що набагато краще використовувати один і той самий ідентифікатор вузла кожен раз, як цей вузол під'єднується до мережі, аніж генерувати його кожен раз заново для поточної сесії.

Дані, що зберігаються або отримуються з мережі Кадемлія, повинні також мати довжину ключа, що дорівнює *B*. Ці ключі також мають бути рівномірно розподіленими. Є декілька шляхів, щоб цього досягти найбільш поширений підхід – розрахувати хеш, такий, наприклад, як дайджест SHA-1, від заданого значення.

Усі операції у мережі Кадемлія базуються на використанні бітової операції Виключаючого Або *(xor)*. Відстань (distance) між будь-якими окремо взятими ключами або ідентифікаторами вузлів *x* та *y* визначається як:

$$dis\tan ce(x, y) = x \oplus y.$$

Результат отримується за допомогою бітової операції Виключаючого Або над кожним бітом обох операндів. Кадемлія представляє ключі (включаючи ідентифікатори вузлів) у вигляді так званих чисел *big-endian.* Це означає, що молодший байт у масиві байтів, що представляють ключ, є найбільш значущим, тому якщо два ключа є близькими один до одного, то початкові байти у відстані будуть нульовими [4].

Кожен вузол Кадемлія упорядковує список своїх *контактів,* тобто вузлів, про яких знає даний вузол, у спеціальні *комірки.* Ці комірки відомі як *k-buckets.*

Комірки упорядковані за відстанню між вузлами і контактами у цій корзинці. Наприклад, для деякої комірки *j,* де *0<=j<k,* гарантується, що node:

$2^j \le distance(\quad,\quad) < 2^{(j+1)}$.

Якщо адресний простір дуже великий, то це означає, що комірка з номером нуль буде містити тільки одне можливе значення, що відрізняється від ідентифікатора вузла лише старшим бітом, і для усіх практичних цілей ніколи не використовується, хіба що для тестування. З іншого боку, якщо усі ідентифікатори вузлів рівномірно розподілені, то імовірно, що половина усіх вузлів буде потрапляти у діапазон, що покривається коміркою *B-1=159.*

У специфікації Кадемлії вказується, що числу *k* надається таке значення, що є мало ймовірним, що усі контакти у межах даної комірки стануть недоступними упродовж однієї години. Кожен, хто намагатиметься розрахувати цю імовірність, повинен взяти до уваги деякі політики, що надаються перевагу довгоживучим вузлам перед короткоживучими.

Контакт – це принаймні трійка значень:

- ідентифікатор вузла;
- IP-адреса;
- UDP-порт.

У комірках контакти відсортовані за часом останнього спілкування. Тобто, ті контакти, з якими вузол спілкувався нещодавно, розміщуються ближче до кінця списку, незалежно від того, який вузол розпочав обмін повідомленнями.

Як тільки вузол отримує повідомлення від іншого вузла, він оновлює відповідну комірку. Якщо такий контакт вже знаходиться у списку, то він переміщується у кінець списку. Якщо дана комірка вже повна, то даний вузол опитує контакт, що знаходиться у голові списку. Якщо контакт, до якого був направлений запит, не відповідає протягом деякого заданого проміжку часу, то він видаляється, а свіжий контакт вставляється у хвіст списку. У протилежному випадку новий контакт відхиляється. У великій, високо навантаженій мережі можлива ситуація, що у процесі очікування вузлом відповіді від старого контакту у голові списку, може надійти запит від контакту, що знаходиться поза діапазоном комірки, що розглядається. Ця ситуація є найбільш імовірною для комірки із номером 159, оскільки вона покриває половину адресного простору Кадемлії. Поведінка вузла у цьому випадку не обговорена у специфікації, і є потенційною можливістю для відкриття DDOS-атаки.

Дослідження показали, що вузли схильні до поділу на дві групи: короткоживучі та довгоживучі. Ця політика оновлення комірок віддає перевагу довгоживучим вузлам, що забезпечує деякий рівень захисту щодо деяких типів мережевих атак, таких як DDOS, Sybil і т. д.

У оригінальній специфікації Кадемлії вказується, що протокол Кадемлії складається з чотирьох примітивів, або віддалених викликів процедур *(RPC)* [5; 6].

- *PING.* Це повідомлення вимагає посилку повідомлення PING до іншого вузла, який повинен на нього відповісти. Ця процедура має подвійний ефект: отримувач цього повідомлення повинен оновити комірку, що відповідає ідентифікатору відправника, а також, якщо має місце відповідь вузла, то відправник повинен оновити свою комірку відповідно до адреси відправника. Усі повідомлення повинні містити деякий ідентифікатор, що надається відправником і повинен міститися у відповіді. Цей ідентифікатор є псевдовипадковим числом довжиною *B (160 bit).*
- *STORE.* Відправник цього повідомлення надає ключ та блок даних і вимагає, щоб отримувач зберіг дані і зробив їх доступними для отримання за заданим ключем. Ця операція є примітивною, а не ітеративною. Не зважаючи на те, що це не обговорено, зрозуміло, що дане повідомлення повинне включати крім ідентифікатора повідомлення принаймні дані, що повинні бути збережені (та їхню довжину), а також відповідний ключ. Оскільки транспортним протоколом є UDP, то повідомлення-запит повинне включати як мінімум ідентифікатор відправника, а повідомлення-відповідь повинне включати ідентифікатор отримувача. Відповідь на будь-яке повідомлення повинна включати результат операції. Наприклад, у повідомленні *STORE* можлива ситуація, коли отримувач не зміг зберегти дані, тому що не вистачило місця у пам'яті або ж помилки вводу-виводу.
- *FIND_NODE.* Цей примітив включає 160-бітний ключ. Отримувач цього повідомлення повертає не більше *k* трійок (адреса, порт, ідентифікатор) для контактів, які є найближчими до ключа. Отримувач повинен повернути трійки, якщо це можливо. Він може повернути менше значень,

якщо число *k* є більшим, ніж загальна кількість контактів у його комірках. Це примітивна, а не ітеративна операція.

- *FIND_VALUE.* Цей примітив включає 160-бітний ключ. Якщо відповідне значення зберігається отримувачем, то відповідні дані повертаються відправнику. У іншому випадку цей виклик аналогічний до попереднього, і повертається список трійок.

Пошук починається з вибору певної кількості ідентифікаторів з комірок вузла, що є найближчими до ключа, який шукається. З цього списку вибираються *alpha* контактів, до яких посилаються запити. Кожен контакт, який знаходиться у мережі і є досяжним, звичайно повертає *k* контактів. Потім вузол додає ці відповіді до свого списку пошуку і процедура пошуку повторюється. Ця послідовність пошуку повторюється доти, доки будуть знаходитися вузли, що ближчі до заданого, або буде знайдений шуканий ідентифікатор.

Якщо упродовж деякого проміжку часу не здійснювалося ніяких пошуків у діапазоні будь-якої комірки (наприклад, однієї години), то вузол обирає будь-який ідентифікатор у цьому діапазоні і виконує оновлення, тобто ітеративно посилає повідомлення FIND_NODE із цим ідентифікатором.

Вузол приєднується до мережі таким чином:

1. Якщо він ще не має ідентифікатора, то він генерує його.
2. Він вставляє ідентифікатор будь-якого вузла, що йому відомий, у відповідну комірку.
3. Виконує ітеративний пошук значення (це значення – його власний ідентифікатор), у той же час оновлюючи відповідні комірки.

Правила реплікації даних:

- Дані зберігаються у ітеративному сховищі, що повинно виконувати реплікацію даних, що у ньому містяться, на найближчі вузли.
- Кожен вузол повинен виконувати реплікацію даних, що у ньому містяться, кожну годину.
- Кожен оригінальний публікатор має публікувати дані кожні 24 години.

Для деякої комірки, що покриває діапазон $[2^i, 2^{i+1})$, визначимо *індекс* комірки таким, що дорівнює *i.* Визначимо також *глибину, h,* вузла такою, що дорівнює 160-*i,* де *i* – це найменший індекс непорожньої комірки. Визначимо *висоту комірки* для вузла *y*, що дорівнює індексу такої комірки, у яку *x* може вставити *y* мінус індекс найменш значущої комірки вузла *x*. Оскільки ідентифікатори вузлів вибрані довільно, то звідси випливає, що вибірки з високою імовірністю будуть рівномірно розподілені. Тому можна сказати, що висота будь-якого вузла буде у межах сталої $\log k$ для мережі із *n* вузлів. Більше того, висота комірки найближчого вузла до заданого ідентифікатора *ID* буде у межах $\log k$.

Припустимо, що кожна комірка кожного вузла включає принаймні один контакт, якщо існують вузли у відповідному діапазоні. З цим припущенням ми покажемо, що пошук вузла є процедурою коректною і потребує логарифмічного часу. Уявимо, що найближчий вузол до ідентифікатора ID має глибину *h.* Якщо ніяка із *h* найбільш значущих комірок не є пустою, то процедура пошуку знайде вузол, що наполовину ближчий (відстань до якого на один біт коротша) на кожному кроці, і тому потребуватиме $h - \log_{10} k$ кроків. Якщо одна з комірок вузла є пустою, то можливо, що цільовий вузол знаходиться у діапазоні пустої комірки. У цьому випадку заключні кроки не зменшать відстань у два рази. Тим не менш, пошук буде продовжено точно так, якби біт у ключі, що відносився до порожньої комірки, мав протилежне значення. Отже, алгоритм пошуку завжди буде повертати найближчий вузол за $h - \log_{10} k$ кроків. Більше того, як тільки найближчий вузол знайдено, параметр одночасності встановлюється у значення *k*. Кількість кроків, що потрібна для визначення *k-1* найближчих вузлів, буде не більшою, ніж висота комірки найближчого вузла у *k-тому* найближчому вузлі.

Тепер розглянемо відновлення пари ключ-значення. Коли пара ключ-значення публікується, то вона розміщується на *к* вузлах, що є найближчими до ключа. Також пари ключ-значення повторно публікуються кожну годину. Оскільки навіть вузли, що тільки зайшли до мережі (найменш надійні) з імовірністю 0,5 залишаться у мережі хоча б на одну годину, то через годину пара ключ-значення все ж таки буде присутня на одному з *к* вузлів, що є найближчими до ключа, з імовірністю 1-2^k. Ця властивість не порушується вставкою нових вузлів, що є близькими до ключа, оскільки як тільки подібні вузлі вставляються, вони контактують зі своїми найближчими сусідами для того, щоб наповнити свої комірки, і тому отримають близькі до них пари ключ-значення, які потрібно зберегти. Звичайно, якщо *к* найближчих до ключа вузлів відмовлять, і пара ключ-значення не була ніде збережена, то ця пара буде втрачена мережею.

Виклад основного матеріалу

Вузли Kademlia взаємодіють між собою шляхом надсилання та приймання конфігураційних та інформаційних повідомлень. Конфігураційними є ті повідомлення, які несуть інформацію щодо віддалених вузлів, а також тих, які містять

інформацію щодо стану вузла-відправника. Інформаційними є повідомлення, що включають дані про пари ключ-значення, тобто списки адрес та портів вузлів, що містять необхідну нам або віддаленим вузлам інформацію. Як вже було сказано вище, крім системи, що забезпечує взаємодію вузлів Кадемлія між собою, повинна також бути реалізована підсистема, власне завантаження потрібних даних на комп'ютер користувача. Тобто, спочатку за допомогою підсистеми обслуговування Кадемлія, даний вузол отримує необхідну інформацію про вузли, на яких потенційно можуть міститися файли, а потім за допомогою підсистеми завантаження файли завантажуються на комп'ютер користувача.

Як бачимо, взяття участі окремого вузла у кооперативному обміні файлами є доволі складним процесом, тому у цій ситуації необхідно забезпечити максимальну надійність у процесі передачі даних. Звичайно, цілісність передачі власне самих даних (файлових фрагментів) можливо відслідкувати шляхом використання контрольних сум за алгоритмом SHA-1. Цей алгоритм ставить у відповідність будь-якому масиву байт унікальне 160-бітне число, тобто число такого самого розміру, який вже використовується як для ідентифікації самих вузлів Кадемлія, так і для ідентифікації пар ключ-значення. Проте, оскільки у системі для забезпечення взаємодії між вузлами використовуються повідомлення, (при цьому кожне повідомлення може бути змінної довжини та містити різні види даних), доцільно організувати розрізнені повідомлення у так званий комунікаційний протокол передачі даних [7].

Розглянемо схему взаємодії потоків в ході функціонування вузла мережі. Рішення призначити окремий потік, що займатиметься виключно прийняттям повідомлень, є очевидним. Оскільки методи прийняття повідомлень є блокуючими операціями, то має бути присутнім спеціальний виділений потік. Задачі, якими займається потік MessageReceiver (рис. 2), повинні виконуватися дуже швидко, тому головною ціллю було спростити потік так, як це тільки можливо. Причина такої оптимізації полягає в тому, що буфер, який прив'язаний до сокету, має фіксований та обмежений розмір. Якщо цей буфер не спустошувати достатньо часто, то існує можливість його переповнення. У цьому випадку інформація, що посилається іншими вузлами, просто буде відхилена. Звичайно, така ситуація повинна мати правильне рішення. Оскільки приймаючий потік не повинен виконувати затратні операції, то він буде готовий продовжити прослуховування вхідного сокету дуже швидко. Завдяки цьому підходу, імовірність переповнення вхідного буфера істотно зменшується. Ситуація, що щойно була описана, не є теоретичною, а може трапитися у процесі виконання програми. Припустимо, що деяка достатньо велика кількість вузлів бере участь в обміні повідомленнями, тобто комірки кожного вузла є більшістю повними. Далі, вузол виконує пошук деякого ідентифікатора. Ідентифікатор має бути таким, що вузол, який опитаний одним із перших, є найближчим. Як наслідок, контакти опитаних вузлів усі знаходитимуться на більшій відстані. У процедурі пошуку, це призводить до запуску повторного опитування найближчих вузлів, що ще не були опитані.

Як припускалося, один із вузлів, що були опитані першими, є найближчим до шуканого. Якщо так, то імовірно, що інші опитані спочатку вузли також є близькими, оскільки вони походять з однієї комірки. Тому цілком можливо припустити, що перший раунд пошуку виявив три найближчі вузли. Для продовження пошуку, 20-3=17 вузлів, що ще не опитувалися, будуть опитані. Кожен з них буде повертати 20 контактів, оскільки у мережі знаходиться достатньо багато вузлів.

Кожна така відповідь має розмір 580 байт. 17 таких повідомлень – це майже десять кілобайт даних. Оскільки усі вузли опитувалися одночасно, то відповіді будуть надходити швидко, одна за одною. У комп'ютері, що використовувався для реалізації, розмір вхідного буфера складає 8192 байти. Тому інформація буде втраченою, якщо вхідні повідомлення не будуть швидко оброблятися.

Як пояснювалося вище, кожне отримане повідомлення потребує посилання відповіді. Спочатку потік MessageReceiver повинен був приймати повідомлення, розділяти повідомлення на запити та відповіді, а потім одразу відповідати на запити. Але такий підхід виявився непридатним до практичного застосування. По-перше, у випадку, якщо повідомлення, що надійшло, виявилося запитом, наприклад, примітивом FIND_NODE, то відповідь на це повідомлення повинна містити список найближчих вузлів до вказаного у повідомленні ідентифікатора відправника. Але ця процедура вимагає пошуку найближчих ідентифікаторів у відповідних комірках, що взагалі є нетривіальною операцією. Тому принцип збереження потоку MessageReceiver як найбільш швидкодіючим буде порушений.

По-друге, і це є набагато гіршим, посилка відповідей на повідомлення це короткотермінова операція блокування, оскільки вона повинна очікувати на відповідь після посилки запиту. Тому упродовж очікування відповіді програма буде заблокованою, що унеможливить усю її подальшу роботу. Тому така ситуація є неприйнятною.

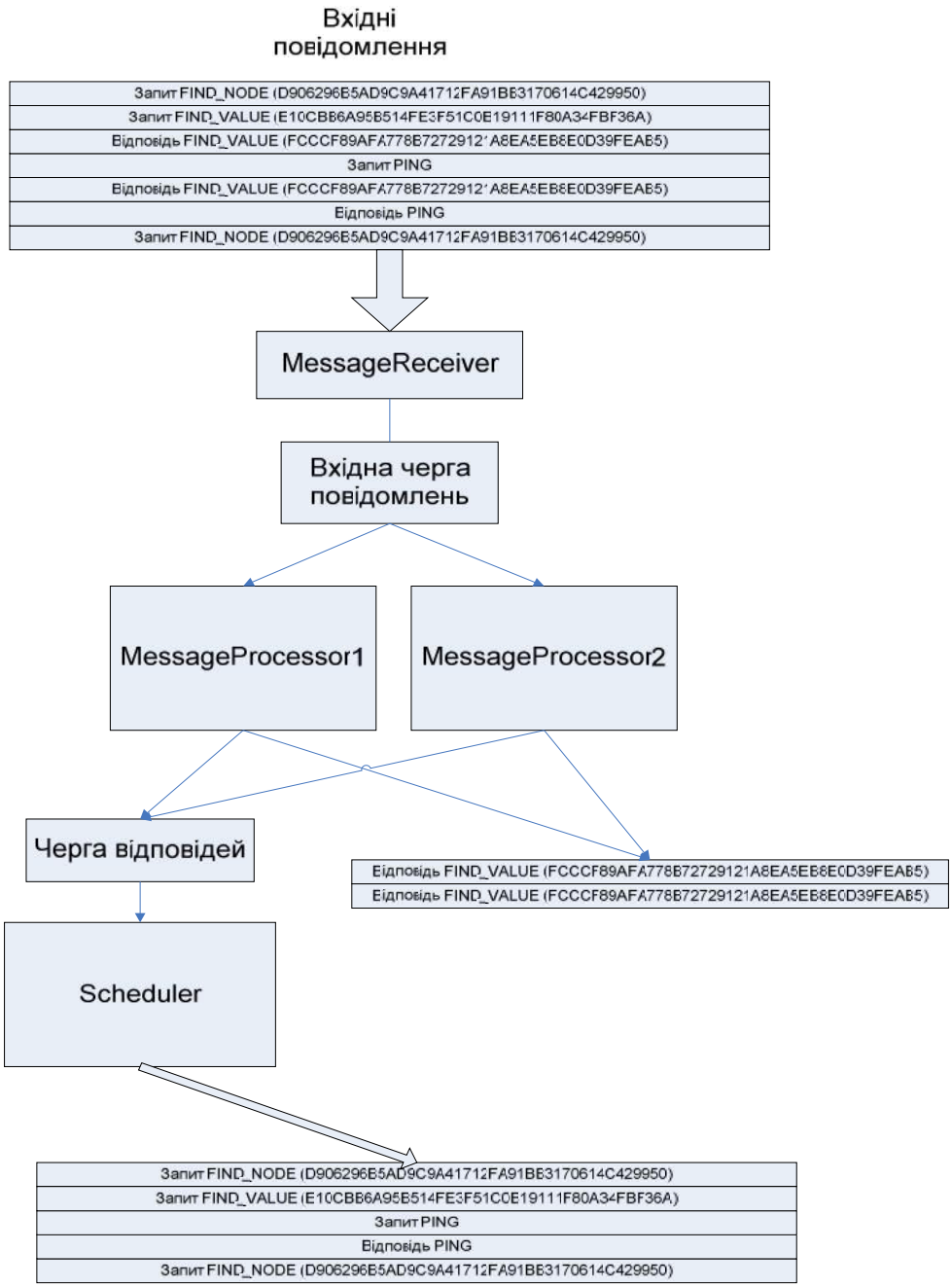

Рис. 2. Схема взаємодії потоків

Рішення полягає у розробці додаткового потоку (або, що краще, кількох потоків), які будуть паралельно з потоком, що приймає повідомлення, виконувати їх сортування на запити і відповіді, а також посилати відповіді на прийняті запити від інших вузлів. Упродовж виконання програми, деякі повідомлення можуть надходити безпосередньо одне за одним, тому програма повинна працювати якнайшвидше, адже вузол, що надіслав запит, зовсім не зацікавлений у довічному очікуванні відповіді.

Формулювання мети статті

Як було описано вище, у даній програмі використовується багато потоків, які активно взаємодіють між собою. Також була показана доцільність використання окремих потоків для виконання специфічних завдань, таких як прийняття взаємного блокування. Тому необхідно провести аналіз, що стосується як роботи системи у цілому, так і її окремих частин.

З огляду на те, що у системі виконуються операції з обробки заявок, то її зручно аналізувати як багатофазну систему масового обслуговування.

Використовуючи модель одноканальної системи масового обслуговування (СМО) визначимо розміри сокетного буфера для накопичення повідомлень, що очікують на опрацювання.

Розглянемо найпростішу СМО з очікуванням – одноканальну систему, до якої надходить потік заявок з інтенсивністю λ, інтенсивністю обслуговування μ (тобто у середньому безперервно зайнятий канал буде видавати λ/μ обслугованих заявок за одиницю часу. Заявка, що надійшла у момент, коли канал зайнятий, стає у чергу і очікує обслуговування.

Припустимо спочатку, що кількість місць у черзі обмежена числом m, тобто, якщо заявка прийшла до системи, коли у черзі стоїть m заявок, то вона залишає систему не обслуженою. У подальшому, спрямувавши m у нескінченність, ми отримаємо характеристики одноканальної СМО без обмежень довжини черги.

Будемо нумерувати стани СМО за числом заявок, що знаходяться у черзі (як тих, що уже обслуговуються, так і тих, що очікують обслуговування):

S_0 – канал вільний;

S_1 – канал зайнятий, черга відсутня;

S_2 – канал зайнятий, одна заявка стоїть у черзі;

S_k – канал зайнятий, k-1 заявок стоять у черзі;

S_{m+1} – канал зайнятий, m заявок знаходиться у черзі.

Граф переходів системи показаний на рис. 3. Усі інтенсивності потоків подій, що переводять систему по стрілках зліва направо, дорівнюють λ, а справа наліво – μ. Дійсно, по стрілках зліва направо систему переводить потік заявок (як тільки прийде заявка, система переходить у наступний стан), справа же наліво – потік «звільнень» зайнятого каналу, що має інтенсивність μ (як тільки буде обслугована чергова заявка, канал або звільниться, або зменшиться кількість заявок у черзі). Зображена на рис. 3 схема являє собою схему розмноження та загибелі. Запишемо вирази для граничних ймовірностей станів:

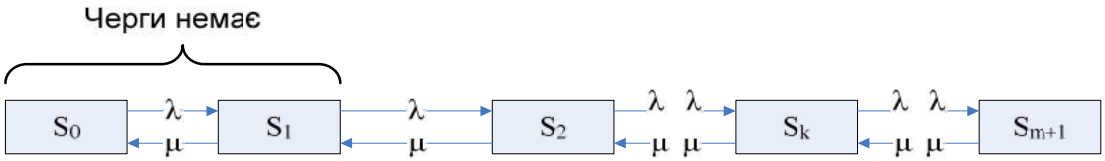

Рис. 3 Граф переходів одноканальної СМО з очікуванням

$$\begin{cases} p_k = \left(\frac{\lambda}{\mu}\right)^k p_0; (k = 1,2,...,m+1) \\ p_0 = \frac{1}{1+\left(\frac{\lambda}{\mu}\right)+\left(\frac{\lambda}{\mu}\right)^2+...+\left(\frac{\lambda}{\mu}\right)^{m+1}}, \end{cases} \quad (1)$$

або із використанням: $\rho = \frac{\lambda}{\mu}$:

$$\begin{cases} p_k = \rho^k p_0; (k = 1,2,...,m+1); \\ p_0 = \frac{1}{1+\rho+\rho^2+...+\rho^{m+1})} = \\ = (1+\rho+\rho^2+...+\rho^{m+1})^{-1}. \end{cases} \quad (2)$$

Останній рядок у (2) містить геометричну прогресію з першим членом *1* і знаменником p, звідки отримуємо:

$$p_0 = \frac{1}{1-\rho^{m+2}/(1-\rho)} = \frac{1-p}{1-p^{m+2}}, \quad (3)$$

у зв'язку з чим граничні імовірності приймають вигляд (4).

Вираз (3) є справедливим тільки при ρ<1; при (ρ=1 вона дає невизначеність типу 0/0):

$$\begin{cases} p_0 = \frac{1-p}{1-p^{m+2}} \\ p1 = \rho p_0 \\ p2 = \rho^2 p_0 \\ \dots \\ p_{m+1} = \rho^{m+1} p_0 \end{cases} . \quad (4)$$

Сума геометричної прогресії зі знаменником *ρ=1* дорівнює *m+2*, і в цьому випадку:

$$p_0 = \frac{1}{m+2}.$$

Визначимо характеристики СМО: імовірність відмови $P_{відм}$, відносну пропускну спроможність q, абсолютну пропускну спроможність A, середню довжину черги $\overline{r}$, середню кількість заявок, пов'язаних із системою $\overline{k}$, середній час очікування

у черзі $t_{оч,}$ середній час перебування заявки у СМО $\overline{r}_{смо}$.

Імовірність відмови. Очевидно, що заявка отримує відмову тільки і у тому випадку, коли канал зайнятий і усі т-місць у черзі також:

$$P_{відм} = p_{m+1} = \frac{\rho^{m+1}(1-\rho)}{1-\rho^{m+2}}. \quad (5)$$

Відносна пропускна спроможність:

$$q = 1 - P_{відм} = 1 - \frac{\rho^{m+1}(1-\rho)}{1-\rho^{m+2}}. \quad (6)$$

Абсолютна пропускна спроможність:

$A = \lambda q$.

Середня довжина черги. Знайдемо середню кількість $\overline{r}$-заявок, що знаходяться у черзі, як математичне сподівання дискретної випадкової величини *R* – числа заявок, що знаходяться у черзі:

$$\overline{r} = M[R].$$

З імовірністю p_2 у черзі стоїть одна заявка, з імовірністю p_3 – дві заявки, взагалі з імовірністю p_k у черзі стоять k-1 заявок, і так далі, звідки:

$$\overline{r} = M[R] = \sum_{k=2} \rho^k p_0 = \rho^2 p_0 \sum_{k=2}^{m+1} (k-1)\rho^{k-2} = \\ = \rho^2 p_0 \sum_{k=1}^{m} k\rho^{k-1}. \quad (7)$$

Оскільки $k\rho^{k-3} = \frac{d\rho^k}{d\rho}$, то суму у (7) можна трактувати як похідну по P_0 від суми геометричної прогресії:

$$\sum_{k=1}^{m} k\rho^{k-1} = \left(\sum_{k=1}^{m} \rho^k \right)' = \left(\frac{\rho - \rho^{m+1}}{1-\rho} \right)' = \\ = \frac{1-(m+1-m\rho)\rho^m}{(1-\rho)^2}.$$

Підставляючи даний вираз у (7) та використовуючи P_0 із (4), отримуємо:

$$\overline{r} = \frac{\rho^2(1-(m+1-m\rho)\rho^m}{(1-\rho)(1-\rho^{m+2)}}. \quad (8)$$

Середнє число заявок, що знаходиться у системі. Отримаємо далі формулу для середньої кількості заявок, що пов'язані з системою (як тих, що стоять у черзі, так і тих, що знаходяться на обслуговуванні). Оскільки $\overline{k} = \overline{r} + \overline{w}$, где $\overline{w}$ – середнє число заявок, що знаходяться під обслуговуванням, а k відомо, то залишається визначити $\overline{w}$. Оскільки канал у нашій системі один, то кількість заявок що обслуговується, може дорівнювати 0 (з імовірністю P_0) або 1 (з імовірністю $1- P_0$), звідки:

$$\overline{w} = 0 \cdot p_0 + 1 \cdot (1 - p_0) = \frac{\rho - \rho^{m+2}}{1-\rho^{m+2}}$$

і середня кількість заявок, що пов'язані зі СМО, дорівнює:

$$k = r + \frac{\rho - \rho^{m+2}}{1-\rho^{m+2}}.$$

Середній час очікування заявки у черзі. Позначимо його як $t_{оч}$, якщо заявка приходить у систему у який-небудь момент часу, то з імовірністю P_0 канал обслуговування не буде зайнятий, і заявці не доведеться стояти у черзі (час очікування дорівнює нулю). З імовірністю P_1 заявка прийде у систему протягом обслуговування іншої заявки, але перед нею не буде черги, і заявка буде очікувати початку свого обслуговування упродовж часу $1/\mu$ (час обслуговування однієї заявки). З імовірністю P_2 у черзі перед заявкою, що розглядається, буде стояти ще одна, і час очікування буде рівний $2/\mu$ і так далі.

Якщо ж $k=m+1$, тобто коли нова заявка застає канал обслуговування зайнятим і m заявок у черзі, то у цьому випадку заявка не стає у чергу і не обслуговується, тому час очікування рівний нулю. Звичайно, у цьому випадку, якщо говорити про нашу систему прийняття та обробки повідомлень, можна сказати, що запит віддаленого вузла або його відповідь на наш запит буде втраченою. Середній час очікування буде такий:

$$t_{оч} = p_1 \frac{1}{\mu} + p_2 \frac{2}{\mu} + ... + p_k \frac{k}{\mu} + ... + p_m \frac{m}{\mu},$$

якщо підставити сюди вирази для ймовірностей, отримаємо:

$$t_{оч} = p_0 \rho \frac{1}{\mu} + p_0 \rho^2 \frac{2}{\mu} + ... + p_0 \rho^k \frac{k}{\mu} + ... \\ ... + p_0 \rho^m \frac{m}{\mu} = \frac{\rho}{\mu} \frac{(1-(m+1-m\rho)\rho^m)}{(1-\rho)(1-\rho^{m+2})}. \quad (10)$$

Тут використані співвідношення (7), (8) (похідна геометричної прогресії), а також P_0 із (4). Порівнюючи цей вираз із (8), помічаємо, що, інакше кажучи, середній час очікування дорівнює середній кількості заявок у черзі, що поділена на інтенсивність потоку заявок:

$$t_{оч} = \frac{1}{\rho\mu}\overline{r} = \frac{\overline{r}}{\lambda} \quad (11)$$

середній час перебування заявки у системі. Позначимо $t_{смо}$ – математичне сподівання випадкової величини – часу перебування заявки у СМО, яке складається із середнього часу очікування у черзі ($t_{оч}$) і середнього часу обслуговування

($t_{обсл}$). Якщо завантаження системи складає 100%, то очевидно, $t_{обсл} = 1/\mu$, у протилежному випадку:

$$t_{обсл} = q/\mu,$$

звідки:

$$t_{смо} = t_{оч} + t_{обсл} = \frac{r}{\lambda} + \frac{q}{\mu}.$$

Тепер розрахуємо довжину черги для нашої системи. Виходити будемо з того, що максимальна кількість необроблених повідомлень обмежена розміром сокетного буферу, який за замовченням складає 8192 байти. Розмір середнього повідомлення складає приблизно 450 байтів. Звідси маємо:

8192/450 = 18 повідомлень.

Розрахована кількість – це максимальна кількість, що може поміститися у сокетному буфері без попередньої обробки застосуванням. Окремий потік виймає повідомлення із буфера та переміщує повідомлення до вхідної черги необроблених повідомлень. Швидкість обробки повідомлень, визначена експериментальним способом, складає приблизно 2500 повідомлень на секунду. Таймаут очікування відповіді на запит у нашій системі складає три секунди, відповідно, за три секунди може бути оброблено:

3*2500 = 7500 повідомлень, що складає 1.5 мегабайтів оперативної пам'яті.

Отже, максимальна довжина черги складає 7500 повідомлень, при досягненні якої всі інші повідомлення будуть відкидатися.

Висновки

Кадемлія – це протокол віртуальної мережі, створений для функціонування повністю децентралізованих файлообмінних мереж. Його основна відмінність від інших протоколів у тому, що він незалежний від центральних серверів. У безсерверній мережі кожен користувач є вузлом, через нього проходять пошукові запити і службова інформація. Усі користувачі мережі є серверами і рівні між собою за рангом.

Інтерпретуючи результати моделювання можемо стверджувати, що під час створення програми-клієнта файлообмінної мережі P2P на базі протоколу Кадемлія, доцільно передбачити розмір сокетного буферу для накопичення повідомлень не менший, ніж на 1,5 Мб.

В подальшому планується програмна реалізація проекту. У ролі платформи для розробки була вибрана мова програмування Java SE 1.6 [8; 9]. Такий вибір пояснюється тим, що, по-перше, програма, розроблена із використанням Java, є більш надійною та стабільною порівняно із аналогічними застосуваннями, розробленими із використанням традиційних мов, таких як C++ та Object Pascal. По-друге, таке застосування буде кросплатформеним, тобто може бути виконане на будь-якій платформі (апаратній та програмній), на якій встановлено віртуальну машину Java. Крім того, бібліотека класів Java містить потужні засоби для роботи із базами даних, комп'ютерними мережами із використанням різних протоколів, а також фундамент для створення користувацького інтерфейсу, вигляд якого не залежить від платформи, на якій виконується застосування [10].

Список літератури

1. TCP/IP sockets in Java : practical guide for programmers / Kenneth L. Calvert, Michael J. Donahoo. – 2nd ed.

2. Maintaining High Bandwidth under Dynamic Network Conditions Dejan Kosti´c, Ryan Braud, Charles Killian, Erik Vandekieft, James W. Anderson, Alex C. Snoeren and Amin Vahdat _Department of Computer Science and Engineering University of California, San Diego – електронний документ.

3. Petar Maymounkov, David Mazi`eres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proceedings of IPTPS, Cambridge, MA, USA, March 2002. http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf

4. Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. Technical Report UW-CSE-01-06-02, University of Washington, Department of Computer Science and Engineering, July 2001.

5. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1.

6. Daniel Stutzbach, Reza Rejaie. Improving Lookup Performance over a Widely-Deployed DHT. In Proceedings of IEEE INFOCOM, Barcelona, Spain, April 2006. http://www.cs.uoregon.edu/~reza/PUB/infocom06-kad.pdf

7. B. Cohen. Incentives Build Robustness in BitTorrent. In Proc. 1st Workshop on Economics of Peer-to-Peer Systems (P2PECON), 2003.

8. Кей С. Хорстманн, Гари Корнелл. Java 2. Библиотека профессионала, том 1. Основы = Core Java 2, Volume I — Fundamentals. — 8-е изд. — М.: Вильямс, 2008. — 816 с.

9. Кей С. Хорстманн, Гари Корнелл. Java 2. Библиотека профессионала, том 2. Тонкости программирования = Core Java 2, Volume II — Advanced Features. — 8-е изд. — М.: Вильямс, 2008, 4 кв. — 992 с.

10. The Java EE 5Tutorial For Sun Java System Application Server 9.1. - SunMicrosystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A. – електронний документ.

Стаття надійшла до редколегії 10.10.2012

Рецензент: д-р техн. наук, проф. С.В. Цюцюра, Київський національний університет будівництва і архітектури, Київ.