УДК 681.3

О.В. Новицький

СЕРВІС-ОРІЄНТОВАНІ, РОЗПОДІЛЕНІ СИСТЕМИ РЕАЛЬНОГО ЧАСУ В ЕЛЕКТРОННИХ БІБЛІОТЕКАХ (ЕБ)

Розглядаються проблеми побудови сервіс-орієнтованих Веб-додатків реального часу. Зокрема для прогнозування часу виконання високонавантажених додатків, за умови стаціонарності навантаження, пропонується використовувати модель Ерланга з чергою нескінченної довжини. Нами також запропоновано новий алгоритм адаптивного прогнозування часу виконання. Як практичну реалізацію, апробовано наші пропозиції на системі виконання виводу класифікації онтологій, та планування часу виконання.

Вступ

У бібліотеках побудованих на основі сервіс-орієнтованого підходу виникають нові сервіси, які потребують математичного опису певних процесів. Такий клас сервісів, вимагає з одного боку, значних обчислювальних ресурсів, з іншого – мають виконатися за прогнозований час. У порівнянні з класичними технологіями GRID, в яких задача яка ставиться в чергу може там знаходитися необмежений час. Розглядатимемо клас задач, які мають бути виконані за прогнозований час. Такий сервіс ми пропонуємо як сервіси конвертації формату документів. Це не єдине застосування запропонованого нами підходу і сфери його застосування, можуть бути будь-які сервіс-орієнтовані Веб-додатки реального часу. В даній роботі розглядатимемо питання прогнозування часу виконання на прикладі задачі класифікації онтологій.

Сервіси реального часу в ЕБ

Сучасні ЕБ мають опрацьовувати значний обсяг інформаційних ресурсів. Поряд з цим, виникає все більше і більше нових сервісів, що мають на меті задовольнити потреби користувачів. При розробці та підтримці складних систем доцільно використовувати сервіс-орієнтований підхід (SOA) до архітектури програмного забезпечення. Існує ряд визначень щодо SOA [1, 2]. Однак більшість з них описуються як компонований додаток, що складається з слабозв'язаних сервісів, запущених на різних вузлах і обмін інформацією між ними відбувається за допомогою передачі повідомлень. Сервіс-орієнтована архітектура, передбачає повторне використання сервісів. Одним із недоліків для сервіс-орієнтованої архітектури, є часові затримки які мають кілька причин. І для якісної оцінки, критерій доступності, часової затримки на відповідь. SOA припускає розподілених обчислень. Компоненти сервісу, як правило, розташовані в різних контейнерах, найчастіше на різних машинах. Потреба в обміні повідомлень по мережі збільшує час відгуку. Типові мереж, що використовуються для SOA, такі як Інтернет, не гарантує наперед визначені затримки. Таким чином, SOA не вважається підходящим рішенням для систем реального часу, де своєчасність є строгою вимогою. SOA створює проблеми в режимі реального часу системи, де затримка є критично важливою вимогою. Для часто використовуваних сервісів, з великою кількістю запитів, формується черга FIFO чи LIFO. Така ситуація може мати значний вплив на затримку, хоча може бути стохастично передбачено [3].

Однак є ряд задач, які потребують використання SOA та мають обмеження за часом. Ці обмеження можуть бути як жорсткі, так і слабкі. Зокрема, при керуванні інформацією, виникає потреба її додаткової обробки, причому для складних завдань користувачами системи можуть висуватися часові обмеження до виконання відповідних сервісів. Для ЕБ ми виділяємо наступні функціональні сервіси:

– сервіс пошуку інформації у семантичному контенті, ця група сервісів, виконує пошук у середовищі Semantic Web. В основу пошуку покладається мова запитів SPARQL та вивід на знаннях. Ці дві задачі потребують значних обчислювальних ресурсів [4];

– сервіси рекомендації інформаційних ресурсів на основі пов'язаності посилань та додаткових відомостей на основі онтологій інтересів користувачів;

– сервіси побудови візуалізації графів у Semantic Web [5];

– сервіси конвертації форматів представлення електронних документів.

Для таких та інших подібних сервісів є важливою задача прогнозування часу виконання сервісу. Це пов'язано з тим, що кінцевим користувачам, при роботі з сервісами реального часу необхідно забезпечити певний рівень зручності роботи з програмним забезпеченням. Одним із елементів якості програмного забезпечення є надання інформації користувачу про час закінчення довготривалої операції. Це дозволить користувачу отримувати відповідну інформацію, про те що система дійсно функціонує і завершить виконання ресурсоємної операції за прогнозований час.

Ми розглядаємо лише такі сервіси, які є високо навантаженими, однопоточними, та часто викликаються. Даний метод не підходить до звичайних Інтернет додатків, які працюють на Веб-серверах. Тобто ми розглядаємо таку задачу, коли один запит може обслуговуватися виключно одним вузлом, і виконання такого запиту потребує значних обчислювальних ресурсів.

Основна ціль – прогнозування часу виконання запиту за найменшу кількість ітерацій.

Формула Ерланга С та модель $M/M/m/\infty$

Системам реального часу необхідно задовольняти жорсткі обмеження за часом, ці обмеження – наслідок тих процесів які системи підтримують. Загалом існування верхньої межі за часом є необхідною умовою функціонування систем реального часу. Одне точне значення цього часу визначити не можливо, проте можливо гарантувати, що процес є скінченним і обов'язково завершиться. Визначити нижню межу виконання за часом є простішим для визначення, якщо відомо всі передумови.

Ми припускаємо, що в режимі реального часу система складається з низки завдань, які реалізують необхідну функціональність. Випадок з найкоротшим часом виконання називається найкращий випадок за часом виконання (BCET best-case execution time). Найдовший час називається найгірший випадок за часом виконання (WCET – worst-case execution time). У більшості випадків простір станів занадто великий, щоб вичерпно вивчити всі можливі страти і тим самим визначити точний гірший і кращий випадок за часом виконання.

Сьогодні в більшості поширений спосіб оцінки межі часу виконання – це вимір часу виконання завдання для підмножини можливих стрес-тестів. Це визначає мінімальні та максимальні спостережувані часи виконання. Ці методи не дозволяють точно визначити BCET та WCET і тому є небезпечними для жорстких систем реального часу. Цей метод часто називають аналіз динамічного часу.

Оскільки наша модель є з нескінченною чергою, то формула Єрланга В не підходить, тому розглядатимемо формулу Єрланга С. У нашій моделі у випадку коли запит не може бути оброблений негайно, то він поміщається в чергу очікування необмеженої довжини, коли один із сервісів звільнюється, то він автоматично бере завдання з черги. Якщо черга порожня, то сервіс знаходиться в стані очікування і може негайно задовольнити запит. Формула Erlang C [6], в оригінальній формі, визначеній як функція двох змінних: число сервісів N і навантаження A. На основі їх значення можна визначити ймовірність $P_c(N, A)$, що вхідний запит не буде обслуговуватися відразу, йому доведеться чекати своєї черги:

$$P_C(N,A)=\frac{\frac{A^N N}{N!(N-A)}}{\sum_{i=0}^{N-1}\frac{A^i}{i!}+\frac{A^N N}{N!(N-A)}}, \quad (1)$$

де

$$A=\frac{\lambda}{\mu}. \quad (2)$$

Тепер використаємо зв'язок між навантаженням A (2), та середньою кількістю запитів λ за проміжок часу і середньою кількістю запитів μ, оброблених за проміжок часу. Далі визначаємо величину η, яка представляє навантаження на один сервіс [7]:

$$\eta=\frac{\lambda}{N\mu}. \quad (3)$$

В зв'язку з чим на основі, (2), (3), ми можемо переписати (1) наступним чином:

$$P_C(N,\eta)=\frac{\frac{\left(\frac{\lambda}{\mu}\right)^N N}{N!\left(N-\frac{\lambda}{\mu}\right)}}{\sum_{i=0}^{N-1}\frac{\left(\frac{\lambda}{\mu}\right)^i}{i!}+\frac{\left(\frac{\lambda}{\mu}\right)^N N}{N!(N-\frac{\lambda}{\mu})}}=$$

$$=\frac{\frac{(N\eta)^N}{N!(1-\eta)}}{\sum_{i=0}^{N-1}\frac{(N\eta)^i}{i!}+\frac{(N\eta)^N}{N!(1-\eta)}}. \quad (4)$$

У формулі (4) визначається ймовірність запитів у системі масового обслуговування $M/M/m/\infty$, які будуть поміщені в чергу, якщо в систему надходить більше запитів ніж m [6].

Використовуючи основну форму формули Ерланга C (1), знаючи значення параметра A (максимальне навантаження) та за відомого числа сервісів N можна обчислити ймовірність $P_C(N,\eta)$. Через складність аналітичного визначення цих невідомих параметрів, використовуються чисельні методи.

Тоді можна розрахувати середній час очікування черги W (середній час очікування виклику в черзі перед готовністю сервісу опрацювати запит):

$$W=\frac{P_C}{\mu(N-A)} \quad (5)$$

і застосування теореми Литтла [6] та формули (2) отримаємо середню кількість запитів у черзі очікування, наступним чином:

$$Q=\lambda W=\frac{\lambda}{\mu(N-A)}P_C=\frac{A}{(N-A)}P_c. \quad (6)$$

Для розрахунку параметра оцінки сервісу (Grade of Service – GoS) (відсоток запитів, які оброблені або для яких виділено сервіс до певного часового порогу AWT – допустимий режим очікування) за відомим значенням AWT [8]:

$$GoS=1-P_C\cdot e^{-\mu(N-A)\cdot AWT}. \quad (7)$$

Середня кількість запитів у системі K (а також середньої чисельності зайнятих сервісів) [9]:

$$K=N\eta+\frac{\eta}{1-\eta}\cdot P_C=$$

$$=A+\frac{A}{(N-A)}\cdot P_C=A+Q. \quad (8)$$

З формули (8) на основі теореми Литтла отримуємо середнє значення часу T, що визначає час виконання запиту:

$$T=\frac{K}{\lambda}=\frac{A+Q}{\lambda}=\frac{1}{\mu}+\frac{P_C}{\mu(N-A)}. \quad (9)$$

Адаптивний алгоритм прогнозування часу виконання для систем реального часу

Оскільки дані є ймовірними, то для покращення обслуговування та прогнозування часу виконання запиту, нами запропонований адаптивний алгоритм прогнозування часу виконання запиту Вебсервісом. У його основі покладено ідею,

що час виконання етапу виконання може бути зпрогнозований на основі формули Ерлангу *C*, водночас якщо параметри очікування відхиляються в ту чи іншу сторону від прогнозованого, на основі вагових коефіцієнтів відбувається корекція таймерів затримки. Це пояснюється тим, що реальні потоки запитів не завжди є стаціонарними, але в деякій мірі наближеними до них. Тому виникає необхідність введенням додаткових коефіцієнтів, які з одного боку давали б можливість застосовувати до псевдо-стаціонарних потоків, теорію стаціонарних потоків, а з іншого – корегували б величину таймера затримки внаслідок ймовірнісного значення часу виконання запиту. Вагові коефіцієнти розраховуються на показниках складності завдання, яке виконується та враховується параметрами середовища. Вихідними даними для вагових коефіцієнтів становить статистична інформація про виконання параметрів попередніх завдань.

$$w = w(p_i, h_j), \quad i = 1...n,\ j = 1...k, \tag{10}$$

де w – функція обчислення вагового коефіцієнта; t – час виконання запиту; p_i – параметр, що характеризує i-ту характеристику складності завдання; h – характеристика, що вказує на обчислювальну потужність системи.

Для побудови функції w необхідно визначити кожен параметр складності та обчислювальної потужності.

Визначити кожен параметр складності завдання та обчислювальної потужності системи в абсолютних величинах не можливо, але можливо визначити відносне значення даних параметрів.

Нехай маємо n завдань, кожне завдання описується набором параметрів p_i, h_j для зручності сукупність цих параметрів будемо позначати s_k. Для того щоб оцінити складність кожного завдання, оцінюємо складність відносного всіх інших завдань і записуємо цей результат a_{ij} у матрицю $A_{n\times m}$, де $m = n + k$:

$$A_{n\times m} = \begin{cases} a_{ij} = \dfrac{s_i}{s_{\max}} - \text{відношення параметрів складності}, \\ a_{ii} = 1 - \text{параметри ідентичні самі по собі}, \\ a_{ij} = \dfrac{s_{\min}}{s_i} - \text{зворотність параметрів}. \end{cases}$$

Матрицю будуємо за правилом, якщо $t(s_k) \xrightarrow{s_j > s_i} 0$ то $s_{\max} = \max(s_k)$, якщо $t(s_k) \xrightarrow{s_j < s_i} 0$ то $s_{mix} = \min(s_k)$.

Функція w задається у вигляді суми параметрів a_{ij}:

$$w_j = \sum_{i=0}^{n} a_{ij}. \tag{11}$$

Загальний час виконання запиту буде визначатися наступним чином:

$$T_{complite} = T(1 + \delta),$$

$$\delta = \begin{cases} w - \text{запит в черзі} \\ 0 - \text{запит оброблено} \end{cases}.$$

Таким чином на кожному етапі ітераційного процесу визначення часу виконання запиту, ми постійно подовжуємо час на величину, що відповідає складності виконання завдання. На основі даних параметрів запропоновано алгоритм, який дозволяє автоматично балансувати навантаження між вузлами Веб-сервісів та прогнозувати час виконання кожного завдання на певному вузлі.

У випадку неадекватного вибору базису для оцінювання складності завдання, пропонуємо вводити зональну складність. Остання дозволяє покращувати прогнозування часу виконання при неповному базисі параметрів, що визначають складність завдання. Зональна складність виконання – це зона на якій розподіл випадкової величини лежить в інтервалі $[2/3\sigma - x + 2/3\sigma]$, де σ – стартове значення випадкової величини, що визначає складність. Потім для зональної складності

будується середнє геометричне за складністю d_{zone}. Ознака належності процесу до певної зональної складності, є $2/3/d_{zone} \le d_i \le 2/3 \bullet d_{zone}$, де d_i – складність i-го процесу для якого треба спрогнозувати час виконання. Додавання кожного процесу в вибірки вливає на d та на час виконання. Всі процеси в рамках зональної складності розглядаються як окремий потік випадкових процесів.

```
    global task_list, count_node,
count_request, time_for_request,
time_for_response, list_parameters
    {
    while (task_complite)
    {
    list_free_nodes[]=get_free_node();
        if (is_have_free_node) // перевірка
вільних вузлів
     {
                if
(count(task_list)/count(list_free_nodes)>1)
//кількість завдань перевищує кількість
вузлів
            {
        foreach (list_nodes as
node_id=>node)
                    {
                        send_task_async_to_no
de(node, task_list[node_id]) //відправка
асинхронного запиту на виконуючий вузол
timer[task_list[node_id])]= adaptive
(list_task, count_node);
                        }
                }
        elseif
(count(task_list)/count(list_free_nodes)<=1)
//кількість вузлів перевищує кількість зав-
дань
            {
                foreach (task_list as
task_id=>task)
                    {
                    send_task_async_to_node(
                list_free_nodes[id_task])
                //відправка асинхронного за-
                питу на виконуючий вузол
timer[task_id]= adaptive (list_task,
count_node);;
                        }
            }
  }
elseif (count(list_task)>0 and
count(list_free_nodes)==0) //перевірка умо-
ви відсутності вільного вузла
            {

        wait=sum(timer_for_each_task); //при
відсутності вільних вузлів необхідно
збільшити час очікування,
                        }
                }
sleeps= timer+ timer* complexity;
return sleeps
}
}

function calc_erlang (time_dif_issue,
count_node, time_send, count_node,
count_task) //обрахунок таймера затримки
на основі рівняння Ерлагу С та теореми
Литтла (9)
{
timer=erlang_c(time_dif_issue, count_node,
time_send, count_node, count_task);
return timer;
}
function calc_complexity (list_parameters)
//обрахунок складності завдання на основі
(11)
{
complexi-
ty=calculation_complexity(list_parameters);
return complexity;
}

function adaptive (list_task, count_node){
        dif=calc_complexity (list_task)
//обраховуємо складність задання
        if (in_to(dif, issue_dif) // пошук зо-
нального розподілу по складності
        $timer=calc_erlang(time_dif_issue,
        count_node, time_send, count_node,
        count_task)
                )
else {
            create_new_dif_zone();
        //створення нової зональної склад-
        ності
time=null;
            }
return time
}
```

Методика моделювання сервіс-орієнтованого Веб-додатка реального часу

Електронна бібліотека має ряд сервісів які забезпечують зберігання, обробку, доступ до інформації. Переважна більшість сервісів при виконанні не мають жорсткого обмеження за часом виконання, і для Веб-сервісів граничний час WCET визначається налаштуванням сервера додатків. Водночас для деякого класу сервісів надзвичайно важливим є виконання за певний проміжок часу.

Для побудови ми змоделювали реальну систему, яка призначена для вирішення задач класифікації та категоризації. Сервіси судження для Дескриптивної логіки, такі як тестування категоризації і класифікації, як правило використовуються для перевірки відповідності кількості баз знань на основі оригінальної онтології [10]. Здійснимість класу, наприклад, зводиться до перевірки послідовності бази знань, в яких один індивідуал є екземпляром цього класу. Таблиці суджень виконано як тести для перевірки послідовності. Труднощі побудови такої моделі виникають з двох джерел. По-перше, часто є велика кількість різних можливих конструкцій, які можуть бути в моделі. На основі таблиць необхідно проаналізувати кожну з цих конструкцій, перш ніж зробити висновок, що база знань має або не має модель. По-друге, моделі, побудовані для таблиць суджень можуть бути дуже великими, навіть для відносно невеликої онтології.

HermiT це прикладне програмне забезпечення для виконання суджень над Дескриптивною логікою, засновано новою архітектурою, яка охоплює вищевказані джерела складності. HermiT реалізує гіпертабличні обчислення, що значно знижує кількість можливих моделей, які мають бути розглянуті [11].

Оскільки HermiT виконує судження над TBox та ABox, ми використали наступні метрики:

- кількість класів;
- кількість властивостей об’єктів;
- кількість властивостей даних;
- кількість індивідуалів;
- кількість логічних аксіом;
- кількість аксіом онтологій анотацій;
- кількість аксіом анотацій сутностей;
- кількість логічних аксіом для класів;
- кількість логічних аксіом для підкласів;
- кількість аксіом еквівалентності;
- кількість аксіом диз’юнкції;
- кількість аксіом об’єднання диз’юнкції.

Як виявили експериментальні дослідження, згадані параметри не в повній мірі відображають час який буде необхідний на проведення суджень. Це пояснюється самим алгоритмом виконання судження, оскільки між часом виконання та запропонованими нами метриками, нема прямої функціональної залежності.

Оскільки основна задача – це прогнозування часу виконання сервісу, то основними компонентами нашої моделі є:

– середня кількість вхідних запитів на виконання сервісу за одиницю часу λ;

– середній час обробки запиту сервісом $\frac{1}{\mu}$;

– число сервісів N;

– загальна протяжність трафіку в секундах.

Для того щоб змоделювати необхідні граничні параметри, нами побудовано реально діючу систему. Моделюючий трафік у цій системі був стаціонарним без наслідків.

Вхідним набором є набір онтологій для яких виконувались тести класифікації та категоризації за допомогою HermiT. Внаслідок побудовано залежність часу виконання від складності, з загальною кількістю фацетів в онтології їх налічується понад 340 тис.

З рис.1 видно, що ця залежність є складною, і визначення точного часу виконання, може бути співставленням з часом виконання тестів класифікації та категоризації.

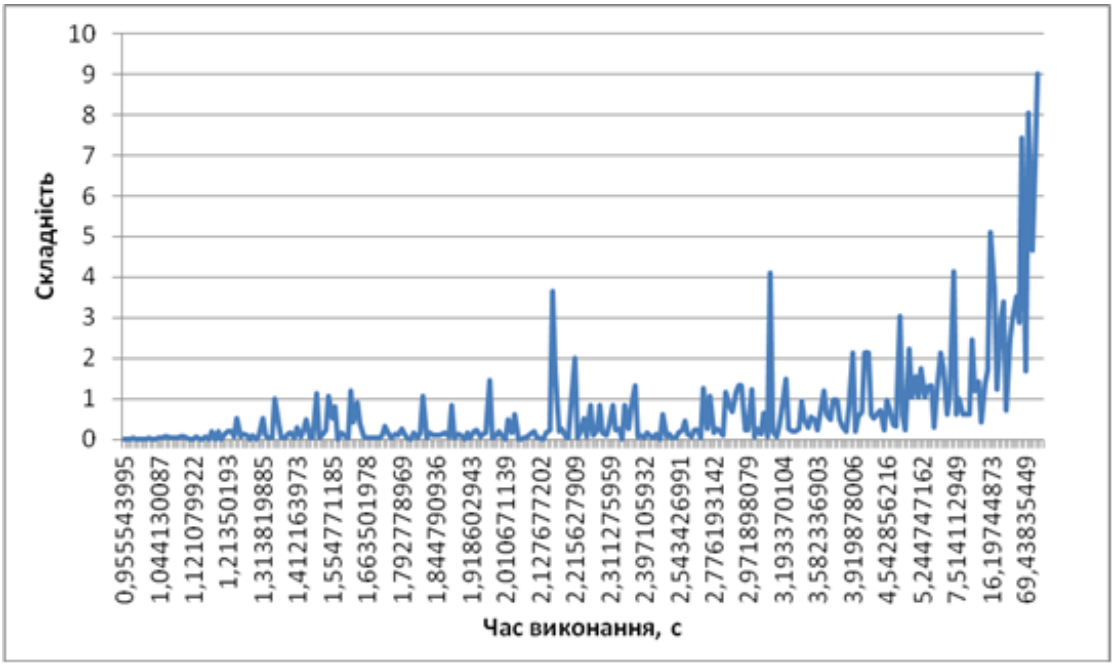

Рис. 1. Залежність часу класифікації та категоризації від складності онтологій, залежність впорядкована за часом

Основна причина значних відхилень, полягає у тому, що онтології, які імпортуються в основну онтологію, вже не є більш доступними за вказаною адресою, це становить певну проблему щодо життєвого циклу підтримки онтологій. Оскільки з одного боку імпортування онтологій, це хороша практика повторного використання онтологій і підтримка онтологій в актуальному стані, з іншого боку, нема ніяких гарантій, що згодом ця онтологія перестане бути доступною. Окрім цього виявлені проблеми, стосовно обмежень, щодо відсутності повної підтримки зі сторони HermiT, типів даних OWL2, наприклад xsd:dateTime. Також існують проблеми щодо розбору XML представлення OWL.

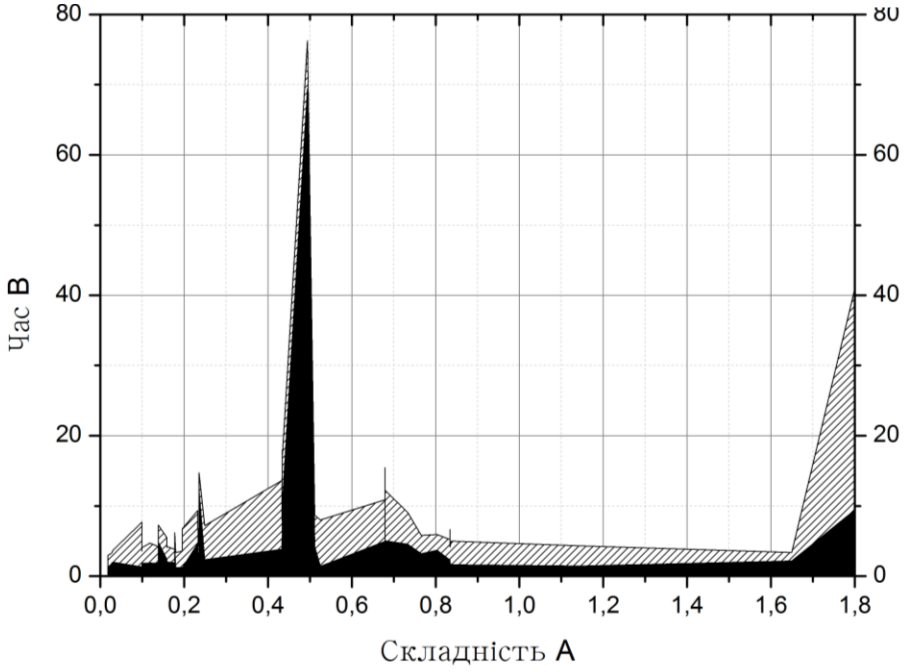

Рис. 2. Моделювання процесу виконання та реальний час виконання в с

На основі розробленої моделі ми провели експериментальне дослідження з двома обчислювальними вузлами. На початку подавалась тестова вибірка на основі якої виділявся зональний час, а згодом подавалась вибірка реальних даних, яка уточнювала зональний час початкової вибірки і для якої прогнозувався час виконання, результати експерименту показано на рис. 2. Тут представлено два процеси Real Time та Forecast, що відображають реальні дані та зпрогнозовані відповідно. Видно, що модель досить якісно відображає процес і набір параметрів які відображають складність завдання.

Висновки

Нами запропоновано алгоритм, який на основі початкових даних та на основі моделі Ерлагу з чергою нескінченної довжини, адаптивно прогнозував час виконання завдання яке знаходиться у черзі або виконується на даний момент. Запропонована та експериментально перевірена сервіс-орієнтована архітектура з асинхронним викликом, що дає можливість розпаралелювати виконання завдань з єдиної черги. Це дає можливість для кінцевого користувача спрогнозувати час очікування, та керувати чергою завдань для складних обчислювальних систем. У подальшому робота буде полягати в аналітичній оцінці алгоритму. Перевагами алгоритму також є відсутність визначення ваг для кожного з параметрів, головною вимогою, що параметри безпосередньо впливали на складність а отже і час виконання задачі.

Запропонований підхід, дає можливість вирішувати задачі, що потребують значних обчислювальних ресурсів, засобами Веб-технологій, використовуючи сервіс-орієнтований підхід. Теоретичні посилання перевірені експериментально та отримали підтвердження.

1. *Josuttis N.* SOA in Practice First Edition edn. O'Reilly Media, Inc., Gravenstein Highway North, Sebastopol (2007).
2. *Huhns M.N., Singh M.P.* Service-oriented computing: key concepts and principles. Internet Computing, IEEE 9(1), 75–81 (2005).
3. *O'Brien L., Bass L., and Merson P.* Quality Attributes and Service-Oriented Architectures. Software Engineering Institute. – 2005.
4. *Ma L.* Towards a Complete OWL Ontology Benchmark. The Semantic Web: Research and Applications Lecture Notes in Computer Science. – 2006. – P. 125–139.
5. *Vladimir Geroimenko C.* Visualizing the Semantic Web. Springer, London, 2006.
6. *Iversen V.* Teletraffic engineering handbook., COM Center Technical University of Denmark, 2001.
7. *Erik Chromy T.* Erlang c formula and its use in the call centers. Information and communication technologies and services. – 9(1). – P. 7–13.
8. *Koole G.* Call Center Mathematics. In: Call Center Optimization. (Accessed 2007) Available at: HYPERLINK "http://www.gerkoole.com/CCO/" http://www.gerkoole.com/CCO/
9. *Gunter Bolch S.* Queueing Networks and Markov Chains : Modeling and Performance Evaluation With Computer Science Applications second edition edn. Wiley-Interscience, 1998.
10. *Franz Baader D.* The Description Logic Handbook. Cambridge University Press, 2003.
11. *Rob Shearer B.* HermiT: A Highly-Efficient OWL Reasoner. In Alan Ruttenberg, U., ed. : Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU), Karlsruhe, Germany, 2008.

Одержано 20.07.2012

Про автора:

Новицький Олександр Вадимович,
молодший науковий співробітник.

Місце роботи автора:

Інститут програмних систем
НАН України,
03187, Київ-187,
проспект Академіка Глушкова, 40.
E-mail: alex@zu.edu.ua