УДК 004.424

А.Ю. Дорошенко, С.О. Шихутська

ПАРАЛЕЛЬНА РОЗПОДІЛЕНА РЕАЛІЗАЦІЯ МОДУЛЯ КЕРУВАННЯ РОБОЧИМИ ПРОЦЕСАМИ ДЛЯ СИСТЕМИ ПІДТРИМКИ ОПЕРАЦІЙ ОПЕРАТОРА ЗВ'ЯЗКУ

Запропонована паралельна реалізація модуля управління робочими процесами для OSS системи оператора зв'язку, побудована на основі фреймворку Hadoop. Проведене первинне дослідження цієї реалізації на прикладі прикладної задачі пошуку слів, що співпадають із шаблоном, у файлах великого розміру.

Ключові слова: модуль, робочий процес, система підтримки операцій, розподілені обчислення, аналіз.

Вступ

Сьогодні телекомунікаційні оператори зв'язку стараються автоматизувати як можна більше своїх промислових та робочих операцій. Для цього використовуються системи підтримки операцій – системи OSS (Operation Support System). Системи OSS дуже складні та складаються з різноманітних модулів та окремих автономних систем, що взаємодіють між собою. Прикладами таких модулів/систем є ERP (Enterprise Resource Planning) система, CRM [1] (Customer Relationship Management) система, SCM (Supply chain management) система та інші. Одним з найважливіших модулів є модуль управління робочими процесами. Цей модуль відповідає за інтеграцію різних систем між собою, а також забезпечує їх спільне виконання та можливість управління робочим процесом. Також модуль може виконувати самостійні задачі, що задаються конфігуратором. З переходом телекомунікаційних операторів зв'язку на системи OSS з'явилися задачі перенесення великого обсягу даних з попередніх систем, що використовувалися операторами, в системи OSS, а також систематизація та структуризація цих даних. В статті запропонована модифікація модуля управління робочими процесами, яка вдосконалить модуль управління робочими процесами та дозволить швидко обробляти великі обсяги даних. Саме для таких задач використовуються паралельні розподілені обчислювальні системи.

З іншого боку, на сьогодні паралельні розподілені обчислювальні системи набувають все більшого значення у зв'язку з технологічним розвитком та збільшенням інформаційних потоків у світі. Первинним призначенням подібних систем було вирішення наукових задач великих обсягів, які не могли бути вирішені на окремих класичних паралельних ресурсах. Вони також знайшли себе і в промислових масштабах. В даній статті розглянута модифікація модуля управління робочими процесами для OSS системи оператора зв'язку, вдосконалення його можливостей оперування великими обсягами даних, їх аналізом та швидкою обробкою.

Системи, що здатні здійснювати паралельні обчислення, зазвичай складні у розробці, конфігурації та не універсальні, що ускладнює їх використання для вирішення широкого спектру задач. Серед OSS систем, на які масово переходять оператори зв'язку, таких, що підтримують паралельні обчислення, вкрай мало. Саме тому, задача створення механізму, що дозволить компонувати класичні паралельні обчислення з розподіленими, з доступним інтерфейсом та гнучкою конфігурацією, є актуальною на сьогодні. Розробка подібного модуля управління робочими процесами для OSS систем операторів зв'язку не тільки спростить конфігурацію та організацію паралельних розподілених обчислень, але й дозволить будувати бізнес-процеси, що будуть здатні працювати з будь-якими обсягами даних.

Необхідно зазначити, що напрямок обробки великих обсягів даних є інноваційним і тільки нещодавно зробив вели-

кий стрибок у своєму розвитку. Саме тому, на разі є всього два фреймворки з відкритим програмним кодом, які мають у своєму арсеналі всі необхідні інструменти для створення систем з паралельними розподіленими обчисленнями – це фреймворк Hadoop та фреймворк Apache Spark. Фреймворк Apache Hadoop вже впроваджений у такі продукти, як «Facebook» та «Yahoo!». Фреймворк Apache Spark тільки набирає обертів та ще не має своєї історії масштабних впроваджень. Саме через нестабільність та дуже швидку зміну програмного коду у фреймворці Apache Spark, був обраний за основу фреймворк Apache Hadoop.

У даній роботі запропонована архітектура гнучкого та розширюваного інструментарію для побудови бізнес процесів у предметному середовищі OSS систем для операторів зв'язку, що здатен компонувати класичні паралельні обчислення з розподільними обчисленнями, що дозволяє значно збільшити швидкість виконання процесів за умови обробки ними великих обсягів даних на базі Java-фреймворку Apache Hadoop. Описано прототип подібного модуля, створеного на основі запропонованої архітектури з використанням наступних фреймворків: Hibernate – для організації доступу до бази даних [2]; Spring IoC – для гнучкості розробки та грамотного управління життєвим циклом об'єктів, що створюються під час виконання програмного коду [3]; Spring Security – для забезпечення захищеного доступу до розробленого програмного забезпечення [3].

В роботі наведені результати роботи прототипу на прикладі відомої задачі про підрахунок кількості слів у файлі. Задача була ускладнена підрахунком кількості певних слів, що відповідають шаблону, заданому регулярним виразом. Задача була ускладнена до відповідного рівня, з урахуванням особливостей предметного середовища та найбільш популярних проблем, що в ньому виникають.

В роботі виконано ще один крок у розвитку інструментарію для підтримки бізнес процесів, що протікають в OSS системах операторів зв'язку, у напрямку підвищення його ефективності в роботі з великими обсягами даних за рахунок розробки його модифікації, що підтримує паралельні розподілені обчислення. Матеріал статті організований наступним чином: у розділі 1 розглянуті можливості фреймворку Apache Hadoop; в розділі 2 описана архітектура фреймворку Apache Hadoop; в розділі 3 розглянуто архітектуру модуля управління робочими процесами для OSS системи оператора зв'язку; розділ 4 описує проведений дослід та його результати. Стаття завершується висновками та подальшими напрямками розвитку роботи.

1. Опис Apache Hadoop фреймворку

1.1. Основні концепції Hadoop платформи. Apache Hadoop — вільна програмна платформа і каркас для організації розподіленої обробки великих обсягів даних (що міряється у петабайтах) з використанням парадигми MapReduce, при якій завдання ділиться на безліч дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера [4]. До складу Hadoop входить також реалізація розподіленої файлової системи Hadoop Distributed Filesystem (HDFS), котра автоматично забезпечує резервування даних і оптимізована для роботи MapReduce-застосунків. Для спрощення доступу до даних в сховищі Hadoop розроблена БД HBase і SQL-подібна мова Hive, яка є свого роду SQL мовою для MapReduce і запити якої можуть бути розпаралелені і оброблені кількома Hadoop-платформами.

Переваги платформи:

1) велика швидкість операцій;

2) файлова система, що дозволяє зберігати в собі файли розмірів >10Gb;

3) відмовостійкість, наявність реплікацій даних;

4) масштабування – додавання нових серверів не викликає проблем;

5) робота задач безпосередньо на нодах, де зберігаються дані, не по віддаленому доступу.

1.2. Можливості Hadoop. Основні можливості Hadoop платформи:

1) зручна файлова система HDFS;

2) використання парадигми MapReduce;

3) розподілена підтримка запитів, операцій;

4) підтримка Java2EE, віддаленого доступу;

5) наявність інструментів інтеграції з такими популярними Java фреймворками, як Spring та Hibernate.

1.3. Коли варто застосовувати Hadoop? Hadoop варто використовувати у тому разі, якщо обчислення повинні бути компонованими, тобто ви повинні мати можливість запустити обчислення паралельно на певному масиві даних, потім сумістити отримані результати, можливо знову розпаралелити обчислення на основі отриманих результатів і т. п. Або якщо ви плануєте обробляти великі обсяги даних (більші, ніж може обробити одна машина) [5].

1.4. Коли не варто застосовувати Hadoop? Не потрібно застосовувати Hadoop платформу, якщо:

1) ваші задачі не компоновані – ви не можете розбити їх на частини, що можуть виконуватися паралельно та незалежно одна від одної;

2) якщо у вас невеликі обсяги даних;

3) якщо ви будуєте систему, що буде виконуватися в режимі реального часу.

2. Архітектура Hadoop платформи

2.1. Архітектура Hadoop кластера. Основною особливістю Hadoop платформи є її файлова система HDFS (Hadoop Distributed File System). По суті, це звичайна файлова система, тільки в рази більша та розкидана по різним серверам (нодам). Розглянемо її архітектуру.

Hadoop-кластер складається з наступних нод:

1) NameNode – це центр системи. Як правило, вона створюється однією на кластер та зберігає в собі всі метадані системи, такі як мапінг між файлами та блоками;

2) Secondary NameNode – це нода, що відслідковує зміни, що відбуваються в файловій системі та в кластері загалом. За рахунок цього з неї можливе ручне відновлення NameNode ноди, хоча вона і не являється реплікою NameNode ноди. На реальних кластерах NameNode та Secondary NameNode – це окремі сервери, які дуже вимогливі до ресурсів;

3) DataNode нод може бути дуже багато. Ці ноди зберігають в собі безпосередньо блоки файлів та містять в собі виконавчі ланки TaskTracker, які займається виконанням задач та операцій над блоками даних, що зберігаються локально на нодах;

4) MapReduce мастер нода (JobTracker) – займається розподіленням по всім нодам задач.

2.2. Парадигма MapReduce. В Apache Hadoop фреймворку знайшла свою реалізацію така парадигма, як MapReduce [5].

При правильній архітектурі системи, інформація про те, на яких машинах зберігаються блоки даних, дозволяє запустити на них же обчислювальні процеси і виконати більшу частину обчислень локально, тобто без передачі даних по мережі. Це в рази збільшує ефективність обчислень. І саме ця ідея лежить в основі парадигми MapReduce та її конкретної реалізації під Hadoop платформу. Кожен MapReduce процес складається з двох фаз:

– Map – виконується паралельно, і по можливості локально над кожним блоком даних. Тобто замість того, щоб доставляти терабайти даних до програмної реалізації, невелика, визначена користувачем програма копіюється на сервера з даними та виконує над ними все, що закладено в її логіці, що в свою чергу не вимагає перемішування та переміщення даних;

– Reduce – доповнює Map агрегуючими операціями, тобто збирає результати паралельних обчислень в одне ціле.

3. Розподілена архітектура модуля управління робочими процесами для OSS системи оператора зв'язку

3.1. Основні поняття, якими оперує модуль. У попередній версії модуля управління робочими процесами було реалізовано паралельне виконання процесів в рамках одного серверу додатків та синхронне/асинхронне виконання задач, із яких складається процес [6]. Було реалізовано кілька задач, які виконують стандартні операції – такі як запис в базу даних, пошук у базі даних, перетворення строкових даних, пошук даних по заданому шаблону і т. п. Також за допомогою базового функціоналу Java [7] Reflection API було реалізовано задачу, яка виконувала будь-який Java клас, вказаний на ній параметром, який є присутнім на сервері додатків та такий, що реалізує інтерфейс com.dipl.Task.

Також, були реалізовані інструменти підтримання життєвих циклів процесів та задач, які дозволяли користувачу відслідковувати в якому стані перебуває процес та його окремі задачі, моніторити помилки, що виникали при виконанні процесів та задач, приймати рішення про повторне виконання задач, перевідкриття задач/процесів тощо [8].

Розглянемо діаграми класів базових інтерфейсів прототипу модуля управління робочими процесами для OSS системи оператора зв'язку (рис. 1).

Опишемо детально кожен з класів.

Base – містить в собі загальні для всіх інших класів поля.

Process – клас є основним та окрім своїх параметрів містить також список задач, які повинні бути послідовно виконані.

Task – це основний об'єкт процесу. Має посилання на операцію, яка виконується при обробці задачі, порядок виконання серед всіх останніх задач процесу та на сам процес, в рамках якого виконується.

Operation – найменша атомарна частина процесу, що виконується. Містить в собі три параметри: path – шлях до класу, method – метод цього класу, params – вхідні параметри до вказаного методу, якщо їх кілька, вони записуються через символ «|».

Також розглянемо інструмент, що дозволив забезпечити життєвий цикл процесу та задачам [9].

3.2. Життєвий цикл процесу та задачі. Процес та кожна його задача мають свої життєві цикли. Життєві цикли нерозривно пов'язані з обробкою процесів і задач, та з їх виконанням. Саме тому розглянемо рівень, який виконує управління виконанням процесів та задач, разом з рівнем забезпечення життєвих циклів, та їх взаємодію.

Для початку, визначимо основні етапи, через які проходить кожен із процесів та задач:

1) створення процесу;
2) створення задач для процесу;
3) заповнення кожної із задач операціями;
4) запуск виконання процесу.

В залежності від того, як завершилося виконання процесу, слідуючи правилам життєвого циклу процесів та задач можна повторити виконання певних задач, перевідкрити задачу, призупинити процес, відмінити виконання задачі та ін.

Перехід процесу чи задачі із одного стану в інший є основним завданням рівня управління життєвим циклом. Також від цього рівня залежать дозволи або заборони на виконання певних дій в залежності від того, чи дозволяє стан процесу чи задачі їх виконувати.

У випадку окремого використання модуля, виклики до Process Manager є наслідками дій користувача та приходять від UI (User Interface). В реакцію на дії користувача Process Manager генерує відповідну подію POEvent. Події POEvent можуть бути різних типів та відповідають діям,

які можна проводити над процесом та над задачею.

Після того, як було згенеровано подію POEvent, Event Manager обробляє її, та в залежності від її типу, звертається до рівня DAO на створення /модифікацію/видалення задачі чи процесу, або звертається до Life Cycle Manager за дозволом на виконання певної операції.

Life Cycle Manager перевіряє поточний статус процесу чи задачі і видає дозвіл або заборону на виконання запитуваної операції. Якщо видається дозвіл, він міняє статус процесу чи задачі на відповідний та дозволяє подальшу обробку, яка здійснюється в класі Process Engine.

Process Engine має всього два метода – виконання процесу та виконання задачі. При виконанні процесу, він послідовно виконує всі задачі, що належать цьому процесу, та перед кожною задачею генерує для задачі необхідну подію для отримання від Life Cycle Manager дозволу на її виконання. В залежності від конфігурації окремої задачі, вона виконується або асинхронно, або синхронно [10].

4. Реалізація задачі пошуку слів у файлах великого розміру, яка буде паралельно розподільно виконувати обчислення

Для реалізації визначимо новий тип операції – DistributedOperation, який буде наслідуватися від Operation, проте матиме наступні параметри:

1) Path – шлях до класу, в якому описана логіка запуску паралельного розподіленого обчислення;

2) Method – назва методу класу, вказаного в параметрі Path, в якому описана логіка запуску паралельного розподіленого обчислення;

3) Map path – шлях до класу, в якому описана логіка, що буде паралельно розподільно виконуватися на всіх DataNode нодах Hadoop кластера.

Клас має бути об'явлений наступним чином та містити метод *map*:

public static class Map extends MapReduceBase implements

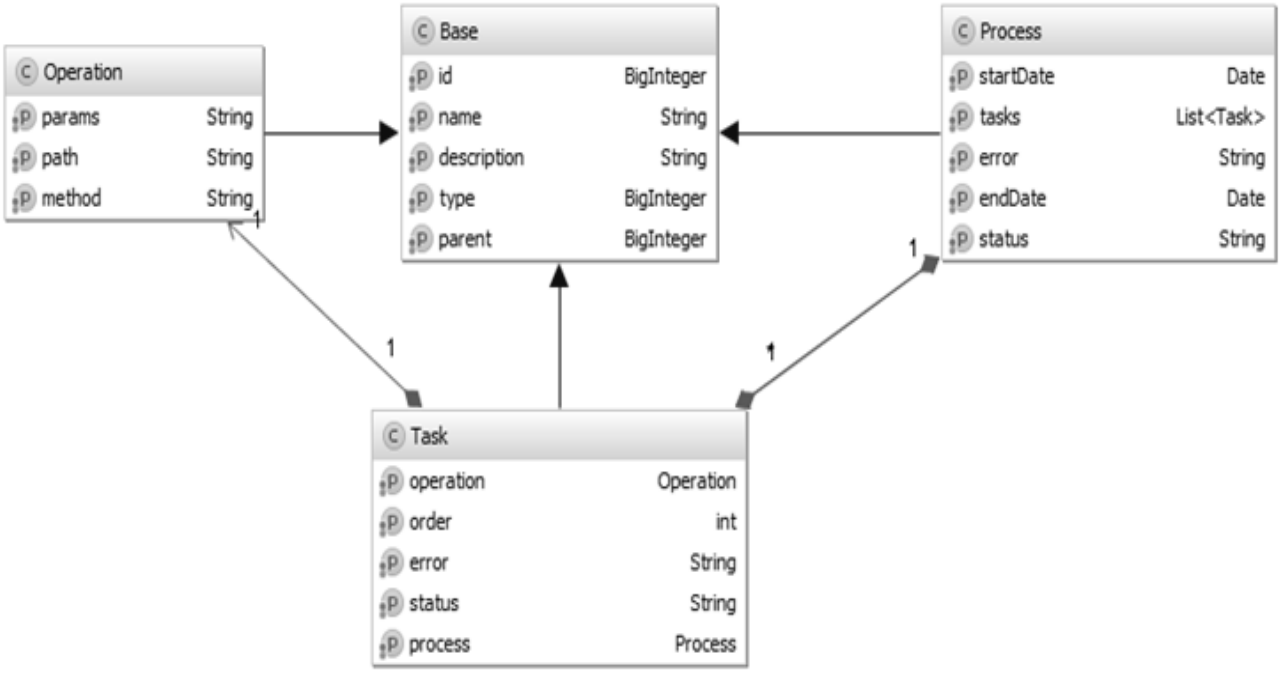

Рисунок 1

```
Mapper<LongWritable, Text, Text,
IntWritable> {
   public void map(LongWritable key, Text
value, OutputCollector<Text, IntWritable>
output, Reporter reporter) {
   }
}
```

4) Reduce path – шлях до класу, в якому описана логіка, що буде збирати результати обчислень з них усіх воєдино.

Клас має бути об'явлений наступним чином та містити метод *reduce*:

```
public static class Reduce extends
MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
   public void reduce(Text key,
Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output,
Reporter reporter){
   }
}
```

5) Input file path – шлях до файлу із вхідними данними;

6) Output file path – шлях до файлу із вихідними даними.

5. Дослід та отримані результати

Для проведення досліду було змодельовано два процеси.

5.1. Послідовне обчислення. Перший складався із задачі, сконфігурованої для послідовної обробки даних.

Для її реалізації було створено та написано клас OperationImpl, який має метод execute. В методі в одному потоці зчитується порядково файл та шукаються співпадіння із заданим шаблоном.

5.2. Паралельне розподілене обчислення. Другий складався із однієї задачі, зконфігурованої для розподіленої обробки даних.

Для її реалізації було створено та написано класи Map, Reduce та DistributedOperationImpl.

Реалізація класу Map полягає в тому, що при зчитуванні з блоку даних інформації, строкове значення ділиться на окремі слова, які перевіряються на те, чи є серед цих слів IP адреса. В якості шаблону, по якому проводився пошук, було використано наступний регулярний вираз:

(.*)+(\\b((25[0-5]|2[0-4][0-9]|[01]?

[0-9][0-9]?)(\\.|$)){4}\\b)*(.*)+

Цьому шаблону відповідають наступні рядки [11]:

1) (some text with symbols) 192.168.42.28 (some text with symbols);

2) 192.168.42.32(some text with symbols);

3) (some text with symbols) 192.168.13.01;

4) 10.0.0.13.

Клас Reduce відповідає за зведення результатів пошуку воєдино. Клас DistributedOperationImpl відповідає за запуск пошуку та виступає інтерфейсом, який інтегрує роботу з розподільними обчисленнями з модулем управління робочими процесами.

5.3. Проведення тестування системи. Для проведення тестування системи, було створено один Hadoop кластер, що містив в собі дві віртуальні DataNode ноди – по одному ядру кожна. Описаний в пункті 4.1 процес запускався послідовно на одній ноді та паралельно на двох та на чотирьох декілька разів для файлів наступних розмірів:

1) 50Mb;
2) 100Mb;
3) 400 Mb;
4) 600Mb;
5) 1Gb.

Для кожного із випадків були проведені виміри часу [12].

5.4. Аналіз отриманих результатів. В результаті дослідження були отриманні наступні середні значення для кожної із умов (таблиця).

Таблиця

Обсяг файлу (Mb)	Послідовно 1 нода (Tl, c)	Паралельно 2 ноди (Tp, c)	Паралельно 4 ноди (Tp, c)
50	4.674	5.417	4.704
100	8.967	7.129	6.839
400	34.925	14.145	8.142
600	52.508	20.081	4.093
1000	85.522	14.286	2.620

На рис. 2 показано графік залежності часу виконання процесу від обсягу оброблених даних. Порівняльний графік часу, затраченого на обробку файлів різного об'єму на одному, двох та чотирьох вузлах.

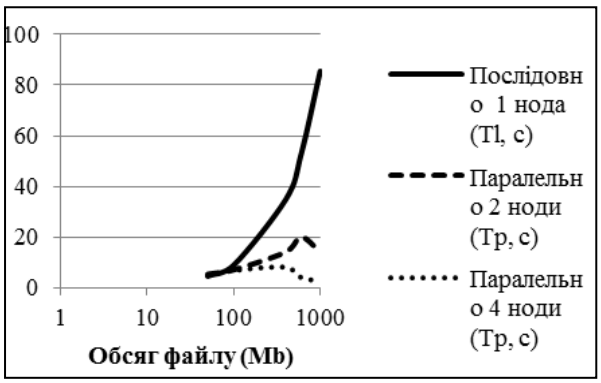

Рисунок 2

Отримані результати показують, що за послідовної обробки файлів при збільшенні їхнього обсягу час їхньої обробки та аналізу збільшується практично лінійно.

Тестування ж паралельної розподіленої обробки файлу показало, що на малих розмірах файлів результати практично співпадають з послідовною обробкою. Проте при збільшенні обсягу даних, паралельна розподілена обробка показує значно кращі результати.

Для досліду було використано одну, дві та чотири ноди, саме тому можна допустити, що при збільшенні кількості нод на великих обсягах даних, їхня паралельна розподілена обробка буде показувати значно кращі результати.

Висновки

У роботі запропонована архітектура модуля управління робочими процесами для OSS системи оператора зв'язку та його модифікація для виконання окремих незалежних операцій з обробки даних паралельно та розподілено. Для цієї реалізації було обрано фреймворк Apache Hadoop [13].

На даний момент розроблено «ядро» модуля та інтегровано його з інструментами, що дозволяють виконувати розподілено-паралельні обчислення. Розроблену частину системи було перевірено на задачі підрахування слів, що співпадають зі шаблоном, у файлах великих розмірів. Отримані результати досліду демонструють значний приріст у швидкодії цієї системи, що підтверджує ефективність обраної архітектури та реалізації.

Одним із головних надбань даного рішення є те, що розроблений модуль управління робочими процесами для OSS системи оператора зв'язку дозволить не тільки нарощувати ресурси для більш швидкої обробки даних користувачів, а й гнучко конфігурувати та слідкувати за виконання бізнес процесів у системі. Рішення є масштабованим та легко підтримуваним.

На базі поточних розробок буде проведено кілька наступних модифікацій, котрі дозволять тісніше інтегрувати операції по розподілено-паралельній обробці з уже існуючим ядром модуля, що дозволить ще більше скоротити час виконання подібних операцій. А також буде розроблено інтерфейс для виконання симуляцій, запуску процесів, їхнього моніторингу та управлінню. Ще один напрямок розвитку модуля – це його інтеграція з хмаринковими сервісами й повне перенесення обчислень в хмару.

1. *What* is BPM? [Електронний ресурс]. – Режим доступу: http://www.bpm.com/ what-is-bpm.html.
2. *Hibernate* ORM [Електронний ресурс]. – Режим доступу: http://hibernate.org/.
3. *Spring* Guides [Електронний ресурс]. – Режим доступу: https://spring.io/guides.
4. *Learning* Apache Hadoop [Електронний ресурс]. – Режим доступу: http://shop. oreilly.com/product/110000753.do.
5. *Building* Hadoop Clusters [Електронний ресурс]. – Режим доступу: http://shop. oreilly.com/product/110000685.do.
6. *Александров Л.В., Шепелев Н.П.* Системный анализ при создании и освоении объектов техники. – М.: НПО «Поиск», 1992. – 88 с.
7. *Java* API – Oracle Documentation [Електронний ресурс]. – Режим доступу: http://docs.oracle.com/javase/7/docs/api/.
8. *Архитектура* приложений – горячие точки [Електронний ресурс]. – Режим доступу: http://habrahabr.ru/post/40660/.
9. *Service*-oriented architecture [Електронний ресурс]. – Режим доступу: http://en.wikipedia.org/wiki/Service-oriented_ architecture
10. *Мартин Фаулер, Дейвид Райс, Мэттью Фоммел, Эдвард Хайет, Роберт Ми, Рэнди Стаффорд*. Шаблоны корпоративных приложений. – Вильямс, 2010. – 544 с.
11. *Кей Хорстман, Гари Корнелл* Java 2. Том 2. Тонкости программирования 9-е издание. – Вильямс, 2013. – 1008 с.
12. *Бейзер Б.* Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – Питер, 2004. – 320 с.
13. *Hadoop* 2 Essentials: An End-to-End Approach [Електронний ресурс]. – Режим доступу: http://www.amazon.com /Hadoop-Essentials-End---End-Approach /dp/1495496120/ref=sr_1_1?s=books&ie=UTF8&qid=1408657210&sr=1-1&keywords=Hadoop+2+Essentials

1. *What is BPM?* [Online] Available from: http://www.bpm.com/what-is-bpm.html. [Accessed: 8 October 2015].
2. *Hibernate ORM* [Online] Available from http://hibernate.org/. [Accessed 8 October 2015].
3. *Spring* Guides [Online] Available from https://spring.io/guides [Accessed: 8 October 2015].
4. *Learning Apache Hadoop* [Online] Available from: http://shop.oreilly.com/product /110000753.do [Accessed: 8 October 2015].
5. *Building Hadoop Clusters* [Online] Available from: http://shop.oreilly.com/product /110000685.do [Accessed: 8 October 2015].
6. *Alexandrov L.V., Shepelev N.P.* Systems analysis in the creation and using technics objects. – Moskow: "Poisk", 1992. – 88 p. (in Russian).
7. *Java API* – Oracle Documentation [Online] Available from http://docs.oracle. com/javase/7/docs/api/. [Accessed: 8 October 2015].
8. *Application architecture* [Online] Available from http://habrahabr.ru/post /40660/. [Accessed: 8 October 2015].
9. *Service*-oriented architecture [Online] Available from http://en.wikipedia.org /wiki/Service-oriented_architecture [Accessed: 8 October 2015].
10. Martin Fowler *Patterns of Enterprise Application Architecture*. Addison Wesley, November 05, 2002. ISBN: 0-321-12742-0, 544 p.
11. Cay Horstmann *Core Java Volume II – Advanced Features (9th Edition)*. Michigan: Ann Arbor, March 6, 2013. 1008 p.
12. *Beyzer B.* BlackBox testing. Functional testing of applications. Saint Petersburg: 2004. – 320 p.
13. *Hadoop* 2 Essentials: An End-to-End Approach [Online] Available from http://www.amazon.com/Hadoop-Essentials-End---End-Approach/dp/1495496120/ref=sr_1_1?s=books&ie=UTF8&qid=1408657210&sr=1-1&keywords=Hadoop+2+Essentials [Accessed: 8 October 2015]

Одержано 09.10.2015

Про авторів:

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень Інституту
програмних систем НАН України,

професор кафедри автоматики і
управління в технічних системах
НТУУ “КПІ”.
Кількість наукових публікацій в
українських виданнях – понад 150.
Кількість наукових публікацій в
іноземних виданнях – понад 30.
Індекс Гірша – 3.
ORCID orcid.org/0000-0002-8435-1451,

Шихутська Світлана Олександрівна,
студентка факультету інформатики
та обчислювальної техніки,
кафедри автоматики і управління
в технічних системах НТУУ “КПІ”,
Кількість наукових публікацій в
українських виданнях – 1.
ORCID orcid.org/0000-0002-8643-8999.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03680, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 1538.
E-mail: dor@isofts.kiev.ua,

Національний технічний університет
України "КПІ"
03056, Київ-56.
Проспект Перемоги, 37.
Тел.: (044) 236 7989.
E-mail: shyk.shyk@gmail.com