УДК 004.4'24

АВТОМАТИЗОВАНА РОЗРОБКА ПАРАЛЕЛЬНОЇ РОЗПОДІЛЕНОЇ СИСТЕМИ ОБРОБКИ ПОТОКОВИХ ДАНИХ

Д.С. Тітов, А.Ю. Дорошенко, О.А. Яценко

Виконана автоматизована розробка паралельної розподіленої динамічно масштабованої відмовостійкої системи для обробки потокових даних великого обсягу. Система розроблена на основі платформи для розподілених обчислень Hazelcast із використанням інструментарію генерації програм за високорівневими специфікаціями алгоритмів. Проведена перевірка та дослідження системи на прикладі обробки даних соціальної мережі Twitter, в якій реалізовано функціональність сентимент-аналізу повідомлень. Розглянуто механізм розгортання створеної системи на хмарній платформі.
Ключові слова: мережа, аналіз, відмовостійкість, масштабування, кластер, потік, хмара, вузол, генерація програм.

Выполнена автоматизированная разработка параллельной распределенной динамически масштабируемой отказоустойчивой системы для обработки потоковых данных большого объема. Система разработана на основе платформы для распределенных вычислений Hazelcast с использованием инструментария генерации программ по высокоуровневым спецификациям алгоритмов. Проведена проверка и исследование системы на примере обработки данных социальной сети Twitter, в которой реализована функциональность сентимент-анализа сообщений. Рассмотрен механизм развертывания созданной системы на облачной платформе.
Ключевые слова: сеть, анализ, отказоустойчивость, масштабирование, кластер, поток, облако, узел, генерация программ.

An automated development of a parallel distributed dynamically scalable fault-tolerant system for processing large amount of streaming data is performed. The system is based on the framework for distributed computing Hazelcast and the usage of the toolkit for generation of programs from high-level specifications of algorithms. The inspection and study of this system is performed on an example of data processing in Twitter social network in which sentiment analysis functionality is implemented. The mechanism of the deployment of the created system on a cloud platform is examined.
Key words: network, analysis, fault-tolerance, scaling, cluster, stream, cloud, node, program generation.

Вступ

На початку розвитку соціальних медіа, дослідники мали можливість оцінювати соціальні настрої в ручному режимі, адже обсяг інформації був відносно малий. Проте наразі потоки даних в Інтернеті в цілому та соціальних мережах, зокрема, сягнули колосальних масштабів, тому виникла потреба у створенні високопродуктивних систем для автоматизованої обробки цих даних [1]. Більш того, сучасні бізнес стратегії, що реалізують CRM-підхід, потребують не лише пасивного аналізу соціального контенту, а ще й передбачають проактивну реакцію на певні сигнали, що надходять з соціального Інтернет-простору. Таким чином, дослідження показують, що 64 % споживачів очікують отримати зворотній зв'язок від компанії у режимі реального часу та 94 % готові відмовитись від послуг певної компанії, якщо обслуговування клієнтів не передбачає цього [2].

У роботі [3] відзначається, що пікове навантаження соціальної мережі Twitter оцінюється в 140000 повідомлень в секунду та час обробки такого обсягу даних за допомогою наївного байєсівського класифікатора з використанням апаратного забезпечення [4] може сягати 12.458 секунди. Це свідчить на користь того, що система, призначена для обробки потокових даних великого обсягу, має бути паралельною та розподіленою, для того щоб встигати обробляти вхідні дані вчасно.

Також обчислення у реальному часі великої кількості даних відіграє велику роль у високочастотному трейдингу – основній формі алгоритмічної торгівлі на фінансових ринках, в якій сучасне обладнання та алгоритми використовуються для швидкого здійснення торгівельних операцій над цінними паперами. Як зазначається в [5], затримка в обробці даних вже навіть в 10 мілісекунд може призвести до суттєвих фінансових збитків.

У попередній роботі [6] була запропонована паралельна розподілена динамічно масштабована відмовостійка система для обробки потокових даних великого обсягу на основі використання фреймворку для розподілених обчислень Hazelcast [7]. Була проведена перевірка та первинне дослідження цієї системи на прикладі обробки даних соціальної мережі Twitter. У даній роботі виконано подальший розвиток розробленої системи, а саме, у систему додано функціональність, що відповідає за сентимент-аналіз повідомлень соціальної мережі. Сентимент-аналіз [8] є набором методів обробки природної мови, аналізу тексту та комп'ютерної лінгвістики, метою яких є виділення з текстів емоційно забарвленої лексики та емоційної оцінки автора по відношенню до об'єктів, про які йдеться у тексті. Новизна даної роботи полягає також у використанні для проектування класів системи алгеброалгоритмічної методології та інструментарію автоматизованої генерації програм на основі високорівневих специфікацій (схем) алгоритмів [9–11].

1. Алгебра алгоритмів та інструментарій автоматизованої розробки програм

В основу підходу до проектування програм, що використовується у даній роботі, покладений апарат модифікованих систем алгоритмічних алгебр (САА-М), що призначені для формалізації послідовних і паралельних обчислень в мультипроцесорних системах [9]. САА-М є двоосновною алгеброю $< Op, Pr; \Omega >$, де Op та

Pr – множини операторів та предикатів, відповідно; Ω – сигнатура операцій. На САА-М ґрунтується алгоритмічна мова САА/1 [9], призначена для багаторівневого структурного проектування та документування алгоритмів і програм. Перевагою її використання є можливість опису алгоритмів у природно-лінгвістичній формі, що полегшує досягнення необхідної якості програм. Подання операторів мовою САА/1 називаються САА-схемами. Основними операціями англомовної версії мови САА/1 є такі:

1) композиція – послідовне виконання двох операторів: *operator* 1; *operator* 2;

2) альтернатива – умовний оператор: IF *predicate* THEN *operator* 1 ELSE *operator* 2 END IF;

3) оператор циклу: WHILE (*predicate*) LOOP *operator* END OF LOOP.

У даній роботі також використовується оператор циклу, у якому ітерації виконуються за елементами колекції (списку), заданої виразом *expression*:

FOR EACH (*Variable type, Variable name*)
  <Collection> *expression*
LOOP
  operator
END OF LOOP

Тут *Variable type* вказує тип даних, а *Variable name* – ім'я змінної, яка послідовно буде приймати значення із колекції; *expression* – вираз, у якому зазначається колекція; *operator* – оператор тіла циклу.

В сигнатуру САА-М також входять операції, призначені для формалізації паралельних обчислень [9–11] та основних понять об'єктно-орієнтованого програмування. Для сумісності з мовою програмування Java в сигнатуру алгебри включені операції для визначення анотованих класів, полів даних, методів та параметрів. Анотації є спеціальною формою синтаксичних метаданих, що додаються у програмний код [12]. Далі наведено перелік основних операцій САА-М, що використовуються у даній роботі для визначення класів та їх складових елементів.

1. Визначення анотованого класу (в узагальненому вигляді):

Annotated class X implements Y (*Class name, Interface name*)
<Annotations>
  Annotation (*Annotation text*);
  <Fields>
  operators
<Methods>
  operators

Тут *Class name* – ім'я класу, який реалізує інтерфейс *Interface name*; <Annotations>, <Fields>, <Methods> – рядки, після яких необхідно вказати оператори визначення однієї або кількох анотацій, полів даних та методів класу, відповідно; *Annotation text* – текст анотації (наприклад, *Component*).

2. Визначення анотованого поля даних:

Field annotated (*Modifiers, Field type, Field name*)
<Annotations>
  Annotation (*Annotation text*)

Тут *Modifiers* – список модифікаторів доступу (наприклад, public, static, final і т. п.); *Field type* – тип даних поля; *Field name* – назва поля; *Annotation text* – текст анотації (наприклад, *Resource*).

3. Визначення анотованого методу:

Method annotated (*Return type, Method name, Parameters list*),
<Annotations>
  Annotation (*Annotation text*);
<Body>
  operators

Тут *Return type* – тип значення, яке повертає метод; *Method name* – назва методу; *Parameters list* – список формальних параметрів; *Annotation text* – текст анотації (наприклад, *Override*); <Body> – рядок, після якого вказуються оператори тіла метода.

Відмітимо, що не анотовані класи, поля даних та методи визначаються за допомогою конструкцій САА-М, аналогічних до вищенаведених. Їх специфікації відрізняються відсутністю ключового слова annotated та списку анотацій.

4. Конструкція для виклику методу екземпляра (об'єкта) класу має вигляд:

Call instance method (*Instance name*, *Method name*, *Arguments*),

де *Instance name* – ідентифікатор екземпляра класу; *Method name* – назва методу класу; *Arguments* – список фактичних параметрів методу.

Приклади САА-схем, у яких застосовуються вищерозглянуті операції, наведено у наступних розділах.

На використанні САА-М ґрунтується розроблена інструментальна система – онлайновий діалоговий конструктор синтаксично правильних програм (ОДСП, або Synthesis) [10, 11]. Система призначена для діалогового проектування, генерації й запуску програм, і містить такі основні компоненти:

- *клієнт* – веб-інтерфейс для діалогової взаємодії користувача із системою; надає можливість виконувати конструювання схеми алгоритму із використанням елементів бази даних, здійснювати генерацію коду цільовою мовою програмування та виконувати запуск згенерованої програми;

- *конструктор схем*, призначений для автоматизованої побудови високорівневих специфікацій алгоритмів. Основна ідея методу проектування полягає в порівневому конструюванні схем зверху вниз за допомогою підстановки мовних конструкцій САА-М (які користувач обирає зі списку) у дерево алгоритму. Специфікації конструкцій ґрунтуються на англомовній версії мови САА/1 (див. вище);

- *генератор програм*, який виконує автоматичний переклад побудованої в інструментарії схеми алгоритму в текст цільовою мовою програмування (Java, C++);

- *база даних алгеброалгоритмічних специфікацій*, у якій зберігається текст конструкцій САА-М та базисних елементів схем, а також шаблони їх програмних реалізацій.

У попередніх роботах [9–11] засоби САА-М та розроблений інструментарій використовувались для автоматизації проектування та генерації паралельних програм для багатоядерних центральних процесорів та графічних прискорювачів. У наступних розділах виконаний подальший розвиток створених засобів у напрямку їх застосування для розробки паралельної розподіленої системи обробки потокових даних соціальної мережі Twitter на основі фреймворку Hazelcast [7]. Цільовою мовою програмування у даному випадку є Java.

2. Отримання даних з соціальної мережі Twitter та сентимент-аналіз повідомлень

Соціальна мережа Twitter надає доступ до своїх даних у вигляді Twitter REST API та Twitter Firehose. В даній статті не розглядається Twitter REST API, оскільки доступ до нього є пакетним: клієнт відправляє запит до серверу, а той відповідає певним пакетом даних. Така форма доступу не є отриманням інформації у реальному часі, на відміну від Twitter Firehose – технології, за якою Twitter надає текстові повідомлення одразу як вони з'являються в системі, тобто у реальному часі [13]. Доступ до повного потоку контенту є платним, проте існує безкоштовна лімітована версія – Twitter Sample Stream, що фактично є Twitter Firehose, обмеженим 1% кількості повідомлень. Існує декілька Java-провайдерів для взаємодії з Twitter Sample Stream, одним з найпопулярніших та найзручніших є Spring Social, а саме його підмодуль – Spring Social Twitter [14]. Для початку прослуховування вхідного потоку достатньо створити Spring Bean, що реалізує інтерфейс *StreamListener* [6].

Для практичного тестування вищезгаданих механізмів була обрана проблема сентимент-аналізу текстових повідомлень. Сентимент-аналіз або аналіз тональності тексту (Sentiment analysis) – це клас методів контент-аналізу в комп'ютерній лінгвістиці, призначений для автоматизованого виявлення в текстах емоційно забарвленої лексики та емоційної оцінки думок авторів по відношенню до об'єктів, мова про які йде у тексті [8]. Набуття популярності соціальними медіа, таких як блоги та соціальні мережі, підігріло інтерес до сентимент-аналізу. З поширенням оглядів, відгуків, рекомендацій та інших форм Інтернет-самовираження, онлайн-думки почали перетворюватися на певну форму віртуальної валюти для бізнесу, що розширює ринки збуту своєї продукції, шукає нові можливості та піклується про свою репутацію. З огляду на те, що бізнес шукає способи автоматизації фільтрації інформаційного шуму, розуміння онлайн-діалогів та знаходження релевантного контенту, проблема сентимент-аналізу стає дуже актуальною в наш час.

Метою аналізу тональності є знаходження думок у тексті та визначення їх властивостей. У залежності від поставленого завдання нас можуть цікавити різні властивості, наприклад: суб'єкт тональності (автор тексту), тональна оцінка (наприклад, позитивна, нейтральна, негативна), об'єкт тональності (предмет, про який висловлюється думка). У комп'ютерних програмах автоматизованого аналізу тональності застосовують алгоритми машинного навчання, інструменти статистики та обробки природної мови (Natural Language Processing, скорочено NLP), що дозволяє обробляти великі масиви тексту.

Для розв'язання задачі аналізу тональності тексту у даній роботі обраний фреймворк Standford CoreNLP [15]. Цей фреймворк є набором програмних бібліотек для аналізу природних мов та дозволяє виконувати певні операції обробки текстів, такі як фільтрація, сегментація, стемінг, побудова лексичних дерев та багато інших. До цього переліку також входить функція сентимент-аналізу.

Далі наведено САА-схему для класу *NLPProcessor*, який забезпечує обробку тексту, поданого природною мовою, на основі використання Standford CoreNLP. Схема побудована із використанням операцій САА-М та інструментальної системи ОДСП, розглянутих у розділі 1.

Comment (*Nature Language Processor*);
Annotated class (*NLPProcessor*)
<Annotations>
  Annotation (*Component*)

<Fields>
  Field annotated (*private*, *StanfordCoreNLP*, *pipeline*)
  <Annotations>
    Annotation (*Autowired*)

<Methods>
  Comment (*Finds sentiment of a message*);
  Method (*public*, *Sentiment*, *findSentiment*, *String text*)
  <Body>
    Declare variable (*int*, *mainSentiment*, 0);
    IF (String is not empty (*text*)) THEN
      Declare variable (*int*, *longest*, 0);
      Declare and initialize variable (*annotation*, *Annotation*);
        <Statement> Call instance method (*pipeline*, *process*, *text*);
      FOR EACH (*CoreMap*, *sentence*)
        <Collection> Get sentences annotation (*annotation*)
      LOOP
        Declare and initialize variable (*tree*, *Tree*)
          <Statement> Get sentiment annotated tree (*sentence*);
        Declare and initialize variable (*sentiment*, *int*)
          <Statement> Call instance method (*RNNCoreAnnotations*, *getPredictedClass*, *tree*);
        Declare and initialize variable (*partText*, *String*)
          <Statement> Get string representation of (*sentence*);
        IF (Larger (Call instance method (*partText*, *length*), Variable (*longest*))) THEN
          Assign value (*mainSentiment*, *sentiment*);
          Assign (*longest*, Call instance method (*partText*, *length*))
        END IF
      END OF LOOP
    END IF
    Return (Call instance method (*Sentiment*, *valueOf*, *mainSentiment*));

Оцінки тональності тексту (невизначена, сильна негативна, слабка негативна, нейтральна, слабка позитивна та сильна позитивна) зазначаються у класі *Sentiment*, САА-схему якого наведено далі. Згаданий клас є перерахованим типом даних (Enum type), множина значень якого вказується за допомогою списку констант (Enum constant). Клас містить також поле даних *sentimentScore* та два методи.

Comment (*Enum representation of a sentiment score of a text*);
Enum type (*Sentiment*)
<Constants>
  Enum constant (*UNDEFINED*(-1)); Enum constant (*STRONG_NEGATIVE*(0));

```
  Enum constant (WEAK_NEGATIVE(1)); Enum constant (NEUTRAL(2));
  Enum constant (WEAK_POSITIVE(3)); Enum constant (STRONG_POSITIVE(4));

<Fields>
  Field (private, int, sentimentScore);

<Methods>
  Class constructor method (Sentiment, int sentimentScore)
  <Body>
    Assign value (this.sentimentScore, sentimentScore);

  Method (public static, Sentiment, valueOf, int sentimentScore)
  <Body>
    FOR EACH (Sentiment, sentiment)
      <Collection> Variable (values())
    LOOP
      IF (Equal(Instance field (sentiment, sentimentScore), Variable (sentimentScore))) THEN
        Return (sentiment);
      END IF
    END OF LOOP
    Return (UNDEFINED);
```

Генератор програм системи ОДСП автоматично виконав трансляцію наведених САА-схем у мову програмування Java.

3. Реалізація паралельної розподіленої системи за допомогою Hazelcast

Hazelcast є платформою з відкритим програмним кодом для Java, яка використовується для побудови кластерів та масштабованого розподілу даних [7]. Типовими функціями фреймворку є:

- організація обміну даними/станом серед багатьох серверів та кешування даних (розподілений кеш);
- кластерне розподілення програми та забезпечення захищеної комунікації між обчислювальними вузлами;
- синхронізоване використання даних у пам'яті;
- розподіл обчислень між багатьма серверами та паралельне виконання задач;
- забезпечення відмовостійкого управління даними.

Перевагою використання Hazelcast є автоматичне розгортання та управління обчислювальним кластером: розробник не повинен передбачати окремого механізму синхронізації обчислювальних вузлів між собою. Це гарантує ще більшу відмовостійкість, оскільки збій жодного окремого серверу не зможе призвести до того, що система перестане функціонувати в цілому. Для комунікації між вузлами використовується розподілений *ExecutorService*, який отримує на вхід екземпляри класів, що реалізують інтерфейси *Callable* або *Runnable*. Ці представляють собою команди (з інкапсульованими всередині даними), які будуть виконані на певному обчислювальному вузлі. Фрагмент САА-схеми такого класу наведений далі.

```
Class X implements Y (public, TwitterCallable, Callable<TweetWrapper>, Serializable)
<Methods>
  Method annotated throws (public, TweetWrapper, call, , Exception)
  <Annotations>
    Annotation (Override)
  <Body>
    Declare and initialize variable (sentiment, Sentiment)
      <Statement> Find sentiment of a message (tweetWrapper);
    Set sentiment (tweetWrapper, sentiment);
    Save tweet wrapper (tweetWrapper);
    Write tweet wrapper information to the log (tweetWrapper);
    Return (tweetWrapper);
```

Hazelcast надає дуже зручний механізм моніторингу подій, які трапляються з вузлами кластеру – інтерфейс *MembershipListener*. Цей інтерфейс має три методи, що дозволяють відслідковувати додавання нового вузла, відключення вузла та зміну атрибутів вузла [6]. Саме можливість задавати значення певних атрибутів на елементах кластеру дозволяє відділяти master-вузол від інших та динамічно змінювати конфігурацію системи.

4. Розгортання кластеру в хмарині Amazon Elastic Compute Cloud

Для розгортання системи на основі Hazelcast була обрана хмарна платформа Amazon Elastic Compute Cloud (EC2). EC2 є веб-сервісом, який дозволяє отримати доступ до обчислювальних потужностей і налаштувати ресурси з мінімальними затратами. Служба скорочує час, необхідний для отримання і завантаження нового сервера [16]. Для тестування кластеру в EC2 було замовлено 5 серверів, які мають фіксовані технічні характеристики [17]. На серверах встановлена операційна система Ubuntu Linux, доступ здійснюється за допомогою протоколу SSH.

Загалом Hazelcast передбачає три опції для знаходження вузлів один-одним: статична IP-адресація, multicast в локальній мережі та AWS EC2 Auto Discovery. Перший підхід не задовольняє потребу динамічного підключення обчислювальних вузлів, а між другим та третім був обраний AWS EC2 Auto Discovery, оскільки він є простою та зручною інтеграцією із середовищем EC2. Далі наведено CAA-схему методу, що відповідає за створення *HazelcastInstance* – основного класу фреймворку Hazelcast, який інкапсулює в собі всі налаштування кластеру.

Method annotated (*HazelcastInstance*, *hazelcastInstance*)
<Parameters>
  Parameter annotated with value (*accessKey*, *String*, *aws.access.key*);
  Parameter annotated with value (*secretKey*, *String*, *aws.secret.key*)
<Annotations>
  Annotation (*Bean*)
<Body>
  Create new configuration (*config*);
  Set port properties (*config*, 5701, *true*);
  Add listener configuration (*config*);
  Get network configuration (*network*, *config*);
  Get join configuration (*join*, *network*);
  Set multicast configuration enabled (*join*, *false*);
  Set access and secret keys (*accessKey*, *secretKey*);
  Create new Hazelcast instance (*hazelcastInstance*, *config*);
  Get cluster members (*members*, *hazelcastInstance*);
  Get cluster local member (*localMember*, *hazelcastInstance*);
  Declare and initialize variable (*membersSize*, *int*);
    <Statement> Get the number of cluster members (*members*);
  Declare and initialize variable (isMasterNode, Boolean)
    <Statement> Equal values (*membersSize*, 1);
  Set the value of cluster member attribute (*localMember*, *IS_MASTER_NODE*, *isMasterNode*);
  Return (*hazelcastInstance*);

Як бачимо, інтеграція з EC2 полягає у зазначенні ключа доступу та секретного ключа доступу до платформи, які можна отримати в панелі керування хмарним сервісом. Після запуску додатку на першому сервері Hazelcast створює кластер, що наразі складається з одного обчислювального вузла. Коли будуть додаватись інші вузли, їх синхронізація між собою відбуватиметься повністю прозоро для розробника і відображатиметься в терміналі. Крім того, існує можливість запуску декількох екземплярів додатків на одному сервері – в такому випадку кожний вузол стартує на своєму власному порту. Для зручного управління кластером у системі реалізований веб-інтерфейс, який дозволяє керувати станом кожного окремого вузла за допомогою запитів до REST-сервісів [6].

5. Тестування системи

Для порівняння ефективності розглянутих технологій, була проведена серія експериментів: розроблена розподілена система запускалась на одному, двох, трьох та чотирьох серверах в хмарині Amazon EC2. Кож-

ний сервер має двоядерний процесор Intel Xeon E52676, 2.40 ГГц. Тривалість вимірювання дорівнює десяти секундам (повідомлення оброблювалися із використанням механізму сентимент-аналізу фреймворку Standford CoreNLP). Результати експерименту та побудований графік подані у таблиці та на рис. 1.

Таблиця 1. Продуктивність розподіленої системи обробки повідомлень в хмарині Amazon EC2

Кількість серверів, *S*	Кількість оброблених повідомлень, *P*
1	288
2	491
3	749
4	1310

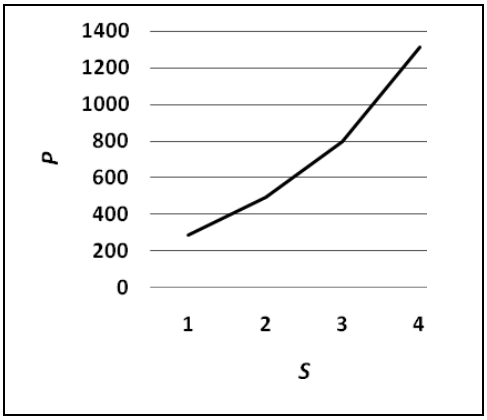

Рис. 1. Графік продуктивності розробленої системи: залежність кількості оброблених повідомлень *P* від кількості серверів *S* в Amazon EC2

Також система була протестована на персональному комп'ютері з восьмиядерним процесором Intel Core i7 4702MQ, 2.20 ГГц та на вузлі обчислювального кластеру з двома чотириядерними процесорами Intel Xeon E5405, 2.00 ГГц. Результати експерименту та побудований графік подані у табл. 2 та на рис. 2. Як видно з наведених даних, найбільш ефективним є виконання системи в хмарині Amazon EC2 на процесорах Intel Xeon E52676.

Таблиця 2. Продуктивність розподіленої системи обробки повідомлень з використанням різних конфігурацій апаратного забезпечення

Тип процесору	Кількість оброблених повідомлень
4702MQ	857
E5405	525
E52676	1310

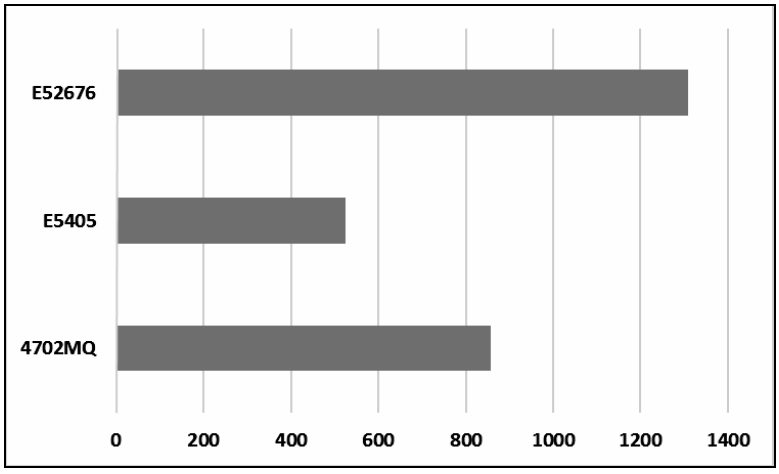

Рис. 2. Графік продуктивності розробленої системи в залежності від апаратної конфігурації

Висновки

У роботі виконано подальший розвиток паралельної розподіленої динамічно масштабованої відмовостійкої системи для обробки потокових даних великого обсягу. Як приклад джерела даних для обробки обрана соціальна мережа Twitter та її потоковий API – Twitter Firehose (а саме його тестова версія Twitter Sample Stream). Система розроблена на основі Hazelcast Framework із використанням інструментарію автоматизованої генерації програм за високорівневими специфікаціями алгоритмів. Програмний комплекс представляє собою динамічно масштабоване та відмовостійке кластерне рішення, придатне до розгортання в локальній мережі або в хмарній платформі Amazon EC2. У систему додано нову функціональність, що відповідає за сентимент-аналіз повідомлень соціальної мережі.

Одним з головних надбань розробленого рішення є те, що дослідник може нарощувати ресурси для обчислень за своїм бажанням, різної конфігурації та потужності, що дає змогу досягати бажаної швидкодії системи. В результаті серії експериментів встановлено, що збільшення кількості обчислювальних вузлів у кластері забезпечує значний приріст продуктивності системи.

В подальшому планується оптимізація рішення на основі Hazelcast, реалізація криптографічного захисту даних, що передаються, та низка інших покращень.

1. *Social* Listening in Practice. Market Research [Електронний ресурс]. – Режим доступу: https://www.brandwatch.com/guide-market-research. – 25.01.2016 р.
2. *Social* Listening in Practice. Social customer service [Електронний ресурс]. – Режим доступу: https://www.brandwatch.com/customer-service-guide. – 25.01.2016 р.
3. *Тітов Д.С., Дорошенко А.Ю.* Моніторинг соціальних мереж в системі реального часу // Наукова дискусія: теорія, практика, інновації: Матеріали V Всеукраїнської з міжнародною участю науково-практичної заочної конференції, 27–28 березня, 2015. – Київ: ГО "ІОМП", 2015. – С. 93–96.
4. *Intel* Core i7-3770k Processor [Електронний ресурс]. – Режим доступу: http://ark.intel.com/products/65523. – 25.01.2016 р.
5. *Mamudi S.* Charlie Munger: HFT is Legalized Front-Running [Електронний ресурс]. – Режим доступу: http://blogs.barrons.com/stockstowatchtoday/2013/05/03/charlie-munger-hft-is-legalized-front-running. – 25.01.2016.
6. *Дорошенко А.Ю., Тітов Д.С.* Паралельна розподілена система для аналізу потокових даних соціальних мереж // Проблеми програмування. – 2015. – № 4. – С. 31–39.
7. *Hazelcast* [Електронний ресурс]. – Режим доступу: http://hazelcast.org. – 25.01.2016.
8. *Sentiment* analysis [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Sentiment_analysis. – 25.01.2016.
9. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
10. *Йовчев В.А., Мохница А.С.* Инструментальные средства алгебры алгоритмики на платформе Web 2.0 // Проблеми програмування. – 2010. – № 2–3. – С. 547–555.
11. *Дорошенко А.Ю., Бекетов О.Г., Іванів Р.Б. та інші*. Автоматизована генерація паралельних програм для графічних прискорювачів на основі схем алгоритмів // Там само. – 2015. – № 1. – С. 19–28.
12. *Oracle* Java documentation. The Java Tutorials. Lesson: Annotations [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/javase/tutorial/java/annotations. – 04.11.2015 р.
13. *Public* streams [Електронний ресурс]. – Режим доступу: https://dev.twitter.com/streaming/public. – 25.01.2016.
14. *Spring* Social [Електронний ресурс]. – Режим доступу: http://projects.spring.io/spring-social. – 25.01.2016.
15. *Stanford* CoreNLP – a suite of core NLP tools [Електронний ресурс]. – Режим доступу: http://nlp.stanford.edu/software/corenlp.shtml. – 25.01.2016.
16. *Amazon* EC2 [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/ec2. – 25.01.2016.
17. *Amazon* EC2 Instances [Електронний ресурс]. – Режим доступу: http://aws.amazon.com/ec2/instance-types. – 25.01.2016.

References

1. Brandwatch. (2016) *Social Listening in Practice. Market Research.* [Online] Available from: https://www.brandwatch.com/guide-market-research. [Accessed: 25th January 2016]
2. Brandwatch. (2016) *Social Listening in Practice. Social customer service*. [Online] Available from: https://www.brandwatch.com/customer-service-guide. [Accessed: 25th January 2016]
3. Titov, D.S. & Doroshenko A.Yu. (2015) Social networks monitoring in real-time systems. In *Proc. of 5th Int. Conf. "Scientific discussion: theory, practice, innovation"*. Kyiv, Ukraine, 27-28 March 2015. Kyiv: IOMP. p. 93-96 (in Ukrainian)
4. Intel. (2016) *Intel Core i7-3770k Processor*. [Online] Available from: http://ark.intel.com/products/65523. [Accessed: 25th January 2016]
5. Mamudi S. (2016) *Charlie Munger: HFT is Legalized Front-Running.* [Online] Available from: http://blogs.barrons.com/stockstowatchtoday/2013/05/03/charlie-munger-hft-is-legalized-front-running. [Accessed: 25th January 2016]
6. Titov, D.S. & Doroshenko A.Yu. (2015) The parallel distributed system for social media data streams analysis. *Problems in programming.* (4). p. 31-39. (in Ukrainian)
7. Hazelcast. (2016) *Hazelcast*. [Online] Available from: https://hazelcast.org. [Accessed: 25th January 2016]
8. Sentiment analysis (2016) *Sentiment analysis.* [Online] Available from: https://en.wikipedia.org/wiki/Sentiment_analysis. [Accessed: 25th January 2016]
9. Andon, P.I. et al. (2007) *Algebra-algorithmic models and methods of parallel programming*. Kiev: Akademperiodika. (in Russian)
10. Iovchev, V.O. & Mokhnitsa, O.S. (2010) Algebra-algorithmic tools on Web 2.0 platform. *Problems in programming.* (2-3). p. 547-555. (in Russian)
11. Doroshenko, A.Yu., Beketov, O.G., Ivaniv R.B., Iovchev, V.O., Myronenko, I.O. & Yatsenko, O.A. (2015) Automated generation of parallel programs for graphics processing units based on algorithm schemes. *Problems in programming.* (1). p. 19-28. (in Ukrainian)

12. Oracle. (2016) *Java documentation. The Java Tutorials. Lesson: Annotations*. [Online] Available from: https://docs.oracle.com/javase/tutorial/java/annotations. [Accessed: 25th January 2016].
13. Twitter. (2016) *Public streams*. [Online] Available from: https://dev.twitter.com/streaming/public. [Accessed: 25th January 2016]
14. Spring. (2016) *Spring Social*. [Online] Available from: http://projects.spring.io/spring-social. [Accessed: 25th January 2016]
15. The Stanford NLP (Natural Language Processing) Group. (2016) *Stanford CoreNLP – a suite of core NLP tools*. [Online] Available from: http://nlp.stanford.edu/software/corenlp.shtml. [Accessed: 25th January 2016]
16. Amazon. (2016) *Amazon EC2*. [Online] Available from: https://aws.amazon.com/ec2. [Accessed: 25th January 2016]
17. Amazon. (2016) *Amazon EC2 Instances*. [Online] Available from: http://aws.amazon.com/ec2/instance-types. [Accessed: 25th January 2016]

Про авторів:

Тітов Дмитро Сергійович,
студент факультету інформатики та обчислювальної техніки,
кафедри автоматики і управління в технічних системах НТУУ “КПІ”.
Кількість наукових публікацій в українських виданнях – 3.
http://orcid.org/0000-0003-1607-5405,

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.
http://orcid.org/0000-0002-8435-1451,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук, старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 29.
Кількість наукових публікацій в іноземних виданнях – 12.
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

Національний технічний університет України “Київський політехнічний інститут”,
Кафедра автоматики і управління в технічних системах,
03056, Київ-056, вул. Політехнічна, 41, корпус 18.
Тел.: +380 (44) 204 86 10, +380 (44) 204 92 85.
E-mail: doroshenkoanatoliy2@gmail.com, dmytro.titov@gmail.com.

Інститут програмних систем Національної академії наук України.
03187, Київ-187, проспект Академіка Глушкова, 40, корпус 5.
Тел.: +380 (44) 526 3559.
Факс: +380 (44) 526 6263.
E-mail: oayat@ukr.net.