НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК “Інститут прикладного системного аналізу”
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

“___”_____2015 р.

Дипломна робота

___першого (бакалаврського)_____**рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності_____6.050101 Комп’ютерні науки_____
(код та назва спеціальності)

на тему: _Розподілений аналіз великих масивів_____
_____
_____

Виконав (-ла): студент (-ка) _4___ курсу, групи ДА-12
(шифр групи)

___Варга Ігор Юрійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент к.т.н. Корначевський Ярослав Ілліч_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Охорона праці к.б.н Гусєв А.М_____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент професор, д.т.н. Бідюк П.І._____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст.. викладач Бритов О.А._____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 6.050101, Компютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Варзі Ігорю Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Розподілений аналіз великих масивів даних

керівник проекту (роботи)) Корначевський Ярослав Ілліч, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 2015 р. №_____

2. Строк подання студентом проекту (роботи) 22.06.2015

3. Вихідні дані до проекту (роботи) _____

Програмний продукт що задовільняє можливості отримання та обробки великих даних, та прогнозує на основі даних по відвідуваності онлайн маркетів, те який товар найкраще виводить на ринок. Використати безкоштовні програмні системи, наприклад Storm чи Spark.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Розглянути можливі варіанти реалізації данного продукту та обрати варіант для розробки.
2. Розробити алгоритм роботи.

3. Розробити архітектуру системи.
4. Розробити програмну модель та протестувати її.

5. Перелік графічного матеріалу (плакатів тощо)
1. Лямда-архітектура кластера – плакат.
2. Схема Storm кластера – плакат.
3. Дані по переходам по маркетам – плакат.
4. Результати роботи програми – плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Гусєв А.М.., доцент		
Основна частина			

7. Дата видачі завдання 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Вибір варіанту для розробки	3.03.2015	
4	Розробка алгоритму та структури	4.04.2015	
5	Розробка плану тестування	14.05.2015	
6	Розробка програмної моделі	21.05.2015	
7	Тестування моделі	12.06.2015	
8	Оформлення дипломної роботи	16.06.2015	
9	Отримання допуску до захисту та подача роботи в ДЕК	22.06.2015	

Студент _____ І.Ю. Варга
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Я.І. Корначевський
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

Зміст

ВСТУП ... 8
1. ОГЛЯД ПРОБЛЕМАТИКИ. ОПИС ТА ВИБІР ВИКОРИСТОВАНИХ СИСТЕМ ТА МАТЕМАТИЧНИХ МЕТОДІВ ... 11
1.1 Огляд проблематики ... 11
1.2 Постановка задачі ... 13
1.3 Вибір методів аналізу. ... 14
1.4 Вибір програмних сервісів ... 18
1.4.1 Apache Kafka ... 18
1.4.2 Apache Hbase ... 21
1.4.3 Apache Storm ... 23
1.4.4 Java ... 29
1.4.5 Python ... 33
1.4.6 Django ... 37
1.4.7 HTML та XHTML ... 40
1.4.8 CSS ... 42
1.4.9 JavaScript та JQuery ... 44
1.4.10 Hadoop Ecosystem Distributives ... 47
1.5 Висновки ... 51
2. ОПИС ПРОГРАМНОГО ПРОДУКТУ. АНАЛІЗ АРХІТЕКТУРИ. СИСТЕМНІ ВИМОГИ. ДОКУМЕНТАЦІЯ. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ... 52
2.1 Загальний опис та аналіз архітектури. ... 52
2.2 Системні вимоги для інсталяції ... 55
2.3 Планування кластера та аналіз вхідних данних ... 57
2.4 Інсталяція програмного забезпечення ... 61
2.5 Аналіз роботи програми та документація користувача ... 65
2.6 Висновки ... 68
3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ ... 69
3.1 Загальна характеристика приміщення і робочого місця ... 69
3.2 Аналіз потенційно небезпечних і шкідливих виробничих факторів на робочому місці 71
3.2.1 Мікроклімат ... 72
3.2.2 Недостатня освітленість і підвищена яскравість світла ... 73
3.2.3 Шум та вібрація ... 74
3.2.4 Небезпека враження людини електричним струмом ... 75
3.2.5 Пожежна безпека ... 76
3.3 Висновки ... 76
ВИСНОВКИ ... 77
ПЕРЕЛІК ПОСИЛАНЬ ... 78

ВСТУП

Оцінка сучасного стану проблеми

При прогнозуванні значну частину проблемних задач складає наявнність можливостей обробки великих об’ємів данних, що є великою проблемою для вирішення на одній машині, навіть при її відносно великих потужностяж або експериментальні задачі, розв’язок яких за допомогою. З розвитком і розповсюдженням мережі Web важливість підтрики рішень в реальному часі зростає ще більше, оскільки тільки в цих умовах можна провести вірне прогнозування складних систем . Надійність і масштабовність системи, а також простота використання, впливають на стабільність і швидкість виконання прийнятого рішення. Перевага кластерних систем обробки данних в тому, що вони частково вирішують цю проблему. Методи класифікації і візуалізації дуже гнучкі, і, будучи розширеними на прогнозування, справляются з задачею з найменшими затратами на них.

Актуальність

Сьогодні прогнозування на данних що приходять з масштабних систем стало реалістичним, воно вимагає все більшої обчислювальній потужності. Саме тому дослідники, що стоять на передових наукових позиціях, звертаються до кластерних систем для отримання максимально можливої обчислювальної потужності. Аналіз складності актуальних обчислювальних задач різних галузей науки і техніки показує, що для їх розв’язку необхідні комп’ютери з дуже високою виробничістю. Отримати яку, залишаючись в межах традиційних підходів побудови векторно-конвеєрних систем не можливо. Розв’язок існує лише на шляху широкого розпаралелювання обробки і створення відповідних обчислювальних систем. Для розв’язку поставлених проблем і проводиться розвиток користувацьких сервісів для можливості обробки данних в режимі реального часу, зокрема з використанням паралельних обчислень. Мультипроцесорні системи зараз є найпотужнішими комп'ютерами в світі. Ці машини містять в собі від декількох сотень до декількох тисяч

процесорів в одному корпусі, сполучені з сотнями гігабайтами пам'яті. Мультипроцесорні системи мають величезну обчислювальну потужність і використовуються для вирішення глобальних обчислювальних проблем, таких як глобальне моделювання клімату і розробка ліків.

Останні декілька років ми були свідками постійного розповсюдження паралельних обчислень, як для високопродуктивних наукових обчислень, так і для задач ширшого призначення. А це стало результатом вимог до високої продуктивності, низької вартості і підтримки продуктивності. Популярність використання паралельних обчислень супроводжувалась двома головними досягненнями: масивами паралельних процесорів (мультипроцесорні системи) і використанням розподілених обчислень. Дані великого об’єму являються дуже поширеними на сьогднішній день завдяки поширенності інтернет мережі, та щоденному збільшенню контенту. Всі системи по обробці даних потребують великих обчислювальних можливостей.

Задача обробки надходящих в режимі реального часу даних на сьогоднішній день є ключовою в системах з високою частотою зміни та надходження данних. У всіх цих системах використовуються сервіси для стріміногової обробки данних на кластерних системах.

В перших системах обробки великих данних, дані сприймались, як дещо одиничне статичне й поповнювались з допомогою файлових данних(як бази данних, або cloud системи збереження данних).

Далі з’явилися системи здатні обробляти дані в безперервному потоці — дані що постійно надходять на системи й вимагають збору й обробки для аналізу. Цей тип обробки має в собі, багато застосувань(стрімінг і обробка відео надходячого на ютуб, системи обробки лікарських діагнозів, соціальний аналіз).

Мета

Метою даної роботи є розробка системи що буде розрахована на збір та обробку даних з онлайн ринків(Ebay Amazon Aliexpress). Для цього визначено задачі, що розв’язуються в роботі:

– аналіз існуючих систем збору та обробки великих данних в режимі реального

часу, та вибір найбільш підходячої системи для вирішення задачі;

– підбір алгоритму для прогнозування результатів

– аналіз існуючих архітектур мультиплатформених багатокористувацьких систем і вибір відповідних до поставленої мети;

– створення програмного модулю рішення задачі збору та обробки данних, що задовільняють вимогам реального часу, аналіз часових затрат виконання різних його частин;

– створення на базі отриманих пропозицій та результатів прогностичного модуля;

– визначення ефективності отриманого рішення.

Зв'язок роботи з науковими програмами, планами, темами

Робота виконується у відповідності до планів наукових досліджень, які виконувалися і виконуються на кафедрі СП по вдосконаленню пакета схемотехнічного проектування ALLTED з точки зору подальшого вдосконалення чисельних процедур.

Методи досліджень

Для вирішення поставлених задач використовувалися методи збору та обробки великих даних на багатьох машинах. Перевірка отриманих результатів здійснювалася шляхом проведення обчислювальних експериментів на ЕОМ.

1. ОГЛЯД ПРОБЛЕМАТИКИ. ОПИС ТА ВИБІР ВИКОРИСТОВАНИХ СИСТЕМ ТА МАТЕМАТИЧНИХ МЕТОДІВ

1.1 Огляд проблематики

Процесси збору та аналізу великих данних в режимі реального набувають актуальності тоді коли дані надходять постійно, а не періодичним процессом.

Ще в 2001 році, аналітик Дуг Лані сформулював, на теперішній час, найкраще визначення великих даних, як 3 ключові фактори:

- Об’єм. Багато факторів сприяють збільшенню обсягу данних. Операційні дані, що зберігаються протягом багатьох років. Неструктуровані дані, що збираються напряму з соціальних медіа. Збільшення кількості датчиків і машина-машина збираючихся даних(логування). У минулому, надмірний обсяг даних визивав питання зберігання. Але зі зменшенням витрат на зберігання, виникають інші питання, в тому числі, як визначити актуальність у великих обсягах даних і як використовувати аналітику, щоб створити цінність з відповідних даних.
- Швидкість. Потокові дані з безпрецедентною швидкістю повинні бути оброблені вчасно. Такі системи як: RFID мітки, датчики і смарт вимірювання є рушійною необхідністю боротьби з потоками даних в режимах близьких до реального часу. Адекватна швидка реакція, щоб впоратися зі швидкістю передачі даних, є ціллю для більшості організацій.
- Різноманітність. Дані сьогодні приходять у всіх видах форматів. Структуровані, числові дані у традиційних базах даних. Інформація, створена напряму з бізнес-додатків.

Недоліками систем таких як Apache Hadoop є неможливість стрімінга процесу, тобто ми не можемо використовувати Hadoop систему для процесингу даних приходять в режимі реального часу.

Цю проблему вирішують інші системи обробки данних. Розробка на одній з подібних систем і буде основною темою огляду. Розглянемо одну з подібних проблематик.

1.2 Постановка задачі

Припустимо у нас є системи онлайн маркетів. У кожної знаменитої системи (Ebay, AliExpress, Taobao, Deal Express, Amazon) в середньому в кожен день приблизно по 10 мільйонів людей, які переглядають різні товари на кожному з цих магазинів. Що являється великим підгрунтям для аналізу ринку, адже отримавши інформацію по переходам можна оцінити зацікавленість в ринку, й спрогнозувати те, що можна виводить на ринок на данний час, які закупки виконать, щоб задовольнить потреби ринку.

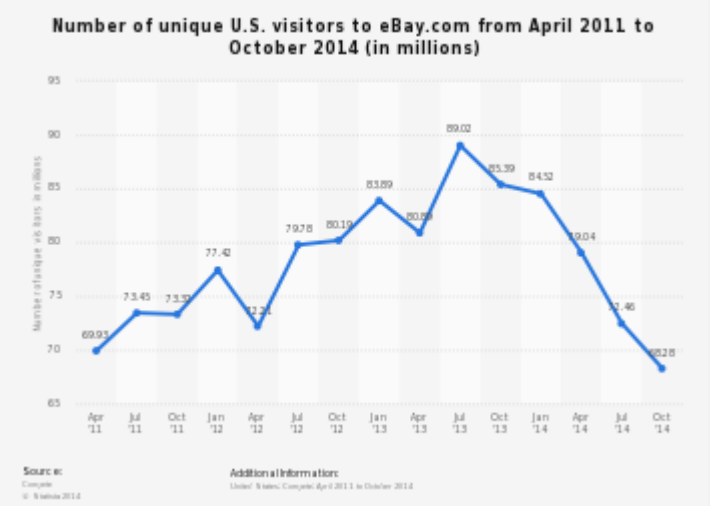

Рисунок 1 Статистика переглядів e-bay з квітня 2011 жовтень 2014

Як ми бачимо за графіком - в середньому у нас 1 мільйон унікальних користувачів в Сполучених Штатах Америки кожен день. Візьмемо за увагу те що статистика не враховує відвідування сайту поза США, відповідно кількість переглядів по світу значно збільшитися. [2]

Мета моєї роботи - зробити систему яка буде уніфікована для всіх подібних магазинів. І зможе оброблювать, аналізувати, виводити дані з докладною статистикою та підтримки прийняття рішення про виведення товару на ринок.

1.3 Вибір методів аналізу.

У вступі було коротко було перераховано методи аналізу. Розглянемо їх більш детально.

Data mining.

Data Mining (нтелектуальний аналіз даних) - загальна назва, що використовується для позначення сукупності методів виявлення в даних раніше невідомих, нетривіальних, практично корисних і доступних інтерпретації знань, необхідних для прийняття рішень у різних сферах людської діяльності. Термін введений Григорієм Пятецким-Шапіро в 1989 році .

Англійське словосполучення «Data Mining» поки не має усталеного перекладу на українську мову. При передачі українською мовою використовуються наступні словосполучення : видобуток даних, вилучення даних, а також, інтелектуальний аналіз даних . Більш повним і точним є словосполучення «виявлення знань в базах даних» (англ. Knowledge discovery in databases, KDD).[5]

Основу методів Data Mining складають всілякі методи класифікації, моделювання і прогнозування, засновані на застосуванні дерев рішень, штучних нейронних мереж, генетичних алгоритмів, еволюційного програмування, асоціативної пам'яті, нечіткої логіки. До методів Data Mining нерідко відносять статистичні методи (дескриптивний аналіз, кореляційний і регресійний аналіз, факторний аналіз, дисперсійний аналіз, компонентний аналіз, дискримінантний аналіз, аналіз часових рядів, аналіз виживаності, аналіз зв'язків). Такі методи, однак, припускають деякі апріорні уявлення про аналізованих даних, що дещо розходиться з цілями Data Mining (виявлення раніше невідомих нетривіальних і практично корисних знань).

Одне з найважливіших призначень методів Data Mining полягає в наочному поданні результатів обчислень (візуалізація), що дозволяє використовувати інструментарій Data Mining людьми, які мають спеціальної

математичної підготовки. У той же час, застосування статистичних методів аналізу даних вимагає доброго володіння теорією ймовірностей і математичної статистикою.

Задачі, які вирішуються методами Data Mining, прийнято розділяти на описові (англ. Descriptive) і передбачувальні (англ. Predictive).

В описових завданнях найголовніше - це дати наочне опис наявних прихованих закономірностей, в той час як в передбачувальних завданнях на першому плані стоїть питан ня про прогнозування для тих випадків, для яких даних ще немає.[6]

До описових завдань відносяться:

- Пошук асоціативних правил або патернів (зразків);
- Групування об'єктів, кластерний аналіз;
- Побудова регресійної моделі.

До прогнозування відносяться:

- класифікація об'єктів (для заздалегідь заданих класів);
- регресійний аналіз, аналіз часових рядів.

В нашому випадку доцільно використовувати аналіз часових рядів, оскільки класифікація й групування запитів робиться ріалтайм системою збору, й базової обробки.

Часовий ряд (англ. Time series) - реалізація випадкового процесу, набір послідовних результатів спостереження. Приклади часових рядів кількість сонячних плям, сила вітру, зміна курсу валюти. Часовий ряд дуже часто побудовані за допомогою лінійних діаграм. Тимчасові ряди використовуються в статистиці, обробки сигналів, розпізнавання образів, економетрики, прогнозування погоди, передбачення землетрусів, електроенцефалографія, контроль інженерних даних, астрономії, інженерних комунікацій, і в значній мірі застосовується в наукових дослідженнях і техніки, який включає часові

виміри.

Аналіз часових рядів включає методи аналізу часових рядів для того, щоб витягти корисну статистику та інші характеристики даних. Прогнозування часових рядів є використання моделі для прогнозування майбутніх значень на основі раніше спостережуваних значень. У той час як регресійний аналіз часто використовується таким чином, як для перевірки теорій, поточні значення одного або більше незалежних часових рядів впливають на поточне значення іншої тимчасової серії, цей тип аналізу часових рядів не називається «Аналіз часових рядів», яка фокусується на порівнянні значень одного часового ряду або декількох залежних часових рядів в різні моменти часу.

Дані часових рядів мають природний тимчасової порядок. Це робить аналіз часових рядів відміну від перехресних досліджень, в яких немає природного упорядкування спостережень (наприклад, пояснюючи заробітної плати людей з урахуванням їх відповідних рівнів освіти, де дані фізичних осіб можуть бути введені в будь-якому порядку). Аналіз часових рядів також відрізняється від аналізу просторових даних, де спостереження, як правило, відносяться до географічним положенням (наприклад, врахування цін на житло за місцем знаходження, а також внутрішні характеристики будинку). Крім того, моделі часових рядів часто роблять використання природного часу в односторонньому порядку, так що значення для даного періоду буде виражатися в отриманні в якийсь із минулих значень, а не з майбутніх значень.

Прогнозні оцінки за допомогою методів екстраполяції розраховуються в кілька етапів:

- перевірка базової лінії прогнозу;
- виявлення закономірностей минулого розвитку явища;
- оцінка ступеня достовірності виявленої закономірності розвитку явища в минулому (підбір трендової функції);
- екстраполяція - перенесення виявлених закономірностей на деякий період

майбутнього;

- коригування отриманого прогнозу з урахуванням результатів змістовного аналізу поточного стану.

Для отримання об'єктивного прогнозу розвитку досліджуваного явища дані базової лінії повинні відповідати таким вимогам:

- крок за часом для всієї базової лінії повинен бути однаковий;
- спостереження фіксуються в один і той же момент кожного часового відрізку (наприклад, на полудень кожен день, першого числа кожного місяця);
- базова лінія повинна бути повною, тобто пропуск не допускається.

Якщо при спостереженні відсутні результати за незначний відрізок часу, то для забезпечення повноти базової лінії необхідно їх заповнити приблизними даними, наприклад, використовувати середнє значення сусідніх відрізків.

Коригування отриманого прогнозу виконується для уточнення отриманих довгострокових прогнозів з урахуванням впливу сезонності або стрибкоподібність розвитку досліджуваного явища.

1.4 Вибір програмних сервісів

1.4.1 Apache Kafka

Apache Kafka - розподілене програмний брокер повідомлень, проект з відкритим вихідним кодом, розроблений в рамках Apache Software Foundation. Написаний на мові програмування Scala.

Відмінності від традиційних системи передачі повідомлень:

• спроектований спочатку як розподілена система, яку легко масштабувати

• підтримує високу пропускну здатність як з боку джерел, так і для систем-споживачів

• підтримує об'єднання споживачів в групи,

• забезпечує можливість тимчасового зберігання даних дляподальшої пакетної обробки.

Однією з особливостей реалізації інструменту є застосування техніки, подібної з журналами транзакцій, використовуваними в системах управління базами даних.

Кафка є розподілена, розділена, відновнувана система обробки повідомлень. Це забезпечує функціональність системи обміну повідомленнями.

Що все це означає?

По-перше, розглянемо деякі базові терміни повідомленнями:

Кафка веде канали повідомлень в категорії, які називаютя "теми".

Ми будемо називати процеси, які публікують повідомлення Кафки "виробники тем".

Ми будемо називати процеси, які підписуються на повідомлення і обробляють канал опублікованих повідомлень "споживачі" ..

Кафка працює як кластер, що складається з одного або декількох серверів, кожен з яких називається брокером.

Так, на високому рівні, виробники відправляють повідомлення по мережі кластера Кафки, який, у свою чергу відправляє їх до споживачів, як це показано на діаграмі:

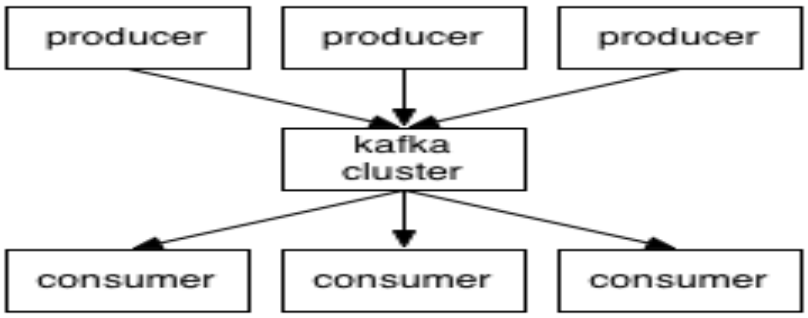

Рисунок 2 Схема Кафка кластера

Зв'язок між клієнтами і серверами здійснюється за допомогою простого, високо продуктивного протоколу TCP. Ми використовуєм клієнт Java для Кафки, але клієнти доступні багатьма мовами.

Теми і Журнали

Давайте спочатку зануримося в абстракцію високого рівня яку Кафка забезпечує - тему.

Тема є першою категорією якої повідомлення будуть опубліковані. По кожній темі, кластер Кафка веде многороздільний журнал, який виглядає наступним чином:

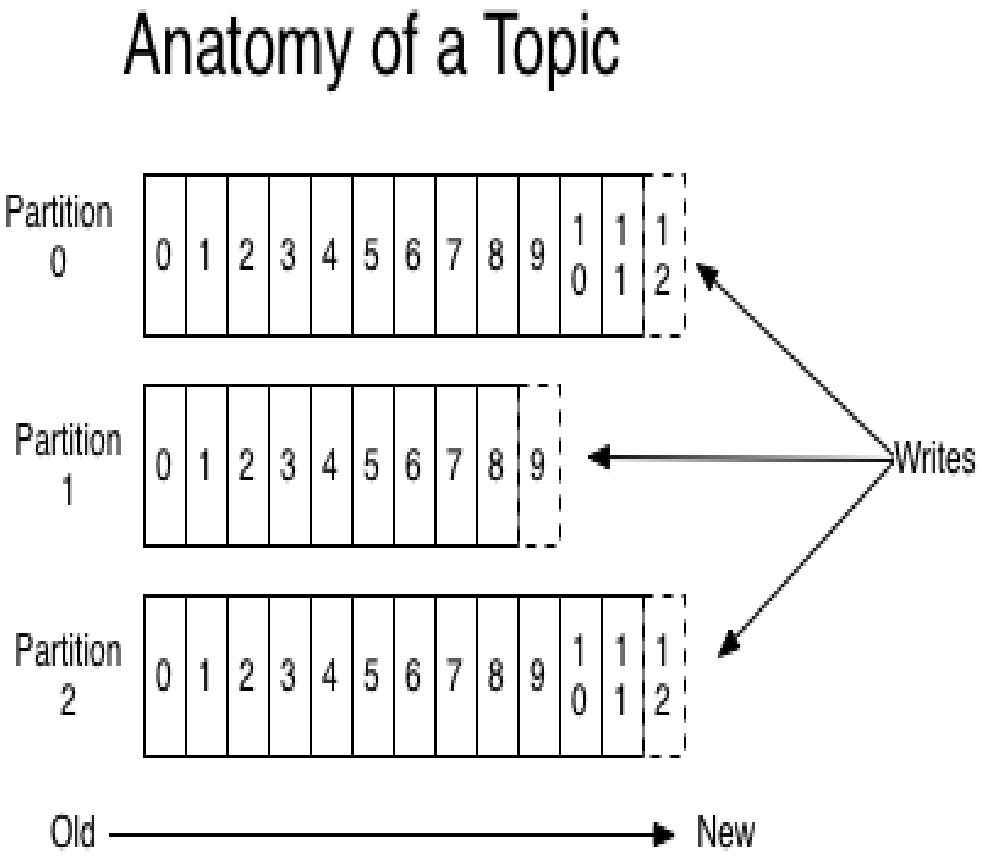

Рисунок 3 Схема теми

Кожен розділ являється частиною, незмінної послідовності повідомлень, які постійно додаються в журнал. Кожним повідомленням в розділах призначений порядковий номер ID який називається зміщенням, та однозначно ідентифікує кожне повідомлення в розділі.

Кафка кластер зберігає всі опубліковані повідомлення, будь вони чи ні, на заданий період часу. Наприклад, якщо збереження журналу встановлено на протяжність двох днів, потім протягом двох днів після публікації повідомлення воно доступне для споживання, після чого воно буде відкинутий, щоб звільнити місце. Продуктивність Кафки ефективно постійна по відношенню до розміру даних так що, збереження безлічі даних не є проблемою.

Насправді тільки метадані зберігаються на кожного споживача, становище споживача в журналі, називається "зсув". Цей зсув контролюється

споживачем: зазвичай споживач буде просувати його зміщення лінійно, як він читає повідомлення, але насправді становище його регулюється споживачем, і він може споживати повідомлення в будь-якому порядку, якому захоче. Наприклад, споживач може скинути з старше зміщення для переробки.

Ця комбінація особливостей означає, що Кафка споживачі являються дуже дешевими: вони приходять і йдуть без особливих впливів на кластері або на інших споживачів. Наприклад, ви можете використовувати наші інструменти командного рядка для "хвоста" вмісту будь-якої теми без змін, що споживається існуючими споживачами.

Перегородки в журналі служать декільком цілям. По-перше, вони дозволяють журналу масштабироваться за межі розміру, який поміщається на одному сервері. Кожен розділ повинен відповідати на серверах, на яких розміщені, але тема може мати багато розділів, так що можуть обробляти довільну кількість даних. По-друге, вони виступають в якості одиниці паралелізму.

1.4.2 Apache Hbase

HBase - нереляційна розподілена база даних з відкритим вихідним кодом; написана на Java; є аналогом Google BigTable. Розробляється в рамках проекту Hadoop фонду Apache Software Foundation. Працює поверх розподіленої файлової системи HDFS і забезпечує BigTable-подібні можливості для Hadoop, тобто забезпечує відмовостійкий спосіб зберігання великих обсягів розріджених даних.

Підтримка компресії, операції в пам'яті і фільтра Блума для кожного базового стовпчика реалізовані в HBase відповідно до документації BigTable. Таблиці в HBase можуть служити входом і виходом для роботи реалізації MapReduce в проекті Hadoop, і можуть бути отримані, не тільки через Java API, але й через API REST, Avro або Thrift.

HBase не є прямою заміною класичних SQL баз даних, хоча останнім часом у цій сфері вона стала працювати значно краще і в даний час

використовується для управління даними на кількох веб-сайтах, в тому числі Facebook використовує її для своєї платформи повідомлень.

Додатки зберігають дані в таблицях, що складаються з рядків і стовпців. Для елементів таблиці (перетину рядків і стовпців) діє контроль версії. За замовчуванням як версії використовується тимчасова мітка, автоматично призначається HBase на момент вставки. Вміст комірки являє собою неінтерпретіруемий масив байтів.

Ключі рядків таблиці теж є байтовими масивами, тому теоретично ключем рядка може бути що завгодно - від рядків до бінарних уявлень long і навіть серіалізованих структур даних. Рядки таблиці упорядковано по ключу рядків (первинному ключу таблиці). Сортування здійснюється в порядку проходження байтів. Усі звернення до таблиці виконуються по первинному ключу. Стовпці об'єднуються в сімейства стовпців. Всі члени сімейства стовпців мають загальний префікс, так наприклад, стовпці temperature: air та temperature: dew_ point належать сімейству temperature, a station: identifier належить сімейству station. Префікс сімейства стовпців повинен складатися з друкованих символів. Завершальна частина (кваліфікатор) може складатися з довільних байтів.

Сімейства стовпців таблиці повинні бути задані заздалегідь, як частина визначення схеми таблиці, проте нові члени родин можуть додаватися в міру потреби. Наприклад, новий стовпець station: address може бути переданий клієнту як частина оновлення, і його значення буде успішно зберігатися - за умови, що сімейство стовпців station вже існує в таблиці. Фізично всі члени родин стовпців зберігаються разом у файловій системі. Таким чином HBase як столбцево-орієнтоване сховище інформації, точніше було б сказати «орієнтоване на сімейства стовпців». Так як налаштування і специфікації задаються на рівні родин стовпців, бажано, щоб всі члени родин мали подібні схеми доступу та характеристики розмірів.

HBase автоматично виробляє горизонтальну розбивку таблиць на регіони.

Кожен регіон утворює підмножину рядків таблиці. Регіон визначається таблицею, якій він належить, своїм першим рядком (включно) і останнім рядком (без включення). Спочатку таблиця складається з одного регіону, але із зростанням розміру регіону після перевищення настроюваного порогового розміру він розбивається на два нових регіони приблизно рівних розмірів. До першого розбиття вся завантаження даних буде здійснюватися на одному сервері, на якому розміщений вихідний регіон. У міру зростання таблиці збільшується кількість її регіонів. Регіони є одиницями, розподілюваними в кластері HBase. Якщо таблиця виявляється занадто великою для одного окремого серверу, вона може обслуговуватися кластером серверів, на кожному вузлі якого розміщується підмножина регіонів таблиці. Крім того, регіони забезпечують розподіл навантаження на таблицю. Сукупність відсортованих регіонів, доступних по мережі, утворює загальний вміст таблиці.

У нашому випадку він використовується як метасховищк періодично (задавано) класіфіціковних параметрів, за певний період часу.

1.4.3 Apache Storm

В Минулому десятилітті відбулася революція в обробці даних. MapReduce, Hadoop, і пов'язані з ними технології зробили можливим зберігання і обробки даних в масштабах, раніше немислимим. На жаль, ці дані технології обробки не є системами обробки в реальному часі, і вони не повинні бути. Немає способі що перетворить Hadoop в систему обробкм в режимі реального часу; в реальному часі обробка даних має принципово інший набір вимог, ніж пакетна обробки.

Однак, обробка даних в реальному часі в масовому масштабі стає все більшою і більшою вимогою для бізнесу. Відсутність "Hadoop в реальному часі" стала найбільшою дірою в екосистемі обробки даних.

Шторм заповнює цю дірку.

Перед Storm, вам, як правило, доведеться вручну побудувати мережу черг і робітників, щоб зробити обробку в реальному часі. Працівники будуть

обробляти повідомлення з черги, оновлювать бази, і відправляти нові повідомлення для інших черг для подальшої обробки. На жаль, цей підхід має серйозні обмеження:

- Занудство : Ви проводите більшу частину часу розробки в налаштуванні того куди відправляти повідомлення, розгортання робітників і розгортання проміжних черг. Логіці обробки в реальному часі відповідає порівняно невеликий відсоток від вашого коду.
- Крихке : Там буде невелика відмовостійкість. Ви несете відповідальність за збереження данних кожного робочого стояти в черзі.
- Неможливість масштабувати: Коли пропускна потреба повідомлення занадто високою для одного працівника або черги, необхідно розділити дані по іншим працівникам. Ви повинні переналаштувати інших працівників, щоб дізнатися про нові доступні місця, для відправки повідомлення. Хоча черги і робочі парадигми ламаються при великій кількості повідомлень, обробка повідомлення явно фундаментальною парадигмою обчислень в реальному часі. Питання: як ви зроюити це таким чином, щоб не втрачати дані, ваги для великих обсягів повідомлень і зберегти простоту у використанні і експлуатації?

Шторм задовольняє цим цілям.

Чому Шторм є важливим?

Шторм надає набір примітивів для ведення обчислень в реальному часі. Наприклад, як MapReduce значно полегшує написання паралельних пакетних обробон, примітиви шторму значно полегшають написання обчислень паралельного реального часу.

Ключові властивості Storm є:

- Надзвичайно широкий набір варіантів використання: Storm може бути використаний для обробки повідомлень і оновлення баз даних (обробка потоку), роблячи безперервний запит на потоки даних і потокових

результатів в клієнтів (безперервних) обчислювальних, розпаралелювання інтенсивних запит як пошуковий запит на льоту (віддалений RPC), і багато іншого. Невеликий набір примітивів задовільняє приголомшливу кількість випадків використання.

- Масштабованість: Storm діє для величезної кількості повідомлень в секунду. Для масштабування топології, все, що вам потрібно зробити, це додати машини і збільшити параметри паралелізму топології. Як приклад в масштабі Шторм, один з первинних заявок шторму оброблено 1000000 повідомлень в секунду на 10 вузлах кластера, в тому числі сотні звернень до бази даних в секунду, як частина топології. Використання Storm з Zookeeper для координації кластера робить масштабироваться до більш великих розмірів кластера.
- Гарантує відсутність втрати даних: система реального часу повинні мати тверді гарантії про успішну обробку даних. Система, яка падає, має дуже обмежені можливості використання. Шторм гарантує, що кожне повідомлення буде оброблено, і це знаходиться в прямій суперечності з іншими системами, такими як S4.
- Надзвичайно міцна: На відміну від систем, таких як Hadoop, які славляться тим що ними важко керувати, Storm кластери просто працюють. Це явною метою Буря проекту, щоб зробити користувальницький досвід управління Storm кластерами, безболісним, наскільки це можливо.
- Відмовостійкі: Якщо є помилки під час виконання ваших обчислень, шторм перепризначить завдання по мірі необхідності. Шторм гарантує, що обчислення може працювати вічно (або до тих пір, поки ви не вб’єте обчислення).
- Не важливість мови програмування : Надійна і масштабований реальному часі обробка не повинна бути обмеженою однією платформою. Storm топології і компоненти обробки можуть бути визначені в будь-якій

мові, що робить доступним Storm майже для кожного.

Компоненти Storm кластера

Шторм кластера зовні схожі на кластери Hadoop. У той час як на Hadoop запуску "MapReduce job", на Storm ви запускаєте "topology". "MapReduce job" і "топології" по суті дуже різні - основна відмінність в тому, що “MapReduce job” в кінцевому підсумку закінчується, в той час як “topology” обробляє повідомлення завжди (або до тих пір, поки її не вбити).

Є два види вузлів шторм кластера: головний вузол і робочих вузлів. Головний вузол працює демон, званий Nimbus, який схожий на JobTracker Hadoop Nimbus несе відповідальність за поширення коду в кластері, призначення завдань машин та моніторингу невдач.

На Кожному працюючому вузлі запущений демон, так званий Supervisor. Supervisor слухає роботу з що призначається його машині і запускає та зупиняє робочі процеси в міру необхідності, на основі того, що Nimbus присвоїв йому. Кожен робочий процес виконує підмножину топології; працююча топологія складається з багатьох робочих процесів, розкиданих по багатьох машинах.

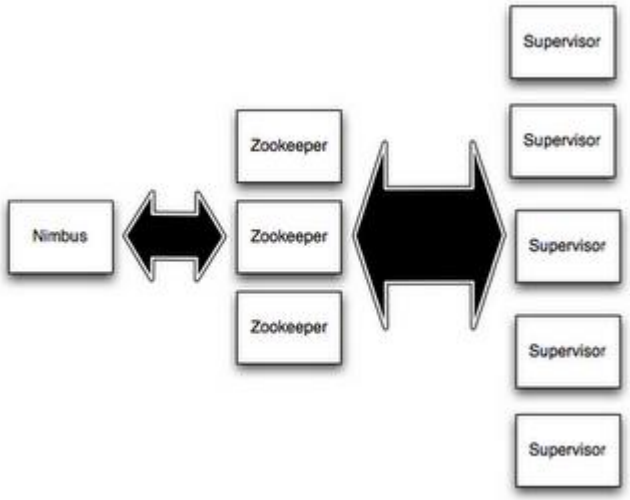

Рисунок 4 Взаємодія Nimbus з Supervisor

Вся координація між Nimbus та Supervisors здійснюється через кластер Zookeeper. Крім того, Керівник демони Німб є стійкими до збоїв та падіннь;

весь стан зберігається в Zookeeper або в локальному диску. Це означає, що ви можете виконати “kill -9” для Німба або наглядачів, і вони відновлять свою роботу знову, ніби нічого не сталося. Ця конструкція призводить до неймовірної стабільності.

Топології

Для обчислення в реальному часі на Storm, ви створюєте те, що називається "топологіями". Топологія представляє собою граф обчислень. Кожен вузол в топології містить логіку обробки, а зв'язки між вузлами показують, як дані повинні бути передані між вузлами навколо.

Запуск топології простий. По-перше, ви упаковуєте весь ваш код і залежності в одному jar архіві. Потім запускаєте команду,:

storm jar code.jar backtype.storm.MyTopology arg1 arg2

Це запускає клас backtype.storm.MyTopology з аргументами arg1 arg2 . Функція main класу визначає топологію і відправляє її Nimbus. Storm jar частина піклується про підключення до Nimbus і завантаження jar-архіву.

Визначеннями топології є Thrift структури(компонент, що буде розглянутий пізніше), а Nimbus є Thrift сервісом, ви можете створити і представляти топології, використовуючи будь-яку мову програмування. Наведений вище приклад є простий спосіб зробити це з JVM на основі мови Java.

Потоки

Центровою абстракцією в Storm є "потік". Потік необмежена послідовність кортежів. Шторм забезпечує примітиви для перетворення потоку в новий потік в розподілений та надійний спосіб. Наприклад, ви можете перетворити потік твітів в потік трендових тем.

Основні примітиви Шторм що дозволяють робити потокові перетворення є Spouts і Bolt. Spout і Bolt мають інтерфейси, які ви повинні бути реалізуваними, щоб запустити логіку програми.

Spout є джерелом потоків. Наприклад, Spout може зчитувати кортежі з черги Kestrel і випромінювати їх у вигляді потоку. Або Bolt можна підключити до API Twitter і випускати потік твітів.

Bolt може отримувать будь-яку кількість вхідних потоків, робить деяку обробку і, можливо, випускати нові потоки. Комплексні перетворення потоку, як обчислення потоку трендів з потоку твітів, вимагають декількох кроків і, таким чином, декілька болтів. Bolt'и можуть робити що-небудь з працюючих функцій, фільтрів кортежів, зробити потокове агрегати, робити потокове приєднання, спілкуватись з базами даних, і багато іншого.

Мережі Spout і Bolt упаковані в "топологію", яка є абстракцією верхнього рівня, що ви надаєте Storm-кластерам для виконання. Топологія представляє собою граф потоку перетворень, де кожен вузол є Spout або Bolt. Ребра в графі вказують, які bolt'и взаємодіють з якими потоками. Коли spout або bolt відправляє кортеж в потік, він посилає кортеж кожному bot, що підписався на цей потік.

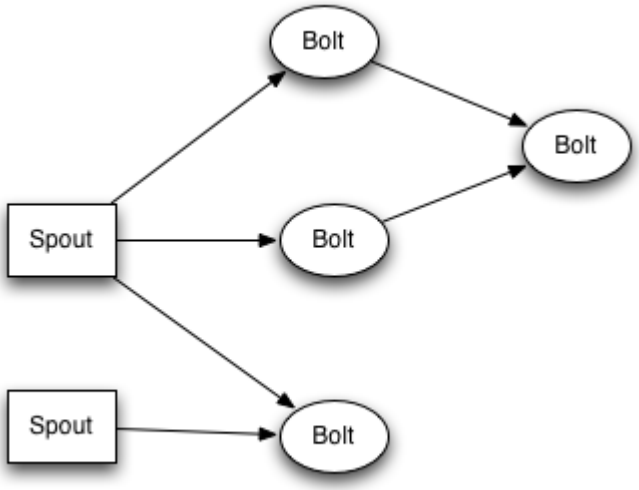

Рисунок 5 Типова взаємодія між процесами в клатері Storm [3]

Зв'язки між вузлами в топології вказують, як кортежі мають бути передані на наступний рівень. Наприклад, якщо є зв'язок між Spout A і Bolt B, ребро з Spout A до Bolt C, і з Bolt B в Bolt C, то відповідно кожен Spout випромінюючи кортеж, буде посилати як Bolt B так і Bolt C. Всі вихідні кортежі Bolt B підуть до Bolt C.

Кожен вузол в топології Storm виконує паралельно. У топології, ви можете вказати, скільки паралельних процесів ви хочете запуститти для кожного вузла, а потім Storm буде породжувати таку кількість потоків в кластері, щоб зробити обчислення.

Топологія працює вічно, або до того момента поки її не вбити. Storm буде автоматично перепризначувати будь невдалі завдання. Крім того, Storm гарантує, що не буде ніякої втрати даних, навіть якщо машини зупинять роботу і повідомлення видалються.

1.4.4 Java

Для реалізації Storm Hbase Kafka частин була обрана мова програмування java

оскільки вона поєднується з кожним з цих компонент.

Java є мовою програмування загального призначення, яка є багатопотоковою та об'єктно-орієнтованою, також спеціально розроблена, щоб

мати стільки імплементації залежностей наскільки це можливо. Вона призначена, щоб розробники додатків "написали один раз, і вона працює скрізь" (Wora), [13] це означає, що скомпілений Java код може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. Java додатки, як правило, збирається у байт-код що може працювати на будь віртуальній машині Java (JVM) незалежно від комп'ютерної архітектури. На 2015, Java є одним з найбільш популярних мов програмування у використанні, особливо для клієнт-серверних веб-додатків, з 9 млн зареєстрованих розробників. Java спочатку розроблена Джеймсом Гослінгом в Sun Microsystems (яка згодом була придбана корпорацією Oracle) і випущена в 1995 році в якості основного компонента Java платформи Sun Microsystems. Мова запозичила більшість синтаксису з C і C ++, але вона має менше низькорівневих можливостей, ніж будь-який з них.

Оригінальні Java компілятори, віртуальні машини і бібліотеки класів були спочатку випущений у SUN в пропрієтарних ліцензіях. Станом на травень 2007 року, у відповідності зі специфікаціями Java Community Process, Sun повторно ліцензувала більшість своїх технологій Java під GNU General Public License. Інші також розробили альтернативні реалізації цих технологій Sun, таких як GNU Compiler для Java (компілятор байт-код), GNU класи (стандартні бібліотеки), і IcedTea-Web (плагіни браузера для аплетів).

Java Virtual Machine (JVM) є абстрактною обчислювальною машиною. Є три тези про JVM: специфікація, реалізація та екземпляр. Специфікація це те, що формально описує, що потрібно від реалізації JVM. Наявність єдиної специфікації гарантує, що всі реалізації сумісні. Реалізація JVM є комп'ютерною програмою, яка відповідає вимогам специфікації JVM. екземпляр JVM це процес, який виконує комп'ютерну програму скомпільовану в Java байт-код.

Метою Java є портативність, що означає, що програми, написані для платформи Java можна запустити на будь-якій комбінації апаратного

забезпечення та операційної системи з адекватною підтримкою виконання. Це досягається шляхом компіляції коду Java мовою проміжного представлення так званий байт-код Java, а не безпосередньо до конкретної архітектури машинного коду. Інструкція Java байт-коду аналогічні машинному код, але вони призначені для виконання віртуальною машиною (VM), написано спеціально для серверного обладнання. Кінцеві користувачі зазвичай використовують Java Runtime Environment (JRE) встановлені на їх власній машині для автономних додатків Java, або у веб-браузері для Java-аплетів.

Стандартні бібліотеки забезпечують загальний спосіб доступу до специфічних функцій, таких як графіка, багатопотоковість , та networking.

Основна перевага використання байт-коду є можливість його портування. Тим не менш, додадткові витрати на інтерпретацію означають, що інтерпретовні програми майже завжди працювати повільніше, ніж скомпільовані . Java є незалежною від платформи. Але Java віртуальній машині необхідно перетворити Java байт-код в машинну мову, яка залежить від операційної системи, а це залежить від платформи.

Програми, написані на Java мають репутацію таких що працюють повільніше і вимагають більше пам'яті, ніж ті, які написані на C ++. Тим не менш, швидкість виконання програм Java "значно покращлась з введенням JIT компіляції в 1997/1998 для Java 1.1, додавання мовних особливостей, що підтримують більш глибокий аналіз коду (наприклад, внутрішніх класів, клас StringBuilder, додаткових тверджень і т.д.), і оптимізацій в віртуальній машині Java, таких як HotSpot стає за замовчуванням для SUN JVM в 2000.

Деякі платформи пропонують пряму підтримку апаратного забезпечення для Java; Є мікроконтролери, які можуть відпрацьовувати Java на апаратному рівні, а не програмному забезпеченні Java Virtual Machine, і процесори базрвані на ARM основі може мати апаратну підтримку для виконання Java байт-код через опції Jazelle.

Автоматичне управління пам'яттю

Java використовує автоматичний збирач сміття для управління пам'яттю в життєвому циклі об'єктів. Програміст визначає, коли об'єкти створюються і Java несе відповідальність за відновлення пам'яті, як тільки об'єкти більше не використовуються. Після того, як жодного посилання на об'єкт не залишається, зайнята пам'ять може бути звільненою автоматично збирачем сміття. Щось подібне до витоку пам'яті все ще може статися, якщо код програміста містить посилання на об'єкт, який більше не потрібен, як правило, коли об'єкти, які більше не потрібні зберігаються в контейнерах, які знаходяться все ще у використанні. Якщо методи для неіснуючого об'єкта визвуться то виникне "null pointer exception".

Одна з ідей, за автоматичної моделі управління пам'яттю в Java є те, що програмісти можуть бути позбавлені від тягаря того, щоб думати про ручне управління пам'яттю. У деяких мовах, пам'ять для створення об'єктів неявно виділяється в стеку, або явно виділяється і звільняється з купи. В останньому випадку відповідальність за управління пам'яттю лежить на програмісті. Якщо програма не звільняє об'єкт, виникає витік пам'яті. Якщо програма намагається отримати доступ або звільняти пам'ять, яка вже була звільнена, то результат не визначений і важко передбачити що трапиться, програма може стати нестабільною або закрешиться. Це може бути частково усунені шляхом використання смарт-вказівників, але вони додають накладні витрати і складність. Зверніть увагу, що збір сміття не заважає "логічному" витоку пам'яті, тобто ті виапдки, де на пам'ять і раніше посилаються, але ніколи не використовується.

Збір сміття може відбутися в будь-який момент. В ідеалі, це відбудеться, коли програма знаходиться в режимі очікування. Це гарантовано спрацьовує при недостатній кількості вільної пам'яті в купі для виділення нового об'єкта. Явне управління пам'яттю не є можливим в Java.

Java не підтримує C / C ++ стиль арифметики покажчиків, де об'єкт адреси та беззнакових цілих чисел (зазвичай довгі цілі) можуть бути

взаємозамінні. Це дозволяє збирачу сміття переміщувати посилання на об’єкти і забезпечує безпеку типів .

Як і в C ++ і деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java не є об'єктами. Значення примітивних типів, або зберігаються безпосередньо в полях (об'єктів) або в стеку (для методів), а не в купі. Це було свідоме рішення дизайнерів Java для підвищення продуктивності. Через це, Java не вважається чистим об'єктно-орієнтованою мовою програмування. Проте, станом на Java 5.0, Autoboxing що дозволяє програмістам, процесить ніби примітивні типи були станами їх класу-оболонки.

Java містить кілька типів збирачів сміття. За замовчуванням, HotSpot використовує збирач сміття паралельної продувки. Тим не менш, є також кілька інших збирачів сміття, які можуть бути використані для управління купою пам’яті. Для 90% додатків в Java, Паралельний Марк-розгортальний збирач сміття достатній. Oracle прагне замінити CMS зі сміттям першим колектором(G1).

1.4.5 Python

Для розробки взаємодії ріалтайм системи з користувачем було розроблено вебсервер що отримує данні з системи та через взаємодію з веб інтерфойсом виводить результат користувачеві.

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом з динамічною семантикою і динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python і стандартні бібліотеки доступні як в скомпільованій так і у вихідній формі на всіх основних платформах. У мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна і аспектно-орієнтований.

Python транслятори доступні для установки на багатьох операційних системах, дозволяючи виконання коду Python на різноманітних системах. Використання сторонніх інструментів, таких як py2exe або Pyinstaller, дозволяє Python коду бути упакованим в автономних виконуваних програмах для деяких з найбільш популярних операційних систем, що дозволяє для розповсюдженню програмного забезпечення на основі Python для використання на цих середовищах без установки інтерпретатора Python.

Серед основних її переваг можна назвати наступні:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносимість програм (що властиво більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включаючи модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисно для експериментування та рішення простих задач);
- стандартний дистрибутив має просте, але разом з тим досить потужне середовище розробки, яка називається IDLE і яке написано на мові Python;
- зручний для вирішення математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, в діалоговому режимі може використовуватися як потужний калькулятор).

Python має ефективні структури даних високого рівня і простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки додатків в багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата стандартна бібліотека (як початкові тексти, так і бінарні дистрибутиви для всіх основних операційних систем)

можуть бути отримані з сайту Python www.python.org, і можуть вільно поширюватися. Цей же сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями і типами даних, розробленими на C або C ++ (або іншою мовою, яку можна викликати за C). Python також зручна як мова розширення для додатків, що вимагають подальшого налагодження.

Розробники мови Python є прихильниками певної філософії програмування, яку називають «The Zen of Python» («Дзен Пайтона») . Її текст можна отримати в інтерпретаторі Python за допомогою команди import this (один раз за сесію). Автором цієї філософії вважається Тім Пейтерс.

Текст філософії:

- Гарне краще, ніж потворне.
- Явна краще, ніж неявне.
- Просте краще, ніж складне.
- Складне краще, ніж заплутане.
- Плоске краще, ніж вкладене.
- Розріджене краще, ніж щільне.
- Легкість читання має значення.
- Особливі випадки не настільки особливі, щоб порушувати правила.
- При цьому практичність важливіше бездоганності.
- Помилки ніколи не повинні замовчуватися.
- Якщо не замовчуються явно.
- Зустрівши двозначність, відкинь спокуса вгадати.
- Повинен існувати один - і, бажано, тільки один - очевидний спосіб

зробити це.

- Хоча спочатку він може бути і не очевидним, якщо ви не голландець [11].
- Зараз краще, ніж ніколи.
- Хоча ніколи, як правило, краще, ніж просто зараз.
- Якщо реалізацію важко пояснити - ідея погана.
- Якщо реалізацію легко пояснити - ідея, можливо, хороша.
- Простір імен - чудова річ! Будемо робити їх побільше!

Python портовано і працює майже на всіх відомих платформах - від КПК до мейнфреймів. Існують порти під Microsoft Windows, всі варіанти UNIX (включаючи FreeBSD і GNU / Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS / 2, Amiga, AS / 400 і навіть OS / 390, Symbian і Android [14].

У міру старіння платформи її підтримка в основний гілці мови припиняється. Наприклад, із серії 2.6 припинена підтримка Windows 95, Windows 98 і Windows ME. Однак на цих платформах можна використовувати попередні версії Python - тепер співтовариство активно підтримує версії Python починаючи від 2.3 (для них виходять виправлення).

При цьому, на відміну від багатьох портованих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM / DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java - Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, яка підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python і навіть бути написаними на ньому. Також кілька проектів забезпечують інтеграцію з платформою Microsoft.NET, основні з яких - IronPython і Python.Net.

Python, як і багато інших різних мов, не застосовують, наприклад, JIT-компілятори, мають загальний недолік - порівняно невисоку швидкість виконання програм. Однак, у випадку з Python цей недолік компенсується

зменшенням часу розробки програми. В середньому програма, написана на Python, в 2-4 рази компактніше, ніж її аналог на C ++ або Java. Збереження байт-коду (файли .pyc і .pyo) дозволяє інтерпретатору не витрачати зайвий час на перекомпіляцію коду модулів при кожному запуску, на відміну, наприклад, від мови Perl. Крім того, існує спеціальна JIT-бібліотека psyco (проте призводить до збільшення споживання оперативної пам'яті). Ефективність psyco в значній мірі залежить від архітектури програми.

Існують проекти реалізацій мови Python, що вводять високопродуктивні віртуальні машини (ВМ) як компілятора заднього плану. Прикладами таких реалізацій може служити PyPy, заснований на LLVM; більш ранньої ініціативою є проект Parrot. Очікується, що використання ВМ типу LLVM призведе до тих самих результатів, що і використання аналогічних підходів для реалізацій мови Java, де низька обчислювальна продуктивність в основному подолана.

Безліч програм / бібліотек для інтеграції з іншими мовами програмування надають можливість використовувати іншу мову для написання критичних ділянок.

У найпопулярнішій реалізації мови Python інтерпретатор досить великий і більш вимогливий до ресурсів, ніж в аналогічних популярних реалізаціях Tcl, Forth, LISP або Lua, що обмежує його застосування у вбудованих системах. Тим не менш, Python знайшов застосування в КПК і деяких моделях мобільних телефонів

1.4.6 Django

Django (Джанго) — високорівневий відкритий Python-фреймворк для розробки веб-систем. Названо його було на честь джазмена Джанго Рейнхардта (відповідно до музичних смаків одного зі засновників проекту).

Сайт на Django будується з однієї або декількох частин, які рекомендується робити модульними. Це одна з істотних архітектурних відмінностей цього фреймворку від деяких інших (наприклад Ruby on Rails).

Архітектура Django подібна на «Модель-Вид-Контролер» (MVC). Однак, те що називається «контролером» в класичній моделі MVC, в Django називається «вид» (англ. view), а те, що мало б бути «видом», називається «шаблон» (англ. template). Таким чином, MVC розробники Django називають MTV («Модель-Шаблон-Вид»).

Початкова розробка Django, як засобу для роботи новинних ресурсів, досить сильно позначилася на його архітектурі: він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований модуль для керування вмістом, який можна включити в будь-який сайт, зроблений на Django, і який може керувати відразу декількома сайтами на одному сервері. Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючі всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав).

У дистрибутив Django також включені програми для системи коментарів, синдикації RSS і Atom, «статичних сторінок»(якими можна управляти без необхідності писати контролери та відображення), перенаправлення URL та інше.

Django був створений для управління сайтами новин LJWorld.com, lawrence.com і KUsports.com компанії The World Company (Лоуренс, Канзас[en]), США), але з моменту початку розповсюдження його у статусі відкритого програмного забезпечення отримав величезну популярність в усьому світі як платформа до численних систем.

Розробники — засновники проекту:

- Адріан Головатий
- Саймон Віллісон (англ. Simon Willison),

- Джекоб Каплан-Мосс (англ. Jacob Kaplan-Moss),
- Вілсон Майнер (англ. Wilson Miner)

Деякі можливості Django:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами на більшість мов;
- диспетчер URL на основі регулярних виразів;
- розширювана система шаблонів з тегами та наслідуванням;
- система кешування;
- інтернаціоналізація;
- архітектура застосунків, що підключаються, які можна встановлювати на будь-які Django-сайти;
- «generic views» - шаблони функцій контролерів;
- авторизація та аутентифікація, підключення зовнішніх модулів аутентифікації: LDAP, OpenID та ін.;
- система фільтрів («middleware») для побудови додаткових обробників запитів, наприклад включені в дистрибутив фільтри для кешування, стиснення, нормалізації URL і підтримки анонімних сесій;
- бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД);
- вбудована автоматична документація по тегам шаблонів та моделям даних, доступна через адміністративний застосунок.

Різні компоненти фреймворку між собою пов'язані слабо, тому достатньо будь-яку частину замінити на аналогічну. Наприклад, замість вбудованих шаблонів можна використовувати Mako або Jinja.

1.4.7 HTML та XHTML

Використано для отримання веб-інтерфейсу що виводить аналізовані данні.

HTML — Мова розмітки гіпертекстових документів —стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документу та ідеологію структурної розмітки тексту.

Попри те, що HTML — штучна комп'ютерна мова, вона не є мовою програмування.

HTML разом із каскадними таблицями стилів та вбудованими скриптами — це три основні технології побудови веб-сторінок.

HTML впроваджує засоби для:

- створення структурованого документу шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

XHTML (Extensible Hypertext Markup Language) — мова розмітки, що має таку саму виразну силу як і HTML але відповідає синтаксичним правилам XML.

В той час як HTML побудовано на основі правил SGML, XHTML побудовано на основі правил XML, суворішої підмножини правил SGML. Оскільки XHTML документи мають бути коректними XML документами, їх обробку можна здійснювати стандартними інструментами обробки XML

документів на відміну від HTML, який вимагає порівняно складніших, важчих і повільніших синтаксичних аналізаторів. XHTML можна розглядати як, багато в чому, перетин HTML і XML, оскільки цей стандарт є переформулюванням HTML засобами XML. XHTML 1.0 став рекомендацією консорціуму W3C 26 січня 2000. XHTML 1.1 став рекомендацією W3C 31 травня 2001.

XHTML 1.0 є «реформулюванням трьох типів документів стандарту HTML 4 засобами XML 1.0».[1] World Wide Web Consortium (W3C) також продовжує підтримку Рекомендації HTML 4.01 та активну роботу над специфікаціями стандартів HTML5 і XHTML5. В поточному документі Рекомендацій XHTML 1.0, який було опубліковано та переглянуто до серпня 2002 року, W3C зазначив, що, "Сімейство XHTML є наступним кроком в еволюції Інтернету. Шляхом переходу на XHTML сьогодні, розробники контенту можуть увійти в світ XML з усіма супутніми перевагами, залишаючись впевненими в зворотній та майбутній сумісності їхнього контенту.

Проте в 2004 році, незалежно від W3C було створено Робочу групу з технологій застосування гіпертексту у Вебі (WHATWG), для роботи по вдосконаленню звичайного HTML не заснованого на XHTML. Більшість великих виробників браузерів не бажали реалізовувати функції з нових проектів стандартів W3C XHTML оскільки вважали, що вони не відповідають сучасним потребам розвитку Інтернету, а W3C захопився формалізмом XML і не реагує на реальні вимоги виробників. Apple, Mozilla та Opera сформували робочу групу WHATWG, яка почала працювати над стандартом HTML5, який допускав, але не вимагав застосування XML. У 2007 році, Робоча група W3C HTML проголосувала за офіційне визнання HTML5 і роботу над ним як наступне покоління стандарту HTML.[3] У 2009 році консорціум W3C дозволив добігти до кінця терміну дії Статуту Робочої групи XHTML 2, визнавши, що HTML 5 буде єдиним наступним поколінням стандарту HTML, як з XML, так і не-XML серіалізацію.

XHTML був розроблений з метою зробити HTML більш розширюваним і

підвищити сумісність з іншими форматами даних. HTML 4 побудований на основі та є застосуванням стандартної узагальненої мови розмітки (SGML), однак специфікація SGML складна, і як веб-браузери, так і Рекомендація HTML 4 не були повністю сумісними з нею. Стандарт XML, затверджений в 1998 році, пропонував простіший формат даних, ближче за духом до HTML 4.[7] Існували сподівання, що за допомогою переходу на формат XML, HTML стане сумісним із загальними інструментами XML; а проксі сервери зможуть перетворювати документи, у разі необхідності, для пристроїв з обмеженими можливостями, таких як мобільні телефони. Завдяки використанню просторів імен, XHTML документи могли б включати фрагменти інших, основаних на XML, мов, таких як Scalable Vector Graphics і MathML. Нарешті, відновлення роботи дала б можливість розділити HTML на компоненти для повторного використання (XHTML Модулі) і очистити неохайні частини мови.

1.4.8 CSS

Каскадні таблиці стилів (англ. Cascading Style Sheets або скорочено CSS) — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці

веб-сторінок. Головна перевага блочної верстки — розділення змісту сторінки (даних) та їхньої візуальної презентації.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.)

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS. Стилі для відображення сторінки можуть бути:

Стилі автора (інформація надана автором сторінки):

- зовнішні таблиці стилів (англ. stylesheet), найчастіше окремий файл або файли .css;
- внутрішні таблиці стилів, включені як частина документу або блоку;
- стилі для окремого елементу.

Стилі користувача

локальний .css-файл, вказаний користувачем для використання на сторінках і вказаний в налаштуваннях браузера (наприклад Opera)

Стилі переглядача (браузера)

стандартний стиль переглядача, наприклад стандартні стилі для елементів,

визначені браузером, використовуються коли немає інформації про стиль елемента або вона неповна.

Стандарт CSS визначає порядок та діапазон застосування стилів, те, в якій послідовності і для яких елементів застосовуються стилі. Таким чином, використовується принцип каскадності, коли для елементів вказується лише та інформація про стилі, що змінилася або не визначена загальнішими стилями.

Переваги

Інформація про стиль для усього сайту або його частин може міститися в одному .css-файлі, що дозволяє швидко робити зміни в дизайні та презентації сторінок;

Різна інформація про стилі для різних типів користувачів: наприклад великий розмір шрифту для користувачів з послабленим зором, стилі для виводу сторінки на принтер, стиль для мобільних пристроїв;

Сторінки зменшуються в об'ємі та стають більш структурованими, оскільки інформація про стилі відділена від тексту та має певні правила застосування і сторінка побудована з урахуванням їх;

Прискорення завантаження сторінок і зменшення обсягів інформації, що передається, навантаження на сервер та канал передачі. Досягається за рахунок того, що сучасні браузери здатні кешувати (запам'ятовувати) інформацію про стилі і використовувати для всіх сторінок, а не завантажувати для кожної.

1.4.9 JavaScript та JQuery

JavaScript – прототипна-орієнтована сценарна мова програмування. Є реалізацією мови ECMAScript (стандарт ECMA-262).

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипна парадигма програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java, але при цьому легким для використання непрограмістів. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від ряду мов програмування, використовуваних у веб-розробці.

Назва «JavaScript» є зареєстрованим товарним знаком компанії Oracle Corporation. Назва "ECMAScript" не є торговим знаком будь-яких компаній.

JavaScript є об'єктно-орієнтованою мовою, але використовне в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними клас-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання - що додає мові додаткову гнучкість.

Незважаючи на схожий з Сі синтаксис, JavaScript у порівнянні з мовою Сі має корінні відмінності:

- об'єкти, з можливістю інтроспекції;
- функції як об'єкти першого класу;
- автоматичне приведення типів;
- автоматична Збірка сміття;
- анонімні функції.

У мові відсутні такі корисні речі , як:

- модульна система: JavaScript не надає можливості управляти залежностями і ізоляцією областей видимості;
- стандартна бібліотека: зокрема, відсутній інтерфейс програмування додатків по роботі з файловою системою, управлінню потоками

введення-виведення, базових типів для бінарних даних;

- стандартні інтерфейси до веб-серверів і баз даних;
- система управління пакетами, яка б відстежувала залежності і автоматично встановлювала їх

jQuery — популярна JavaScript-бібліотека з відкритим сирцевим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресіґом (John Resig). Згідно з дослідженнями організації W3Techs, jQuery використовується понад половиною від мільйона найвідвідуваніших сайтів.[2] jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день.

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії).

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

Основне завдання jQuery — це надавати розробнику легкий та гнучкий інструментарій кросбраузерної адресації DOM об'єктів за допомогою CSS та XPath селекторів. Також даний фреймворк надає інтерфейси для Ajax-застосунків, обробників подій і простої анімації.

Принцип роботи jQuery полягає в використанні класу (функції), який при звертанні до нього повертає сам себе. Таким чином, це дозволяє будувати послідовний ланцюг методів.

1.4.10 Hadoop Ecosystem Distributives

В одній з статей були розглянуті переваги та недоліки основних поставників Hadoop Ecosystem

Для всіх тих, хто хоче використовувати потенціал великих даних, Hadoop є основною платформою для вибору. Це програмне забезпечення з відкритим доступом що дозволяє обробляти джерело величезних наборів даних, розподіляючи їх по серверах. Таким чином, він усуває залежність від високого рівня апаратних вимог і робить весь процес економічно вигідним для бізнесу в реалізації. Всі промислові системи обробки великих данних сьогодні використовувати Apache Hadoop в тій чи іншій мірі. Для спрощення роботи з Hadoop з'явилися підприємницькі версії, такі як Cloudera, MapR і Hortonworks.

У своєму первісному варіанті, Hadoop була розроблена як проста інфраструктура зберігання однократного запису. Але через роки вона перетворилася, розширивши рамки простої потужності веб-індексації. Грунтуючись на моделі MapReduce Google, Hadoop призначений для зберігання і обробки великих обсягів і різноманітності даних, які можуть знаходитись в декількох серверах.

У той час як розподілена файлова система Hadoop (в HDFS) допомагає розділити всі вхідні дані і зберігати їх на кількох вузлах, компонент MapReduce полегшує одночасну обробку даних по декільком вузлам.

Hadoop ні в якому разі не рішення з коробки. Для того, щоб побудувати дійсно інформаційно орієнтовані продукти, де рішення на базуються на основі даних, а не емпіричних робіт, компанії потрібно рішення для управління даними, яке не тільки пропонує надійне управління даними, але також легко кероване і легко інтегрується з існуючою інфраструктурою підприємства.

Гнучка модульна архітектура hadoop дозволяє додавати нові функціональні можливості для виконання різноманітних завдань з обробки великих даних. Кілька постачальників скористалися Hadoop та оптимізували свої коди, щоб змінити або розширити функціональні можливості. У процесі

вони змогли виправити деякі з недоліків, властивих Apache Hadoop. Три компанії, які дійсно виділяються в роботі: Cloudera, MapR і Hortonworks.

Порівнюючи три топ поставники Hadoop: Cloudera проти Hortonworks проти MapR

Cloudera був протягом самого довгого часу з моменту створення Hadoop. Hortonworks утворився пізніше. У той час як Cloudera і Hortonworks 100 відсотків з відкритим вихідним кодом, більшість версій MapR оснащені фірмовими модулями. Кожен постачальник має свою унікальну сильні та слабкі сторони, у кожного є певні пересічні функції, а також.

Cloudera

Cloudera Inc. була заснована розробниками Big data з Facebook, Google, Oracle і Yahoo в 2008 році була першою компанією для розробки та розповсюдження програмного забезпечення Apache Hadoop, і досі має велику базу користувачів з найбільшою кількістю клієнтів. Хоча ядро розподілу базується на основі Apache Hadoop, він також посточає фірмова Cloudera Management Suite для автоматизації процесу установки та надавання інших послуг для підвищення зручності користувачів, які включають скорочення часу розгортання, показуючи кількість вузлів в режимі реального часу », і т.д.

Hortonworks

Hortonworks, заснована в 2011 році, швидко перетворився в один з провідних постачальників Hadoop. Поставник забезпечує платформу з відкритим вихідним кодом на основі Apache Hadoop для аналізу, зберігання і управління великими даними. Hortonworks є єдиним комерційним постачальником, що поширює продукт з повним відкритим вихідним кодом Apache Hadoop без додаткового пропрієтарного програмного забезпечення. Дистрибутив HDP2.0 Hortonworks можна безпосередньо завантажити з веб-сайту, безкоштовно і легко встановити. Інженери Hortonworks знаходяться позаду більшості останніх нововведень Hadoop, в тому числі Yarn, який краще,

ніж MapReduce, позаду, в тому сенсі, що це дозволить включити більше каркасів обробки даних.

MapR

У стандартній версії з відкритим вихідним кодом програмне забезпечення Apache Hadoop приходить з низкою обмежень. Дистрибутиви, спрямовані на подолання проблем, що користувачі, як правило, стикаються в стандартних виданнях. Під вільною ліцензією Apache, всі три дистрибутиви надають користувачам версії з оновленнями, основного програмного забезпечення Hadoop.

Всі три топ-дистрибутивів Hadoop, Cloudera, MapR і Hortonworks пропонують консультації, навчання і технічну допомогу. Але на відміну від своїх двох суперників, Hortonworks стверджує, що на 100 відсотків з відкритим вихідним кодом. Cloudera включає масив патентованих елементів в його версії Enterprise 4.0, додаючи шари адміністративних та управлінських можливостей в основне програмне забезпечення Hadoop.

Йдучи далі, MapR замінює компонент HDFS і замість цього використовує свою власну фірмову файлову систему, звану MapRFS. MapRFS надає більш ефективне управління даними, надійність,і найголовніше, простота використання. В інших словах, більш готовий до виробництва, ніж його два інших конкуренти.

Через недавнє партнерство з Canonical(творець операційної системи Ubuntu), MapR пропонує Hadoop як компонент операційної системи Ubuntu за замовчуванням. Відповідно до умов партнерства, MapR в M3 видання для Apache Hadoop будуть інтегровані в операційну систему Ubuntu.

До M3 видання, MapR безкоштовно, але безкоштовна версія не вистачає деяких з її власних особливостей, а саме, JobTracker HA, NameNode HA, NFS-HA, Mirroring, Snapshot і декілька інших.

	Hortonworks	Cloudera	MapR
Performance and Scalability			
Data Ingest	Batch	Batch	Batch and streaming writes
Metadata Architecture	Centralized	Centralized	Distributed
HBase Performance	Latency spikes	Latency spikes	Consistent low latency
NoSQL Applications	Mainly batch applications	Mainly batch applications	Batch and online/real-time applications
Dependability			
High Availability	Single failure recovery	Single failure recovery	Self healing across multiple failures
MapReduce HA	Restart jobs	Restart jobs	Continuous without restart
Upgrading	Planned downtime	Rolling upgrades	Rolling upgrades
Replication	Data	Data	Data + metadata
Snapshots	Consistent only for closed files	Consistent only for closed files	Point-in-time consistency for all files and tables
Disaster Recovery	No	File copy scheduling (BDR)	Mirroring
Manageability			
Management Tools	Ambari	Cloudera Manager	MapR Control System
Volume Support	No	No	Yes
Heat map, Alarms, Alerts	Yes	Yes	Yes
Integration with REST API	Yes	Yes	Yes
Data and Job Placement Control	No	No	Yes
Data Access			
File System Access	HDFS, read-only NFS	HDFS, read-only NFS	HDFS, read/write NFS (POSIX)
File I/O	Append only	Append only	Read/write
Security: ACLs	Yes	Yes	Yes
Wire-level Authentication	Kerberos	Kerberos	Kerberos, Native

Рисунок 6 Порівняння дистрибутивів

В моєму випадку було обрано MapR оскільки він являється найпродуктивнішим серед дистрибутивів

1.5 Висновки

В цьому розділі було проведено опис доступних методів та програмних сервісів що використовувались в роботі. Також був проведений аналіз вхідних параметрів та проблематики для того що б правильно підібрати методи та сервіси для реалізації.

2. ОПИС ПРОГРАМНОГО ПРОДУКТУ. АНАЛІЗ АРХІТЕКТУРИ. СИСТЕМНІ ВИМОГИ. ДОКУМЕНТАЦІЯ. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

2.1 Загальний опис та аналіз архітектури.

Лямбда архітектура - архітектура обробки даних призначена для обробки величезної кількості даних, скориставшись обома batch і stream процесінг методами. Такий підхід до архітектури намагається збалансувати затримки, пропускної здатності та відмовостійкості з допомогою пакетної обробки, щоб забезпечити всебічну та точну обробку пакетних даних, в той час, одночасно з використанням обробки потоку в режимі реального часу, щоб забезпечити оперативну обробку на online даних. Два оброблені виходи можуть бути з'єднані перед представленням результатів. Підйом лямбда архітектури корелює з ростом великих даних, аналітики в режимі реального часу, і пом'якшенням затримки з Map-Reduce.

Лямбда архітектура залежить від моделі даних з тільки конкатеовним постійним джерелом данних, який служить в якості системи запису. Він призначений для поглинання і обробки повторних подій, які будуть додані до існуючих подій, а не перезаписуючи їх(наприклад надійшли нові данні а старі не затираються). Стан визначається з природно-погодинного впорядкування даних.

Лямбда архітектура описує систему, що складається з трьох шарів: пакетної обробки, обробки в реальному часі, і обслуговуючого прошарку для реагування на запити. Обробляючі частини отримують з незмінної майстер-копії всієї множини даних.

Пакетна обробка

Пакетна обробка попередньо обчислює результати, використовуючи розподілену систему обробки, яка може обробляти дуже великі обсяги даних. Пакетна обробка спрямована на ідеальну точність, будучи в змозі обробляти всі

доступні дані при генерації обробки. Це означає, що можна виправити будь-які помилки, повторно на основі повного набору даних запустивши цей процесс. Вихід, як правило, зберігається в базі даних,тільки для читання, для того щоб з поновленням повністю замінити існуючі передвичесленн

Apache Hadoop є стандартною системою пакетної обробки, та де-факто використовується в більшості високою пропускних архітектур.

Обробка в реальному часі

Обробка в реальному часі обробляє потоки даних в реальному часі і без вимог виправи або повноти. Цей спрямований звести до мінімуму затримки, забезпечуючи уявлення про ситуацію в реальному часі в самих останніх даних. По суті, швидкість шар відповідає за наповнення пробілів, викликаних відставанням пакетної обробки в наданні результату на основі самих останніх даних. Перегляди цей шар не може бути точним або повним, як ті, в кінцевому підсумку, вироблені пакетному шарі, але вони доступні практично відразу після прийому даних, і можуть бути замінений, коли данні з пакетного шару для тих же даних стають доступними.

Потік обробки технології, зазвичай використовуються в цьому шарі включають Apache Storm, SQLstream і Apache Spark. Вихід, як правило, зберігаються на швидких баз даних NoSQL.

Обслуговуючий прошарок

Вихід з пакетної обробки і обробки в реальному часі зберігаються в обслуговуючому прошарку, який реагує на вузькоспеціалізовані запити, повертаючи злиті результати від обох данних.

Приклади технологій, що використовуються в прямому шару включають Druid, який забезпечує єдиний кластер для обробки виведення з обох шарів, або більш прості системи коли данні що вимагають злиття більш прості Виділені бази данних, використовувані в прямому шару включають Apache Cassandra або

Apache HBase для виведення обробки в реальному часі, і Elephant DB або Cloudera Impala для виведення пакетного шару.

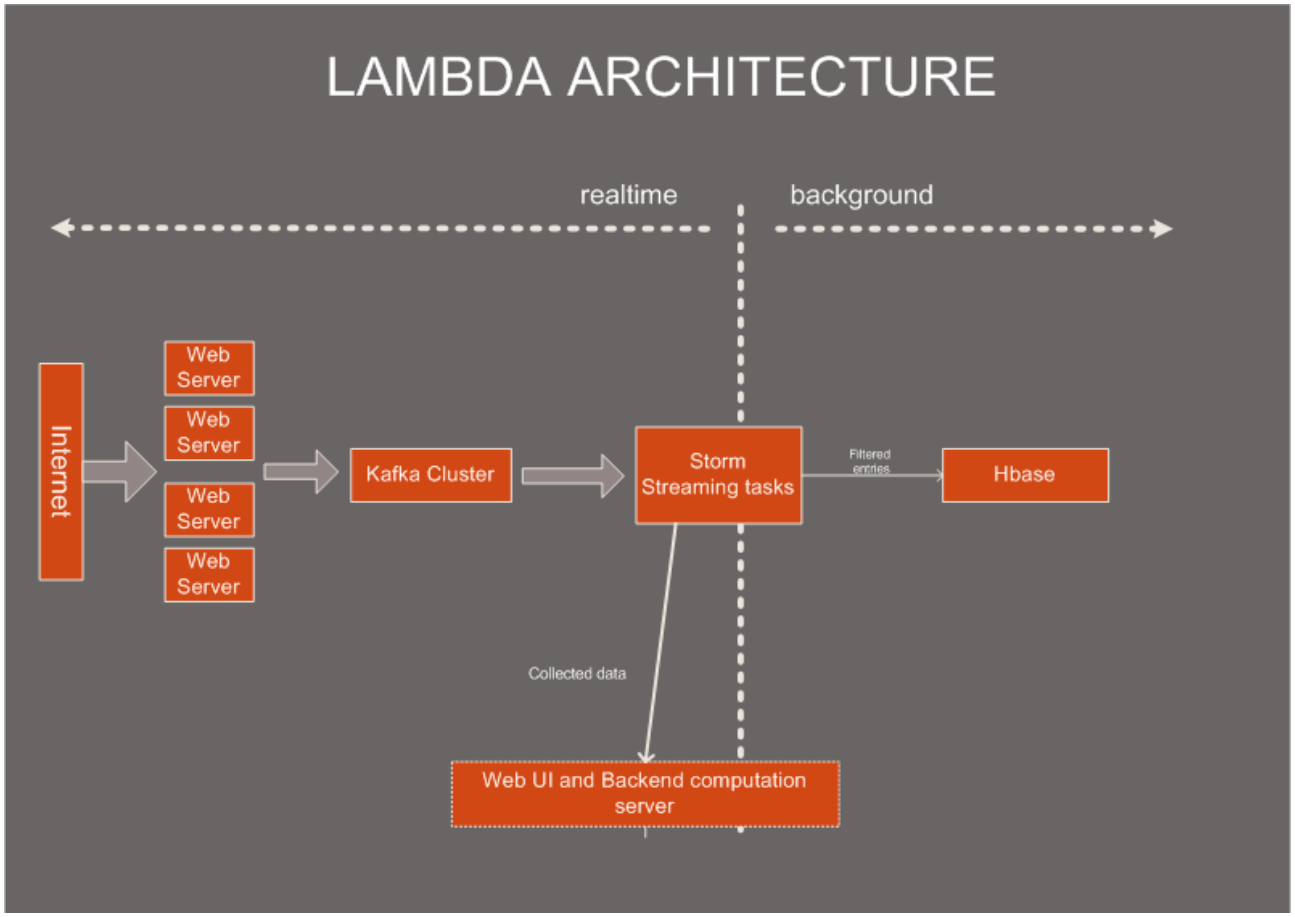

Рисунок 7 Лямда архітектура розробленої системи

В моєму випадку лямбда архітектура виглядає таким чином :

- Спочатку данні(url з данними що приходить з інтерент переглядів) потрапляють на Kafka cluster, що забезпечує стійкість системи і не дає данним бути втраченими
- Потім данні потрапляють на Storm кластер, який в свою чергу проводить обчислення класифікації первинних даних. Та передачі процессу обчислення пакетних данних. Та запис в hbase/
- Далі данні потрапляють на вебсервер який збирає запроцесені данні до купи та проводить кінцевий аналіз з виводом найпродуктивнішого товару.

2.2 Системні вимоги для інсталяції

Вимоги до операційної системи:

Основою для системи обробки був взятий MapR кластер оскільки в ньому файлова система показує найкращі результати серед усіх дистрибутивів Hadoop ecosystem. Також були використані MapR-DB таблиці, що являються покращеним варіантом Hbase таблиць і дають більшу швидкодію. MapR дистрибутив підтримує тільки Linux подібні системи

Таблиця 2.1 Мінімальні вимоги машини в кластері Storm

Unit	Type
CPU	64-bit
OS	Red Hat, CentOS, SUSE, or Ubuntu
Memory	4 GB minimum, more in production
Disk	Raw, unformatted drives and partitions
DNS	Hostname, reaches all other nodes
Users	Common users across all nodes; passwordless ssh (optional)
Java	Must run Java
Other	NTP, Syslog, PAM

Мінімальні вимоги для веб-сервера

Оскільки веб-сервер працює на Django то він повинен запускатись на Linux дистрибутиві, також там працює обчислювальна система, що теж накладує свої обмеження, зазвичай там будуть відбуватись процеси з’єднання з сервером та контроль над самою системою, в подальших версіях планується розробити підтримку так званого High Avalability. High Avalability – це підтримка для стабільності системи, коли падає сервер, то сама система не впала, а спрацював інший елемент

Таблиця 2.2 Мінімальні вимоги машини з вебсервером

Unit	Type
CPU	64-bit
OS	Red Hat, CentOS, SUSE, or Ubuntu
Memory	8 GB minimum, more in production
Disk	Raw, unformatted drives and partitions
DNS	Hostname, reaches Nimbus node
Users	Common users across all nodes; passwordless ssh (optional)
Python	Python 3.4 Django 1.8
Other	NTP, Syslog, PAM

В результаті нам потрібна система , що базується на MapR кластері, Storm частину нйкраще відділити від пакетної обробки та Hbase частини. Оскільки при запису нагрузка йде не тільки на Storm але й на Hbase.

2.3 Планування кластера та аналіз вхідних данних

Оцінимо об’єм вхідних данних. Оскільки в середньому в штатах активність проявляють 3 млн користувачів на сайті eBay.

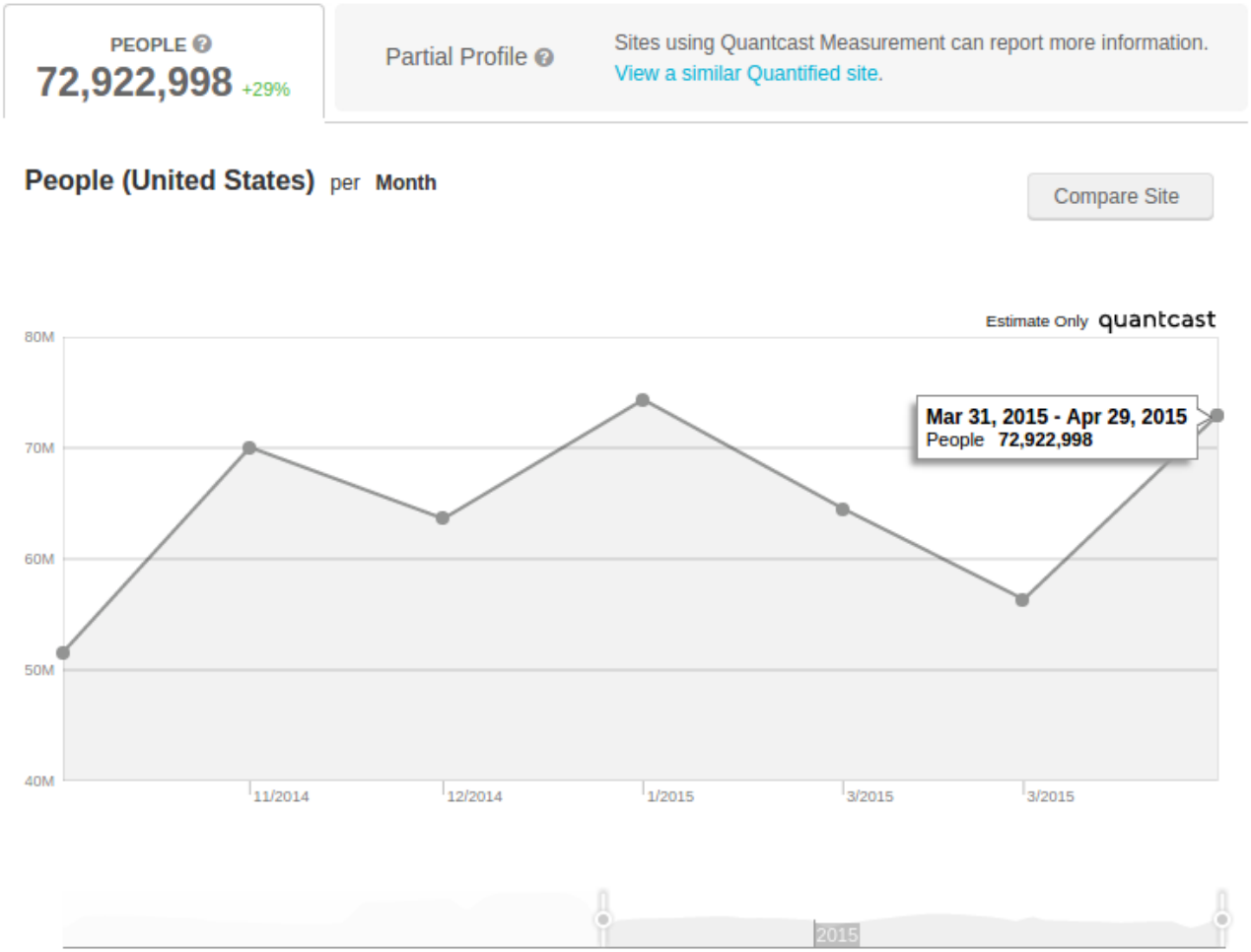

Рисунок 8 Активність користувачів в США на eBay

Та в світі об’єм продаж саме в штатах є 48%(дані показані на графіку нижче)

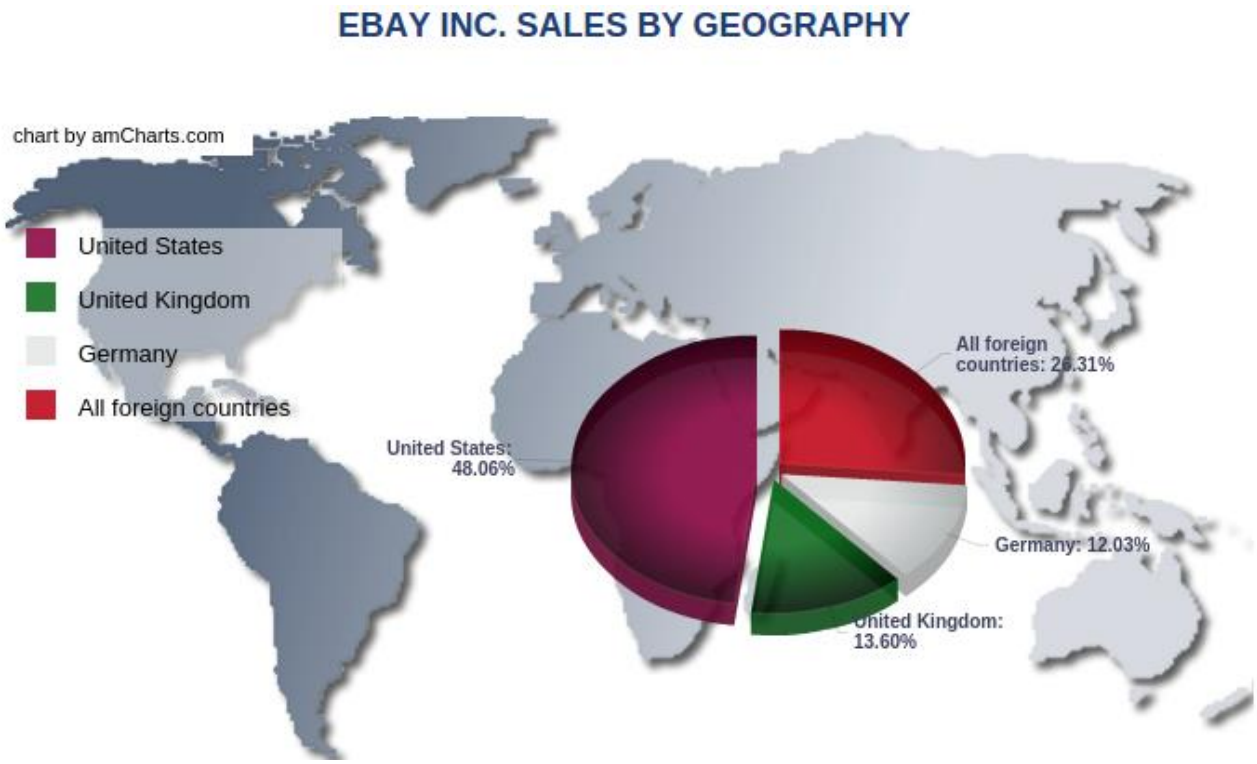

Рисунок 9 Активність покупок в світі

Тобто в середньому з данних по штатах, ми можемо оцінити кількість користувачів ebay в 7 мільйонів. Також для оцінки нам потрібно кількість запитів на товари. Дана статистика надається нижче.

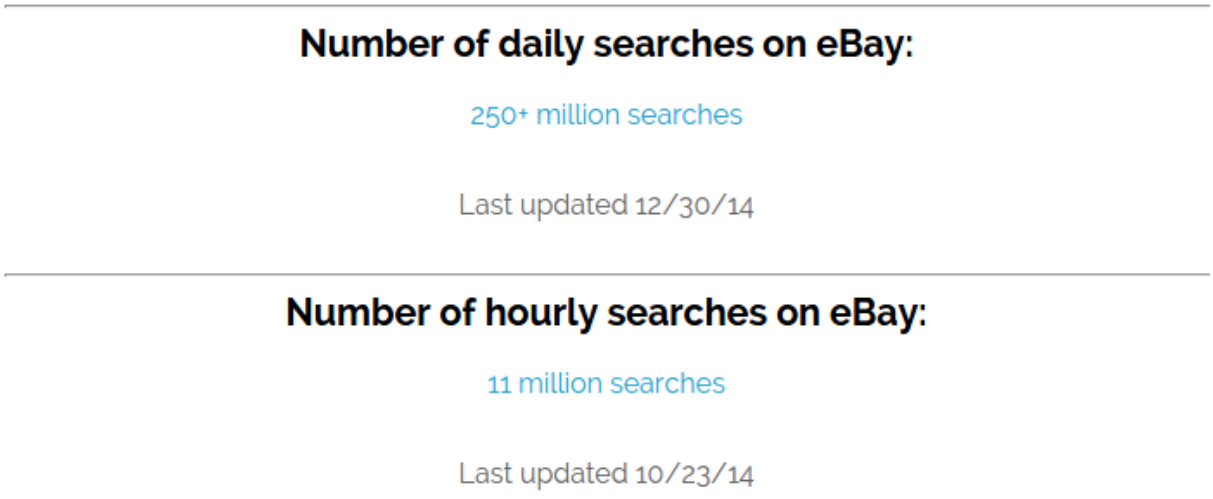

Рисунок 10 Статистичні дані по пошукам на сайті eBay

Тобто в середньому кожен користувач прогляда по 35 товарів в день.

Проте оскільки система не являється аналізатором одного eBay, то нижче наведена статистика двох внших сервісів (проте данні там не являються повними).

Amazon.com

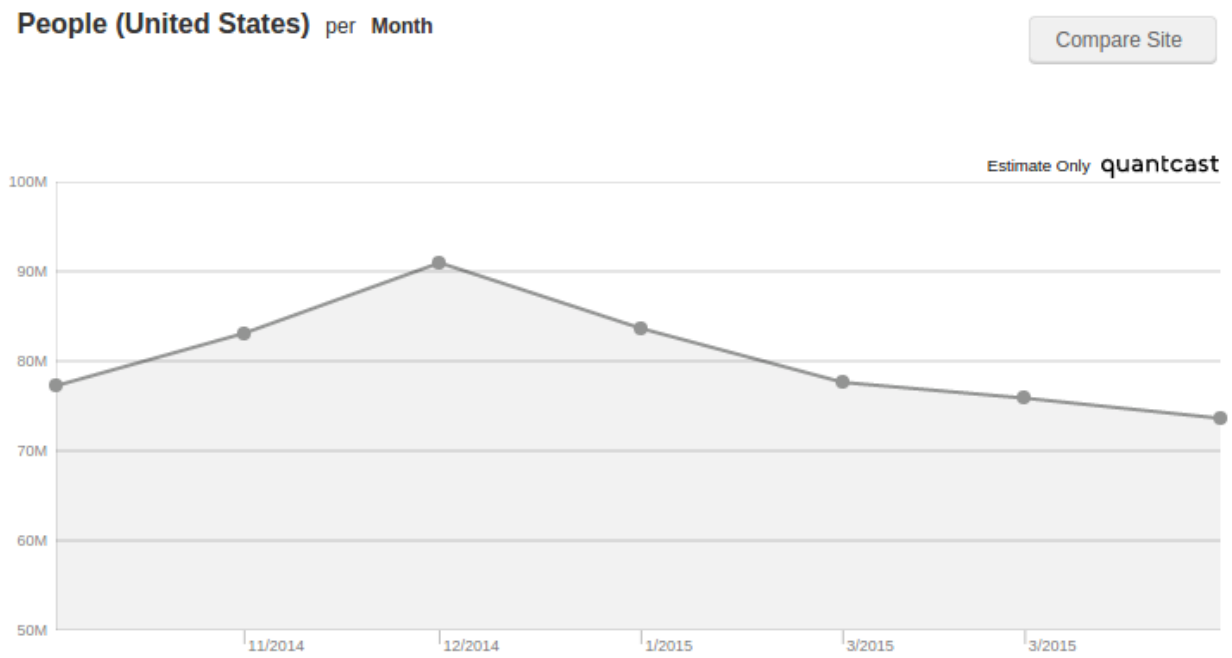

Рисунок 11 Активність користувачів в США Amazon.com

Як бачимо з наданого графіку в середньому дані наші мають досить подібну статистику до ebay і можна оцінити цю кількість рівною йому

Aliexpress.

Aliexpress являється найбільш багаточисельною серед наданих в середньому ми маємо 600 мільйонів користувачів в місяць.

З отриманих данних можемо прорахувать що в середньому у нас 3000 з ebay та amazon та 15000 з aliexpress записів в секунду. Тобто наша система має оброблять по 21 тисячу записів в секунду.

Було обраховано те що на одній машині що задовільняє мінімальниим вимогам, в середньому виконується обробка 930 записів в секунду. Нам потрібно як мінімум 23 таких сервера для того що б задовільнити обчислювальні вимоги.

Оскільки наші дані пишуться на роподілену файлову систему. То для повного зберігання данних з низькою частотою обробки нам потрібно велике сховище. При генерованих данних вийшло що в середньому в нас повинно оброблятись 400 гігабайт данних в день. Щоб мати можливість переобробляти дані раз в рік нам потрібне сховище хоча б на 150 терабайт.

Оскільки наша система підтримує Mirroring то нам потрібно хоча б вдвічі більше пам’яті.

2.4 Інсталяція програмного забезпечення

Для початку опишемо встановку MapR кластера.

Найпростішим варіантом встановки MapR кластера є встановка його задопомогою пакету MapR Installer.

Кроки при встановці MapR Installer.

- Виберіть вузол для запуску MapR Installer.

  Вузол, з якого запускається MapR Installer також має бути одним з вузлів, на які ви плануєте встановити кластер. Перш ніж почати, ви можете перевірити передумови і відомі обмеження на сайті MapR.

- Завантажити mapr-setup.sh сценарій.

  Після того як ви увійшли в систему або зареєструватися на mapr.com, ви можете завантажити mapr-setup.sh. Потім скопіюйте mapr-setup.sh на вузол, в якому буде запущений MapR Installer.

- Запустіть скрипт mapr-setup.sh щоб налаштувати вузол для запуску MapR Installer.

  Виконайте наступну команду як користувач root з каталогу, що містить сценарій:

  bash ./mapr-setup.sh

- Запустіть інсталятор MapR.

  Відкрийте URL MapR Installer: https://<MapR Installer Node hostname/Ipaddress>:9443

  Вам буде запропоновано увійти в систему як користувач Administrator MapR, який налаштований під час роботи mapr-setup.sh сценарію.

MapR Installer встановлює програмне забезпечення MapR 4.1 після проходження вас через процес вибору послуг та налаштування кластера. Після завершення установки, ви можете використовувати той же URL MapR

установника, щоб додати вузли та додаткові послуги в кластер. На цьому кроці потрібно вибрати ноду де будуть встановлені Nimbus Supervisor Ui Zookeeper та Hbase сервіси.

Опції встановки

1. Встановка вузлів в “хмарі”

При запуску установника MapR на вузлах в хмарі, зверніть увагу на наступні моменти:

- Переконайтеся, що порт 9443 відкритий.

  MapR Installer вимагає, щоб цей порт був доступний.

- URL-адреси MapR установника та обслуговування інтерфейсу повинен відноситись до зовнішнього URL, а не внутрішньою URL.

  Наприклад, коли ви відкриваєте URL MapR Installer, замініть будь-який внутрішніх хоста або IP-адресу та пов'язані з ним зовнішній адресу. Для Amazon EC2 і Google Compute Engine (GCE) кластерів, установник MapR автоматично переводить внутрішні адреси на зовнішні адреси.

- На сторінці Налаштування вузлів веб-інтерфейс установника MapR, переконайтеся, що ви виконаєте наступні дії:

  - Визначити кожного вузла з внутрішнім статичним IP-адресом або внутрішніх вирішуваних hostname.

  - Для віддаленої аутентифікації, використовувати той же ідентифікатор користувача і секретний ключ, який ви використовуєте для SSH в хмарних станах. Цей користувач повинен бути root або користувач з правами SUDO.

На підставі вимог і обмежень кластера брандмауера, вам, можливо, буде потрібно відкрити наступні типи портів для зовнішнього доступу:

- Порти, використовувані для користувача інтерфейсів, таких як MCS,

Resouce Manager UI, JobTracker UI, Storm UI, Django app ui.

- Порти, використовувані клієнтами, такими як Hive, Impala, Drill, and Spark-SQL.
- Порти, використовувані на клієнтських машинах MapR для доступу до MapR-FS або запуску Hadoop job Storm Job.

Вимоги до встановки.

Таблиця 2.3 Мінімальні вимоги для встановки кластера

Name	Version
MapR Node	Це має бути один з вузлів на якій планується встановка кластера
Package Dependencies	Відносно операційних систем він вимагає наступні пакети: на Ubuntu : python-pycurl openssh-client libssl1.0.0 sshpass Red Hat/ CentOS : python-pycurl openssh-clients openssl sshpass якщо не буде знайдено, mapr-setup.sh утиліта запропонує викачати дані пакети з інтернету.
Java	JDK 1.7 чи вище. якщо JDK 1.7 чи вище є недоступна, mapr-setup.sh встановить OpenJDK Java 1.7.
SSH Access	Повинен бути доступ (SSH access) для всіх вузлів що входять в кластер
Port Availability	Порт 9443 має бути відкритий між всіма вузлами в кластері.

Повинен бути встановлений браузер

Встановка компоненти.

- Компонента має стояти на одній з нод кластера.
- Потрібно встановити віртуальне середовище Python3.4
- в директорію з ним розпакувати архів з web-ui
- далі налаштувати setup.ini файл на роботу з середовищем
- зайти в віртуальне середовище

  source bin/activate
- з нього встановити залежності

  pip install requirements.txt
- запустити веб сервер

python manage.py collectstatic

python manage.py runserver

Далі програма готова до роботи.

2.5 Аналіз роботи програми та документація користувача

Запустивши програму ми повинні перейти з машини з доступом на сервер на адрес http://<Hostname>:8000 ми отримаєм таке вікно

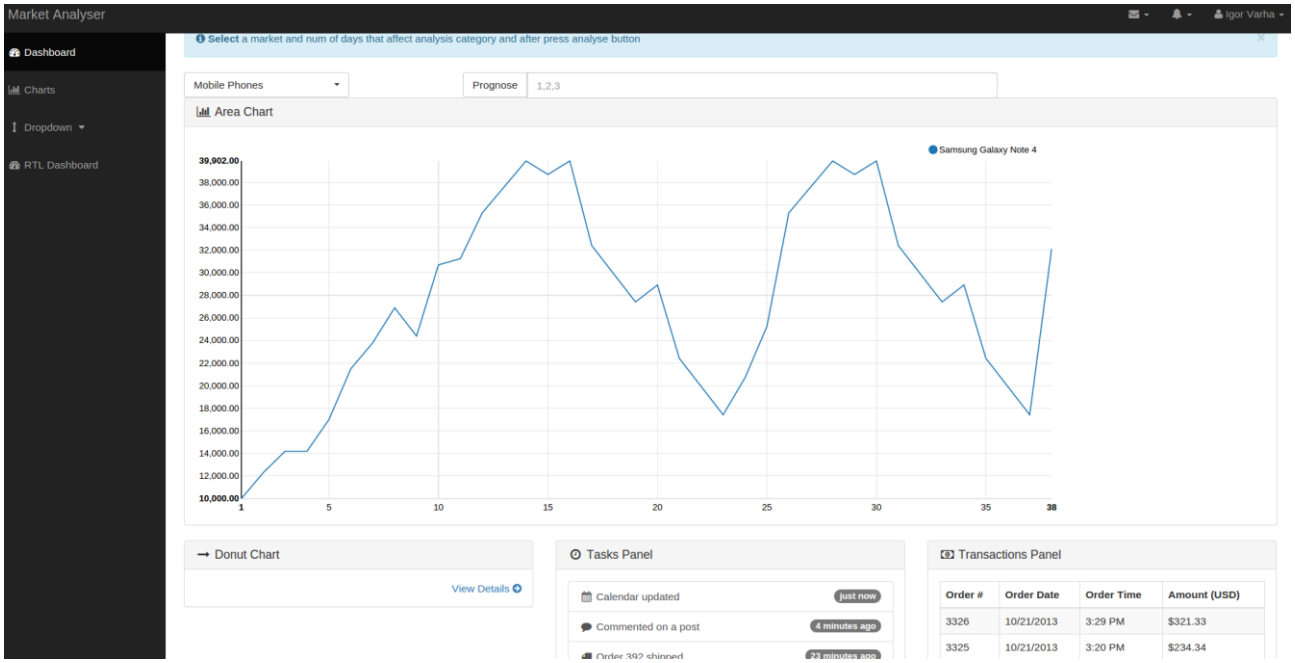

Рисунок 12 Вікно програми

Як ми бачимо в початковій версії в нас є вибір секції товару по якій ми будемо проводить аналіз та в вікно з інпутом числельних значень на скільки ми проводимо прогноз наші дані постійно оновлюються тому ми визначаєм який саме графік ми будем показувать

Далі що б запустить аналіз нам потрібно ввести кількість днів на яку буде проводитись прогноз, зацікавленості та нажати на кнопку “prognose”

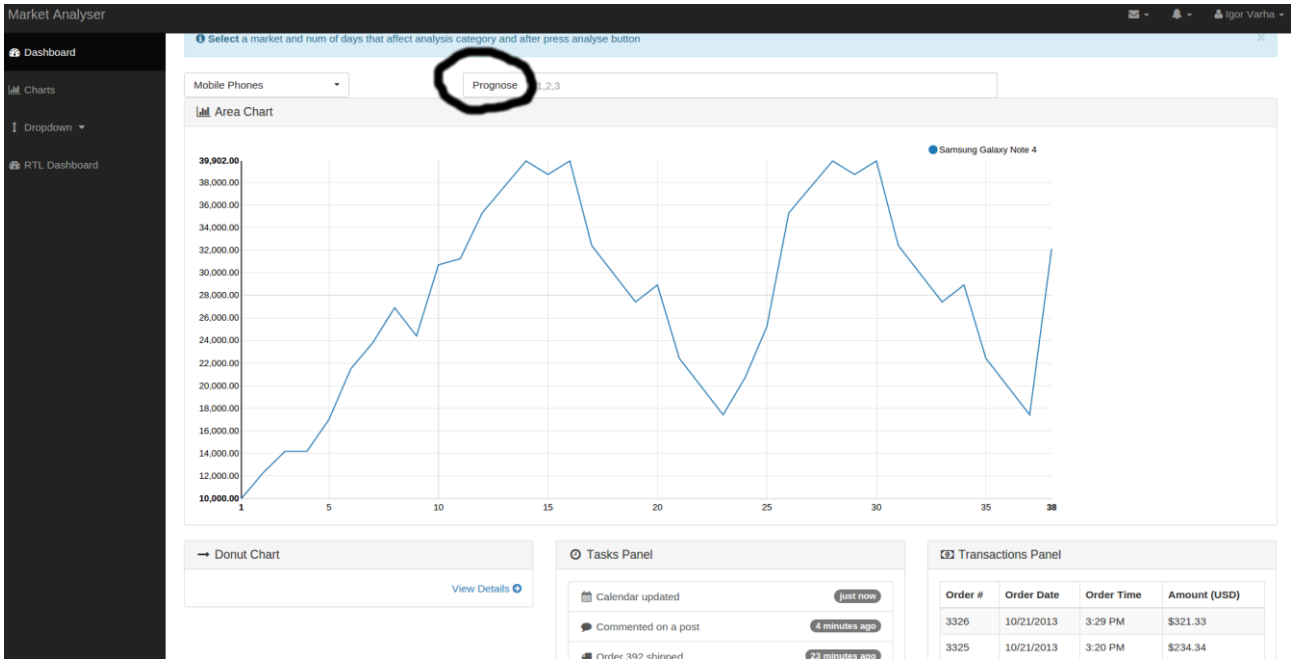

Рисунок 13 Вікно програми з виділеним елементом керування

Далі в нас виведуться данні по прогнозу:

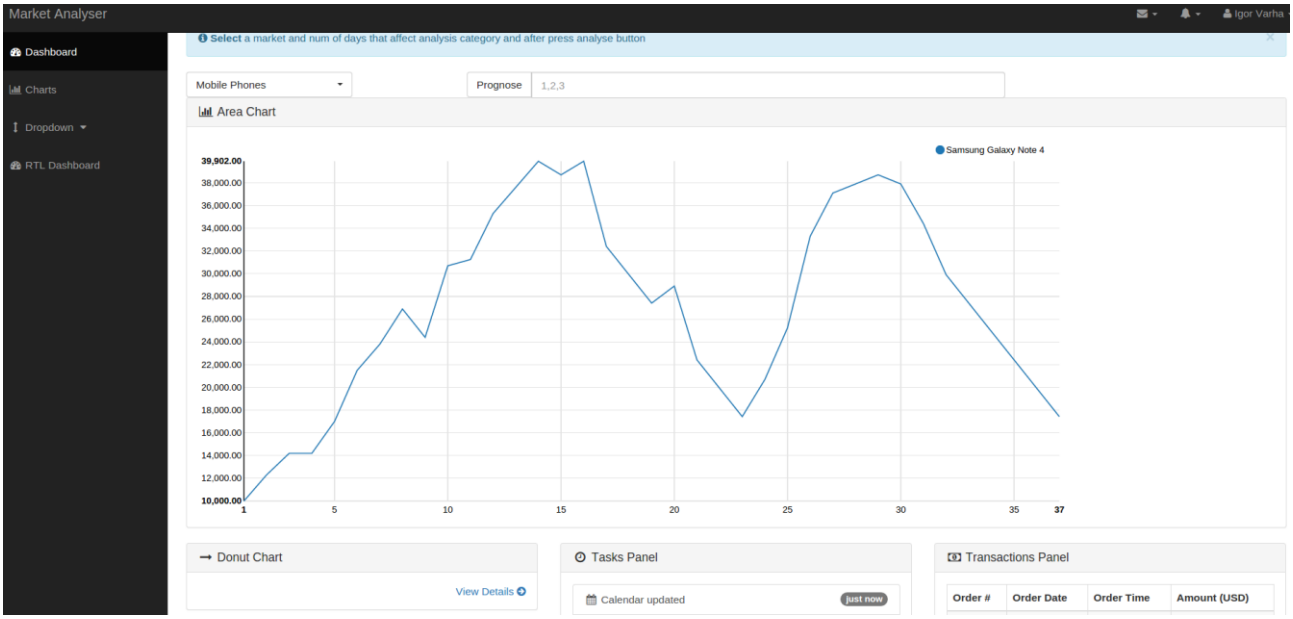

Рисунок 14 Вікно програми з виділеним елементом керування

Розглянемо їх ближче:

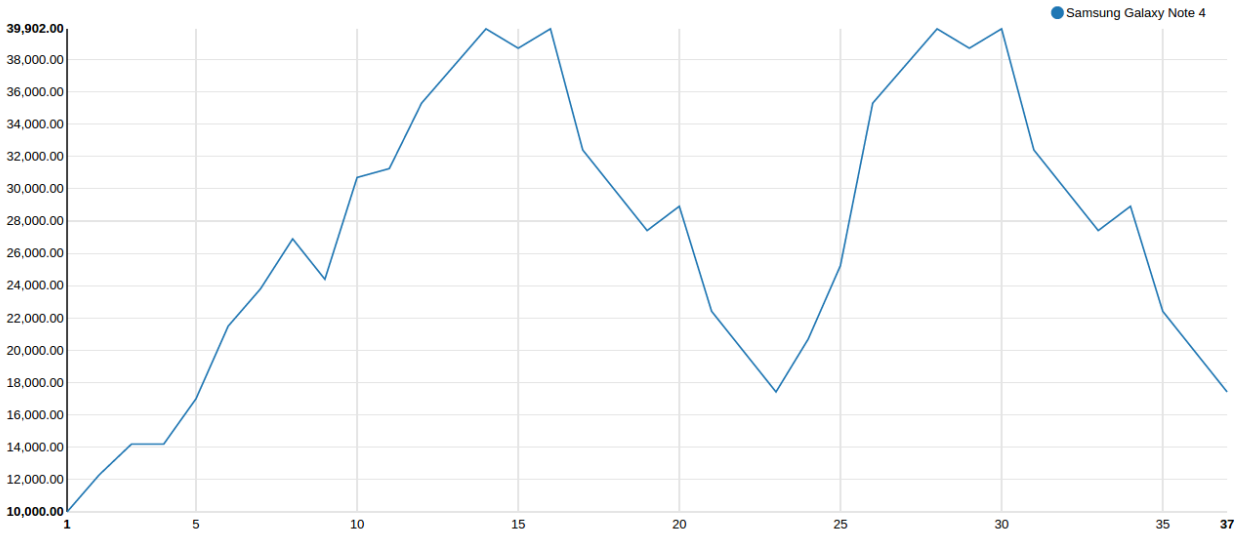

Рисунок 15 Вікно графіку непрогнозованої системи

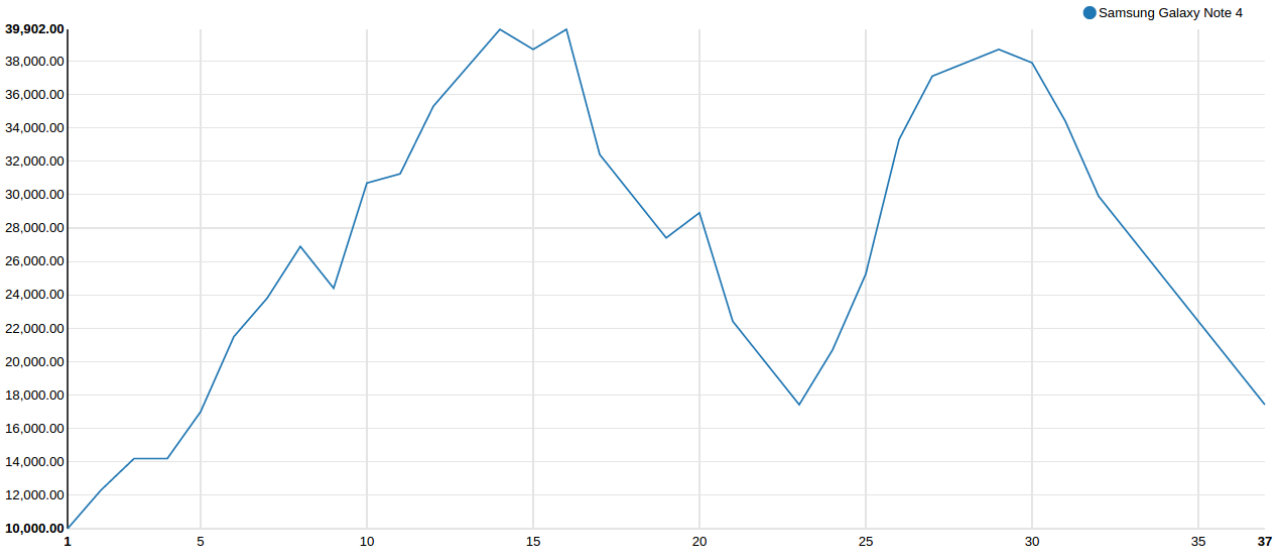

Рисунок 16 Вікно графіку тренованої на 25 періодах й прогнозовано після

Як бачимо на малюнку 14 дані в нас генеровні. Зробивши зріз і прогноз ми отримали графік на малюнку 15, що являється доволі точною моделлю за порахувавши середньоквадратичне відхилення ми отримали похибку рівну 30 відсоткам(по всіх товарах). Дані генерувались програмно з налаштуваннями, данна модель була завчасно згенерована з більшою продуктивністю, для того

що б оцінити що програма дійсно виконує свою головну ціль – прогнозування максимального прибутку

2.6 Висновки

В данному розділі було розлянуто розроблену систему, проведено аналіз результатів, показано, що данна програма виконує поставлені перед нею задачі

3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Результатом дипломної роботи є розроблена програма з визначення кредитоспроможності клієнта, безпечне використання якої вимагає якісного аналізу потенційно-небезпечних і шкідливих виробничих факторів, що створюються обладнанням в приміщенні під час його експлуатації, та заходів щодо їх усунення.

Приміщення, на якому досліджуються умови праці, є робочою кімнатою в підприємстві. В данному приміщенні проводиться аналіз ринку з подальшим прийняттям рішення про випуск товару в обіг.

3.1 Загальна характеристика приміщення і робочого місця

1. Розробка програмного забезпечення виконується в приміщенні (рис.4.1), яке знаходиться на другому поверсі 3-ох поверхового будинку з загальним та місцевим освітленням. В приміщенні одностороннє освітлення, вікна орієнтовані на схід, на вікнах є жалюзі. Стіни цегляні світлого кольору з коефіцієнтом відбиття 0,5, стеля білого кольору з коефіцієнтом відбиття 0,7. В приміщенні працює 4 людини, відповідно до цього отримаємо вхідні дані для аналізу потенційно-небезпечних і шкідливих виробничих факторів, які наведено в табл.3.1.

Таблиця 3.1 Вхідні дані

Параметри приміщення	**Значення**
Довжина х ширина х висота	6,6 х 6 х 2,7 м
Площа	39,6 м^2
Об’єм	106,92 м^3
Номер робочого місця	**Специфіка роботи**
I робоче місце	Програміст (спеціаліст з розробки десктопних програм)
II робоче місце	Інженер-алгоритміст (спеціаліст з дата майнингу та побудови алгоритмів)
III робоче місце	Аналітик
IV робоче місце	Аналітик
Технічні засоби (кількість)	**Назва та характеристики**
Монітор (4 шт.)	HP 22Xi/21,5"/1920x1080px/IPS
Комп’ютер (4 шт.)	HP Probook 4530s / 15.6" (1366x768) LED / Intel Core i5-2410M (2.30 ГГц) / RAM 8 ГБ / HDD 500 ГБ
Кондиціонер (1 шт.)	DEKKER DSH105R/G / 26м^2 / 2,65кВт-2,9кВт / 25 х 74,5 х 19,5 см / 9 кг
Підлоговий кулер (1 шт.)	CRYSTAL YLR3-5V208
Світильники загального призначення (3 шт.)	Світильник растровий вмонтований 4x18W
Світильники місцевого призначення (4 шт.)	DeLux Décor TF-05 / 1 х 40Вт

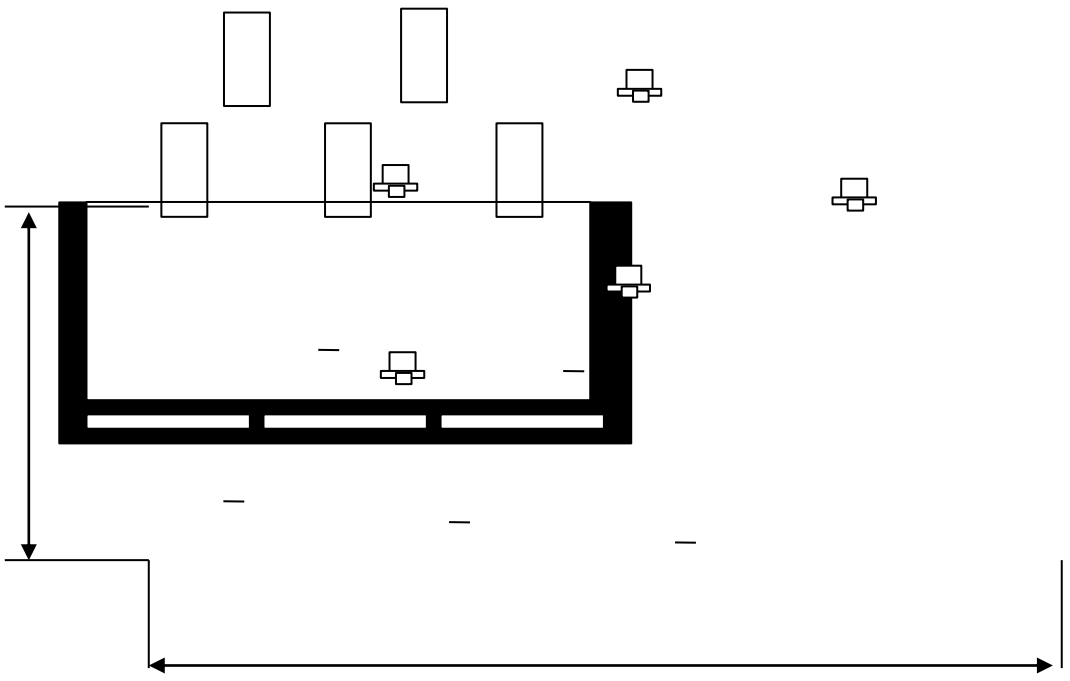

Рисунок 17 Схема приміщення

Площа S', виділена для одного робочого місця з персональною ЕОМ, повинна складати не менше 6 кв. м, а об'єм V' – не менше 20 куб. м. Розрахуємо фактичні значення цих показників, розділивши загальну площу та об'єм приміщення на кількість працюючих:

$$S' = \frac{S}{N} = \frac{6{,}6 * 6}{4} = 9{,}9 \left(\frac{\text{м}^2}{\text{люд.}}\right) (4.1)$$

$$V' = \frac{V}{N} = \frac{6{,}6 * 6 * 2{,}7}{4} = 26{,}73 \left(\frac{\text{м}^3}{\text{люд.}}\right) (4.2)$$

Отже, виходячи з отриманих результатів за характеристиками площі та об'єму приміщення відповідає нормам.

Таблиця 3.2 Характеристики робочого місця

№	Найменування параметра	Значення	
		Фактичне	**Нормативне**
1.	Висота робочої поверхні, мм	750	680 – 800
2.	Ширина робочої поверхні, мм	1400	не менше 600
3.	Глибина робочої поверхні, мм	750	не менше 600
4.	Висота простору для ніг, мм	730	не менше 600
5.	Ширина простору для ніг, мм	790	не менше 500
6.	Глибина простору для ніг, мм	700	не менше 450
7.	Висота поверхні сидіння, мм	490	400 – 500
8.	Ширина сидіння, мм	500	не менше 400
9.	Глибина сидіння, мм	470	не менше 400
10.	Висота опорної поверхні спинки, мм	650	не менше 300
11.	Ширина поверхні спинки, мм	480	не менше 380
12.	Довжина підокітників, мм	310	не менше 250
13.	Ширина підокітників, мм	55	50 – 70
14.	Відстань від очей до екрану, мм	620	600 – 700

Виходячи з заданих параметрів, можна зробити висновок, що розміри робочого місця програміста відповідають встановленим нормам.

3.2 Аналіз потенційно небезпечних і шкідливих виробничих факторів на робочому місці

Варто зазначити, що під час роботи на працівника діє ряд небезпечних і шкідливих чинників

Таблиця 3.3 Шкідливі чинники на робочому місці

Фізичні	Психофізіологічні
Підвищений рівень шуму	Розумове перенапруження
Підвищений рівень статичної електрики	Перенапруження аналізаторів
Підвищений рівень електромагнітного випромінювання	Монотонність праці
Недостатній рівень освітленості	
Неоптимальний мікроклімат	

3.2.1 Мікроклімат

Робота при створенні програмного продукту виконувалася сидячи без фізичних зусиль, а тому відноситься до категорії легка Іа. У таблиці 3.5 наведені нормативні та фактичні показники мікроклімату.

Таблиця 3.4 Аналіз шкідливих факторів, пов’язаних з мікрокліматом

№	Шкідливий фактор	Наслідки
1	Відхилення t від оптимальних параметрів	Відсутність теплового комфорту, тимчасове погіршення самопочуття і зниження працездатності, хвороби
2	Відхилення вологості повітря від оптимальних параметрів	Тимчасове погіршення самопочуття і зниження працездатності, хвороби, роздратованість
3	Відхилення V руху повітря від оптимальних параметрів	Тимчасове погіршення самопочуття і зниження працездатності, хвороби

Заходи, що забезпечують запобігання порушення встановлених мікрокліматичних норм наведені в таблиці 3.7.

Таблиця 3.5 мікроклімат в теплий період року

Параметр мікроклімату			
Найменування	Значення		
	Фактичне		Оптимальне
t, ^{o}C	21	21 – 23	18 –27
w, %	55	60 – 40	до 75
V, м/с	0,2	0,3	0,4 – 0,2

Таблиця 3.6 мікроклімат в холодний період року

Параметр мікроклімату			
Найменування	Значення		
	Фактичне		Оптимальне
t, ^{o}C	18	21 – 23	18 –27
w, %	70	60 – 40	до 75
V, м/с	0,4	0,3	0,4 – 0,2

Таблиця 3.7 Запобіжні заходи в теплий та холодний періоди року

№	**Технічні**	**Організаційні**	**ЗІЗ**
1	Контроль параметрів за допомогою термометра La Crosse WS8005; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання)	Перерви в роботі, з метою провітрювання кімнати; вологе прибирання на робочих місцях	відсутні
2	Контроль параметрів за допомогою психрометра Т-04; використання зволожувача повітря ZELMER AH1500	Перерви в роботі, з метою провітрювання кімнати; вологе прибирання на робочих місцях	відсутні
3	Контроль параметрів за допомогою анемометра Extech AN100; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання).	відсутні	відсутні

3.2.2 Недостатня освітленість і підвищена яскравість світла

Згідно ДБН В.2.5-28-2006 ця робота відноситься до розряду зорових робіт. Передбачається використання природного, штучного і змішаного освітлення. В табл. 3.8 наведені шкідливі фактори порушень норм яскравості світла.

Таблиця 3.8 Шкідливі фактори порушень норм яскравості світла

№	**Шкідливий фактор**	**Наслідки**
1	Недостатня освітленість робочої зони	Погіршення зору і самопочуття, втомлюваність, підвищення ризику здійснення помилки
2	Підвищена яскравість світла	

Таблиця 3.9 Параметри освітлення

Найменування	**Значення**		
	Фактичне	**Оптимальне**	
При змішаному освітленні		450	400
При загальному освітленні		300	300
Коефіцієнт природного освітлення		1,23	1,2

Таблиця. 3.10 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою люксметра DT-1308; використання нових світильників загального призначення ELSTEAD FINSBURY PARK FP6 POL NICKEL; урахування природного освітлення кімнати	Встановлення мінімального рівня освітлення; чищення скла вікон та світильників; заміна ламп, що перегоріли	Додаткове освітлення на робочих місцях (світильники DeLux Décor TF-05); окуляри для роботи з комп'ютером.
2	Контроль параметрів за допомогою люксметра DT-1308;використання регульованих пристроїв для відкривання вікон, а також жалюзі; використання світильників нового типу	Відсутні	Окуляри для роботи з комп'ютером.

3.2.3 Шум та вібрація

Джерелами шуму в приміщенні є вентилятор ноутбуку та кондиціонер. Звук, що створюється вентилятором та кондиціонером, можна класифікувати як постійний.

Таблиця 3.11 Шум і вібрація

Шкідливий фактор	Наслідки
Підвищений рівень шуму	Погіршення слуху, зниження продуктивності роботи, підвищення ймовірності виникнення помилки.
Вібрації на робочому місці	Роздратування, погіршення самопочуття, зниження працездатності

Таблиця 3.12 Джерела шуму

Джерело шуму	Фактичний рівень шуму, дБ	Оптимальний рівень шуму, дБ	Час роботи, год.
Кондиціонер DEKKER DSH105R/G	22	< 50	8
Кулер комп’ютеру HP Probook 4530s	20		8

Таблиця 3.13 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою приладу для виміру шуму DT-8852; якісний монтаж окремих вузлів комп'ютера	Проведення планового попереджувального ремонту (чищення від пилу і інших забруднень)	Відсутні
2	Контроль параметрів за допомогою приладу для виміру вібрацій TV260; встановлення спеціальної підставки під ноутбук	Організаційне вирішення: проведення планового попереджувального ремонту (чищення від пилу і інших забруднень)	Відсутні

3.2.4 Небезпека враження людини електричним струмом

ЕОМ є однофазним споживачем електроенергії, що живиться від змінного струму 220В від мережі із заземленою нейтраллю. IBM PC відноситься до електроустановок до 1 000В закритого виконання, всі струмопровідні частини знаходяться в кожухах. За способом захисту людини від ураження електричним струмом, ЕОМ і периферійна техніка повинні відповідати 1 класу захисту.

В табл. 3.12 наведені шкідливі фактори випромінювання при роботі з обчислювальною технікою.

Таблиця 3.14 Небезпечні фактори ураження людини електричним струмом

№	Шкідливий фактор	Наслідки	Заходи
1	Небезпечний рівень напруги струмопровідних частин обчислювальної та побутової техніки	Зростання ризику ураження електричним струмом	Релейний захист струму дотику, захисні заземлюючі корпуси . Попереджувальні знаки про рівень напруги.

Таблиця 3.15 Параметри електропостачання на робочому місці

	Напруга, В	Частота, Гц	Тип розетки/вилки	Тип фази
Фактичне	220	50	F	Однафазна, трипровідна
Оптимальне	220	50	C,F	Однофазна, трипровідна

Таблиця 3.16 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Релейний захист струму дотику, захисні заземлюючі корпуси .	Створення плану короткострокових відпочинків. Проведення робіт з електричним обладнанням лише проінструктованим персоналом	відсутні

3.2.5 Пожежна безпека

Запобігання пожежі досягається виключенням утворення горючого середовища і джерел загорянь.

Таблиця 3.17 Шкідливі фактори, пов’язані з пожежною безпекою

№	Шкідливий фактор	Наслідок
1	Коротке замикання, електротравми, пожежі, летальні наслідки	Коротке замикання, електротравми, пожежі, летальні наслідки
	Коротке замикання	Електротравми, пожежі, летальні наслідки
	Порушення протипожежного режиму	Електротравми, пожежі, летальні наслідки

В цьому приміщенні можливі пожежі таких класів: А – горіння твердих речовин , Е – горіння електроустановок під напругою. Для забезпечення цих категорій застосовуються заходи, що вказані в таблиці 4.18.

Таблиця 3.18 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою термометра La Crosse WS8005; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання)	Розвантаження електровузлів після виконання роботи; ознайомлення з інструкціями по використанню електроприладів;	Відсутні
2	Наявність вогнегасника порошкового типу ОП-5 та автоматичної системи «ГАРАНТ-Р» (ПО-2), узгоджений план евакуації	Ознайомлення з інструкціями по використанню протипожежних засобів; узгоджений план евакуації	відсутні
3	Наявність вогнегасника порошкового типу ОП-5 та автоматичної системи «ГАРАНТ-Р» (ПО-2), узгоджений план евакуації	Ознайомлення з інструкціями по використанню протипожежних засобів; узгоджений план евакуації	відсутні

3.3 Висновки

Аналіз умов праці в розглянутому робочому приміщенні показав, що умови праці з ПЕОМ відповідають нормативам.

В зв’язку з тим, що даний вид праці є шкідливим через напруженість, розробнику ПЗ рекомендується роботи перерви в роботі із виконанням невеликих гімнастичних вправ та вправ для очей.

ВИСНОВКИ

В даній роботі було розглянуто поняття обробки великий данних в режимі реального часв, проблеми, які виникають при одночасному надхдженні великого об’єму данних

Була розроблена програмна реалізація методів Візуальної побудови данних з подальшим розширенням на аналіз результатів за допомогою бібліотек обробки данних Apache Kafka, Apache Hbase, Apache Storm, на мовах програмування Java, Python, JavaScript методу відображення за допомогою пакета прикладних програм nvd3.

Після підрахунку кількості операцій виявилося, що метод розроблена система відповідає вимогам стабільності масштабовності та описана пропускна здатність й розраховані вимоги до апаратних частин які повинні для підтримки очікуваної пропускної здатності

Реалізовані алгоритми були протестовані на різних послідовностях, данних за отриманими результатами алгоритми був запропонований результать прийнаття рішення. Оскільки данні були згенеровані й можна було точно визначить й порівнть запропоноване рішення(прогноз) в данний момент з тим що дійсно відбувається далі. Тому можна зробити висновок, що система дійсно пропонує оптимальні рішення.

В подальшому слід зосередити роботу на покращенні точності прогнозованих результатів за допомогою існуючих алгоритмів аналізу данних. Отримана програмна реалізація може ібути використана в системах прогнозу прибутку при випуску товарів на онлайн ринки, в незалежності від самих джерел(ринків) й типів товарів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Олдендерфер М. С., Блэшфилд Р. К. Кластерный анализ / Факторный, дискриминантный и кластерный анализ: пер. с англ.; Под. ред. И. С. Енюкова. — М.: Финансы и статистика, 1989. — 215 с

2. Г. Пятецкий-Шапиро, Data Mining и перегрузка информацией // Вступительная статья к книге: Анализ данных и процессов / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. 3-е изд. перераб. и доп. СПб.: БХВ-Петербург, 2009. 512 с

3. Wolfram Language Documentation Center Introduction to Manipulate – Режим доступу:

https://storm.apache.org/documentation/Tutorial.html - Дата доступу – 03.06.1015.

4. Бокс Дж., Дженкинс Г. Анализ временных рядов, прогноз и управление: Пер. с англ. // Под ред. В.Ф. Писаренко. – М.: Мир, 1974, кн. 1. – 406 с.

5. Дрейпер Н., Смит Г. Прикладной регрессионный анализ. Множественная регрессия — 3-е изд. — М.: «Диалектика», 2007. — С. 912. — ISBN 0-471-17082-8.

6. Фёрстер Э., Рёнц Б. Методы корреляционного и регрессионного анализа — М.: Финансы и статистика, 1981. — 302 с.

7. Захаров С. И., Холмская А. Г. Повышение эффективности обработки сигналов вибрации и шума при испытаниях механизмов // Вестник машиностроения : журнал. — М.: Машиностроение, 2001. — № 10. — С. 31—32. — ISSN 0042-4633.

8. Радченко Станислав Григорьевич,. Устойчивые методы оценивания статистических моделей: Монография. — К.: ПП «Санспарель», 2005. — С. 504. — ISBN 966-96574-0-7, УДК: 519.237.5:515.126.2, ББК 22.172+22.152.

9. Радченко Станислав Григорьевич,. Методология регрессионного анализа: Монография. — К.: «Корнийчук», 2011. — С. 376. — ISBN 978-966-7599-72-0.

10. Preimesberger, Chris Hadoop, Yahoo, 'Big Data' Brighten BI Future (англ.). EWeek (15 August 2011). Проверено 12 ноября 2011.

11. Черняк, Леонид Большие Данные — новая теория и практика (рус.) // Открытые системы. СУБД. — М.: Открытые системы, 2011. — № 10. — ISSN 1028-7493.

12. Типові норми належності вогнегасників (затверджено наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 2 квітня 2004 р. N 151)

13. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст] / К., 2000.- 16 с.

14. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. НАПБ Б.03.002-2007. (затверджено наказом МНС України від 03.12.2007 № 833)

15. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу (затверджено наказом МОЗ України від 12.08.2014р № 248

16. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. № 7).

17. Правила охорони праці під час експлуатації електронно-обчислювальних машин. НПАОП 0.00-1.28-10 (затверджено наказом Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010р. № 65).

18. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу (затверджено наказом МОЗ України від 08.04.2014 № 248)

Зміст

ВСТУП ... 8
1. ОГЛЯД ПРОБЛЕМАТИКИ. ОПИС ТА ВИБІР ВИКОРИСТОВАНИХ СИСТЕМ ТА МАТЕМАТИЧНИХ МЕТОДІВ ... 11
1.1 Огляд проблематики ... 11
1.2 Постановка задачі ... 13
1.3 Вибір методів аналізу. ... 14
1.4 Вибір програмних сервісів ... 18
1.4.1 Apache Kafka ... 18
1.4.2 Apache Hbase ... 21
1.4.3 Apache Storm ... 23
1.4.4 Java ... 29
1.4.5 Python ... 33
1.4.6 Django ... 37
1.4.7 HTML та XHTML ... 40
1.4.8 CSS ... 42
1.4.9 JavaScript та JQuery ... 44
1.4.10 Hadoop Ecosystem Distributives ... 47
1.5 Висновки ... 51
2. ОПИС ПРОГРАМНОГО ПРОДУКТУ. АНАЛІЗ АРХІТЕКТУРИ. СИСТЕМНІ ВИМОГИ. ДОКУМЕНТАЦІЯ. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ... 52
2.1 Загальний опис та аналіз архітектури. ... 52
2.2 Системні вимоги для інсталяції ... 55
2.3 Планування кластера та аналіз вхідних данних ... 57
2.4 Інсталяція програмного забезпечення ... 61
2.5 Аналіз роботи програми та документація користувача ... 65
2.6 Висновки ... 68
3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ ... 69
3.1 Загальна характеристика приміщення і робочого місця ... 69
3.2 Аналіз потенційно небезпечних і шкідливих виробничих факторів на робочому місці 71
3.2.1 Мікроклімат ... 72
3.2.2 Недостатня освітленість і підвищена яскравість світла ... 73
3.2.3 Шум та вібрація ... 74
3.2.4 Небезпека враження людини електричним струмом ... 75
3.2.5 Пожежна безпека ... 76
3.3 Висновки ... 76
ВИСНОВКИ ... 77
ПЕРЕЛІК ПОСИЛАНЬ ... 78

ВСТУП

Оцінка сучасного стану проблеми

При прогнозуванні значну частину проблемних задач складає наявнність можливостей обробки великих об’ємів данних, що є великою проблемою для вирішення на одній машині, навіть при її відносно великих потужностяж або експериментальні задачі, розв’язок яких за допомогою. З розвитком і розповсюдженням мережі Web важливість підтрики рішень в реальному часі зростає ще більше, оскільки тільки в цих умовах можна провести вірне прогнозування складних систем . Надійність і масштабовність системи, а також простота використання, впливають на стабільність і швидкість виконання прийнятого рішення. Перевага кластерних систем обробки данних в тому, що вони частково вирішують цю проблему. Методи класифікації і візуалізації дуже гнучкі, і, будучи розширеними на прогнозування, справляются з задачею з найменшими затратами на них.

Актуальність

Сьогодні прогнозування на данних що приходять з масштабних систем стало реалістичним, воно вимагає все більшої обчислювальній потужності. Саме тому дослідники, що стоять на передових наукових позиціях, звертаються до кластерних систем для отримання максимально можливої обчислювальної потужності. Аналіз складності актуальних обчислювальних задач різних галузей науки і техніки показує, що для їх розв’язку необхідні комп’ютери з дуже високою виробничістю. Отримати яку, залишаючись в межах традиційних підходів побудови векторно-конвеєрних систем не можливо. Розв’язок існує лише на шляху широкого розпаралелювання обробки і створення відповідних обчислювальних систем. Для розв’язку поставлених проблем і проводиться розвиток користувацьких сервісів для можливості обробки данних в режимі реального часу, зокрема з використанням паралельних обчислень. Мультипроцесорні системи зараз є найпотужнішими комп'ютерами в світі. Ці машини містять в собі від декількох сотень до декількох тисяч

процесорів в одному корпусі, сполучені з сотнями гігабайтами пам'яті. Мультипроцесорні системи мають величезну обчислювальну потужність і використовуються для вирішення глобальних обчислювальних проблем, таких як глобальне моделювання клімату і розробка ліків.

Останні декілька років ми були свідками постійного розповсюдження паралельних обчислень, як для високопродуктивних наукових обчислень, так і для задач ширшого призначення. А це стало результатом вимог до високої продуктивності, низької вартості і підтримки продуктивності. Популярність використання паралельних обчислень супроводжувалась двома головними досягненнями: масивами паралельних процесорів (мультипроцесорні системи) і використанням розподілених обчислень. Дані великого об’єму являються дуже поширеними на сьогднішній день завдяки поширенності інтернет мережі, та щоденному збільшенню контенту. Всі системи по обробці даних потребують великих обчислювальних можливостей.

Задача обробки надходящих в режимі реального часу даних на сьогоднішній день є ключовою в системах з високою частотою зміни та надходження данних. У всіх цих системах використовуються сервіси для стріміногової обробки данних на кластерних системах.

В перших системах обробки великих данних, дані сприймались, як дещо одиничне статичне й поповнювались з допомогою файлових данних(як бази данних, або cloud системи збереження данних).

Далі з’явилися системи здатні обробляти дані в безперервному потоці — дані що постійно надходять на системи й вимагають збору й обробки для аналізу. Цей тип обробки має в собі, багато застосувань(стрімінг і обробка відео надходячого на ютуб, системи обробки лікарських діагнозів, соціальний аналіз).

Мета

Метою даної роботи є розробка системи що буде розрахована на збір та обробку даних з онлайн ринків(Ebay Amazon Aliexpress). Для цього визначено задачі, що розв’язуються в роботі:

– аналіз існуючих систем збору та обробки великих данних в режимі реального

часу, та вибір найбільш підходячої системи для вирішення задачі;

– підбір алгоритму для прогнозування результатів

– аналіз існуючих архітектур мультиплатформених багатокористувацьких систем і вибір відповідних до поставленої мети;

– створення програмного модулю рішення задачі збору та обробки данних, що задовільняють вимогам реального часу, аналіз часових затрат виконання різних його частин;

– створення на базі отриманих пропозицій та результатів прогностичного модуля;

– визначення ефективності отриманого рішення.

Зв'язок роботи з науковими програмами, планами, темами

Робота виконується у відповідності до планів наукових досліджень, які виконувалися і виконуються на кафедрі СП по вдосконаленню пакета схемотехнічного проектування ALLTED з точки зору подальшого вдосконалення чисельних процедур.

Методи досліджень

Для вирішення поставлених задач використовувалися методи збору та обробки великих даних на багатьох машинах. Перевірка отриманих результатів здійснювалася шляхом проведення обчислювальних експериментів на ЕОМ.

1. ОГЛЯД ПРОБЛЕМАТИКИ. ОПИС ТА ВИБІР ВИКОРИСТОВАНИХ СИСТЕМ ТА МАТЕМАТИЧНИХ МЕТОДІВ

1.1 Огляд проблематики

Процесси збору та аналізу великих данних в режимі реального набувають актуальності тоді коли дані надходять постійно, а не періодичним процессом.

Ще в 2001 році, аналітик Дуг Лані сформулював, на теперішній час, найкраще визначення великих даних, як 3 ключові фактори:

- Об’єм. Багато факторів сприяють збільшенню обсягу данних. Операційні дані, що зберігаються протягом багатьох років. Неструктуровані дані, що збираються напряму з соціальних медіа. Збільшення кількості датчиків і машина-машина збираючихся даних(логування). У минулому, надмірний обсяг даних визивав питання зберігання. Але зі зменшенням витрат на зберігання, виникають інші питання, в тому числі, як визначити актуальність у великих обсягах даних і як використовувати аналітику, щоб створити цінність з відповідних даних.
- Швидкість. Потокові дані з безпрецедентною швидкістю повинні бути оброблені вчасно. Такі системи як: RFID мітки, датчики і смарт вимірювання є рушійною необхідністю боротьби з потоками даних в режимах близьких до реального часу. Адекватна швидка реакція, щоб впоратися зі швидкістю передачі даних, є ціллю для більшості організацій.
- Різноманітність. Дані сьогодні приходять у всіх видах форматів. Структуровані, числові дані у традиційних базах даних. Інформація, створена напряму з бізнес-додатків.

Недоліками систем таких як Apache Hadoop є неможливість стрімінга процесу, тобто ми не можемо використовувати Hadoop систему для процесингу даних приходять в режимі реального часу.

Цю проблему вирішують інші системи обробки данних. Розробка на одній з подібних систем і буде основною темою огляду. Розглянемо одну з подібних проблематик.

1.2 Постановка задачі

Припустимо у нас є системи онлайн маркетів. У кожної знаменитої системи (Ebay, AliExpress, Taobao, Deal Express, Amazon) в середньому в кожен день приблизно по 10 мільйонів людей, які переглядають різні товари на кожному з цих магазинів. Що являється великим підгрунтям для аналізу ринку, адже отримавши інформацію по переходам можна оцінити зацікавленість в ринку, й спрогнозувати те, що можна виводить на ринок на данний час, які закупки виконать, щоб задовольнить потреби ринку.

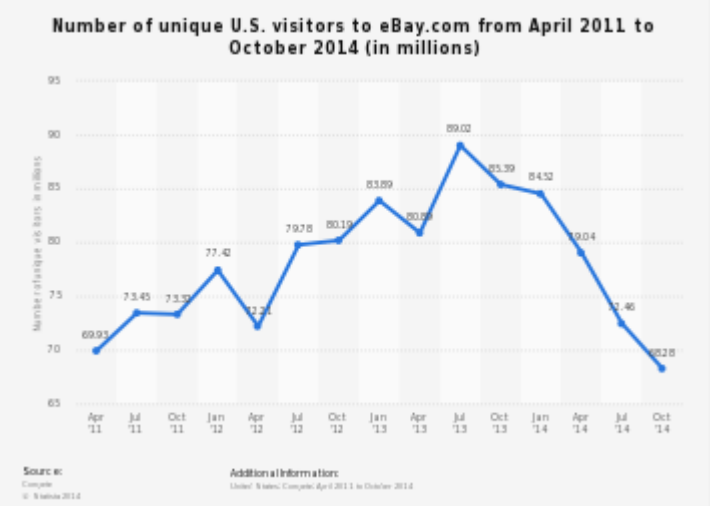

Рисунок 1 Статистика переглядів e-bay з квітня 2011 жовтень 2014

Як ми бачимо за графіком - в середньому у нас 1 мільйон унікальних користувачів в Сполучених Штатах Америки кожен день. Візьмемо за увагу те що статистика не враховує відвідування сайту поза США, відповідно кількість переглядів по світу значно збільшитися. [2]

Мета моєї роботи - зробити систему яка буде уніфікована для всіх подібних магазинів. І зможе оброблювать, аналізувати, виводити дані з докладною статистикою та підтримки прийняття рішення про виведення товару на ринок.

1.3 Вибір методів аналізу.

У вступі було коротко було перераховано методи аналізу. Розглянемо їх більш детально.

Data mining.

Data Mining (нтелектуальний аналіз даних) - загальна назва, що використовується для позначення сукупності методів виявлення в даних раніше невідомих, нетривіальних, практично корисних і доступних інтерпретації знань, необхідних для прийняття рішень у різних сферах людської діяльності. Термін введений Григорієм Пятецким-Шапіро в 1989 році .

Англійське словосполучення «Data Mining» поки не має усталеного перекладу на українську мову. При передачі українською мовою використовуються наступні словосполучення : видобуток даних, вилучення даних, а також, інтелектуальний аналіз даних . Більш повним і точним є словосполучення «виявлення знань в базах даних» (англ. Knowledge discovery in databases, KDD).[5]

Основу методів Data Mining складають всілякі методи класифікації, моделювання і прогнозування, засновані на застосуванні дерев рішень, штучних нейронних мереж, генетичних алгоритмів, еволюційного програмування, асоціативної пам'яті, нечіткої логіки. До методів Data Mining нерідко відносять статистичні методи (дескриптивний аналіз, кореляційний і регресійний аналіз, факторний аналіз, дисперсійний аналіз, компонентний аналіз, дискримінантний аналіз, аналіз часових рядів, аналіз виживаності, аналіз зв'язків). Такі методи, однак, припускають деякі апріорні уявлення про аналізованих даних, що дещо розходиться з цілями Data Mining (виявлення раніше невідомих нетривіальних і практично корисних знань).

Одне з найважливіших призначень методів Data Mining полягає в наочному поданні результатів обчислень (візуалізація), що дозволяє використовувати інструментарій Data Mining людьми, які мають спеціальної

математичної підготовки. У той же час, застосування статистичних методів аналізу даних вимагає доброго володіння теорією ймовірностей і математичної статистикою.

Задачі, які вирішуються методами Data Mining, прийнято розділяти на описові (англ. Descriptive) і передбачувальні (англ. Predictive).

В описових завданнях найголовніше - це дати наочне опис наявних прихованих закономірностей, в той час як в передбачувальних завданнях на першому плані стоїть питан ня про прогнозування для тих випадків, для яких даних ще немає.[6]

До описових завдань відносяться:

- Пошук асоціативних правил або патернів (зразків);
- Групування об'єктів, кластерний аналіз;
- Побудова регресійної моделі.

До прогнозування відносяться:

- класифікація об'єктів (для заздалегідь заданих класів);
- регресійний аналіз, аналіз часових рядів.

В нашому випадку доцільно використовувати аналіз часових рядів, оскільки класифікація й групування запитів робиться ріалтайм системою збору, й базової обробки.

Часовий ряд (англ. Time series) - реалізація випадкового процесу, набір послідовних результатів спостереження. Приклади часових рядів кількість сонячних плям, сила вітру, зміна курсу валюти. Часовий ряд дуже часто побудовані за допомогою лінійних діаграм. Тимчасові ряди використовуються в статистиці, обробки сигналів, розпізнавання образів, економетрики, прогнозування погоди, передбачення землетрусів, електроенцефалографія, контроль інженерних даних, астрономії, інженерних комунікацій, і в значній мірі застосовується в наукових дослідженнях і техніки, який включає часові

виміри.

Аналіз часових рядів включає методи аналізу часових рядів для того, щоб витягти корисну статистику та інші характеристики даних. Прогнозування часових рядів є використання моделі для прогнозування майбутніх значень на основі раніше спостережуваних значень. У той час як регресійний аналіз часто використовується таким чином, як для перевірки теорій, поточні значення одного або більше незалежних часових рядів впливають на поточне значення іншої тимчасової серії, цей тип аналізу часових рядів не називається «Аналіз часових рядів», яка фокусується на порівнянні значень одного часового ряду або декількох залежних часових рядів в різні моменти часу.

Дані часових рядів мають природний тимчасової порядок. Це робить аналіз часових рядів відміну від перехресних досліджень, в яких немає природного упорядкування спостережень (наприклад, пояснюючи заробітної плати людей з урахуванням їх відповідних рівнів освіти, де дані фізичних осіб можуть бути введені в будь-якому порядку). Аналіз часових рядів також відрізняється від аналізу просторових даних, де спостереження, як правило, відносяться до географічним положенням (наприклад, врахування цін на житло за місцем знаходження, а також внутрішні характеристики будинку). Крім того, моделі часових рядів часто роблять використання природного часу в односторонньому порядку, так що значення для даного періоду буде виражатися в отриманні в якийсь із минулих значень, а не з майбутніх значень.

Прогнозні оцінки за допомогою методів екстраполяції розраховуються в кілька етапів:

- перевірка базової лінії прогнозу;
- виявлення закономірностей минулого розвитку явища;
- оцінка ступеня достовірності виявленої закономірності розвитку явища в минулому (підбір трендової функції);
- екстраполяція - перенесення виявлених закономірностей на деякий період

майбутнього;

- коригування отриманого прогнозу з урахуванням результатів змістовного аналізу поточного стану.

Для отримання об'єктивного прогнозу розвитку досліджуваного явища дані базової лінії повинні відповідати таким вимогам:

- крок за часом для всієї базової лінії повинен бути однаковий;
- спостереження фіксуються в один і той же момент кожного часового відрізку (наприклад, на полудень кожен день, першого числа кожного місяця);
- базова лінія повинна бути повною, тобто пропуск не допускається.

Якщо при спостереженні відсутні результати за незначний відрізок часу, то для забезпечення повноти базової лінії необхідно їх заповнити приблизними даними, наприклад, використовувати середнє значення сусідніх відрізків.

Коригування отриманого прогнозу виконується для уточнення отриманих довгострокових прогнозів з урахуванням впливу сезонності або стрибкоподібність розвитку досліджуваного явища.

1.4 Вибір програмних сервісів

1.4.1 Apache Kafka

Apache Kafka - розподілене програмний брокер повідомлень, проект з відкритим вихідним кодом, розроблений в рамках Apache Software Foundation. Написаний на мові програмування Scala.

Відмінності від традиційних системи передачі повідомлень:

• спроектований спочатку як розподілена система, яку легко масштабувати

• підтримує високу пропускну здатність як з боку джерел, так і для систем-споживачів

• підтримує об'єднання споживачів в групи,

• забезпечує можливість тимчасового зберігання даних дляподальшої пакетної обробки.

Однією з особливостей реалізації інструменту є застосування техніки, подібної з журналами транзакцій, використовуваними в системах управління базами даних.

Кафка є розподілена, розділена, відновнувана система обробки повідомлень. Це забезпечує функціональність системи обміну повідомленнями.

Що все це означає?

По-перше, розглянемо деякі базові терміни повідомленнями:

Кафка веде канали повідомлень в категорії, які називаютя "теми".

Ми будемо називати процеси, які публікують повідомлення Кафки "виробники тем".

Ми будемо називати процеси, які підписуються на повідомлення і обробляють канал опублікованих повідомлень "споживачі" ..

Кафка працює як кластер, що складається з одного або декількох серверів, кожен з яких називається брокером.

Так, на високому рівні, виробники відправляють повідомлення по мережі кластера Кафки, який, у свою чергу відправляє їх до споживачів, як це показано на діаграмі:

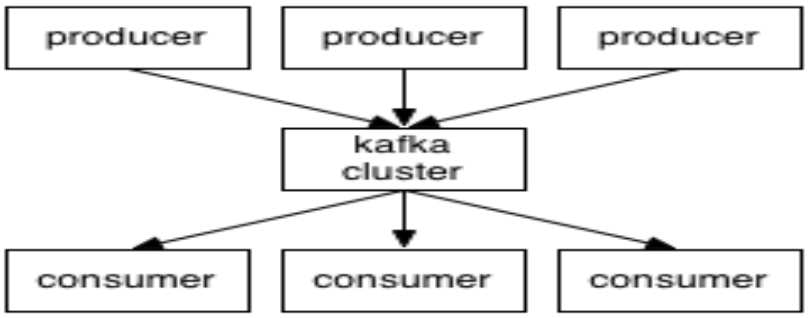

Рисунок 2 Схема Кафка кластера

Зв'язок між клієнтами і серверами здійснюється за допомогою простого, високо продуктивного протоколу TCP. Ми використовуєм клієнт Java для Кафки, але клієнти доступні багатьма мовами.

Теми і Журнали

Давайте спочатку зануримося в абстракцію високого рівня яку Кафка забезпечує - тему.

Тема є першою категорією якої повідомлення будуть опубліковані. По кожній темі, кластер Кафка веде многороздільний журнал, який виглядає наступним чином:

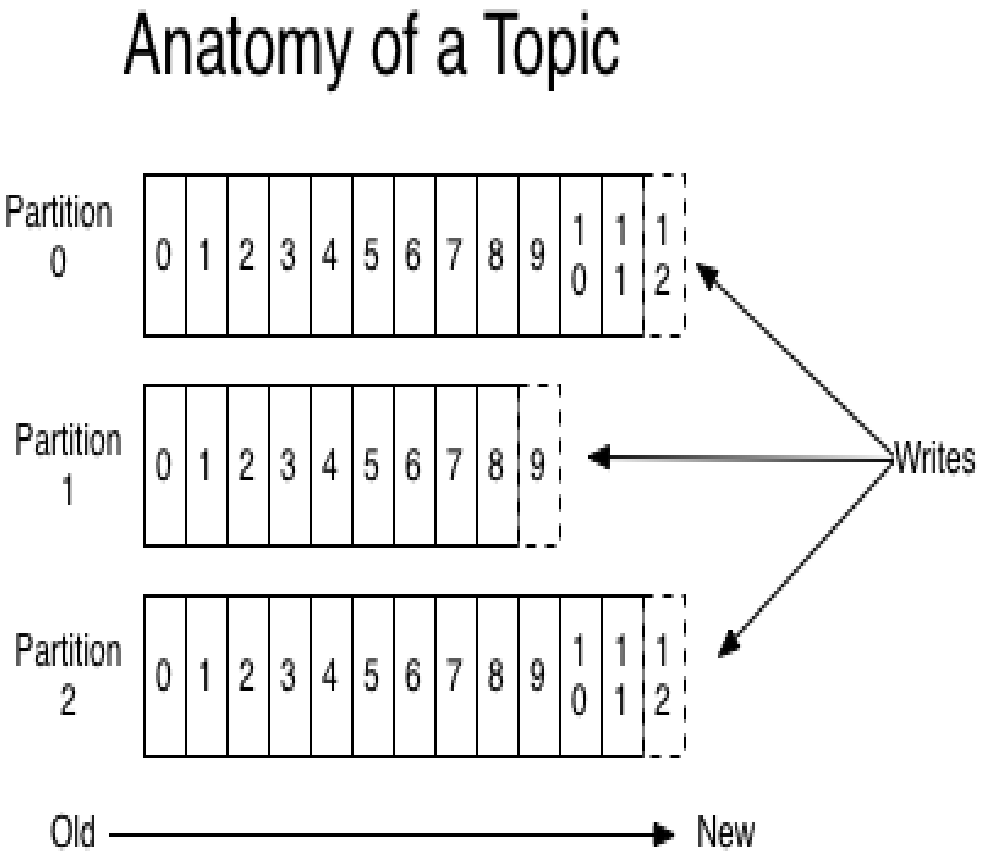

Рисунок 3 Схема теми

Кожен розділ являється частиною, незмінної послідовності повідомлень, які постійно додаються в журнал. Кожним повідомленням в розділах призначений порядковий номер ID який називається зміщенням, та однозначно ідентифікує кожне повідомлення в розділі.

Кафка кластер зберігає всі опубліковані повідомлення, будь вони чи ні, на заданий період часу. Наприклад, якщо збереження журналу встановлено на протяжність двох днів, потім протягом двох днів після публікації повідомлення воно доступне для споживання, після чого воно буде відкинутий, щоб звільнити місце. Продуктивність Кафки ефективно постійна по відношенню до розміру даних так що, збереження безлічі даних не є проблемою.

Насправді тільки метадані зберігаються на кожного споживача, становище споживача в журналі, називається "зсув". Цей зсув контролюється

споживачем: зазвичай споживач буде просувати його зміщення лінійно, як він читає повідомлення, але насправді становище його регулюється споживачем, і він може споживати повідомлення в будь-якому порядку, якому захоче. Наприклад, споживач може скинути з старше зміщення для переробки.

Ця комбінація особливостей означає, що Кафка споживачі являються дуже дешевими: вони приходять і йдуть без особливих впливів на кластері або на інших споживачів. Наприклад, ви можете використовувати наші інструменти командного рядка для "хвоста" вмісту будь-якої теми без змін, що споживається існуючими споживачами.

Перегородки в журналі служать декільком цілям. По-перше, вони дозволяють журналу масштабироваться за межі розміру, який поміщається на одному сервері. Кожен розділ повинен відповідати на серверах, на яких розміщені, але тема може мати багато розділів, так що можуть обробляти довільну кількість даних. По-друге, вони виступають в якості одиниці паралелізму.

1.4.2 Apache Hbase

HBase - нереляційна розподілена база даних з відкритим вихідним кодом; написана на Java; є аналогом Google BigTable. Розробляється в рамках проекту Hadoop фонду Apache Software Foundation. Працює поверх розподіленої файлової системи HDFS і забезпечує BigTable-подібні можливості для Hadoop, тобто забезпечує відмовостійкий спосіб зберігання великих обсягів розріджених даних.

Підтримка компресії, операції в пам'яті і фільтра Блума для кожного базового стовпчика реалізовані в HBase відповідно до документації BigTable. Таблиці в HBase можуть служити входом і виходом для роботи реалізації MapReduce в проекті Hadoop, і можуть бути отримані, не тільки через Java API, але й через API REST, Avro або Thrift.

HBase не є прямою заміною класичних SQL баз даних, хоча останнім часом у цій сфері вона стала працювати значно краще і в даний час

використовується для управління даними на кількох веб-сайтах, в тому числі Facebook використовує її для своєї платформи повідомлень.

Додатки зберігають дані в таблицях, що складаються з рядків і стовпців. Для елементів таблиці (перетину рядків і стовпців) діє контроль версії. За замовчуванням як версії використовується тимчасова мітка, автоматично призначається HBase на момент вставки. Вміст комірки являє собою неінтерпретіруемий масив байтів.

Ключі рядків таблиці теж є байтовими масивами, тому теоретично ключем рядка може бути що завгодно - від рядків до бінарних уявлень long і навіть серіалізованих структур даних. Рядки таблиці упорядковано по ключу рядків (первинному ключу таблиці). Сортування здійснюється в порядку проходження байтів. Усі звернення до таблиці виконуються по первинному ключу. Стовпці об'єднуються в сімейства стовпців. Всі члени сімейства стовпців мають загальний префікс, так наприклад, стовпці temperature: air та temperature: dew_ point належать сімейству temperature, a station: identifier належить сімейству station. Префікс сімейства стовпців повинен складатися з друкованих символів. Завершальна частина (кваліфікатор) може складатися з довільних байтів.

Сімейства стовпців таблиці повинні бути задані заздалегідь, як частина визначення схеми таблиці, проте нові члени родин можуть додаватися в міру потреби. Наприклад, новий стовпець station: address може бути переданий клієнту як частина оновлення, і його значення буде успішно зберігатися - за умови, що сімейство стовпців station вже існує в таблиці. Фізично всі члени родин стовпців зберігаються разом у файловій системі. Таким чином HBase як столбцево-орієнтоване сховище інформації, точніше було б сказати «орієнтоване на сімейства стовпців». Так як налаштування і специфікації задаються на рівні родин стовпців, бажано, щоб всі члени родин мали подібні схеми доступу та характеристики розмірів.

HBase автоматично виробляє горизонтальну розбивку таблиць на регіони.

Кожен регіон утворює підмножину рядків таблиці. Регіон визначається таблицею, якій він належить, своїм першим рядком (включно) і останнім рядком (без включення). Спочатку таблиця складається з одного регіону, але із зростанням розміру регіону після перевищення настроюваного порогового розміру він розбивається на два нових регіони приблизно рівних розмірів. До першого розбиття вся завантаження даних буде здійснюватися на одному сервері, на якому розміщений вихідний регіон. У міру зростання таблиці збільшується кількість її регіонів. Регіони є одиницями, розподілюваними в кластері HBase. Якщо таблиця виявляється занадто великою для одного окремого серверу, вона може обслуговуватися кластером серверів, на кожному вузлі якого розміщується підмножина регіонів таблиці. Крім того, регіони забезпечують розподіл навантаження на таблицю. Сукупність відсортованих регіонів, доступних по мережі, утворює загальний вміст таблиці.

У нашому випадку він використовується як метасховищк періодично (задавано) класіфіціковних параметрів, за певний період часу.

1.4.3 Apache Storm

В Минулому десятилітті відбулася революція в обробці даних. MapReduce, Hadoop, і пов'язані з ними технології зробили можливим зберігання і обробки даних в масштабах, раніше немислимим. На жаль, ці дані технології обробки не є системами обробки в реальному часі, і вони не повинні бути. Немає способі що перетворить Hadoop в систему обробкм в режимі реального часу; в реальному часі обробка даних має принципово інший набір вимог, ніж пакетна обробки.

Однак, обробка даних в реальному часі в масовому масштабі стає все більшою і більшою вимогою для бізнесу. Відсутність "Hadoop в реальному часі" стала найбільшою дірою в екосистемі обробки даних.

Шторм заповнює цю дірку.

Перед Storm, вам, як правило, доведеться вручну побудувати мережу черг і робітників, щоб зробити обробку в реальному часі. Працівники будуть

обробляти повідомлення з черги, оновлювать бази, і відправляти нові повідомлення для інших черг для подальшої обробки. На жаль, цей підхід має серйозні обмеження:

- Занудство : Ви проводите більшу частину часу розробки в налаштуванні того куди відправляти повідомлення, розгортання робітників і розгортання проміжних черг. Логіці обробки в реальному часі відповідає порівняно невеликий відсоток від вашого коду.
- Крихке : Там буде невелика відмовостійкість. Ви несете відповідальність за збереження данних кожного робочого стояти в черзі.
- Неможливість масштабувати: Коли пропускна потреба повідомлення занадто високою для одного працівника або черги, необхідно розділити дані по іншим працівникам. Ви повинні переналаштувати інших працівників, щоб дізнатися про нові доступні місця, для відправки повідомлення. Хоча черги і робочі парадигми ламаються при великій кількості повідомлень, обробка повідомлення явно фундаментальною парадигмою обчислень в реальному часі. Питання: як ви зроюити це таким чином, щоб не втрачати дані, ваги для великих обсягів повідомлень і зберегти простоту у використанні і експлуатації?

Шторм задовольняє цим цілям.

Чому Шторм є важливим?

Шторм надає набір примітивів для ведення обчислень в реальному часі. Наприклад, як MapReduce значно полегшує написання паралельних пакетних обробон, примітиви шторму значно полегшають написання обчислень паралельного реального часу.

Ключові властивості Storm є:

- Надзвичайно широкий набір варіантів використання: Storm може бути використаний для обробки повідомлень і оновлення баз даних (обробка потоку), роблячи безперервний запит на потоки даних і потокових

результатів в клієнтів (безперервних) обчислювальних, розпаралелювання інтенсивних запит як пошуковий запит на льоту (віддалений RPC), і багато іншого. Невеликий набір примітивів задовільняє приголомшливу кількість випадків використання.

- Масштабованість: Storm діє для величезної кількості повідомлень в секунду. Для масштабування топології, все, що вам потрібно зробити, це додати машини і збільшити параметри паралелізму топології. Як приклад в масштабі Шторм, один з первинних заявок шторму оброблено 1000000 повідомлень в секунду на 10 вузлах кластера, в тому числі сотні звернень до бази даних в секунду, як частина топології. Використання Storm з Zookeeper для координації кластера робить масштабироваться до більш великих розмірів кластера.
- Гарантує відсутність втрати даних: система реального часу повинні мати тверді гарантії про успішну обробку даних. Система, яка падає, має дуже обмежені можливості використання. Шторм гарантує, що кожне повідомлення буде оброблено, і це знаходиться в прямій суперечності з іншими системами, такими як S4.
- Надзвичайно міцна: На відміну від систем, таких як Hadoop, які славляться тим що ними важко керувати, Storm кластери просто працюють. Це явною метою Буря проекту, щоб зробити користувальницький досвід управління Storm кластерами, безболісним, наскільки це можливо.
- Відмовостійкі: Якщо є помилки під час виконання ваших обчислень, шторм перепризначить завдання по мірі необхідності. Шторм гарантує, що обчислення може працювати вічно (або до тих пір, поки ви не вб’єте обчислення).
- Не важливість мови програмування : Надійна і масштабований реальному часі обробка не повинна бути обмеженою однією платформою. Storm топології і компоненти обробки можуть бути визначені в будь-якій

мові, що робить доступним Storm майже для кожного.

Компоненти Storm кластера

Шторм кластера зовні схожі на кластери Hadoop. У той час як на Hadoop запуску "MapReduce job", на Storm ви запускаєте "topology". "MapReduce job" і "топології" по суті дуже різні - основна відмінність в тому, що “MapReduce job” в кінцевому підсумку закінчується, в той час як “topology” обробляє повідомлення завжди (або до тих пір, поки її не вбити).

Є два види вузлів шторм кластера: головний вузол і робочих вузлів. Головний вузол працює демон, званий Nimbus, який схожий на JobTracker Hadoop Nimbus несе відповідальність за поширення коду в кластері, призначення завдань машин та моніторингу невдач.

На Кожному працюючому вузлі запущений демон, так званий Supervisor. Supervisor слухає роботу з що призначається його машині і запускає та зупиняє робочі процеси в міру необхідності, на основі того, що Nimbus присвоїв йому. Кожен робочий процес виконує підмножину топології; працююча топологія складається з багатьох робочих процесів, розкиданих по багатьох машинах.

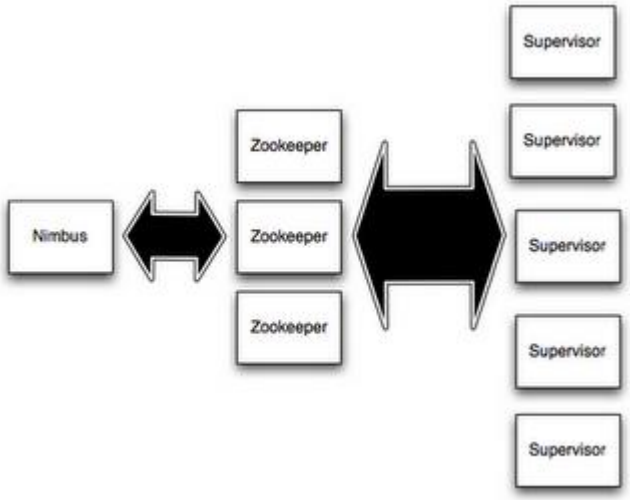

Рисунок 4 Взаємодія Nimbus з Supervisor

Вся координація між Nimbus та Supervisors здійснюється через кластер Zookeeper. Крім того, Керівник демони Німб є стійкими до збоїв та падіннь;

весь стан зберігається в Zookeeper або в локальному диску. Це означає, що ви можете виконати “kill -9” для Німба або наглядачів, і вони відновлять свою роботу знову, ніби нічого не сталося. Ця конструкція призводить до неймовірної стабільності.

Топології

Для обчислення в реальному часі на Storm, ви створюєте те, що називається "топологіями". Топологія представляє собою граф обчислень. Кожен вузол в топології містить логіку обробки, а зв'язки між вузлами показують, як дані повинні бути передані між вузлами навколо.

Запуск топології простий. По-перше, ви упаковуєте весь ваш код і залежності в одному jar архіві. Потім запускаєте команду,:

storm jar code.jar backtype.storm.MyTopology arg1 arg2

Це запускає клас backtype.storm.MyTopology з аргументами arg1 arg2 . Функція main класу визначає топологію і відправляє її Nimbus. Storm jar частина піклується про підключення до Nimbus і завантаження jar-архіву.

Визначеннями топології є Thrift структури(компонент, що буде розглянутий пізніше), а Nimbus є Thrift сервісом, ви можете створити і представляти топології, використовуючи будь-яку мову програмування. Наведений вище приклад є простий спосіб зробити це з JVM на основі мови Java.

Потоки

Центровою абстракцією в Storm є "потік". Потік необмежена послідовність кортежів. Шторм забезпечує примітиви для перетворення потоку в новий потік в розподілений та надійний спосіб. Наприклад, ви можете перетворити потік твітів в потік трендових тем.

Основні примітиви Шторм що дозволяють робити потокові перетворення є Spouts і Bolt. Spout і Bolt мають інтерфейси, які ви повинні бути реалізуваними, щоб запустити логіку програми.

Spout є джерелом потоків. Наприклад, Spout може зчитувати кортежі з черги Kestrel і випромінювати їх у вигляді потоку. Або Bolt можна підключити до API Twitter і випускати потік твітів.

Bolt може отримувать будь-яку кількість вхідних потоків, робить деяку обробку і, можливо, випускати нові потоки. Комплексні перетворення потоку, як обчислення потоку трендів з потоку твітів, вимагають декількох кроків і, таким чином, декілька болтів. Bolt'и можуть робити що-небудь з працюючих функцій, фільтрів кортежів, зробити потокове агрегати, робити потокове приєднання, спілкуватись з базами даних, і багато іншого.

Мережі Spout і Bolt упаковані в "топологію", яка є абстракцією верхнього рівня, що ви надаєте Storm-кластерам для виконання. Топологія представляє собою граф потоку перетворень, де кожен вузол є Spout або Bolt. Ребра в графі вказують, які bolt'и взаємодіють з якими потоками. Коли spout або bolt відправляє кортеж в потік, він посилає кортеж кожному bot, що підписався на цей потік.

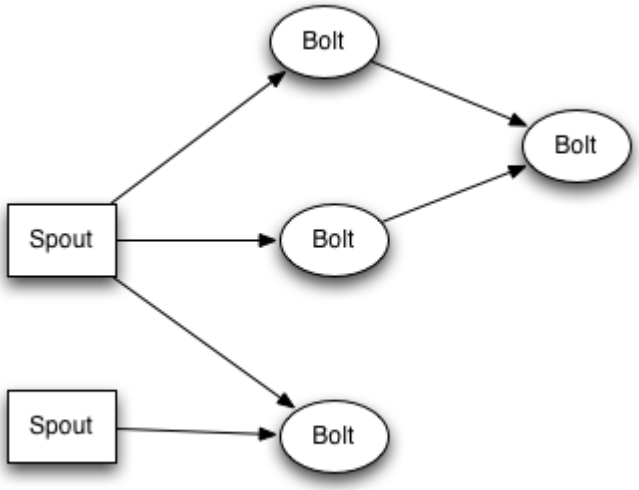

Рисунок 5 Типова взаємодія між процесами в клатері Storm [3]

Зв'язки між вузлами в топології вказують, як кортежі мають бути передані на наступний рівень. Наприклад, якщо є зв'язок між Spout A і Bolt B, ребро з Spout A до Bolt C, і з Bolt B в Bolt C, то відповідно кожен Spout випромінюючи кортеж, буде посилати як Bolt B так і Bolt C. Всі вихідні кортежі Bolt B підуть до Bolt C.

Кожен вузол в топології Storm виконує паралельно. У топології, ви можете вказати, скільки паралельних процесів ви хочете запуститти для кожного вузла, а потім Storm буде породжувати таку кількість потоків в кластері, щоб зробити обчислення.

Топологія працює вічно, або до того момента поки її не вбити. Storm буде автоматично перепризначувати будь невдалі завдання. Крім того, Storm гарантує, що не буде ніякої втрати даних, навіть якщо машини зупинять роботу і повідомлення видалються.

1.4.4 Java

Для реалізації Storm Hbase Kafka частин була обрана мова програмування java

оскільки вона поєднується з кожним з цих компонент.

Java є мовою програмування загального призначення, яка є багатопотоковою та об'єктно-орієнтованою, також спеціально розроблена, щоб

мати стільки імплементації залежностей наскільки це можливо. Вона призначена, щоб розробники додатків "написали один раз, і вона працює скрізь" (Wora), [13] це означає, що скомпілений Java код може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. Java додатки, як правило, збирається у байт-код що може працювати на будь віртуальній машині Java (JVM) незалежно від комп'ютерної архітектури. На 2015, Java є одним з найбільш популярних мов програмування у використанні, особливо для клієнт-серверних веб-додатків, з 9 млн зареєстрованих розробників. Java спочатку розроблена Джеймсом Гослінгом в Sun Microsystems (яка згодом була придбана корпорацією Oracle) і випущена в 1995 році в якості основного компонента Java платформи Sun Microsystems. Мова запозичила більшість синтаксису з C і C ++, але вона має менше низькорівневих можливостей, ніж будь-який з них.

Оригінальні Java компілятори, віртуальні машини і бібліотеки класів були спочатку випущений у SUN в пропрієтарних ліцензіях. Станом на травень 2007 року, у відповідності зі специфікаціями Java Community Process, Sun повторно ліцензувала більшість своїх технологій Java під GNU General Public License. Інші також розробили альтернативні реалізації цих технологій Sun, таких як GNU Compiler для Java (компілятор байт-код), GNU класи (стандартні бібліотеки), і IcedTea-Web (плагіни браузера для аплетів).

Java Virtual Machine (JVM) є абстрактною обчислювальною машиною. Є три тези про JVM: специфікація, реалізація та екземпляр. Специфікація це те, що формально описує, що потрібно від реалізації JVM. Наявність єдиної специфікації гарантує, що всі реалізації сумісні. Реалізація JVM є комп'ютерною програмою, яка відповідає вимогам специфікації JVM. екземпляр JVM це процес, який виконує комп'ютерну програму скомпільовану в Java байт-код.

Метою Java є портативність, що означає, що програми, написані для платформи Java можна запустити на будь-якій комбінації апаратного

забезпечення та операційної системи з адекватною підтримкою виконання. Це досягається шляхом компіляції коду Java мовою проміжного представлення так званий байт-код Java, а не безпосередньо до конкретної архітектури машинного коду. Інструкція Java байт-коду аналогічні машинному код, але вони призначені для виконання віртуальною машиною (VM), написано спеціально для серверного обладнання. Кінцеві користувачі зазвичай використовують Java Runtime Environment (JRE) встановлені на їх власній машині для автономних додатків Java, або у веб-браузері для Java-аплетів.

Стандартні бібліотеки забезпечують загальний спосіб доступу до специфічних функцій, таких як графіка, багатопотоковість , та networking.

Основна перевага використання байт-коду є можливість його портування. Тим не менш, додадткові витрати на інтерпретацію означають, що інтерпретовні програми майже завжди працювати повільніше, ніж скомпільовані . Java є незалежною від платформи. Але Java віртуальній машині необхідно перетворити Java байт-код в машинну мову, яка залежить від операційної системи, а це залежить від платформи.

Програми, написані на Java мають репутацію таких що працюють повільніше і вимагають більше пам'яті, ніж ті, які написані на C ++. Тим не менш, швидкість виконання програм Java "значно покращлась з введенням JIT компіляції в 1997/1998 для Java 1.1, додавання мовних особливостей, що підтримують більш глибокий аналіз коду (наприклад, внутрішніх класів, клас StringBuilder, додаткових тверджень і т.д.), і оптимізацій в віртуальній машині Java, таких як HotSpot стає за замовчуванням для SUN JVM в 2000.

Деякі платформи пропонують пряму підтримку апаратного забезпечення для Java; Є мікроконтролери, які можуть відпрацьовувати Java на апаратному рівні, а не програмному забезпеченні Java Virtual Machine, і процесори базрвані на ARM основі може мати апаратну підтримку для виконання Java байт-код через опції Jazelle.

Автоматичне управління пам'яттю

Java використовує автоматичний збирач сміття для управління пам'яттю в життєвому циклі об'єктів. Програміст визначає, коли об'єкти створюються і Java несе відповідальність за відновлення пам'яті, як тільки об'єкти більше не використовуються. Після того, як жодного посилання на об'єкт не залишається, зайнята пам'ять може бути звільненою автоматично збирачем сміття. Щось подібне до витоку пам'яті все ще може статися, якщо код програміста містить посилання на об'єкт, який більше не потрібен, як правило, коли об'єкти, які більше не потрібні зберігаються в контейнерах, які знаходяться все ще у використанні. Якщо методи для неіснуючого об'єкта визвуться то виникне "null pointer exception".

Одна з ідей, за автоматичної моделі управління пам'яттю в Java є те, що програмісти можуть бути позбавлені від тягаря того, щоб думати про ручне управління пам'яттю. У деяких мовах, пам'ять для створення об'єктів неявно виділяється в стеку, або явно виділяється і звільняється з купи. В останньому випадку відповідальність за управління пам'яттю лежить на програмісті. Якщо програма не звільняє об'єкт, виникає витік пам'яті. Якщо програма намагається отримати доступ або звільняти пам'ять, яка вже була звільнена, то результат не визначений і важко передбачити що трапиться, програма може стати нестабільною або закрешиться. Це може бути частково усунені шляхом використання смарт-вказівників, але вони додають накладні витрати і складність. Зверніть увагу, що збір сміття не заважає "логічному" витоку пам'яті, тобто ті виапдки, де на пам'ять і раніше посилаються, але ніколи не використовується.

Збір сміття може відбутися в будь-який момент. В ідеалі, це відбудеться, коли програма знаходиться в режимі очікування. Це гарантовано спрацьовує при недостатній кількості вільної пам'яті в купі для виділення нового об'єкта. Явне управління пам'яттю не є можливим в Java.

Java не підтримує C / C ++ стиль арифметики покажчиків, де об'єкт адреси та беззнакових цілих чисел (зазвичай довгі цілі) можуть бути

взаємозамінні. Це дозволяє збирачу сміття переміщувати посилання на об’єкти і забезпечує безпеку типів .

Як і в C ++ і деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java не є об'єктами. Значення примітивних типів, або зберігаються безпосередньо в полях (об'єктів) або в стеку (для методів), а не в купі. Це було свідоме рішення дизайнерів Java для підвищення продуктивності. Через це, Java не вважається чистим об'єктно-орієнтованою мовою програмування. Проте, станом на Java 5.0, Autoboxing що дозволяє програмістам, процесить ніби примітивні типи були станами їх класу-оболонки.

Java містить кілька типів збирачів сміття. За замовчуванням, HotSpot використовує збирач сміття паралельної продувки. Тим не менш, є також кілька інших збирачів сміття, які можуть бути використані для управління купою пам’яті. Для 90% додатків в Java, Паралельний Марк-розгортальний збирач сміття достатній. Oracle прагне замінити CMS зі сміттям першим колектором(G1).

1.4.5 Python

Для розробки взаємодії ріалтайм системи з користувачем було розроблено вебсервер що отримує данні з системи та через взаємодію з веб інтерфойсом виводить результат користувачеві.

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом з динамічною семантикою і динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python і стандартні бібліотеки доступні як в скомпільованій так і у вихідній формі на всіх основних платформах. У мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна і аспектно-орієнтований.

Python транслятори доступні для установки на багатьох операційних системах, дозволяючи виконання коду Python на різноманітних системах. Використання сторонніх інструментів, таких як py2exe або Pyinstaller, дозволяє Python коду бути упакованим в автономних виконуваних програмах для деяких з найбільш популярних операційних систем, що дозволяє для розповсюдженню програмного забезпечення на основі Python для використання на цих середовищах без установки інтерпретатора Python.

Серед основних її переваг можна назвати наступні:

- чистий синтаксис (для виділення блоків слід використовувати відступи);
- переносимість програм (що властиво більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних модулів (включаючи модулем для розробки графічного інтерфейсу);
- можливість використання Python в діалоговому режимі (дуже корисно для експериментування та рішення простих задач);
- стандартний дистрибутив має просте, але разом з тим досить потужне середовище розробки, яка називається IDLE і яке написано на мові Python;
- зручний для вирішення математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, в діалоговому режимі може використовуватися як потужний калькулятор).

Python має ефективні структури даних високого рівня і простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки додатків в багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата стандартна бібліотека (як початкові тексти, так і бінарні дистрибутиви для всіх основних операційних систем)

можуть бути отримані з сайту Python www.python.org, і можуть вільно поширюватися. Цей же сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями і типами даних, розробленими на C або C ++ (або іншою мовою, яку можна викликати за C). Python також зручна як мова розширення для додатків, що вимагають подальшого налагодження.

Розробники мови Python є прихильниками певної філософії програмування, яку називають «The Zen of Python» («Дзен Пайтона») . Її текст можна отримати в інтерпретаторі Python за допомогою команди import this (один раз за сесію). Автором цієї філософії вважається Тім Пейтерс.

Текст філософії:

- Гарне краще, ніж потворне.
- Явна краще, ніж неявне.
- Просте краще, ніж складне.
- Складне краще, ніж заплутане.
- Плоске краще, ніж вкладене.
- Розріджене краще, ніж щільне.
- Легкість читання має значення.
- Особливі випадки не настільки особливі, щоб порушувати правила.
- При цьому практичність важливіше бездоганності.
- Помилки ніколи не повинні замовчуватися.
- Якщо не замовчуються явно.
- Зустрівши двозначність, відкинь спокуса вгадати.
- Повинен існувати один - і, бажано, тільки один - очевидний спосіб

зробити це.

- Хоча спочатку він може бути і не очевидним, якщо ви не голландець [11].
- Зараз краще, ніж ніколи.
- Хоча ніколи, як правило, краще, ніж просто зараз.
- Якщо реалізацію важко пояснити - ідея погана.
- Якщо реалізацію легко пояснити - ідея, можливо, хороша.
- Простір імен - чудова річ! Будемо робити їх побільше!

Python портовано і працює майже на всіх відомих платформах - від КПК до мейнфреймів. Існують порти під Microsoft Windows, всі варіанти UNIX (включаючи FreeBSD і GNU / Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS / 2, Amiga, AS / 400 і навіть OS / 390, Symbian і Android [14].

У міру старіння платформи її підтримка в основний гілці мови припиняється. Наприклад, із серії 2.6 припинена підтримка Windows 95, Windows 98 і Windows ME. Однак на цих платформах можна використовувати попередні версії Python - тепер співтовариство активно підтримує версії Python починаючи від 2.3 (для них виходять виправлення).

При цьому, на відміну від багатьох портованих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM / DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java - Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, яка підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python і навіть бути написаними на ньому. Також кілька проектів забезпечують інтеграцію з платформою Microsoft.NET, основні з яких - IronPython і Python.Net.

Python, як і багато інших різних мов, не застосовують, наприклад, JIT-компілятори, мають загальний недолік - порівняно невисоку швидкість виконання програм. Однак, у випадку з Python цей недолік компенсується

зменшенням часу розробки програми. В середньому програма, написана на Python, в 2-4 рази компактніше, ніж її аналог на C ++ або Java. Збереження байт-коду (файли .pyc і .pyo) дозволяє інтерпретатору не витрачати зайвий час на перекомпіляцію коду модулів при кожному запуску, на відміну, наприклад, від мови Perl. Крім того, існує спеціальна JIT-бібліотека psyco (проте призводить до збільшення споживання оперативної пам'яті). Ефективність psyco в значній мірі залежить від архітектури програми.

Існують проекти реалізацій мови Python, що вводять високопродуктивні віртуальні машини (ВМ) як компілятора заднього плану. Прикладами таких реалізацій може служити PyPy, заснований на LLVM; більш ранньої ініціативою є проект Parrot. Очікується, що використання ВМ типу LLVM призведе до тих самих результатів, що і використання аналогічних підходів для реалізацій мови Java, де низька обчислювальна продуктивність в основному подолана.

Безліч програм / бібліотек для інтеграції з іншими мовами програмування надають можливість використовувати іншу мову для написання критичних ділянок.

У найпопулярнішій реалізації мови Python інтерпретатор досить великий і більш вимогливий до ресурсів, ніж в аналогічних популярних реалізаціях Tcl, Forth, LISP або Lua, що обмежує його застосування у вбудованих системах. Тим не менш, Python знайшов застосування в КПК і деяких моделях мобільних телефонів

1.4.6 Django

Django (Джанго) — високорівневий відкритий Python-фреймворк для розробки веб-систем. Названо його було на честь джазмена Джанго Рейнхардта (відповідно до музичних смаків одного зі засновників проекту).

Сайт на Django будується з однієї або декількох частин, які рекомендується робити модульними. Це одна з істотних архітектурних відмінностей цього фреймворку від деяких інших (наприклад Ruby on Rails).

Архітектура Django подібна на «Модель-Вид-Контролер» (MVC). Однак, те що називається «контролером» в класичній моделі MVC, в Django називається «вид» (англ. view), а те, що мало б бути «видом», називається «шаблон» (англ. template). Таким чином, MVC розробники Django називають MTV («Модель-Шаблон-Вид»).

Початкова розробка Django, як засобу для роботи новинних ресурсів, досить сильно позначилася на його архітектурі: він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований модуль для керування вмістом, який можна включити в будь-який сайт, зроблений на Django, і який може керувати відразу декількома сайтами на одному сервері. Адміністративний модуль дозволяє створювати, змінювати і вилучати будь-які об'єкти наповнення сайту, протоколюючі всі дії, а також надає інтерфейс для управління користувачами і групами (з призначенням прав).

У дистрибутив Django також включені програми для системи коментарів, синдикації RSS і Atom, «статичних сторінок»(якими можна управляти без необхідності писати контролери та відображення), перенаправлення URL та інше.

Django був створений для управління сайтами новин LJWorld.com, lawrence.com і KUsports.com компанії The World Company (Лоуренс, Канзас[en]), США), але з моменту початку розповсюдження його у статусі відкритого програмного забезпечення отримав величезну популярність в усьому світі як платформа до численних систем.

Розробники — засновники проекту:

- Адріан Головатий
- Саймон Віллісон (англ. Simon Willison),

- Джекоб Каплан-Мосс (англ. Jacob Kaplan-Moss),
- Вілсон Майнер (англ. Wilson Miner)

Деякі можливості Django:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами на більшість мов;
- диспетчер URL на основі регулярних виразів;
- розширювана система шаблонів з тегами та наслідуванням;
- система кешування;
- інтернаціоналізація;
- архітектура застосунків, що підключаються, які можна встановлювати на будь-які Django-сайти;
- «generic views» - шаблони функцій контролерів;
- авторизація та аутентифікація, підключення зовнішніх модулів аутентифікації: LDAP, OpenID та ін.;
- система фільтрів («middleware») для побудови додаткових обробників запитів, наприклад включені в дистрибутив фільтри для кешування, стиснення, нормалізації URL і підтримки анонімних сесій;
- бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД);
- вбудована автоматична документація по тегам шаблонів та моделям даних, доступна через адміністративний застосунок.

Різні компоненти фреймворку між собою пов'язані слабо, тому достатньо будь-яку частину замінити на аналогічну. Наприклад, замість вбудованих шаблонів можна використовувати Mako або Jinja.

1.4.7 HTML та XHTML

Використано для отримання веб-інтерфейсу що виводить аналізовані данні.

HTML — Мова розмітки гіпертекстових документів —стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документу та ідеологію структурної розмітки тексту.

Попри те, що HTML — штучна комп'ютерна мова, вона не є мовою програмування.

HTML разом із каскадними таблицями стилів та вбудованими скриптами — це три основні технології побудови веб-сторінок.

HTML впроваджує засоби для:

- створення структурованого документу шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

XHTML (Extensible Hypertext Markup Language) — мова розмітки, що має таку саму виразну силу як і HTML але відповідає синтаксичним правилам XML.

В той час як HTML побудовано на основі правил SGML, XHTML побудовано на основі правил XML, суворішої підмножини правил SGML. Оскільки XHTML документи мають бути коректними XML документами, їх обробку можна здійснювати стандартними інструментами обробки XML

документів на відміну від HTML, який вимагає порівняно складніших, важчих і повільніших синтаксичних аналізаторів. XHTML можна розглядати як, багато в чому, перетин HTML і XML, оскільки цей стандарт є переформулюванням HTML засобами XML. XHTML 1.0 став рекомендацією консорціуму W3C 26 січня 2000. XHTML 1.1 став рекомендацією W3C 31 травня 2001.

XHTML 1.0 є «реформулюванням трьох типів документів стандарту HTML 4 засобами XML 1.0».[1] World Wide Web Consortium (W3C) також продовжує підтримку Рекомендації HTML 4.01 та активну роботу над специфікаціями стандартів HTML5 і XHTML5. В поточному документі Рекомендацій XHTML 1.0, який було опубліковано та переглянуто до серпня 2002 року, W3C зазначив, що, "Сімейство XHTML є наступним кроком в еволюції Інтернету. Шляхом переходу на XHTML сьогодні, розробники контенту можуть увійти в світ XML з усіма супутніми перевагами, залишаючись впевненими в зворотній та майбутній сумісності їхнього контенту.

Проте в 2004 році, незалежно від W3C було створено Робочу групу з технологій застосування гіпертексту у Вебі (WHATWG), для роботи по вдосконаленню звичайного HTML не заснованого на XHTML. Більшість великих виробників браузерів не бажали реалізовувати функції з нових проектів стандартів W3C XHTML оскільки вважали, що вони не відповідають сучасним потребам розвитку Інтернету, а W3C захопився формалізмом XML і не реагує на реальні вимоги виробників. Apple, Mozilla та Opera сформували робочу групу WHATWG, яка почала працювати над стандартом HTML5, який допускав, але не вимагав застосування XML. У 2007 році, Робоча група W3C HTML проголосувала за офіційне визнання HTML5 і роботу над ним як наступне покоління стандарту HTML.[3] У 2009 році консорціум W3C дозволив добігти до кінця терміну дії Статуту Робочої групи XHTML 2, визнавши, що HTML 5 буде єдиним наступним поколінням стандарту HTML, як з XML, так і не-XML серіалізацію.

XHTML був розроблений з метою зробити HTML більш розширюваним і

підвищити сумісність з іншими форматами даних. HTML 4 побудований на основі та є застосуванням стандартної узагальненої мови розмітки (SGML), однак специфікація SGML складна, і як веб-браузери, так і Рекомендація HTML 4 не були повністю сумісними з нею. Стандарт XML, затверджений в 1998 році, пропонував простіший формат даних, ближче за духом до HTML 4.[7] Існували сподівання, що за допомогою переходу на формат XML, HTML стане сумісним із загальними інструментами XML; а проксі сервери зможуть перетворювати документи, у разі необхідності, для пристроїв з обмеженими можливостями, таких як мобільні телефони. Завдяки використанню просторів імен, XHTML документи могли б включати фрагменти інших, основаних на XML, мов, таких як Scalable Vector Graphics і MathML. Нарешті, відновлення роботи дала б можливість розділити HTML на компоненти для повторного використання (XHTML Модулі) і очистити неохайні частини мови.

1.4.8 CSS

Каскадні таблиці стилів (англ. Cascading Style Sheets або скорочено CSS) — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці

веб-сторінок. Головна перевага блочної верстки — розділення змісту сторінки (даних) та їхньої візуальної презентації.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.)

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS. Стилі для відображення сторінки можуть бути:

Стилі автора (інформація надана автором сторінки):

- зовнішні таблиці стилів (англ. stylesheet), найчастіше окремий файл або файли .css;
- внутрішні таблиці стилів, включені як частина документу або блоку;
- стилі для окремого елементу.

Стилі користувача

локальний .css-файл, вказаний користувачем для використання на сторінках і вказаний в налаштуваннях браузера (наприклад Opera)

Стилі переглядача (браузера)

стандартний стиль переглядача, наприклад стандартні стилі для елементів,

визначені браузером, використовуються коли немає інформації про стиль елемента або вона неповна.

Стандарт CSS визначає порядок та діапазон застосування стилів, те, в якій послідовності і для яких елементів застосовуються стилі. Таким чином, використовується принцип каскадності, коли для елементів вказується лише та інформація про стилі, що змінилася або не визначена загальнішими стилями.

Переваги

Інформація про стиль для усього сайту або його частин може міститися в одному .css-файлі, що дозволяє швидко робити зміни в дизайні та презентації сторінок;

Різна інформація про стилі для різних типів користувачів: наприклад великий розмір шрифту для користувачів з послабленим зором, стилі для виводу сторінки на принтер, стиль для мобільних пристроїв;

Сторінки зменшуються в об'ємі та стають більш структурованими, оскільки інформація про стилі відділена від тексту та має певні правила застосування і сторінка побудована з урахуванням їх;

Прискорення завантаження сторінок і зменшення обсягів інформації, що передається, навантаження на сервер та канал передачі. Досягається за рахунок того, що сучасні браузери здатні кешувати (запам'ятовувати) інформацію про стилі і використовувати для всіх сторінок, а не завантажувати для кожної.

1.4.9 JavaScript та JQuery

JavaScript – прототипна-орієнтована сценарна мова програмування. Є реалізацією мови ECMAScript (стандарт ECMA-262).

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипна парадигма програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java, але при цьому легким для використання непрограмістів. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від ряду мов програмування, використовуваних у веб-розробці.

Назва «JavaScript» є зареєстрованим товарним знаком компанії Oracle Corporation. Назва "ECMAScript" не є торговим знаком будь-яких компаній.

JavaScript є об'єктно-орієнтованою мовою, але використовне в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними клас-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання - що додає мові додаткову гнучкість.

Незважаючи на схожий з Сі синтаксис, JavaScript у порівнянні з мовою Сі має корінні відмінності:

- об'єкти, з можливістю інтроспекції;
- функції як об'єкти першого класу;
- автоматичне приведення типів;
- автоматична Збірка сміття;
- анонімні функції.

У мові відсутні такі корисні речі , як:

- модульна система: JavaScript не надає можливості управляти залежностями і ізоляцією областей видимості;
- стандартна бібліотека: зокрема, відсутній інтерфейс програмування додатків по роботі з файловою системою, управлінню потоками

введення-виведення, базових типів для бінарних даних;

- стандартні інтерфейси до веб-серверів і баз даних;
- система управління пакетами, яка б відстежувала залежності і автоматично встановлювала їх

jQuery — популярна JavaScript-бібліотека з відкритим сирцевим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресіґом (John Resig). Згідно з дослідженнями організації W3Techs, jQuery використовується понад половиною від мільйона найвідвідуваніших сайтів.[2] jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день.

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії).

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

Основне завдання jQuery — це надавати розробнику легкий та гнучкий інструментарій кросбраузерної адресації DOM об'єктів за допомогою CSS та XPath селекторів. Також даний фреймворк надає інтерфейси для Ajax-застосунків, обробників подій і простої анімації.

Принцип роботи jQuery полягає в використанні класу (функції), який при звертанні до нього повертає сам себе. Таким чином, це дозволяє будувати послідовний ланцюг методів.

1.4.10 Hadoop Ecosystem Distributives

В одній з статей були розглянуті переваги та недоліки основних поставників Hadoop Ecosystem

Для всіх тих, хто хоче використовувати потенціал великих даних, Hadoop є основною платформою для вибору. Це програмне забезпечення з відкритим доступом що дозволяє обробляти джерело величезних наборів даних, розподіляючи їх по серверах. Таким чином, він усуває залежність від високого рівня апаратних вимог і робить весь процес економічно вигідним для бізнесу в реалізації. Всі промислові системи обробки великих данних сьогодні використовувати Apache Hadoop в тій чи іншій мірі. Для спрощення роботи з Hadoop з'явилися підприємницькі версії, такі як Cloudera, MapR і Hortonworks.

У своєму первісному варіанті, Hadoop була розроблена як проста інфраструктура зберігання однократного запису. Але через роки вона перетворилася, розширивши рамки простої потужності веб-індексації. Грунтуючись на моделі MapReduce Google, Hadoop призначений для зберігання і обробки великих обсягів і різноманітності даних, які можуть знаходитись в декількох серверах.

У той час як розподілена файлова система Hadoop (в HDFS) допомагає розділити всі вхідні дані і зберігати їх на кількох вузлах, компонент MapReduce полегшує одночасну обробку даних по декільком вузлам.

Hadoop ні в якому разі не рішення з коробки. Для того, щоб побудувати дійсно інформаційно орієнтовані продукти, де рішення на базуються на основі даних, а не емпіричних робіт, компанії потрібно рішення для управління даними, яке не тільки пропонує надійне управління даними, але також легко кероване і легко інтегрується з існуючою інфраструктурою підприємства.

Гнучка модульна архітектура hadoop дозволяє додавати нові функціональні можливості для виконання різноманітних завдань з обробки великих даних. Кілька постачальників скористалися Hadoop та оптимізували свої коди, щоб змінити або розширити функціональні можливості. У процесі

вони змогли виправити деякі з недоліків, властивих Apache Hadoop. Три компанії, які дійсно виділяються в роботі: Cloudera, MapR і Hortonworks.

Порівнюючи три топ поставники Hadoop: Cloudera проти Hortonworks проти MapR

Cloudera був протягом самого довгого часу з моменту створення Hadoop. Hortonworks утворився пізніше. У той час як Cloudera і Hortonworks 100 відсотків з відкритим вихідним кодом, більшість версій MapR оснащені фірмовими модулями. Кожен постачальник має свою унікальну сильні та слабкі сторони, у кожного є певні пересічні функції, а також.

Cloudera

Cloudera Inc. була заснована розробниками Big data з Facebook, Google, Oracle і Yahoo в 2008 році була першою компанією для розробки та розповсюдження програмного забезпечення Apache Hadoop, і досі має велику базу користувачів з найбільшою кількістю клієнтів. Хоча ядро розподілу базується на основі Apache Hadoop, він також посточає фірмова Cloudera Management Suite для автоматизації процесу установки та надавання інших послуг для підвищення зручності користувачів, які включають скорочення часу розгортання, показуючи кількість вузлів в режимі реального часу », і т.д.

Hortonworks

Hortonworks, заснована в 2011 році, швидко перетворився в один з провідних постачальників Hadoop. Поставник забезпечує платформу з відкритим вихідним кодом на основі Apache Hadoop для аналізу, зберігання і управління великими даними. Hortonworks є єдиним комерційним постачальником, що поширює продукт з повним відкритим вихідним кодом Apache Hadoop без додаткового пропрієтарного програмного забезпечення. Дистрибутив HDP2.0 Hortonworks можна безпосередньо завантажити з веб-сайту, безкоштовно і легко встановити. Інженери Hortonworks знаходяться позаду більшості останніх нововведень Hadoop, в тому числі Yarn, який краще,

ніж MapReduce, позаду, в тому сенсі, що це дозволить включити більше каркасів обробки даних.

MapR

У стандартній версії з відкритим вихідним кодом програмне забезпечення Apache Hadoop приходить з низкою обмежень. Дистрибутиви, спрямовані на подолання проблем, що користувачі, як правило, стикаються в стандартних виданнях. Під вільною ліцензією Apache, всі три дистрибутиви надають користувачам версії з оновленнями, основного програмного забезпечення Hadoop.

Всі три топ-дистрибутивів Hadoop, Cloudera, MapR і Hortonworks пропонують консультації, навчання і технічну допомогу. Але на відміну від своїх двох суперників, Hortonworks стверджує, що на 100 відсотків з відкритим вихідним кодом. Cloudera включає масив патентованих елементів в його версії Enterprise 4.0, додаючи шари адміністративних та управлінських можливостей в основне програмне забезпечення Hadoop.

Йдучи далі, MapR замінює компонент HDFS і замість цього використовує свою власну фірмову файлову систему, звану MapRFS. MapRFS надає більш ефективне управління даними, надійність,і найголовніше, простота використання. В інших словах, більш готовий до виробництва, ніж його два інших конкуренти.

Через недавнє партнерство з Canonical(творець операційної системи Ubuntu), MapR пропонує Hadoop як компонент операційної системи Ubuntu за замовчуванням. Відповідно до умов партнерства, MapR в M3 видання для Apache Hadoop будуть інтегровані в операційну систему Ubuntu.

До M3 видання, MapR безкоштовно, але безкоштовна версія не вистачає деяких з її власних особливостей, а саме, JobTracker HA, NameNode HA, NFS-HA, Mirroring, Snapshot і декілька інших.

	Hortonworks	Cloudera	MapR
Performance and Scalability			
Data Ingest	Batch	Batch	Batch and streaming writes
Metadata Architecture	Centralized	Centralized	Distributed
HBase Performance	Latency spikes	Latency spikes	Consistent low latency
NoSQL Applications	Mainly batch applications	Mainly batch applications	Batch and online/real-time applications
Dependability			
High Availability	Single failure recovery	Single failure recovery	Self healing across multiple failures
MapReduce HA	Restart jobs	Restart jobs	Continuous without restart
Upgrading	Planned downtime	Rolling upgrades	Rolling upgrades
Replication	Data	Data	Data + metadata
Snapshots	Consistent only for closed files	Consistent only for closed files	Point-in-time consistency for all files and tables
Disaster Recovery	No	File copy scheduling (BDR)	Mirroring
Manageability			
Management Tools	Ambari	Cloudera Manager	MapR Control System
Volume Support	No	No	Yes
Heat map, Alarms, Alerts	Yes	Yes	Yes
Integration with REST API	Yes	Yes	Yes
Data and Job Placement Control	No	No	Yes
Data Access			
File System Access	HDFS, read-only NFS	HDFS, read-only NFS	HDFS, read/write NFS (POSIX)
File I/O	Append only	Append only	Read/write
Security: ACLs	Yes	Yes	Yes
Wire-level Authentication	Kerberos	Kerberos	Kerberos, Native

Рисунок 6 Порівняння дистрибутивів

В моєму випадку було обрано MapR оскільки він являється найпродуктивнішим серед дистрибутивів

1.5 Висновки

В цьому розділі було проведено опис доступних методів та програмних сервісів що використовувались в роботі. Також був проведений аналіз вхідних параметрів та проблематики для того що б правильно підібрати методи та сервіси для реалізації.

2. ОПИС ПРОГРАМНОГО ПРОДУКТУ. АНАЛІЗ АРХІТЕКТУРИ. СИСТЕМНІ ВИМОГИ. ДОКУМЕНТАЦІЯ. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

2.1 Загальний опис та аналіз архітектури.

Лямбда архітектура - архітектура обробки даних призначена для обробки величезної кількості даних, скориставшись обома batch і stream процесінг методами. Такий підхід до архітектури намагається збалансувати затримки, пропускної здатності та відмовостійкості з допомогою пакетної обробки, щоб забезпечити всебічну та точну обробку пакетних даних, в той час, одночасно з використанням обробки потоку в режимі реального часу, щоб забезпечити оперативну обробку на online даних. Два оброблені виходи можуть бути з'єднані перед представленням результатів. Підйом лямбда архітектури корелює з ростом великих даних, аналітики в режимі реального часу, і пом'якшенням затримки з Map-Reduce.

Лямбда архітектура залежить від моделі даних з тільки конкатеовним постійним джерелом данних, який служить в якості системи запису. Він призначений для поглинання і обробки повторних подій, які будуть додані до існуючих подій, а не перезаписуючи їх(наприклад надійшли нові данні а старі не затираються). Стан визначається з природно-погодинного впорядкування даних.

Лямбда архітектура описує систему, що складається з трьох шарів: пакетної обробки, обробки в реальному часі, і обслуговуючого прошарку для реагування на запити. Обробляючі частини отримують з незмінної майстер-копії всієї множини даних.

Пакетна обробка

Пакетна обробка попередньо обчислює результати, використовуючи розподілену систему обробки, яка може обробляти дуже великі обсяги даних. Пакетна обробка спрямована на ідеальну точність, будучи в змозі обробляти всі

доступні дані при генерації обробки. Це означає, що можна виправити будь-які помилки, повторно на основі повного набору даних запустивши цей процесс. Вихід, як правило, зберігається в базі даних,тільки для читання, для того щоб з поновленням повністю замінити існуючі передвичесленн

Apache Hadoop є стандартною системою пакетної обробки, та де-факто використовується в більшості високою пропускних архітектур.

Обробка в реальному часі

Обробка в реальному часі обробляє потоки даних в реальному часі і без вимог виправи або повноти. Цей спрямований звести до мінімуму затримки, забезпечуючи уявлення про ситуацію в реальному часі в самих останніх даних. По суті, швидкість шар відповідає за наповнення пробілів, викликаних відставанням пакетної обробки в наданні результату на основі самих останніх даних. Перегляди цей шар не може бути точним або повним, як ті, в кінцевому підсумку, вироблені пакетному шарі, але вони доступні практично відразу після прийому даних, і можуть бути замінений, коли данні з пакетного шару для тих же даних стають доступними.

Потік обробки технології, зазвичай використовуються в цьому шарі включають Apache Storm, SQLstream і Apache Spark. Вихід, як правило, зберігаються на швидких баз даних NoSQL.

Обслуговуючий прошарок

Вихід з пакетної обробки і обробки в реальному часі зберігаються в обслуговуючому прошарку, який реагує на вузькоспеціалізовані запити, повертаючи злиті результати від обох данних.

Приклади технологій, що використовуються в прямому шару включають Druid, який забезпечує єдиний кластер для обробки виведення з обох шарів, або більш прості системи коли данні що вимагають злиття більш прості Виділені бази данних, використовувані в прямому шару включають Apache Cassandra або

Apache HBase для виведення обробки в реальному часі, і Elephant DB або Cloudera Impala для виведення пакетного шару.

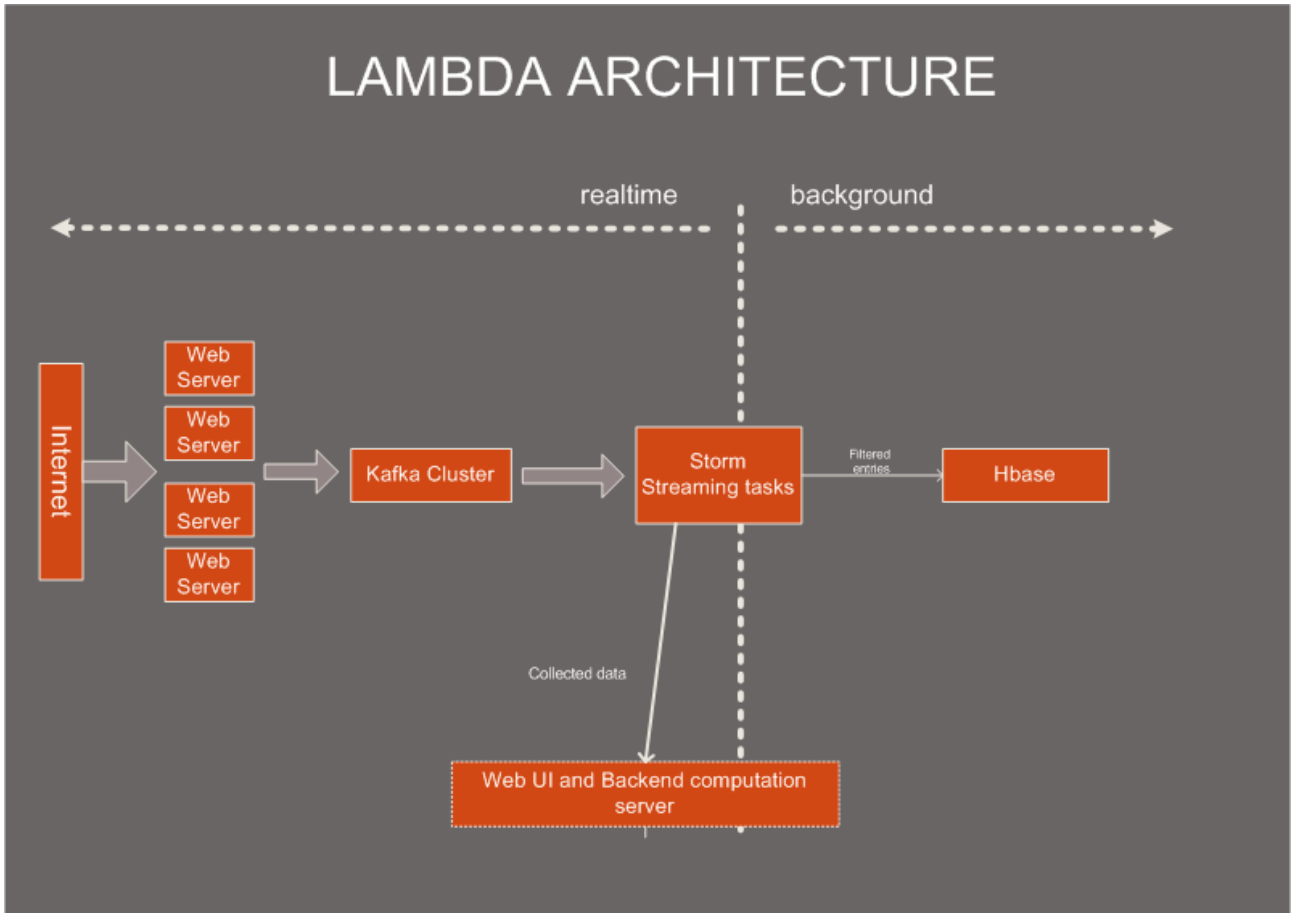

Рисунок 7 Лямда архітектура розробленої системи

В моєму випадку лямбда архітектура виглядає таким чином :

- Спочатку данні(url з данними що приходить з інтерент переглядів) потрапляють на Kafka cluster, що забезпечує стійкість системи і не дає данним бути втраченими
- Потім данні потрапляють на Storm кластер, який в свою чергу проводить обчислення класифікації первинних даних. Та передачі процессу обчислення пакетних данних. Та запис в hbase/
- Далі данні потрапляють на вебсервер який збирає запроцесені данні до купи та проводить кінцевий аналіз з виводом найпродуктивнішого товару.

2.2 Системні вимоги для інсталяції

Вимоги до операційної системи:

Основою для системи обробки був взятий MapR кластер оскільки в ньому файлова система показує найкращі результати серед усіх дистрибутивів Hadoop ecosystem. Також були використані MapR-DB таблиці, що являються покращеним варіантом Hbase таблиць і дають більшу швидкодію. MapR дистрибутив підтримує тільки Linux подібні системи

Таблиця 2.1 Мінімальні вимоги машини в кластері Storm

Unit	Type
CPU	64-bit
OS	Red Hat, CentOS, SUSE, or Ubuntu
Memory	4 GB minimum, more in production
Disk	Raw, unformatted drives and partitions
DNS	Hostname, reaches all other nodes
Users	Common users across all nodes; passwordless ssh (optional)
Java	Must run Java
Other	NTP, Syslog, PAM

Мінімальні вимоги для веб-сервера

Оскільки веб-сервер працює на Django то він повинен запускатись на Linux дистрибутиві, також там працює обчислювальна система, що теж накладує свої обмеження, зазвичай там будуть відбуватись процеси з’єднання з сервером та контроль над самою системою, в подальших версіях планується розробити підтримку так званого High Avalability. High Avalability – це підтримка для стабільності системи, коли падає сервер, то сама система не впала, а спрацював інший елемент

Таблиця 2.2 Мінімальні вимоги машини з вебсервером

Unit	Type
CPU	64-bit
OS	Red Hat, CentOS, SUSE, or Ubuntu
Memory	8 GB minimum, more in production
Disk	Raw, unformatted drives and partitions
DNS	Hostname, reaches Nimbus node
Users	Common users across all nodes; passwordless ssh (optional)
Python	Python 3.4 Django 1.8
Other	NTP, Syslog, PAM

В результаті нам потрібна система , що базується на MapR кластері, Storm частину нйкраще відділити від пакетної обробки та Hbase частини. Оскільки при запису нагрузка йде не тільки на Storm але й на Hbase.

2.3 Планування кластера та аналіз вхідних данних

Оцінимо об’єм вхідних данних. Оскільки в середньому в штатах активність проявляють 3 млн користувачів на сайті eBay.

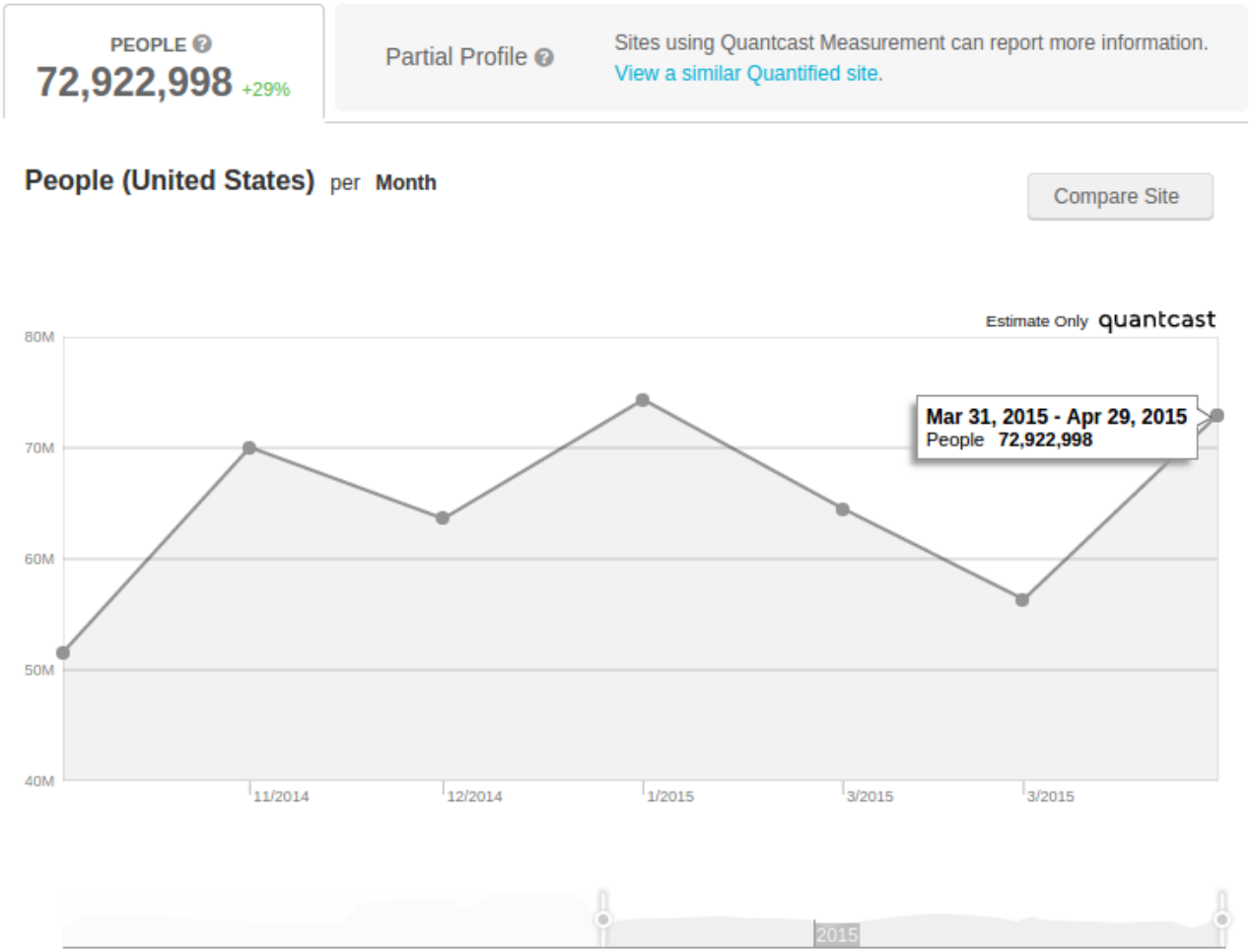

Рисунок 8 Активність користувачів в США на eBay

Та в світі об’єм продаж саме в штатах є 48%(дані показані на графіку нижче)

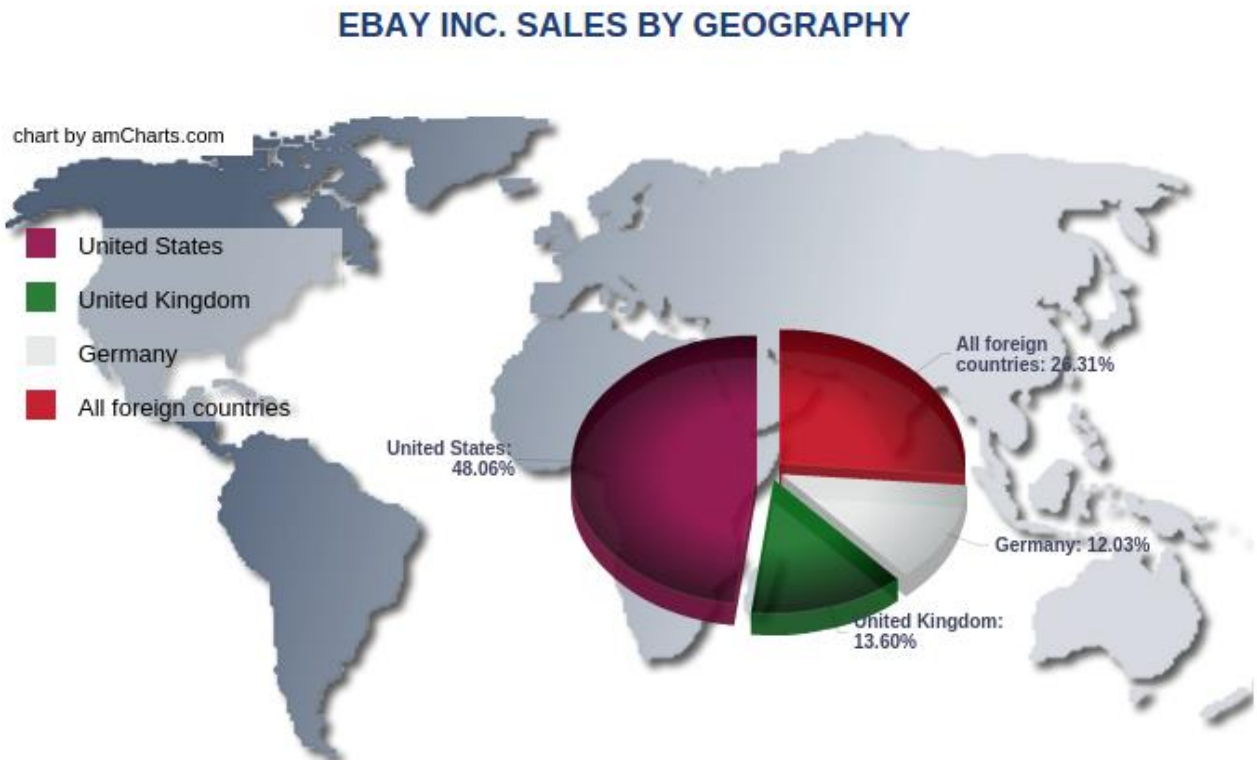

Рисунок 9 Активність покупок в світі

Тобто в середньому з данних по штатах, ми можемо оцінити кількість користувачів ebay в 7 мільйонів. Також для оцінки нам потрібно кількість запитів на товари. Дана статистика надається нижче.

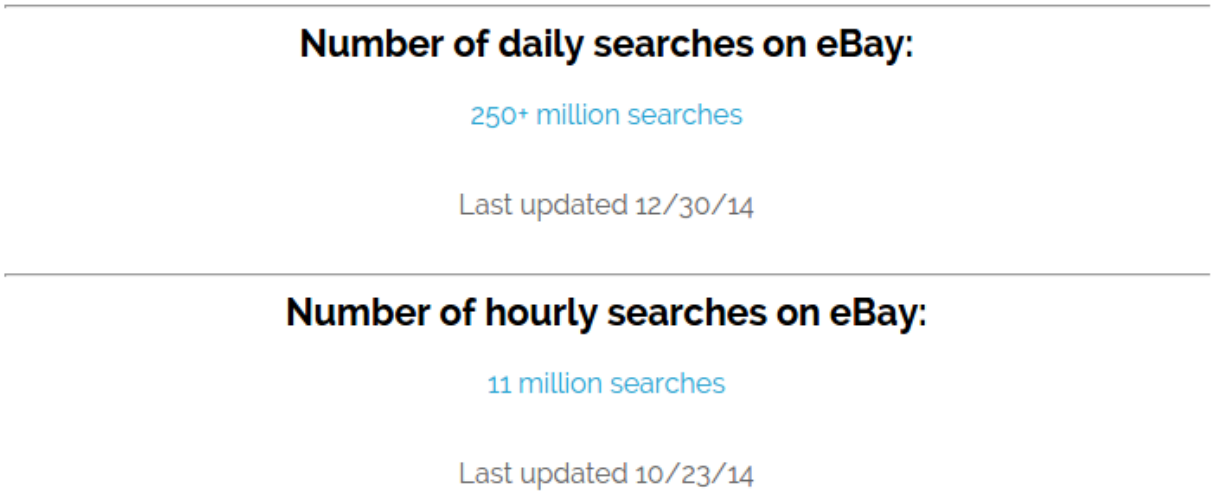

Рисунок 10 Статистичні дані по пошукам на сайті eBay

Тобто в середньому кожен користувач прогляда по 35 товарів в день.

Проте оскільки система не являється аналізатором одного eBay, то нижче наведена статистика двох внших сервісів (проте данні там не являються повними).

Amazon.com

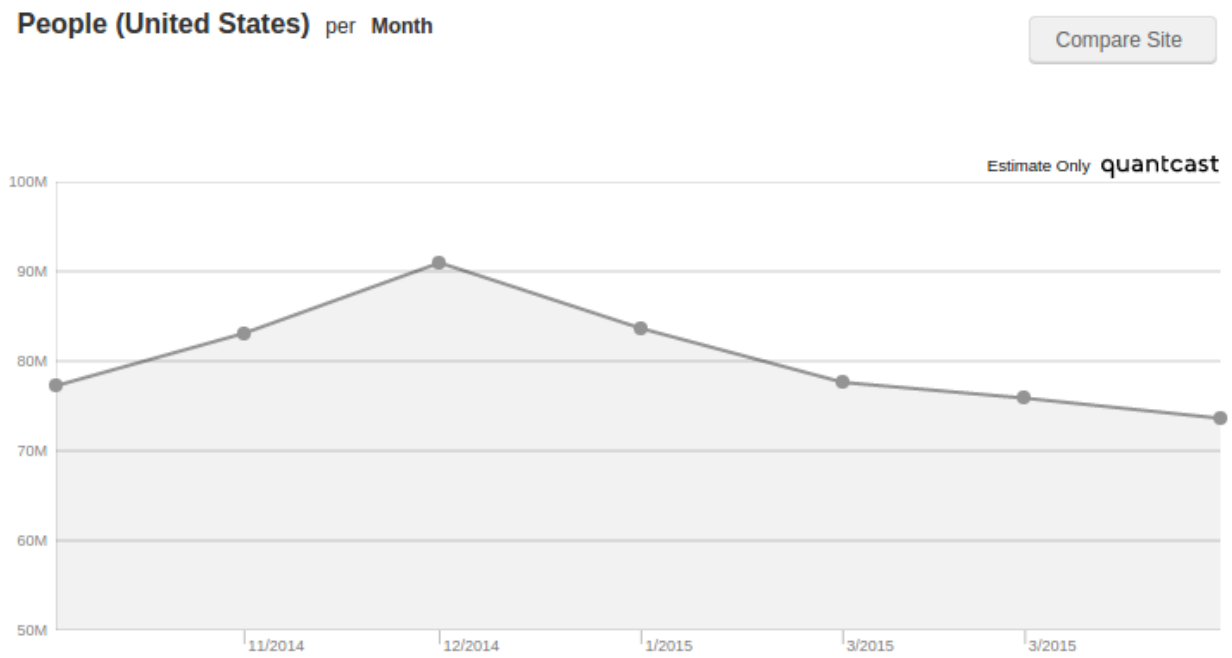

Рисунок 11 Активність користувачів в США Amazon.com

Як бачимо з наданого графіку в середньому дані наші мають досить подібну статистику до ebay і можна оцінити цю кількість рівною йому

Aliexpress.

Aliexpress являється найбільш багаточисельною серед наданих в середньому ми маємо 600 мільйонів користувачів в місяць.

З отриманих данних можемо прорахувать що в середньому у нас 3000 з ebay та amazon та 15000 з aliexpress записів в секунду. Тобто наша система має оброблять по 21 тисячу записів в секунду.

Було обраховано те що на одній машині що задовільняє мінімальниим вимогам, в середньому виконується обробка 930 записів в секунду. Нам потрібно як мінімум 23 таких сервера для того що б задовільнити обчислювальні вимоги.

Оскільки наші дані пишуться на роподілену файлову систему. То для повного зберігання данних з низькою частотою обробки нам потрібно велике сховище. При генерованих данних вийшло що в середньому в нас повинно оброблятись 400 гігабайт данних в день. Щоб мати можливість переобробляти дані раз в рік нам потрібне сховище хоча б на 150 терабайт.

Оскільки наша система підтримує Mirroring то нам потрібно хоча б вдвічі більше пам’яті.

2.4 Інсталяція програмного забезпечення

Для початку опишемо встановку MapR кластера.

Найпростішим варіантом встановки MapR кластера є встановка його задопомогою пакету MapR Installer.

Кроки при встановці MapR Installer.

- Виберіть вузол для запуску MapR Installer.

  Вузол, з якого запускається MapR Installer також має бути одним з вузлів, на які ви плануєте встановити кластер. Перш ніж почати, ви можете перевірити передумови і відомі обмеження на сайті MapR.

- Завантажити mapr-setup.sh сценарій.

  Після того як ви увійшли в систему або зареєструватися на mapr.com, ви можете завантажити mapr-setup.sh. Потім скопіюйте mapr-setup.sh на вузол, в якому буде запущений MapR Installer.

- Запустіть скрипт mapr-setup.sh щоб налаштувати вузол для запуску MapR Installer.

  Виконайте наступну команду як користувач root з каталогу, що містить сценарій:

  bash ./mapr-setup.sh

- Запустіть інсталятор MapR.

  Відкрийте URL MapR Installer: https://<MapR Installer Node hostname/Ipaddress>:9443

  Вам буде запропоновано увійти в систему як користувач Administrator MapR, який налаштований під час роботи mapr-setup.sh сценарію.

MapR Installer встановлює програмне забезпечення MapR 4.1 після проходження вас через процес вибору послуг та налаштування кластера. Після завершення установки, ви можете використовувати той же URL MapR

установника, щоб додати вузли та додаткові послуги в кластер. На цьому кроці потрібно вибрати ноду де будуть встановлені Nimbus Supervisor Ui Zookeeper та Hbase сервіси.

Опції встановки

1. Встановка вузлів в “хмарі”

При запуску установника MapR на вузлах в хмарі, зверніть увагу на наступні моменти:

- Переконайтеся, що порт 9443 відкритий.

  MapR Installer вимагає, щоб цей порт був доступний.

- URL-адреси MapR установника та обслуговування інтерфейсу повинен відноситись до зовнішнього URL, а не внутрішньою URL.

  Наприклад, коли ви відкриваєте URL MapR Installer, замініть будь-який внутрішніх хоста або IP-адресу та пов'язані з ним зовнішній адресу. Для Amazon EC2 і Google Compute Engine (GCE) кластерів, установник MapR автоматично переводить внутрішні адреси на зовнішні адреси.

- На сторінці Налаштування вузлів веб-інтерфейс установника MapR, переконайтеся, що ви виконаєте наступні дії:

  - Визначити кожного вузла з внутрішнім статичним IP-адресом або внутрішніх вирішуваних hostname.

  - Для віддаленої аутентифікації, використовувати той же ідентифікатор користувача і секретний ключ, який ви використовуєте для SSH в хмарних станах. Цей користувач повинен бути root або користувач з правами SUDO.

На підставі вимог і обмежень кластера брандмауера, вам, можливо, буде потрібно відкрити наступні типи портів для зовнішнього доступу:

- Порти, використовувані для користувача інтерфейсів, таких як MCS,

Resouce Manager UI, JobTracker UI, Storm UI, Django app ui.

- Порти, використовувані клієнтами, такими як Hive, Impala, Drill, and Spark-SQL.
- Порти, використовувані на клієнтських машинах MapR для доступу до MapR-FS або запуску Hadoop job Storm Job.

Вимоги до встановки.

Таблиця 2.3 Мінімальні вимоги для встановки кластера

Name	Version
MapR Node	Це має бути один з вузлів на якій планується встановка кластера
Package Dependencies	Відносно операційних систем він вимагає наступні пакети: на Ubuntu : python-pycurl openssh-client libssl1.0.0 sshpass Red Hat/ CentOS : python-pycurl openssh-clients openssl sshpass якщо не буде знайдено, mapr-setup.sh утиліта запропонує викачати дані пакети з інтернету.
Java	JDK 1.7 чи вище. якщо JDK 1.7 чи вище є недоступна, mapr-setup.sh встановить OpenJDK Java 1.7.
SSH Access	Повинен бути доступ (SSH access) для всіх вузлів що входять в кластер
Port Availability	Порт 9443 має бути відкритий між всіма вузлами в кластері.

Повинен бути встановлений браузер

Встановка компоненти.

- Компонента має стояти на одній з нод кластера.
- Потрібно встановити віртуальне середовище Python3.4
- в директорію з ним розпакувати архів з web-ui
- далі налаштувати setup.ini файл на роботу з середовищем
- зайти в віртуальне середовище

  source bin/activate
- з нього встановити залежності

  pip install requirements.txt
- запустити веб сервер

python manage.py collectstatic

python manage.py runserver

Далі програма готова до роботи.

2.5 Аналіз роботи програми та документація користувача

Запустивши програму ми повинні перейти з машини з доступом на сервер на адрес http://<Hostname>:8000 ми отримаєм таке вікно

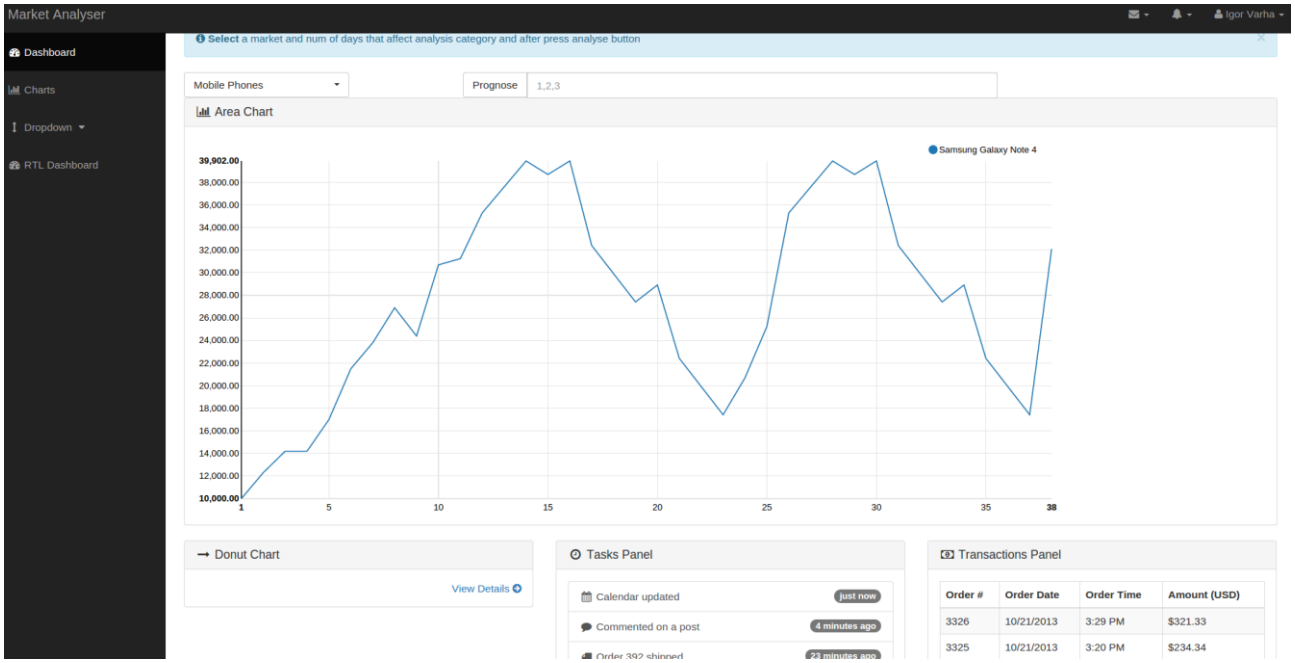

Рисунок 12 Вікно програми

Як ми бачимо в початковій версії в нас є вибір секції товару по якій ми будемо проводить аналіз та в вікно з інпутом числельних значень на скільки ми проводимо прогноз наші дані постійно оновлюються тому ми визначаєм який саме графік ми будем показувать

Далі що б запустить аналіз нам потрібно ввести кількість днів на яку буде проводитись прогноз, зацікавленості та нажати на кнопку “prognose”

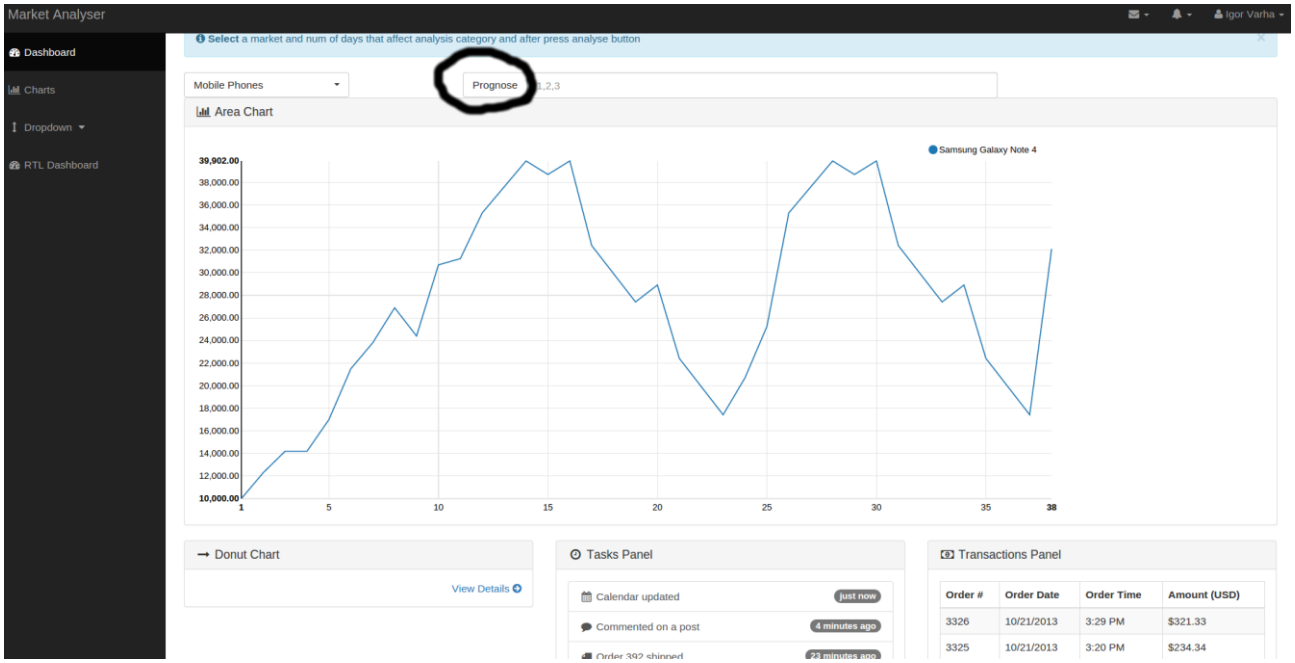

Рисунок 13 Вікно програми з виділеним елементом керування

Далі в нас виведуться данні по прогнозу:

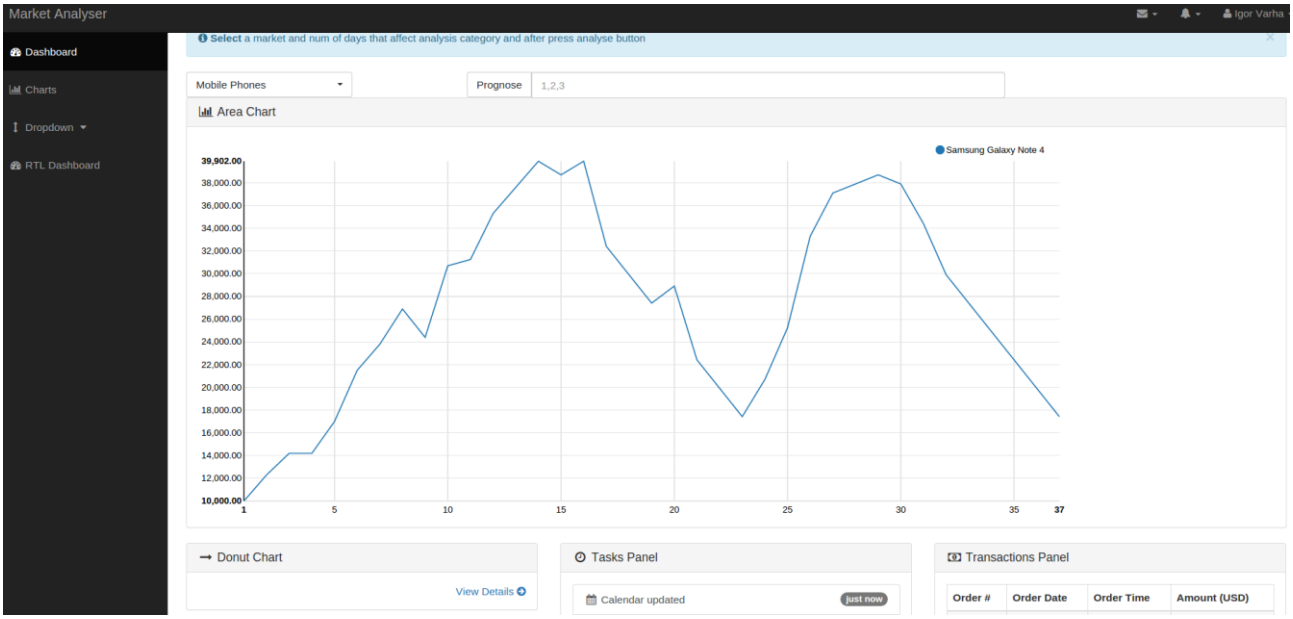

Рисунок 14 Вікно програми з виділеним елементом керування

Розглянемо їх ближче:

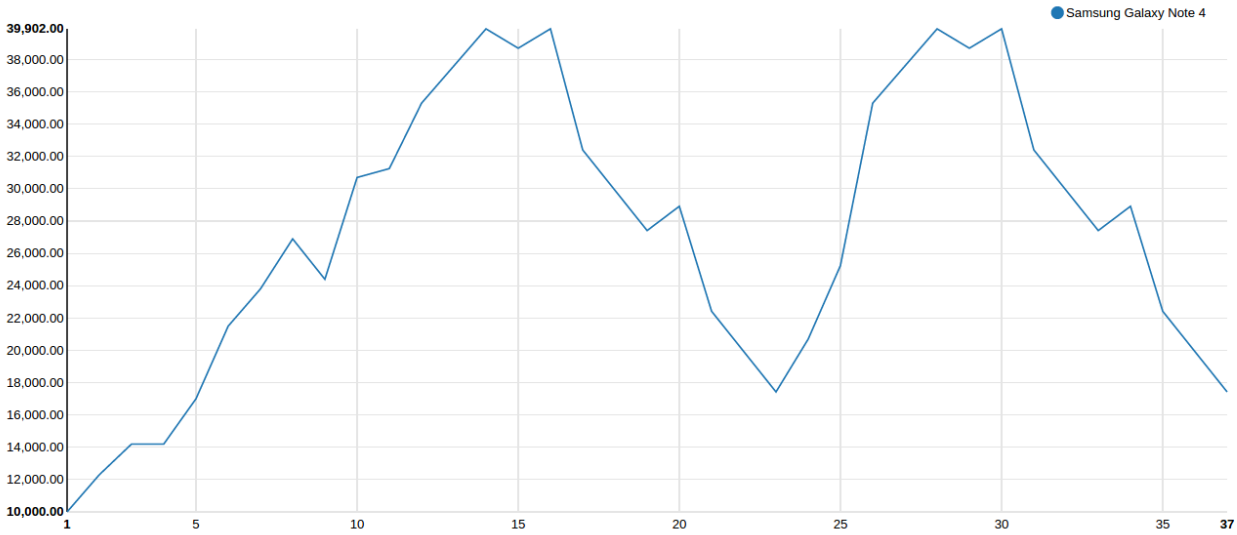

Рисунок 15 Вікно графіку непрогнозованої системи

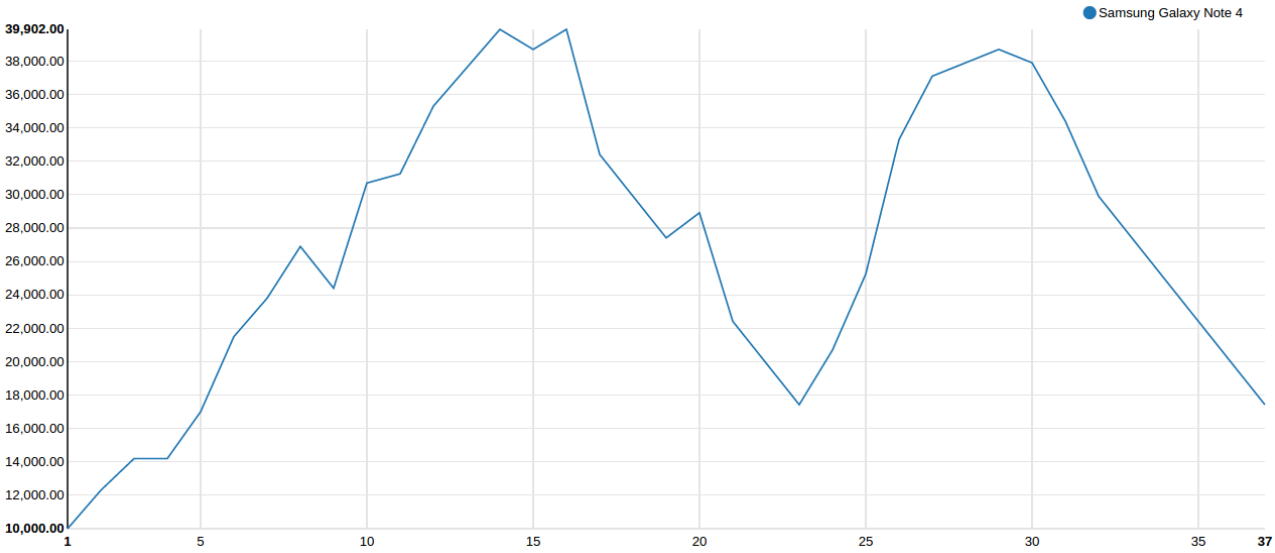

Рисунок 16 Вікно графіку тренованої на 25 періодах й прогнозовано після

Як бачимо на малюнку 14 дані в нас генеровні. Зробивши зріз і прогноз ми отримали графік на малюнку 15, що являється доволі точною моделлю за порахувавши середньоквадратичне відхилення ми отримали похибку рівну 30 відсоткам(по всіх товарах). Дані генерувались програмно з налаштуваннями, данна модель була завчасно згенерована з більшою продуктивністю, для того

що б оцінити що програма дійсно виконує свою головну ціль – прогнозування максимального прибутку

2.6 Висновки

В данному розділі було розлянуто розроблену систему, проведено аналіз результатів, показано, що данна програма виконує поставлені перед нею задачі

3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Результатом дипломної роботи є розроблена програма з визначення кредитоспроможності клієнта, безпечне використання якої вимагає якісного аналізу потенційно-небезпечних і шкідливих виробничих факторів, що створюються обладнанням в приміщенні під час його експлуатації, та заходів щодо їх усунення.

Приміщення, на якому досліджуються умови праці, є робочою кімнатою в підприємстві. В данному приміщенні проводиться аналіз ринку з подальшим прийняттям рішення про випуск товару в обіг.

3.1 Загальна характеристика приміщення і робочого місця

1. Розробка програмного забезпечення виконується в приміщенні (рис.4.1), яке знаходиться на другому поверсі 3-ох поверхового будинку з загальним та місцевим освітленням. В приміщенні одностороннє освітлення, вікна орієнтовані на схід, на вікнах є жалюзі. Стіни цегляні світлого кольору з коефіцієнтом відбиття 0,5, стеля білого кольору з коефіцієнтом відбиття 0,7. В приміщенні працює 4 людини, відповідно до цього отримаємо вхідні дані для аналізу потенційно-небезпечних і шкідливих виробничих факторів, які наведено в табл.3.1.

Таблиця 3.1 Вхідні дані

Параметри приміщення	**Значення**
Довжина х ширина х висота	6,6 х 6 х 2,7 м
Площа	39,6 м^2
Об’єм	106,92 м^3
Номер робочого місця	**Специфіка роботи**
I робоче місце	Програміст (спеціаліст з розробки десктопних програм)
II робоче місце	Інженер-алгоритміст (спеціаліст з дата майнингу та побудови алгоритмів)
III робоче місце	Аналітик
IV робоче місце	Аналітик
Технічні засоби (кількість)	**Назва та характеристики**
Монітор (4 шт.)	HP 22Xi/21,5"/1920x1080px/IPS
Комп’ютер (4 шт.)	HP Probook 4530s / 15.6" (1366x768) LED / Intel Core i5-2410M (2.30 ГГц) / RAM 8 ГБ / HDD 500 ГБ
Кондиціонер (1 шт.)	DEKKER DSH105R/G / 26м^2 / 2,65кВт-2,9кВт / 25 х 74,5 х 19,5 см / 9 кг
Підлоговий кулер (1 шт.)	CRYSTAL YLR3-5V208
Світильники загального призначення (3 шт.)	Світильник растровий вмонтований 4x18W
Світильники місцевого призначення (4 шт.)	DeLux Décor TF-05 / 1 х 40Вт

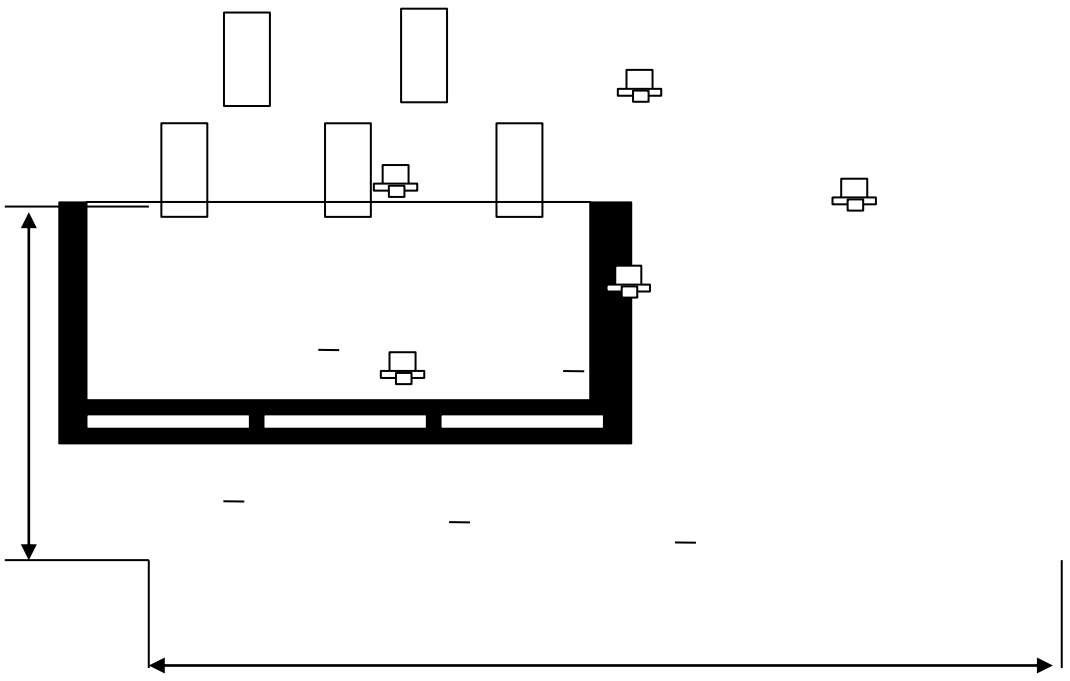

Рисунок 17 Схема приміщення

Площа S', виділена для одного робочого місця з персональною ЕОМ, повинна складати не менше 6 кв. м, а об'єм V' – не менше 20 куб. м. Розрахуємо фактичні значення цих показників, розділивши загальну площу та об'єм приміщення на кількість працюючих:

$$S' = \frac{S}{N} = \frac{6{,}6 * 6}{4} = 9{,}9 \left(\frac{\text{м}^2}{\text{люд.}}\right) (4.1)$$

$$V' = \frac{V}{N} = \frac{6{,}6 * 6 * 2{,}7}{4} = 26{,}73 \left(\frac{\text{м}^3}{\text{люд.}}\right) (4.2)$$

Отже, виходячи з отриманих результатів за характеристиками площі та об'єму приміщення відповідає нормам.

Таблиця 3.2 Характеристики робочого місця

№	Найменування параметра	Значення	
		Фактичне	**Нормативне**
1.	Висота робочої поверхні, мм	750	680 – 800
2.	Ширина робочої поверхні, мм	1400	не менше 600
3.	Глибина робочої поверхні, мм	750	не менше 600
4.	Висота простору для ніг, мм	730	не менше 600
5.	Ширина простору для ніг, мм	790	не менше 500
6.	Глибина простору для ніг, мм	700	не менше 450
7.	Висота поверхні сидіння, мм	490	400 – 500
8.	Ширина сидіння, мм	500	не менше 400
9.	Глибина сидіння, мм	470	не менше 400
10.	Висота опорної поверхні спинки, мм	650	не менше 300
11.	Ширина поверхні спинки, мм	480	не менше 380
12.	Довжина підокітників, мм	310	не менше 250
13.	Ширина підокітників, мм	55	50 – 70
14.	Відстань від очей до екрану, мм	620	600 – 700

Виходячи з заданих параметрів, можна зробити висновок, що розміри робочого місця програміста відповідають встановленим нормам.

3.2 Аналіз потенційно небезпечних і шкідливих виробничих факторів на робочому місці

Варто зазначити, що під час роботи на працівника діє ряд небезпечних і шкідливих чинників

Таблиця 3.3 Шкідливі чинники на робочому місці

Фізичні	Психофізіологічні
Підвищений рівень шуму	Розумове перенапруження
Підвищений рівень статичної електрики	Перенапруження аналізаторів
Підвищений рівень електромагнітного випромінювання	Монотонність праці
Недостатній рівень освітленості	
Неоптимальний мікроклімат	

3.2.1 Мікроклімат

Робота при створенні програмного продукту виконувалася сидячи без фізичних зусиль, а тому відноситься до категорії легка Іа. У таблиці 3.5 наведені нормативні та фактичні показники мікроклімату.

Таблиця 3.4 Аналіз шкідливих факторів, пов’язаних з мікрокліматом

№	Шкідливий фактор	Наслідки
1	Відхилення t від оптимальних параметрів	Відсутність теплового комфорту, тимчасове погіршення самопочуття і зниження працездатності, хвороби
2	Відхилення вологості повітря від оптимальних параметрів	Тимчасове погіршення самопочуття і зниження працездатності, хвороби, роздратованість
3	Відхилення V руху повітря від оптимальних параметрів	Тимчасове погіршення самопочуття і зниження працездатності, хвороби

Заходи, що забезпечують запобігання порушення встановлених мікрокліматичних норм наведені в таблиці 3.7.

Таблиця 3.5 мікроклімат в теплий період року

Параметр мікроклімату			
Найменування	Значення		
	Фактичне		Оптимальне
t, ^{o}C	21	21 – 23	18 –27
w, %	55	60 – 40	до 75
V, м/с	0,2	0,3	0,4 – 0,2

Таблиця 3.6 мікроклімат в холодний період року

Параметр мікроклімату			
Найменування	Значення		
	Фактичне		Оптимальне
t, ^{o}C	18	21 – 23	18 –27
w, %	70	60 – 40	до 75
V, м/с	0,4	0,3	0,4 – 0,2

Таблиця 3.7 Запобіжні заходи в теплий та холодний періоди року

№	**Технічні**	**Організаційні**	**ЗІЗ**
1	Контроль параметрів за допомогою термометра La Crosse WS8005; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання)	Перерви в роботі, з метою провітрювання кімнати; вологе прибирання на робочих місцях	відсутні
2	Контроль параметрів за допомогою психрометра Т-04; використання зволожувача повітря ZELMER AH1500	Перерви в роботі, з метою провітрювання кімнати; вологе прибирання на робочих місцях	відсутні
3	Контроль параметрів за допомогою анемометра Extech AN100; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання).	відсутні	відсутні

3.2.2 Недостатня освітленість і підвищена яскравість світла

Згідно ДБН В.2.5-28-2006 ця робота відноситься до розряду зорових робіт. Передбачається використання природного, штучного і змішаного освітлення. В табл. 3.8 наведені шкідливі фактори порушень норм яскравості світла.

Таблиця 3.8 Шкідливі фактори порушень норм яскравості світла

№	**Шкідливий фактор**	**Наслідки**
1	Недостатня освітленість робочої зони	Погіршення зору і самопочуття, втомлюваність, підвищення ризику здійснення помилки
2	Підвищена яскравість світла	

Таблиця 3.9 Параметри освітлення

Найменування	**Значення**		
	Фактичне	**Оптимальне**	
При змішаному освітленні		450	400
При загальному освітленні		300	300
Коефіцієнт природного освітлення		1,23	1,2

Таблиця. 3.10 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою люксметра DT-1308; використання нових світильників загального призначення ELSTEAD FINSBURY PARK FP6 POL NICKEL; урахування природного освітлення кімнати	Встановлення мінімального рівня освітлення; чищення скла вікон та світильників; заміна ламп, що перегоріли	Додаткове освітлення на робочих місцях (світильники DeLux Décor TF-05); окуляри для роботи з комп'ютером.
2	Контроль параметрів за допомогою люксметра DT-1308;використання регульованих пристроїв для відкривання вікон, а також жалюзі; використання світильників нового типу	Відсутні	Окуляри для роботи з комп'ютером.

3.2.3 Шум та вібрація

Джерелами шуму в приміщенні є вентилятор ноутбуку та кондиціонер. Звук, що створюється вентилятором та кондиціонером, можна класифікувати як постійний.

Таблиця 3.11 Шум і вібрація

Шкідливий фактор	Наслідки
Підвищений рівень шуму	Погіршення слуху, зниження продуктивності роботи, підвищення ймовірності виникнення помилки.
Вібрації на робочому місці	Роздратування, погіршення самопочуття, зниження працездатності

Таблиця 3.12 Джерела шуму

Джерело шуму	Фактичний рівень шуму, дБ	Оптимальний рівень шуму, дБ	Час роботи, год.
Кондиціонер DEKKER DSH105R/G	22	< 50	8
Кулер комп’ютеру HP Probook 4530s	20		8

Таблиця 3.13 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою приладу для виміру шуму DT-8852; якісний монтаж окремих вузлів комп'ютера	Проведення планового попереджувального ремонту (чищення від пилу і інших забруднень)	Відсутні
2	Контроль параметрів за допомогою приладу для виміру вібрацій TV260; встановлення спеціальної підставки під ноутбук	Організаційне вирішення: проведення планового попереджувального ремонту (чищення від пилу і інших забруднень)	Відсутні

3.2.4 Небезпека враження людини електричним струмом

ЕОМ є однофазним споживачем електроенергії, що живиться від змінного струму 220В від мережі із заземленою нейтраллю. IBM PC відноситься до електроустановок до 1 000В закритого виконання, всі струмопровідні частини знаходяться в кожухах. За способом захисту людини від ураження електричним струмом, ЕОМ і периферійна техніка повинні відповідати 1 класу захисту.

В табл. 3.12 наведені шкідливі фактори випромінювання при роботі з обчислювальною технікою.

Таблиця 3.14 Небезпечні фактори ураження людини електричним струмом

№	Шкідливий фактор	Наслідки	Заходи
1	Небезпечний рівень напруги струмопровідних частин обчислювальної та побутової техніки	Зростання ризику ураження електричним струмом	Релейний захист струму дотику, захисні заземлюючі корпуси . Попереджувальні знаки про рівень напруги.

Таблиця 3.15 Параметри електропостачання на робочому місці

	Напруга, В	Частота, Гц	Тип розетки/вилки	Тип фази
Фактичне	220	50	F	Однафазна, трипровідна
Оптимальне	220	50	C,F	Однофазна, трипровідна

Таблиця 3.16 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Релейний захист струму дотику, захисні заземлюючі корпуси .	Створення плану короткострокових відпочинків. Проведення робіт з електричним обладнанням лише проінструктованим персоналом	відсутні

3.2.5 Пожежна безпека

Запобігання пожежі досягається виключенням утворення горючого середовища і джерел загорянь.

Таблиця 3.17 Шкідливі фактори, пов’язані з пожежною безпекою

№	Шкідливий фактор	Наслідок
1	Коротке замикання, електротравми, пожежі, летальні наслідки	Коротке замикання, електротравми, пожежі, летальні наслідки
	Коротке замикання	Електротравми, пожежі, летальні наслідки
	Порушення протипожежного режиму	Електротравми, пожежі, летальні наслідки

В цьому приміщенні можливі пожежі таких класів: А – горіння твердих речовин , Е – горіння електроустановок під напругою. Для забезпечення цих категорій застосовуються заходи, що вказані в таблиці 4.18.

Таблиця 3.18 Запобіжні заходи

№	Технічні	Організаційні	ЗІЗ
1	Контроль параметрів за допомогою термометра La Crosse WS8005; використання кондиціонеру DEKKER DSH105R/G (для кондиціонування і провітрювання)	Розвантаження електровузлів після виконання роботи; ознайомлення з інструкціями по використанню електроприладів;	Відсутні
2	Наявність вогнегасника порошкового типу ОП-5 та автоматичної системи «ГАРАНТ-Р» (ПО-2), узгоджений план евакуації	Ознайомлення з інструкціями по використанню протипожежних засобів; узгоджений план евакуації	відсутні
3	Наявність вогнегасника порошкового типу ОП-5 та автоматичної системи «ГАРАНТ-Р» (ПО-2), узгоджений план евакуації	Ознайомлення з інструкціями по використанню протипожежних засобів; узгоджений план евакуації	відсутні

3.3 Висновки

Аналіз умов праці в розглянутому робочому приміщенні показав, що умови праці з ПЕОМ відповідають нормативам.

В зв’язку з тим, що даний вид праці є шкідливим через напруженість, розробнику ПЗ рекомендується роботи перерви в роботі із виконанням невеликих гімнастичних вправ та вправ для очей.

ВИСНОВКИ

В даній роботі було розглянуто поняття обробки великий данних в режимі реального часв, проблеми, які виникають при одночасному надхдженні великого об’єму данних

Була розроблена програмна реалізація методів Візуальної побудови данних з подальшим розширенням на аналіз результатів за допомогою бібліотек обробки данних Apache Kafka, Apache Hbase, Apache Storm, на мовах програмування Java, Python, JavaScript методу відображення за допомогою пакета прикладних програм nvd3.

Після підрахунку кількості операцій виявилося, що метод розроблена система відповідає вимогам стабільності масштабовності та описана пропускна здатність й розраховані вимоги до апаратних частин які повинні для підтримки очікуваної пропускної здатності

Реалізовані алгоритми були протестовані на різних послідовностях, данних за отриманими результатами алгоритми був запропонований результать прийнаття рішення. Оскільки данні були згенеровані й можна було точно визначить й порівнть запропоноване рішення(прогноз) в данний момент з тим що дійсно відбувається далі. Тому можна зробити висновок, що система дійсно пропонує оптимальні рішення.

В подальшому слід зосередити роботу на покращенні точності прогнозованих результатів за допомогою існуючих алгоритмів аналізу данних. Отримана програмна реалізація може ібути використана в системах прогнозу прибутку при випуску товарів на онлайн ринки, в незалежності від самих джерел(ринків) й типів товарів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Олдендерфер М. С., Блэшфилд Р. К. Кластерный анализ / Факторный, дискриминантный и кластерный анализ: пер. с англ.; Под. ред. И. С. Енюкова. — М.: Финансы и статистика, 1989. — 215 с

2. Г. Пятецкий-Шапиро, Data Mining и перегрузка информацией // Вступительная статья к книге: Анализ данных и процессов / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. 3-е изд. перераб. и доп. СПб.: БХВ-Петербург, 2009. 512 с

3. Wolfram Language Documentation Center Introduction to Manipulate – Режим доступу:

https://storm.apache.org/documentation/Tutorial.html - Дата доступу – 03.06.1015.

4. Бокс Дж., Дженкинс Г. Анализ временных рядов, прогноз и управление: Пер. с англ. // Под ред. В.Ф. Писаренко. – М.: Мир, 1974, кн. 1. – 406 с.

5. Дрейпер Н., Смит Г. Прикладной регрессионный анализ. Множественная регрессия — 3-е изд. — М.: «Диалектика», 2007. — С. 912. — ISBN 0-471-17082-8.

6. Фёрстер Э., Рёнц Б. Методы корреляционного и регрессионного анализа — М.: Финансы и статистика, 1981. — 302 с.

7. Захаров С. И., Холмская А. Г. Повышение эффективности обработки сигналов вибрации и шума при испытаниях механизмов // Вестник машиностроения : журнал. — М.: Машиностроение, 2001. — № 10. — С. 31—32. — ISSN 0042-4633.

8. Радченко Станислав Григорьевич,. Устойчивые методы оценивания статистических моделей: Монография. — К.: ПП «Санспарель», 2005. — С. 504. — ISBN 966-96574-0-7, УДК: 519.237.5:515.126.2, ББК 22.172+22.152.

9. Радченко Станислав Григорьевич,. Методология регрессионного анализа: Монография. — К.: «Корнийчук», 2011. — С. 376. — ISBN 978-966-7599-72-0.

10. Preimesberger, Chris Hadoop, Yahoo, 'Big Data' Brighten BI Future (англ.). EWeek (15 August 2011). Проверено 12 ноября 2011.

11. Черняк, Леонид Большие Данные — новая теория и практика (рус.) // Открытые системы. СУБД. — М.: Открытые системы, 2011. — № 10. — ISSN 1028-7493.

12. Типові норми належності вогнегасників (затверджено наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 2 квітня 2004 р. N 151)

13. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст] / К., 2000.- 16 с.

14. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. НАПБ Б.03.002-2007. (затверджено наказом МНС України від 03.12.2007 № 833)

15. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу (затверджено наказом МОЗ України від 12.08.2014р № 248

16. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. № 7).

17. Правила охорони праці під час експлуатації електронно-обчислювальних машин. НПАОП 0.00-1.28-10 (затверджено наказом Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010р. № 65).

18. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу (затверджено наказом МОЗ України від 08.04.2014 № 248)