**Національний технічний університет України**

**«Київський політехнічний інститут**

**імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ ІНФОРМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Тарасенко В.П.

(підпис) (ініціали, прізвище)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

**Дипломний проект**

**на здобуття ступеня бакалавра**

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: Сховище даних для розподілених сенсорних мереж

Виконав: студент IV курсу, групи КВ-31

Анастасьєв Дмитро Вадимович \_\_\_\_\_\_\_\_\_

(підпис)

Керівник к.т.н, доцент кафедри СПіСКС Замятін Д.С. \_\_\_\_\_\_\_\_

(підпис)

Нормоконтроль к.т.н., доцент каф. СПіСКС Клятченко Я.М.

(підпис)

Рецензент д.т.н., професор, професор каф. ОТ Кулаков Ю.О.

(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2017 року

**Національний технічний університет України**

**«Київський політехнічний інститут**

**імені Ігоря Сікорського»**

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки **6.050102 «Комп'ютерна інженерія»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Тарасенко В.П.

(підпис) (ініціали, прізвище)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

**ЗАВДАННЯ**

**на дипломний проект студенту**

Анастасьєву Дмитру Вадимовичу

1. Тема проекту Сховище даних для розподілених сенсорних мереж, керівник проекту к.т.н, доцент кафедри СПіСКС Замятін Д.С., затверджені наказом по університету від «19» травня 2017 р. №1574-С

2. Термін подання студентом проекту 20 червня 2017 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки

- аналіз існуючих рішень збереження даних з сенсорних мереж;

- огляд переваг та недоліків баз даних часових рядів;

- пояснення можливості використання баз даних часових рядів для збереження даних з розподілених сенсорних мереж.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

- UML діаграма емулятора сенсорної мережі. Схема структурна.

- Емуляція польоту літального об’єкту. Схема алгоритму.

- Загальна схема емулятора сенсорної мережі. Схема структурна.

- Алгоритм тестування навантаженості бази даних часових рядів. Схема алгоритму.

- Презентація за темою роботи.

6. Консультанти розділів проекту[[1]](#footnote-1)\*

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада  консультанта | Підпис, дата | |
| завдання  видав | завдання прийняв |
| Нормоконтроль | Клятченко Я. М. |  |  |
|  |  |  |  |
|  |  |  |  |

7. Дата видачі завдання

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання  дипломного проекту | Термін виконання  етапів проекту | Примітка |
| 1. | Вивчення літератури за тематикою проекту |  |  |
| 2. | Розроблення та узгодження технічного завдання |  |  |
| 3. | Аналіз існуючих рішень |  |  |
| 4. | Підготовка матеріалів першого розділу дипломного проекту |  |  |
| 5. | Підготовка матеріалів другого розділу дипломного проекту |  |  |
| 6. | Підготовка графічної частини дипломного проекту |  |  |
| 7. | Оформлення документації дипломного проекту |  |  |
| 8. | Попередній огляд матеріалів диплому на кафедрі |  |  |
|  |  |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Анастасьєв

(підпис)

Керівник проекту \_\_\_\_\_\_\_\_\_\_\_\_ Д.С. Замятін

(підпис)

# РОЗДІЛ 3

# РОЗРОБКА ЕМУЛЯТОРА СЕНСОРНОЇ МЕРЕЖІ

Для створення сенсорних даних було вирішено розробити додаток, який буде емулювати поведінку сенсорної мережі. За приклад сенсорної мережі взято множину територіально розподілених сенсорів, які працюють на основі алгоритму акустичної локалізації та призначені для виявлення повітряних безпілотних об’єктів. На даному етапі нам не важливо як саме працюють сенсори, тому самі алгоритми виявлення та ідентифікування цілей не розглядаються. За замовчуванням кожен сенсор в мережі може встановлювати координати знаходження повітряного об’єкта в радіусі його дії, та виконувати ідентифікацію об’єкта (відрізняти одні цілі від інших).

Ділянка на якій використовується мережа має круглу форму та поділена на сектори. Сенсори які знаходяться в певному секторі відсилають отримані дані до свого територіально відокремленого серверу. На даному етапі виникає проблема об’єднання даних про місце знаходження певного об’єкта у випадку коли частина інформації була зафіксована сенсорами з одного сектору, а інша частина з іншого, тобто дані знаходяться на різних серверах. Цю проблему вирішує обрана база даних часових рядів InfluxDB, в якій узгодження даних в кластері відбувається за допомогою алгоритму Raft. Даний алгоритм дозволяє узгоджувати дані так, що на кожному вузлі кластеру будуть доступні всі дані в базі, незалежно до якого вузла вони були відправлені.

Додаток був розроблений за допомогою мови програмування Python 2.7 в інтегрованому середовищі розробки PyCharm.

Для генерування траєкторії польоту безпілотних повітряних цілей була використана бібліотека, розроблена мовою python для генерації траєкторій польоту квадрокоптерів. Вхідними параметрами для створення траекторій є координати початкової і кінцевої точки, діапазон швидкості та висоти польоту, коефіцієнт зміщення від найкоротшої траекторії. Різні комбінації вхідних параметрів дозволяють створювати різні траекторії. Тому якщо використати генерацію випадкових чисел для встановлення деяких з вхідних параметрів, можна створювати випадкові траекторії. Початкова та кінцева точки траєкторії обираються на краю ділянки на якій буде працювати сенсорна мережа. Розташування цих точок також обмежується певним кутовим діапазоном (рис. 3.1)

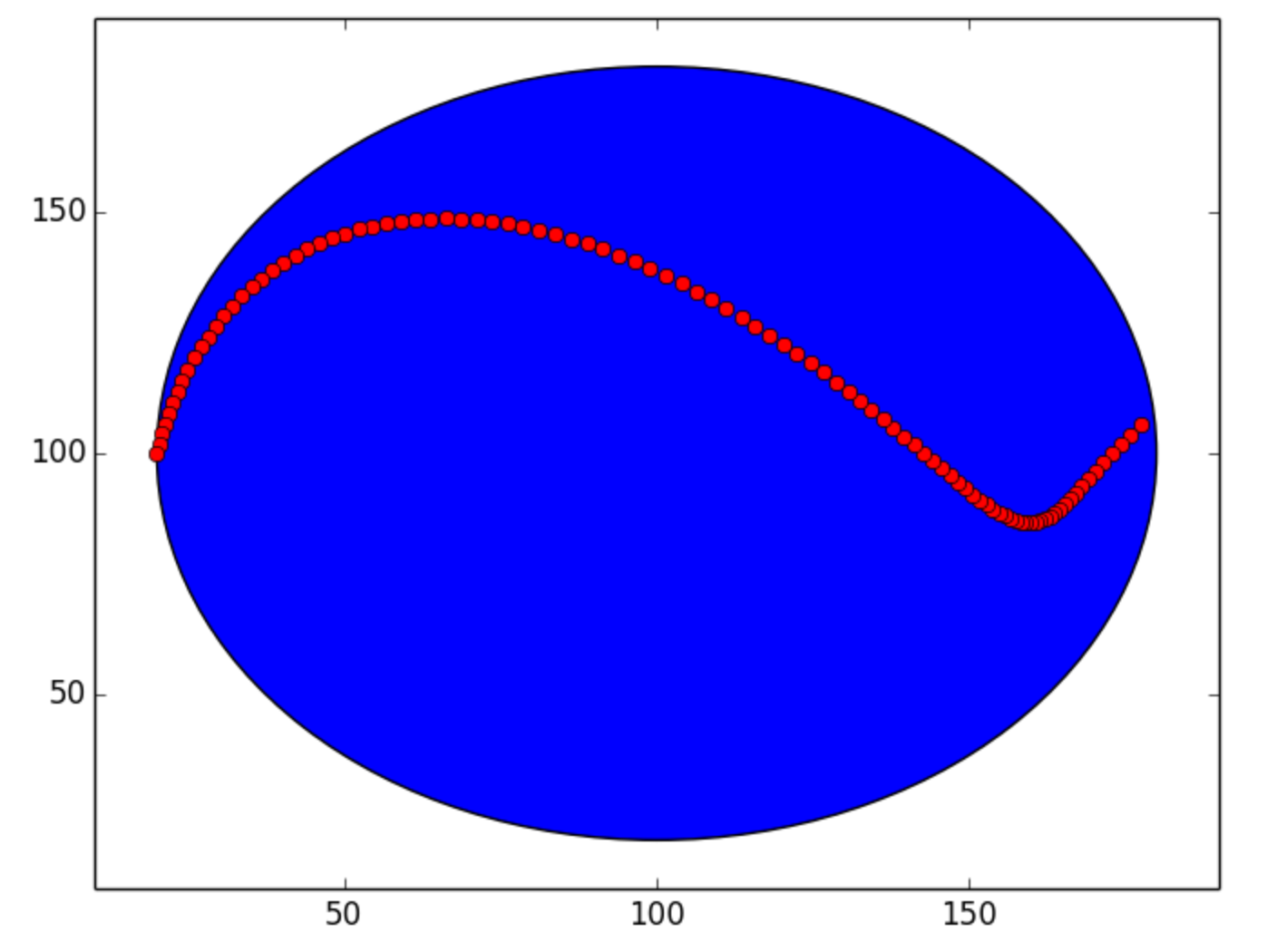


Рис. 3.1 Приклад траєкторії польоту безпілотного літального об’єкту

Створення сенсорів відбувається в певному секторі, тобто за програмною архітектурою сенсори саме належать до певного сектору, а не всій ділянці в цілому. Вхідні параметри для кожного серсору, це координати розташування на ділянці та радіус дії. Кількість сенсорів в кожному секторі задається в конфігураційному файлі а розташування в секторі генерується випадковим чином (рис. 3.2).

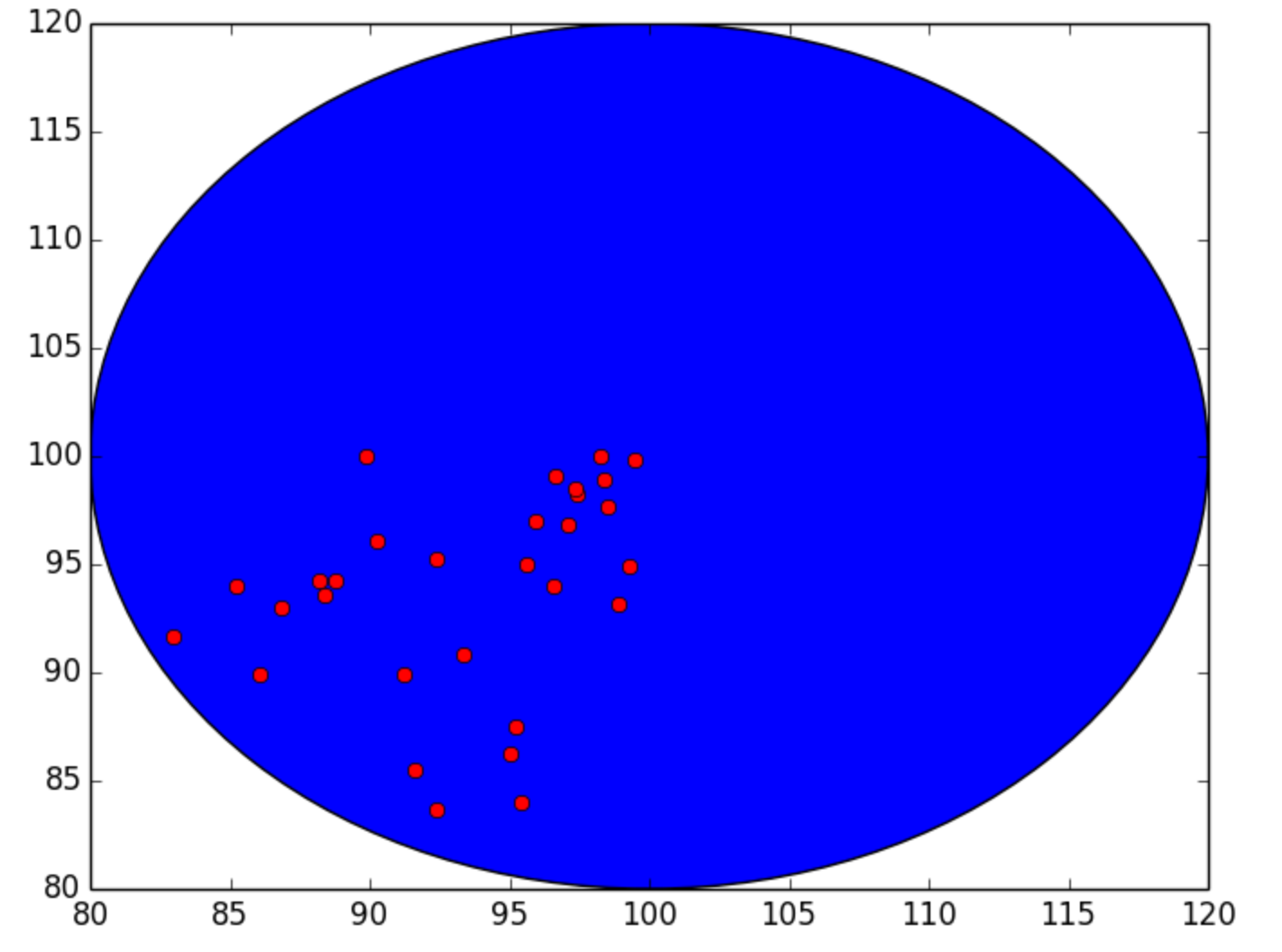


Рис. 3.2 Приклад генерації 30 сенсорів в секторі ділянки

Параметри створення ділянки для роботи сенсорної мережі, а саме центральна точка розташування ділянки, радіус, кількість секторів та сенсорів, тощо, задаються в конфігураційному файлі.

Приклад частини конфігураційного файлу для додатку, який емулює роботу сенсорної мережі показаний на рис. 3.3

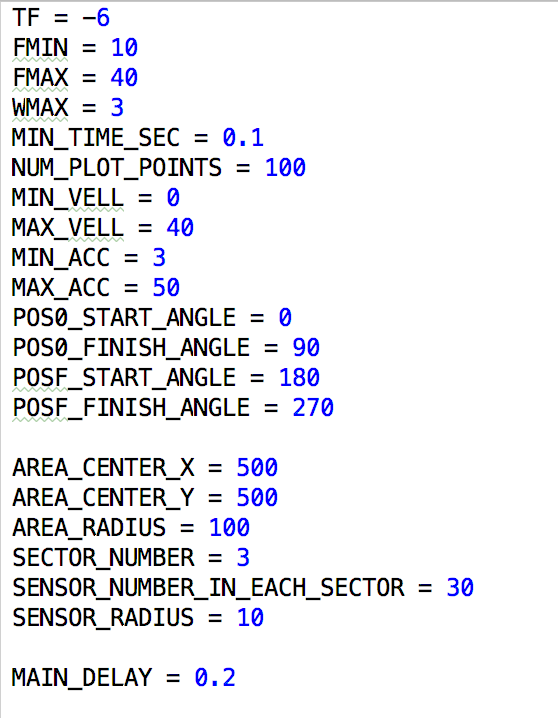


Рис. 3.3 Приклад вмісту конфігураційного файлу

Розберемо важливі параметри. NUM\_PLOT\_POINTS – кількість точок в траєкторії літального об’єкта, TF – коефіцієнт відхилення від найкоротшої траєкторії, MIN\_VELL та MAX\_VELL – це мінімальне та максимальне значення висоти польоту, MIN\_ACC та MAX\_ACC – мінімальне та максимальне значення прискорення польоту. POS0\_START\_ANGLE та POS0\_FINISH\_ANGLE – використовується для задання діапазону кута для генерації стартової точки траєкторії на кінці обраної ділянки. Відповідно POSF\_START\_ANGLE та POSF\_FINISH\_ANGLE використовується для знаходження кінцевої точки траєкторії. Параметри ділянки: AREA\_CENTER\_X, AREA\_CENTER\_Y та AREA\_RADIUS використовуються для задання координат положення ділянки, а також її радіусу. SECTOR\_NUMBER – кількість секторів на які буде розбито ділянку, а SENSOR\_NUMBER\_IN\_EACH\_SECTOR – кількість сенсорів, яка буде створена в кожному секторі. SENSOR\_RADIUS – радіус дії кожного сенсора, MAIN\_DELAY – час затримки між ітераціями алгоритму.

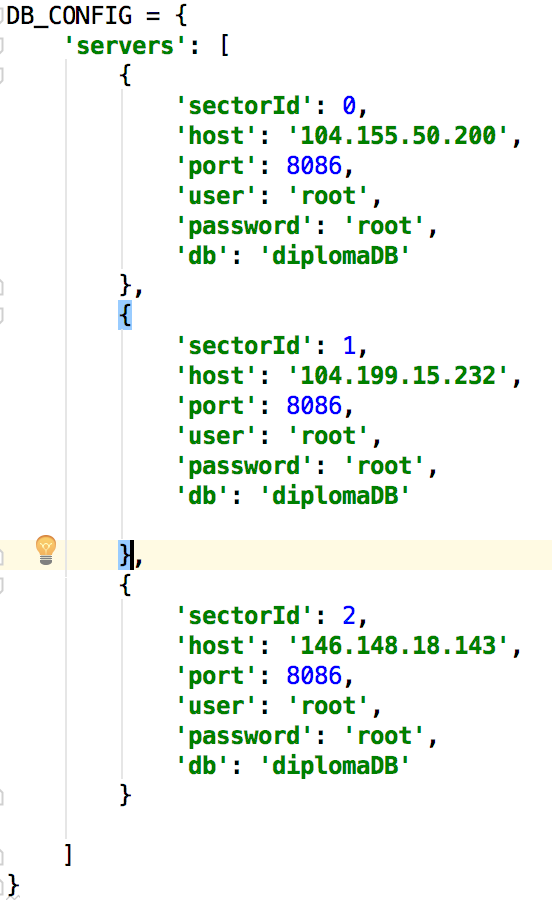
 Підключення до бази даних відбувається за допомогою бібліотеки *influxdb*, а саме класу *InfluxDBClient.* Розроблений клас DBInfrastructure містить масив підключень до різних серверів, під час запису даних береться до уваги сектор з якого відправляються дані і в залежності від нього обирається відповідний сервер. Конфігурація підключень до баз даних відбувається за допомогою конфігураційного файлу і записується у форматі JSON. Приклад такої конфігурації зображений на рис 3.4.

Рис. 3.4 Приклад вмісту конфігураційного файлу для налаштування підключення до серверів баз даних

У цій конфігурації зазначається перелік серверів з якими буде працювати система. Для підключення для кожного серверу потрібно зазначити: ip-адресу або host-адресу, порт, параметри користувача бази даних (ім’я та пароль), вказати назву бази даних. Також для кожного серверу необхідно вказати ідентифікатор сектору з яким працює даний сервер.

Опишемо алгоритм роботи емулятора сенсорної мережі. Для початку роботи потрібно створити основні об’єкти, якими ми будемо маніпулювати в процесі генерації даних. Першим із них є ділянка в просторі на території якої буде встановлена сенсорна мережа. Така ділянка поділена на сектори всередині кожного з яких створюється певна кількість сенсорів. Тому для створення ділянки потрібно використати наступні параметри: координати центру ділянки, радіус ділянки, кількість секторів, кількість сенсорів в кожному секторі, радіус дії сенсору. Під час створення ділянки, створяться сектори, які займають рівну площу, а також всі сенсори, які випадковим чином розташуються всередині секторів. Другим важливим елементом системи є літальний об’єкт, який здійснює польоти по ділянці за певною траєкторією. Для створення такого об’єкту потрібно створити траєкторію, в свою чергу для створення траєкторії потрібно створити генератор траєкторій. Отже після створення ділянки створюється генератор траєкторій за допомогою дій якого в майбутньому будуть створюватися літальні об’єкти. Далі для роботи з кластером бази даних створюється об’єкт який містить у собі підключення до всіх серверів, інформацію про деталі підключення цей об’єкт бере із конфігураційного файлу.

Після цього починається циклічний процес створення літальних об’єктів, проходження ними траєкторій, фіксування сенсорами поточних координат об’єктів та запис в базу даних цієї інформації.

Розберемо цю частину алгоритму детальніше. Спочатку генератор траєкторій створює траєкторію, далі використовуючи цю траєкторію створюється літальний об’єкт, який має ідентифікаційний номер та ім’я. Далі визначається поточна позиція літального об’єкта (координати по осі x, y та z), далі використавши координати x та y можна дізнатися в якому секторі ділянки знаходиться об’єкт. Далі сектор повертає всі сенсори, які зафіксували літальний об’єкт в даному положенні. За допомогою об’єкта роботи з базою даних, інформація з кожного сенсору записується у сховище. Процес повторюється доки не закінчиться траєкторія. Далі створюється новий літальний об’єкт з новою траєкторією і починається наступна ітерація алгоритму.

Структура даних, які записуються в сховище має наступний вигляд. Вимірювання в яке записуються дані має назву *flight*, кожен запис має теги: *objectId* – ідентифікатор літального об’єкта, *objectName* – ім’я літального об’єкта та sector – ідентифікатор сектору з якого було зроблене вимірювання. Приклад записів в InfluxDB зображений на рис. 3.5

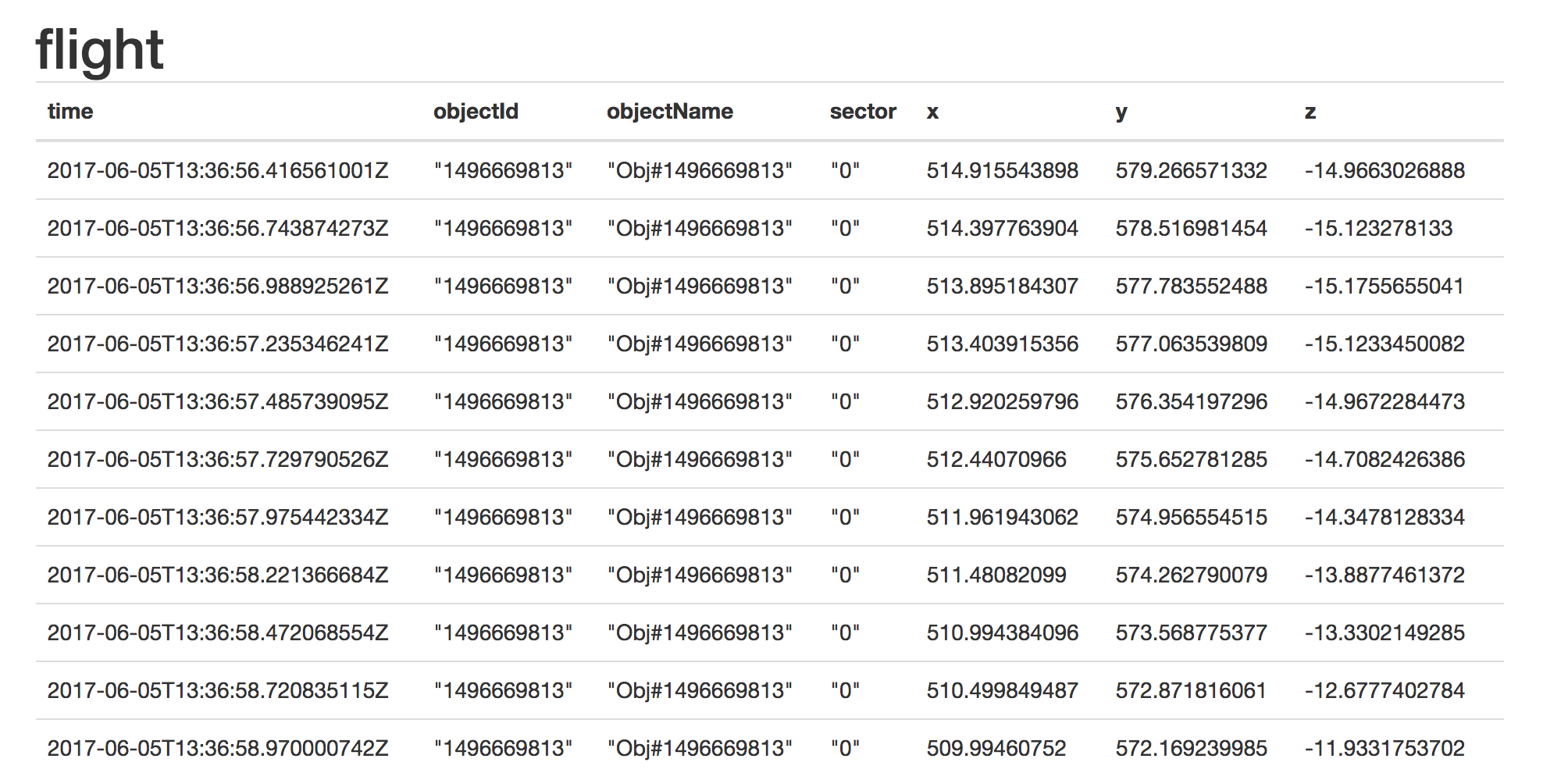


Рис. 3.5 Приклад записів в InfluxDB

Для зручнішого подальшого аналізу даних та візуального представлення існють рішення, які дозволяють візуалізувати дані часових рядів у вигляді різноманітних графіків. Для відображення даних отриманих з сенсорної мережі було вирішено використовувати візуалізатор Grafana. Даний інструмент дозволяє підключатся до великої кількості сховищ одних з яких є InfluxDB, надає можливість надсилати різні запити до баз даних і на основі їх результатів будувати графіки, надає зручний графічний веб-інтерфейс для керування стилями відображення графіків. Grafana має можливість підключення плагінів, які розширюють фінкціонал цього додатку.

Після встановлення Grafana для початку роботи потрібно виконати команду: *brew services start grafana* (використовується операційна система OS X). Після цього на порті 3000 піднімається веб-сервер.

Grafana надає зручний інтерфейс для підключення потрібного джерела даних рис. 3.6

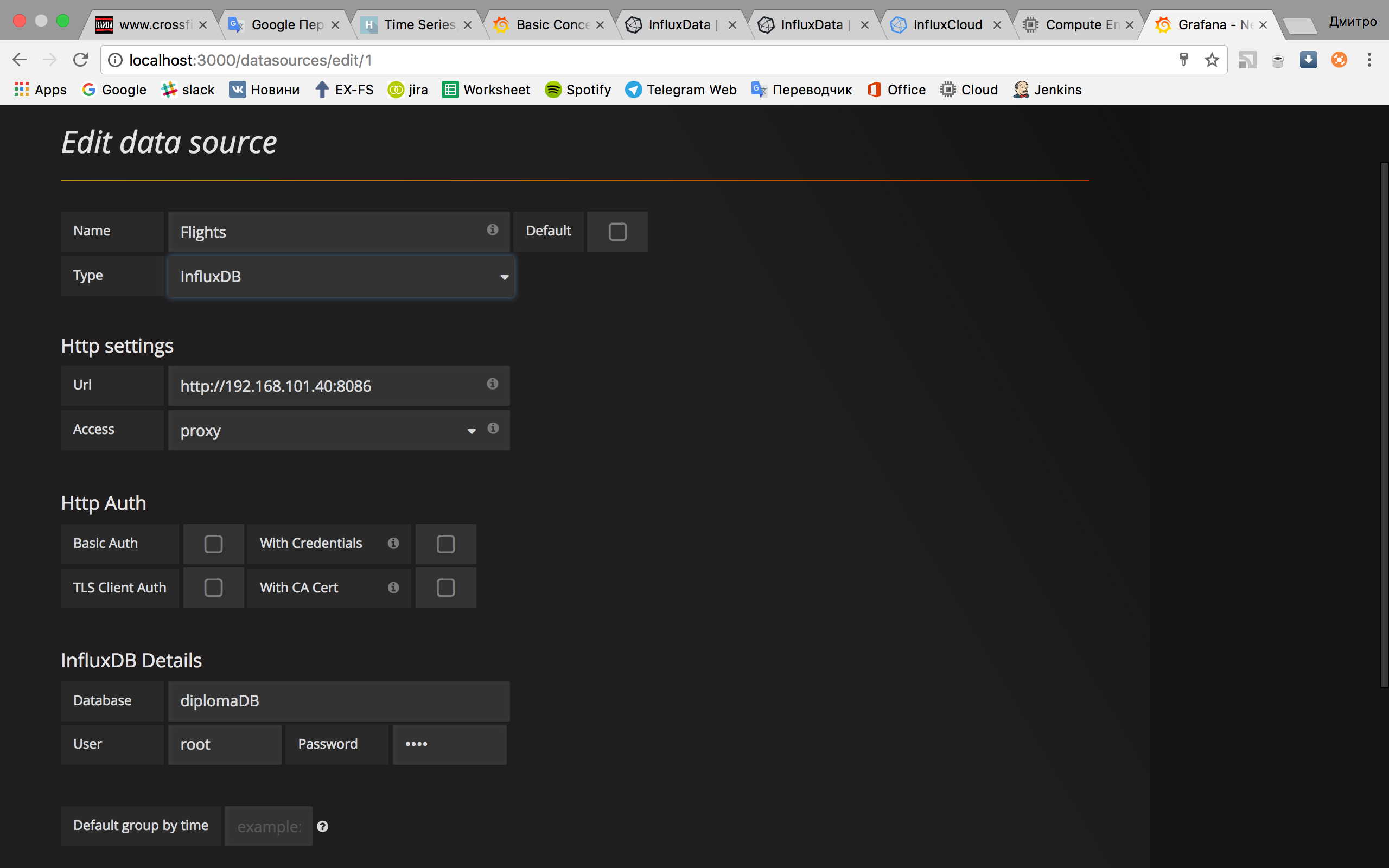


Рис. 3.6. Інтерфейс підключення сховища даних в Grafana

Для демонстрації процесу генерації та збереження даних, а також відображення їх у реальному часі було запущено додаток емуляції роботи сенсорної мережі, а також налаштовано візуалізатор даних часових рядів Grafana. Для прикладу, вибірка даних включае в себе дані про координати по осі X та Y для кожного окремого об’єкта, для цього використовуємо запит:

*SELECT “x”, “y” from “flight” WHERE $timeFilter GROUP BY “objectId”*, де *$timeFilter* проміжок часу в якому будуть показані дані (рис. 3.7).

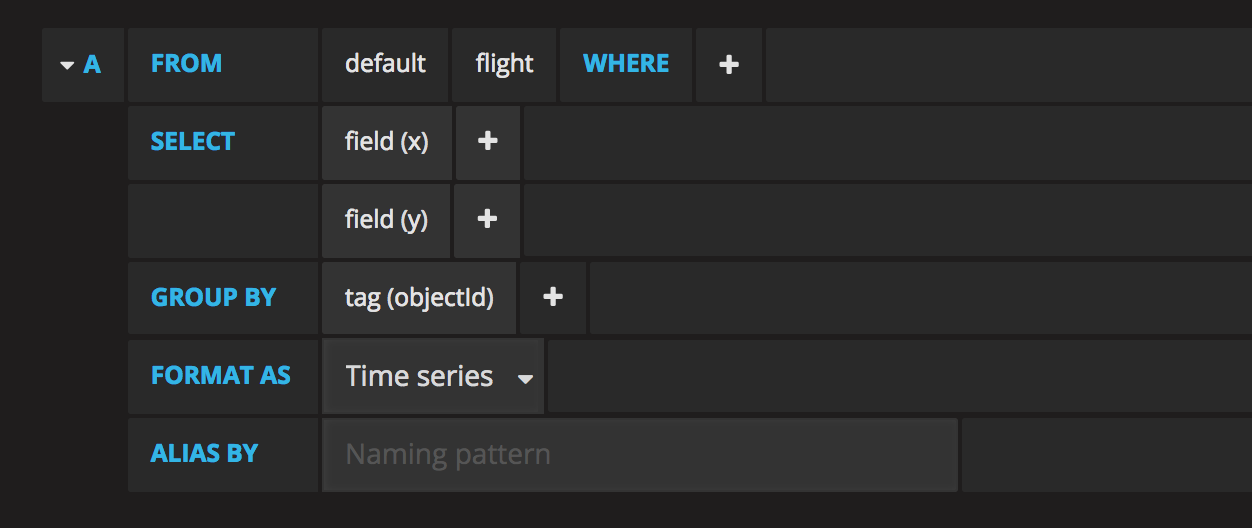


Рис. 3.7 Приклад вибірки даних для візуалізації

В результаті отримуємо графіки наступного вигляду (рис. 3.8). Ми отримали ряд пар кривих, які описують зміни положення цілей за осями Х та У в залежності від часу, розташування ділянки обрано так, що координати по обох осях мають значення в однакових діапазонах, це дозволяє відобразити їх на одному графіку. Ми можемо спостерігати за зміною даних в реальному часі, що і було одним з основних вимог у даній роботі

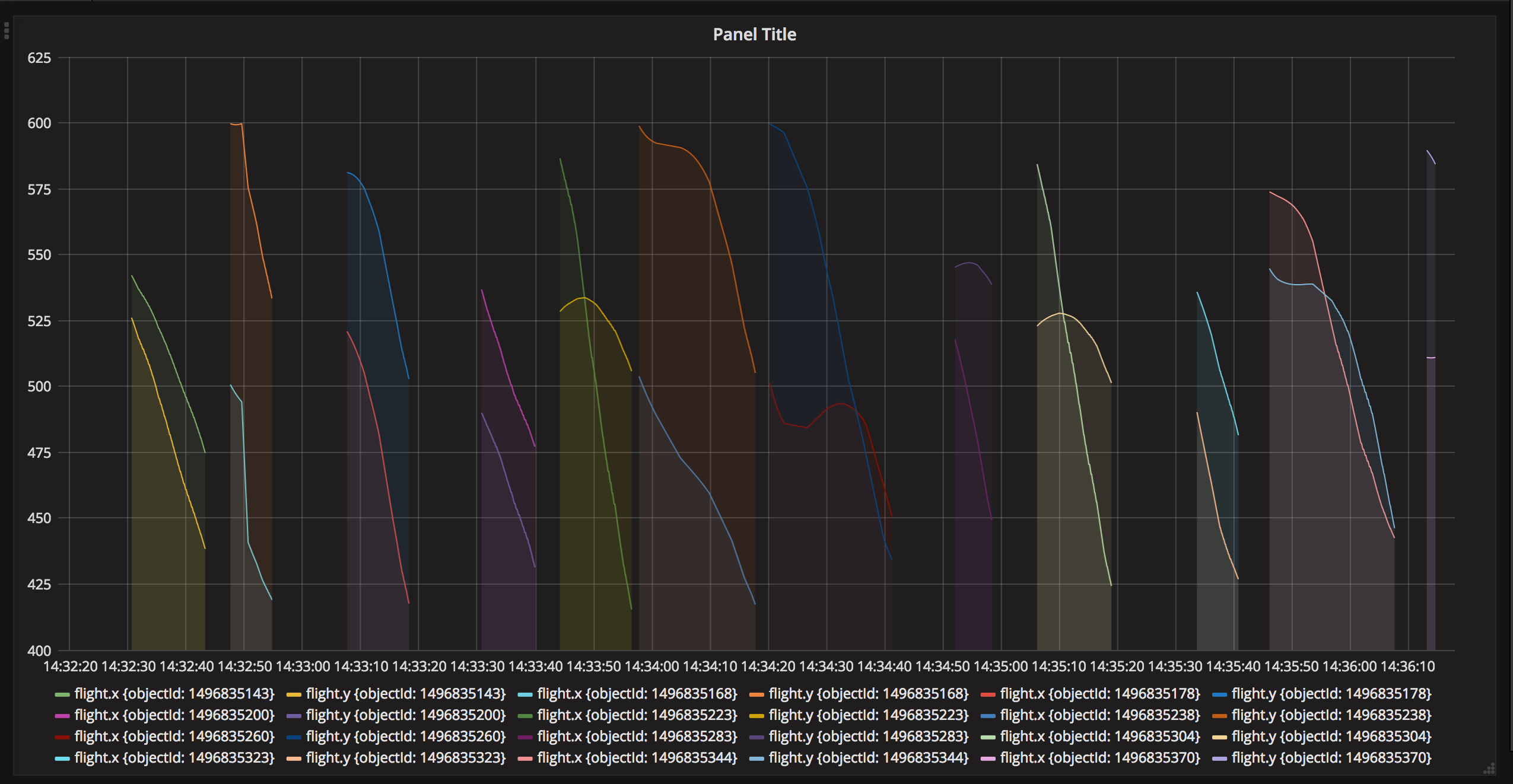


Рис. 3.8. Графіки залежності значень координат по осям X та У від часу для об’єктів зафіксованих сенсорною мережою в певному проміжку часу

# РОЗДІЛ 4

# ТЕСТУВАННЯ РОБОТИ КЛАСТЕРУ БАЗИ ДАНИХ ЧАСОВИХ РЯДІВ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

# 4.1 ТЕСТУВАННЯ НАВАНТАЖЕНОСТІ БАЗИ ДАНИХ ЧАСОВИХ РЯДІВ

Для тестування роботи бази даних часових рядів використовувалась база даних описана в розділі 2, а саме InfluxDB, а також програма емуляції роботи сенсорної мережі описана в розділі 3.

Було проведено ряд тестів, за результатами яких можна проаналізувати можливість роботи бази даних за різних умов. Ціллю даного тестування є перевірка можливостей бази під час роботи з великою навантаженістю, а також перевірка виконання вимог зазначених у технічному завданні.

Під час тестування була перевірена робота бази з різною кількістю серверів у кластері та з різною навантаженістю. Регулювання навантаженості роботи було виконане за допомогою зміни швидкості генерування даних сенсорною мережею. Зміна швидкості генерування даних досягається за домогою зміни деяких параметрів роботи додатку емуляції сенсорної мережі. Такими параметрами є: кількість сенсорів в кожному секторі ділянки на якій працює сенсорна мережа, радіус дії сенсорів, а також затримка між ітераціями алгоритму.

Тестування було проведене за наступним алгоритмом: спочатку було налаштовано 4 окремі клієнтські сервери для одночасної генерації даних та запису їх в кластер бази даних. Для цього на кожен із цих серверів було встановлено додаток емуляції сенсорної мережі та всі залежності, які він вимагає. Потім налаштовано кластер InfluxDB на трьох серверах. Далі на всіх чотирьох клієнтських серверах запущено додатки генерації даних з певною конфігурацією, яка надає можливість генерувати дані з швидкістю яка є нижче середньої порівняно з максимальною швидкістю, яка використовувалася під час тестування. Дані генеруються протягом однієї хвилини, якщо після проходження часу останній літальний об’єкт не закінчив рух по своїй траєкторії, цей об’єкт продовжує рухатись і сенсорна мережа продовжує працювати поки траєкторія не закінчиться. Після закінчення роботи програм на всіх клієнтських серверах відбувається підрахунок запитів з сенсорної мережі до кластеру бази даних. Далі виконується запит для відображення кількості записів, які були збережені в базі. Для цього потрібно виконати команду *select count(x) from flight*. Якщо система працювала правильно, то кількість запитів з сенсорної мережі та кількість збережених записів має бути однаковою. Після цього зінюємо конфігурацію всіх клієнтських додатків так, щоб мережа почала генерувати дані швидше і повторюємо запуск. Якщо система витримує навантаження, після зупинки програм знову виконуємо підрахунок записів в кластері та кількість відправлених записів з мережі. Продовжуємо збільшення швидкості генерації даних з мережі до моменту появи помилок (коли кількість відправлених даних не відповідає збереженій кількості) або до аварійної зупинки бази даних.

Після тестування кластеру з трьох серверів видаляємо один сервер та повторюємо тестування. Для цього виконуємо команду *show servers*, дізнаємося ідентифікаційний номер одного з серверів, наприклад 3 та виконуємо команди *drop data server 3* та *drop meta server 3*. Перед запуском програм змінюємо їх конфігурацію, до тієї з якої починалося тестування кластуру на трьох серверах. Продовжуємо збільшення швидкості генерації даних між запусками програм та записуємо отримані результати для подальшого аналізу. Далі повторюємо всі кроки алгоритму для кластеру з одного серверу. Якщо під час підвищення швидкості генерації даних не вдається отримати помилок в роботі бази, останньою ітерацією для кожної конфігурації кластеру є запуск програм з конфігурацією, яка дозволяє досягти максимальної швидкість генерації даних.

Спочатку конфігуруємо кластер на трьох серверах. Стартовий набір конфігураційних параметрів має наступний вигляд:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 10
* Затримка між ітераціми – 100 мс

Отримані результати відображені в табл. 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 700 |  |  |
| Сервер 2 | 554 | 2150 | 2150 |
| Сервер 3 | 518 |  |  |
| Сервер 4 | 378 |  |  |

Табл. 4.1 Результати тестування роботи кластеру бази даних

Починаємо збільшувати швидкість генерації даних, запускаємо програми з наступною конфігурацією:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 15
* Затримка між ітераціми – 10 мс

Отримані результати відображені в табл. 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 3478 |  |  |
| Сервер 2 | 3500 | 11383 | 11383 |
| Сервер 3 | 3583 |  |  |
| Сервер 4 | 822 |  |  |

Табл. 4.2 Результати тестування роботи кластеру бази даних

Конфігурація для наступного тесту:

* Кількість сенсорів в кожному секторі ділянки – 50
* Радіус роботи кожного сенсору – 20
* Затримка між ітераціми – 10 мс

Отримані результати відображені в табл. 4.3

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 3476 |  |  |
| Сервер 2 | 3696 | 11602 | 11602 |
| Сервер 3 | 3577 |  |  |
| Сервер 4 | 853 |  |  |

Табл. 4.3 Результати тестування роботи кластеру бази даних

Запустимо програми з конфігурацією, яка дозволить генерувати дані з максимальною швидкістю:

* Кількість сенсорів в кожному секторі ділянки – 50
* Радіус роботи кожного сенсору – 20
* Затримка між ітераціми – 0 мс

Отримані результати відображені в табл. 4.4

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 3543 |  |  |
| Сервер 2 | 3598 | 11426 | 11426 |
| Сервер 3 | 3378 |  |  |
| Сервер 4 | 907 |  |  |

Табл. 4.4 Результати тестування роботи кластеру бази даних

З отриманих результатів стає зрозуміло, що отримати помилку роботи кластеру InfluxDB за таких умов(одночасний запис даних про стан 4 об’єктів) не вдалося, база даних витримує навантаження, під час всіх тестів, сумарна кількість запитів, які були відправлені та кількість збережених записів співпадають. Тому сховище для збереження даних з сенсорних мереж за такої конфігурації повністю відповідає вимогам, поставленим у технічному завданні.

Наступним кроком тестування є вилучення одного сервера та запуск кластеру InfluxDB на двох серверах. Після налаштування такого кластеру виконуємо перший тест з наступними параметрами:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 10
* Затримка між ітераціми – 100 мс

Отримані результати відображені в табл. 4.5

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 602 |  |  |
| Сервер 2 | 733 | 2148 | 2148 |
| Сервер 3 | 425 |  |  |
| Сервер 4 | 388 |  |  |

Табл. 4.5 Результати тестування роботи кластеру бази даних

Порівняємо дані з таблиць 4.1 та 4.5. За таких умов дані практично не відрізняються і зменшення кількості серверів у кластері не впливає на результат.

Збільшимо швидкість генерування даних:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 10
* Затримка між ітераціми – 50 мс

Отримані результати відображені в табл. 4.6

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 1061 |  |  |
| Сервер 2 | 1290 | 4063 | 4063 |
| Сервер 3 | 1037 |  |  |
| Сервер 4 | 675 |  |  |

Табл. 4.6 Результати тестування роботи кластеру бази даних

В цьому випадку затримка між ітераціями зменшилася вдвічі, в свою чергу швидкість генерування збільшилася вдвічі, що підтверджують результати. За таких умов база даних працєю прогнозовано та стабільно.

Збільшуємо швидкість створення даних:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 10
* Затримка між ітераціми – 10 мс

Отримані результати відображені в табл. 4.7

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 3195 |  |  |
| Сервер 2 | 3451 | 11261 | 11261 |
| Сервер 3 | 3584 |  |  |
| Сервер 4 | 1031 |  |  |

Табл. 4.7 Результати тестування роботи кластеру бази даних

В цьому випадку база даних поводить себе так само як і в попередньому. Наступним тестом є запуск програм з конфігурацією, якак була максимальною під час тестування кластеру на трьох серверах, а саме:

* Кількість сенсорів в кожному секторі ділянки – 50
* Радіус роботи кожного сенсору – 20
* Затримка між ітераціми – 0 мс

За таких умов приблизно через 30 секунд роботи кластер бази даних перестав приймати запити на збереження даних та перестав працювати. Для відновлення роботи системи потрібно було перезапустити кластер на всіх серверах. Дані, які були записані до зупинки не втратилися.

З отриманих результатів можна зробити висновок, що до певної межі навантаженості кластер бази даних сконфігурований на двох серверах працює так само як і кластер на трьох серверах. Але при досяганні певної межі починає працювати нестабільно і виходить з ладу.

Сконфігуруємо кластер бази наних на одному сервері. Запустимо програми з наступною конфігурацією:

* Кількість сенсорів в кожному секторі ділянки – 40
* Радіус роботи кожного сенсору – 10
* Затримка між ітераціми – 100 мс

Отримані результати відображені в табл. 4.8

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 602 |  |  |
| Сервер 2 | 733 | 2148 | 2148 |
| Сервер 3 | 425 |  |  |
| Сервер 4 | 388 |  |  |

Табл. 4.8 Результати тестування роботи кластеру бази даних

Результати мають такий самий вигляд, як і в двох попередніх випадках. Можна спрогнозувати, що під час поступового збільшення швидкості генерування даних система буде себе поводити так само, як і в попередніх випадках. Тому запустимо програми з конфігурацією, яка була максимальною у випадках с трьома і двома серверами:

* Кількість сенсорів в кожному секторі ділянки – 50
* Радіус роботи кожного сенсору – 20
* Затримка між ітераціми – 0 мс

Отримані результати відображені в табл. 4.9

|  |  |  |  |
| --- | --- | --- | --- |
| Сервер | Кількість запитів | Сумарна кількість запитів | Кількість збережених записів |
| Сервер 1 | 3582 |  |  |
| Сервер 2 | 4150 | 12618 | 12618 |
| Сервер 3 | 4068 |  |  |
| Сервер 4 | 818 |  |  |

Табл. 4.9 Результати тестування роботи кластеру бази даних

Несподівано система спрацювала так само, як і під час роботи на трьох серверах за такої самої конфігурації. Звернувшись до документації InfluxDB стає зрозуміло, що система кластеру побудована так, що для стабільної роботи системи потрібно мати непарну кількість вузлів у кластері. Що і було підтверджено на практиці. За даної конфігурації система працю стабільно і показує такі самі результати, як і система налаштована на трьох серверах. З цього можна зробити висновок, що межа навантаженості системи налаштованої на трьох серверах знаходиться вище ніж значення, яке було перевірене під час тестування, що є позитивним результатом і підтверджує правильність вибору саме цієї реалізації бази даних часових рядів для збереження даних з сенсорних мереж

# 4.2 ВИСНОВКИ

В даній роботі був проведений аналіз існуючих рішень для збереження даних з сенсорних мереж. Взявши до уваги специфічні особливості таких даних, було виділено як позитивні так і негативні характеристики кожного з таких рішень та обрано рішення, яке набільш точно виконує поставлені задачі. Таким рішенням виявилися бази даних часових рядів, а саме InfluxDB. В роботі було описано основні можливості InfluxDB та налаштовано і підготовлено до роботи кластер з трьох серверів цієї бази даних. Було обрано саме це рішення тому, що ця база даних швидко виконує запити на створення та читання даних; дані зберігаються у зручній структурі; InfluxDB відноситься до класу нереляційних баз даних, тому вона легко масштабується; для організації роботи у кластері використовується алгоритм узгодження Raft, який дозволяє як додавати, так і вилучати сервери із кластеру; у разі виходу із ладу серверів база даних не припиняє роботу.

Був створений додаток емуляції роботи сенсорної мережі. Додаток емулює роботу сенсорів, які виявляють поточну позицію траєкторії безпілотних літальних об’єктів в просторі та виконує їх збереження до бази даних. Для відображення даних у реальному часі був використаний інструмент візуалізації даних зі сховищ часових рядів Grafana.

Для підтвердження правильності обраного рішення було проведено моделювання різних умов та поведінки сенсорної мережі. Було проведено ряд тестів навантаженості бази даних з різною кількість серверів, які входять в склад кластеру, а також з різною швидкістю створення даних. Тести підтвердили теоритичні можливості кластеру бази даних часових рядів InfluxDB. Також тести вказали на деякі недоліки, на які треба звернути увагу для пришвидшення роботи бази в цілому. Загалом тести показали, що обрана база даних може опрацьовувати велику кількість запитів одночасно і дійсно використання такого рішення чудово підходить для збереження сенсорних даних.

Отже, було підтверджено, що обране сховище може ефективно використовуватися для збереження даних в системах з великою кількістю територіально розподілених сенсорів, які продукують однотипні дані та надсилають їх до розподілених серверів та подальшого їх аналізу.

# 

# СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. RRDtool — About RRDtool – Електрон. дані – [Україна] 2017. – Режим доступу: http://oss.oetiker.ch/rrdtool/ - Загол. з титул. екрану. – Мова: англ.
2. Big Data Platform — Enterprise Distributed NoSQL & Object Storage —Basho  – Електрон. дані – [Україна] 2017. – Режим доступу:

https://www.basho.com - Загол. з титул. екрану. – Мова: англ.

1. Grafana documentation – Електрон. дані – [Україна] 2017. – Режим доступу: https://grafana.com/ - Загол. з титул. екрану. – Мова: англ.

* 4. Benchmark Report Scaling the eXtremeDB In Memory Database System Beyond the Terabyte Size Boundary – Електрон. дані – [Україна] 2017. – Режим доступу: http://www.mcobject.com/terabyte-plus-benchmark

1. Hypertable — Big Data. Big Performance - Електрон. дані – [Україна] 2017. – Режим доступу: http://hypertable.org/ - Загол. з титул. екрану. – Мова: англ.
2. InfluxDB documentation – Електрон. дані – [Україна] 2017. – Режим доступу: https://docs.influxdata.com/- Загол. з титул. екрану. – Мова: англ.

1. \* Консультантом не може бути зазначено керівника дипломного проекту. [↑](#footnote-ref-1)