TỐI ƯU SERVER DỰ ÁN AIRENSE.

GVHD: TS. HÀN HUY DŨNG
NHÓM 7 – HỆ ĐIỀU HÀNH

PHẠM QUANG SÁNG 20193076
BÙI THỌ VINH 20193195
NGUYỄN KHẮC THÀNH 2019311
BÙI NHẬT MINH 20193006

***Abstract*—Báo cáo này tập trung vào tối ưu hoá hiệu suất của server dự án Airsense, một hệ thống đo đạc các chỉ số môi trường thông qua các trạm quan trắc. Ứng dụng sử dụng REST API để cung cấp dữ liệu cho giao diện người dùng. Do dữ liệu đo đạc khá lớn nên hiệu năng server còn tồn tại nhiều điểm nghẽn, nhóm nghiên cứu đã thực hiện tối ưu hoá hiệu suất bằng cách tối ưu cơ sở dữ liệu và thêm một lớp caching cho backend. Kết quả nghiên cứu cho thấy rằng, việc tối ưu hoá này đã giúp cải thiện đáng kể hiệu suất của server, tăng khả năng xử lý các yêu cầu của người dùng và giảm thời gian phản hồi. Nghiên cứu này có thể cung cấp thông tin hữu ích cho các nhà phát triển web application và làm tài liệu cho việc tối ưu hoá hiệu suất của dự án AirSense sau này.**

I. TỔNG QUAN

Dự án AIRSENSE Vietnam dưới sự triển khai của SPARC LAB là dự án với mục đích thu thập, cung cấp chính xác các dữ liệu về chất lượng không khí dưới website airsense.vn. Dữ liệu thu thập được cung cấp phục vụ nghiên cứu khí hậu và đào tạo STEM.

Người dùng có thể truy cập vào trang web và chọn trạm không khí cũng như khoảng thời gian cần xem để xem dữ liệu. Kiến trúc của dự án được mô tả cụ thể trong đường tài liệu.

Có thể chia kiến trúc của dự án thành 3 phần:

- Khối cảm biến đo đạc dữ liệu không khí
- Khối backend (NodeJS) chạy trên Ubuntu server gồm cơ sở dữ liệu (MongoDB) và các khối tính toán
- Client gồm trình duyệt, giao diện người dùng

Tình trạng của dự án hiện tại gặp phải là thời gian khi yêu cầu dữ liệu từ các trạm không khí là rất lâu, ảnh hưởng đến trải nghiệm của người dùng. Nhận thấy server chưa được tối ưu nhóm 7 chúng em tiếp cận dự án với mục tiêu cải thiện tốc độ truy vấn, tối ưu khả năng xử lý của server và đồng thời thông qua bài tập lớn nhóm tiếp cận với server sử dụng hệ điều hành Ubuntu, sử dụng và lập trình trên Linux, từ đó mang kiến thức từ môn hệ điều hành vào thực tiễn sử dụng.

II. VẤN ĐỀ HIỆN TẠI

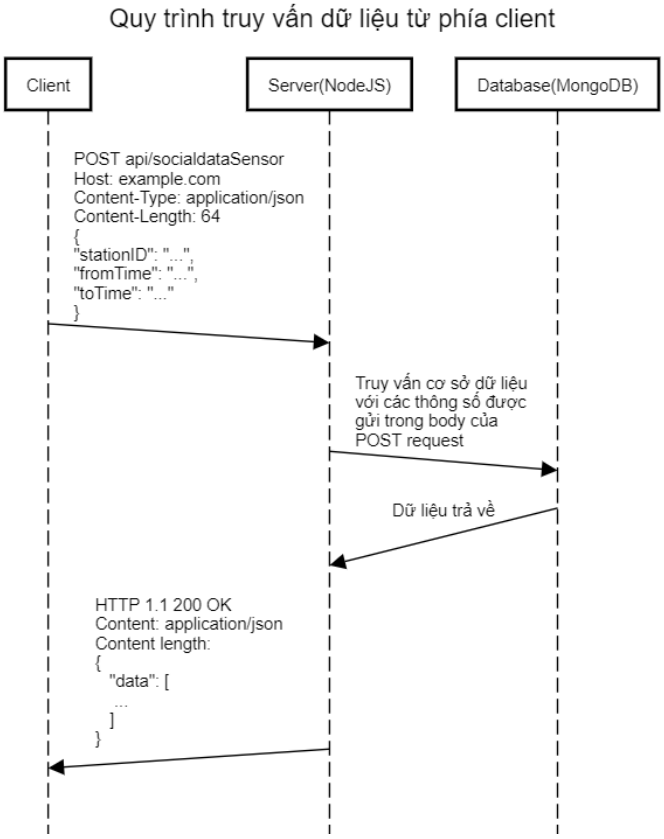
Hiện tại, server của dự án đang gặp vấn đề về tốc độ truy vấn dữ liệu. Khi yêu cầu dữ liệu từ phía frontend, thời gian đáp ứng của server thường mất hơn 20 giây tùy thuộc vào độ trễ của mạng. Tình trạng này ảnh hưởng đến trải nghiệm người dùng khi truy cập vào website của dự án.

Để xác định được nguyên nhân của vấn đề này, khi yêu cầu dữ liệu từ trang web công cụ Devtools được bật lên và chuyển sang tab Network để kiểm tra và phân tích các yêu cầu mạng của trang web. Khi ấn nút tìm kiếm browser gửi 1 yêu cầu HTTP với method là POST tới đường dẫn "htttps://airsense.vn/api/social/dataSensor". Quy trình truy vấn được mô tả như trong hình *Hình II-1* Trên trình duyệt cho thấy rằng thời gian phản hồi của API này là rất lâu và dữ liệu trả về thường rất lớn khoảng 2-3 Mb trong thời gian 1 ngày và 11-12 Mb trong khoảng thời gian một tuần.

Nhóm đã xác định được đây chính là điểm nghẽn của dự án, để cài thiện và tối ưu server bước đầu tiên cần phải chia và đo đạc từng các khối nhỏ để tìm ra điểm nghẽn sâu hơn và đề xuất phương án cải thiện. Sau khi tham khảo tài liệu thiết kế và mã nguồn của dự án, nhóm đã nhận ra các vấn đề sau cần phải khắc phục:

- Cơ sở dữ liệu chưa được tối ưu và indexing.
- Chưa có lớp caching dữ liệu
- Câu lệnh truy vấn còn chưa tối ưu.
- Nghiệp vụ và trải nghiệm của người dùng còn chưa hợp lý.

Ở phần III, nhóm sẽ thực hiện các bước đo đạc và triển khai các giải pháp cho vấn đề nêu trên.



Hình II-1. Quy trình truy vấn dữ liệu không khí

III. TRIỂN KHAI TỐI ƯU SERVER

Để tối ưu hiện năng của dự án trước tiên nhóm thực hiện đo đạt lại các tham số quan trọng nhằm đánh giá hiệu năng hiện tại của server. Ở bước này các công cụ chuyên dụng cho kiểm thử API được sử dụng như Insomina và Apache Jmeter. Dữ liệu lấy được sẽ được dùng làm mốc để đánh giá hiệu năng của server sau khi tối ưu.

A. Đo hiệu năng của dự án hiện tại

Để đo hiệu năng của hệ thống, Apache Jmeter 5.5 được sử dụng. Đây là công cụ thường được sử dụng với mục đích mô phỏng được tải nặng trên máy chủ bằng cách tạo ra hàng loạt những người dùng ảo cùng lúc trên máy chủ, từ đó tính toán, đo đạc được các thông số cần thiết để đánh giá, tối ưu server.

Các thông số cần quan tâm khi đo hiệu năng là:

- *Latency:* Độ trễ tính từ lúc request được khởi tạo, cho tới khi có byte đầu tiên của response nhận được từ server trả về
- *Average:* Trung bình xử lý yêu cầu được tính bằng milisecond
- *Median:* Trung vị xử lý yêu cầu được tính bằng milisecond
- *Bytes:* Lượng dữ liệu server truy vấn và trả về
- Throughput: Thông lượng của mạng (Số lượng request trên giây - KB/s)
- *Min, Max:* Latency nhỏ nhất và lớn nhất trong các trường hợp mô phỏng
- *X% Line:* Là giá trị thời gian truy vấn *t* xác định mà X% mẫu thử không vượt quá giá trị này; [100 – X]% mẫu còn lại sẽ cần khoảng thời gian trả về dữ liệu lớn hơn hoặc bằng *t*.

Để tiến hành đo, xây dựng các kịch bản cụ thể như sau: 1 user, 10 users gửi POST request truy vấn dữ liệu đo được trong 1 ngày, 1 tuần; 10 users gửi GET request tới trang chủ airsense.vn và airsense.vn/map. Trong đó, Number of Threads (users) mô phỏng lượng users kết nối tới máy chủ; Loop Count là số lần thực hiện test; Ramp-Up Period cho biết thời gian trì hoãn trước khi bắt đầu tới người dùng tiếp theo.

Thông số cài đặt chung của mỗi request để tiến hành đo trong các kịch bản như sau:

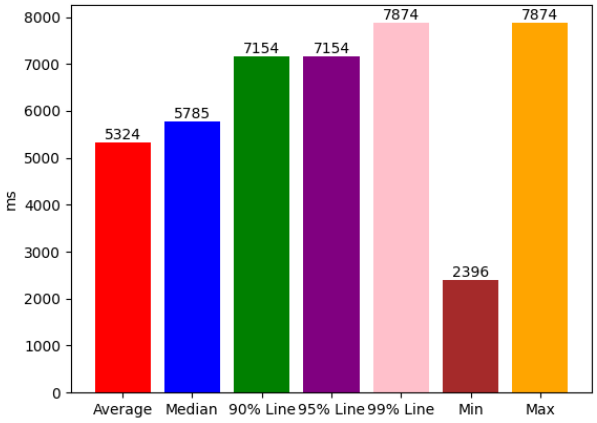
	POST		GET	
	1 ngày	1 tuần	Trang chủ	Trang map
Protocol	HTTP		HTTP	
HOST	airsense.vn		airsense.vn	
Method	POST		GET	
Path	api/social/dataSensor			/map
Body	{ "stationID": "3371212442", "fromTime": 1672506000, "toTime": 1672592400 }	{ "stationID": "3371212442", "fromTime": 1672506000, "toTime": 1673110800 }		

Bảng III-1 Các kịch bản đo

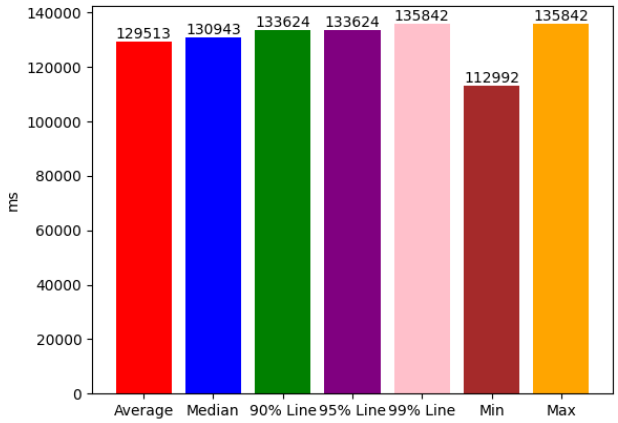
Tiến hành đo các kịch bản với tốc độ mạng Download: 114,55 Mbps và Upload: 114,42 Mbps, thu được các kết quả như sau:

Đối với các kịch bản đo với 1 user gửi POST request, Latency truy vấn dữ liệu trong 1 ngày - với lượng dữ liệu thu được là 2,453533 MB - là xấp xỉ 3 giây; 1 tuần - với lượng dữ liệu thu được là 11,821895 MB - là xấp xỉ 20 giây.

Đối với các kịch bản mô phỏng 10 users cùng gửi POST request tới server, thời gian trễ giữa mỗi user là 1 giây, ta thu được các thông số như sau:



Hình III-1. 10 users truy vấn dữ liệu trong 1 ngày



Hình III-2. 10 users truy vấn dữ liệu trong 1 tuần

	1 ngày	1 tuần
Dữ liệu trả về	2,43533MB	11,821895MB
Average	5,324s	129,513s
Median	5,785s	130,943s
90% Line	7,154s	133,624s
95% Line	7,154s	133,624s
99% Line	7,874s	135,842s
Min	2,396s	112,992s
Max	7,874s	135,842s

Bảng III-2 Thống kê thông số giữa một ngày và một tuần

Với lượng dữ liệu truy vấn trong 1 ngày, thông số mô phỏng như sau: Min – Max: 2,4s - 7,9s; Average: 5,325s; Medium: 5,785s; 90% Line: 7,154s. Với lượng dữ liệu truy vấn trong 1 tuần, thông số mô phỏng như sau: Min – Max: 113s - 135,8s; Average: 123,5s; Medium: 130,9s; 90% Line: 133,6s.

Kết hợp với các kịch bản trước đó, các thông số thời gian khi truy vấn dữ liệu trong 1 tuần lớn hơn nhiều so với thời gian truy vấn dữ liệu trong 1 ngày. Cụ thể: Average, Medium từ 5s lên 130s, 90% line từ 7s lên 133s, Min từ 2s lên 113s, Max từ 8s lên 136s. Qua đó, có thể kết luận khoảng thời gian truy vấn tỉ lệ thuận với lượng user truy cập và lượng dữ liệu trả về.

Đối với kịch bản mô phỏng 100 users cùng gửi GET request tới trang chủ và trang map, ta thu được lần lượt các trễ trung bình là 246 ms và 281 ms. Có thể thấy kịch bản này có độ trễ thấp hơn gấp nhiều lần các kịch bản trước, vì các trang trên là trang tĩnh, không phải truy vấn từ cơ sở dữ liệu, nên độ trễ chủ yếu từ việc gửi HTML, CSS, JavaScript,... tới người dùng. Từ đó, nhóm sẽ tập trung chủ yếu vào tối ưu phần truy vấn dữ liệu vào server bằng POST request.

B. Thực hiện tối ưu

Đầu tiên, nếu không tính đến thời gian xử lý của NodeJS thì thời gian chủ yếu mất cho việc người dùng yêu cầu dữ liệu và thời gian cần để truyền dữ liệu qua Internet. Điều này được kiểm chứng bằng việc nhóm đã thêm câu lệnh timer trước câu lệnh truy vấn dữ liệu trong hàm điều khiển việc lấy dữ liệu từ MongoDB.

```
console.log(new Date());
const result = await DataSensor.find({
    content: { $exists: true },
    topic: stationSelect,
    $and: [{ time: { $gt: fromTime } }, {
time: { $lt: toTime } }],
}
console.log(new Date());
res.send(
   JSON.stringify({
     station: stationID,
     data: result
 }));
```

Code III-1 Thêm timer vào hàm điểm khiển

Sau khi chạy một vài lần, nhóm đã khẳng định được với một request từ 25 – 26 giây thì việc truy vấn cơ sở dữ liệu tốn mất 20-22 giây. Cơ sở dữ liệu được sử dụng trong dự án là MongoDB, một dạng cơ sở dữ liệu không quan hệ NoSQL và được chạy trên cùng server. Chính vì vậy thời gian network latency giữa server và database là không đáng kể so với các kiến trúc database được tổ chức trên cloud khác. Có thể nhận thấy ở câu lệnh truy vấn trên, chưa sử dụng index khi truy vấn, điều này ảnh hưởng rất lớn tới hiệu năng khi truy vấn lượng dữ liệu lớn như của dự án AirSense.

Bước tiếp theo để kiểm tra tốc độ truy vấn dữ liệu thực sự từ MongoDB sử dụng công cụ MongoDB Compass. MongoDB Compass giúp trỏ thẳng trực tiếp vào cơ sở dữ liệu và sử dụng tính năng ở của sổ "Explain" có thể biết được hiệu năng của câu truy vấn. Sau đó nhóm sẽ thực hiện việc đánh chỉ mục cho MongoDB.

Đánh chỉ mục (indexing) là quá trình tạo ra một cấu trúc dữ liệu phụ trợ trong cơ sở dữ liệu để tăng tốc độ tìm kiếm, truy xuất và xử lý dữ liệu. Chỉ mục được sử dụng để tìm kiếm dữ liệu nhanh hơn bằng cách tạo ra một danh sách sắp xếp của các giá trị trong một trường cụ thể. Khi một truy vấn được thực hiện, cơ sở dữ liệu có thể sử dụng chỉ mục để truy xuất các giá trị nhanh chóng hơn thay vì phải quét toàn bộ dữ liệu trong bảng. Khi sử dụng chỉ mục, hệ thống sẽ tìm kiếm và truy xuất dữ liệu nhanh hơn, giảm thiểu thời gian phản hồi của hệ thống và tăng hiệu suất của ứng dụng.

Với câu truy vấn tương tự như trong controller. Ban đầu MongoDB chưa được đánh chỉ mục (indexing) thời gian thực hiện câu truy vấn là: *11647 ms*. Sau đó, các trường trong "topic;time;content" được đánh chỉ mục. lúc này thời gian thực hiện truy vấn đã giảm xuống còn *2142(ms)* gần 81.59% so với trước khi đánh chỉ mục.

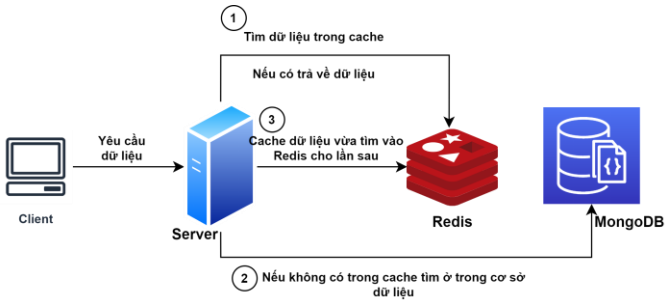
Sau khi đánh chỉ mục, câu truy vấn trong hàm điều khiển cũng cần được điều chỉnh để sử dụng chỉ mục khi truy vấn. Ngoài ra, các trường không cần thiết cũng được bỏ đi để giảm khối lượng dữ liệu gửi về ngoài ra hàm lean() được sử dụng để giảm bớt độ trễ và tăng tốc độ truy xuất dữ liệu từ MongoDB bằng cách chuyển đổi các kết quả truy vấn từ đối tượng Moongose sang đối tượng JavaScript thuần tuý. Sử dụng hàm *lean()* theo [1] có thể giảm được khối lượng dữ liệu tới 5 lần. Kết quả thực tế nhóm đánh giá, thời gian cho request đã giảm được khoảng 50% xuống còn dưới 10s.

```
  const result = await DataSensor.find(
    {
      content: { $exists: true },
      topic: stationSelect,
      $and: [{ time: { $gt: fromTime } }, {
time: { $lt: toTime } }],
    },
    { _id: 0, __v: 0, topic: 0 } // Hide các
trường không cần thiết
  )
    .hint({ time: 1 }) // Sử dụng index
    .lean(); // Hàm lean() tinh gọn dữ liệu
```

Code III-2. Tối ưu câu lệnh truy vấn

Việc này đã cải thiện được đáng kể tốc độ truy vấn tuy nhiên với lượng lớn dữ liệu người dùng vẫn sẽ phải đợi lâu và việc truy vấn cơ sở dữ liệu thường xuyên đòi hỏi khối lượng tính toán cao. Khi server có tải lớn việc truy cập vào cơ sở dữ liệu thường xuyên khiến server chịu tải nặng và có thể dẫn đến sập hệ thống. Để xử lý việc này một kiến trúc phổ biến thường được sử dụng là thêm một lớp đệm (caching) trước database. Redis là một hệ thống cơ sở dữ liệu lưu trữ ngay trên RAM (in-memory) hay được sử dụng cho việc caching này giúp tăng đáng kể khả năng truy vấn cơ sở dữ liệu, đồng thời giúp giảm tải cho server khi các dữ liệu thường xuyên được yêu cầu sẽ được cache vào RAM.

Kiến trúc phổ biến của Redis cho việc caching như sau.



Code III-3. Kiến trúc bộ đệm sử dụng Redis

Để triển khai với dự án AirSense nhóm đã cài đặt Redis Server trên máy chủ của dự án, kết nối Redis với backend NodeJS và thêm logic tương tự như hình … vào hàm controller.

```
// Tạo cache key dựa vào body cửa request
const cachedKey = `${stationID}:
${fromTime}:${toTime}`;
// Tìm trong Redis, nếu có thì trả về dữ liệu
const cachedData = await
RedisClient.get(cachedKey);
  if (cachedData !== null) {
    const data = JSON.parse(cachedData);
    return res.send({
      station: stationID,
      data
    })
  }
// Nếu không tìm trong database
...
```

```
// Sau đó cache dữ liệu vừa tìm được trong Redis với thời gian hết hạn exp time = 3600
  await    RedisClient.setEx(cachedKey,    3600,
JSON.stringify(result))
```

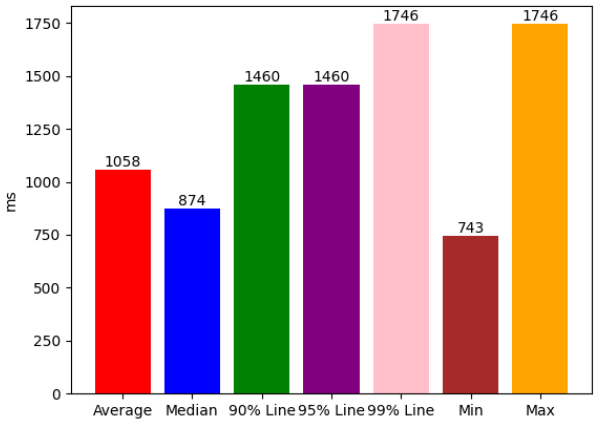
Code III-4. Thêm Redis cho hàm điều khiển

Lần đầu khi truy vấn dữ liệu mới do chưa tồn tại trong RAM nên thời gian phản hồi vẫn phụ thuộc vào tốc độ truy vấn từ MongoDB. Sau lần đó dữ liệu được Cache bằng Redis và tốc độ phản hồi được cải thiện đáng kể. Với kết nối mạng nhanh nhóm đo đạc được thực tế tốc độ phản hồi ở ngưỡng dưới 5s với dữ liệu đã được cache. Ở phần IV, đánh giá cụ thể bằng phương pháp đo đạc tại phần III.A sẽ được sử dụng lại để so sánh sự khác biệt

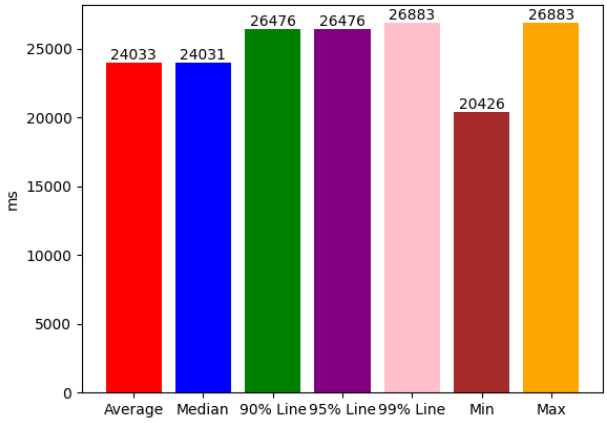
IV. KẾT QUẢ VÀ ĐÁNH GIÁ

Sau khi thực hiện tối ưu, các kịch bản đo ban đầu được sử dụng lại để đánh giá hiệu năng server. Các kịch bản được thay đổi lượng user truy cập, tăng lên 10 users và 50 users.

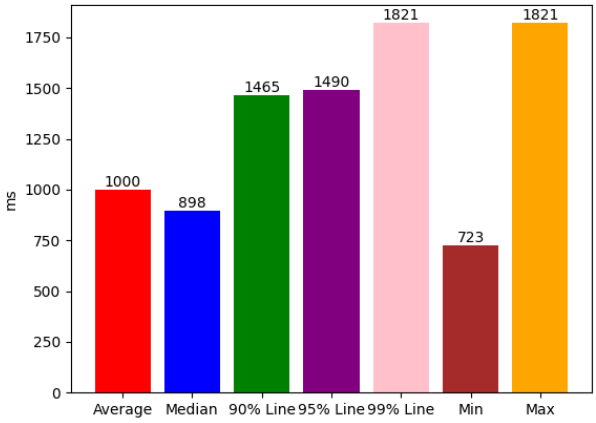
Tiến hành đo với kịch bản 10, 50 users truy vấn dữ liệu trong 1 ngày và 1 tuần, ta thu được các kết quả sau:



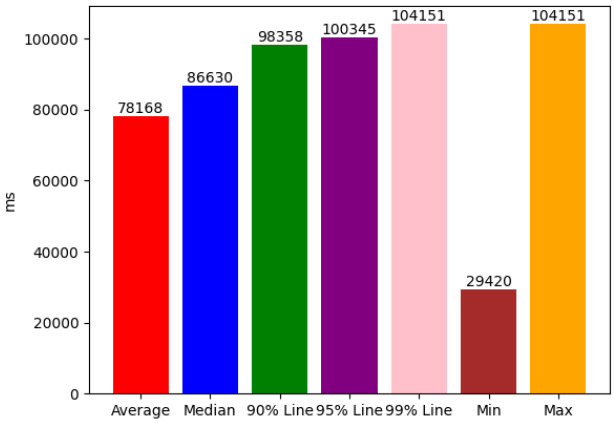
Hình IV-1. 10 users truy vấn dữ liệu trong 1 ngày



Hình IV-2. 10 users truy vấn dữ liệu trong 1 tuần



Hình IV-3. 50 users truy vấn dữ liệu trong 1 ngày



Hình IV-4. 50 users truy vấn dữ liệu trong 1 tuần

	10 users		50 users	
	1 ngày	1 tuần	1 ngày	1 tuần
Average	1058ms	24033ms	1000ms	78168ms
Median	874ms	24031ms	898ms	86630ms
90% Line	1460ms	26476ms	1465ms	98358ms
95% Line	1460ms	26476ms	1490ms	100345ms
99% Line	1746ms	26883ms	1821ms	104151ms
Min	734ms	20426ms	723ms	29420ms
Max	1746ms	26883ms	1821ms	104151ms

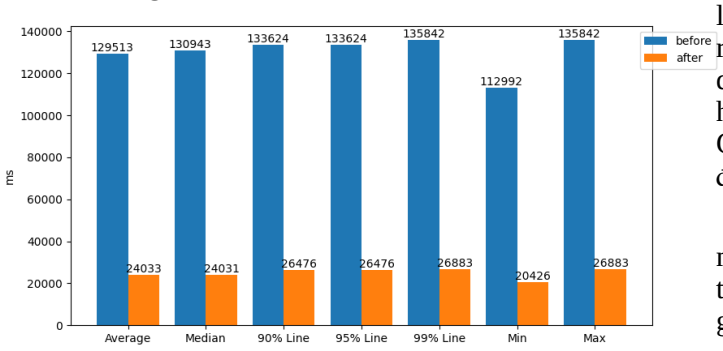
Bảng IV-1. Bảng thống kê các thông số đo được sau khi tối ưu

Với lượng dữ liệu trong 1 ngày, các thông số không thay đổi nhiều giữa 2 kịch bản 10 users và 50 users truy cập. Cụ thể: Average: 1s; Median: 0,8s; 90% Line: 1,46s; Min: 0,7s; Max: 1,8s.

Với lượng dữ liệu trong 1 tuần, các thông số thay đổi khá nhiều giữa 2 kịch bản 10 users và 50 users truy cập. Cụ thể: Average từ 24s lên 78s; Medium từ 24s lên 86s; 90% Line từ 26s lên 98s; Min từ 20s lên 29s; Max từ 27s lên 104s.

Có thể thấy, với lượng dữ liệu truy vấn lớn, latency sẽ tỉ lệ thuận với lượng users truy cập. Còn đối với lượng dữ liệu truy vấn thấp, tương đương với 1 ngày, thì latency sẽ biến thiên gần như bằng 0 nếu như lượng users không thay đổi quá nhiều.

So sánh server trước và sau khi tối ưu với cùng 1 kịch bản là 10 users truy vấn dữ liệu trong 1 tuần, ta thu được đồ thị và bảng sau:



Bảng IV-2. Biểu đồ so sánh trước và sau khi tối ưu

	Trước tối ưu	Sau tối ưu
Average	129,513s	24,033s
Medium	130,943s	24,031s
90% Line	133,624s	26,476s
95% Line	133,624s	26,476s
99% Line	135,842s	26,883s
Min	112,992s	20,426s
Max	135,842s	26,883s

Bảng IV-3. Bảng thống kê các thông số quan trọng của server trước và sau khi tối ưu

Các thông số sau khi tối ưu đều giảm 5 lần so với trước đó. Cụ thể: Average, Median giảm từ 130s xuống 24s; 90% Line giảm từ 133s xuống 26s; Min giảm từ 113s xuống 20s; Max giảm từ 135s xuống 27s. Điều đó cho thấy tốc độ truy vấn của server đã được cải thiện rõ rệt sau khi tối ưu.

V. CÁC CẢI TIẾN ĐỀ XUẤT

Sau khi tối ưu thì với kiến trúc hiện tại đã cải thiện nhiều và đảm bảo được tải lớn với độ trễ thấp hơn so với hệ thống ban đầu. Nhóm trong quá trình làm việc cũng đã nhận ra thêm một số vấn đề như sau có thể cải tiến để cải thiện hiệu năng dự án hơn.

Vấn đề đầu tiên nhóm nhận thấy được là việc tổ chức cơ sở dữ liệu chưa thực sự được tối ưu cho việc truy vấn dữ liệu IoT. Điều này do hạn chế của MongoDB trong việc truy vấn dữ liệu IoT. Để khắc phục việc này một trong những hệ cơ sở dữ liệu thường xuyên được sử dụng là cho lĩnh vực IoT là InfluxDB. InfluxDB là dữ liệu dạng TimeSeries. Ở trong [2] cho thấy InfluxDB nhanh hơn đáng kể với 1.9 lần thông lượng ghi, 7.3 lần ít dung lượng sử dụng, 5 lần hiệu năng so với MongoDB. Ngoài ra. MongoDB thực chất cũng vẫn được sử dụng cho kiểu dữ liệu IoT, gần đây MongoDB 6.0 ra mắt và đã bắt đầu hỗ trợ thêm các tính năng cho dữ liệu dạng IoT này[3]. Các doanh nghiệp lớn trong lĩnh vực IoT như Bosch cũng đã và đang sử dụng MongoDB cho hạ tầng của mình[4].

Vấn đề thứ 2, là cần giảm tần số dữ liệu. Các dữ liệu môi trường thường không thay đổi quá nhanh nên với hệ thống hiện tại có thể giãn khoảng thu thập dữ liệu. Từ đó giảm đáng kể khối lượng dữ liệu trả về.

Vấn đề thứ 3, là việc xử lý tải lớn. Với cấu hình của server hiện tại khi kiểm tra lượng tải lớn 100 user. Server có hiện tượng sập. Điều này có thể xử lý bằng 2 cách là Horizontal Scaling và Vertical Scaling. Horizontal Scaling là tăng khả năng xử lý của server bằng việc cải

thiện phần cứng. Vertical Scaling là chỉ khi có tải lớn sẽ thêm các node xử lý nhằm giảm bớt tải của node chính.

Ngoài ra, giao diện người dùng cần được làm thân thiện hơn, tập trung vào trải nghiệm người dùng.

VI. KẾT LUẬN

Qua quá trình làm bài tập lớn môn Hệ Điều Hành, nhóm đã được tiếp cận với dự án AirSense với mục tiêu đo đạc lại hiệu năng và cải tiến hiệu năng của dự án. Kết quả cuối cùng nhóm đã tối ưu cơ sở dữ liệu hiện tại bằng việc đánh chỉ mục, bổ sung thêm một tầng bộ đệm dữ liệu. Từ đó tối ưu được đáng kể hiểu năng của dự án. Kết quả của bài tập lớn này có thể là bước đệm và tài liệu giúp cho dự án AirSense phát triển hơn trong tương lai.

TÀI LIỆU THAM KHẢO

[1] Churilo, C. (2022, 11 17). *InFlux Data*. Retrieved from https://www.influxdata.com/blog/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/

[2] *mongoose*. (n.d.). Retrieved from mongoose: https://mongoosejs.com/docs/tutorials/lean.html

[3] Schmitz, J. (2022, 11 1). *Youtube*. Retrieved from https://www.youtube.com/watch?v=tryvfydZXPE

[4] Team MongoDB (n.d.). Retrieved from https://www.mongodb.com/customers/bosch

PHỤ LỤC

[I]. Các kết quả đo đạc được. https://drive.google.com/drive/folders/1QTkmaDUVf4afstl0-2bALqDO2EJyna6r?usp=sharing

[II]. Github bao gồm các code môn Hệ điều hành. https://github.com/saltmurai/OS