BỘ GIÁO DỤC VÀ ĐÀO TẠO

VIỆN HÀN LÂM
KHOA HỌC VÀ CÔNG NGHỆ VN

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

NGUYỄN ĐÌNH MẠNH LINH

NGHIÊN CỨU PHƯƠNG PHÁP XỬ LÝ LUỒNG DỮ LIỆU LỚN CÓ KHẢ NĂNG CO DÃN CHO HỆ THỐNG IOT

LUẬN VĂN THẠC SĨ HỆ THỐNG THÔNG TIN

Hà Nội, 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠC

VIỆN HÀN LÂM
KHOA HỌC VÀ CÔNG NGHỆ VN

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

NGUYỄN ĐÌNH MẠNH LINH

NGHIÊN CỨU PHƯƠNG PHÁP XỬ LÝ LUỒNG DỮ LIỆU LỚN CÓ KHẢ NĂNG CO DÃN CHO HỆ THỐNG IOT

LUẬN VĂN THẠC SĨ HỆ THỐNG THÔNG TIN
Mã số: 8 48 01 04

NGƯỜI HƯỚNG DẪN KHOA HỌC
PGS.TS Nguyễn Trường Thắng

Hà Nội, 2023

LỜI CAM ĐOAN

Tôi xin cam đoan đề tài nghiên cứu trong luận văn này là công trình nghiên cứu của tôi dựa trên những tài liệu, số liệu do chính tôi tự tìm hiểu và nghiên cứu. Chính vì vậy, các kết quả nghiên cứu đảm bảo trung thực và khách quan nhất. Đồng thời, kết quả này chưa từng xuất hiện trong bất cứ một nghiên cứu nào. Các số liệu, kết quả nêu trong luận văn là trung thực nếu sai tôi hoàn chịu trách nhiệm trước pháp luật.

Hà Nội, tháng 12 năm 2023

Học viên thực hiện

Nguyễn Đình Mạnh Linh

LỜI CẢM ƠN

Đầu tiên em xin gửi lời cảm ơn đến PGS.TS Nguyễn Trường Thắng, thầy đã tận tình giúp đỡ, hướng dẫn em hoàn thành tốt luận văn này.

Em cũng xin chân thành cảm ơn các thầy cô giáo, phòng Đào tạo tại Học viện khoa học công nghệ đã tận tình chỉ bảo, tạo điều kiện cho tôi hoàn thành bài luận văn của mình. Qua đây, em cũng gửi lời cảm ơn tới gia đình, bạn bè đã động viên, khuyến khích và tạo điều kiện cho tôi trong suốt quá trình học tập cũng như trong quá trình làm luận văn.

Do còn hạn chế nhiều về kiến thức, kinh nghiệm và thời gian tìm hiểu nên luận văn chắc chắn còn nhiều thiếu sót. Em rất mong sẽ nhận được nhiều đóng góp của thầy, cô để có thể hoàn thiện hơn bài luận văn này. Và em cũng hy vọng rằng đây sẽ là tài liệu bổ ích cho những người quan tâm về lĩnh vực này, mọi chi tiết cần điều chỉnh, bổ sung xin liên hệ tới manhlinh.nguyendinh@gmail.com.

Em xin chân thành cảm ơn!

Hà Nội, tháng 12 năm 2023

Học viên thực hiên

Nguyễn Đình Mạnh Linh

DANH MỤC TỪ VIẾT TẮT

Chữ viết tắt	Ý nghĩa
LZ77	Lempel-Ziv-1977
LZMA	Lempel-Ziv-Markov chain algorithm
IoT	Internet of Things
TCP	Transmission control protocol
UDP	User Datagram Protocol
MQTT	Message Queuing Telemetry Transport

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

MỞ ĐẦU

Tại Việt Nam, đi cùng với việc dân số tăng thì nhu cầu về việc sử dụng công nghệ thông minh tăng dần, vì vậy các dịch vụ, tiện ích cũng tăng theo. Trong khi đó, hạ tầng mạng không thể theo kịp nhu cầu này, từ đó đẫn đến yêu cầu xây dựng hệ thống xử lý luồng dữ liệu có tính linh động (khả năng co dãn) trong các đề tài liên quan đến lĩnh vực Internet of Things (IoTs).

Ở đề tài này, hướng nghiên cứu là khảo sát các phương pháp xử lý luồng dữ liệu cho các ứng dụng IoT trên điện toán đám mây và nghiên cứu mô hình sử dụng các thuật toán nén bao gồm là Deflate, LZMA, LZ4, và Bzip2 để tối ưu về sử dụng hạ tầng mạng, tài nguyên tính toán và lưu trữ.

Khóa luận được bố cục như sau:

- Chương 1: Tổng quan về lý thuyết và các nghiên cứu liên quan phương pháp xử lý luồng dữ liệu. Từ đó, xác định vấn đề cần giải quyết.
- Chương 2: Nghiên cứu về các phương pháp xử lý luồng dữ liệu và đề xuất mô hình luồng dữ liệu cho các hệ thống IoT.
- Chương 3: Lựa chọn dữ liệu đầu vào và các kết quả đã đạt được.
- Chương 4: Kết luận và các kiến nghị.

CHƯƠNG 1: TỔNG QUAN LÝ THUYẾT

1.1 Sự cần thiết tiến hành nghiên cứu

Sự tăng lên cả về tính khả dụng và số lượng của các thiết bị cảm biến, các thiết bị di động, và các thiết bị thông minh đã dẫn tới sự bùng nổ về khối lượng, chủng loại và tốc độ sản sinh dữ liệu cũng như nhu cầu phân tích khối dữ liệu khổng lồ này ở các mức độ nhất định. Mặt khác, bởi vì xã hội đang trở nên ngày càng kết nối, những tổ chức khác nhau cũng đang sản sinh ra những khối lượng dữ liệu khổng lồ tương ứng từ các hoạt động điều phối kinh doanh hàng ngày, từ việc theo dõi các hoạt động khách hàng, từ các thiết bị đeo tay hỗ trợ người dùng, từ các ứng dụng tài chính kế toán hay từ các thí nghiệm khoa học đòi hỏi nguồn tài nguyên trang bị lớn. Lượng dữ liệu lớn này được gọi chung là *Big Data* bởi vì những thách thức nó tạo ra cho các cơ sở hạ tầng điện toán hiện tại cả về mặt truyền tải, lưu trữ, và xử lý dữ liệu.

Trên thực tế, một phần lớn lượng dữ liệu này sẽ mang lại lượng thông tin có nhiều giá trị nhất chỉ khi nó được phân tích càng nhanh càng tốt ngay sau khi được tạo ra. Với một vài ngữ cảnh ứng dụng đang nổi lên trong những năm gần đây như thành phố thông minh, giám sát hoạt động của các cơ sở hạ tầng lớn, hay Internet của vạn vật (IoT), những luồng dữ liệu liên tục này phải được xử lý với độ trễ rất nhỏ. Trong một số lĩnh vực, tồn tại nhu cầu xử lý luồng dữ liệu (data stream processing) để phát hiện ra các mẫu cần tìm, xác định các lỗi có thể gặp phải hoặc là thu được những thông tin chi tiết và nâng cao về đối tượng đang theo dõi.

Để có thể đạt được mục tiêu này một trong số các giải pháp tiềm năng đó là sử dụng những ưu điểm của điện toán đám mây như tính kinh tế của đầu tư hạ tầng, tính co dãn của tài nguyên hay giảm tải và tiết kiệm băng thông. Đề tài này khảo sát các phương pháp xử lý luồng dữ liệu trên điện toán đám mây và nghiên cứu mô hình sử dụng các thuật toán nén để tối ưu về sử dụng hạ tầng mạng, tài nguyên tính toán và lưu trữ. Bên cạnh đó, kết quả nghiên cứu của đề tài có tiềm năng đưa vào sử dụng trực tiếp cho các dự án IoT đang triển khai tại Việt Nam như nhà thông minh, nông nghiệp thông minh giúp tăng cường hiệu năng xử lý và phân tích dữ liệu của các hệ thống này.

1.2 *Cơ sở lý thuyết*

1.2.1 Hệ thống IoT

Khái niệm "Internet of Things" (IoT) là sự kết hợp của hai yếu tố: "Internet" và "Things," mang ý nghĩa tạo ra một mạng kết nối giữa các vật được định danh và khả năng giao tiếp thông qua các giao thức chuẩn [1]. Một cách đơn giản, IoT đại diện cho sự giao thoa giữa các thiết bị thông minh và khả năng xử lý thông tin mà chúng tạo ra. Công nghệ này cho phép các hệ thống vật lý có khả năng quan sát, nghe, xử lý thông tin, và thực hiện các hoạt động tương tự như con người thông qua việc trao đổi thông tin giữa các thành phần trong hệ thống. Điều này giúp chuyển đổi các hệ thống tự động truyền thống thành các hệ thống thông minh, mô phỏng cách con người hoạt động. IoT tạo nên sự kết hợp và tương tác của nhiều công nghệ, ví dụ như hệ thống nhúng, công nghệ truyền thông, mạng cảm biến, và nhiều công nghệ khác … Hình 1-1 trình bày các ứng dụng của IoT trong các lĩnh vực khác nhau, với mỗi lĩnh vực đều có những đặc điểm riêng, trong đó sự trao đổi thông tin trực tiếp giữa các thiết bị đóng một vai trò quan trọng.

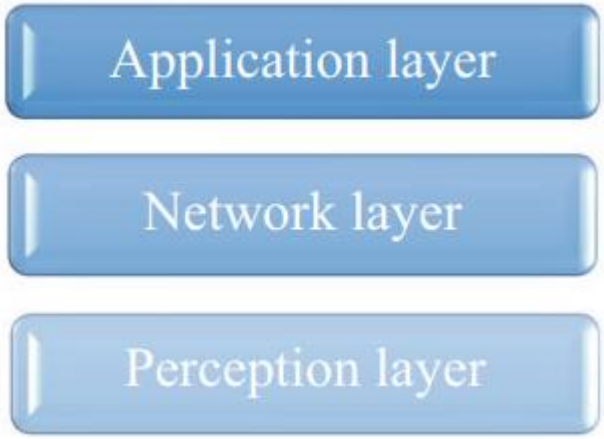
Hình 0-1: Các lĩnh vực áp dụng công nghệ IoT

Để một hệ thống IoT có thể hoạt động, cốt lõi của nó nằm trong kiến trúc mà nó áp dụng. Đây chính là lý do mà IoT có thể được hiểu như một giải pháp, bao gồm các bước cụ thể để thực hiện ý tưởng của hệ thống. Tuy nhiên, cho đến thời điểm

hiện tại, vẫn chưa tồn tại một kiến trúc IoT chung mà có thể áp dụng cho mọi lĩnh vực, và các giải pháp hiện có chỉ ở mức độ nghiên cứu. Do đó, giải pháp này thường được gọi là "Internet of Things reference architecture." hay mô hình tham chiếu Trong hàng loạt các mô hình đề xuất, mô hình ba lớp được coi là kiến trúc truyền thống [2]–[4]. Từ mô hình này, các biến thể đã được phát triển để phù hợp với từng lĩnh vực cụ thể.

Kiến trúc truyền thống

Theo các nghiên cứu trong [2]–[4], kiến trúc IoT truyền thống sẽ bao gồm ba lớp, Hình 1-2:



Hình 0-2: Kiến trúc IoT truyền thống

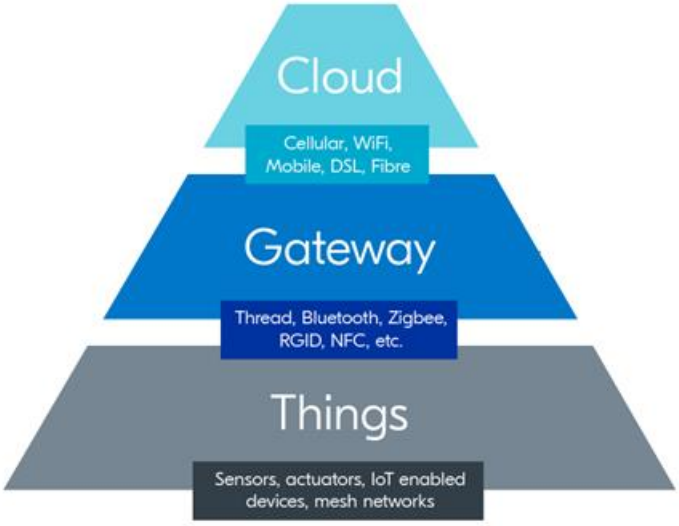
The perception layer là tầng tương tác với môi trường vật lý để thu thập các thông tin vật lý thông qua các cảm biến và tác động thông qua các cơ cấu chấp hành. Các chuẩn "plug-and-play" được ưu tiên sử dụng cho các vật với các yêu cầu thiết kế khác nhau [4]. Tầng nhận thức sẽ số hóa và gửi dữ liệu qua các kênh bảo mật cho các tầng phía trên.

The network layer hoạt động như những chiêc cầu kết nối các thiết bị thông minh ở tầng perception layer và các máy chủ server ở tầng application layer. Do các thiết bị ở tầng Things sử dụng rất nhiều giao thức khác nhau, tầng này như một cộng cụ lọc để chuẩn hóa cho các giao tiếp ở tầng trên.

The application layer có nhiệm vụ cung cấp cho người dùng những chức năng cụ thể. Vì vậy nhiệm vụ của tầng này sẽ phụ thuộc vào mục đích mà giải pháp IoT được áp dụng, ví dụ như nhà thông minh, thành phố thông minh, …

Kiến trúc tham khảo

Kiến trúc truyền thống chỉ mang ý nghĩa nghiên cứu, trong các ứng dụng sẽ sử dụng kiến trúc IoT tham khảo:



Hình 0-3: Kiến trúc IoT tham khảo

Tầng Things: Theo Hình 1-3, các mạch đo Sensors hoặc các cơ cấu chấp hành Actuators sẽ tương tác với môi trường vật lý và tạo ra dữ liệu để gửi lên phía Gateway thông qua các giao thức thông dụng như Bluetooth, Zigbee,. . . Các thiết bị này đa phần là nhỏ gọn với sức mạnh tính toán không quá lớn vì mục tiêu của các thiết bị là (1) đối với mạch đo Sensor là thu thập dữ liệu thông qua và gửi các thông tin đó cho các tầng phía trên xử lý, (2) đối với mạch cơ cấu chấp hành Actuator là nhận lệnh điều khiển hoặc các process value tương ứng từ các tầng phía trên (thường là Gateway) và thực hiện yêu cầu đó.

Ví dụ trong các ứng dụng tòa nhà thông minh, các mạch đo sau khi thu thập dữ liệu sẽ gửi về phía người dùng để theo dõi hoặc để người dùng ra các quyết định điều khiển. Giả sử trong trường hợp nhiệt độ căn phòng tăng quá cao so với ngưỡng đã được thiết lập, dưới tác động của cơ cấu chấp hành, máy sưởi (heater) có thể được tắt đi. Với sử dụng một lượng lớn các mạch đo và cơ cấu chấp hành này, người dùng có thể tương tác nhiều hơn với chính căn nhà của mình, thậm trí từ các khoảng cách các xa. Điều này có thể dẫn đến nâng cao độ thoải mái và giảm năng lượng tiêu thụ của cả tòa nhà.

Tầng Gateway là một tầng vô cùng quan trọng trong kiến trúc của một hệ thống IoT, vì khác với các ứng dụng web hay mobile, các dự án IoT phải tính đến vấn đề xử lý dữ liệu thời gian thực, đặc biệt là ra quyết định thời gian thực. Thời gian thực có thể hiểu là độ trễ ít nhất. Vì vậy cần một thiết bị có khả năng xử lý dữ liệu thời gian thực, đảm bảo độ trễ ít nhất mà lại không nằm quá xa so với các thiết bị tầng Things, đó là lý tầng này được sinh ra. Về mặt sức mạnh tính toán, các thiết bị của

tầng này có sức mạnh tính toán lớn hơn các thiết bị tầng Things nhưng nhỏ hơn các thiết bị tầng Cloud. Nhiệm vụ của Gateway thông thường gồm:

- Xử lý, loại bỏ bớt các dữ liệu lỗi mà các thiết bị tầng Things gửi lên,
- Lưu trữ dữ liệu tạm thời trong trường hợp Web Server bị mất kết nối,
- Ra quyết định trong các trường hợp cần có những tác động lại về môi trường vật lý bị gới hạn thời hạn (soft/ hard time-bound).

Tầng Cloud là tầng có sức mạnh tính toán mạnh nhất trong kiến trúc của một hệ thống IoT. Với sức mạnh tính toán mạnh mẽ, các thiết bị tầng Cloud có thể xử lý một lượng lớn dữ liệu. Thông thường, tầng Cloud được sử dụng để lưu trữ dữ liệu lâu dài, xử lý dữ liệu lớn, đưa ra các quyết định dựa trên dữ liệu lớn đó. Ví dụ, trong các ứng dụng y tế, các thiết bị tầng Things sẽ gửi dữ liệu về tình trạng sức khỏe của bệnh nhân lên tầng Cloud, tầng Cloud sẽ lưu trữ dữ liệu này và đưa ra các quyết định dựa trên dữ liệu lớn đó. Các quyết định này có thể là đưa ra lời khuyên cho bệnh nhân, hoặc đưa ra các quyết định về việc điều khiển các thiết bị tầng Things. Ví dụ, nếu tình trạng sức khỏe của bệnh nhân không tốt, tầng Cloud sẽ đưa ra lời khuyên cho bệnh nhân nên nghỉ ngơi, hoặc đưa ra quyết định điều khiển các thiết bị tầng Things như máy đo huyết áp, máy đo đường huyết, máy đo nhịp tim, . . . để đo lại các chỉ số này.

Mô hình điện toán đám mây

Trong kiến trúc truyền thống, các dữ liệu được tạo ra từ các thiết bị ở tầng dưới được truyền lên đám mây để xử lý tập trung. Khi các dịch vụ của đám mây được cung cấp bởi bên thứ ba là các tập đoàn công nghệ lớn như Amazon hay Google thì mô hình này còn được gọi là điện toán đám mây (Cloud Computing). Ý tưởng của mô hình này là người phát triển có thể tập trung vào phát triển các tính năng của dự án và việc quản lý hạ tầng nơi dự án được triển khai sẽ được lo liệu bởi các tập đoàn công nghệ lớn như Microsoft Azure[1] hay Amazon AWS[2]. Những ưu điểm của mô hình này mang lại bao gồm tính kinh tế của đầu tư hạ tầng, tính co dãn của tài nguyên sử dụng, và sự giảm tải, tiết kiệm băng thông. Trong đó, tính năng co dãn là đặc biệt quan trọng đối với các nhà cung cấp dịch vụ mô hình này [5]–[7]. Theo [8], tính năng co dãn được định nghĩa là khả năng có thể thêm hoặc bớt tài nguyên một cách tự động và nhanh chóng để đáp ứng nhu cầu thay đổi của tải theo cách tối ưu nhất. Nghiên

[1] http://www.windowsazure.com/

[2] https://aws.amazon.com/

cứu đã đưa ra công thức co dãn như trong Hình 1-4 bao gồm ba khía cạnh là sự tăng giảm (scalability), sự tự động (automation) và tối ưu (optimization). Như vậy, một ứng dụng có khả năng co dãn là ứng dụng đó sử dụng ít tài nguyên nhất để có thể đáp ứng nhu cầu của tải.

$$Elasticity = \underbrace{scalability + automation}_{\text{auto-scaling}} + optimization$$

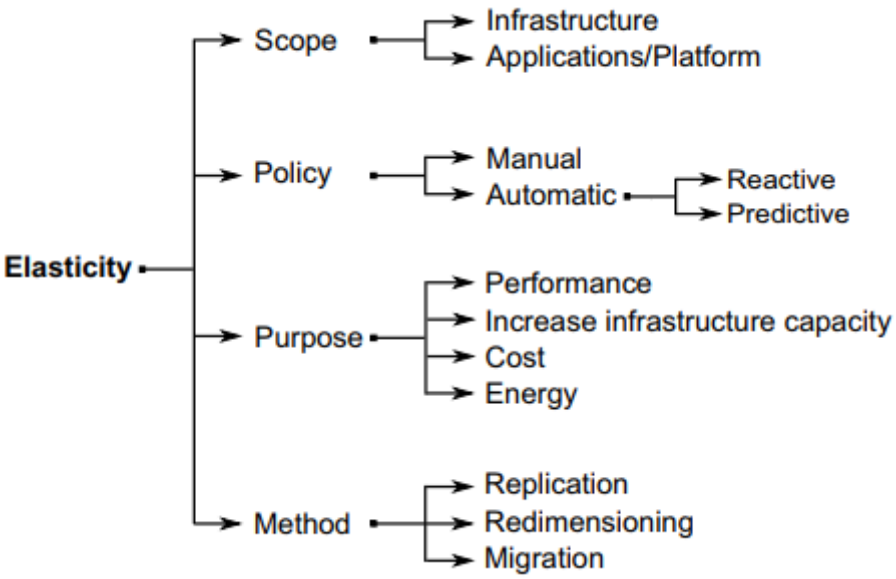
Hình 0-4: Định nghĩa về tính năng co dãn của điện toán đám mây

Để hiểu rõ hơn về tính năng co dãn, cần phải phân biệt rõ với hai khái niệm liên quan, đó là sự tăng giảm (scalability) và sự hiệu quả (efficiency). Khái niệm thứ nhất là điều kiện cần để đạt được sự co dãn, là tính năng của hệ thống sử dụng thêm hoặc giảm tải nguyên phụ thuộc vào yêu cầu phía tải từ phía người dung. Tuy nhiên, tính năng này không liên quan trực tiếp đến sự hiệu quả mà tài nguyên sử dụng có thể đáp ứng nhu cầu sử dụng, tức là tính tối ưu tài nguyên sử dụng. Trong khi đó, khái niệm thứ hai có thể hiểu là điều kiện đủ của tính co dãn, vì khái niệm này mang ý nghĩa tài nguyên được sử dụng hoàn toàn ứng với nhu cầu sử dụng. Nhưng có một điểm khác biệt là sự hiệu quả không chỉ nằm ở việc sử dụng tài nguyên mà còn mở rộng ở các khía cạnh khác như cách dự án được triển khai hay giao tiếp giữa các thành phàn của tài nguyên.

Về mặt lý thuyết, co dãn có thể được thực hiện theo hai cách là (1) theo chiều ngang, tức là thêm hoặc bớt tài nguyên tương tự như tài nguyên đang có và (2) theo chiều dọc, tức mở rộng hoặc giảm bớt tính năng của tài nguyên đang có thể. Theo một cách dễ hiểu hơn, điện toán đám mây là các server có sức mạnh tính toán lớn. Khi co dãn theo chiều ngang có nghĩa là thêm các server có sực mạnh tính toán tương tự như server đang sử dụng và theo chiều dọc là sự tăng, giảm các tính năng, đặc điểm như sức mạnh CPU, bộ nhớ RAM, hay băng thông mạng. Các giải pháp được nghiên cứu và triển khai trong thực tế thì được phân loại dựa trên nhiều đặc điểm khác nhau như trong Hình 1-5. Đáng chú ý trong việc phân loại các giải pháp này là phân loại theo chế độ hay chính sách (policy). Với cơ chế điều khiển bằng tay, người dùng phải chịu trách nhiệm cho việc quan sát nhu cầu của tải và tăng giảm tài nguyên tương ứng thông qua các cài đặt trong giao diện cung cấp bởi các nhà cung cấp. Ngược lại, trong cơ chế tự động, nhà cung cấp kiểm soát việc co dãn này thông qua các thảo thuận trong Service Level Agreement. Các hàng động co dãn trong cơ chế này có thể là bị

động khi dựa trên các thông tin về tài nguyên như CPU, RAM hoặc chủ động, dự đoán trước bằng các giải thuật Heuristics hoặc kĩ thuật phân tích.

Đối với các ứng dụng sử dụng dữ liệu luồng, việc tận dụng mô hình điện toán đám mây là cần thiết để giải quyết các yêu cầu đặc ra về việc xử lý thời gian và sử dụng tối ưu tài nguyên. Việc co dãn các ứng dụng này vẫn thuộc một theo hai cách theo chiều ngang và chiều dọc. Tuy vậy, cho đến hiện nay, để có thể cung cấp một giải pháp toàn diện cho các ứng dụng này vẫn cần thêm thời gian nghiên cứu và thực nghiệm để chứng minh tính hiệu quả.



Hình 0-5: Các phương pháp co dãn của điên toán đám mây

Tuy vậy, mô hình điện toán đám mây có nhược điểm là độ trễ cao truyền thông giữa các thiết bị tầng dưới và đám mây, tốc độ tính toán thấp và thiếu linh hoạt trong các hệ thống yêu cầu thời gian thực và ứng dụng có tính linh hoạt cao. Các nghiên cứu gần đây đã đề xuất các kiến trúc mới để giải quyết các nhược điểm này, đó là điện toán biên (Edge Computing).

Mô hình điện toán biên

Sự cần thiết của mô hình Điện toán biên này bắt nguồn từ hai lý do chính, đó là: (1) Các giới hạn về băng thông trong kiến trúc mạng hiện nay và (2) Lượng dữ liệu khổng lồ do các thiết bị IoT tạo ra. Lý do thứ nhất bắt nguồn từ sự giới hạn về mặt phần cứng trong hạ tầng mạng đang được triển khai. Ví dụ, kết nối bằng cáp quang có băng thông cao nhất đạt 1 Gbps và kết nối WiFi có thể đạt 40 Mbps. Trong khi đó, một máy bay Boeing 787 được trang bị 6000 cảm biến có thể tạo ra đến 2.5 Terabytes dữ liệu trong một ngày. Như vậy, việc truyền thông lượng dữ liệu khổng

lồ này lên Đám mây qua các kết nối có dây hay không dây hiện nay là không khả thi. Kể cả trong trường hợp có truyền thông thành công cũng sẽ tốn một khoảng thời gian rất lớn mà các máy bay có yêu cầu xử lý thời gian thực vì liên quan đến sinh mạng con người. Đối với lý do thứ hai bắt nguồn từ thực tế cuộc sống. Một nhà thông minh được dự đoán có thể tạo ra đến một Gigabytes dữ liệu trong một tháng. Việc xứ lý và lưu trữ lượng dữ liệu này như thế nào là một vấn đề khó đối với kiến trúc thông thường.

Trong kiến trúc IoT tham khảo, mô hình điện toán biên chính là nâng cấp của tầng Gateway còn đối với môi hình truyền thống là sự nâng cấp của tầng Network. Nghiên cứu [9] định nghĩa điện toán biên là một mô hình thực hiện các tính toán, xử lý dữ liệu ở biên giới của kiến trúc mạng, gần với nơi mà dữ liệu được sinh ra. Cơ sở của ý tưởng này là việc xử lý dữ liệu phải gần với nguồn dữ liệu. Kết quả của việc này là giảm lượng dữ liệu phải chuyển về phía Đám mây, dẫn đến không còn việc bị nghẽn dữ liệu trong hạ tầng mạng. Đồng thời, mô hình này có thể tăng tính bảo mật và tối ưu hóa tài nguyên sử dụng trong các thiết bị trên Đám mây.

1.2.2 Luồng dữ liệu

Luồng dữ liệu (data streams) là loại dữ liệu được sản sinh ra với số lượng lớn, liên lục, và nhanh chóng [10]. Thành phần này được sinh ra từ các hệ thống sử dụng số lượng lớn các thiết bị thông minh và sự kết nối giữa các thiết bị này. Một số ví dụ có thể được kể đến như các ứng dụng về dự báo tài chính, mạng cảm biến, hay hệ thống quản lí truyền thông (telecommunication management). Đặc biệt, sự nổi lên của các hệ thống IoT hiện nay càng làm tăng sự vai trò của loại dữ liệu này. Khác với loại dữ liệu truyền thống có dung lượng gần như không thay đổi và được sử lý theo các bước từ lưu trữ đến phân tích, luồng dữ liệu có các đặc điểm:

- Yều cầu xử lý thời gian thực với độ trễ là bé nhất. Đây là đặc điểm quan trọng nhất do nguồn gốc sản sinh của dữ liệu này từ đa dạng nguồn dữ liệu online, dẫn đến hệ quả là sự không đồng nhất về thời gian xuất hiện và gần như không thể dự đoán được khi nào có yêu cầu xử lý.
- Sự xuất hiện của loại dữ liệu này là ngẫu nhiên, không biết trước thứ tự xuất hiện.
- Độ lớn của loại dữ liệu này là không cố định,
- Khi dữ liệu đã được xử lý, việc truy xuất lại kết quả xử lý rất khó khăn trừ khi được lưu trữ theo cách đối với loại dữ liệu truyền thống. Tuy nhiên loại dữ liệu

này về mặt số lượng là quá lớn nên việc lưu trữ chỉ có thể áp dụng cho một phần nào đó.

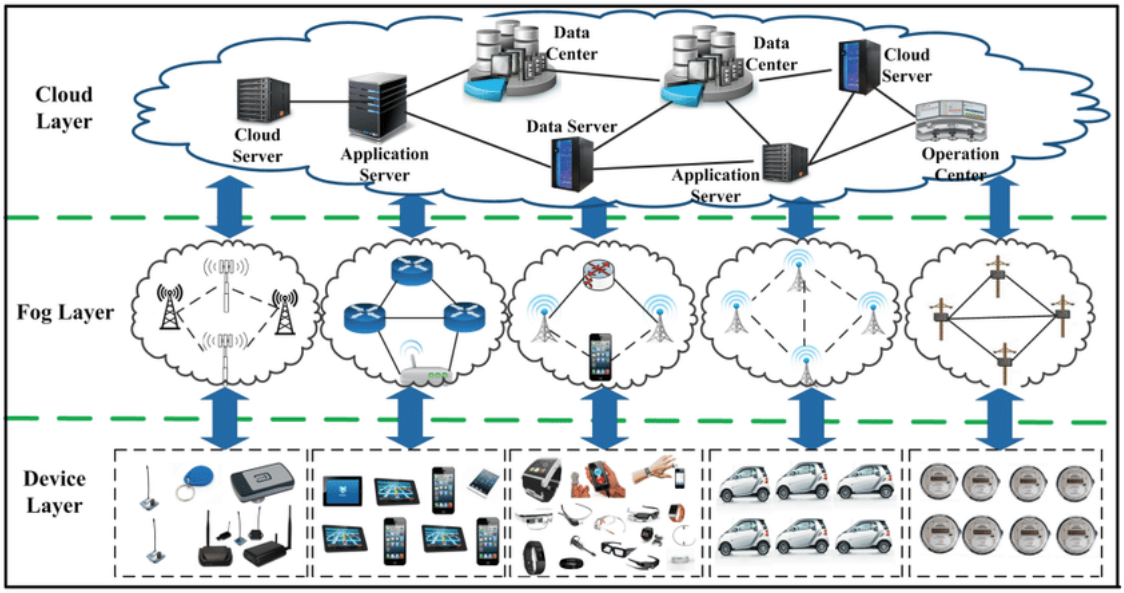
Từ khía cạnh của *Big Data* với ba đặc trưng là dung lượng (Volume), vận tốc (Velocity), và tính đạ dạng (Variety), luồng dữ liệu là thành phần thể hiện rõ nhất khía cạnh thứ hai với yêu cầu dữ liệu được tạo ra và xử lý theo thời gian thực hoặc gần thời gian thực nhất. Một số giải pháp đã được nghiên cứu và phát triển trong cả học thuật và thương mại, hệ thống tích hợp giải pháp này được gọi là *Stream Processing Engines*, tạm dịch là *Cơ chế giải quyết luồng dữ liệu*. Theo [11], cơ chế giải quyết này đã trải qua bốn giai đoạn phát triển:

- Giai đoạn 1 là Data Stream Management System là môt mở rộng của hệ quản trị cơ sở dữ liệu truyền thống (DBMS). Giai đoạn này cho phép sử dụng loại ngôn ngữ gần giống SQL để thực hiện các tác vụ truy xuất. Ví dụ là Aurora [12].
- Giai đoạn 2 là các cơ chế xử lý phân tán (distributed execution) bằng cách phân chia thành các thực thể để thực hiện xử lý và cách thực thể này có thể giao tiếp với nhau. Mô hình này tồn tại hạn chế ở khía cạnh cân bằng tải đến các thực thể này và việc quản lý các tài nguyên được phân cho các thực thể. Ví dụ là Borealis [13].
- Giai đoạn 3, các hệ thống tận dụng các ưu điểm của các hệ thống phân tán và tích hợp thêm các chức năng mà người dùng mong muốn (User Defined Functions). Các tính năng này thuộc một trong hai yêu cầu xử lý sau đây [14]:
  - Liên tục, từng dữ liệu một (one-at-a-time) thông qua sơ đồ kết nối các tiến trình (operators).
  - Rời rạc thành các mẻ dữ liệu nhỏ (micro-batch) để xử lý.
- Giai đoạn 4 đang nổi lên những năm gần đây với ý tưởng luồng dữ liệu được xử lý tại biên giới của mạng, gần với nơi sản sinh dữ liệu đồng thời trong quá trình dữ liệu được truyền về nơi xử lý tập trung [15], [16]. Điều này là đặc biệt quan trọng trong các ứng dụng IoT.

1.3 Tổng quan tình hình nghiên cứu

Trên thế giới, các công cụ và nền tảng đã được đề xuất để thực hiện những nhiệm vụ phân tích dữ liệu luồng quy mô lớn một cách hiệu quả. Đa phần các công cụ này triển khai cách tiếp cận dòng dữ liệu (dataflow) nơi luồng dữ liệu đang tới sẽ được điều chuyển thông qua một sơ đồ kết nối của các tiến trình điều hành (operators)

được đặt ở những nút (nodes) phân tán khác nhau. Những operators này sẽ thực thi những phép tính đại số cơ bản hoặc thực thi những hàm chức năng phức tạp hơn đã được định nghĩa trước. Một số nền tảng khác lại thực hiện việc rời rạc hóa các luồng dữ liệu tới bằng cách tạm thời lưu chúng trong một khoảng thời gian cửa sổ rất ngắn và sau đó thực hiện vi xử lý hàng loạt (micro-batch) ở những nơi đang tạo ra việc tính toán phân tán trên những dữ liệu đã được lưu trước đó. Cách tiếp cận thứ hai này giúp cho các công cụ xử lý dữ liệu luồng phân tán cải thiện khả năng chống chịu lỗi và khả năng thích ứng với sự mở rộng quy mô của tải dữ liệu bằng cách xử lý các tác vụ tốn quá nhiều thời gian hoặc bị lỗi một cách hiệu quả hơn. Cũng trên phương diện cải thiện khả năng thích ứng với sự mở rộng quy mô của tải dữ liệu, rất nhiều nền tảng xử lý luồng dữ liệu đã được triển khai trên môi trường đám mây (Cloud) để tận dụng đặc trưng nổi bật của dạng điện toán này đó là khả năng cung ứng và triển khai tài nguyên có thể co dãn (elasticity). Tính co dãn (hay đàn hồi) của điện toán đám mây, khi được khai thác phù hợp, là muốn nói đến khả năng của một đám mây cho phép một dịch vụ phân phối thêm tài nguyên hoặc giải phóng những tài nguyên không dùng đến theo nhu cầu để đáp ứng được sự thay đổi về tải của ứng dụng. Mặc dù nhiều nỗ lực đã được thực hiện để làm cho việc xử lý luồng dữ liệu có khả năng co dãn nhiều hơn, rất nhiều vấn đề liên quan vẫn cần phải được giải quyết. Những thách thức này bao gồm bài toán tối ưu về phân phối tác vụ xử lý luồng dữ liệu trên những tài nguyên khả dụng, bài toán xác định nút cổ chai gây tắc nghẽn hay việc làm sao để thích nghi các ứng dụng với sự thay đổi về tài nguyên. Những vấn đề này sẽ càng trở nên trầm trọng nếu những dịch vụ được triển khai chỉ là một phần của một kiến trúc lớn hơn chứa đựng nhiều mô hình thực thi khác nhau (ví dụ như kiến trúc Lambda [14], mô hình luồng công việc (workflow), hoặc mô hình kết nối tài nguyên quản lý với những khái niệm lập trình trừu tượng bậc cao [17]) hoặc trong môi trường lai có sử dụng tài nguyên của cả điện toán đám mây và điện toán sương mù (Fog computing) như trong Hình 1-6.



Hình 0-6: Mô hình của điện toán sương mù

Trong thời gian gần đây, một vài nền tảng và kiến trúc đã được đề xuất để thực hiện việc xử lý dữ liệu luồng sử dụng những tài nguyên được đặt ngay tại phần biên của Internet nơi gần hơn với các thiết bị đầu cuối (edge computing). Một trong số những công nghệ theo hướng này đang thu hút được nhiều sự chú ý hiện nay là điện toán sương mù được giới thiệu bởi Cisco từ năm 2012 [18]. Những nghiên cứu gần nhất về công nghệ này cố gắng khai thác phần biên của Internet để đặt những thành phần xử lý dữ liệu luồng cụ thể vào trong những trung tâm dữ liệu nhỏ (hoặc siêu nhỏ - micro data centers) được đặt gần nơi mà dữ liệu đã được tạo ra [9]. Những thành phần này sẽ chuyển một phần dữ liệu đã được tiền xử lý lên trên đám mây để sau đó được xử lý hàng loạt (batch processing) nếu có nhu cầu [19]. Hướng thứ hai lại tận dụng tài nguyên của chính những thiết bị di động hoặc những thiết bị thông minh được phân tán trong điện toán sương mù để thực hiện việc xử lý luồng dữ liệu [20]. Bằng cách này hay cách khác, mục đích cuối cùng của những nền tảng này là cố gắng đặt một phần việc phân tích dữ liệu tại phần biên của Internet để giảm thiểu việc phải truyền tải khối lượng lớn dữ liệu từ nguồn tới các trung tâm dữ liệu trên đám mây, cải thiện độ trễ đầu cuối, hoặc giảm tải cho đám mây bằng cách đưa một số bước phân tích dữ liệu phù hợp về phía cận biên. Mặc dù đã có nhiều nỗ lực để xây dựng và phát triển những cơ sở hạ tầng xử lý luồng dữ liệu dạng này, hầu hết các đề tài nghiên cứu hoặc dự án hiện tại mới chỉ dừng lại ở mức độ xây dựng khái niệm và kiến trúc mà chưa có những sản phẩm hoàn thiện được công bố [21]. Trong [22], Agarwal và các cộng sự đã giới thiệu kiến trúc 3 lớp ban đầu của điện toán sương mù

để khắc phục những nhược điểm cố hữu của kiến trúc điện toán đám mây 2 lớp. Những nghiên cứu khác khai thác phần biên của Internet bằng cách cố gắng đặt những thành phần xử lý luồng dữ liệu trong những trung tâm dữ liệu siêu nhỏ (thường được gọi là Cloudlet) gần hơn với nơi dữ liệu được sinh ra, chuyển những thông tin sự kiện lên đám mây theo khối, hoặc khai thác các thiết bị di động trong điện toán sương mù để thực hiện tiền xử lý luồng dữ liệu.

1.4 Đặt vấn đề

Dựa trên các nghiên cứu, vấn đề đặt ra của nghiên cứu này là thiết kế hệ thống xử lý luồng dữ liệu dựa trên mô hình điện toán biên và điện toán đám mây. Trong đó, hệ thống có các tính năng cụ thể như sau:

- Hệ thống phải xử lý được dữ liệu từ nhiều nguồn dữ liệu khác nhau.
- Hệ thống sử dụng tối ưu ba khía cạnh tài nguyên là hạ tầng mạng (network), hiệu năng tính toán (computing), và khả năng lưu trữ (storage).

1.5 Cấu trúc bài nghiên cứu

Nội dung của nghiên cứu này được sắp xếp như sau. Chương 1 trình bày tổng quan về tình hình nghiên cứu, từ đó xác định vấn đề cần giải quyết. Chương 2 trình bày phương pháp luận để xây dựng lên mô hình kiến trúc đề xuất. Chương 3 trình bày kết quả thực nghiệm và các thảo luận liên quan. Kết luận và các hướng phát triển tương lại được trình bày ở chương 4.

2. CHƯƠNG 2: CÁC PHƯƠNG PHÁP XỬ LÝ LUỒNG DỮ LIỆU VÀ MÔ HÌNH ĐỀ XUẤT

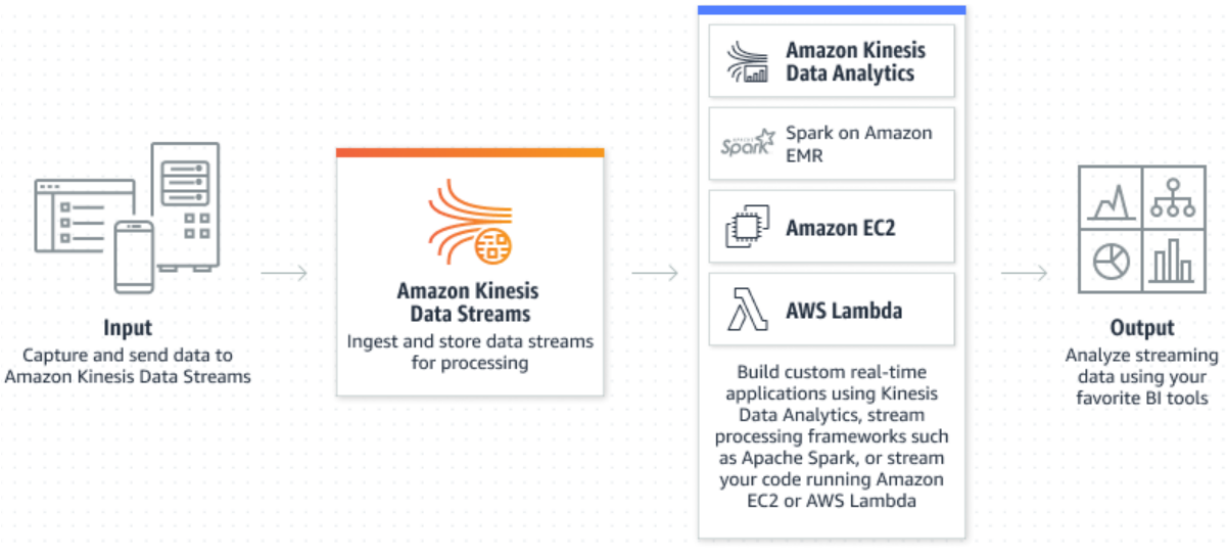
Trên thế giới hiện nay, các phương pháp xử lý luồng dữ liệu phụ thuộc nhiều vào hạ tầng phần cứng, hệ quả là xu hướng sử dụng các giải pháp do các nhà cung cấp Cloud cung cấp như Amazon, Google, hay Microsoft. Nghiên cứu này thử nghiệm và khảo sát phương pháp do Amazon Kinestic và Google Dataflow. Từ đó, chương này sẽ đề xuất mô hình xử lý luồng dữ liệu cho các ứng dụng IoT dựa trên mô hình kiến trúc điện toán đám biên và điện toán đám mây.

2.1 Các phương pháp xử lý luồng dữ liệu

2.1.1 Amazon Kinesis

Amazon Kinesis Data Streams[3] là một dịch vụ cung cấp bởi Amazon nhằm đơn giản hóa quá trình tiêu thụ, xử lý, và lưu trữ luồng dữ liệu cho bất kì ứng dụng nào với phạm vi khác nhau. Dịch vụ này cho phép người dùng xây dựng các ứng dụng có thể đáp ứng nhu cầu thời gian thực nhờ sử dụng các framework xử lý luồng và lưu trữ các luồng này vào các nguồn dữ liệu khác nhau (data stores). Các ứng dụng phổ biến khi sử dụng dịch vụ này bao gồm:

- Giám sát và báo cáo các dịch vụ yêu cầu thời gian thực.
- Phân tích dữ liệu thời gian thực.
- Các dịch vụ xử lý luồng dữ liệu phực tạp



Hình 2-1: Kiến trúc dịch vụ Amazon Kinesis

[3] https://aws.amazon.com/kinesis/

Hình cho thấy kiến trúc của các dự án sử dụng dịch vụ này. Điều đáng chú ý trong kiến trúc là thành phần dùng để xử lý luồng dữ liệu chính là Amazon Elastic Cloud Computing (Amazon EC2)[4]. Đây là một dịch vụ khác của Amazon cung cấp các tài nguyên tính toán như CPU, RAM và bộ nhớ; tổng hợp các tài nguyên này gọi chung là instance, có thể hiểu là một loại phần cứng cụ thể để triển khai dịch vụ người dùng. Amazon EC2 sẽ triển khai qua việc khởi tạo và chạy các instance; để tăng tính ổn đinh (stability) và tính khả dụng (availability), các dự án trong triển khai thực tế sẽ được triển khai nhiều instance cho một dịch vụ và dùng LoadBalancing[5] để phân phối yêu cầu người dùng đến các instance này. Điều này nhằm mục đích đáp ứng nhu cầu thay đổi phía tải, đặc biệt là trong các ứng dụng xử lý luồng dữ liệu có nhu cầu thay đổi thất thường.

Để tự động hóa việc quản lý các instance của Amazon EC2, một dịch vụ khác của Amazon là AutoScaling[6] sẽ tăng giảm các tài nguyên một cách tự động. Nguyên lý hoạt động dựa trên các cài đặt của người dùng như tăng giảm theo lịch trình (scheduled scaling), tăng giảm theo yêu cầu phía tải (dynamic scaing), và tăng giảm được dự báo trước (predictive scaling). Nhìn chung, cơ chế hoạt động dựa trên việc quan sát các thông số của instance thông qua CloudWatch[7], nếu vi phạm các cài đặt của người dùng thì các instance sẽ đươợc triển khai thêm.

2.1.2 Google Dataflow

Google Dataflow[8] là một dịch vụ của Google Cloud cung cấp xử lý đồng thời dữ liệu luồng (strem processing) và xử lý theo mẻ (batch processing). Nguyên lý hoạt động của dịch vụ này dựa trên một pipeline bao gồm bốn bước:

- Planning: Dựa trên yêu cầu từ phía người dùng, các ảnh hưởng từ nguồn dữ liệu (data sources), bước này sẽ quyết định thay đổi pipeline theo hướng tăng hoặc giảm cả về một phạm vi và hiệu năng để đáp ứng yêu cầu đó. Nhờ việc theo dõi này, dịch vụ này đạt được tính availability cao, có thể nhanh chóng khắc phục lỗi, và đánh giá tác động của hạ tầng mạng nơi hệ thống được triển khai.

[4] https://aws.amazon.com/ec2/
[5] https://aws.amazon.com/elasticloadbalancing/
[6] https://aws.amazon.com/autoscaling/
[7] https://aws.amazon.com/cloudwatch/
[8] https://cloud.google.com/dataflow

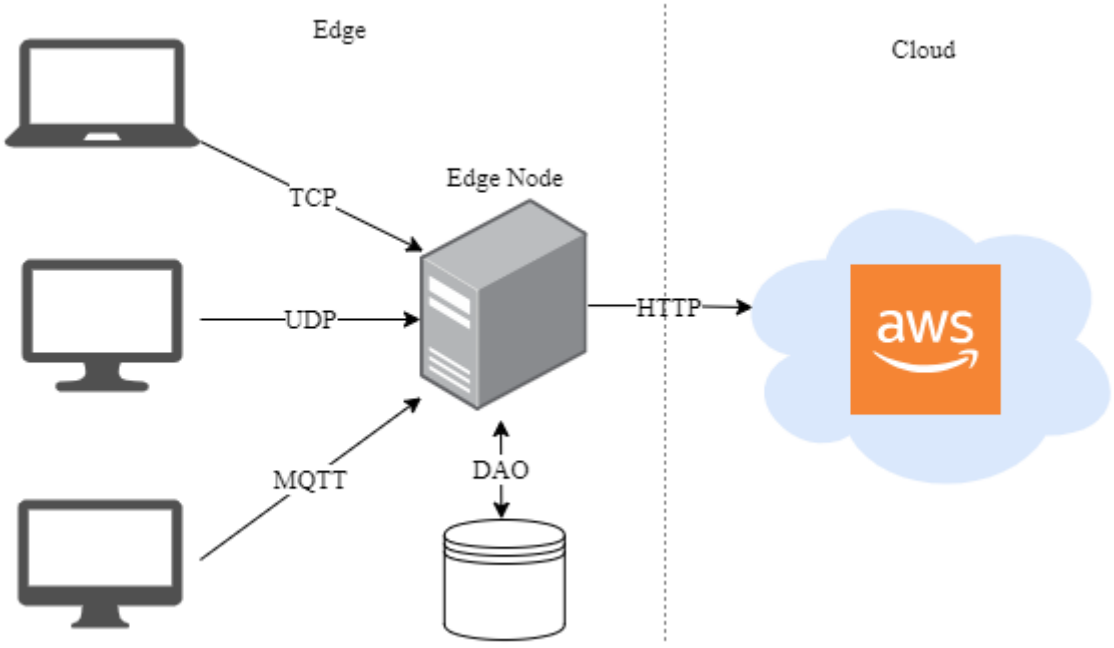
- Developing and testing: Bước này sẽ triển khai môi trường phát triển theo các phương pháp như (1) Sử dụng queues để tránh việc mất mát dữ liệu, (2) Sử dụng operators để giảm độ trễ mạng và kinh phí sử dụng, (3) Sử dụng xử lý theo mẻ để tránh việc quá tải hệ thống.
- Deploying: Bước này phục vụ ý tưởng triển khai CI/CD hiện nay cho các ứng dụng xử lý luồng và đồng thời tối ưu tài nguyên sử dụng.
- Monitoring: Giám sát hiệu năng hoạt động của pipeline để đảm bảo đáp ứng nhu cầu của người dùng.

Công nghệ mà dịch vụ này sử dụng chính là Kubernetes[9] để hiện thực hóa pipeline xử lý luồng dữ liệu. Kubernetes là một dịch vụ quản lý triển khai container với container là các môi trường cần thiết để triển khai một tác vụ nào đó. Lý do sử dụng Kubernetes vì công nghệ này quản lý các container, mà các container có khả năng gia tăng và giảm đi nhanh chóng nhằm đáp ứng nhu cầu thất thường của các ứng dụng xử lý luồng dữ liệu.

2.2 Mô hình xử lý luồng dữ liệu đề xuất

2.2.1 Kiến trúc đề xuất

Dưa vào yêu cầu bài toán và các tài liệu tham khảo, kiến trúc hệ thống được xây dựng như sau:



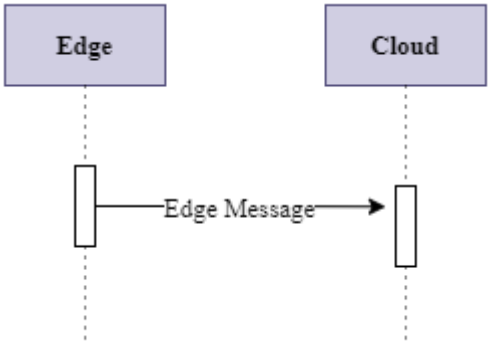
Hình 2-2: Mô hình kiến trúc đề xuất

[9] https://kubernetes.io/

Tầng Edge là nơi triển khai mô hình Điện toán biên. Trong kiến trúc truyền thống, các thiết bị đầu cuối khi sản sinh ra dữ liệu sẽ ngay lập gửi lên các tầng phía trên, nhưng ở kiến trúc này, lượng dữ liệu này sẽ đi qua các Fog nodes để thực hiện xử lý một phần dữ liệu này. Tầng Edge phải làm rõ khái niệm về kiến trúc và cách triển khai thực tế. Ví dụ ở Hình 2.1 các fog nodes là IVSS và Face Camera vì các thiết bị này có đủ sức mạnh tính toán để vừa sản sinh dữ liệu và thực hiện các phương pháp co dãn. Trong khi đó đối với các thiết bị như Sensors hay Acutators có sức mạnh tính toán thấp chỉ có thể sản sinh dữ liệu thì Gateway hoạt động Fog nodes. Mục tiêu của nghiên cứu này là áp dụng các phương pháp xử lý luồng dữ liệu các thiết bị Edge nodes này, cụ thể là các thuật toán nén bao gồm Deflate, LZMA, LZ4, và Bzip2.

Tầng Cloud hoạt động như một Điện toán đám mây thông thường. Nghiên cứu này sử dụng điện toán đám mây cung cấp bởi AWS.

2.2.2 Sơ đồ giao tiếp



Các thành phần của hệ thống đề xuất sẽ giao tiếp qua bản tin Edge Message có nội dung bao gồm các thành phần sau đây:

Key	Type	Values	Description
id	String	UUID	Unique identifier
fileName			
fileType	String	Ex: mp4, png, jpeg, …	Information about content
fileSize	String	In bits.	The length of the content
timeSent	Long Integer	Ex: 1655396252	Unix timestamp
content	String		The actual delivered information

hardware	String	Ex: Sensors, Actuator, Face Camera, …	Information about hardware resource

Trong các thành phần của bản tin, khóa content là quan trọng nhất vì giá trị của khóa này sẽ được tạo ra một cách tự động, có thể là video, ảnh, hay chỉ bảo gồm các chuỗi kí tự. Sự thay đổi về mặt nội dung để có thể đạt được sử thay đổi dung lượng của luồng dữ liệu, từ đó kiểm tra phản ứng của hệ thống.

2.3 Xây dựng mô hình điện toán biên

2.3.1 Lý thuyết áp dụng

Đề tài mô phỏng ứng dụng luồng dữ liệu lớn bằng các công cụ phần mềm, cụ thể là Docker và Kubernetes vì việc xây dựng phần cứng để tạo ra lượng dữ liệu lớn một cách nhanh chóng là không khả thi.

Docker

Docker là một nền tảng phần mềm cho việc phát triển, đóng gói và chạy ứng dụng trong các môi trường cô lập gọi là containers. Docker hỗ trợ cho các dịch vụ khác có thể phát triển và triển khai một cách nhanh chóng vì nó cho phép tách biệt giữa phần cơ sở hạ tầng cần triển khai lên và dịch vụ chạy trên cơ sở hạ tầng đó. Để sử dụng được Docker, có hai thứ người dùng cần biết là containers và images

Containers là môi trường chạy ứng dụng đóng gói. Chúng bao gồm mã ứng dụng, thư viện và các phụ thuộc cần thiết vì vậy nếu một dịch vụ sử dụng containers để chạy thì dịch vụ đó hoàn toàn không phải phụ thuộc vào nơi mà nó được chạy hoặc triển khai

Images là các gói đóng gói chứa tất cả các thành phần cần thiết để chạy một container. Một image bao gồm các tệp cấu hình, mã ứng dụng và tất cả các thư viện cần thiết, gọi chung là filesystem. Image được sử dụng để tạo ra các container chạy ứng dụng. Chúng có thể được chia sẻ và triển khai trên nhiều máy chủ và môi trường.

Sử dụng công cụ Docker sẽ nhanh chóng mô phỏng được một client trong thực tế mà không cần quan tâm đến nơi triển khai client đó.

Kubernetes

Đây là một dịch vụ được Google phát triển và tung ra thị trường dưới dạng mã nguồn mở vào năm 2014. Dịch vụ này dùng để quản lý các dự án, các dịch vụ được triển khai dưới dạng các containers. Cách triển khai dạng containers trở nên phổ biến hiện nay vì yêu cầu phát triển nhanh và nhanh chóng triển khai đến tay người dùng (CI/CD). Để hiểu rõ hơn điều này hãy cùng thử so sánh với các cách triển khai khác.

Bảng 2-1: So sánh các phương pháp triển khai

Triển khai truyền thống	Triển khai bằng máy ảo	Triển khai bằng container
Một dịch vụ trên cùng một máy tính	Nhiều dịch vụ trên cùng một máy tính vật lý	Nhiều dịch vụ trên cùng máy tính vật lý
	Filesystem bao gồm cả OS	Filesystem không bao gồm OS
		Nhanh chóng phát triển, triển khai, thu hồi dịch vụ

Khi sử dụng cách triển khai bằng các containers, kubernetes sẽ đảm bảo việc hệ thống triển khai sẽ không bao giờ ở trạng thái bảo trì vì bất cứ khi nào môt container bị hỏng thì ngay lập tức kubernetes sẽ thực hiện thay thế bằng một container khác. Thêm vào đó, kubernetes giúp có thể mở rộng hệ thống theo chiều ngang để tận dụng tối đa phần cứng đang có và chức năng cân bằng tải khi có rất nhiều yêu cầu truy cập đến dịch vụ của bạn. Tuy vậy kubernetes có một điểm yếu là các dịch vụ chạy trên nó sẽ không theo dõi được log, ngoài ra cách kubernetes hoạt động cũng không theo một trình tự nhất định vì mục tiêu chính của kubernetes là làm sao đưa hệ thống đến một trạng thái mong muốn.

2.3.2 Thiết kế phần mềm

Điện toán biên thực hiện ba tác vụ là (1) nhận dữ liệu từ các client thông qua ba phương thức (MQTT, UDP, và TCP), (2) nén các dữ liệu nhân được bằng các phương pháp nén (Deflate, LZMA, LZ4, và Bzip2), và (3) gửi các dữ liệu được nén lên tầng Cloud.

Nhận dữ liệu từ phía clients

Phần mềm sẽ tiếp nhận các yêu cầu từ ba giao thức TCP, UDP và MQTT nên dựa vào nguyên lý hoạt động của các giao thức sẽ có cách xử lý khác nhau.

Đối với giao thức TCP, sẽ gồm hai giai đoạn để xử lý dữ liệu, đó là thiết lập liên kết và phân luồng xử lý. Giai đoạn thứ nhất là thiết lập liên kết giữa phía client và phần mềm, giai đoạn này phải đảm bảo phầm mềm đã xác định được client và cho phép liên lạc.

```
def start(self):
    '''Start TCP Server'''
    # A server must perform the sequence: socket(), bind(), listen(),
accept() (repeat accpet() for multiple connections)
    # AF_INET: a hostname in internet domain or an IPv4 address
    # SOCK_STREAM: stream data through socket
```

```
    self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        self._socket.bind((self.__host, self.__port))
    except socket.error as e:
        print("[TCP Server] " + str(e))

    # Enable the server to accpet connections and specifies the number of
allowed unaccepted connections before refusing new connections
    self._socket.listen(5)

    print("[TCP Server] Server is starting ...")
    print(f"[TCP Server] Server is listening on {self.__host}")
```

Giai đoạn thứ hai là xử lý dữ liệu khi có nhiều kết nối đến với phần mềm. Phần mềm này có thể đồng thời nói chuyện với nhiều clients nhưng khi một client ngừng nói thì phần mềm không thể ngừng nói chuyện với đồng thời tất cả các client. Vì vậy, code phần mềm phải sinh ra các luồng để nói chuyện với từng client tương ứng và khi một client ngắt kết nối thì giao tiếp giữa Server và client cũng được loại bỏ để đảm bảo sự tối ưu về mặt phần cứng.

```
def run(self):
    '''Run TCP Server'''
    while True:
        self.__conn, self.__addr = self._socket.accept()
        self.__thread = threading.Thread(target=self.handle_client,
args=(self.__conn, self.__addr))
        self.__thread.start()
        self._count = self._count + 1

    self._socket.close()
```

Đối với giao thức UDP, vốn không cần đảm bảo sự tin cậy nên chỉ cần thiết lập liên kết với các client là có thể thực hiện được giao tiếp.

```
def start(self):
    '''Start UDP Server'''
    self._socket = socket. socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        self._socket.bind((self.__host, self.__port))
    except socket.error as e:
        print("[UDP Server] " + str(e))

    print("[UDP Server] Server is starting ...")
```

Đối với giao thức MQTT, bằng cách sử dụng library hỗ trợ paho, việc thiết lập đến broker được giảm tải. Hệ thống chỉ cần phải xác định địa chỉ broker liên kết đến.

```
def start(self):
    '''Start MQTT Server'''
    # Callback function
    self.__conn.on_connect = self.on_connect
    self.__conn.on_connect_fail = self.on_connect_fail
    self.__conn.on_disconnect = self.on_disconnect
    self.__conn.on_message = self.on_message
    self.__conn.on_publish = self.on_publish
    self.__conn.on_subscribe = self.on_subscribe

    # Start MQTT Server
    self.__conn.connect(self.__broker, self.__port, 60)
    print("[MQTT Server] Server is starting ...")
```

Sử dụng các thuật toán nén

Trong Python, thư viện zlib hỗ trợ việc nén và giải nén dữ liệu với thuật toán Deflate, từ đó xây dựng được Class Deflate, với các tham số đầu vào:

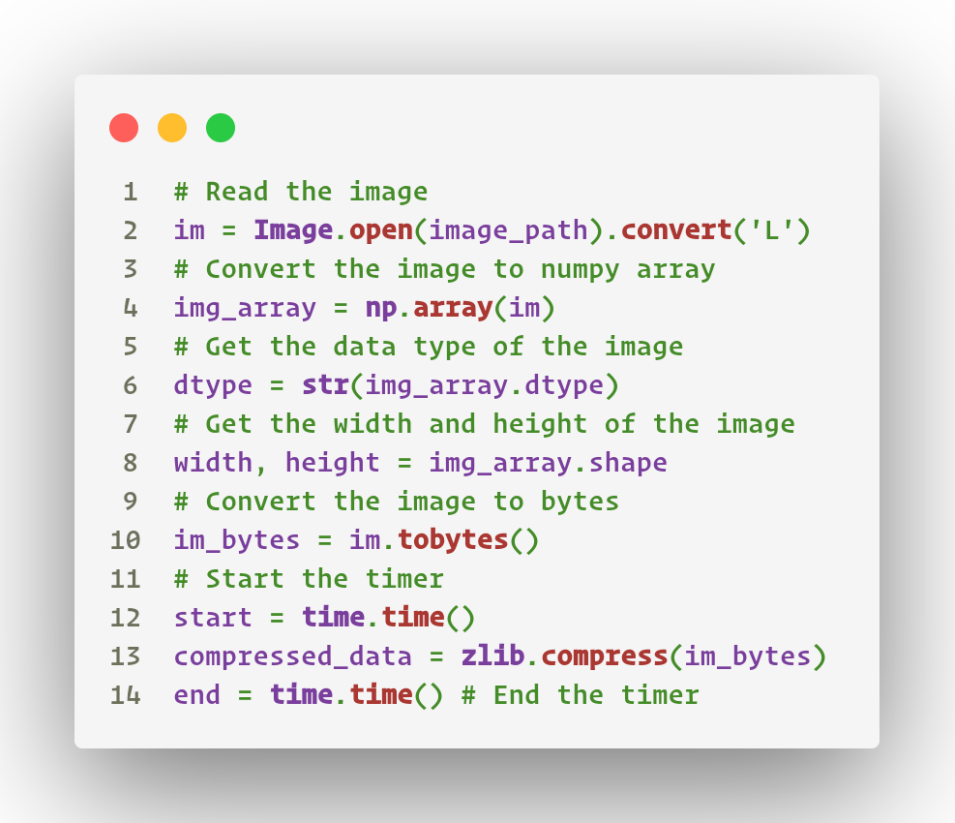
- files: Danh sách các tệp (mặc định là một danh sách rỗng)
- folder_name: tên của folder (mặc định)

```
1  class Deflate:
2      def __init__(self, files: list = [], folder_name: str = '') -> None:
3          '''
4              Constructor of the class
5              Parameters:
6                  files (list, optional): List of files. Defaults to [].
7                  folder_name (str, optional): Folder name. Defaults to ''.
8
9          '''
10         # Storing the files and folder names
11         self.__files = files
12         self.__folder_name = folder_name
```

Hình 2-3: Class Deflate

Khi dùng class Deflate để nén một ảnh, đầu tiên dùng Image trong thư viện PIL để mở ảnh và chuyển sang kiểu ảnh xám (gray). Sau đó lấy một vài thông tin về ảnh như dtype(dạng dữ liệu của ma trận), width (chiều rộng), height(chiều cao). Để sử dụng được hàm compress trong thư viện zlib, thì ảnh đầu vào cần dưới dạng bytes, do đó sử dụng hàm tobytes() để đổi kiểu dữ liệu. Từ dòng 8 đến dòng 10, chúng ta ước lượng thời gian mà Deflate nén ảnh đầu vào. Cuối cùng lưu dữ liệu được nén dưới tập '.df' với một quy tắc đặt tên như sau name_dtype_width_height.df.

Khi dùng Class này để nén nhiều ảnh, phần mềm sẽ duyệt qua các file dữ liệu đầu vào (self.__files). Đầu tiên lấy tên file (dòng 2), chúng ta tạo ra đường dẫn từ ngoài thư mục đến tệp ở trong tệp data đầu vào (dòng 3). Tương tự như nén một ảnh, ảnh sẽ được đọc bằng thư viện PIL, lấy các thông số (dtype, width, height), chuyển sang kiểu dữ liệu bytes, ước lượng thời gian nén một ảnh và lưu dữ liệu đã nén với quy tắc name_dtype_width_height.df.



```
1  # Read the image
2  im = Image.open(image_path).convert('L')
3  # Convert the image to numpy array
4  img_array = np.array(im)
5  # Get the data type of the image
6  dtype = str(img_array.dtype)
7  # Get the width and height of the image
8  width, height = img_array.shape
9  # Convert the image to bytes
10 im_bytes = im.tobytes()
11 # Start the timer
12 start = time.time()
13 compressed_data = zlib.compress(im_bytes)
14 end = time.time() # End the timer
```

Hình 2-4: Sử dụng Class Deflate để nén một ảnh

```python
for file in self.__files:
    # Get the name of file
    name = file.split('.')[0]
    # Concatnate the path of the file
    image_path = os.path.join(BASE_PATH, self.__folder_name, file)
    # Read the image
    im = Image.open(image_path).convert('L')
    # Convert the image to numpy array
    img_array = np.array(im)
    # Get the data type of the image
    dtype = str(img_array.dtype)
    # Get width and height of the image
    width, height = img_array.shape
    # Convert the image to bytes
    im_bytes = im.tobytes()
    # Start the compression
    start = time.time()
    compressed_data = zlib.compress(im_bytes)
    end = time.time() # End the compression
```

Hình 2-5: Sử dụng Class Deflate để nén nhiều ảnh

Trong Python, thư viện lzma hỗ trợ việc nén và giải nén ảnh để xây dựng Class LZMA. Tương tự với class Deflate, class LZMA cũng có hàm khởi tạo tương tự. Giống như với hàm thủ tục compress của thuật toán Deflate ở phần trước, hàm thủ tục nén của class LZMA cũng có đầu vào là image_path và csv_path và cũng chia làm hai phần

Khi nén một ảnh và nhiều ảnh, các bước sẽ tương tự như ở trong thuật toán Deflate tuy nhiên chúng ta lưu tệp nén dưới dạng name_dtype_width_height.lzma

```python
class LZMA:
    def __init__(self, files: list = [], folder_name: str = '') -> None:
        '''
            Constructor of the class
            Parameters:
                files (list, optional): List of files. Defaults to [].
                folder_name (str, optional): Folder name. Defaults to ''.

        '''
        # Storing the files and folder names
        self.__files = files
        self.__folder_name = folder_name
```

Hình 2-6: Class LZMA

```python
# Read the image
im = Image.open(image_path).convert('L')
# Convert the image to numpy array
img_array = np.array(im)
# Get the data type of the image
dtype = str(img_array.dtype)
# Get the width and height of the image
width, height = img_array.shape
# Convert the image to bytes
im_bytes = im.tobytes()
# Start the timer
start = time.time()
compressed_data = lzma.compress(im_bytes)
end = time.time() # End the timer
```

Hình 2-7: Sử dụng Class LZMA để nén một ảnh

```python
for file in self.__files:
    # Get the name of file
    name = file.split('.')[0]
    # Concatnate the path of the file
    image_path = os.path.join(BASE_PATH, self.__folder_name, file)
    # Read the image
    im = Image.open(image_path).convert('L')
    # Convert the image to numpy array
    img_array = np.array(im)
    # Get the data type of the image
    dtype = str(img_array.dtype)
    # Get width and height of the image
    width, height = img_array.shape
    # Convert the image to bytes
    im_bytes = im.tobytes()
    # Start the compression
    start = time.time()
    compressed_data = lzma.compress(im_bytes)
    end = time.time()  # End the compression
```

Hình 2-8: Sử dụng Class LZMA để nén nhiều ảnh

Trong Python, thư viện lz4 hỗ trợ việc nén và giải nén sử dụng thuật toán LZ4 để tạo Class LZ4. Các bước nén một ảnh và nhiều ảnh cũng tương tự như trên. Kiểu dữ liệu nén được lưu dưới dạng name_dtype_width_height.lz4

```python
class LZ4:
    def __init__(self, files: list = [], folder_name: str = '') -> None:
        '''
            Constructor of the class
            Parameters:
                files (list, optional): List of files. Defaults to [].
                folder_name (str, optional): Folder name. Defaults to ''.

        '''
        # Storing the files and folder names
        self.__files = files
        self.__folder_name = folder_name
```

Hình 2-9: Class LZ4

```python
# Read the image
im = Image.open(image_path).convert('L')
# Convert the image to numpy array
img_array = np.array(im)
# Get the data type of the image
dtype = str(img_array.dtype)
# Get the width and height of the image
width, height = img_array.shape
# Convert the image to bytes
im_bytes = im.tobytes()
# Start the timer
start = time.time()
compressed_data = lz4.compress(im_bytes)
end = time.time()  # End the timer
```

Hình 2-10: Sử dụng Class LZ4 để nén một ảnh

```python
for file in self.__files:
    # Get the name of file
    name = file.split('.')[0]
    # Concatnate the path of the file
    image_path = os.path.join(BASE_PATH, self.__folder_name, file)
    # Read the image
    im = Image.open(image_path).convert('L')
    # Convert the image to numpy array
    img_array = np.array(im)
    # Get the data type of the image
    dtype = str(img_array.dtype)
    # Get width and height of the image
    width, height = img_array.shape
    # Convert the image to bytes
    im_bytes = im.tobytes()
    # Start the compression
    start = time.time()
    compressed_data = lz4.compress(im_bytes)
    end = time.time()  # End the compression
```

Hình 2-11: Sử dụng Class LZ4 để nén nhiều ảnh

Cuối cùng là thuật toán nén Bzip2 cũng tương tự như các thuậ toán trên. Kiểu dữ liệu nén được lưu dưới dạng name_dtype_width_height.bz2

```python
class BZ2:
    def __init__(self, files: list = [], folder_name: str = '') -> None:
        '''
            Constructor of the class
            Parameters:
                files (list, optional): List of files. Defaults to [].
                folder_name (str, optional): Folder name. Defaults to ''.

        '''
        # Storing the files and folder names
        self.__files = files
        self.__folder_name = folder_name
```

Hình 2-12: Class BZ2

```python
# Read the image
im = Image.open(image_path).convert('L')
# Convert the image to numpy array
img_array = np.array(im)
# Get the data type of the image
dtype = str(img_array.dtype)
# Get the width and height of the image
width, height = img_array.shape
# Convert the image to bytes
im_bytes = im.tobytes()
# Start the timer
start = time.time()
compressed_data = bz2.compress(im_bytes)
end = time.time()  # End the timer
```

Hình 2-13: Sử dụng Class BZ2 để nén một ảnh

```python
for file in self.__files:
    # Get the name of file
    name = file.split('.')[0]
    # Concatnate the path of the file
    image_path = os.path.join(BASE_PATH, self.__folder_name, file)
    # Read the image
    im = Image.open(image_path).convert('L')
    # Convert the image to numpy array
    img_array = np.array(im)
    # Get the data type of the image
    dtype = str(img_array.dtype)
    # Get width and height of the image
    width, height = img_array.shape
    # Convert the image to bytes
    im_bytes = im.tobytes()
    # Start the compression
    start = time.time()
    compressed_data = bz2.compress(im_bytes)
    end = time.time()  # End the compression
```

Hình 2-14: Sử dụng Class BZ2 để nén nhiều ảnh

Đối với nén dữ liệu text (csv), nhìn chung các bước giải nén trong từng thuật toán là giống nhau, và quy tắc lưu tệp nén cũng giống nhau, nên chúng ta sẽ sử dụng đại diện thuật toán nén dữ liệu text là thuật toán Deflate. Cũng như nén ảnh, nén dữ liệu text cũng được chia làm hai phần gồm nén một tệp và nhiều tệp text.

Gửi các dữ liệu lên tầng Cloud

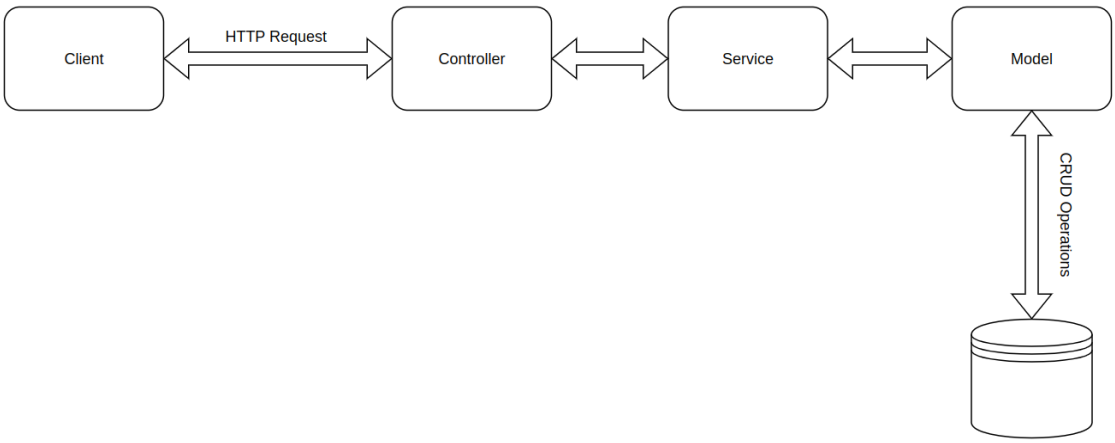
Các file được tạo ra sau quá trình nén, sẽ được đọc lần lượt và gửi lên trên Amazon Cloud thông qua giao thức HTTP. Quá trình khảo sát sẽ lần lượt từ file không nén đến các thuật toán nén được sử dụng.

```
def sendMessage(PATH, files):
    for i in range(5):
        if i == 0:
            print("uncompressed files")
        elif i == 1:
            print("deflate compressed files")
        elif i == 2:
            print('lzma compressed files')
        elif i == 3:
            print('lz4 compressed files')
        elif i == 4:
            print('bz2 compressed files')

        for file in files:
            path = PATH + '/' + file
            payload = createMessage(path, file, i)
            payload['fileSize'] = len(payload['content'])
            payload['timeSent'] = int(time.time() * 10e2)
            response = requests.post(url, json=payload, headers=headers)
```

2.4 Xây dựng mô hình điện toán đám mây

2.4.1 Thiết kế phần mềm

Để phục vụ cho việc tiếp nhận dữ liệu từ phía Edge, Cloud sử dụng một service có tên là Data Receiver được xây dựng bằng Java Spring Boot cho phép xây dựng ra các cổng API có công việc chính là giao tiếp với cơ sở dữ liệu. Hình 2-15 là kiến trúc phần được xây dựng.



Hình 2-15: Kiến trúc phần mềm trên đám mây

Thiết lập kết nối cơ sở dữ liệu

Để có thể giao tiếp được với cơ sở dữ liệu, trước hết cần thiết lập một số biến môi trường liên quan đến cơ sở dữ liệu được sử dụng là Postgresql, thông qua file có tên là application.properites

```
spring.datasource.url=jdbc:postgresql://localhost:5432/DataLogger
spring.datasource.username=myserver
spring.datasource.password=1

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update
```

Sau đó để nhận dữ liệu từ phía edge và có thể lưu được, cần một định dạng về thực thể có các thuộc tính được định nghĩa trước, thực thể sẽ được lưu trong cơ sở dữ liệu có tên là Data, gồm các thuộc tính như sau:

```java
package com.demo.DataReciever.entity;

import jakarta.persistence.*;
import lombok.*;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "data")
@Builder
public class Data {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "file_name")
    private String fileName;
    @Column(name = "file_type")
    private String fileType;
    @Column(name = "content_length")
    private int fileSize;
    @Column(name = "time_received")
    private Long timeReceived;
    @Column(name = "time_sent")
    private Long timeSent;

    @Lob
    @Column(length = 500*1024)
    private byte[] content;
}
```

- Id: Một giá trị tự tăng xác định chỉ mục của dữ liệu được truyền vào
- fileName: tên tệp dữ liệu truyền đi
- fileType: loại tệp dữ liệu được truyền đi
- fileSize: Kích cỡ dữ liệu được truyền đi
- timeReceived: Thời gian nhận được dữ liệu trên
- timeSent: Thời gian được gửi từ phía Edge
- Content: Nội dung của dữ liệu trên (tối đa 500MB)

Sau khi thiết lập xong kết nối với cơ sở dữ liệu, khi chạy service, thư viện Spring Boot Data JPA sẽ tự tạo thực thể Data trong cơ sở dữ liệu

Nhận dữ liệu được truyền từ phía Edge

Một đối tượng cần được định nghĩa phục vụ cho việc nhận dữ liệu từ phía Edge. Về cơ bản, các thuộc tính của DataFromClient và Data là giống nhau, sau khi phía

cloud nhận được tín hiệu HTTP từ Edge, các thuộc tính của DataFromClient sẽ được ánh xạ với các thuộc tính trong thực thể Data

```java
package com.demo.DataReciever.dto;


import jakarta.persistence.Lob;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class DataFromClient {
    private String fileName;
    private String fileType;
    private int fileSize;
    private double timeSent;
    private String content;
}
```

Thiết lập cổng API

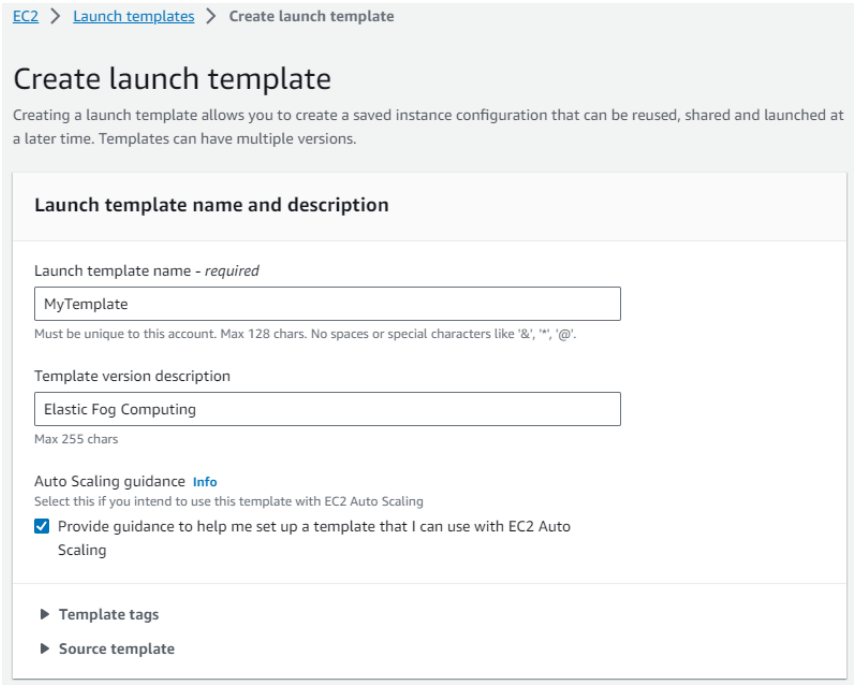
Để truyền dữ liệu vào cơ sở dữ liệu, cần sử dụng đường dẫn http://[IP]/api/data với phương thức POST và dữ liệu là một định dạng JSON được định nghĩa trong Sơ đồ giao tiếp. Sau đó, Data Receiver sẽ gọi đến phương thức uploadData, phương thức này sẽ gọi đến đối tượng IDataService để thực hiện việc lưu dữ liệu.

```java
package com.demo.DataReciever.service;

import com.demo.DataReciever.dto.DataFromClient;

import java.io.IOException;
import java.util.List;

public interface IDataService {
    String uploadData(DataFromClient data) throws
            IOException;
}
```

2.4.2 Thiết lập cơ chế co dãn

Khởi tạo Template cho Amazon EC2

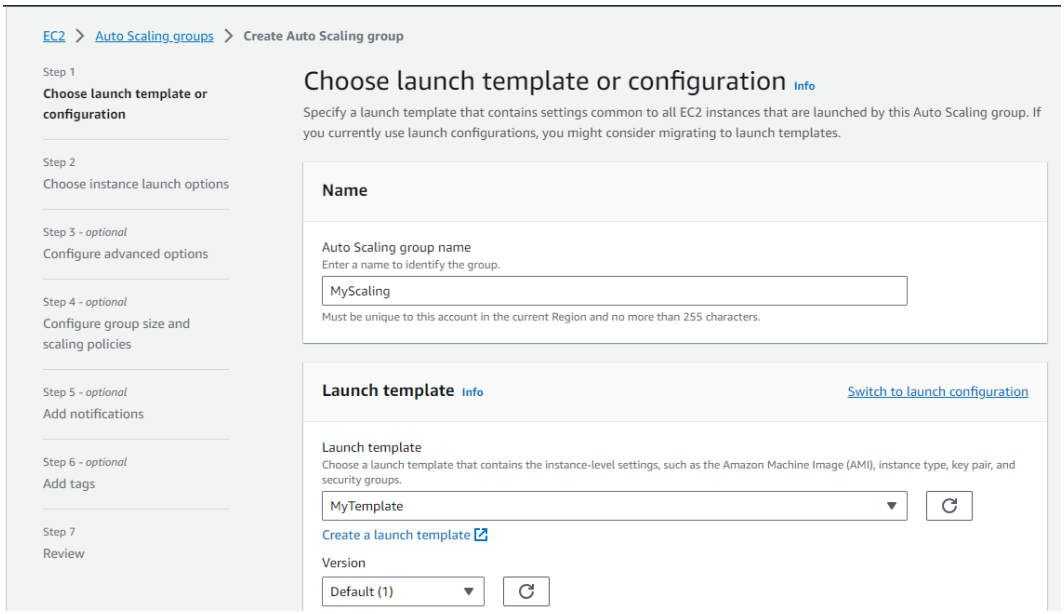
Các instance được khởi tạo sẽ có chung yêu cầu về CPU, RAM nên được gọi chung là template. Template được khởi tạo sử dụng OS là Ubuntu Server 22.04 LTS, 64-bit; loại instance là t2.micro (free tier eligible); có dung lượng bộ nhớ là 8GB.



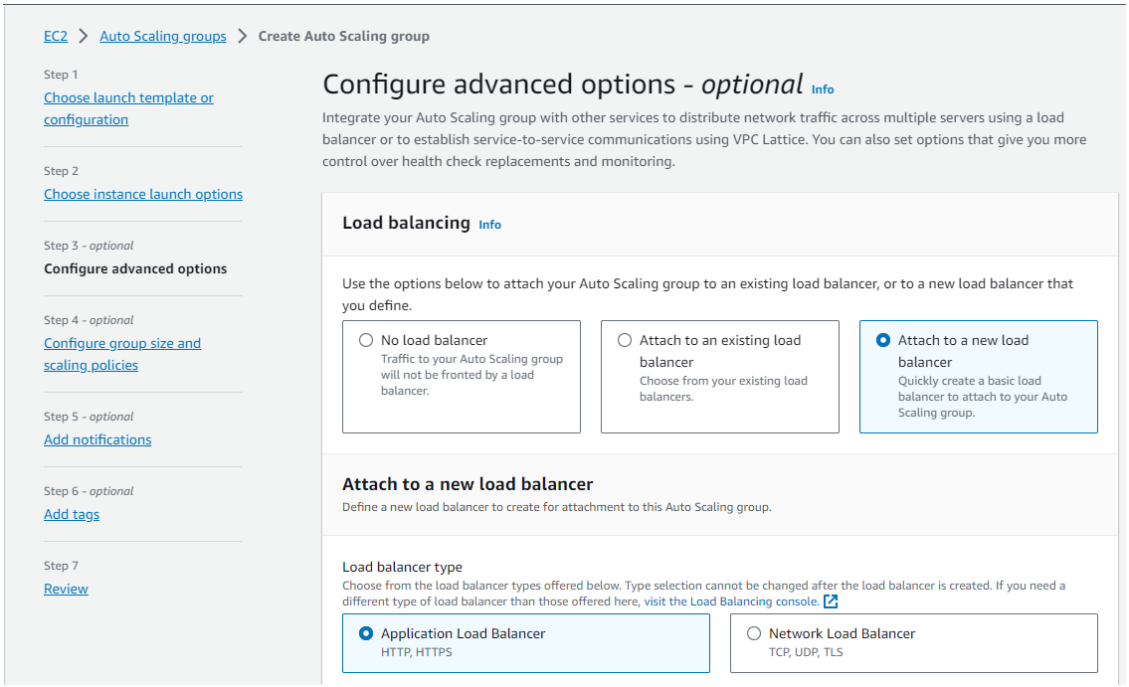
Hình 2-16: Thiết lập template cho các tài nguyên của instance

Khởi tạo Auto Scaling groups

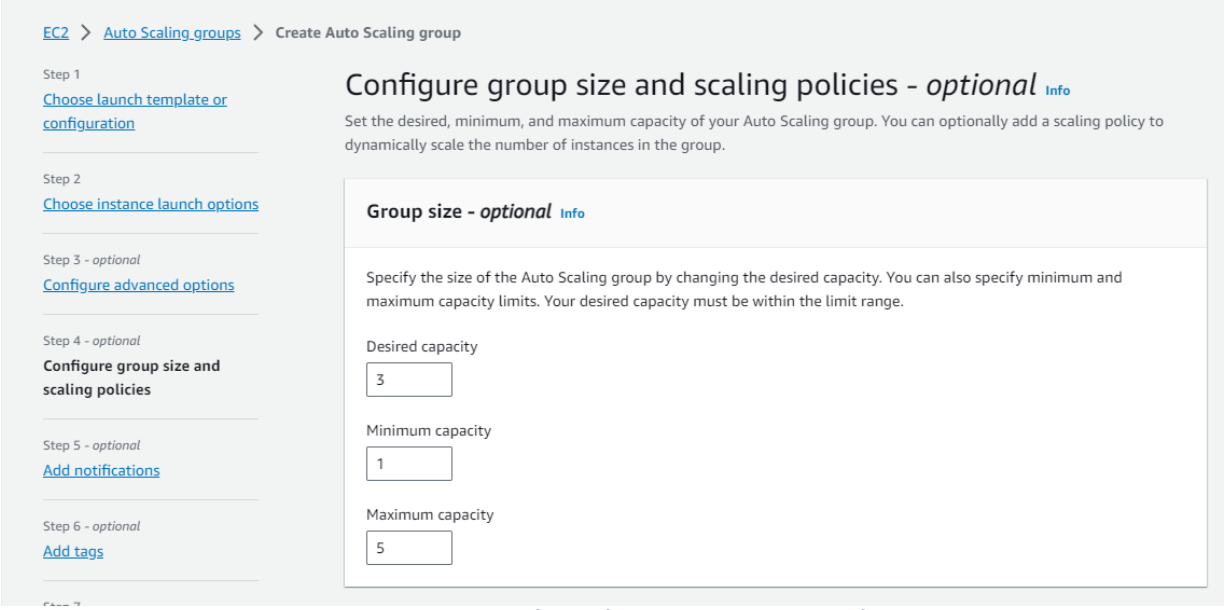
Lần lượt thực hiện theo hướng dẫn mà Amazon cung cấp. Điểm đáng chú ý là bước 3 và bước 4. Tại bước 3, LoadBalancer được khởi tạo để phân chia yêu cầu từ phía tải với các instance được khởi tạo. Số lượng instance được khởi tạo ở bước 4 với số lượng ít nhất là 1, nhiều nhất là 5.



Hình 2-17: Khởi tạo AutoScaling group



Hình 2-18: Thiệt lập Load Balancer cho Auto Scaling group

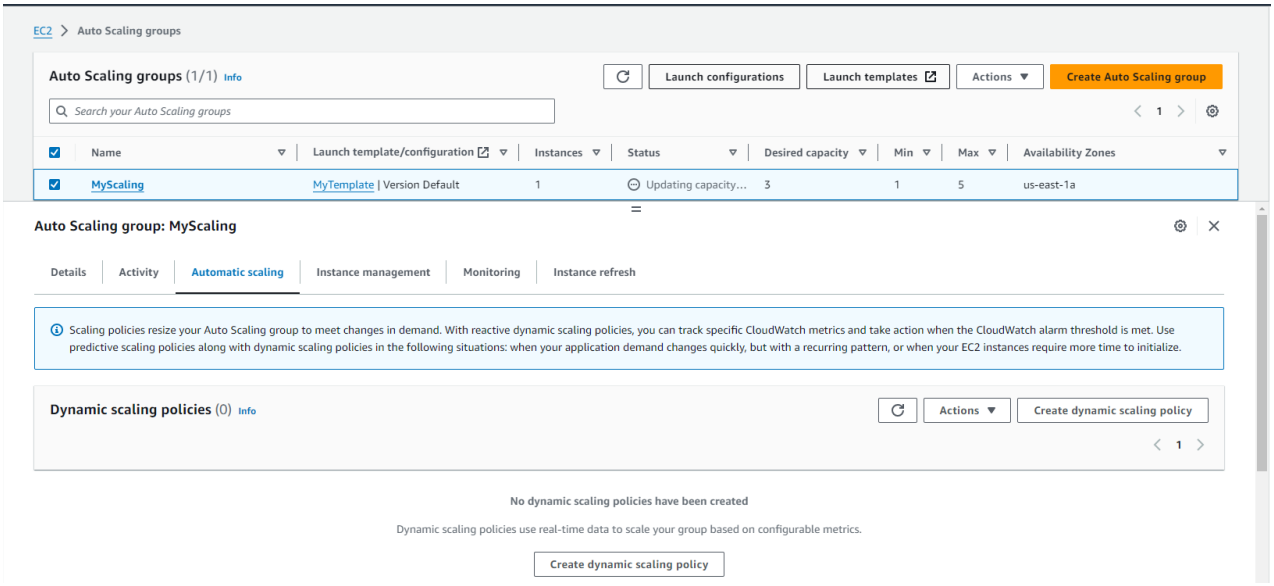


Hình 2-19: Thiết lập số lượng instance mong muốn

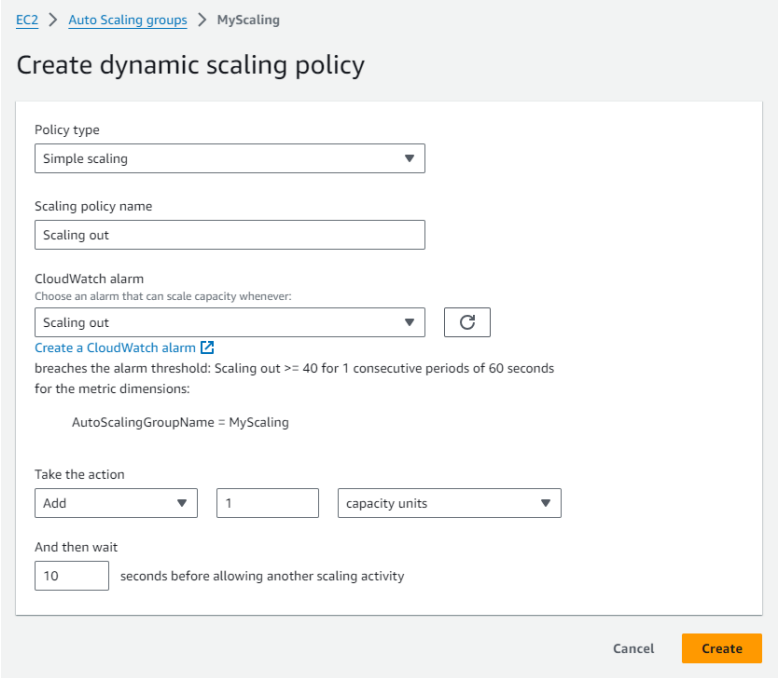
Thiết lập Dynamic Scaling Policy

Yêu cầu phía tải được chia nhỏ thành ba cơ chế là (1) Target tracking scaling theo dõi và đảm bảo Metrics bám theo một giá trị đặt có sẵn, ví dụ như tỷ lệ sử dụng CPU trung bình (Average CPU utilization) luôn đạt 30% (2) Simple scaling là cơ chế tăng giảm instance dựa trên theo dõi Metrics vượt quá giá trị đặt (threshold value) và (3) Step scaling có thể hiểu là tổng hợp của nhiều simple scaling và các cơ chế co dãn được kích hoạt khi nhiều Metrics vượt quá giá trị đặt. Nghiên cứu này thử nghiên với cơ chế co dãn đơn giản nhất là Simple scaling và lựa chọn theo dõi phần trăm CPU sử dụng không quá 40%.

Trong bảng theo dõi các EC2, lựa chọn MyScaling được khởi tạo ở trên và vào tab Automatic scaling. Trong tab này, có ba policy như đã trình bày bao gồm Dynamic scaling dựa trên vào yêu cầu phía tải, Predictive scaling dự báo trước yêu cầu và Scheduled tăng giảm theo lịch trình. Tại đây, policy dựa trên theo dõi CPU được khởi tạo bằng cách ấn nút Create dynamic scaling policy.

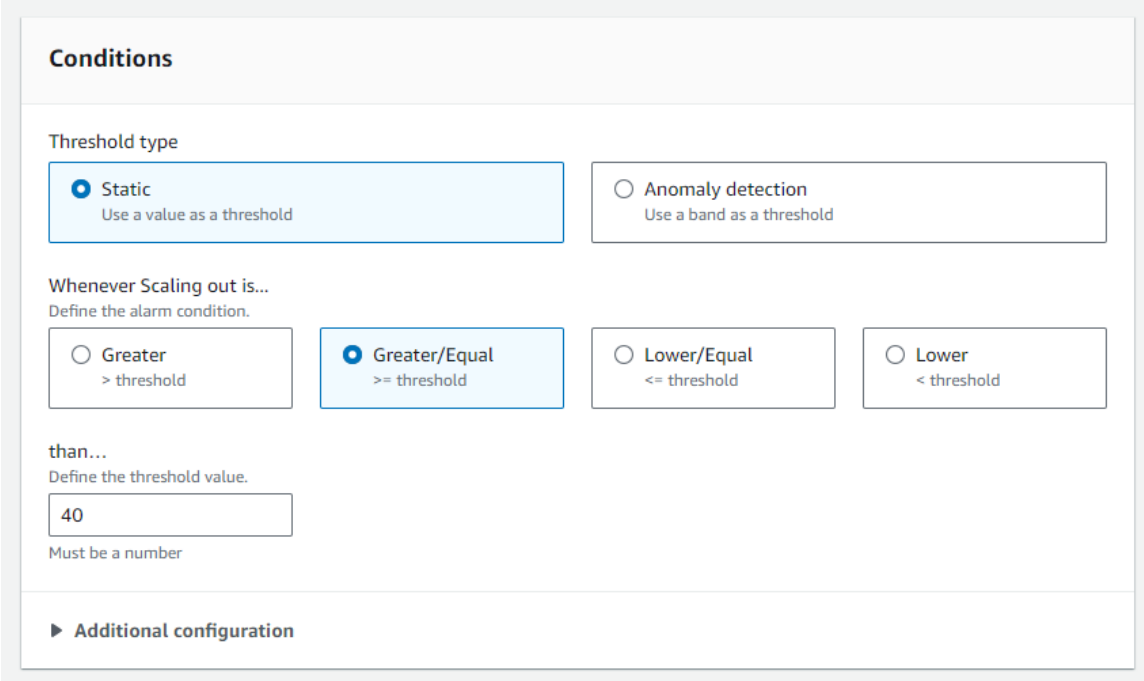


Hình 2-20: Thiết lập các cơ chế Automatic scaling



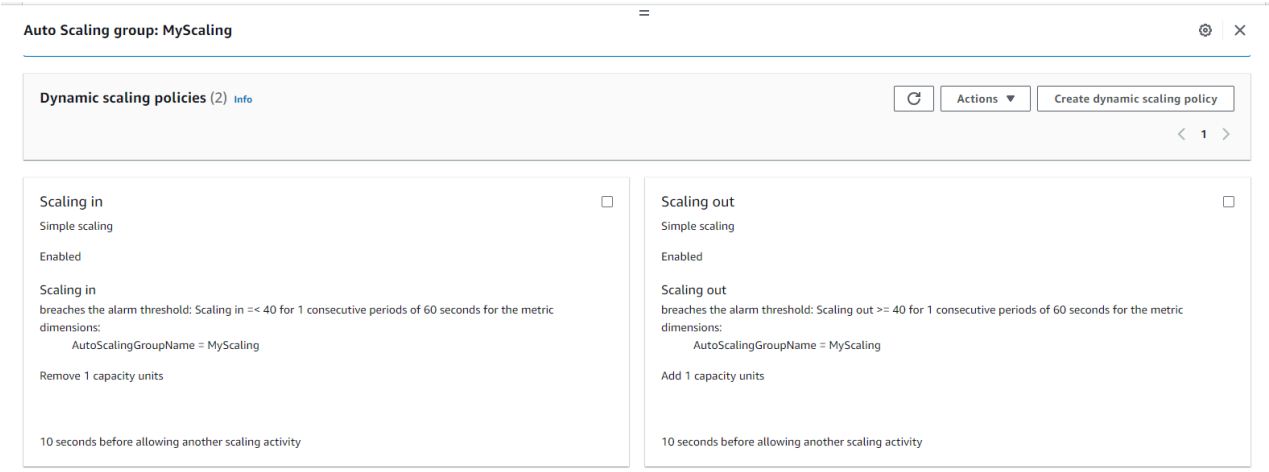
Hình 2-21: Thiết lập cơ chế tự động tăng instance

Hai cơ chế co dãn được thiết lập, cơ chế tăng thêm instance khi phần trăm CPU lớn hơn hoặc bằng 40% và cơ chế giảm instance khi phần trăm CPU dưới 40%. Trước tiên là thiết lập cơ chế tăng instance dựa trên giám sát phần trăm CPU sử dụng trong CloudWatch. Trong khi khởi tạo Policy, lựa chọn Create a CloudWatch alarm và thiết lập thông số tương ứng.



Hình 2-22: Thiết lập giám sát phần trăm CPU sử dụng trên CloudWatch

Thực hiện tương tự, cơ chế Scaling giảm instance khi CPU dưới 40% được thiết lập. Hai cơ chế sau khi khởi tạo xuất hiện trong tab Automatic scaling.



Hình 2-23: Hoàn thành thiết lập các cơ chế co dãn

3. CHƯƠNG 3: CÁC THỰC NGHIỆM VÀ KẾT QUẢ

3.1 Dữ liệu thử nghiệm

Để có thể đạt được sự đa dạng về chủng loại và dung lượng, dữ liệu thử nghiệm cho nghiên cứu này được lấy từ ba nguồn dữ liệu khác nhau. Các nguồn này là bộ dữ liệu hình ảnh của tập đoàn FPT, dữ liệu từ mạng cảm biến, và các loại văn bản từ Kaggle.

Dữ liệu hình ảnh khuôn mặt

Bộ dữ liệu này được lấy từ các tòa nhà của Tập đoàn FPT với dạng hình ảnh được mã hóa base64, có dung lượng 1.5GB. Sau khi giải mã, các file ảnh này sẽ có đuôi là PNG, JPG, và BMP.

Dữ liệu văn bản

Loại dữ liệu này được lấy từ địa chỉ trang web Kaggle[10], với keyword là electrion, được lưu trữ trong cơ sở dữ liệu sqlite với ba bảng là (1) coutry_facts với 54 cột và 3195 dòng, (2) country_facts_dictionary với 2 cột và 51 dòng, và (3) primary_results với 8 cột và 24611 dòng. Cơ sở dữ liệu được thay đổi lần cuối vào 4 năm trước. Dliệu này sẽ được phân thành các file csv với dung lượng khác nhau để đạt được tính đa dạng và dung lượng không biết trước.

Table	Total Rows	Total Columns
county_facts	3195	54
county_facts_dictionary	51	2
primary_results	24611	8

Hình 3-1: Thông tin về dữ liệu văn bản

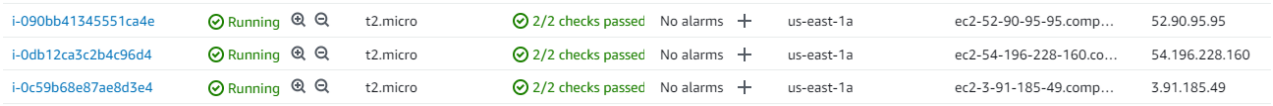
3.2 Kết quả thực nghiệm

3.2.1 Hiệu quả tính năng co dãn

Khởi tạo ban đầu của dịch vụ trên Cloud AWS là 3 instances, điều này được theo dõi qua console cung cấp. Hình 3-2 cho thấy, các instance này được phân bố cùng tại vùng us-east-1a nhưng lại có địa chỉ Public IPv4 khác nhau. Điều này không ảnh hưởng tới truy cập do người dùng sẽ truy cập đến địa chỉ DNS cung cấp bởi AWS LoadBalancer, và dịch vụ LoadBalancer sẽ tự động phân bố yêu cầu truy cập đó vào

[10] https://www.kaggle.com/datasets

3 instances này. Điều này đảm bảo tính ổn định và khả dụng khi có sự thay đổi từ phía yêu cầu người dùng.



i-090bb41345551ca4e	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-52-90-95-95.comp...	52.90.95.95
i-0db12ca3c2b4c96d4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-196-228-160.co...	54.196.228.160
i-0c59b68e87ae8d3e4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-3-91-185-49.comp...	3.91.185.49

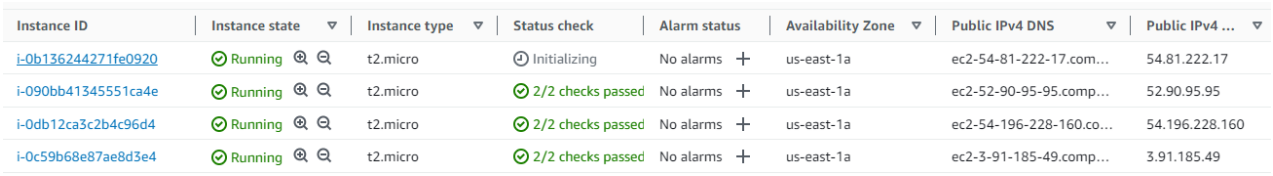
Hình 3-2: 3 instances đang chạy trên Cloud AWS

Tự động tăng số lượng instances

Truy cập vào một trong instance đang được sử dụng và sử dụng lệnh, để tăng phần trăm CPU sử dụng lên quá 40%.

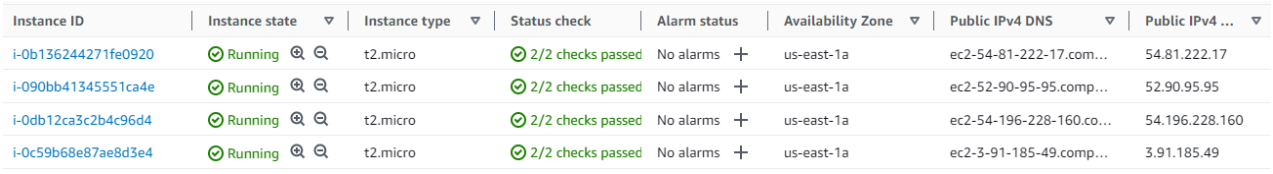
sudo stress --cpu 9 --timeout 1200

AWS AutoScaling sẽ tự động khởi tạo thêm một instance như trong Hình 3-3 và hoàn thành ở Hình 3-4. Hiện tại cơ chế tự động này dựa trên giám sát CPUUtilization Metrics, tức phần trăm CPU sử dụng.



Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
i-0b136244271fe0920	Running	t2.micro	Initializing	No alarms	us-east-1a	ec2-54-81-222-17.com...	54.81.222.17
i-090bb41345551ca4e	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-52-90-95-95.comp...	52.90.95.95
i-0db12ca3c2b4c96d4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-196-228-160.co...	54.196.228.160
i-0c59b68e87ae8d3e4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-3-91-185-49.comp...	3.91.185.49

Hình 3-3: Khởi tạo thêm instance trên Cloud AWS

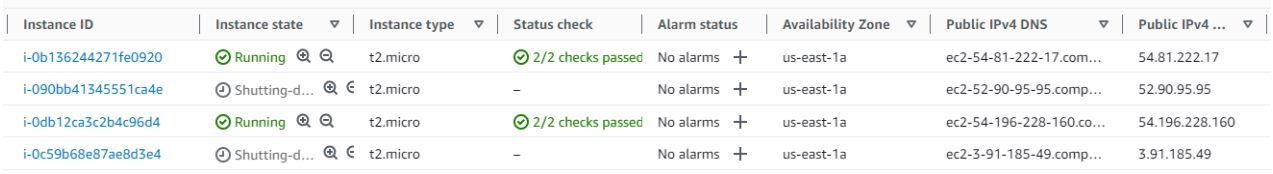


Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
i-0b136244271fe0920	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-81-222-17.com...	54.81.222.17
i-090bb41345551ca4e	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-52-90-95-95.comp...	52.90.95.95
i-0db12ca3c2b4c96d4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-196-228-160.co...	54.196.228.160
i-0c59b68e87ae8d3e4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-3-91-185-49.comp...	3.91.185.49

Hình 3-4: 4 instances đang chạy trên Cloud AWS

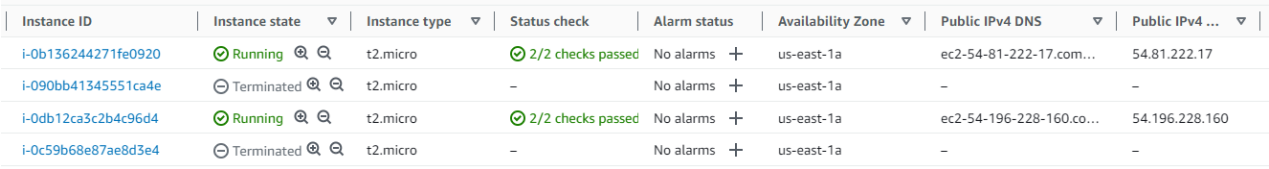
Tự động giảm số lượng instances

Tương tự khi phần trăm CPU sử dụng giảm xuống dưới 40%, AWS AutoScaling sẽ tự động giảm số lượng instance tương ứng. Hình 3-5 cho thấy hai instance có id là i-090bb41345551ca4e và i-0c59b68e87ae8d3e4 đang được tắt đi và hoàn toàn bị loại bỏ trong Hình 3-6.



Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
i-0b136244271fe0920	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-81-222-17.com...	54.81.222.17
i-090bb41345551ca4e	Shutting-d...	t2.micro	–	No alarms	us-east-1a	ec2-52-90-95-95.comp...	52.90.95.95
i-0db12ca3c2b4c96d4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-196-228-160.co...	54.196.228.160
i-0c59b68e87ae8d3e4	Shutting-d...	t2.micro	–	No alarms	us-east-1a	ec2-3-91-185-49.comp...	3.91.185.49

Hình 3-5: Giảm bớt instance trên Cloud AWS



Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
i-0b136244271fe0920	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-81-222-17.com...	54.81.222.17
i-090bb41345551ca4e	Terminated	t2.micro	–	No alarms	us-east-1a	–	–
i-0db12ca3c2b4c96d4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-54-196-228-160.co...	54.196.228.160
i-0c59b68e87ae8d3e4	Terminated	t2.micro	–	No alarms	us-east-1a	–	–

Hình 3-6: 2 instances đang chạy trên Cloud AWS

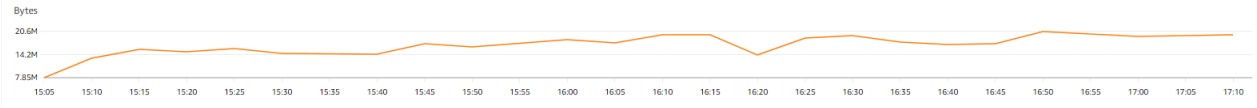
Cơ chế co dãn này hiên mới dựa trên CPU đang sử dụng trên các instance. Để có thể tăng hiệu quả co dãn, có thể tích hợp thêm một cơ chế giám sát thông lượng dữ liệu đến LoadBalancer. Điều này là cần thiết do luồng dữ liệu thường có đặc điểm thay đổi về dung lượng, có thể dẫn đến sự tăng đột biến của thông lượng mạng, được giám sát qua Network In Metrics trên CloudWatch. Việc mở rộng cơ chế co dãn để bao gồm giám sát thông lượng mạng sẽ giúp đảm bảo tính ổn định và hiệu quả của hệ thống trong các tình huống hạ tầng mạng có biến động do ảnh hưởng của luồng dữ liệu.

3.2.2 Hiệu quả về hạ tầng mạng

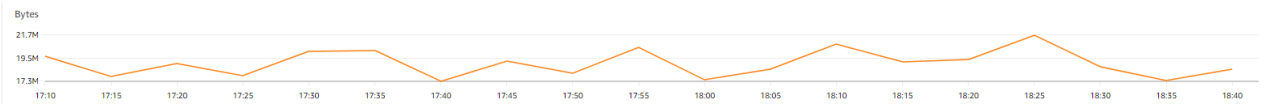
Đánh giá về sử dụng hiệu quả hạ tầng mạng dựa trên rất nhiều khía cạnh, có thể kể đến như băng thông và thông lượng (bandwidth and throughput), mô hình mạng (topology), độ trễ (latency), hay khả năng mất mát gói tin (packet lot). Nghiên cứu này đánh giá hai khía cạnh là băng thông mạng (network bandwidth) và độ trễ của mạng (network latency).

Thông lượng mạng

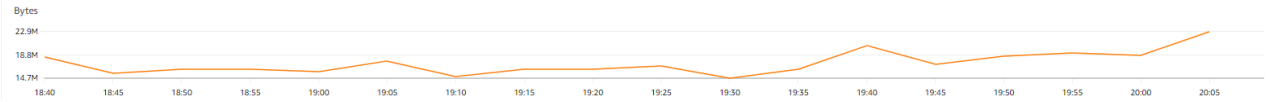
Thông lượng mạng là lượng dữ liệu được truyền thực chất trên hạ tầng mạng. Thông thường, các nhà mạng đều cung cấp cho người dùng khả năng truyền dữ liệu trên hạ tầng mạng. Khả năng này được biểu thị bằng con số có đơn vị thường là bits per second (bps) hoặc bytes per second (Bps). Tuy vậy, con số này chỉ mang tính chất lý thuyết và thường được gọi là băng thông (network). Giá trị thực tế của lượng dữ liệu truyền trên hạ tầng mạng chính là thông lượng mạng. Hình 3-7 khi không sử dụng điện toán biên cho thấy, thông lượng mạng sử dụng luôn duy trì trung bình ở mức 18MBps. Khi sử dụng hai thuật toán nén Deflate và LZMA, chỉ số này không có sự thay đổi quá nhiều và sự ổn định cũng không quá lớn. Điều này thể hiện rõ nhất đối với thuật toán Deflate Hình 3-8, có thời điểm có điểm ngoại lai lên đến 21.7MB ở lúc 18:25, nhưng trong khoảng 15 phút đầu, chỉ số luôn duy trì dưới mức 19.5MB. Thuật toán LZMA cũng có phản ứng tượng tự nhưng duy trì được thông lượng dưới mức 18.8MB trong một khoảng thời gian gần 50ph đầu.



Hình 3-7: Thông lượng mạng khi không sử dụng điện toán biên



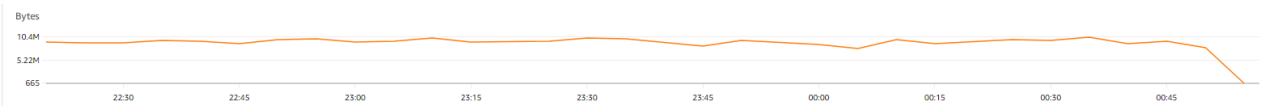
Hình 3-8: Thông lượng mạng khi sử dụng điện toán biên, thuật toán nén Deflate



Hình 3-9: Thông lượng mạng khi sử dụng điện toán biên, thuật toán nén LZMA

Tuy nhiên, khi sử dụng điện toán biên áp dụng hai thuật toán nén LZ4 và Bzip2, tác động tích cực được thấy nhìn thấy rõ khi chỉ số thông lương mạng trung bình giảm đáng kể xuống còn khoảng 14MB và 9MB tương ứng. Đặc biệt thuật toán Bzip2 Hình 3-10 duy trì được sự ổn định hơn hẳn so với ba thuật toán còn lại, các điểm ngoại lai không khác biệt quá lơn so với giá trị trung bình. Thuật toán LZ4 Hình 3-11 có thời gian chạy ổn định từ lúc 21:00 đến 22:00 ở mức 8.4MB. Trong giai đoạn đầu, thuật toán này có sự giao động nhẹ trong khoảng từ 15.6MB đến 22.9MB.

Hình 3-10: Thông lượng mạng khi sử dụng điện toán biên, thuật toán nén LZ4



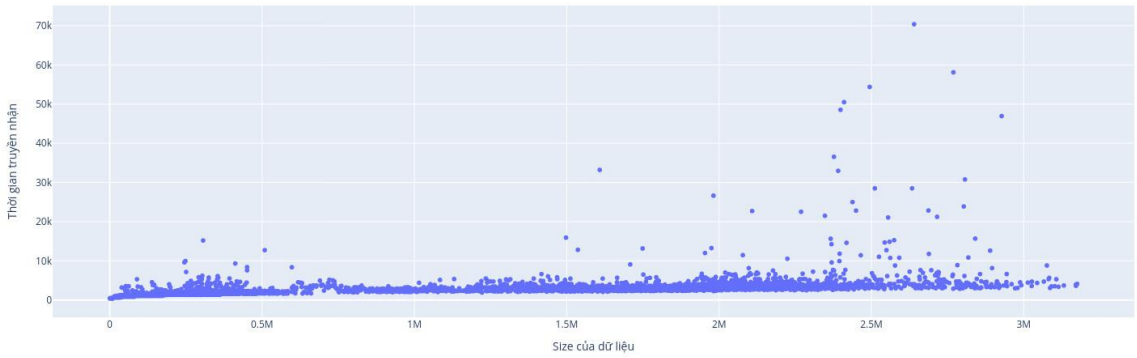
Hình 3-11: Thông lượng mạng khi sử dụng điện toán biên, thuật toán nén Bzip2

Các biểu đồ này cũng cho thấy, sử dụng băng thông của hạ tầng mạng có sự thay đổi khi áp dụng điện toán biên. Nếu coi băng thông mạng khi không sử dụng các thuật toán nén là giới hạn của băng thông thì hai thuật toán nén LZ4 và Bzip2 đã giúp việc sử dụng hạ tầng mạng tối ưu hơn khi sử dụng ít băng thông hơn để truyền tải dữ liệu.

Độ trễ mạng

Chỉ số này cho biết thời gian cần thiết để chuyển gói tin từ nơi bắt đầu đến điểm đích. Ý tưởng khi áp dụng thuật toán nén của nghiên cứu này là để giảm thời gian truyền nhân bản tin, tức cải thiện chỉ số này. Hình 3-12 cho thấy dữ liệu truyền từ nguồn tới đích trong cả hai trường hợp không và có áp dụng thuật toán nén khoảng 2 giây và rõ ràng khi dung lượng dữ liệu tăng lên thì thời gian truyền nhận cũng tăng

lên tương ứng. Các điểm ngoại lại cho thấy chỉ số này còn phụ thuộc vào tốc độ mạng tại thời điểm truyền, mặc dù dữ liệu truyền đi có dung lượng không quá khác biệt. Trong Hình 3-13, file 463.png có thời gian truyền là 12.741 giây, trong khi file 17.png chỉ mất có 1.693 giây

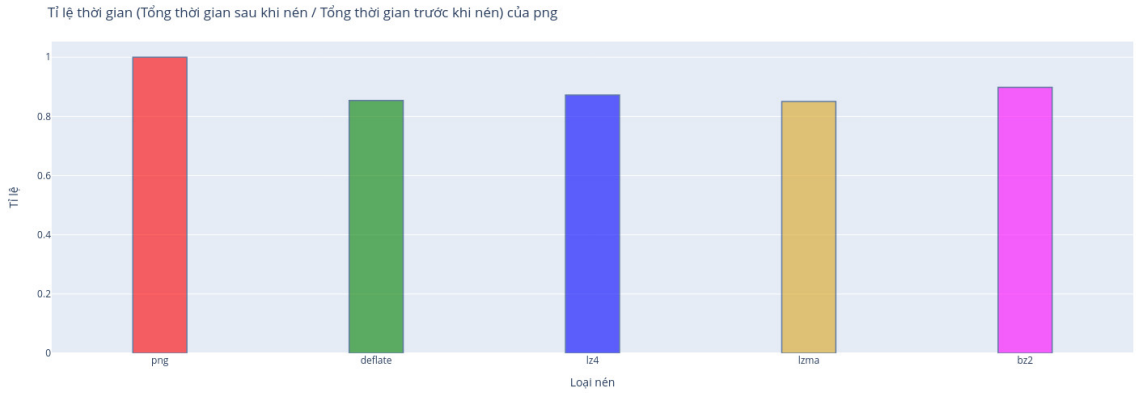


Hình 3-12: Mối quan hệ giữa dung lượng và thời gian truyền dữ liệu

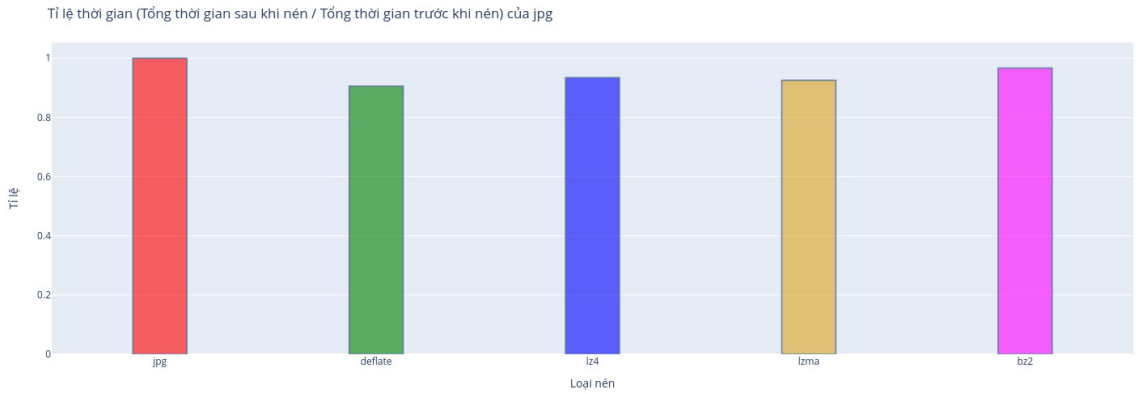
	content	file_name	content_length	file_type	time_received	time_sent	period
31	16477	463	508630	png	1699024470069	1699024457328	12741
360	17380	250	521942	png	1699027658319	1699027656607	1712
538	17864	364	533681	png	1699029069360	1699029066273	3087
703	18288	17	504609	png	1699030323806	1699030322113	1693
746	18391	300	527641	png	1699030639878	1699030636943	2935
767	18443	715	548203	png	1699030788820	1699030786707	2113

Hình 3-13: Ví dụ về các điểm ngoại lai

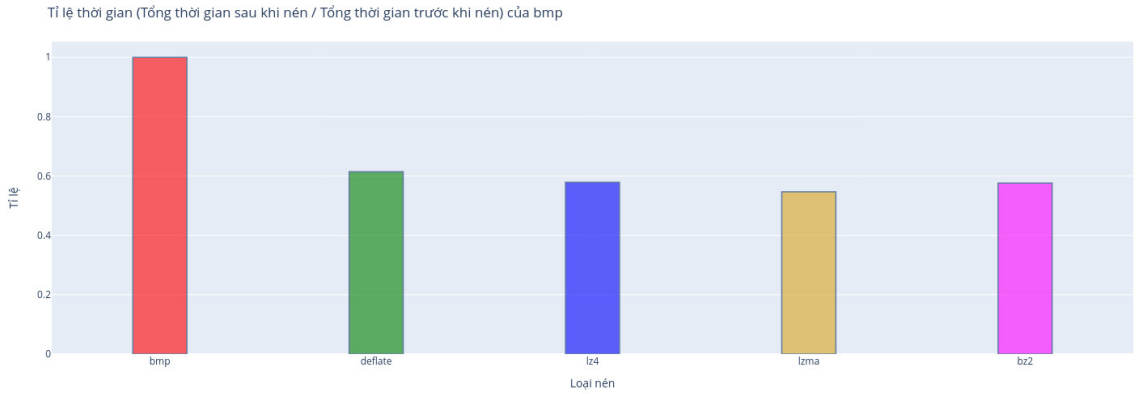
Để đánh giá kĩ hơn về ảnh hưởng của thuật toán nén, nghiên cứu xét đến ảnh hưởng của các thuật toán nén với từng loại dữ liệu. Lần lượt là dữ liệu dạng ảnh (JPG, PNG, BMP), dữ liệu âm thanh (WAV), và dữ liệu dạng text (CSV). Thời gian truyền nhận không nén và có nén được normalize để so sánh. Đối với dữ liệu dạng ảnh, sử dụng thuật toán nén lên hai file đuôi PNG và JPG không quá thay đổi quá rõ rệt như trong Hình 3-14 và 3-15. Trong bốn thuật toán nén được sử dụng, thuật toán LZMA có ảnh hưởng tốt nhất khi thời gian gian truyền nhận chỉ bằng khoảng 0.85 lần khi không nén. Nhưng nhiều chung, thời gian khi áp dụng thuật toán nén đều giảm.



Hình 3-14: Thời gian truyền nhận của file dạng PNG



Hình 3-15: Thời gian truyền nhận của file dạng JPG



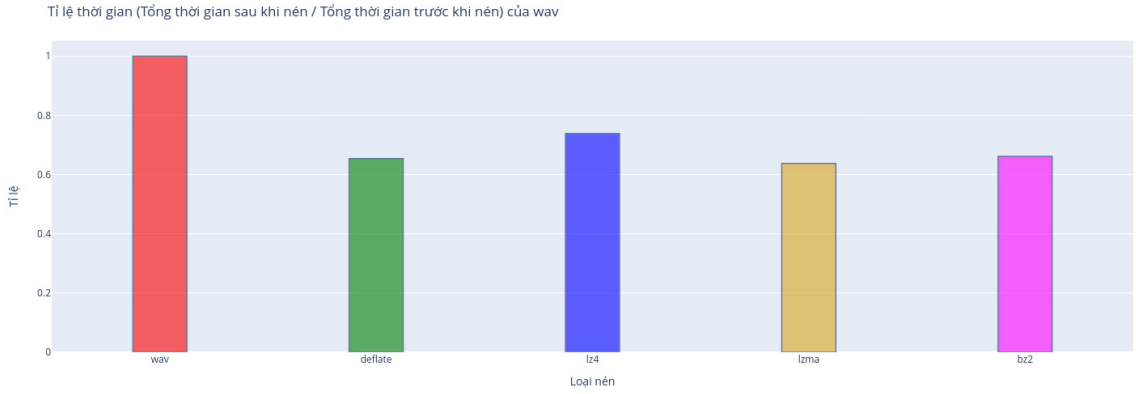
Hình 3-16: Thời gian truyền nhận của file dạng BMP

Ngược lại với dữ liệu dạng BMP, ảnh hưởng các thuật toán nén là rõ rệt khi thời gian truyền nhận chỉ bằng khoảng 0.6 lần. Như vậy, trong các ứng dụng phải sử dụng file ảnh, lưu file ảnh dưới dạng BMP là một phương pháp nên được cân nhắc so với dữ liệu dạng JPG và PNG.

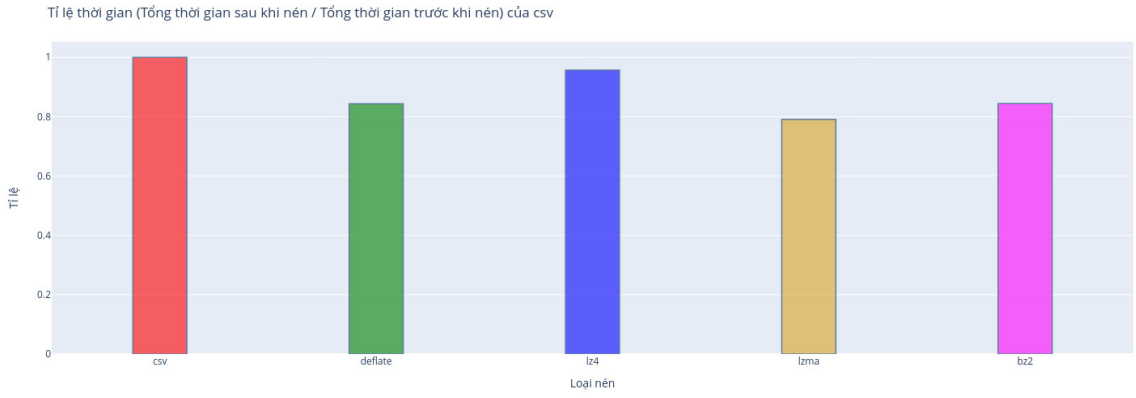
Đối với dữ liệu âm thanh được lưu dưới dạng WAV, ảnh hưởng khi áp dụng điện toán biên và các thuật toán nén cũng là tích cực. Hình 3-17 cho thấy, các thuật

toán nén làm giảm thời giảm truyền nhận xuống chỉ còn khoảng 0.65 lần so với khi không nén. Thuật toán LZMA là thuật toán có ảnh hưởng tích cực nhất so với ba thuật toán còn lại, giúp thời gian gửi chỉ còn khoảng 0.61 lần. Thuật toán LZ4 có tác động kém nhất khi chỉ giảm khoảng 0.7 lần. Như vậy, các thuật toán toán nén hoạt động tương đối hiệu quả đối với dữ liệu âm thanh.

Đối với các tệp dữ liệu dạng text, ảnh hưởng của các thuật toán nén khá giống dữ liệu dạng ảnh khi kết quả không có sự thay đổi đáng kể. Thuật toán LZMA đạt hiệu quả tốt nhất khi giảm thời gian truyền nhân còn 0.8 lần so với khi không nén. Tác động của thuật toán Deflate và Bzip2 là gần như nhau, trong khi thuật toán LZ4 chỉ khiến thời gian truyền nhận giảm đi khoảng 0.1 lần.



Hình 3-17: Thời gian truyền nhận của file dạng WAV

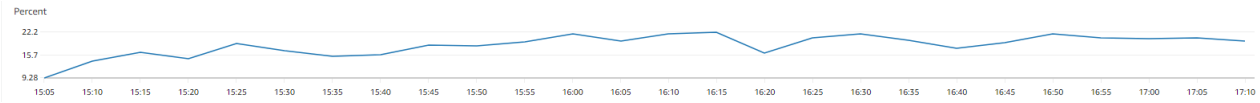


Hình 3-18: Thời gian truyền nhận của file dạng CSV

3.2.3 Hiệu quả về hiệu năng tính toán

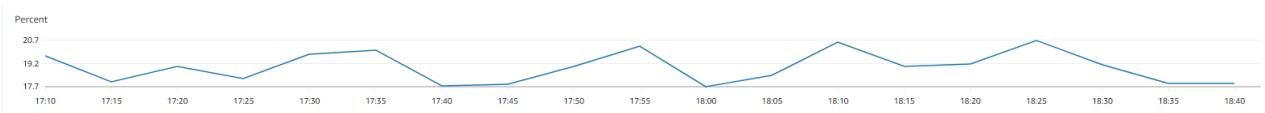
Hiệu năng tính toán được đánh giá qua phần trăm CPU được tận dụng trong nghiên cứu này. Khi không áp dụng điện toán biên, phần trăm CPU sử dụng trên Amazon AWS đạt trong khoảng từ 15.7% đến 22.2%, Hình 3-19. Thêm vào đó, do

sự thay đổi của dung lượng dữ lệu truyền nhận, phần trăm này có nhiều điểm ngoại lai. Sự không ổn định này có thể giảm tuổi thọ sử dụng của thiết bị phần cứng này.

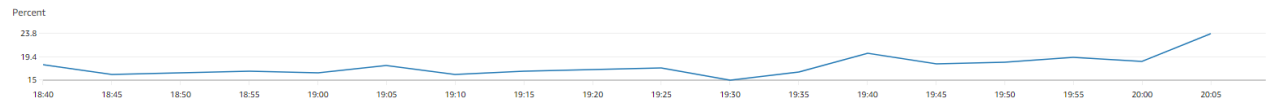


Hình 3-19: Hiệu năng tính toán khi không sử dụng điện toán biên

Khi áp dụng điện toán biên với hai thuật toán nén Deflate và LZMA, phần trăm CPU không có sự thay đổi quá nhiều về mặt ổn định. Thuật toán Deflate Hình 3-20 còn gây sự bất ổn định hơn khi không áp dụng nhưng các điểm ngoại lai có biên đô không còn lớn bằng khi không áp dụng thuật toán nén, chỉ đạt 20.7% trong khi 22.2% là phần trăm điểm ngoại lai lớn nhất khi không áp dụng thuật toán nén. Thuật toán LZMA có ảnh hưởng tích cực hơn khi trong 50ph giai đoạn đầu, phần trăm CPU được duy trì ổn định dưới 19.4%. Trong giai đoạn sau này, chỉ số này có thời điểm tăng vọt lên 23.8%.

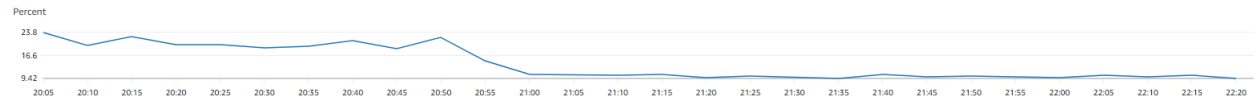


Hình 3-20: Hiệu năng tính toán khi sử dụng điện toán biên, thuật toán nén Deflate

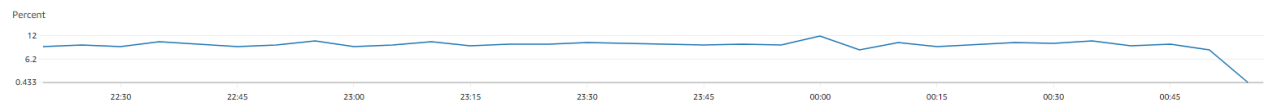


Hình 3-21: Hiệu năng tính toán khi sử dụng điện toán biên, thuật toán nén LZMA

Hai thuật toán LZ4 Hình 3-22 và Bzip2 Hình 3-23 có ảnh hưởng tích cực hơn rất nhiều. Thuật toán LZ4 có tỷ lên phần trăm CPU sử dụng trung bình trong suốt khoảng thời giản khảo sát đạt khoảng 15%. Từ lúc 21:00 trở đi, chỉ số này được duy trì ổn định ở mức 9.42%. Trong khi đó, thuật toán Bzip2 đạt được sự ổn định khoảng 10% trong suốt khoảng thời gian khảo sát. Đáng chú ý là các điểm ngoại lai cũng không bao giờ vượt quá giới hạn 12%.



Hình 3-22: Hiệu năng tính toán khi sử dụng điện toán biên, thuật toán nén LZ4



Hình 3-23: Hiệu năng tính toán khi sử dụng điện toán biên, thuật toán nén Bzip2

Chỉ số này có sự tương đồng với chỉ số thông lượng mạng được đánh giá ở trên. Ảnh hưởng của hạ tầng mạng lên chỉ số thông lượng có rất nhiều yếu tố. Sự tương

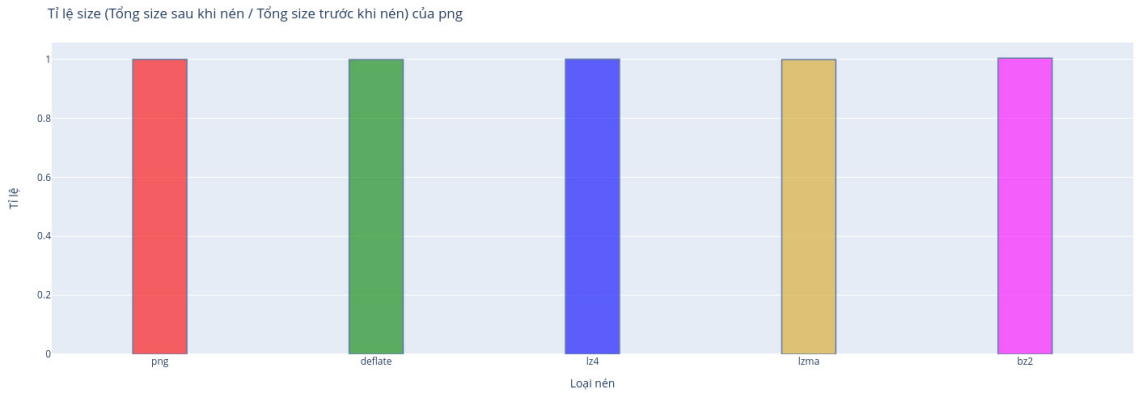
đồng này, cho thấy ảnh hưởng của hạ tầng mạng cũng có sự tác động lên hiệu năng tính toán tại phía Cloud sử dụng.

3.2.4 Hiệu quả về tài nguyên lưu trữ

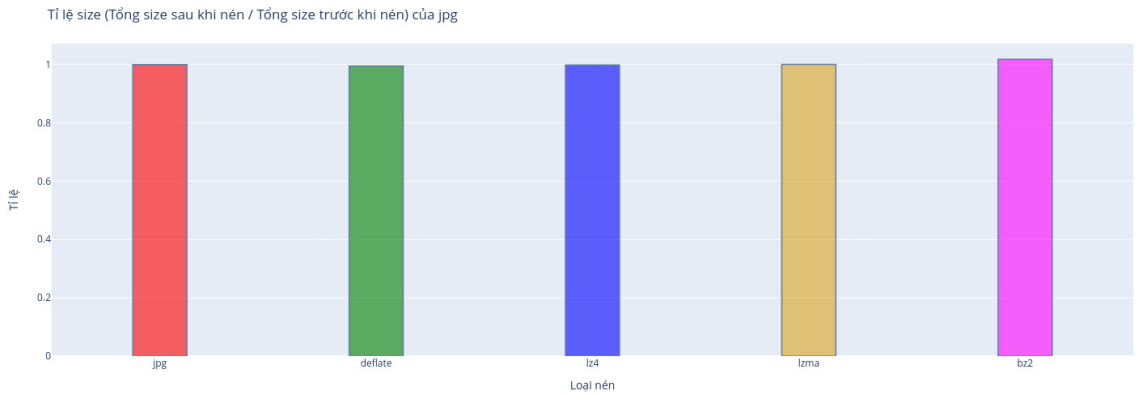
Trong bốn thuật toán nén sử dụng, ảnh hưởng của các thuật toán này lên các loại dữ liệu khác nhau sẽ khác nhau, dẫn đến phải khảo sát với lần lượt từng loại dữ liệu. Kết quả khảo sát được thể hiện qua biểu đồ cột đã được normalize với size ban đầu có dung lượng là 1 ở trục tung, trục hoành thể hiện năm loại file tương ứng là khi chưa nén, và nén với thuật toán tương ứng thể hiện ở tên bz2, deflate, lz4, và lzma.

Dữ liệu dạng ảnh

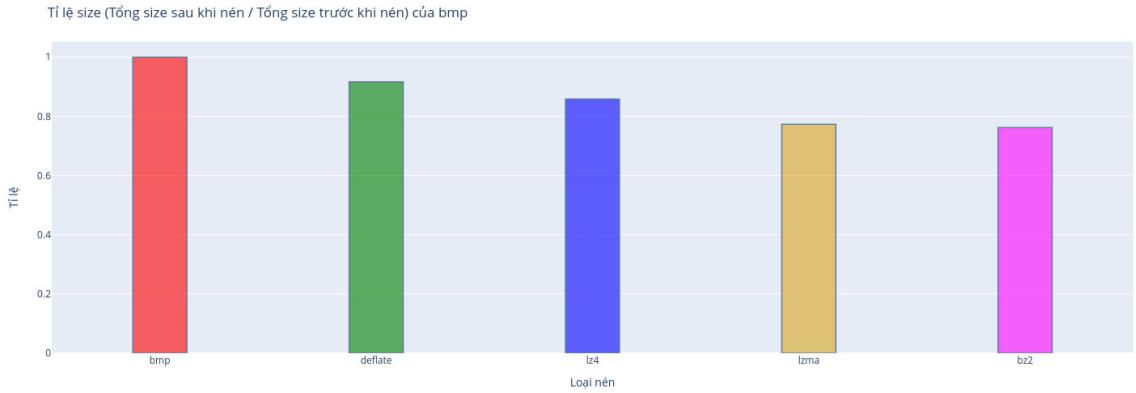
Trong tập dữ liệu được sau khi được giải mã từ mã hóa Base64 với extension đã biết trước là JPG, PNG, và BMP. Kết quả thực nghiệm cho thấy, bốn thuật toán nén sử dụng gần như không làm thay đổi dung lượng của các file ảnh dạng JPG và PNG. Điều này lý giải vì sao thời gian truyền nhận của hai dạng file này không có sự thay đổi đáng kể. Trong khi đó file ảnh dạng BMP có sự thay đổi rõ rệt, đặc biệt với thuật toán nén Bzip2 khi dung lượng chỉ bằng 0.8 dung lượng ban đầu.



Hình 3-24: So sánh tỉ lệ size trước và sau nén của tệp PNG



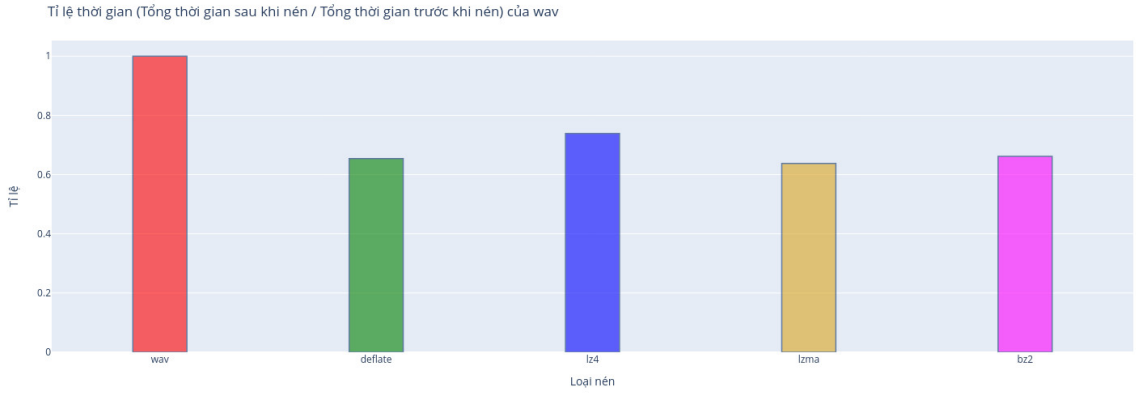
Hình 3-25: So sánh tỉ lệ size trước và sau nén của tệp JPG



Hình 3-26: So sánh tỉ lệ size trước và sau nén tệp BMP

Dữ liệu âm thanh

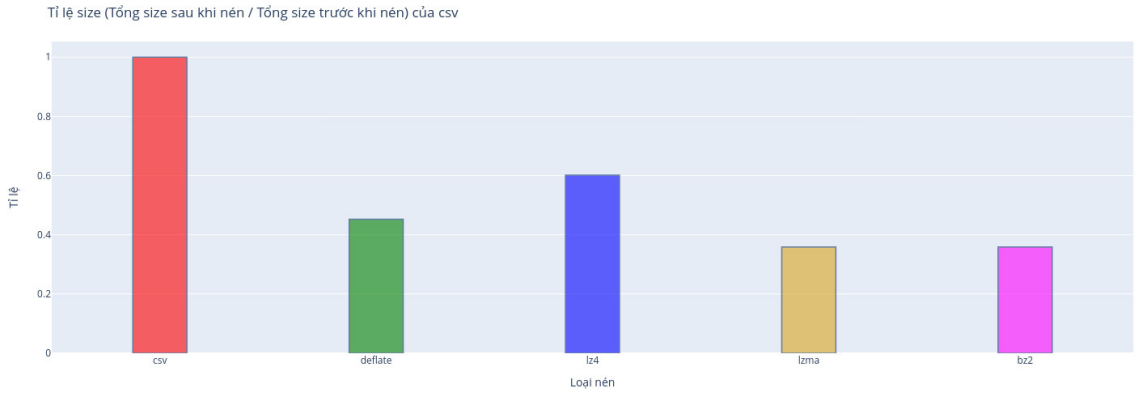
Trái ngược với dữ liệu dạng ảnh, ảnh hưởng của thuật toán nén lên dữ liệu này là vô cùng đáng kể. Dữ liệu sau khi nén cho kết quả tốt hơn, với thuật toán LZ4, dữ liệu giảm khoảng 20% so với dữ liệu gốc và với thuật toán BZ2 giảm khoảng 55% so với dữ liệu gốc.



Ảnh 3-3: So sánh tỉ lệ size trước và sau nén tệp wav

Dữ liệu văn bản

Đối với tệp dataset đang xét, thì các thuật toán nén hoạt động tốt nhất đối với các loại dữ liệu dạng text khi dữ liệu sau nén giảm tối thiểu khoảng 40% và tối đa khoảng 60%.



Hình 3-27: So sánh tỉ lệ size trước và sau nén tệp csv

Như vậy, ảnh hưởng của các thuật toán nén lên khả năng lưu trữ là vô cùng đáng kể. Băng cách tiết kiêm tài nguyên lưu trữ, chi phí vận hành và thuê tài nguyên của các nhà cung cấp Cloud cũng sẽ được giảm đáng kể.

CHƯƠNG 4: KẾT LUẬN

Bài nghiên cứu này đã khảo sát các phương pháp xử lý luồng dữ liệu cho các ứng dụng IoT cung cấp bởi Amazon và Google; từ đó đề xuất mô hình xử lý luồng dữ liệu dựa trên mô hình điện toán biên và tính năng co dãn của điện toán đám mây. Mô hình điện toán biên được xây dựng với bốn thuật toán nén là Deflate, LZMA, LZ4, và Bzip2 nhằm tối ưu về sử dụng hạ tầng mạng, hiệu năng của tài nguyên tính toán và lưu trữ. Một số kết luận của đồ án rút ra bao gồm:

1. Cơ chế co dãn mặc định cung cấp của AWS có thời gian phản ứng chậm, chưa đáp ứng được đầy đủ các yêu cầu xử lý thời gian thực của luồng dữ liệu trong các ứng dụng IoT. Trong tương lại, để hạn chế thời gian phản ứng, mô hình co dãn cần phải được tích hợp các phương thức dự báo dựa trên các giải thuật Machine Learning.
2. Cần thử nghiệm với các thuật toán nén khác để mang lại hiệu quả về hạ tầng mạng, tài nguyên tính toán và lưu trữ một cách rõ rệt hơn. Thêm vào đó, việc lựa chọn thuật toán nén phù hợp với kiểu dữ liệu là cần thiết để tăng hiệu quả của mô hình điện toán biên.
3. Yêu cầu về phân tích thời gian thực của luồng dữ liệu chưa được chú trọng ở nghiên cứu này, vì vậy mô hình tương lại cần giải quyết yêu cầu này.

KIẾN NGHỊ VÀ GIẢI PHÁP

1. Tập dữ liệu thực nghiệm cần được mở rộng với nhiều loại dữ loại và từ nghiều nguồn khác nhau
2. Các phương pháp đánh giá hiệu quả về hạ tầng mạng, tài nguyên tính toán và lưu trữ cần được xem xét kĩ hơn.

4. TÀI LIỆU THAM KHẢO

[1] L. Atzori, A. Iera, and G. Morabito, Oct. 2010, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, doi: 10.1016/j.comnet.2010.05.010.

[2] M. A. Jabraeil Jamali, B. Bahrami, A. Heidari, P. Allahverdizadeh, and F. Norouzi, 2020, "IoT Architecture," in *Towards the Internet of Things: Architectures, Security, and Applications*, M. A. Jabraeil Jamali, B. Bahrami, A. Heidari, P. Allahverdizadeh, and F. Norouzi, Eds., in EAI/Springer Innovations in Communication and Computing. , Cham: Springer International Publishing, pp. 9–31. doi: 10.1007/978-3-030-18468-1_2.

[3] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, Dec. 2012, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," in *2012 10th International Conference on Frontiers of Information Technology*, pp. 257–260. doi: 10.1109/FIT.2012.53.

[4] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, Aug. 2010, "Research on the architecture of Internet of Things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, pp. V5-484-V5-487. doi: 10.1109/ICACTE.2010.5579493.

[5] N. Herbst, S. Kounev, and R. Reussner, Jan. 2013, "Elasticity in cloud computing: What it is, and what it is not," *Int. Conf. Auton. Comput.*, pp. 23–27.

[6] G. Galante and L. C. E. De Bona, 2012, "A survey on cloud computing elasticity," presented at the Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012, pp. 263–270. doi: 10.1109/UCC.2012.30.

[7] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, 2015, "Elasticity in cloud computing: a survey," *Ann. Telecommun. Telecommun.*, vol. 70, no. 7–8, pp. 289–309, doi: 10.1007/s12243-014-0450-7.

[8] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, 2018, "Elasticity in Cloud Computing: State of the Art and Research Challenges," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 430–447, doi: 10.1109/TSC.2017.2711009.

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, Oct. 2016, “Edge Computing: Vision and Challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, doi: 10.1109/JIOT.2016.2579198.

[10] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, 2002, “Models and issues in data stream systems,” presented at the Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 1–16. doi: 10.1145/543613.543615.

[11] M. Dias de Assunção, A. da Silva Veith, and R. Buyya, 2018, “Distributed data stream processing and edge computing: A survey on resource elasticity and future directions,” *J. Netw. Comput. Appl.*, vol. 103, pp. 1–17, doi: 10.1016/j.jnca.2017.12.001.

[12] D. J. Abadi *et al.*, 2003, “Aurora: A new model and architecture for data stream management,” *VLDB J.*, vol. 12, no. 2, pp. 120–139, doi: 10.1007/s00778-003-0095-z.

[13] D. J. Abadi *et al.*, 2005, “The design of the Borealis stream processing engine,” presented at the 2nd Biennial Conference on Innovative Data Systems Research, CIDR 2005, pp. 277–289.

[14] N. Marz and J. Warren, 2015, *Big data: principles and best practices of scalable real-time data systems*. Shelter Island, NY: Manning.

[15] A. D. S. Veith, M. D. De Assuncao, and L. Lefevre, 2023, “Latency-Aware Strategies for Deploying Data Stream Processing Applications on Large Cloud-Edge Infrastructure,” *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 445–456, doi: 10.1109/TCC.2021.3097879.

[16] H. Liu, Q. Chen, and P. Liu, 2023, “An Optimization Method of Large-Scale Video Stream Concurrent Transmission for Edge Computing,” *Mathematics*, vol. 11, no. 12, doi: 10.3390/math11122622.

[17] O. Boykin, S. Ritchie, I. O’Connell, and J. Lin, “Summingbird: A framework for integrating batch and online MapReduce computations,” presented at the Proceedings of the VLDB Endowment, 2014, pp. 1441–1451. doi: 10.14778/2733004.2733016.

[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, Aug. 2012, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, in MCC ’12. New York, NY, USA:

Association for Computing Machinery, pp. 13–16. doi: 10.1145/2342509.2342513.

[19] R. Tudoran, A. Costan, O. Nano, I. Santos, H. Soncu, and G. Antoniu, Jan. 2016, "JetStream: Enabling high throughput live event streaming on multi-site clouds," *Future Gener. Comput. Syst.*, vol. 54, pp. 274–291, doi: 10.1016/j.future.2015.01.016.

[20] J. Morales, E. Rosas, and N. Hidalgo, Jun. 2014, "Symbiosis: Sharing mobile resources for stream processing," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, Funchal, Madeira, Portugal: IEEE, pp. 1–6. doi: 10.1109/ISCC.2014.6912641.

[21] S. Sarkar, S. Chatterjee, and S. Misra, Jan. 2018, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, doi: 10.1109/TCC.2015.2485206.

[22] S. Agarwal, S. Yadav, and A. Yadav, 2016, "An architecture for elastic resource allocation in Fog Computing," *IJIEEB*, vol. 8, pp. 48-61, doi:10.5815/ijieeb.2016.01.0.