TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────

ĐỒ ÁN

**TỐT NGHIỆP ĐẠI HỌC**

NGÀNH CÔNG NGHỆ THÔNG TIN

**Xây dựng ứng dụng, thử nghiệm và đánh giá hiệu quả của mô hình dự đoán PD-GABP trong dự đoán co giãn tài nguyên trên hạ tầng đám mây OpenStack.**

Sinh viên thực hiện : **Vũ Trung Dũng**

Lớp KSTN CNTT – K57

Giáo viên hướng dẫn: TS. **Nguyễn Bình Minh**

HÀ NỘI 05-2017

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Vũ Trung Dũng

Điện thoại liên lạc: 01678136637 Email: vutrungdungbk@gmail.com

Lớp: KSTN CNTT K57 Hệ đào tạo: Tài năng và Chất lượng cao

Đồ án tốt nghiệp được thực hiện tại: Bộ môn Hệ Thống Thông Tin, Viện Công Nghệ Thông Tin và Truyền Thông, Đại học Bách Khoa Hà Nội.

Thời gian làm ĐATN: Từ ngày 01/03/2017 đến 20/8/2017.

2. Mục đích nội dung của ĐATN

Xây dựng ứng dụng và thử nghiệm, đánh giá hiệu quả của mô hình dự đoán PD-GABP được đưa ra trong bài báo khoa học “PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment” của các tác giả Đặng Trần, Nhuận Trần, Bình Minh Nguyễn và Hiếu Lê, trong việc dự đoán tài nguyên tiêu thụ và co giãn tài nguyên tự động trên đám mây OpenStack.

3. Các nhiệm vụ cụ thể của ĐATN

- Xây dựng ứng dụng theo dõi và dự đoán co giãn tài nguyên dựa trên mô hình dự đoán PD GABP.

- Triển khai ứng dụng và môi trường thử nghiệm trên đám mây OpenStack.

- Thử nghiệm và đánh giá hiệu quả của mô hình dự đoán PD GABP trên môi trường thử nghiệm.

4. Lời cam đoan của sinh viên:

Tôi – *Vũ Trung Dũng* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS. Nguyễn Bình Minh*.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm 2017*  Tác giả ĐATN  *Vũ Trung Dũng* |

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm*  Giáo viên hướng dẫn |

*TS. Nguyễn Bình Minh*

TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Trên thực tế việc triển khai ứng dụng trên đám mây luôn có hai vấn đề cần phải giải quyết: cung cấp đủ tài nguyên để đảm bảo ứng dụng hoạt động không bị gián đoạn; và hạn chế lượng tài nguyên dư thừa để tiết kiệm chi phí. Mô hình dự đoán PD-GABP được trình bày trong bài báo “PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment” của nhóm tác giả Dang Tran, Nhuan Tran, Binh Minh Nguyen, Hieu Le, là mô hình sử dụng dữ liệu tài nguyên tiêu thụ của ứng dụng để dự đoán trước tài nguyên tiêu thụ trong tương lai. Mô hình này đã chứng minh độ chính xác rất tốt trên lý thuyết. Việc áp dụng mô hình dự đoán PD-GABP vào đám mây sẽ giúp việc co giãn tài nguyên hiệu quả hơn nhờ khả năng dự đoán trước tài nguyên tiêu thụ, từ đó biết trước được thời điểm tài nguyên quá tải để đưa ra quyết định giãn tài nguyên. Đây là việc rất khó trong thực tế. Vì thế, đóng góp của đồ án là đưa mô hình này thử nghiệm trong môi trường thực tế, cụ thể là trên đám mây OpenStack, để đánh giá hiệu quả và khả năng ứng dụng của mô hình này. Đồ án đã giải quyết được các vấn đề sau:

* Xây dựng ứng dụng theo dõi, dự đoán và co giãn tài nguyên trên đám mây OpenStack dựa trên mô hình dự đoán PD-GABP.
* Triển khai ứng dụng trên môi trường thử nghiệm, mô phỏng dữ liệu thực để chạy ứng dụng đó.
* Thử nghiệm và đánh giá hiệu quả của mô hình dự đoán PD-GABP trên môi trường thử nghiệm.

**Cấu trúc đồ án này bao gồm 3 chương:**

* *Chương 1: Đặt vấn đề, các phương pháp và công nghệ sử dụng.*

Chương này sẽ đưa ra bài toán, khảo sát các dịch vụ thực tế, giới thiệu thuật toán PD-GABP và hướng giải quyết bài toán dựa vào các công nghệ sử dụng.

* *Chương 2: Các kết quả đạt được.*

Chương này đi vào chi tiết thiết kế ứng dụng và thực hiện các thử nghiệm và đưa ra đánh giá.

* *Chương 3: Kết luận*.

ABSTRACT OF THESIS

The aim of this thesis is to develop a basic auto scaling system in cloud environment powered by OpenStack software. This system is based on a novel prediction model presented in the paper named “PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment” written by Dang Tran, Nhuan Tran, Binh Minh Nguyen, Hieu Le in Hanoi University of Science and Technology. This system thus can enhance the application performances running on cloud environment with the ability that to forecast the application resource consumptions in the future by measurement of the one in the past. That leads cloud will scale out before the resources are overload. However, as based on a prediction method, this system has risk of wrong prediction, so it has a lot of work to do to apply this system in reallife cloud environment. This thesis contributes is to develop a forecast and auto scale system in OpenStack cloud environment used real resource consumptions and scale real cloud resources.

LỜI CẢM ƠN

Thực hiện đồ án tốt nghiệp luôn luôn có rất nhiều khó khăn và thử thách cần phải vượt qua. Để hoàn thành đồ án này không chỉ cần sự có nỗ lực của bản thân mà quan trọng hơn hết là sự giúp đỡ cả về kiến thức lẫn tinh thần của thầy cô, bạn bè và người thân.

Lời đầu tiên, tôi xin trân thành cảm ơn TS. Nguyễn Bình Minh – Viện Công nghệ thông tin và Truyền thông, Đại học Bách Khoa Hà Nội đã hướng dẫn tôi tận tình trong quá trình thực hiện đồ án này.

Tôi cũng xin cảm ơn các thầy cô, bạn bè đã truyền đạt và trao đổi những kiến thức quý báu trong suốt quá trình 5 năm học tập. Tôi cũng xin cảm ơn bạn Trần Văn Đặng đã hỗ trợ giải đáp những thắc mắc trong thời gian làm đồ án.

Cuối cùng, con xin gửi lời cảm ơn đến gia đình đã luôn quan tâm, tin tưởng ngay cả trong thời gian khó khăn thử thách nhất mà con cần vượt qua, đã động viên và là chỗ dựa tinh thần vững chắc giúp con hoàn thành đồ án tốt nghiệp này.

Hà Nội, ngày 30 tháng 5 2017

Vũ Trung Dũng

Lớp KSTN CNTT K57

Viện CNTT&TT – ĐH Bách Khoa HN

**Mục lục**

[PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP 2](#_Toc490522353)

[TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP 3](#_Toc490522354)

[ABSTRACT OF THESIS 4](#_Toc490522355)

[LỜI CẢM ƠN 5](#_Toc490522356)

[DANH MỤC HÌNH ẢNH 8](#_Toc490522357)

[DANH MỤC BẢNG BIỂU 9](#_Toc490522358)

[DANH MỤC CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ 10](#_Toc490522359)

[CHƯƠNG 1: ĐẶT VẤN ĐỀ, PHƯƠNG PHÁP VÀ CÁC CÔNG NGHỆ SỬ DỤNG. 11](#_Toc490522360)

[1.1. Đặt vấn đề. 11](#_Toc490522361)

[1.1.1. Điện toán đám mây. 11](#_Toc490522362)

[1.1.2. Nhu cầu co giãn tài nguyên tự động. 13](#_Toc490522363)

[1.1.3. Các phương pháp co giãn tài nguyên trên đám mây. 13](#_Toc490522364)

[1.1.4. Các phương pháp tự động co giãn tài nguyên. 15](#_Toc490522365)

[1.2. Mô hình dự đoán PD-GABP 19](#_Toc490522366)

[1.3. OpenStack. 23](#_Toc490522367)

[1.4. Mục đích, phạm vi đồ án và các công nghệ sử dụng. 25](#_Toc490522368)

[1.4.1. Mục đích của đồ án. 25](#_Toc490522369)

[1.4.2. Phạm vi của đồ án. 25](#_Toc490522370)

[1.4.3. Các công nghệ sử dụng. 26](#_Toc490522371)

[1.5. Tóm tắt chương. 28](#_Toc490522372)

[CHƯƠNG 2: XÂY DỰNG ỨNG DỤNG, TRIỂN KHAI MÔI TRƯỜNG THỬ NGHIỆM VÀ THỰC HIỆN THỬ NGHIỆM 29](#_Toc490522373)

[2.1. Xây dựng ứng dụng dự đoán co giãn tài nguyên. 29](#_Toc490522374)

[2.1.1. Phân tích chức năng. 29](#_Toc490522375)

[2.1.2. Kiến trúc phần mềm. 30](#_Toc490522376)

[2.1.3. Biểu đồ hành động các luồng thực hiện quan trọng. 32](#_Toc490522377)

[2.1.4. Cấu trúc dữ liệu. 35](#_Toc490522378)

[2.1.5. Các API hệ thống cung cấp. 36](#_Toc490522379)

[2.1.6. Phương pháp theo dõi máy ảo. 36](#_Toc490522380)

[2.1.7. Giao diện chương trình. 39](#_Toc490522381)

[2.2. Triển khai môi trường thử nghiệm và thực hiện thử nghiệm. 43](#_Toc490522382)

[2.2.1. Mục đích. 43](#_Toc490522383)

[2.2.2. Dữ liệu thử nghiệm. 44](#_Toc490522384)

[2.2.3. Môi trường thử nghiệm. 47](#_Toc490522385)

[2.2.4. Thử nghiệm. 47](#_Toc490522386)

[2.2.5. Kết luận. 0](#_Toc490522387)

[CHƯƠNG 3: KẾT LUẬN 2](#_Toc490522388)

[**TÀI LIỆU THAM KHẢO** 3](#_Toc490522389)

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ

CHƯƠNG 1: ĐẶT VẤN ĐỀ, PHƯƠNG PHÁP VÀ CÁC CÔNG NGHỆ SỬ DỤNG.

* 1. Đặt vấn đề.
     1. Điện toán đám mây.

Điện toán đám mây (Cloud computing) được biết đến từ giữa năm 2007 là công nghệ rất được quan tâm trong những năm trở lại đây. Nó được định nghĩa bởi Viện tiêu chuẩn và công nghệ quốc gia của Mỹ (NIST) như sau[2]:

“Điện toán đám mây là mô hình cho phép truy cập qua mạng để lựa chọn và sử dụng tài nguyên có thể được tính toán (ví dụ: mạng, máy chủ, lưu trữ, ứng dụng và dịch vụ) theo nhu cầu một cách thuận tiện và nhanh chóng; đồng thời cho phép kết thúc sử dụng dịch vụ, giải phóng tài nguyên dễ dàng, giảm thiểu các giao tiếp với nhà cung cấp”.

Điện toán đám mây được phát triển dựa trên công nghệ ảo hóa, đây là công nghệ lõi của mọi nền tảng đám mây.

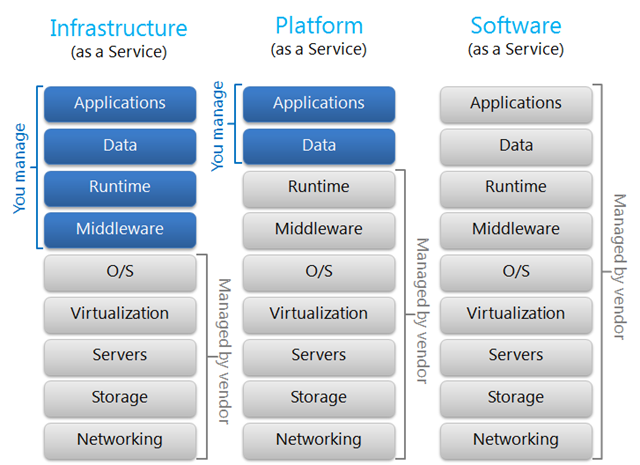
Ảo hóa ở đây là ảo hóa phần cứng, thiết bị mạng, thiết bị lưu trữ … , có thể quản lý bằng phần mềm và lập trình được, nằm ở giữa hệ thống phần cứng máy tính và phần mềm chạy trên nó. Nhờ đó phần cứng máy tính có thể chia nhỏ như là các máy chạy độc lập có hệ điều hành riêng với đầy đủ các thiết bị ảo cần thiết. Cho nên các đám mây có thể cấp phát, giải phóng tài nguyên dễ dàng. Các công nghệ ảo hóa có thể kể đến là KVM, VirtualBox, VMWare, …

Lợi ích của việc dùng đám mây có thể thấy rõ khi so sánh với việc dùng máy chủ vật lý theo kiểu truyền thống[3]:

* **Tiết kiệm chi phí (Cost Savings)**: Các công ty có thể tiết kiệm chi phí so với việc duy trì một hệ thống máy chủ vật lý vốn tốn diện tích và nhân lực, trong khi vẫn đảm bảo hiệu năng cần thiết khi dùng dịch vụ đám mây. Với mô hình trả phí “pay as you go”, người dùng đám mây chỉ trả chi phí theo nhu cầu sử dụng của mình.
* **Khả năng mở rộng, mềm dẻo (Scalability/Flexibility)**: Việc triển khai ứng dụng trên đám mây với khả năng mở rộng nhanh và dễ dàng giúp ứng dụng chạy ổn định hơn ngay cả khi lượng người dùng tăng cao khiến tài nguyên bị quá tải. Tài nguyên trên đám mây có thể dễ dàng thêm hoặc bớt tùy nhu cầu sử dụng, giúp tiết kiệm tài nguyên cho nhà cung cấp và chi phí cho bên sử dụng đám mây.
* **Đáng tin cậy (Reliability)**: Dịch vụ đám mây có độ tin cậy cao, do các nhà cung cấp dịch vụ đám mây với đội ngũ chuyên môn của mình đã có sẵn các dịch vụ giúp giảm thiểu rủi ro như tự động sao lưu, khôi phục… trong suốt với bên sử dụng đám mây.
* **Dễ theo dõi, bảo trì (Maintenance)**: Người sử dụng đám mây có thể theo dõi tình trạng máy chủ ảo, điều khuyển từ xa máy chủ… qua các API của các nhà cung cấp đám mây.
* **Dễ truy cập (Accessible):** Có thể truy cập đám mây mọi lúc, mọi nơi.

Đám mây có 3 mô hình dịch vụ:

* **Mô hình dịch vụ hạ tầng (Infrastructure as a Service - IaaS)**: là mô hình mà người dùng có thể thuê ổ cứng, mạng… và các tài nguyên khác từ nhà cung cấp và tùy ý sử dụng như một máy ảo riêng biệt. Các nhà cung cấp: Oracle, Amazon AWS, Microsoft Azure, Google Compute Engine, Rackspace, IBM…
* **Mô hình dịch vụ nền tảng (Platform as a Service - PaaS)**: là mô hình mà người dùng có thể cài đặt các ứng dụng của mình trên nền tảng có sẵn, sử dụng các API của nhà cung cấp, nhưng không thể thay đổi phần dưới như hệ điều hành, cài đặt mạng, dung lượng ổ cứng … Các nhà cung cấp: RedHat OpenShift, Google App Engine, Heroku, Microsoft Azure, Amazon AWS…
* **Mô hình dịch vụ phần mềm (Software as a Service – SaaS)**: là mô hình mà dịch vụ được cung cấp dưới dạng ứng dụng chạy trên nền tảng đám mây, có thể kết nối để sử dụng từ nhiều thiết bị như điện thoại, máy tính… tuy nhiên người dùng chỉ sử dụng dịch vụ, không thể thay đổi cài đặt. Ví dụ: Google Mail, Google Drive, Dropbox…



Hình : Ba mô hình đám mây. Nguồn: [13]

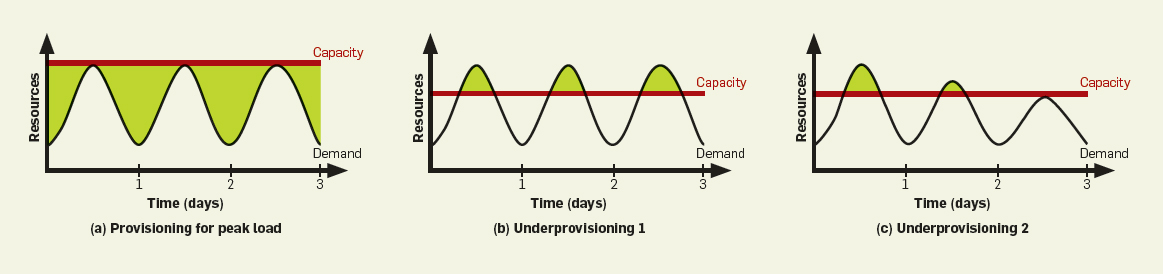
Trong đó, mô hình IaaS là mô hình mà người dùng đám mây có thể tùy biến về tài nguyên, hệ điều hành, … một cách tự do, tùy nhu cầu sử dụng. Còn mô hình PaaS, việc tùy biến tài nguyên theo nhu cầu do nhà cung cấp dịch vụ máy ảo, người dùng cài đặt hoặc đưa ra yêu cầu, hệ thống sẽ thực hiện tự động cho họ. Ngoài ra, mô hình PaaS còn cung cấp các phần mềm nền tảng như cơ sở dữ liệu, API, web runtime, các dịch vụ co giãn tài nguyên tự động v.v…

* + 1. Nhu cầu co giãn tài nguyên tự động.

Như đã giới thiệu ở phần trước, một trong các ưu điểm của đám mây là Tiết kiệm chi phí (cost savings). Tức là, người dùng chỉ phải trả tiền thuê tài nguyên mà họ thỏa thuận với nhà cung cấp đám mây, hoặc tính phí theo lượng dùng hàng ngày (pay as you go), hoặc cả hai.

Ví dụ, một người muốn sử dụng dịch vụ PaaS của Amazon EC2, Amazon sẽ cho anh ta lựa chọn hệ điều hành và thiết lập capacities phù hợp với nhu cầu. Khái niệm capacities ở đây là lượng tài nguyên muốn thuê, ví dụ với 1 vCPU, 500MB RAM, storage EBS Only và hệ điều hành Linux/UNIX, người dùng sẽ phải trả 0.0059$ cho mỗi giờ sử dụng.

Nhưng không phải lúc nào lượng tài nguyên người dùng thuê cũng đáp ứng đủ nhu cầu. Cũng có khi lượng tài nguyên được thuê bị dư thừa gây lãng phí. Thực tế cho thấy lượng tài nguyên sử dụng của người dùng thường có chu kỳ theo ngày, do gắn liền với thời gian sinh hoạt của con người. Do đó, nhà cung cấp đám mây phải có phương án sao cho cấp đủ tài nguyên đảm bảo chất lượng dịch vụ cho người dùng, cũng như tránh cấp thừa tài nguyên gây lãng phí cho nhà cung cấp. Nếu sử dụng đám mây của các nhà cung cấp như Amazon AWS, Microsoft Azure, Rackspace… người dùng có thể dễ dàng thiết lập dịch vụ co giãn tự động cung cấp sẵn và không phải lo về chất lượng dịch vụ, tất nhiên họ trả tiền theo tài nguyên sử dụng (pay as you go). Còn vấn đề co giãn tài nguyên là về phía nhà cung cấp, việc co giãn tài nguyên hiệu quả hay không sẽ ảnh hưởng quan trọng tới việc giữ chân người dùng.



Hình : So sánh tài nguyên tiêu thụ (Demand) và tài nguyên được cấp (Capacity)[5]

Ví dụ, hình a), tài nguyên được cấp luôn lớn hơn tài nguyên tiêu thụ, dư thừa nhiều. Hình b, c) tài nguyên được cấp không đủ trong một nửa chu kỳ và dư thừa ở nửa chu kỳ còn lại.

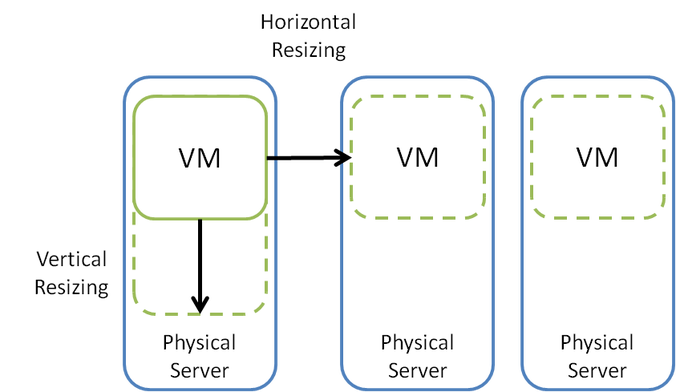
* + 1. Các phương pháp co giãn tài nguyên trên đám mây.

Co giãn tài nguyên trên đám mây là khả năng điều chỉnh lượng tài nguyên cấp cho người dùng đám mây dựa trên lượng tài nguyên tiêu thụ (cần để chạy ứng dụng), có thể theo một trong 2 hướng thêm hoặc bớt.

Co giãn tài nguyên có thể thực hiện được trên đám mây nhờ tính khả năng mở rộng, mềm dẻo (Scalability/Flexibility) của nó.

Có 2 phương pháp co giãn tài nguyên[6]:

* **Theo chiều ngang**: Một nhóm các máy chủ ảo có cùng chức năng, khi nhóm này quá tải, hệ thống tạo thêm một hoặc một số máy chủ ảo giống với các máy chủ ảo đó và thêm vào nhóm, và hủy bớt máy ảo khi tài nguyên không dùng hết. Yêu cầu máy chủ tạo thêm phải cung cấp các dịch vụ giống với các máy chủ trước đó.
* **Theo chiều dọc**: Mỗi máy chủ ảo có chức năng khác nhau, ví dụ máy chủ web, máy chủ dữ liệu, máy chủ dịch vụ…, mỗi máy được cấp cho một lượng tài nguyên RAM, CPU nhất định. Khi một máy quá tải, nó được cấp thêm một lượng RAM, CPU cần thiết để hoạt động bình thường.

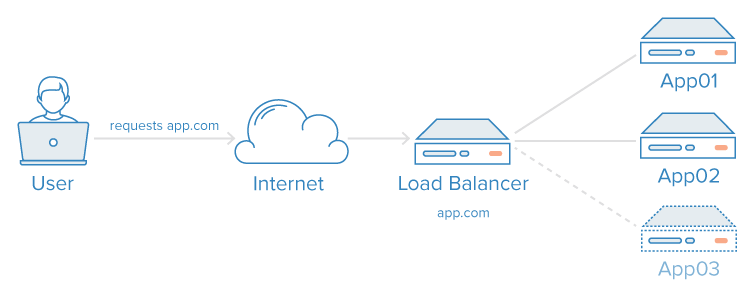


Hình : Co giãn theo chiều dọc (Vertical Resizing) và co giãn theo chiều ngang (Horizontal Resizing) [6]

Co giãn theo chiều ngang được dùng nhiều trong thực tế bởi phương pháp này cho phép co giãn không giới hạn và không phụ thuộc vào một máy duy nhất, khi một máy hỏng, tạo lại máy khác. Còn phương pháp co giãn theo chiều dọc khi một máy hỏng thì gây dừng cả hệ thống.

Mô hình chung của phương pháp co giãn theo chiều ngang:

* Xác định nhóm các máy ảo cùng chức năng, nằm sau các cân bằng tải (load balancer).
* Theo dõi tài nguyên tiêu thụ trên các máy ảo đó.
* Thêm, bớt máy ảo giống với các máy ảo định nghĩa trong nhóm, đăng ký với cân bằng tải tương ứng.



Hình : Ví dụ co giãn tài nguyên theo chiều ngang [7].

* + 1. Các phương pháp tự động co giãn tài nguyên.
       1. Phương pháp chung.

Hiện nay có 3 phương pháp co giãn tài nguyên chính[1]:

1. **Dựa trên chu kỳ (Periodicity):**

Trong quá trình sử dụng, người thuê dịch vụ đám mây có được thông tin tài nguyên sử dụng theo chu kỳ (giờ, ngày, tháng … ) nhờ các công cụ theo dõi và trích xuất báo cáo từ nhà cung cấp dịch vụ. Dựa vào đó, người quản trị sẽ cài đặt, lập lịch co giãn tài nguyên theo chu kỳ đó.

Ví dụ với một trang web đọc báo, lượng người dùng có thể tăng cao vào các buổi sáng và tối, giảm bớt trong giờ làm việc và cực thấp vào buổi đêm. Khi đó người cung cấp dịch vụ web có thể lập lịch với nhà cung cấp đám mây thêm một máy ảo từ 7h sáng đến 9h sáng, từ 6h tối đến 12h tối, giảm bớt một máy trong thời gian còn lại và duy trì một hoặc hai máy vào buổi đêm.

1. **Dựa trên ngưỡng(Threshold):**

Người dùng dịch vụ đám mây đặt ngưỡng tài nguyên “cao” và “thấp”, khi tài nguyên ví dụ như CPU, RAM, lượng kết nối… lớn hơn mức “cao” thì giãn tài nguyên, nhỏ hơn mức “thấp” thì co tài nguyên. Phương pháp này được sử dụng như giải pháp tức thời với những thay đổi tài nguyên đột biến trong quá trình sử dụng mà không lường trước được.

Đặc điểm của nó là ngưỡng cao phải chọn phù hợp mới đạt hiệu quả. Bởi vì một đám mây muốn tạo máy ảo mới cần thời gian, bao gồm thời gian cấp phát của hệ thống, khởi chạy máy ảo, cài đặt dịch vụ, tổng thời gian có thể tính bằng phút. Do đó ngưỡng phải chọn sao cho đảm bảo hệ thống không bị quá tải cho đến khi tài nguyên sẵn sàng được sử dụng.

1. **Dựa trên dự đoán (Prediction):**

Phương pháp này dùng dữ liệu tài nguyên tiêu thụ của hệ thống trong quá khứ và hiện tại đưa vào một mô hình để dự đoán lượng tài nguyên tiêu thụ trong tương lai. Dựa vào dự đoán này có thể co giãn tài nguyên trước khi hệ thống quá tải mà không phải đặt ngưỡng. Tuy nhiên hiệu quả của phương pháp phụ thuộc rất lớn vào thuật toán và mô hình sử dụng để dự đoán. Đã có nhiều bài nghiên cứu về phương pháp này, nhưng tính chính xác của các phương pháp đó khó có thể áp dụng trong thực tế.

* + - 1. Giải pháp của các nhà cung cấp.

Trong thực tế, 2 phương pháp Dựa vào chu kỳ (Periodicity) và Dựa vào ngưỡng (Threshold) đang được nhiều nhà cung cấp dịch vụ đám mây áp dụng.

Ví dụ như Amazon[8], họ cung cấp dịch vụ tự động co giãn tài nguyên (dịch vụ Auto Scale) cho phép người dùng định nghĩa các nhóm (group) gọi là các Auto Scaling Groups. Mỗi nhóm bao gồm các máy ảo (instance) đang chạy với số máy ảo tối thiểu, số máy ảo tối đa trong nhóm là xác định. Người dùng cần định nghĩa các cài đặt khởi tạo máy ảo (launch configuration) như AMI ID (Amazon Machine Image ID – định danh image của máy ảo), hệ điều hành, cặp khóa mã đối xứng (key pair), các security groups rules và ổ cứng ảo (block device) để khởi tạo máy ảo mới. Các nhóm đều được gán với các kế hoạch co giãn tài nguyên (scaling plans) xác định. Sau đó, các máy ảo trong nhóm này được theo dõi theo một hoặc nhiều số đo (metric, ví dụ CPU, Ram…) bởi các Amazon CloudWatch, và khi đạt điều kiện phải co giãn, các CloudWatch này sẽ gửi thông báo tới mô dun Auto Scaling để tiến hành tạo hoặc hủy máy ảo. Ngoài ra, các CloudWatch còn cung cấp một dịch vụ tự động kiểm tra trạng thái của máy ảo (dịch vụ Health Checking) cho nhóm, hệ thống tự động kiểm tra máy ảo bị lỗi, hủy nó đi và tạo máy ảo mới theo các cài đặt (launch configuration) đã định nghĩa. Không chỉ thế, Amazon còn đưa ra dịch vụ cân bằng tải gọi là Elastic Load Balancing giúp các ứng dụng chạy trên máy ảo mới có thể được sử dụng từ bên ngoài. Tất nhiên tất cả các dịch vụ này đều phải trả phí.

Một số dịch vụ của các nhà cung cấp lớn và phần mềm OpenStack.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dịch vụ | Amazon EC2 | Microsoft Azure | OpenStack | Phương pháp | Mô tả |
| Manual Scaling | Có | Có | Không |  | Điều chỉnh tài nguyên bằng tay |
| Scale based on schedule | Có | Có | Không | Periodicity | Theo kế hoạch. |
| Scale based on demand | Có | Có | Heat  /Ceilometer | Threshold | Theo lượng dùng. |

Bảng : Khảo sát các phương pháp tự động co giãn tài nguyên trên thực tế.

Mặc dù phương pháp Dựa trên dự đoán có những ưu điểm hơn 2 phương pháp Dựa trên chu kỳ và Dựa trên ngưỡng, tuy nhiên trong thực tế phương pháp này chưa được các nhà cung cấp dịch vụ áp dụng. Nếu được nghiên cứu và áp dụng trong thực tế, phương pháp Dựa trên dự đoán có thể đem lại lợi nhuận rất lớn cho các nhà cung cấp dịch vụ đám mây.

* + - 1. So sánh các phương pháp.
  1. **Phương pháp Dựa trên chu kỳ (Periodicity):**

**Ưu điểm:**

* *Dễ thực hiện với nhà cung cấp.*

Ví dụ muốn sử dụng dịch vụ này, người dùng cần lập lịch co giãn tài nguyên, có thể là theo giờ trong ngày, hoặc theo ngày trong tuần. Nhà cung cấp chỉ cần thực hiện co giãn theo lịch mà người dùng yêu cầu.

* *Giải quyết được trường hợp tài nguyên tiêu thụ có chu kỳ dài.*

Như trên, người dùng lập lịch co giãn tài nguyên theo chu kỳ đã được người dùng tính toán trước.

**Nhược điểm:**

* *Sử dụng chu kỳ để co giãn tài nguyên và yêu cầu người dùng dịch vụ đám mây phải lập lịch bằng tay, do đó họ phải theo dõi chu kỳ trong khoảng thời gian dài để có thể lập lịch một cách chính xác.*
* *Vì dựa trên kinh nghiệm của người lập lịch nên không có khả năng phản ứng với các thay đổi tức thời, đột biến.*
  1. **Phương pháp Dựa trên ngưỡng (Threshold).**

**Ưu điểm:**

* *Dễ thực hiện với nhà cung cấp.*

Phương pháp này yêu cầu người dùng dịch vụ thiết lập các tham số ngưỡng “cao” và “thấp”. Nhà cung cấp theo dõi tài nguyên tiêu thụ của máy ảo và có các thuật toán quyết định khi so sánh dữ liệu tài nguyên tiêu thụ với các ngưỡng, từ đó có các quyết định co giãn tài nguyên.

* *Giải quyết được các phát sinh đột biến của tài nguyên tiêu thụ.*

Phương pháp này dựa trên việc theo dõi tài nguyên tiêu thụ nên có thể phát hiện được phát sinh tiêu thụ tài nguyên đột biến của máy ảo.

**Nhược điểm:**

* *Trên thực tế người dùng đám mây rất khó chọn được một ngưỡng phù hợp.*
* *Phần lớn trường hợp khó có thể cân bằng giữa tối ưu chi phí và tính ổn định của hệ thống.*

Việc co giãn tài nguyên luôn cần một lượng thời gian đủ để hệ thống cấp phát hoặc thu hồi tài nguyên, có thể là vài phút hoặc nhiều hơn nếu cần cài đặt nhiều khi khởi chạy máy ảo. Do đó ngưỡng “cao” luôn phải đặt ở ngưỡng 80-90% để hệ thống kịp giãn trước khi quá tải, đôi khi đó là sai lầm và khiến tài nguyên giãn mà không cần thiết. Cũng có trường hợp tốc độ cấp phát tài nguyên chậm hơn tốc độ tài nguyên tiêu thụ từ 80-90% cho đến ngưỡng quá tải.

* 1. **Phương pháp Dựa trên dự đoán (Prediction):**

**Ưu điểm:**

* *Dễ dàng với người dùng đám mây.*

Người dùng đám mây không cần cài đặt chu kỳ, ngưỡng như 2 phương pháp trên, họ chỉ cần yêu cầu sử dụng dịch vụ, việc quyết định co giãn tài nguyên, co giãn như thế nào là do nhà cung cấp dịch vụ.

* *Dự đoán trước tài nguyên bị quá tải một khoảng thời gian sao cho thời gian đám mây cấp phát thêm tài nguyên cũng như thời gian tài nguyên sẵn sàng sử dụng là đủ trước khi hệ thống quá tải.*

**Nhược điểm:**

* *Vì là phương pháp dự đoán nên phù thuộc rất nhiều vào độ tốt của thuật toán và mô hình dự đoán.*

Có thể thấy trong dịch vụ đám mây thực tế, phần lớn các phương pháp tự động co giãn tài nguyên được dùng là 2 phương pháp dựa trên chu kỳ và dựa trên ngưỡng, bởi vì 2 phương pháp này có đặc điểm:

* *Đơn giản, trực quan có thể bù đắp những nhược điểm của nhau.*

Với phương pháp Dựa trên chu kỳ dùng để co giãn theo chu kỳ dài, phương pháp Dựa trên ngưỡng dùng để xử lý các trường hợp đột biến là đủ để giải quyết hầu hết các bài toán trên thực tế.

* *Trách nghiệm cài đặt tham số là của người dùng.*

Các tham số ví dụ ngưỡng, lập lịch co giãn là do người dùng cài đặt, có thể coi như là thỏa thuận với nhà cung cấp. Do đó nhà cung cấp chỉ có nhiệm vụ thực hiện đúng thỏa thuận và việc đảm bảo các cài đặt là hiệu quả với thực tế là trách nghiệm cài đặt của người dùng.

Tuy nhiên cần xem xét tới phương pháp Dựa trên dự đoán, bởi vì phương pháp này có những ưu điểm so với các phương pháp trên:

* *Phương pháp này có thể khắc phục được nhược điểm của phương pháp Dựa trên ngưỡng*, đó là không tối ưu giữa chi phí tiết kiệm và chất lượng dịch vụ, bởi vì phương pháp này có thể dự đoán để giãn tài nguyên trước khi quá tải xảy ra mà không cần đặt ngưỡng.
* *Dễ dàng cho người dùng đám mây*, do đó lôi kéo được nhiều người dùng dịch vụ đám mây.

**Kết luận chung:**

Có thể thấy phương pháp Dựa trên dự đoán trong vấn đề co giãn tài nguyên mang lại lợi ích rất lớn khi áp dụng vào thực tế. Tuy nhiên như đã nói, phương pháp này khá phức tạp và phụ thuộc rất lớn vào thuật toán và mô hình dự đoán, khó triển khai trên thực tế hơn 2 phương pháp Dựa trên chu kỳ và Dựa trên ngưỡng.

* 1. Mô hình dự đoán PD-GABP
     + 1. Giới thiệu.

*“PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment”* là bài báo nghiên cứu của các tác giả Đặng Trần, Nhuận Trần, Bình Minh Nguyễn, Hiếu Lê, được hỗ trợ bởi Đại Học Bách Khoa Hà Nội như là một phần của dự án *“Research and Development of Semantic Interoperability Framework for IaaS Clouds”* (No.T2015-227).

Mục tiêu của bài báo là đưa ra một mô hình dự đoán dựa trên phân tích chuỗi dữ liệu thời gian thực áp dụng vào dự đoán tài nguyên tiêu thụ trong môi trường điện toán đám mây.

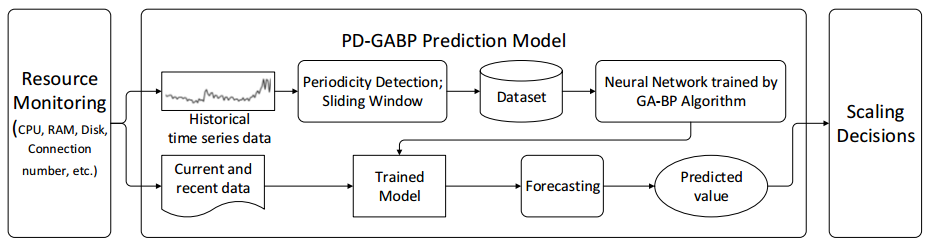
Mô hình mà bài báo đưa ra là kết hợp giữa xác định chu kỳ (PD – Periodicity Detection) của chuỗi dữ liệu thời gian thực và dùng giải thuật tiến hóa (GA – Genetic Algorithm) cùng thuật toán lan truyền ngược (BP – Back Propagation) để tăng độ chính xác của mạng nơ ron. Mạng nơ ron này với đầu vào là chuỗi dữ liệu quá khứ và hiện tại sẽ dự đoán được dữ liệu tương lai. Mô hình này yêu cầu chuỗi dữ liệu thời gian có tính chu kỳ nên hoạt động rất tốt trong bài toán dự đoán tài nguyên tiêu thụ trên đám mây.

Ưu điểm của mô hình mà bài báo đưa ra là có độ chính xác rất tốt khi so sánh với các mô hình khác).

Tuy nhiên bài báo vẫn có hạn chế khi mới chỉ thử nghiệm và chứng minh độ hiệu quả trên bộ dữ liệu số yêu cầu đến trang web World Cup 1998. Trên thực tế còn nhiều độ đo khác cần phải nghiên cứu và cần kết hợp dự đoán trên nhiều độ đo khác nhau để có được kết quả dự đoán tốt. Mô hình đề xuất mới chỉ đưa ra mô hình lấy dữ liệu và dự đoán, để áp dụng vào thực tế cần nhiều việc phải làm.

Sau đây sẽ trình bày cụ thể hơn về mô hình dự đoán PD-GABP mà bài báo đề xuất.

* + - 1. Kiến trúc.



Hình 5: Kiến trúc mô hình dự đoán PD-GABP

Mô hình PD-GABP là mô hình dự đoán sử dụng dữ liệu quá khứ và hiện tại để dự đoán dữ liệu tương lai, dựa trên tử tưởng của phương pháp co giãn dựa trên dự đoán (Prediction). Mô hình gồm các thành phần[1]:

* **Resource monitoring**: lấy dữ liệu tài nguyên tiêu thụ từ các máy ảo đám mây.
* **Prediction Module**: gồm data analysis và prediction.
* **Data analysis** với đầu vào là dữ liệu trong quá khứ thu thập từ Resource monitoring, chạy thuật toán phát hiện chu kỳ (AUTOPERIOD) và sử dụng cửa sổ trượt (sliding window) để tạo tập dữ liệu huấn luyện, cuối cùng được đưa vào một mạng nơ ron kết hợp giải thuật tiến hóa và lan truyền ngược (genetic-backpropagation neural network - GA-BPNN). Dữ liệu đầu vào có thể là phần trăm tiêu thụ CPU, RAM, Disk I/O… Sau khi huấn luyện xong mạng nơ ron, dữ liệu vẫn được tiếp tục lấy bởi Resource monitoring module và được bổ sung vào tập dữ liệu huấn luyện, huấn luyện mạng nơ ron mới sau mỗi để tăng độ chính xác.
* **Prediction** với đầu vào là dữ liệu hiện tại và quá khứ kết hợp với mạng nơ ron đã huấn luyện, dự đoán giá trị tài nguyên tiêu thụ trong tương lai.
* **Scaling Decisions**: dựa vào kết quả dự đoán của module prediction, ra quyết định co giãn tài nguyên.

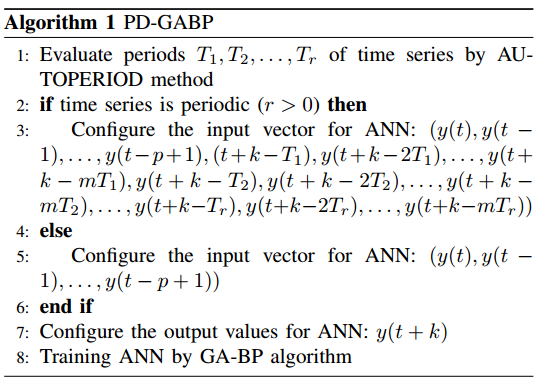
Sau khi mạng nơ ron được huấn luyện và sẵn sàng sử dụng, cứ sau mỗi , dữ liệu mới được lấy một lần và dự đoán dữ liệu tương lai.

* + - 1. Phương pháp phát hiện chu kỳ AUTO PERIOD.
      2. Mô hình PD-GABP.

Bước đầu tiên thuật toán phát hiện chu kỳ của dãy dữ liệu trong quá khứ, ví dụ phát hiện được chu kỳ là . Tiếp theo sử dụng kỹ thuật cửa sổ trượt lấy ra p giá trị , kết hợp lấy các giá trị với là số chu kỳ quá khứ.

Theo đó:

Thuật toán PD-GABP.



Hình : Thuật toán PD-GABP [1]

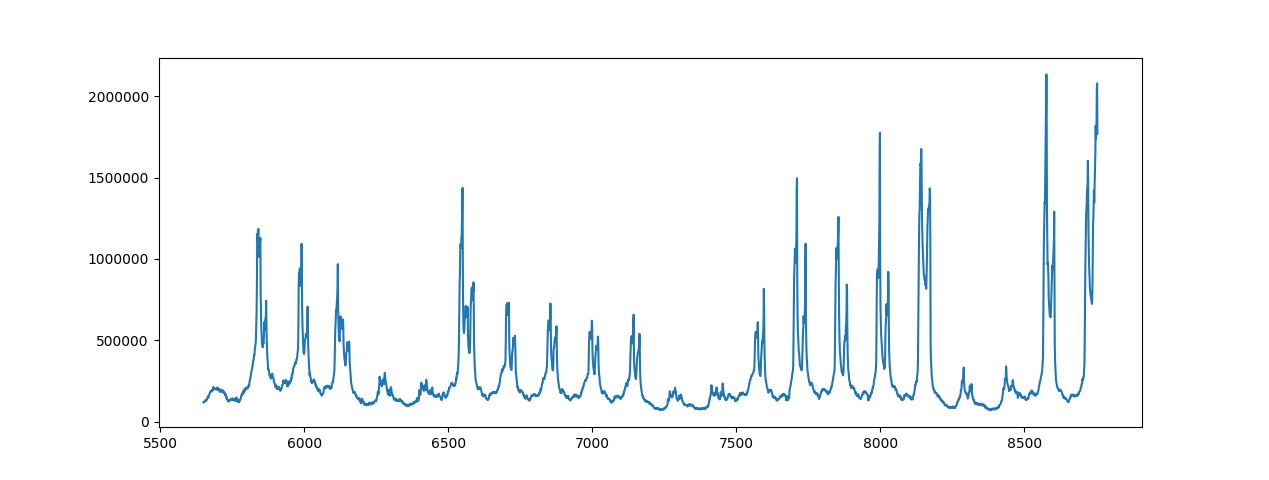
Ý tưởng chính của thuật toán PD-GABP là:

* Bước 1 phát hiện chu kỳ của dữ liệu. (PD – Periodicity Detection)
* Bước 2 xây dựng bộ dữ liệu huấn luyện dựa trên chu kỳ tìm được và cửa sổ trượt.
* Bước 2 kết hợp giải thuật tiến hóa (GAs – Genetic Algorithm) kết hợp với lan truyền ngược (BP – Backpropagation) để huấn luyện mạng nơ ron tăng độ chính xác của dự đoán.

Nhờ áp dụng giải thuật tiến hóa, thuật toán có thể vượt qua cực trị địa phương để tìm được cực trị toàn cục trong quá trình tìm bộ trọng số của mạng nơ ron bằng phương pháp lan truyền ngược. Bài báo đã chứng minh phương pháp GA-BP là tốt hơn các phương pháp huấn luyện truyền thống.

Trong khuôn khổ đồ án, tôi không đi vào trình bày chi tiết cụ thể thuật toán PD-GABP, thuật toán đã được trình bày trong bài nghiên cứu “PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment”[1] ở mục tài liệu tham khảo.

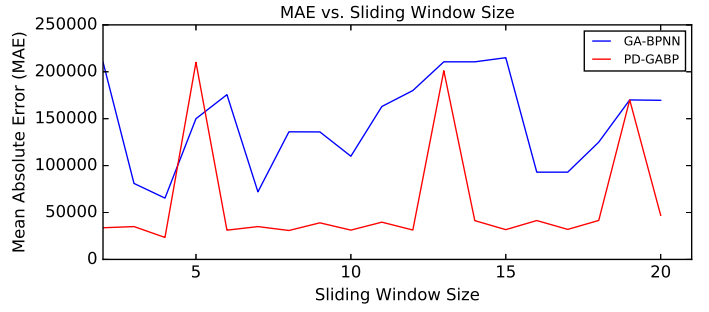
* + - 1. Nhắc lại độ hiệu quả trên lý thuyết của mô hình PD-GABP.



Hình : Dữ liệu số yêu cầu gửi đến trang web World Cup 1998.

Bài báo đưa ra thử nghiệm chứng minh tính chính xác bằng bộ dữ liệu số yêu cầu gửi đến trang web World Cup 1998 lấy mỗi 10 phút. Thuật toán phát hiện chu kỳ là khoảng 1 ngày (khoảng 142 điểm dữ liệu). Dữ liệu từ ngày 40 đến 46 được đưa vào huấn luyện mạng, từ ngày 46 đến 47 được dùng để thử nghiệm. Dùng hàm đánh giá MAE (Mean absolute error) với là kết quả và là giá trị ban đầu.

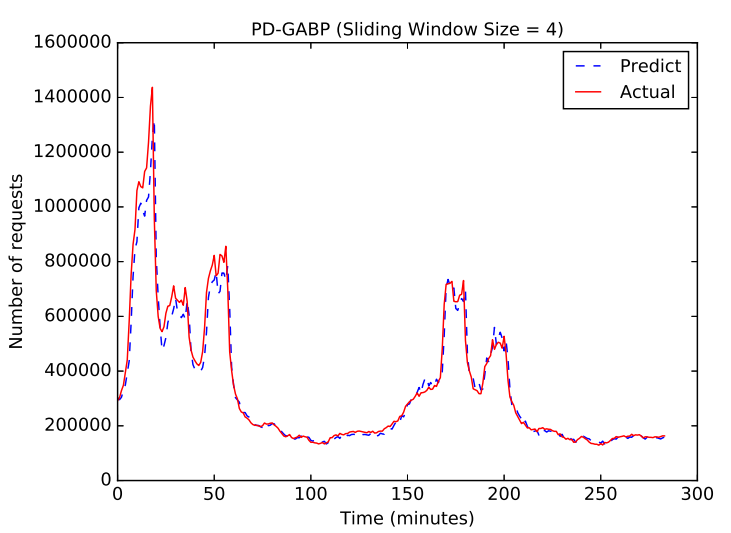
Kết quả thử nghiệm với .



Hình : Kết quả MAE chạy PD-GABP với m=1, r=1 với p khác nhau [1].

Theo kết quả trên, p = 4 có kết quả MAE tốt nhất (giá trị MAE càng nhỏ càng tốt).

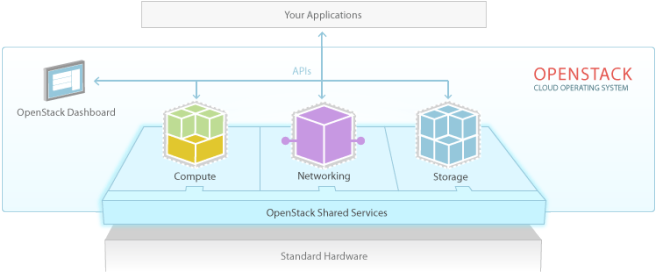
Kết quả thử nghiệm với độ rộng cửa sổ Window Size = 4



Hình : So sánh kết quả dự đoán PD-GABP, p=4, m=1, r=1 với dữ liệu thật [1].

Đồ án sẽ thực hiện thử nghiệm sẽ dựa trên kết quả thực nghiệm của nghiên cứu, với các giá trị tham số cho GABP như thực nghiệm, độ rộng cửa sổ là 4 và chu kỳ 1.

* 1. OpenStack.



Hình : Các thành phần cơ bản của OpenStack [10].

OpenStack là phần mềm mã nguồn mở, chủ yếu viết bằng python dùng để triển khai mô hình Dịch vụ hạ tầng đám mây (IaaS). Nó được khởi xướng bởi Rackspace Hosting và NASA từ năm 2010 và được OpenStack Foundation quản lý từ năm 2015. OpenStack ra phiên bản mới mỗi 6 tháng một lần, chữ cái đầu của tên phiên bản theo thứ tự trong bảng chữ cái, phiên bản mới nhất tính tới thời điểm viết đồ án là Ocata.

OpenStack có thể cài đặt trên một hoặc nhiều máy (gọi là các nodes), thậm chí là máy tính cá nhân.

Vì dùng để triển khai IaaS, nó thường không được dùng bởi những người dùng phổ thông mà được dùng để xây dựng các hệ thống đám mây phục vụ doanh nghiệp với đội ngũ kỹ thuật có chuyên môn, quản trị riêng. Tuy nhiên OpenStack còn cần hoàn thiện hơn nếu muốn xây dựng các hệ thống đám mây lớn. OpenStack thường được dùng để các nhà phát triển thử nghiệm hoặc xây dựng các dịch vụ với lượng người dùng không quá lớn.

OpenStack gồm nhiều thành phần, trong đó quan trọng cần có Nova (Compute), Neutron (Networking), Glance (Image Service), tất cả được xác thực và quảng bá vị trí bằng Keystone (Identity). OpenStack thường được tương tác bằng dòng lệnh vì không được dùng bởi người dùng phổ thông, tuy nhiên OpenStack vẫn cung cấp một giao diện web là Horizon (Dashboard) để tương tác với hệ thống.

* **Nova (Compute):**

Quản lý các máy ảo, việc cấp phát, thu hồi tài nguyên CPU, RAM, Storage… hỗ trợ nhiều công nghệ ảo hóa như KVM, Vmware, Xen…

* **Neutron (Networking):**

Neutron cung cấp 2 loại mạng, đó là mạng nội bộ (project network) và mạng bên ngoài (provider networks).

**Mạng bên ngoài (Provider networks).**

Chỉ được cài đặt bởi các quản trị viên. Mạng bên ngoài được cài đặt trên mạng vật lý tại lớp mạng 2 (layer 2 trên mô hình mạng OSI), có khả năng kết nối với mạng vật lý bên ngoài hệ thống cloud.

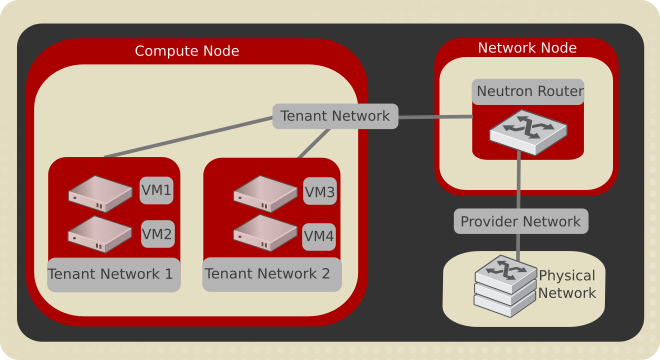
Một máy ảo muốn được nhìn thấy từ bên ngoài cần có địa chỉ IP do mạng bên ngoài cung cấp.

**Mạng nội bộ (Project networks)**

Hay còn gọi là Self-service networks, tức là người dùng trong phạm vi project có thể thoải mái tạo số lượng và địa chỉ mạng tùy ý, cấp phát cho các máy ảo trong project. Việc tạo các mạng này là do người dùng và hoàn toàn không liên quan tới quản trị viên.

Mạng nội bộ sử dụng dải địa chỉ ip nội bộ (private IP address - RFC1918) và có thể kết nối với mạng bên ngoài qua NAT (Network Address Translation) trên một bộ định tuyến (router) ảo, từ đó kết nối tới thế giới bên ngoài.

Vì sử dụng NAT nên thế giới bên ngoài không thể nhìn thấy máy ảo bên trong mạng nội bộ. Do đó, mạng bên ngoài có thể cung cấp một địa chỉ floating ip để bên ngoài có thể giao tiếp với máy ảo đằng sau bộ định tuyến ảo, nhưng số lượng ip này nên hạn chế.



Hình :Mô hình mạng nội bộ và mạng ngoài của OpenStack [11].

Tài nguyên tính toán CPU, Ram, lưu trữ Storage được cung cấp bởi Nova, mạng ảo được cung cấp bởi Neutron là các thành phần chủ yếu để tạo một máy chủ ảo đám mây. Ngoài ra để khởi chạy một máy ảo cần có Image cung cấp bởi Glance và một flavor (quy định lượng CPU, Ram …).

* 1. Mục đích, phạm vi đồ án và các công nghệ sử dụng.
     1. Mục đích của đồ án.

Đồ án được thực hiện nhằm mục đích xây dựng ứng dụng sử dụng mô hình dự đoán PD-GABP và thử nghiệm, đánh giá hiệu quả của mô hình dự đoán PD-GABP trên đám mây OpenStack trong việc dự đoán trước để co giãn tài nguyên. Để làm đươc điều này, đồ án cần xây dựng:

* Một ứng dụng có khả năng:
  + Theo dõi tài nguyên tiêu thụ của máy chủ ảo trên đám mây OpenStack
  + Dự đoán tài nguyên tiêu thụ trong tương lai sử dụng mô hình PD-GABP.
  + Có khả năng co giãn tài nguyên trên đám mây OpenStack.
  + Ghi lại log dữ liệu tài nguyên tiêu thụ được theo dõi, kết quả dự đoán, kết quả co giãn tài nguyên để phục vụ đánh giá, kết luận.
* Xây dựng môi trường thử nghiệm, trên đó mô phỏng requests gửi đến máy chủ ảo để thử nghiệm theo dõi và dự đoán tài nguyên tiêu thụ.

Sau khi xây dựng ứng dụng và môi trường thử nghiệm, đồ án thực hiện thử nghiệm và đưa ra đánh giá độ hiệu quả của mô hình PD-GABP trên đám mây OpenStack.

* + 1. Phạm vi của đồ án.

Đồ án thực hiện dựa trên phần mềm nguồn mở OpenStack để cài đặt đám mây thử nghiệm, vì đây là nền tảng mở nên dễ dàng chỉnh sửa, xây dựng ứng dụng trên đó.

Thử nghiệm và đưa ra đánh giá dựa trên việc chạy ứng dụng và môi trường thử nghiệm đã cài đặt trên OpenStack.

* + 1. Các công nghệ sử dụng.
       1. Các thành phần OpenStack.

Ứng dụng triển khai trên OpenStack Newton với các thành phần cần thiết:

* Identity (Keystone)
* Compute (Nova)
* Image Service (Glance)
* Networking (Neutron)
* Dashboard (Horizon)

Trong đó, ứng dụng cung cấp một giao diện được thêm vào như một phần của Horizon.

Horizon được xây dựng như một phần mềm nguồn mở, viết bằng python, dùng Django web framework và AngularJS, là giao diện để tương tác với hệ thống đám mây OpenStack.

Là phần mềm mã nguồn mở, Horizon thường được sửa theo ý muốn của các nhà phát triển để thêm các giao diện thực hiện các chức năng gắn với OpenStack.

* + - 1. Các thư viện Python.

Các công nghệ:

* Ngôn ngữ: Python 2.7
* Thư viện:
* Falcon: thư viện tối giản, có tốc độ nhanh và dung lượng nhẹ, dùng để xây dựng Restful service bằng python.
* Requests: thư viện để gửi Restful API.
* SqlAlchemy: một ORM dùng để tương tác với CSDL.
* PyJWT: tạo jwt token đảm bảo liên lạc giữa các mô đun của ứng dụng.
* Python-novaclient: giao tiếp với OpenStack nova.
* Python-neutronclient: giao tiếp với OpenStack neutron.
* Django: làm việc với Horizon.
* Mã nguồn core thuật toán PD-GABP được sử dụng trong bài báo *“PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment”*.
  + - 1. Công cụ theo dõi máy ảo.

Các công cụ:

* Influxdb: dùng để lưu dữ liệu thu được từ việc theo dõi máy ảo.
* Cadvisor: một phần mềm do Google phát triển, có khả năng thu thập dữ liệu tài nguyên tiêu thụ như CPU, RAM, dữ liệu vào ra (I/O)… của một máy ảo docker hoặc máy thật, chạy trên docker.

Việc lấy dữ liệu tài nguyên tiêu thụ của máy ảo có thể dùng nhiều phương án khác nhau như sử dụng các công cụ Nagios, Zabbix[1]… nhưng ở đây sử dụng Cadvisor/Influxdb bởi tính dễ cài đặt, dễ sử dụng của nó.

Influxdb là một cơ sở dữ liệu tối ưu hóa cho việc lưu và truy vấn dữ liệu dòng thời gian, cung cấp nhiều tính năng xử lý dữ liệu hữu ích như giảm tần số lưu mẫu dữ liệu (downsampling), hẹn giờ xóa … có tốc độ nhanh và khả năng mở rộng tốt, sử dụng truy vấn giống SQL phù hợp với những nhà phát triển quen làm việc với ngôn ngữ SQL, hỗ trợ Restful để truy vấn dữ liệu.

Cadvisor có khả năng lấy nhiều loại dữ liệu như CPU, RAM, ghi ổ cứng (Disk I/O), lưu lượng mạng (Network) … Tuy hiện tại trong phạm vi đồ án chỉ sử dụng dữ liệu CPU, nhưng với việc hỗ trợ nhiều loại dữ liệu sẽ rất hữu ích để ứng dụng mở rộng trong tương lai.

* + - 1. HA proxy như là một Load Balancer.

Dùng HA proxy có nhược điểm không hỗ trợ thêm, bớt địa chỉ máy chủ theo thời gian thực. Để thêm bớt buộc phải thay đổi file haproxy.conf và khởi động lại HA proxy, dẫn đến các request đang chờ bởi HA proxy có thể bị hủy (drop), dẫn đến trải nghiệm không mong muốn của ứng dụng. Theo tìm hiểu Nginx Plus có hỗ trợ tính năng thêm bớt địa chỉ máy chủ theo thời gian thực qua Restful API, nhưng bản này có tính phí. Trong khuôn khổ đồ án, tôi sử dụng HA proxy và tạm bỏ qua nhược điểm này.

* + - 1. Cài đặt các gói cần thiết khi tạo mới máy chủ ảo.

Sử dụng cloud-init.

Cloud-init là một gói phần mềm (package) có chức năng chạy một đoạn mã (script) khi máy ảo đám mây lần đầu tiên được khởi động. Cloud-init được cài sẵn trong image của máy ảo đám mây.

Đoạn mã cloud-init phổ biến nhất là dạng shell bash script của hệ điều hành linux.

Trong OpenStack, tập tin chứa mã cloud-init được gọi là user-data script.

* + - 1. Cơ sở dữ liệu sử dụng.

Sử dụng SQLite để lưu trữ dữ liệu. Do sử dụng ORM là SQLAlchemy, cơ sở dữ liệu có thể thay thế dễ dàng bằng các hệ quản trị cơ sở dữ liệu khác chỉ với vài dòng code.

Sử dụng Influxdb để lưu trữ các dữ liệu thời gian thực như dữ liệu tài nguyên tiêu thụ được theo dõi, dữ liệu tài nguyên tiêu thụ được dự đoán, thời điểm tạo thêm và hủy bớt server ảo …

* 1. Tóm tắt chương.

Đồ án đặt ra vấn đề tối ưu tài nguyên cấp phát sao cho sát với lượng tài nguyên tiêu thụ, từ đó đưa ra bài toán tự động co giãn tài nguyên nhằm mục đích tránh gây lãng phí tài nguyên trên đám mây. Hiện nay có 3 phương pháp tự động co giãn tài nguyên đó là Dựa trên chu kỳ, Dựa trên ngưỡng và Dựa trên dự đoán. Theo khảo sát trên các nhà cung cấp dịch vụ đám mây lớn như Amazon, Microsoft Azure đều cung cấp dịch vụ co giãn tài nguyên tự động nhưng chỉ hỗ trợ 2 phương pháp đó là Dựa trên chu kỳ và Dựa trên ngưỡng. Với phần mềm nguồn mở OpenStack, người dùng có thể triển khai phương pháp Dựa trên ngưỡng bằng cách dùng kết hợp Heat/Ceilometer.

Sở dĩ phương pháp Dựa trên dự đoán ít được áp dụng trong thực tế bởi phần lớn bài toán co giãn tài nguyên trên thực tế đều có thể được đáp ứng bằng 2 phương pháp Dựa trên chu kỳ và Dựa trên ngưỡng, tuy nhiên hiệu quả không được coi là tối ưu do phương pháp Dựa trên ngưỡng luôn phải quyết định giãn tài nguyên khi tài nguyên sử dụng mới chỉ ở mức 80-90% để đảm bảo tài nguyên mới (máy ảo đám mây) sẵn sàng hoạt động trước khi tài nguyên bị sử dụng hết 100%.

Phương pháp Dựa trên dự đoán vẫn tồn tại rủi ro cao bởi nhược điểm của mọi phương pháp dự đoán là không thể dự đoán chính xác 100% tài nguyên tiêu thụ trong tương lai, phụ thuộc rất nhiều vào thuật toán dự đoán. Nhưng nếu có một phương pháp co giãn Dựa trên dự đoán có độ hiệu quả tốt mà có thể áp dụng được trong thực tế thì đó là một đóng góp rất lớn cho công nghệ điện toán đám mây hiện nay.

Do đó, đồ án này sẽ xây dựng một ứng dụng và thử nghiệm một phương pháp Dựa trên dự đoán, đó là mô hình dự đoán PD-GABP. Ứng dụng và thử nghiệm được triển khai trên đám mây OpenStack để đánh giá hiệu quả của mô hình PD-GABP trong môi trường thử nghiệm cũng như khả năng áp dụng trong thực tế.

CHƯƠNG 2: XÂY DỰNG ỨNG DỤNG, TRIỂN KHAI MÔI TRƯỜNG THỬ NGHIỆM VÀ THỰC HIỆN THỬ NGHIỆM

2.1. Xây dựng ứng dụng dự đoán co giãn tài nguyên.

2.1.1. Phân tích chức năng.

Dựa theo yêu cầu bài toán co giãn tài nguyên, ứng dụng có các chức năng:

* Định nghĩa các cài đặt cho Nhóm:
* Cấu hình về tạo thêm tài nguyên: image, flavor, nics id, floating ip pool, user\_data script.
* Cấu hình cho mô đun dự đoán: các tham số đầu vào của mô đun dự đoán.
* Periodict Number: số chu kỳ trong quá khứ (tham số )
* Data Length: độ lớn chuỗi dữ liệu cần để huấn luyện mạng nơ ron, đơn vị phút.
* Recent Point: tham số .
* Update In Period: thời gian cập nhật lại mạng, đơn vị phút, là .
* Max Scale VM: số máy chủ ảo tối đa tạo thêm khi giãn tài nguyên.
* Các máy chủ ảo của Nhóm. Các máy chủ này đã được cài sẵn Cadvisor/Influxdb.
* Tạo và chạy Nhóm
* Xóa và dừng Nhóm.
* Theo dõi và khởi động lại máy chủ ảo ảo nếu bị treo.
* Co giãn tài nguyên.
* Log dữ liệu tài nguyên tiêu thụ được theo dõi, dự đoán, thời điểm tạo thêm và hủy máy ảo.



Hình : Use Case tổng quan.

Các tác nhân:

* Người dùng có tài khoản đám mây.
* Đám mây OpenStack.
* Cân bằng tải.

2.1.2. Kiến trúc phần mềm.



Hình : Kiến trúc hệ thống.

* **Giao diện Bảng điều khiển (Dashboard):**

Lập trình như một phần của Horizon giúp người dùng tương tác và thực hiện các thao tác như Tạo Nhóm, Xóa Nhóm, Xem Log Của Nhóm.

* **Restful API:**

Sử dụng thư viện Falcon. Giao diện tương tác với bảng điều khiển, nhận các yêu cầu của người dùng.

Người dùng có tài khoản đám mây sẽ xác thực với OpenStack qua Horizon, sau đó giao tiếp với hệ thống bằng bảng điều khiển qua Restful API.

* **Manager:**
  + Nhận yêu cầu từ Restful API mô đun, xử lý các thao tác Tạo Nhóm, Xóa Nhóm, Lấy Log của Nhóm.
  + Quản lý các Group Controller.
* **Group Controller**
  + Mỗi Nhóm được tạo được gắn với một Group Controller.
  + Quản lý việc lưu cấu hình Nhóm (cấu hình tạo thêm tài nguyên, cấu hình mô đun dự đoán …) trong CSDL SQL.
  + Quản lý log dữ liệu trong CSDL Influxdb.
  + Quản lý log sự kiện (sự kiện Tạo Nhóm, Hủy Nhóm, Tạo thêm máy chủ ảo …) trong File Log.
  + Với mỗi Group Controller được tạo thì một Forecast Controller, Monitor Controller, Scale Controller được tạo tương ứng:
    - Quản lý theo dõi tài nguyên qua Monitor Controller.
    - Quản lý dự đoán tài nguyên tiêu thụ qua Forecast Controller.
    - Quản lý co giãn tài nguyên qua Scale Controller.
  + Kịch bản:
    - (1) Khi người dùng yêu cầu tạo Nhóm:
      * Ghi cấu hình vào CSDL SQL.
      * Thêm địa chỉ của các máy chủ ảo trong nhóm vào cấu hình của Cân bằng tải.
      * Sử dụng mô đun Monitor Controller lấy dữ liệu tài nguyên tiêu thụ trong quá khứ của một trong các server ảo trong Nhóm, ghi vào CSDL Influxdb. Nếu độ dài dữ liệu thu được lớn hơn tham số Data Length, chuyển qua bước huấn luyện model. Nếu chưa đủ dữ liệu, tiếp tục lấy dữ liệu qua mỗi vòng lặp, ghi vào CSDL Influxdb cho đến khi đủ dữ liệu.
    - (2) Huấn luyện model: Lấy dữ liệu từ CSDL Influxdb đã ghi lại ở pha (1), huấn luyện mô hình PD-GABP bằng mô đun Forecast Controller. Chuyển qua pha (3) ở vòng lặp tiếp theo.
    - (3) Tại mỗi vòng lặp, sử dụng mô đun Monitor Controller lấy dữ liệu mới nhất, sau đó dự đoán tài nguyên tiêu thụ trong tương lai sử dụng mô đun Forecast Controller. Giá trị dự đoán được sẽ là đầu vào của mô đun Scale Controller, mô đun này sẽ quyết định có co giãn tài nguyên hay không dựa vào giá trị dự đoán đầu vào.
    - (4) Tại mỗi vòng lặp, đếm ngược thời gian cập nhật lại mô hình PD-GABP. Khi đếm ngược đến thời điểm cập nhật lại mô hình, tiến hành lấy dữ liệu huấn luyện từ mô đun Monitor Controller và yêu cầu mô đun Forecast Controller cập nhật lại mô hình theo dữ liệu mới.
    - (5) Khi người dùng yêu cầu xóa Nhóm:
      * Xóa cấu hình trong CSDL SQL.
      * Xóa địa chỉ các máy chủ ảo của nhóm trong Cân bằng tải.
* **Forecast Controller**
  + Chạy mô hình PD-GABP.
* **Monitor Controller**:
  + Sử dụng Restful API gọi tới máy chủ ảo được theo dõi để lấy toàn bộ dữ liệu tài nguyên tiêu thụ được theo dõi bởi Cadvisor trong quá khứ, trên máy chủ ảo này có cài sẵn Influxdb/Cadvisor.
  + Sử dụng Restful API gọi tới CSDL Influxdb lấy dữ liệu tài nguyên tiêu thụ mới nhất.
  + Theo dõi và khởi động lại máy chủ ảo nếu máy chủ ảo bị treo.
* **Scale Controller:**
  + Kết nối tới OpenStack yêu cầu tạo thêm/hủy bớt máy chủ ảo.
  + Kết nối tới Cân bằng tải thêm/bớt địa chỉ máy chủ ảo.

2.1.3. Biểu đồ hành động các luồng thực hiện quan trọng.

* **Tạo Nhóm.**



Hình : Biểu đồ hành động tạo một nhóm.

* + Người dùng định nghĩa trên giao diện Bảng điều khiển các cấu hình của Nhóm (cấu hình mô hình PD-GABP, cấu hình co giãn tài nguyên, máy chủ ảo trong nhóm). Các tham số được đẩy tới mô đun Restful API.
  + Mô đun Restful API gọi tới Manager là mô đun quản lý việc tạo và hủy Nhóm.
  + Lúc này hệ thống dựa vào các tham số, khởi tạo một Group Controller chứa các tham số đó và ghi vào CSDL SQL. Group Controller chọn ra một máy chủ ảo trong nhóm để theo dõi, tạo một Monitor Controller để lấy dữ liệu theo dõi từ máy chủ ảo này, ghi dữ liệu lấy được vào CSDL Influxdb. Có 2 trường hợp xảy ra:
    - Nếu đủ dữ liệu để huấn luyện, tạo Forecast Controller để huấn luyện mô hình PD-GABP, tạo Scale Controller để quản lý việc co giãn tài nguyên.
    - Nếu chưa đủ dữ liệu, tiếp tục đợi cho đến khi đủ dữ liệu.
* **Xóa Nhóm.**



Hình : Biểu đồ hành động xóa một nhóm.

* + Người dùng gửi yêu cầu xóa nhóm tới mô đun Restful API.
  + Restful API chuyển tiếp yêu cầu tới mô đun Manager.
  + Manager lấy Group Controller tương ứng với nhóm người dùng yêu cầu xóa. Lúc này Group Controller xóa những máy chủ ảo được tạo do co giãn tài nguyên và xóa dữ liệu của nhóm trong CSDL SQL, xóa mô đun Monitor Controller, Forecast Controller, Scale Controller liên quan.

**- Cập nhật model.**



Hình : Biểu đồ hành động cập nhật model.

* + Mô đun Group Controller yêu cầu mô đun Monitor Controller liên quan lấy dữ liệu để huấn luyện.
  + Mô đun Monitor Controller kiểm tra trong CSDL Influxdb trước, nếu có dữ liệu thì lấy ở đây, nếu không thì lấy dữ liệu trực tiếp từ Influxdb cài đặt sẵn trên máy chủ ảo qua Restful API.
  + Khi đã có dữ liệu để huấn luyện, yêu cầu Forecast Controller cập nhật lại mô hình.

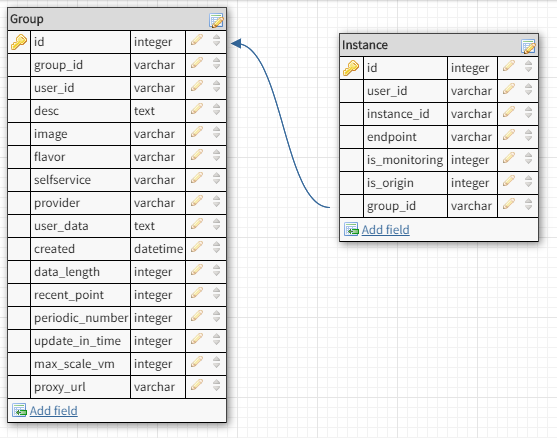
2.1.4. Cấu trúc dữ liệu.

Bảng Group định nghĩa Nhóm và bảng Instance định nghĩa các máy chủ ảo trong của Nhóm được lưu trong CSDL SQL.

Bảng Group chứa các thông tin cần thiết để huấn luyện mạng và các thông tin co giãn tài nguyên.

Bảng Instance chứa các thông tin:

* endpoint: địa chỉ (ip) máy chủ ảo.
* is\_monitoring: bằng 0 nếu máy chủ ảo không được theo dõi tài nguyên tiêu thụ.
* is\_origin: bằng 0 nếu máy chủ ảo được tạo bởi mô đun Scale Controller. Những máy chủ ảo có thuộc tính is\_origin bằng 0 sẽ bị xóa khi người dùng xóa Nhóm.



Hình : Cơ sở dữ liệu SQL.

Dữ liệu tài nguyên tiêu thụ được lấy bởi Monitor Controller và thời điểm tạo, hủy máy ảo được log trong CSDL Influxdb.

Influxdb có thể bao gồm nhiều bảng, mỗi bảng gồm nhiều độ đo (measurement) trong đó được gán nhãn (tags).

Mỗi dòng dữ liệu có cấu trúc gồm:

* measurement
* tags
* value
* time

Hệ thống sao lưu với các nhãn (tags) là *‘real’, ‘predict’, ‘scale\_up’, ‘scale\_down’*.

* *‘real’*: dữ liệu thật thu thập được tại thời điểm quá khứ và hiện tại.
* *‘predict’*: dữ liệu dự đoán tại thời điểm tương lai.
* *‘scale\_up’:* thời điểm tạo thêm máy ảo.
* *‘scale\_down’:* thời điểm xóa bớt máy ảo.

2.1.5. Các API hệ thống cung cấp.

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Method | API | Mô tả |
| 1 | GET | /v1/users/{user\_id}/groups | Lấy danh sách Nhóm |
| 2 | POST | /v1/users/{user\_id}/groups | Tạo Nhóm |
| 3 | GET | /v1/users/{user\_id}/groups/{id} | Lấy thông tin Nhóm theo id |
| 4 | DELETE | /v1/users/{user\_id}/groups/{id} | Xóa Nhóm theo id |
| 6 | GET | /v1/users/{user\_id}/groups/{id}/log | Lấy log của Nhóm |

Bảng : API hệ thống cung cấp.

2.1.6. Phương pháp theo dõi máy ảo.

2.1.6.1. Lấy dữ liệu tài nguyên tiêu thụ theo thời gian thực.

**a. Yêu cầu 1: Lấy dữ liệu thời gian thực.**

Như đã nói, ứng dụng sử dụng các công cụ Influxdb/Cadvisor để lấy dữ liệu.

Influxdb với ưu điểm nhanh và dễ dùng, sử dụng Restful API và truy vấn SQL quen thuộc với lập trình viên, đặc biệt thích hợp lưu dữ liệu theo dòng thời gian.

Cadvisor là ứng dụng được Google phát triển, có thể lấy dữ liệu tài nguyên tiêu thụ theo thời gian thực, hỗ trợ backend là Influxdb.

Cadvisor hỗ trợ lấy dữ liệu theo thời gian thực trực tiếp qua Restful API:

* *GET /api/v1.0/containers/*
* *GET /api/v1.1/subcontainers/*

Nhưng qua thực nghiệm, tôi nhận thấy cách lấy trực tiếp dữ liệu như trên là không phù hợp, vì với 200 request, có khoảng 125 MB dữ liệu truyền trên mạng. Hơn nữa nếu sử dụng phương án này để lấy dữ liệu, dữ liệu thời gian thực cần lưu trữ tập trung tại một cơ sở dữ liệu lớn nằm bên ngoài máy ảo, và cần lấy dữ liệu theo thời gian thực, gây tốn băng thông mạng không đáng có.

Tôi chọn phương án thứ 2, dùng influxdb làm cơ sở dữ liệu cho cadvisor, cả 2 nằm trên máy ảo. Dữ liệu chỉ được lấy từ influxdb qua restful api mỗi khi huấn luyện mạng (lần đầu và mỗi lần update) cũng như mỗi vòng lặp 1 lần.

**b. Yêu cầu 2: Lấy dữ liệu không làm ảnh hưởng tới hiệu năng của máy ảo.**

Theo khảo sát, Cadvisor trong quá trình theo dõi máy ảo có sử dụng sao lưu và ghi dữ liệu vào Influxdb khoảng 2 giây 1 lần, nhưng số giá trị với mỗi độ đo là khoảng 42 giá trị/phút. Một chu kỳ phổ biến là 1 ngày tức cần 24\*60\*42 = 60480 giá trị, với số chu kỳ dùng để huấn luyện là m, mỗi lần truy vấn dữ liệu cần lấy giá trị và chuyển thành dạng json để truyền qua Restful, là một khối lượng xử lý lớn cho CPU và ảnh hướng không tốt tới hiệu năng của máy ảo và trải nghiệm người dùng.

Do đó, ứng dụng sử dụng phương án lấy dữ liệu theo lô, với độ rộng (batch\_size) được chọn khoảng 190 phút.

Một vấn đề gặp phải khi lấy dữ liệu theo lô, theo thực nghiệm có xảy ra trường hợp lấy dữ liệu lô thì kết quả trả về là rỗng, mặc dù thực tế có dữ liệu trong Influxdb, nếu sử dụng độ rộng lớn hơn thì có kết quả trả về. Hiện nay Influxdb chưa có tài liệu đề cập đến về vấn đề này, nhưng vấn đề được giải quyết bằng cách tăng dần độ rộng theo mỗi lần truy vấn dữ liệu.

**Thuật toán lấy dữ liệu influxdb theo lô:**

***input****: là độ dài dữ liệu cần lấy, là thời gian hiện tại, là độ rộng (batch\_size).*

***output****: Dữ liệu cần lấy.*

***while*** *dữ liệu chưa đủ:*

***if*** *, thoát vòng lặp.*

*Lấy dữ liệu lô , kết quả trả về .*

***if******then*** *ghi lại và tiếp tục vòng lặp*

***if*** *độ dài rl < b* ***then***

*Ghi lại rl*

*Thoát khỏi vòng lăp.*

***else***

*Tiếp tục vòng lặp*

Thuật toán trên được cài đặt trong lớp Influxdb Driver như một hàm lấy dữ liệu. Với yêu cầu lấy được đoạn dữ liệu liên tục mới nhất từ thời điểm hiện tại trở về trước để huấn luyện, Đầu vào là độ dài dữ liệu, thời gian hiện tại, độ rộng lấy mỗi vòng lặp (batch\_size) và trả về dữ liệu liên tục lấy từ influxdb. Thuật toán có khả năng phát hiện đoạn dữ liệu không liên tục vì nếu độ dài dữ liệu thu được nhỏ hơn độ rộng mong muốn (batch\_size) thì thoát khoải vòng lặp.

2.1.6.2. Truy vấn lấy dữ liệu bằng Influxdb.

Như đã nói ở trên, truy vấn dữ liệu của Influxdb qua Restful API và truy vấn SQL.

Trong Influxdb dữ liệu được tổ chức thành các cơ sở dữ liệu (database), mỗi cơ sở dữ liệu gồm các dòng cấu trúc như sau:

*<measurement>[,<tag-key>=<tag-value>…] <field-key>=<field-value>[,<field2-key>=<field2-value>…] [unix-nano-timestamp]*

Cadvisor đưa dữ liệu monitor vào Influxdb theo cấu trúc trên, trong đó *measurement* gồm:

* *cpu\_usage\_per\_cpu*
* *cpu\_usage\_system*
* *cpu\_usage\_total*
* *cpu\_usage\_user*
* *fs\_limit*
* *fs\_usage*
* *load\_average*
* *memory\_usage*
* *memory\_working\_set*
* *rx\_bytes*
* *rx\_errors*
* *tx\_bytes*
* *tx\_errors*

Các *measurement* phân biệt bằng các *tags* bao gồm:

* *container\_name*
* *machine*

Trong đó, *container\_name* có thể là ‘/’, *‘docker’, ‘docker.cadvisor’, ‘docker.influxdb’*…

Dữ liệu có tag *“container\_name”=”/”* là phản ánh dữ liệu tài nguyên tiêu thụ của cả máy ảo.

Ví dụ:

Dữ liệu CPU được đếm theo số đếm (clock) tích lũy trong một giây của CPU. Ví dụ với giá trị thu được là , để chuyển sang % ta dùng công thức[12]:

Thường lấy , ta dùng truy vấn sau để lấy CPU theo container\_name:

*SELECT derivative(value, 1s)/1000000000 FROM cpu\_usage\_total WHERE time > now() – 1h GROUP BY container\_name fill(null)*

Đây là phần trăm CPU tổng của tất cả các core. Giả sử CPU có n lõi (core), phần trăm CPU thực tế là .

Dữ liệu RAM

*SELECT mean("value") FROM "memory\_usage" WHERE time > now() – 1h GROUP BY time(1m), "container\_name" fill(null)*

Dữ liệu này đơn vị là byte, ta có thể chuyển sang KB, MB, GB tương ứng.

Cách tính các độ đo khác tương tự như tính RAM.

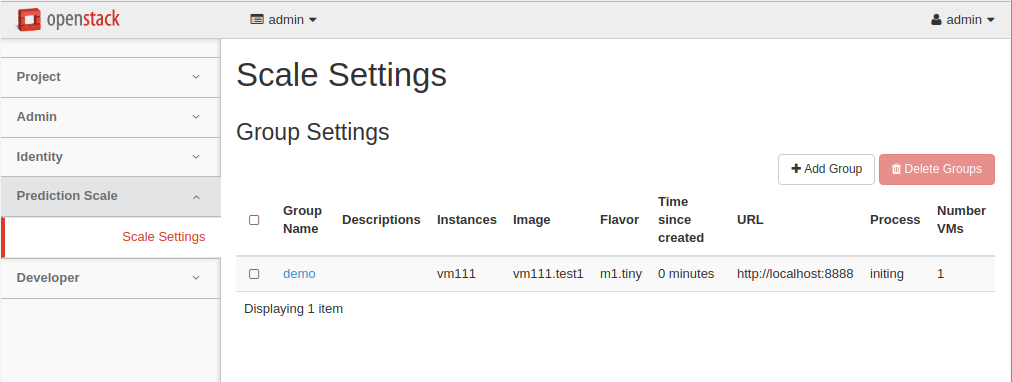
2.1.7. Giao diện chương trình.

Ứng dụng gồm có 2 giao diện chính là:

* Giao diện tạo và xem danh sách các nhóm.
* Giao diện xem log của một nhóm.

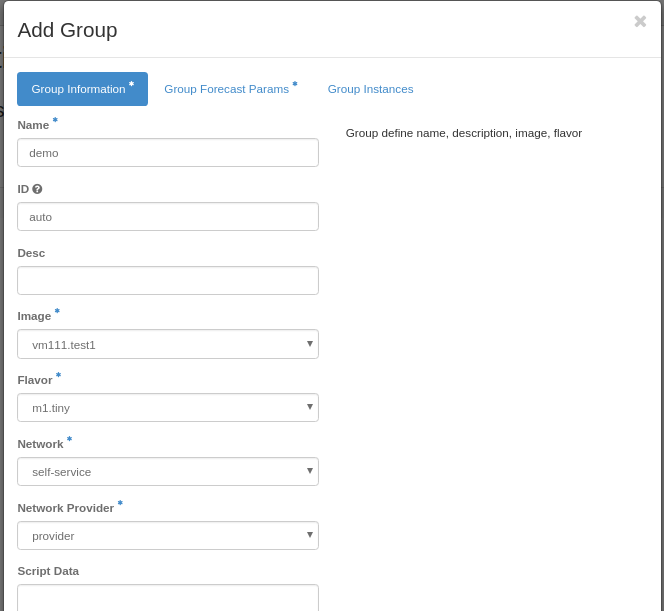
Hình 18 là giao diện danh sách các nhóm thể hiện một bảng danh sách các nhóm hiện có của người dùng. Tại giao diện này có các chức năng Tạo nhóm (Add Group) và Xóa nhóm (Delete Groups).

Tên của nhóm là đường dẫn (link) tới giao diện xem log của nhóm đó (hình 22).

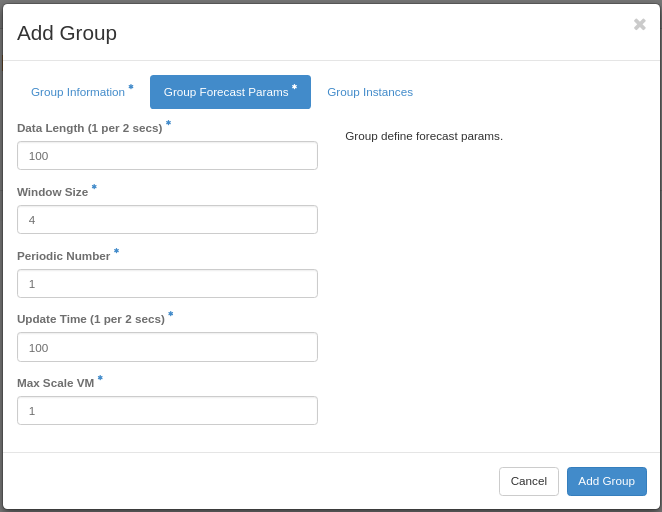


Hình : Giao diện danh sách các nhóm.

Hình 19, 20, 21 là các giao diện tạo nhóm để thiết lập các tham số co giãn tài nguyên, tham số của mô hình PD-GABP, các máy chủ ảo muốn thêm vào nhóm.



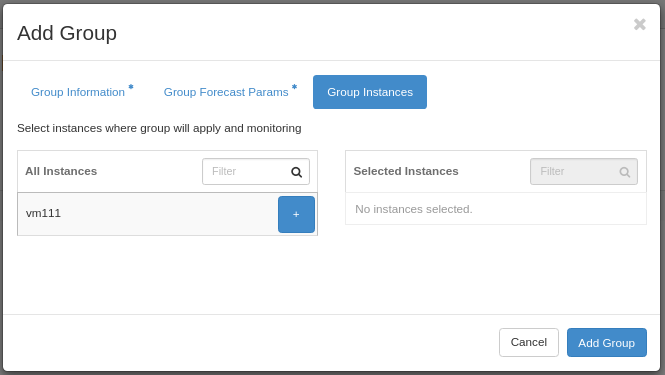
Hình : Giao diện Tạo nhóm: các tham số co giãn tài nguyên.



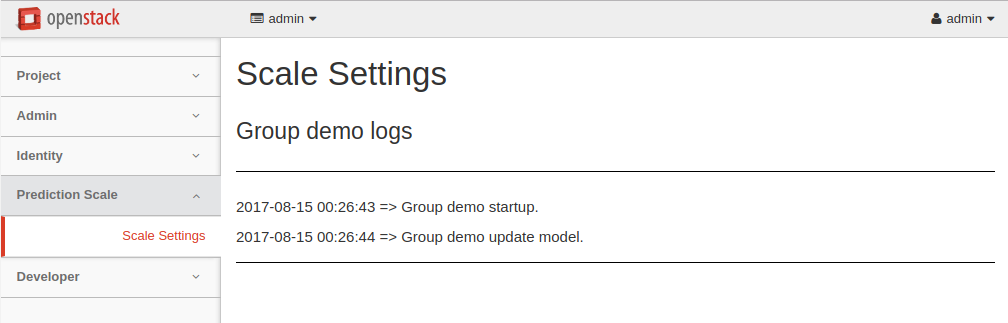
Hình : Giao diện Tạo nhóm: tham số của mô hình dự đoán PD-GABP.

Hình 21 là giao diện thêm danh sách máy chủ ảo cho nhóm. Giao diện này hiển thị các máy chủ ảo tự do và hợp lệ mà có thể thêm vào nhóm.

* Máy chủ ảo tự do: là những máy chủ ảo chưa thuộc nhóm nào.
* Máy chủ ảo hợp lệ: là những máy chủ ảo đang chạy và trên đó được cài sẵn Cadvisor/Influxdb.



Hình : Giao diện Tạo group: thêm danh sách máy chủ ảo cho nhóm.



Hình : Giao diện xem log của nhóm.

2.2. Triển khai môi trường thử nghiệm và thực hiện thử nghiệm.

2.2.1. Mục đích.

Xây dựng một môi trường thử nghiệm triển khai trên đám mây thực tế, cụ thể là OpenStack với máy ảo thật chạy ứng dụng xử lý request gửi đến, theo dõi dữ liệu tài nguyên CPU tiêu thụ và co giãn tài nguyên thật. Với môi trường thử nghiệm này, tôi mong muốn thử nghiệm và đánh giá hiệu quả của phương pháp dự đoán PD GABP trên thực tế với các mô hình:

* Mô hình 4-4: : lấy dữ liệu trung bình và dự đoán 4 phút 1 lần, dự đoán trước 4 phút tương lai.
  + Đánh giá hiệu quả của phương pháp dự đoán PD GABP trong trường hợp dữ liệu tài nguyên tiêu thụ thay đổi đột ngột
* Mô hình 2-2: lấy dữ liệu trung bình và dự đoán 2 phút 1 lần, dự đoán trước 4 phút tương lai.
  + Đánh giá hiệu quả của phương pháp dự đoán PD GABP trong trường hợp dữ liệu tài nguyên tiêu thụ thay đổi đột ngột
  + Đánh giá hiệu quả của phương pháp dự đoán PD GABP trong trường hợp dữ liệu tài nguyên tiêu thụ không có đột biến trong thời gian dài.

Với các thử nghiệm trên, tôi mong muốn đưa ra một kết luận về việc áp dụng phương pháp dự đoán PD GABP trong thực tế, cũng như các điểm lưu ý để áp dụng phương pháp này một cách hiệu quả.

2.2.2. Dữ liệu thử nghiệm.

Đối với một server chạy ứng dụng trên thực tế, khi một request được gửi đến, server xác định yêu cầu và thực hiện một loạt các thao tác để trả lại kết quả. Các thao tác này có thể là tính toán, đọc/ghi dữ liệu (trong CSDL, trong cache) từ Ram, ổ cứng, hay qua mạng… Tất cả các thao tác trên đều tiêu tốn tài nguyên của server.Tùy từng tao tác có thể tiêu thụ tài nguyên bộ nhớ Ram, ổ cứng, mạng nhiều ít khác nhau, nhưng đều tiêu thụ CPU. Các tài nguyên tiêu thụ nhìn chung luôn tăng lên hay giảm đi tỉ lệ thuận với số lượng request mà server cần xử lý.

Bỏ qua các thao tác đọc/ghi dữ liệu, thử nghiệm của tôi chỉ quan tâm tới tài nguyên CPU tiêu thụ khi server xử lý các request. Do đó, tôi sử dụng một server thử nghiệm, trên đó có cài đặt một ứng dụng python đơn giản, sử dụng đầu vào để tính giá trị đầu ra, không có CSDL hay đọc/ghi qua mạng, và cài webserver Gunicorn trên đó để chạy ứng dụng.

Dữ liệu tài nguyên mà ứng dụng python này khiến server tiêu thụ được theo dõi theo thời gian thực bằng công cụ Cadvisor và ghi lại vào CSDL Influxdb. Dữ liệu ghi lại trong Influxdb dưới dạng chuỗi thời gian với tần số lấy mẫu khoảng 47 lần/giây.

Để tạo request từ Jmeter, tôi sử dụng plugin Ultimate Thread Group của Jmeter.

Nói chung, Jmeter hoạt động dựa trên việc tạo các thread, các thread này có thể được coi là những người dùng ảo. Mỗi người dùng này thường được xác định sẵn hoạt động, quy định trong các controller chứa trong nó. Các controller định nghĩa:

- Địa chỉ để gửi request đến.

- Định nghĩa request được gửi.

- Tần suất gửi.

Thời gian sống của mỗi người dùng, thời điểm bắt đầu hoạt động cũng được quy định sẵn. Các quy định này được gọi chung là các kịch bản, và plugin Ultimate Thread Group là một plugin chạy các kịch bản này.

Kịch bản trong Ultimate Thread Group bao gồm các dòng, mỗi dòng lại được định nghĩa bởi 5 tham số:

- Start Threads Count: số lượng thread dòng này tạo ra.

- Initial Delay, sec: thời gian trễ trước khi các thread được tạo.

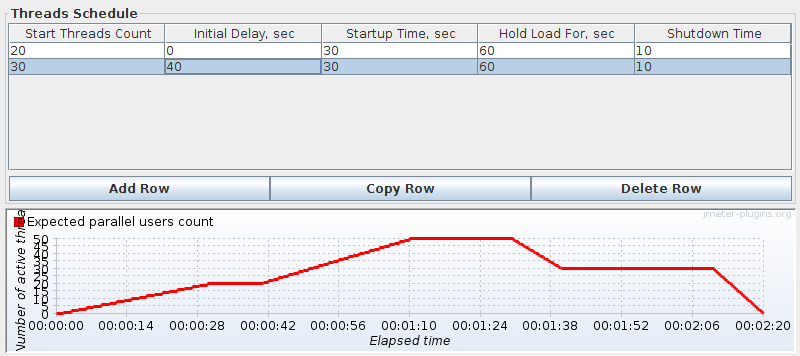
- Startup Time, sec: thời gian số thread của dòng tăng lên từ 0 đến khi đủ Start Threads Count.

- Hold Load For, sec: thời gian các thread hoạt động.

- Shutdown Time, sec: thời gian hủy, số lượng thread của dòng giảm xuống từ Start Threads Count đến 0.

Bằng cách kết hợp các dòng này tôi có thể quy định được số lượng thread hoạt động tại mỗi thời điểm.

Ví dụ:



Hình 23: Ví dụ hoạt động của Ultimate Thread Group.

Trong Ultimate Thread Group tôi định nghĩa 2 dòng như hình trên. Hình dưới là số lượng thread dự kiến theo thời gian, trục hoành là thời gian tính theo giây từ thời điểm khởi chạy kịch bản, trục tung là số lượng thread tương ứng.

Trong thực tế, số lượng request gửi tới một server ứng dụng thường có sự lặp lại (có chu kỳ), cũng như luôn có những ngẫu nhiên nhất định trong mỗi chu kỳ. Giả lập số lượng request với yêu cầu như trên là rất khó, vì cần phải vừa mô phỏng sự lặp lại này, vừa phải đảm bảo số lượng request tạo ra phải khách quan.

Để tạo ra kịch bản ví dụ gồm n chu kỳ, tôi sử dụng một mẫu gồm 4 dòng và tạo ra n chu kỳ đó bằng cách tạo ra 4n dòng dựa trên mẫu này kết hợp với random các tham số.

Mẫu ban đầu:

*Pattern = [*

*{10, 0, 120, 500, 200},*

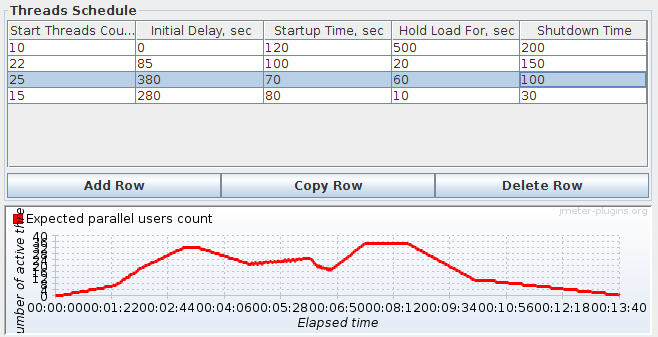
*{22,85,100,20,150},*

*{25,380,70,60,100},*

*{15,280,80,10,30}*

*]*

Biểu diễn mẫu trong Jmeter. Các giá trị trong mảng tương ứng với các tham số (Start Threads Count, Initial Delay, Startup Time, Hold Load For, Shutdown Time).



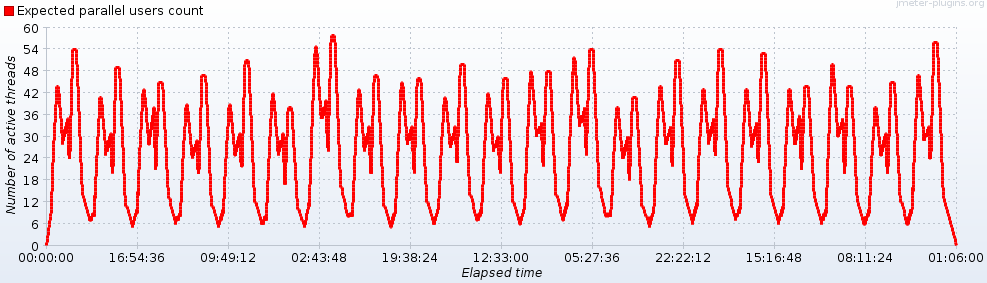
Hình 24: Mẫu Pattern biểu diễn trong Jmeter.

Để tạo ra kịch bản có độ dài chu kỳ bất kỳ, tôi co giãn tuyến tính các tham số (Initial Delay, Startup Time, Hold Load For, Shutdown Time) bằng hệ số periodTime.

Để tạo chu kỳ với số lượng request bất kỳ, tôi co giãn tuyến tính tham số Start Threads Count bằng 4 hệ số nThread0, nThread1, nThread2, nThread3 tương ứng với 4 dòng của mẫu.

Để tạo ra ngẫu nhiên giữa các chu kỳ, với mỗi chu kỳ, tôi tạo ra 4 dòng kịch bản với các hệ số nThread0, nThread1, nThread2, nThread3 được lấy ngẫu nhiên trong một khoảng nhất định.

Với việc kết hợp dùng 1 mẫu có sẵn kết hợp với tính ngẫu nhiên, tôi mong muốn rằng kịch bản mình tạo ra có đủ tính lặp lại lẫn ngẫu nhiên của dữ liệu trong thực tế.

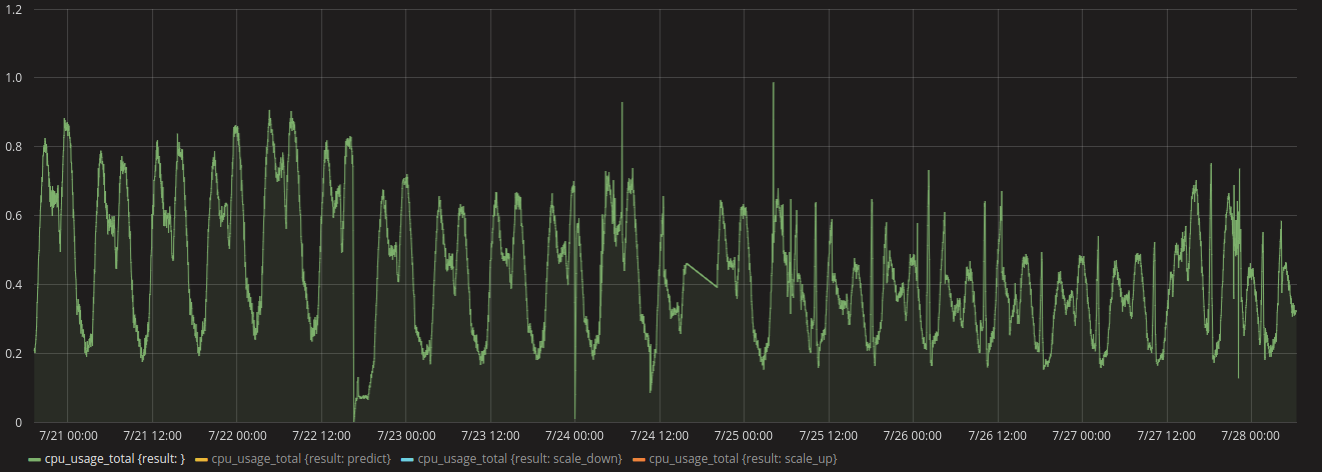


Hình 25: Kịch bản tạo ra với 21 chu kỳ, mỗi chu kỳ dài 8 tiếng. Trục hoành là thời gian, trục tung là số lượng thread còn sống tại mỗi thời điểm.

Toàn bộ công việc sinh kịch bản được thử hiện bằng một chương trình java, tạo ra kịch bản và ghi vào file .jmx. Jmeter sẽ sử dụng kịch bản này (file .jmx) để giả lập người dùng theo số thread quy định trong kịch bản.

Dữ liệu tài nguyên CPU tiêu thụ của server thử nghiệm khi Jmeter gửi request theo thời gian thực được ghi lại bằng công cụ Cadvisor và CSDL Influxdb. Dữ liệu này như đã nói được lấy mẫu khoảng 47 lần mỗi giây. Dữ liệu đưa vào mạng neural để huấn luyện sẽ được lấy mẫu lại khoảng t phút (2, 4 … phút ) 1 lần tùy mô hình sử dụng.

Ví dụ dữ liệu được lấy mẫu 4 phút 1 lần vẽ bằng công cụ Grafana.



Hình 26: Dữ liệu tài nguyên CPU tiêu thụ của server thử nghiệm được ghi lại bằng công cụ Cadvisor và CSDL Influxdb, lấy mẫu 4 phút 1 lần. Trục hoành là thời gian, trục tung là phần trăm CPU tiêu thụ đưa về khoảng 0-1 (1 tương ứng 100%).

Qua biểu đồ dữ liệu tài nguyên CPU tiêu thụ của server , có thể thấy phần trăm CPU tiêu thụ của server thử nghiệm tỉ lệ thuận với số lượng request cần xử lý. Ví dụ tại chu kỳ đầu, để phục vụ 42 thread của Jmeter gửi request đến server sẽ tương tứng tiêu thụ 82% CPU, với 54 thread sẽ cần 87% CPU, nếu số thread chỉ là 6 thì chỉ 20% CPU tiêu thụ.

Ta cũng có thể thấy dữ liệu này vừa có tính chu kỳ (lặp lại) và vẫn đảm bảo tính ngẫu nhiên của dữ liệu tài nguyên tiêu thụ trên thực tế.

2.2.3. Môi trường thử nghiệm.

2.2.3.1. Cài đặt.

- Chạy hệ thống dự đoán và co giãn tài nguyên trên Ubuntu Server 16.04

- Chạy Openstack Newton trên Ubuntu Server 16.04

- Công cụ Apache Jmeter và plugin Ultimate Thread Group để gửi request giả lập người dùng theo thời gian thực.

- Server ảo đám mây thử nghiệm:

o Cloud image: Ubuntu Server 16.04, flavor 1 vCPU, 1GB Ram, 5GB ổ

cứng.

o Server thử nghiệm trên đó được cài web server Gunicorn để xử lý các request được gửi tới từ Jmeter, công vụ Cadvisor và CSDL Influxdb để theo dõi và ghi lại tài nguyên tiêu thụ của server thử nghiệm.

2.2.3.2. Thiết lập chung.

Cho phép tối đa 2 máy ảo đám mây một lúc (1 theo dõi, 1 giãn thêm, máy ảo giãn

thêm không được theo dõi).

Trong khuôn khổ thử nghiệm xoay quan phần trăm CPU tiêu thụ. Thử nghiệm quy

định nếu phần trăm CPU lớn hơn 55% được coi là quá tải (tương đương 100%

trong thực tế).

Thiết lập luật đơn giản để hủy bớt máy ảo khi tài nguyên tiêu thụ thấp: nếu phần

trăm CPU nhỏ hơn 20% thì giảm bớt 1 máy ảo.

Mạng neural được huấn luyện lại mỗi 8 tiếng một lần. Jmeter giả lập số lượng

request theo chu kỳ 8 tiếng.

Phương pháp PDGABP, tham số windowsize = 4, m = 1.

2.2.4. Thử nghiệm.

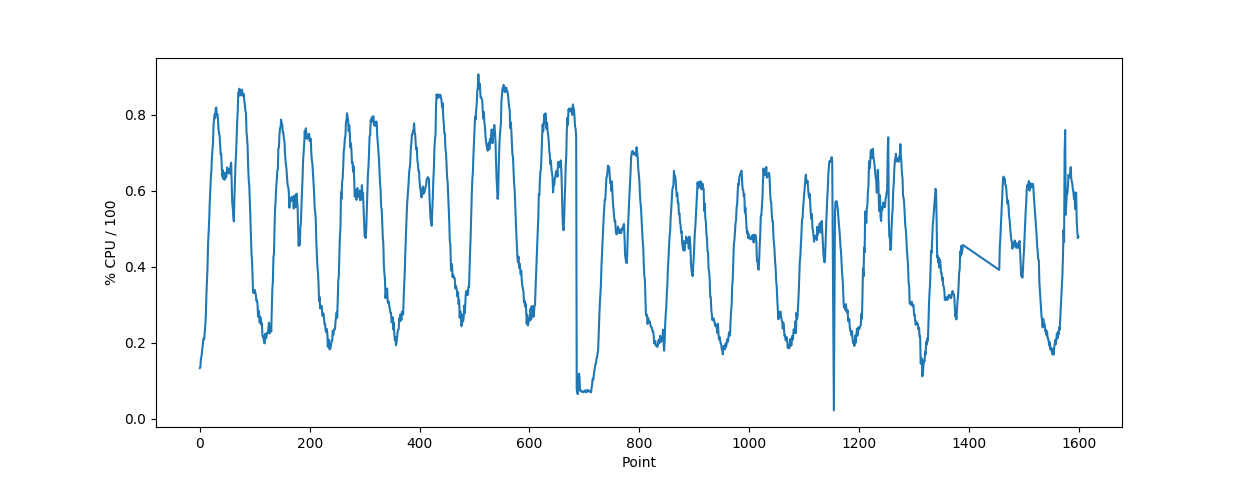
2.2.4.1. Thử nghiệm mô hình 4-4 trong trường hợp dữ liệu thay đổi đột ngột.

***Mô hình 4-4***: Dữ liệu lấy mẫu mỗi 4 phút 1 lần, dự đoán 4 phút tương lai.

Mỗi điểm dữ liệu là phần trăm CPU tiêu thụ trung bình trong 4 phút. Sau mỗi 4 phút, hệ thống lấy dữ liệu mới và sử dụng để dự đoán 4 phút tương lai.

Mục đích của thử nghiệm là đánh giá độ hiệu quả của mô hình 4-4 khi dữ liệu thay đổi đột ngột qua nhiều lần cập nhật lại mạng neural.

Dữ liệu sử dụng để huấn luyện có độ dài khoảng gần 5 ngày.

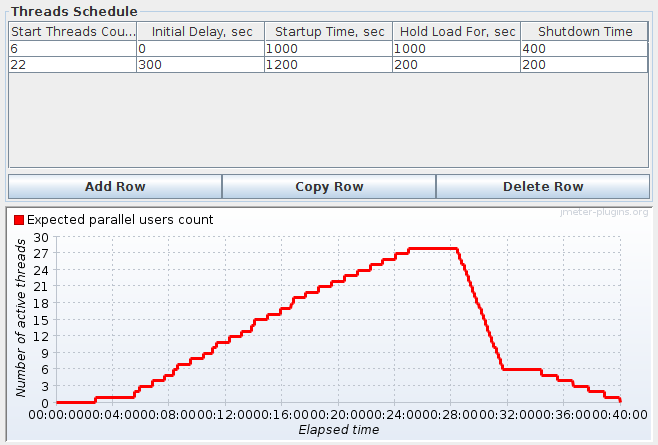


Hình 27: Dữ liệu phần trăm CPU tiêu thụ dùng trong mô hình 4-4, lấy mẫu mỗi 4 phút 1 lần, độ dài khoảng 5 ngày. Trục hoành là điểm dữ liệu, trục tung là phần trăm CPU đưa về khoảng 0-1.

**Kịch bản thử nghiệm:**

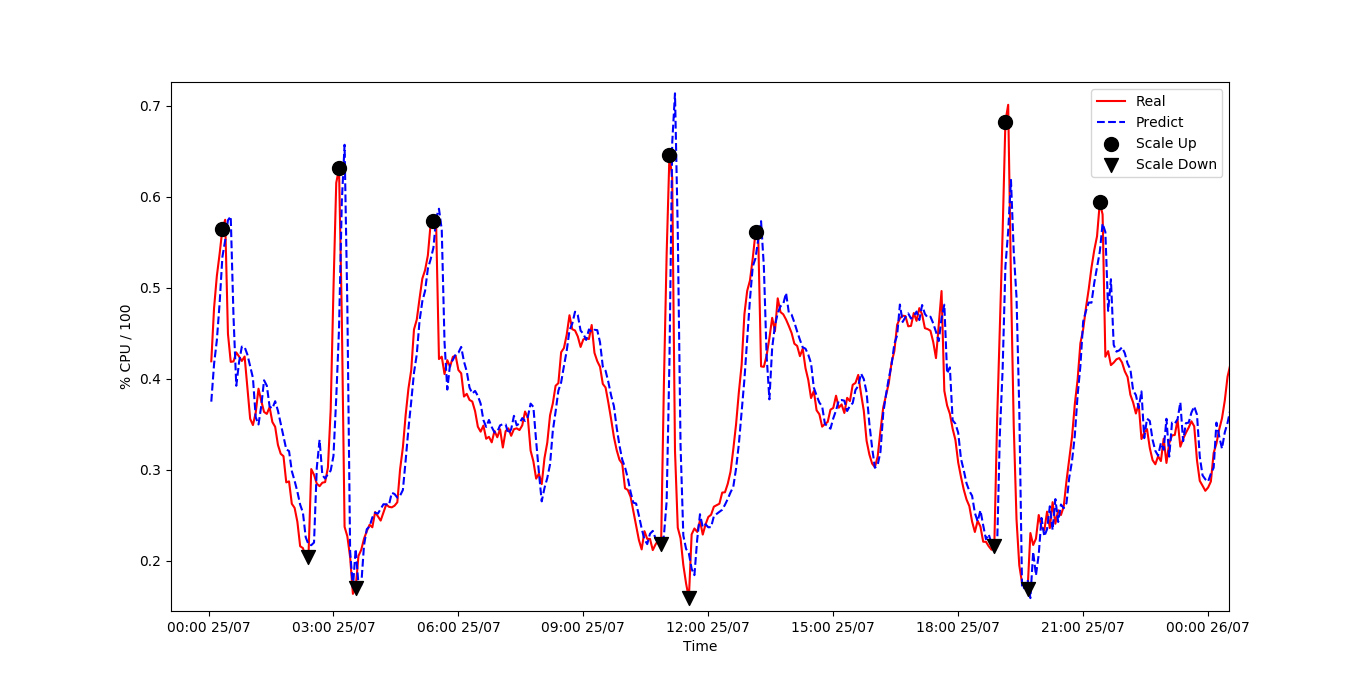
Thử nghiệm trong 24 giờ, 3 lần cập nhật mạng neural (8 tiếng cập nhật 1 lần).

Để tạo được dữ liệu thay đổi đột ngột, ở đây tôi chạy một chương trình Jmeter biệt lập với chương trình Jmeter chạy kịch bản đã có, sử dụng Ultimate Thread Group với tham số như hình dưới để giả lập khoảng tối đa 30 thread gửi request tới server thử nghiệm trong khoảng 40 phút. Việc này được thực hiện trong mỗi 8 tiếng một lần.



Hình 28: Kịch bản thread đột biến gửi tới server thử nghiệm.

**Kết quả thử nghiệm qua 3 lần cập nhật mạng neural**:



*Hình 29: Kết quả thử nghiệm mô hình 4-4.*

*Trục hoành: thời gian.*

*Trục tung: phần trăm CPU tiêu thụ trong khoảng 0-1.*

*Đường màu đỏ: phần trăm CPU tiêu thụ của server thử nghiệm, lấy mẫu 4 phút/lần.*

*Đường màu xanh nét đứt: phần trăm CPU dự đoán bằng phương pháp PD GABP, dự đoán sau trước 4 phút.*

*Hình tròn đen: Giá trị x: thời điểm phát hiện và tạo thêm máy ảo giãn tài nguyên.*

*Giá trị y: giá trị phần trăm CPU tiêu thụ tại thời điểm x.*

*Hình tam giác đen: Giá trị x: thời điểm phát hiện và xóa bớt máy ảo co tài nguyên.*

*Giá trị y: giá trị phần trăm CPU tiêu thụ tại thời điểm x.*

**Cập nhật lần 1:**

Dữ liệu huấn luyện: dữ liệu trình bày ở trên.

**Tìm chu kỳ:**

Sử dụng thuật toán AUTO PERIOD.

Chọn threshold = 0.2. Tần số lấy mẫu fs=360 (360 điểm tương ứng 1 ngày).

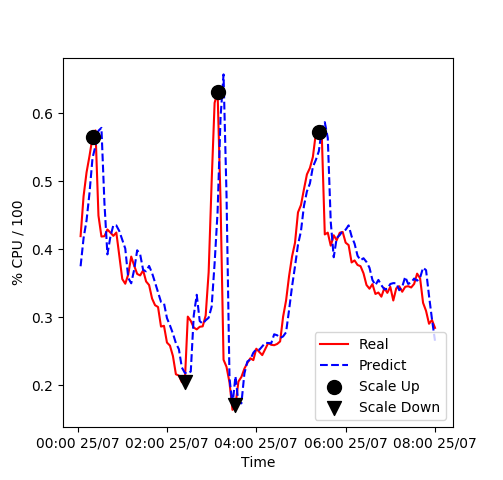
Hình 9 biểu diễn periodogram của dữ liệu huấn luyện. Các điểm tròn màu xanh trên hình 9 là các điểm cực trị có Power > threshold \* max(Power).

Hình 10 biểu diễn hàm tự tương quan ACF của dữ liệu huấn luyện. Các điểm tròn màu xanh trên hình 10 là các chu kỳ tìm được ở hình 9.

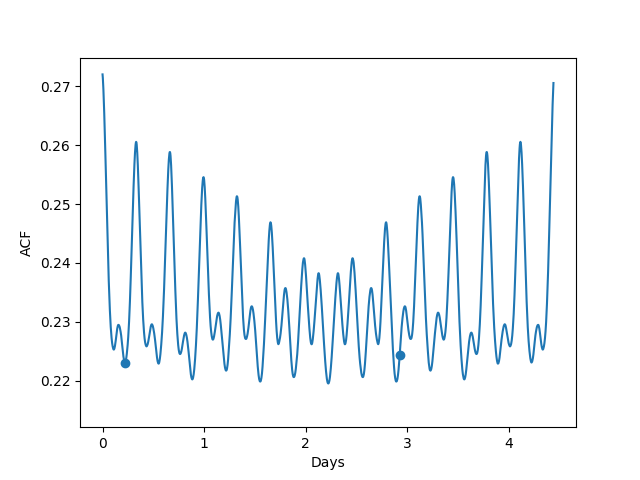
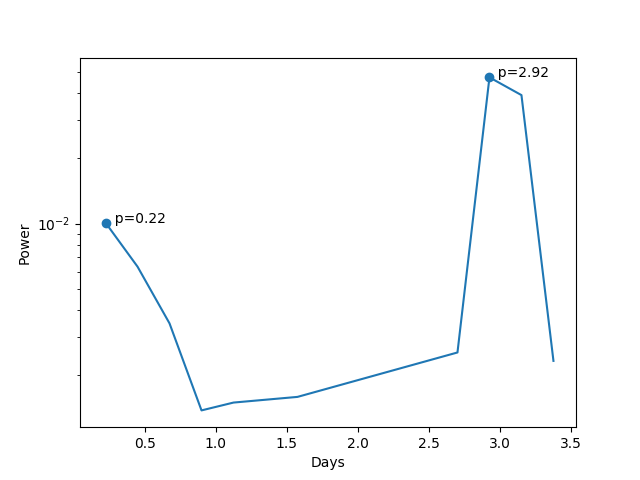
Thuật toán AUTO PERIOD tính periodogram để tìm các chu kỳ có khả năng là chu kỳ đúng, sau đó tính ACF và kiểm tra lại và chọn ra chu kỳ phù hợp nhất.

Sau khi chạy thuật toán AUTO PERIOD, 2 chu kỳ tìm được ở bước 1 đều không phải là chu kỳ phù hợp.

Do đó, dữ liệu huấn luyện là không có chu kỳ.



Hình 30: Kết quả thử nghiệm mô hình 4-4 lần cập nhật 1.



Hình 31: Periodogram của dữ liệu huấn luyện lần cập nhật 1. Các chu kỳ tìm được p=0.22, p = 2.92.

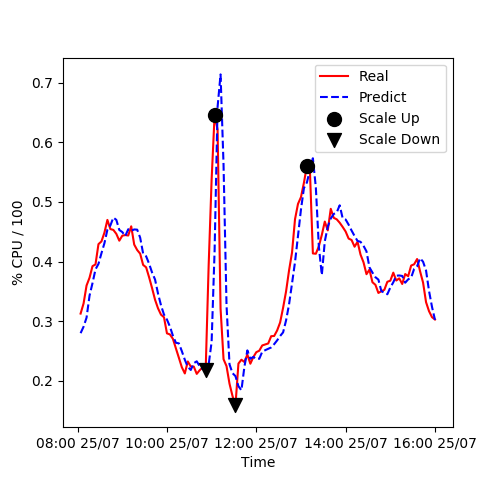
Hình 32: ACF của dữ liệu huấn luyện lần cập nhật 1. Các điểm trên hình là chu kỳ p=0.22 và p=2.92 tìm được trên hình 31.

**Cập nhật lần 2:**

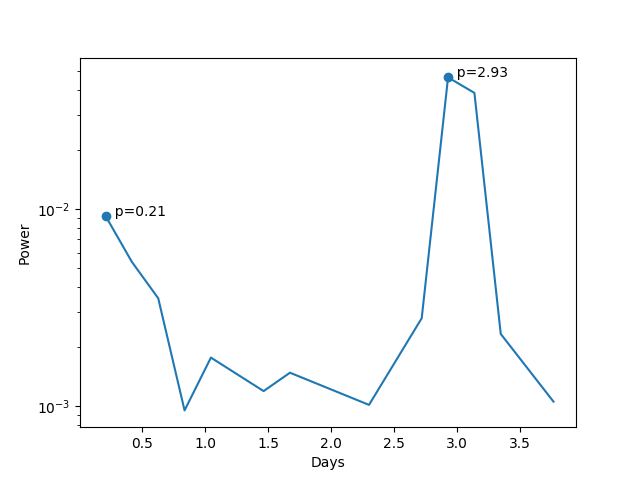
Dữ liệu huấn luyện: Là dữ liệu huấn luyện trình bày ở trên kết hợp với dữ liệu theo dõi được của server thử nghiệm trong khi chạy cập nhật 1.

Tương tự lần cập nhật 1, sau khi chạy thuật toán AUTO PERIOD 2 chu kỳ tìm được từ việc tính periodogram không phải 2 chu kỳ phù hợp.

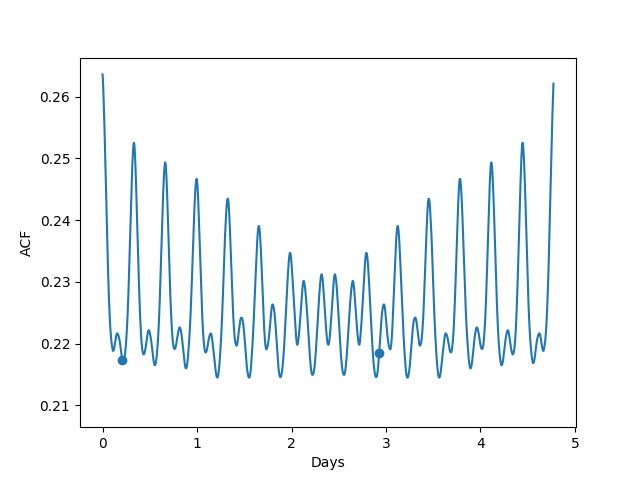
Do đó, dữ liệu huấn luyện là không có chu kỳ.



Hình 33: Kết quả thử nghiệm mô hình 4-4 lần cập nhật 2.



Hình 34: Periodogram của dữ liệu huấn luyện lần cập nhật 2. Các chu kỳ tìm được p=0.21, p = 2.93.

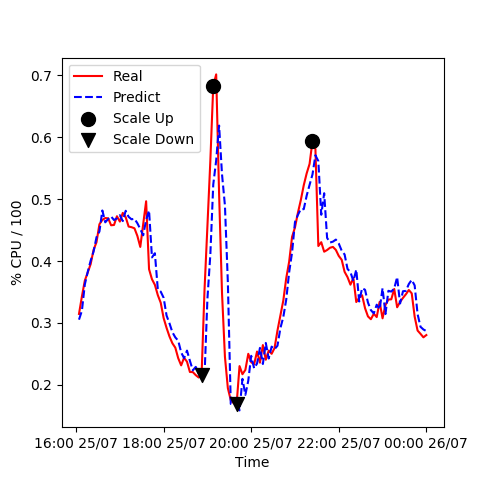


Hình 35: ACF của dữ liệu huấn luyện lần cập nhật 2. Các điểm trên hình là chu kỳ p=0.21 và p=2.93 tìm được trên hình 34.

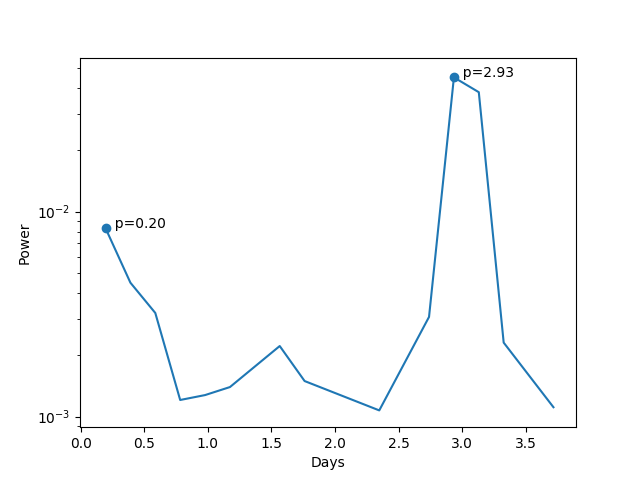
**Cập nhật lần 3:**

Dữ liệu huấn luyện: Là dữ liệu huấn luyện trình bày trên kết hợp với dữ liệu theo dõi được của server thử nghiệm trong khi chạy cập nhật 1 và 2.

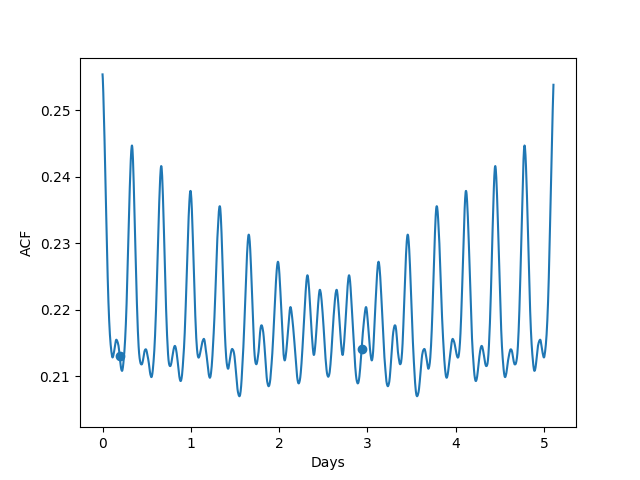
Tương tự, sau khi chạy thuật toán AUTO PERIOD, dữ liệu huấn luyện không có chu kỳ.



Hình 36: Kết quả thử nghiệm mô hình 4-4 lần cập nhật 3.



Hình 37: Periodogram của dữ liệu huấn luyện lần cập nhật 3. Các chu kỳ tìm được p=0.20, p = 2.93



Hình 38: ACF của dữ liệu huấn luyện lần cập nhật 3. Các điểm trên hình là chu kỳ p=0.20 và p=2.93 tìm được trên hình 37.

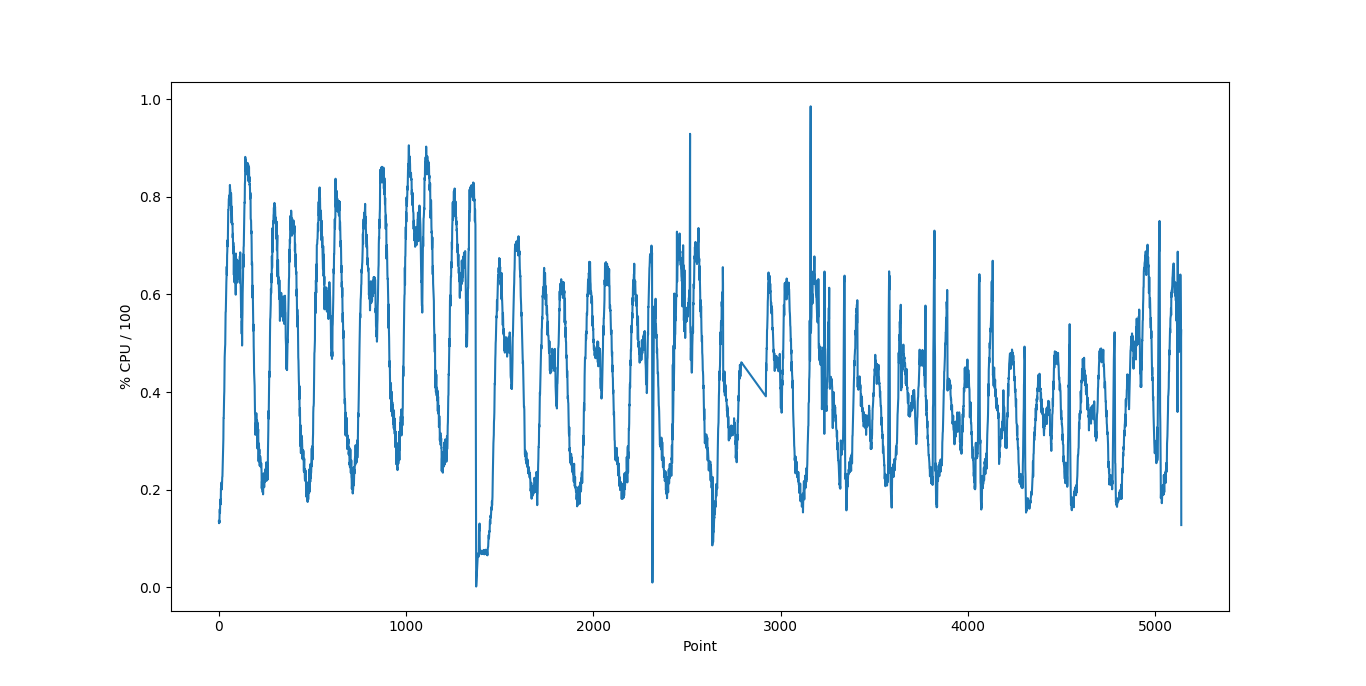
**Nhận xét:**

* Trong trường hợp dữ liệu tài nguyên tiêu thụ thay đổi đột ngột, rất khó tìm ra chu kỳ phù hợp cho dữ liệu, do đó, thuật toán PD GABP trở thành GA BPNN, nên cho độ chính xác chưa cao.
  + Cụ thể, thời điểm tạo thêm máy ảo trong thử nghiệm luôn chậm hơn so với tài nguyên CPU tiêu thụ của máy thử nghiệm (tất cả các giá trị tài nguyên tiêu thụ luôn lớn hơn 55% tại thời điểm phát hiện quá tải và tạo thêm máy ảo), nhất là tại những vùng dữ liệu thay đổi đột ngột. Theo kết quả thử nghiệm, các đỉnh tương ứng với thời gian tạo và gửi thêm request đột biến luôn có phần trăm CPU tiêu thụ khi phát hiện và tạo thêm máy ảo > 60%, còn những đỉnh khác nằm trong khoảng 55% đến 58%.
* Có kết quả như vậy vì dữ liệu dự đoán không bám sát sườn đi lên của dữ liệu tài nguyên tiêu thụ, luôn chậm hơn 1 đến 2 điểm dữ liệu, do đó có dự đoán chậm hơn mong muốn.
* Từ các đặc điểm trên có thể kết luận mô hình 4-4 không phù hợp với trường hợp dự đoán tài nguyên tiêu thụ thay đổi đột ngột.

2.2.4.2. Thử nghiệm mô hình 2-4 trong trường hợp dữ liệu thay đổi đột ngột.

Mỗi điểm dữ liệu là phần trăm CPU tiêu thụ trung bình trong 2 phút. Sau mỗi 2 phút lấy dữ liệu mới và sử dụng để dự đoán 4 phút tương lai.

Dữ liệu huấn luyện có độ dài khoảng 7 ngày.



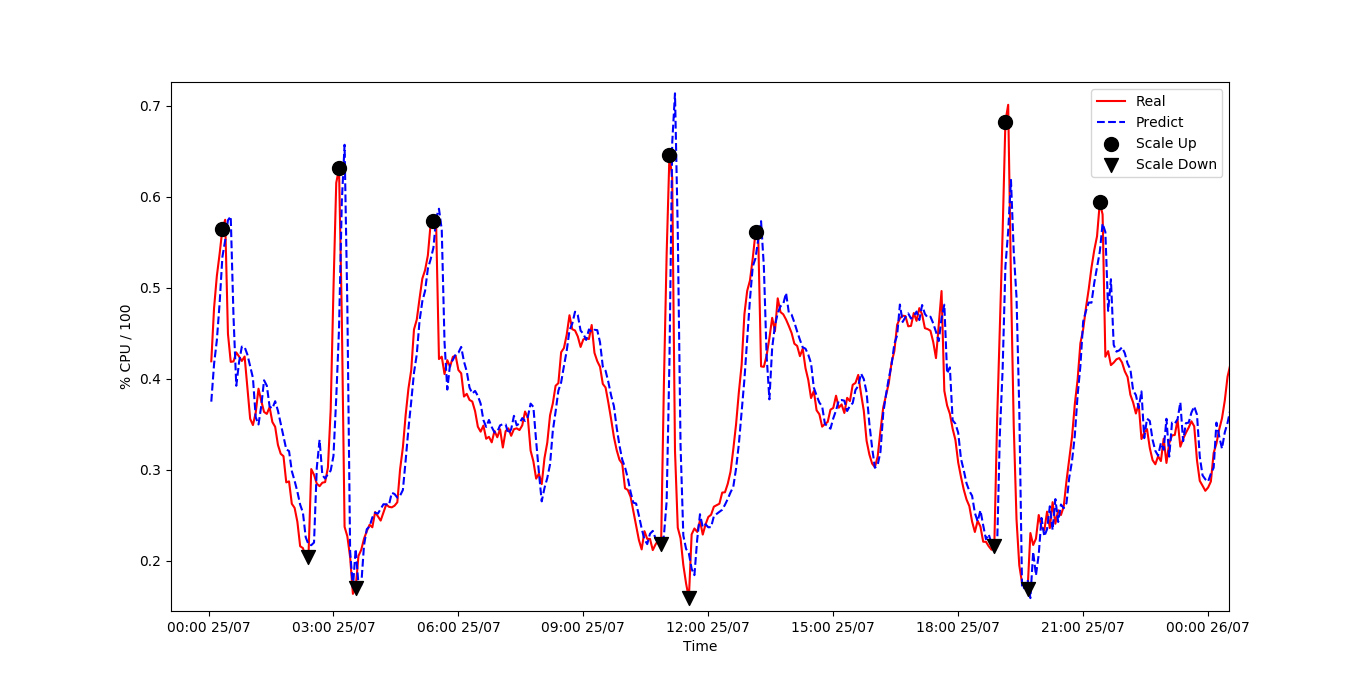
Hình 39: Dữ liệu phần trăm CPU tiêu thụ dùng trong mô hình 2-4, lấy mẫu mỗi 2 phút 1 lần, độ dài khoảng 7 ngày. Trục hoành là điểm dữ liệu, trục tung là phần trăm CPU đưa về khoảng 0-1.

Mục đích và thiết lập giống thử nghiệm trong mô hình 4-4. Chạy 3 lần cập nhật trong 24 tiếng, dữ liệu đầu vào mỗi lần cập nhật được làm tương tự như mô hình 4-4.

Các tham số của thuật toán AUTO PERIOD:

* Tần số lấy mẫu fs=720 (720 điểm tương ứng 1 ngày)
* Threshold=0.2.

Kết quả thử nghiệm.



Hình 40: Kết quả thử nghiệm mô hình 2-4.

*Trục hoành: thời gian.*

*Trục tung: phần trăm CPU tiêu thụ trong khoảng 0-1.*

*Đường màu đỏ: phần trăm CPU tiêu thụ của server thử nghiệm, lấy mẫu 2 phút/lần.*

*Đường màu xanh nét đứt: phần trăm CPU dự đoán bằng phương pháp PD GABP, dự đoán sau trước 4 phút.*

*Hình tròn đen: Giá trị x: thời điểm phát hiện và tạo thêm máy ảo giãn tài nguyên.*

*Giá trị y: giá trị phần trăm CPU tiêu thụ tại thời điểm x.*

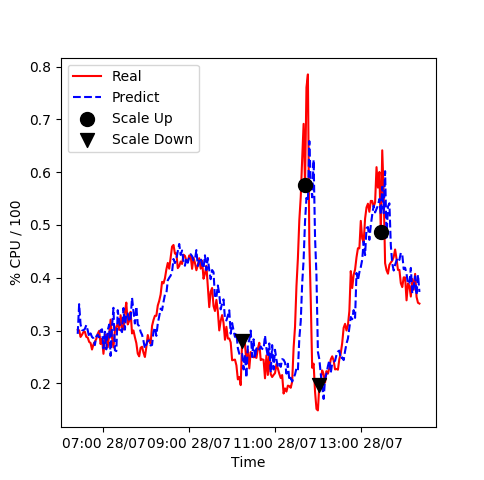
*Hình tam giác đen: Giá trị x: thời điểm phát hiện và xóa bớt máy ảo co tài nguyên.*

*Giá trị y: giá trị phần trăm CPU tiêu thụ tại thời điểm x.*

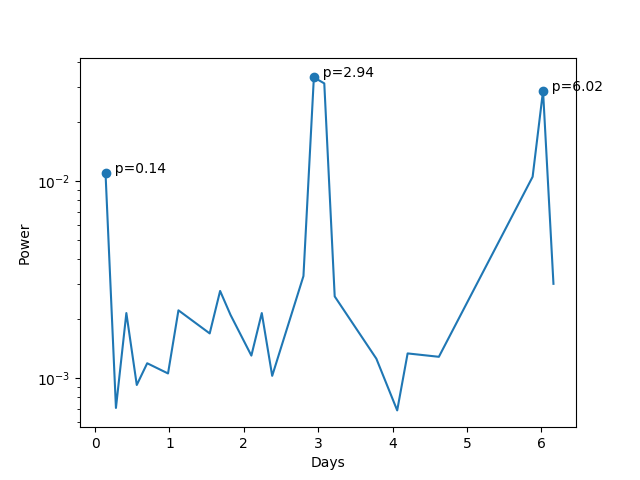
**Cập nhật 1:**

Dữ liệu huấn luyện là dữ liệu đề cập ở hình 10.

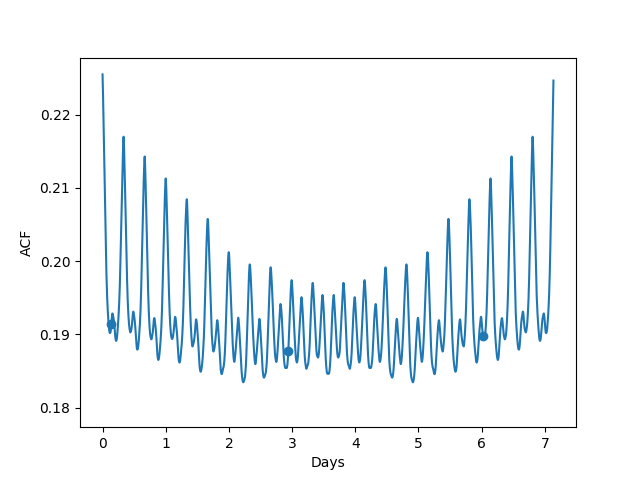
Sau khi chạy thuật toán AUTO PERIOD, các chu kỳ tìm được trên periodogram đều không phù hợp. Dữ liệu không có chu kỳ.

**

Hình 41: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 1.



Hình 42: Periodogram của dữ liệu huấn luyện lần cập nhật 1. Các chu kỳ tìm được p=0.14, p = 2.94, p =6.02

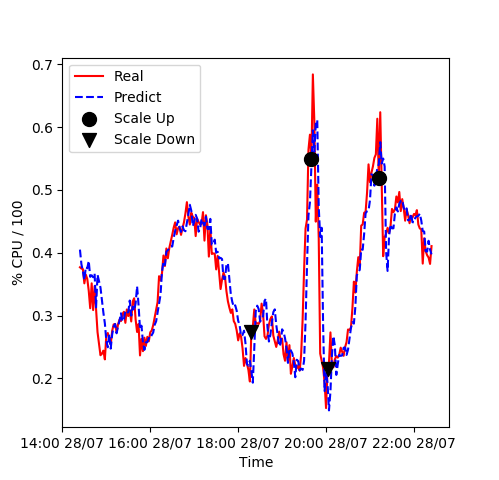


Hình 43: ACF của dữ liệu huấn luyện lần cập nhật 1. Các điểm trên hình là chu kỳ trong hình 42.

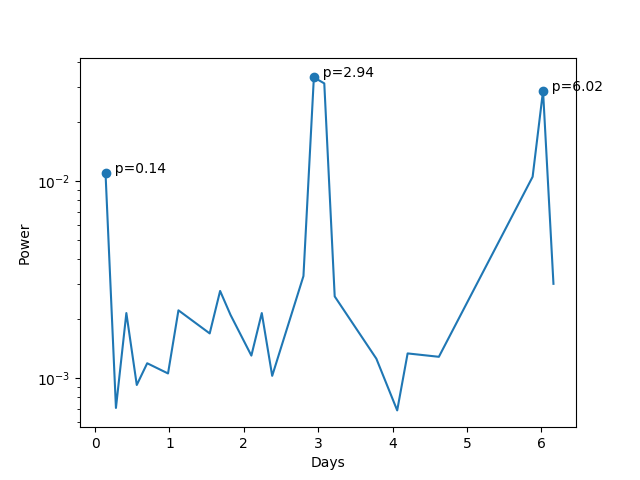
**Cập nhật 2:**

Dữ liệu huấn luyện là dữ liệu ở hình 10 kết hợp dữ liệu tài nguyên tiêu thụ trong lần cập nhật 1.

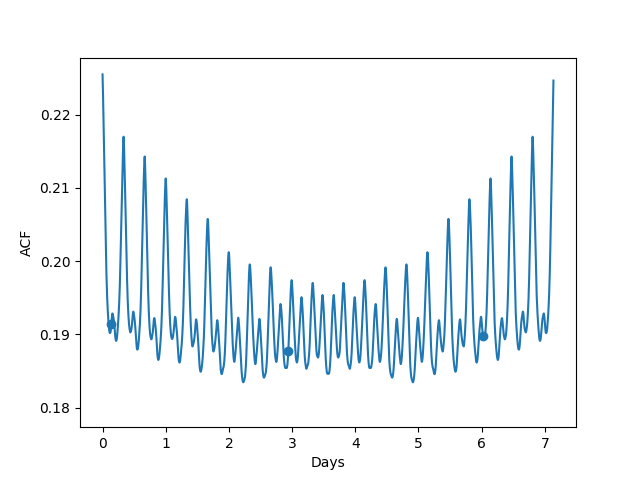
Tương tự, thuật toán AUTO PERIOD không tìm thấy chu kỳ dữ liệu.



Hình 44: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 2.



Hình 45: Periodogram của dữ liệu huấn luyện lần cập nhật 2. Các chu kỳ tìm được p=0.14, p = 2.94, p =6.02

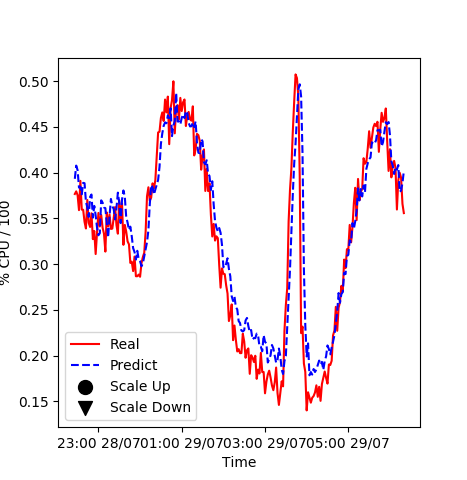


Hình 46: ACF của dữ liệu huấn luyện lần cập nhật 2. Các điểm trên hình là chu kỳ trong hình 45.

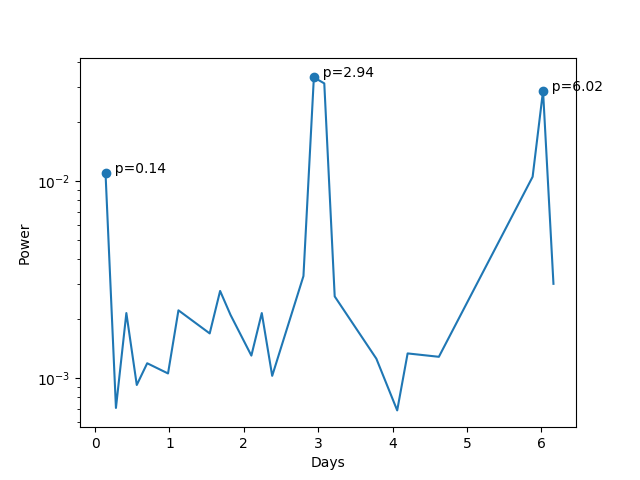
**Cập nhật 3:**

Dữ liệu huấn luyện là dữ liệu ở hình 10 kết hợp dữ liệu tài nguyên tiêu thụ trong lần cập nhật 1 và lần cập nhật 2.

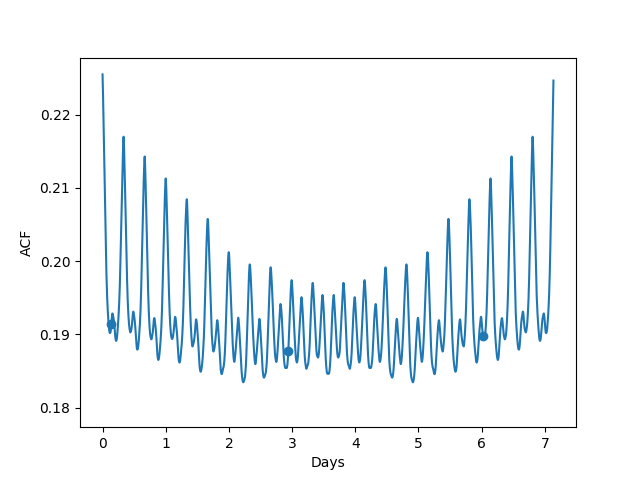
Tương tự, thuật toán AUTO PERIOD không tìm thấy chu kỳ dữ liệu.



Hình 47: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 3.



Hình 48: Periodogram của dữ liệu huấn luyện lần cập nhật 3. Các chu kỳ tìm được p=0.14, p = 2.94, p =6.02



Hình 49: ACF của dữ liệu huấn luyện lần cập nhật 3. Các điểm trên hình là chu kỳ trong hình 48.

**Nhận xét:**

* Nhìn chung, giống như mô hình 4-4, mô hình 2-4 cũng dự đoán thời điểm quá tải chậm hơn tài nguyên tiêu thụ, bởi các lần cập nhật đều không tìm ra chu kỳ phù hợp nên độ chính xác chưa cao.
* Tuy nhiên tại các thời điểm phát hiện quá tải và tạo thêm máy ảo, giá trị tài nguyên tiêu thụ có nhỏ hơn mô hình 4-4, nằm trong khoảng 50% đến 57%. Mặc dù vậy dự đoán vẫn chưa chính xác tại các đỉnh tương ứng với thời gian dữ liệu thay đổi đột ngột khi gửi thêm request, do dự đoán có giá trị nhỏ hơn giá trị tài nguyên tiêu thụ.
* Có thể kết luận mô hình 2-4 đưa ra dự đoán sát hơn so với mô hình 4-4 khi dữ liệu thay đổi đột ngột, tuy nhiên độ chính xác vẫn chưa cao.

2.2.4.3. So sánh các mô hình dự đoán 4-4 và 2-4.

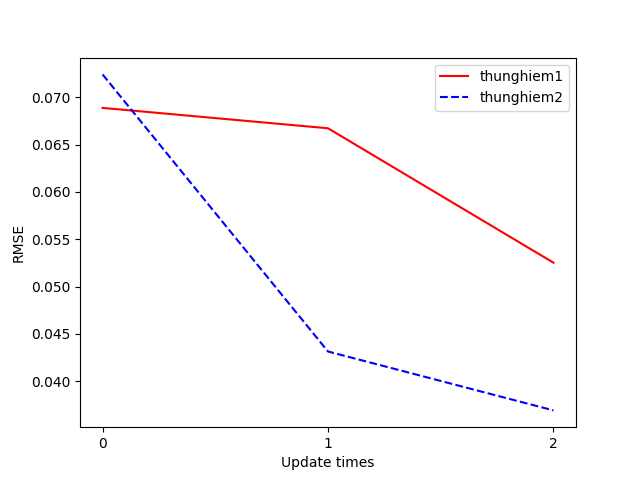
Đô đo được sử dụng để so sánh:

Trong đó, là giá trị dự đoán được tại thời điểm j

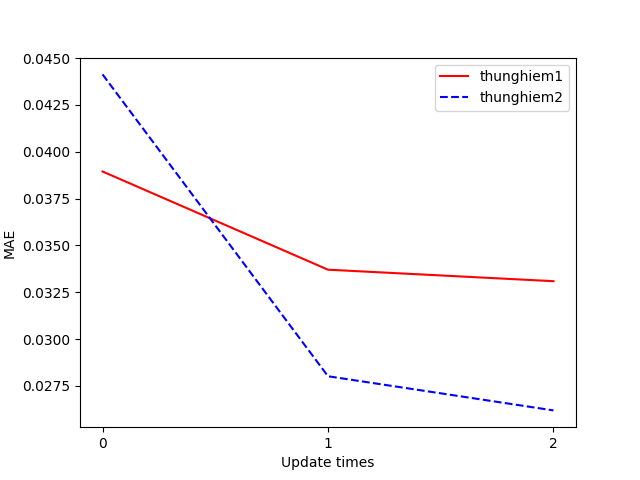
là giá trị thực sự.

là số điểm

So sánh 2 mô hình sau 3 lần cập nhật.



Hình 50: So sánh RMSE 2 mô hình 4-4 và 2-4. Đường màu đỏ là RMSE của mô hình 4-4 sau 3 lần cập nhật. Đường màu xanh là RMSE của mô hình 2-4 sau 3 lần cập nhật.



Hình 51: So sánh MAE của 2 mô hình 4-4 và 2-4. Đường màu đỏ là MAE của mô hình 4-4 sau 3 lần cập nhật. Đường màu xanh là MAE của mô hình 2-4 sau 3 lần cập nhật.

**Nhận xét:**

* Nhìn chung cả 2 mô hình đều có kết quả dự đoán tốt hơn sau nhiều lần cập nhật (có các sai số giảm dần).
* Mô hình 2-4 sau 3 lần cập nhật có tốc độ giảm sai số nhanh hơn mô hình 4-4. Kết hợp với kết luận ở phần 4.2 cho thấy mô hình 2-4 cho kết quả dự đoán tốt hơn trong trường hợp dữ liệu thay đổi đột ngột.

2.2.4.4. Thử nghiệm với dữ liệu không có đột biến trong thời gian dài.

**Mục đích:**

Sử dụng mô hình 2-4 để đánh giá độ hiệu quả của phương pháp dự đoán PD GABP trong co giãn tài nguyên khi không có đột biến.

**Thiết lập môi trường:**

Để tiện so sánh và đánh giá, tôi tạo một máy ảo có cấu hình giống với máy ảo thử nghiệm, trên cùng một hạ tầng phần cứng, gọi là vm\_mirror. Với việc tạo máy ảo này, tôi mong muốn rằng, với cùng 1 lượng request gửi đến, lượng phần trăm tài nguyên tiêu thụ của vm\_mirror với máy ảo thử nghiệm là như nhau.

Nếu với mỗi request gửi đến máy ảo thử nghiệm, request đó được sao chép và gửi tới vm\_mirror, ta có thể nói rằng, nếu không có mô hình PDGABP để dự đoán co giãn tài nguyên trên máy ảo thử nghiệm, thì phần trăm tài nguyên tiêu thụ của máy ảo thử nghiệm sẽ giống như phần trăm tài nguyên tiêu thụ của máy ảo vm\_mirror.

Để sao chép request, tôi sử dụng chương trình duplicator (<https://github.com/agnoster/duplicator>) để copy request gửi từ Jmeter đến máy ảo thử nghiệm.

Chạy lệnh:

*duplicator -f addr\_a:port\_a -d addr\_b:port\_b -p port [-i addr]*

Với addr\_a là địa chỉ haproxy của hệ thống, addr\_b là địa chỉ máy ảo vm\_mirror.

Dữ liệu huấn luyện:

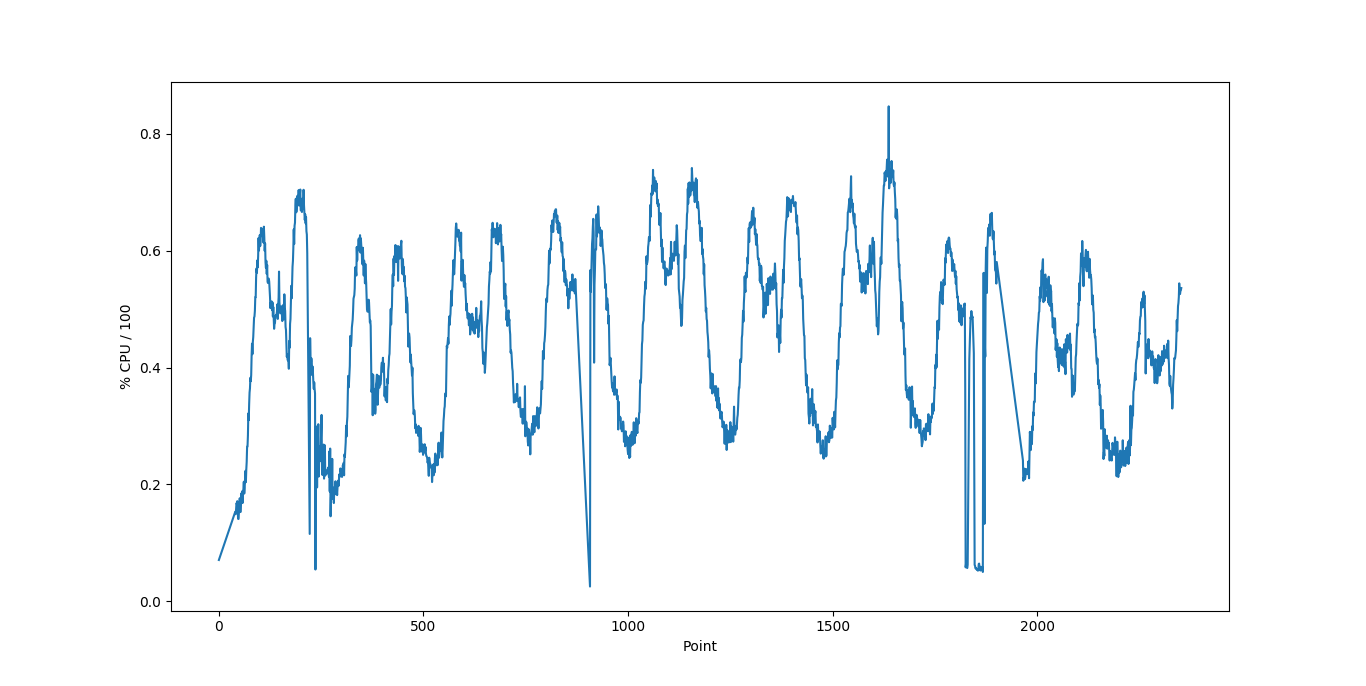
Theo mô hình 2-4, lấy mẫu 2 phút/lần.

Độ dài dữ liệu khoảng hơn 3 ngày.

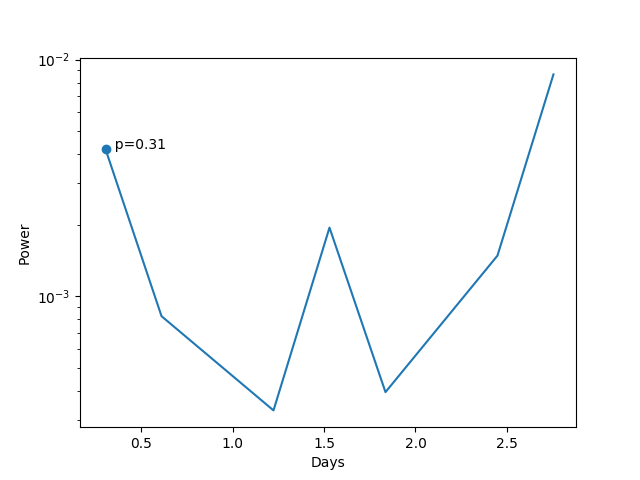
Các tham số của thuật toán AUTO PERIOD:

* Tần số lấy mẫu fs=720 (720 điểm tương ứng 1 ngày).
* Threshold=0.2

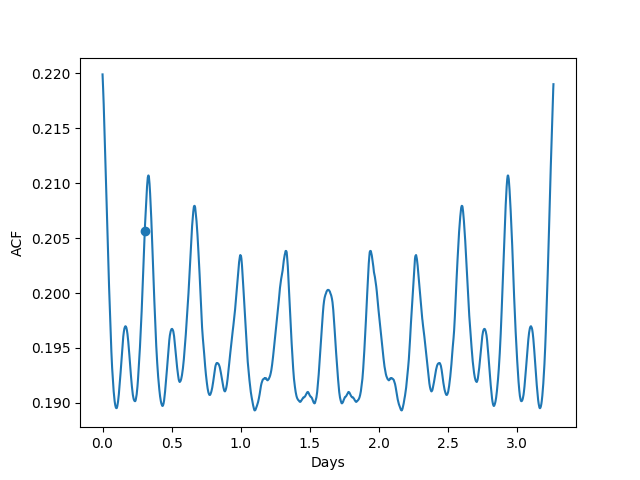
Sau khi chạy thuật toán, chu kỳ tìm được là 0.31 ngày.



Hình 52: Dữ liệu phần trăm CPU tiêu thụ dùng trong mô hình 2-4, lấy mẫu mỗi 2 phút 1 lần, độ dài khoảng hơn 3 ngày. Trục hoành là điểm dữ liệu, trục tung là phần trăm CPU đưa về khoảng 0-1.

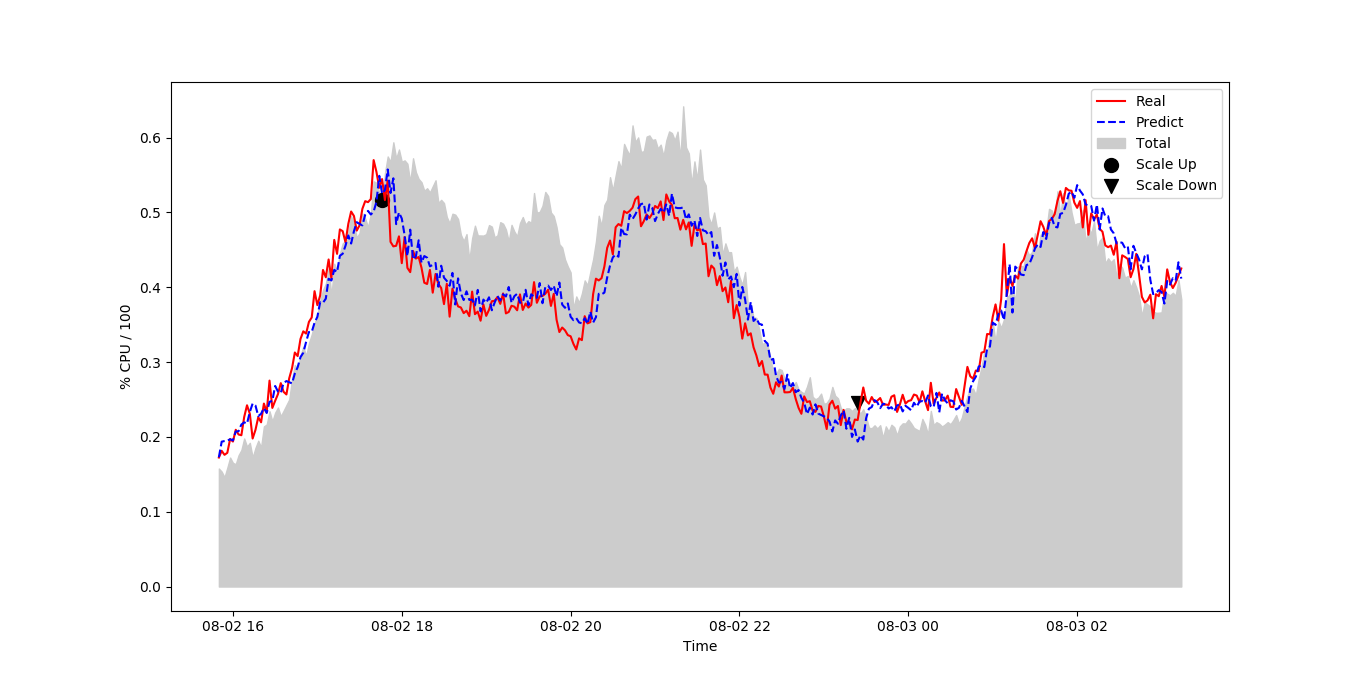


Hình 53: Periodogram của dữ liệu huấn luyện luyện. Chu kỳ tìm được p=0.31



Hình 54: ACF của dữ liệu huấn luyện. Các điểm trên hình là chu kỳ trong hình 53.

Kết quả thử nghiệm:



Hình 55: *Kết quả thử nghiệm:*

*Trục hoành: thời gian*

*Trục tung: phần trăm CPU tiêu thụ trong khoảng 0-1.*

*Đường màu đỏ: dữ liệu tài nguyên tiêu thụ đo được ở máy ảo thử nghiệm.*

*Đường màu xanh nét đứt: dữ liệu dự đoán.*

*Nền màu xám: dữ liệu tài nguyên tiêu thụ đo được ở vm\_mirror với mong muốn là gần giống với tài nguyên tiêu thụ ở máy ảo thử nghiệm nếu không có mô hình dự đoán.*

*Hình tròn đen: Thời điểm và giá trị tại thời điểm tạo thêm máy ảo.*

*Hình tam giác đen: Thời điểm và giá trị tại thời điểm giảm bớt máy ảo.*

*Độ dài thử nghiệm: 10 tiếng.*

**Nhận xét:**

* Kết quả dự đoán trường hợp này đã có sườn đi lên bám sát hơn với dữ liệu tài nguyên tiêu thụ của máy ảo thử nghiệm nên dự đoán khá tốt, máy ảo không bị quá tải (không bị vượt giả định 55% CPU tiêu thụ).
* Kết quả này cho thấy dự đoán có độ chính xác tốt hơn khi chu kỳ tìm được là phù hợp.

2.2.5. Kết luận.

* Phương pháp PD GABP cho độ chính xác tốt trong lý thuyết nhưng việc ứng dụng trong thực tế việc dự đoán quá tải trước để giãn tài nguyên cần phải lưu ý:
  + Việc dự đoán có độ sai số nhỏ là tốt, nhưng quan trọng hơn là sườn đi lên của dữ liệu dự đoán phải sát với dữ liệu tài nguyên tiêu thụ thực. Điều này mô hình cho độ chính xác khá tốt nếu dữ liệu có phát hiện được chu kỳ, trường hợp khác mô hình vẫn chưa có độ chính xác cao.
* Dự đoán thường xuyên hơn (so sánh giữa dự đoán 2 phút 1 lần và 4 phút 1 lần) sẽ tốt hơn trong trường hợp lượng request tăng đột biến.
* Điểm hạn chế là mô hình là cần phát hiện được chu kỳ thì dự đoán mới chứng minh được là tốt, nếu chỉ sử dụng sliding window để dự đoán thì chỉ ở mức chấp nhận được. Trong thực tế lượng request gửi đến 1 ứng dụng là không thể đoán trước và luôn thay đổi nên khó có thể có được chu kỳ nếu thời gian ngắn. Nhưng về tổng thể lâu dài, dữ liệu vẫn có thể có chu kỳ trong trường hợp lý tưởng. Các ứng dụng trên thực tế có chu kỳ là khá phổ biến. Có thể trong thời gian ngắn tính bằng ngày, không thể phát hiện ra chu kỳ, nhưng thời gian dài hơn ví dụ tính bằng tháng hoặc năm, chu kỳ phát hiện được có thể là 1, 2 tháng, hoặc là 1, 2 năm.

CHƯƠNG 3: KẾT LUẬN

Đồ án đã thu được các kết quả sau:

* Xây dựng được ứng dụng dự đoán co giãn tài nguyên dựa trên phương pháp dự đoán PD-GABP trên hạ tầng đám mây OpenStack.
* Triển khai ứng dụng và xây dựng môi trường thử nghiệm, mô phỏng dữ liệu requests theo thời gian thực trên đám mây OpenStack.
* Thử nghiệm và đưa ra kết luận, đánh giá hiệu quả của mô hình PD-GABP trong thực tế.

Do thời gian có hạn, đồ án vẫn còn những điểm hạn chế:

* Chỉ mới xây dựng ứng dụng và thử nghiệm, đánh giá trên độ đo phần trăm CPU, cần nghiên cứu mở rộng trên các độ đo khác để kết luận tốt hơn về khả năng ứng dụng thực tế của mô hình PD-GABP.

**TÀI LIỆU THAM KHẢO**

[1] Dang Tran, Nhuan Tran, Binh Minh Nguyen, Hieu Le, **PD-GABP – A Novel Prediction Model Applying for Elastic Applications in Distributed Environment, School of Information and Communication Technologies**, Hanoi University of Science and Technology, Hanoi, Vietnam.

[2] Mell, P. & Grance, T., 2011, **The NIST Definition of Cloud Computing**, National Institute of Standards and Technology, U.S. Department of Commerce

[3**] Introduction to Cloud Computing White Paper**, Dialogic Inc.

[4] Eromosele Omomhenle , **Introlduction to Auto Scaling - Horizontal and Vertial Auto-Scaling.** https://www.linkedin.com/pulse/introduction-auto-scaling-horizontal-vertical-eromosele-omomhenle. Truy cập lần cuối 30/5/2017

[5] **A View of Cloud Computing.**

<https://cacm.acm.org/magazines/2010/4/81493-a-view-of-cloud-computing/fulltext>?

searchterm=Amazon.com%3A+Removing+the+Spam%3A+Email+Processing. Truy cập lần cuối 30/5/2017.

[6] **Scaling in Azure**. https://learnwithshahriar.wordpress.com/tag/horizontal-scaling/. Truy cập lần cuối 30/5/2017

[7] **Horizontally Scaling Php Applications**, https://blog.digitalocean.com/horizontally-scaling-php-applications/. Truy cập lần cuối 30/5/2017.

[8] **Getting Started with Auto Scales**, http://docs.aws.amazon.com/autoscaling/latest/userguide/GettingStartedTutorial.html#gs-create-lc. Truy cập lần cuối 30/5/2017.

[9**] Automatic Scaling with Amazon EC2**, https://aws.amazon.com/blogs/compute/automatic-scaling-with-amazon-ecs/. Truy cập lần cuối 30/5/2017

[10] OpenStack, https://vietstack.wordpress.com/2014/02/15/OpenStack-la-gi-va-de-lam-gi/. Truy cập lần cuối 30/5/2017.

[11] **OpenStack Networking**, https://docs.OpenStack.org/newton/networking-guide/intro-os-networking.html. Truy cập lần cuối 30/5/2017.

**[12]** Cadvisor github issues**,https://github.com/google/cadvisor/issues/1563. Truy cập lần cuối 30/5/2017.**

**[13] What is Cloud Computing?** [**http://researchhubs.com/post/computing/cloud-computing/what-is-cloud-computing.html**](http://researchhubs.com/post/computing/cloud-computing/what-is-cloud-computing.html)**. Truy cập lần cuối 30/5/2017.**