**Thử nghiệm**

**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 trong các trường hợp:

- Đánh giá hiệu quả của phương pháp dự đoán PD GABP trong trường hợp dữ liệu tài nguyên tiêu thụ thay đổi đột ngột. So sánh các mô hình dự đoán:

* 4-4: lấy dữ liệu trung bình và dự đoán 4 phút 1 lần, dự đoán trước 4 phút tương lai.
* 2-4: 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 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. 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.

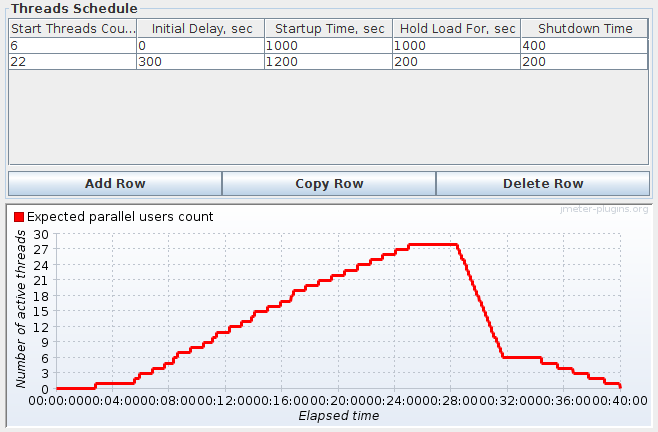
- Startup Time, sec: thời gian số thread của dòng tăng lên từ 0 đến khi đủ Start Threads Count.

- 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 1: 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},*

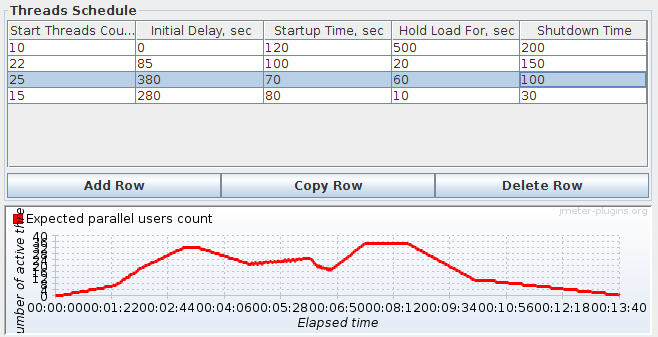
*{22,85,100,20,150},*

*{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).



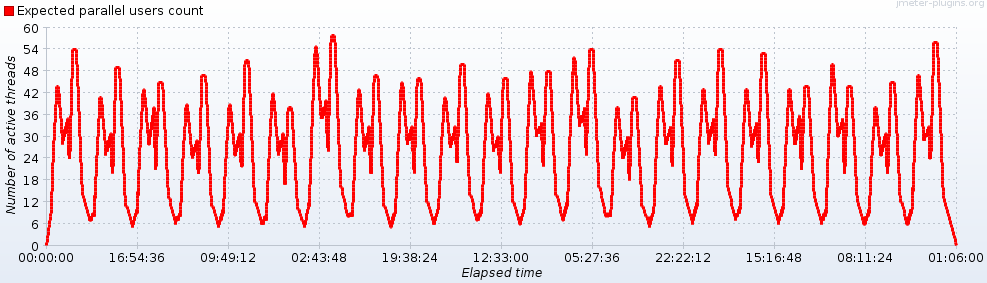
Hình 2: Mẫu Pattern biểu diễn trong Jmeter.

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.

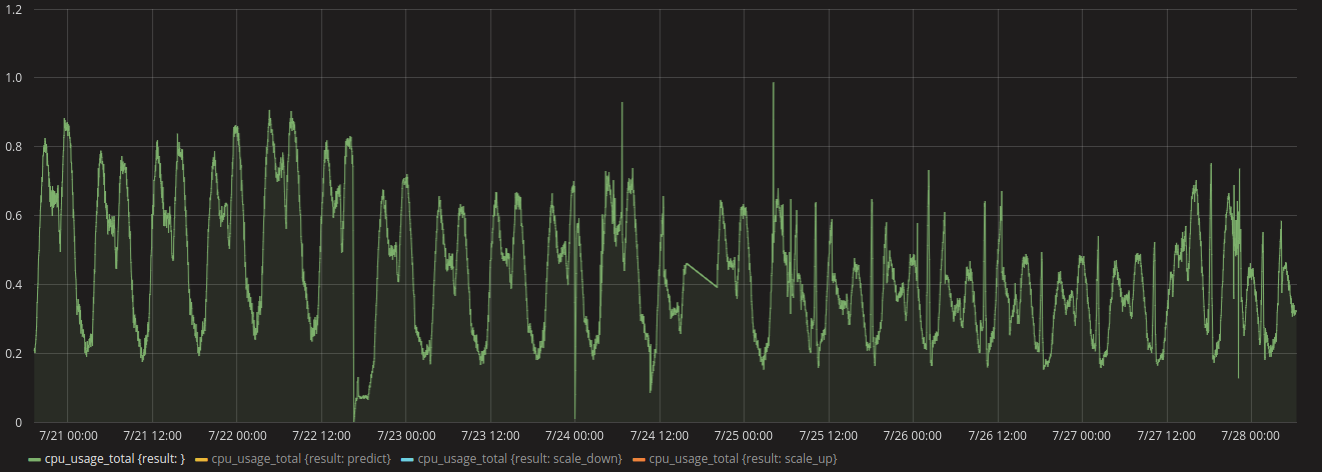


Hình 3: Kịch bản tạo ra với 21 chu kỳ, mỗi chu kỳ dài 8 tiếng. Trục hoành là thời gian, trục tung là số lượng thread còn sống tại mỗi thời điểm.

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 4: 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.

**3. Môi trường thử nghiệm.**

**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.

**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.

**4. Thử nghiệm.**

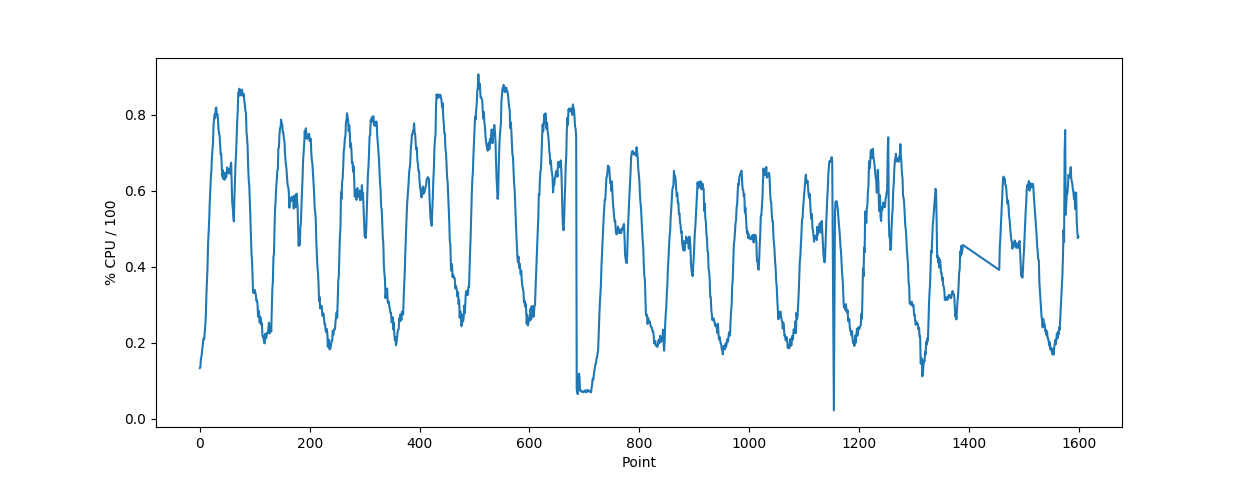
**4.1. Thử nghiệm mô hình 4-4 trong trường hợp dữ liệu thay đổi đột ngột.**

***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.

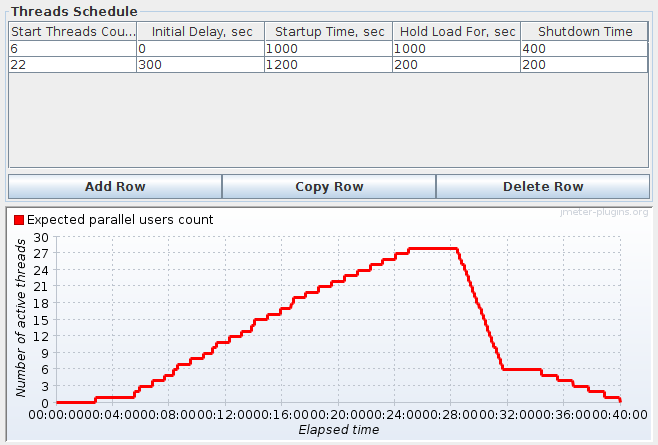


Hình 5: Dữ liệu phần trăm CPU tiêu thụ dùng trong mô hình 4-4, lấy mẫu mỗi 4 phút 1 lần, độ dài khoảng 5 ngày. Trục hoành là điểm dữ liệu, trục tung là phần trăm CPU đưa về khoảng 0-1.

**Kịch bản thử nghiệm:**

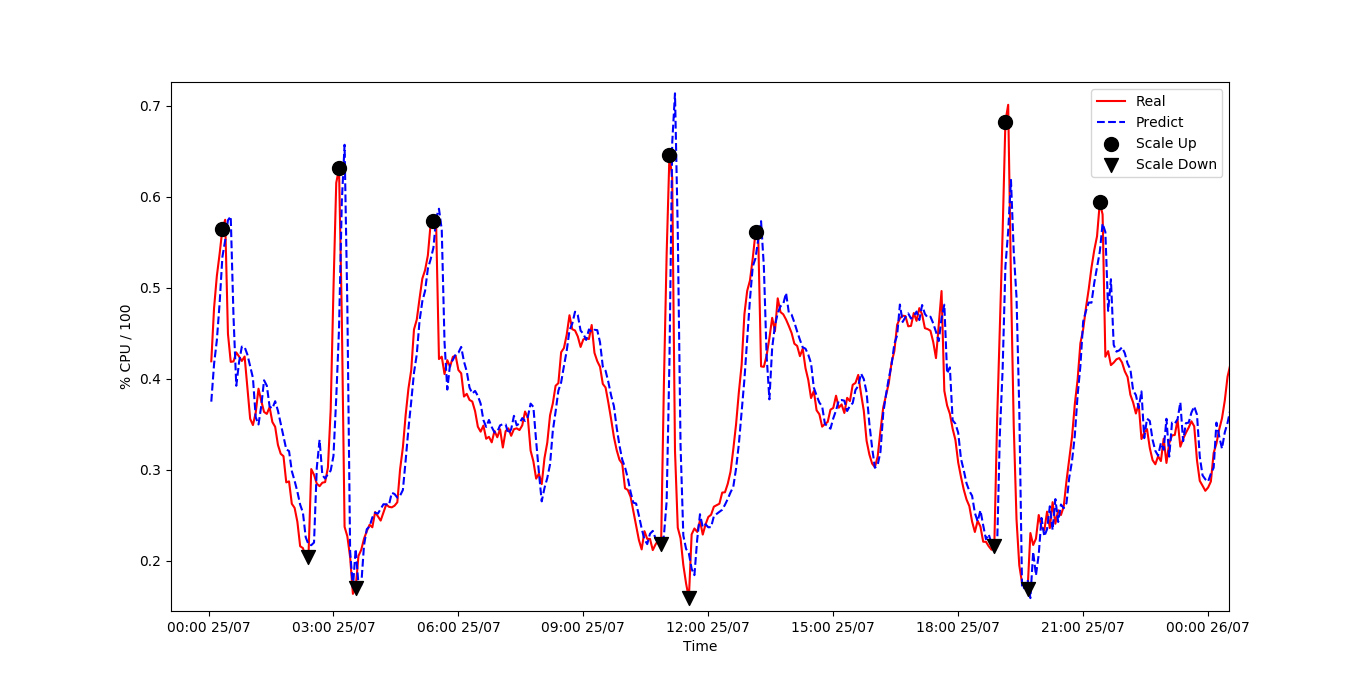
Thử nghiệm trong 24 giờ, 3 lần cập nhật mạng neural (8 tiếng cập nhật 1 lần).

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

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



*Hình 7: Kết quả thử nghiệm mô hình 4-4.*

*Trục hoành: thời gian.*

*Trục hoành: 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).

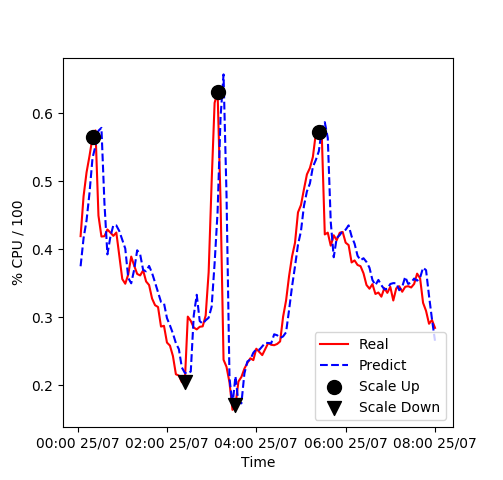
Hình 9 biểu diễn periodogram của dữ liệu huấn luyện. Các điểm tròn màu xanh trên hình 9 là các điểm cực trị có Power > threshold \* max(Power).

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.

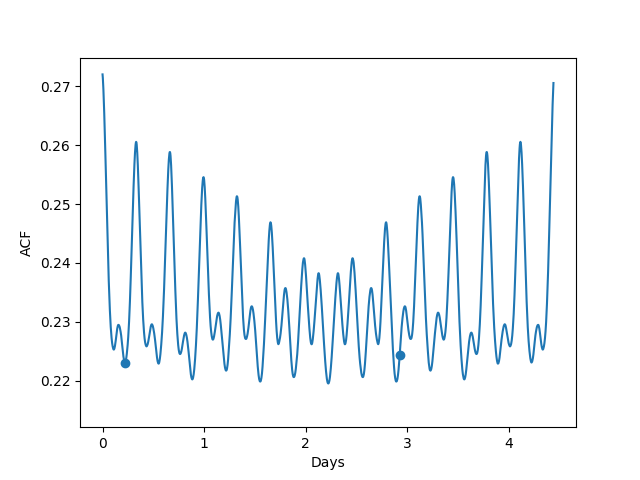
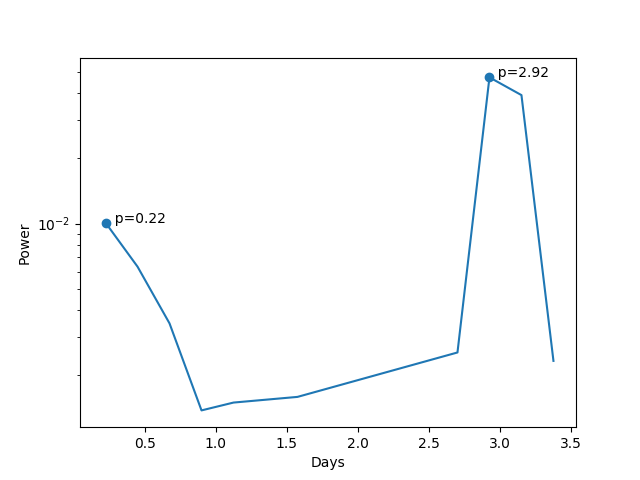
Thuật toán AUTO PERIOD tính periodogram để tìm các chu kỳ có khả năng là chu kỳ đúng, sau đó tính ACF và kiểm tra lại và chọn ra chu kỳ phù hợp nhất.

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



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

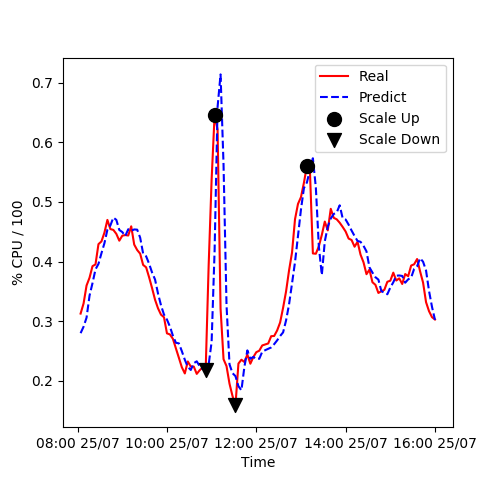
Hình 10: ACF của dữ liệu huấn luyện lần cập nhật 1. Các điểm trên hình là chu kỳ p=0.22 và p=2.92 tìm được trên hình 9.

**Cập nhật lần 2:**

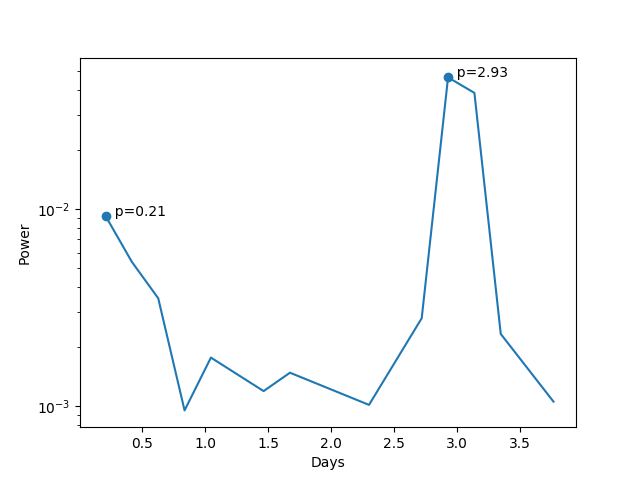
Dữ liệu huấn luyện: Là dữ liệu huấn luyện trình bày ở trên kết hợp với dữ liệu theo dõi được của server thử nghiệm trong khi chạy cập nhật 1.

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.

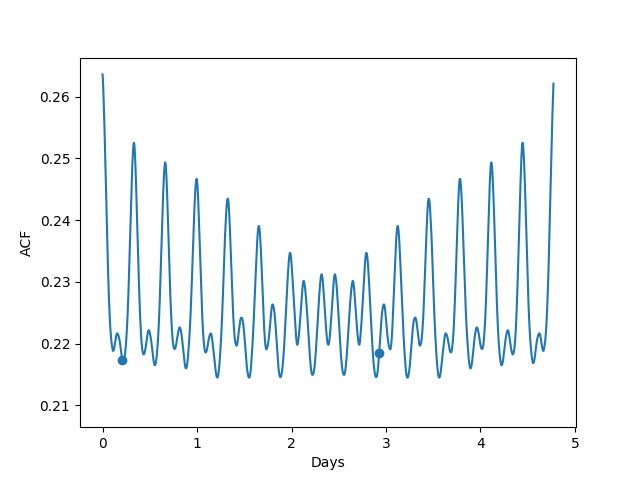
Do đó, dữ liệu huấn luyện là không có chu kỳ.



Hình 11: Kết quả thử nghiệm mô hình 4-4 lần cập nhật 2.



Hình 12: Periodogram của dữ liệu huấn luyện lần cập nhật 2. Các chu kỳ tìm được p=0.21, p = 2.93.

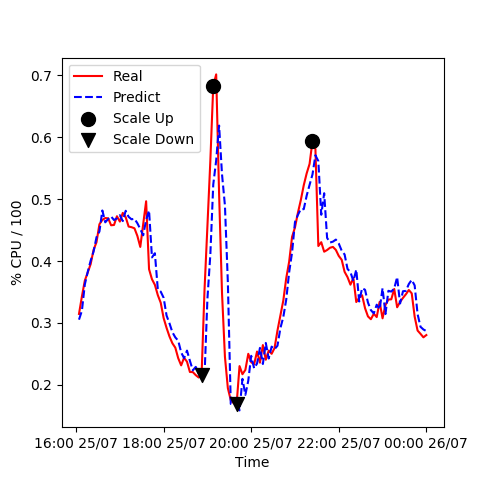


Hình 13: ACF của dữ liệu huấn luyện lần cập nhật 2. Các điểm trên hình là chu kỳ p=0.21 và p=2.93 tìm được trên hình 12.

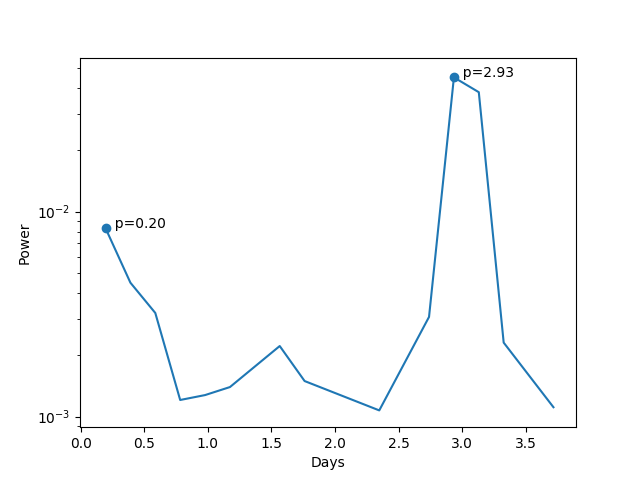
**Cập nhật lần 3:**

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.

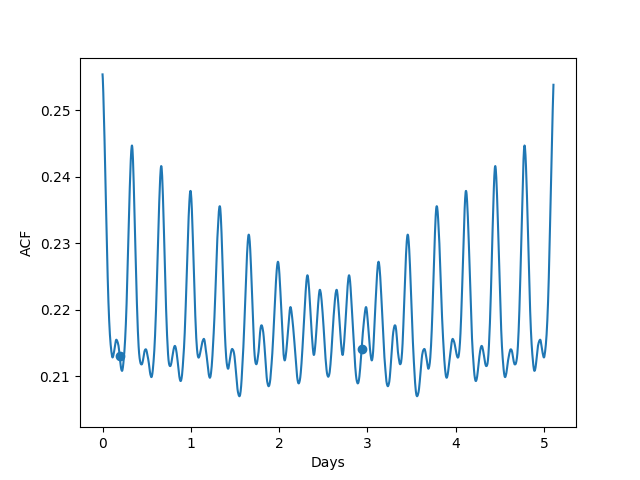
Tương tự, sau khi chạy thuật toán AUTO PERIOD, dữ liệu huấn luyện không có chu kỳ.



Hình 14: Kết quả thử nghiệm mô hình 4-4 lần cập nhật 3.



Hình 15: Periodogram của dữ liệu huấn luyện lần cập nhật 3. Các chu kỳ tìm được p=0.20, p = 2.93



Hình 16: 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 15.

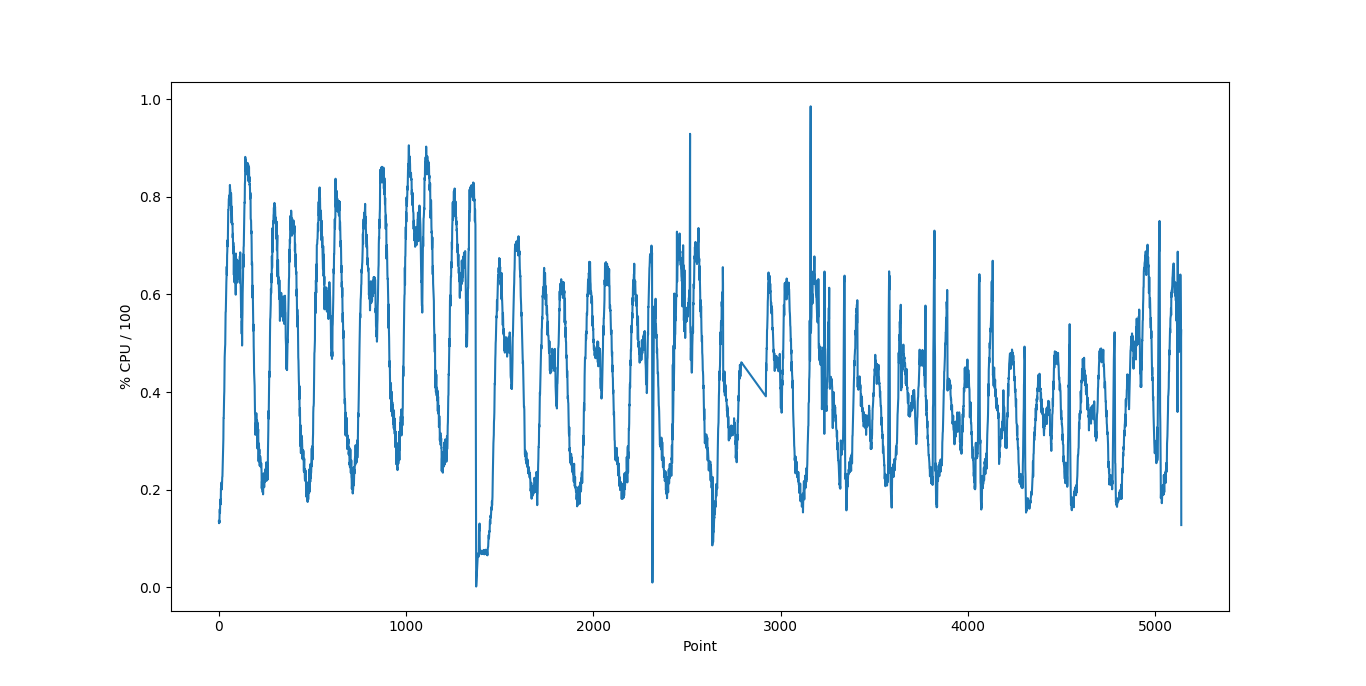
**Nhận xét:**

* 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.

**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.



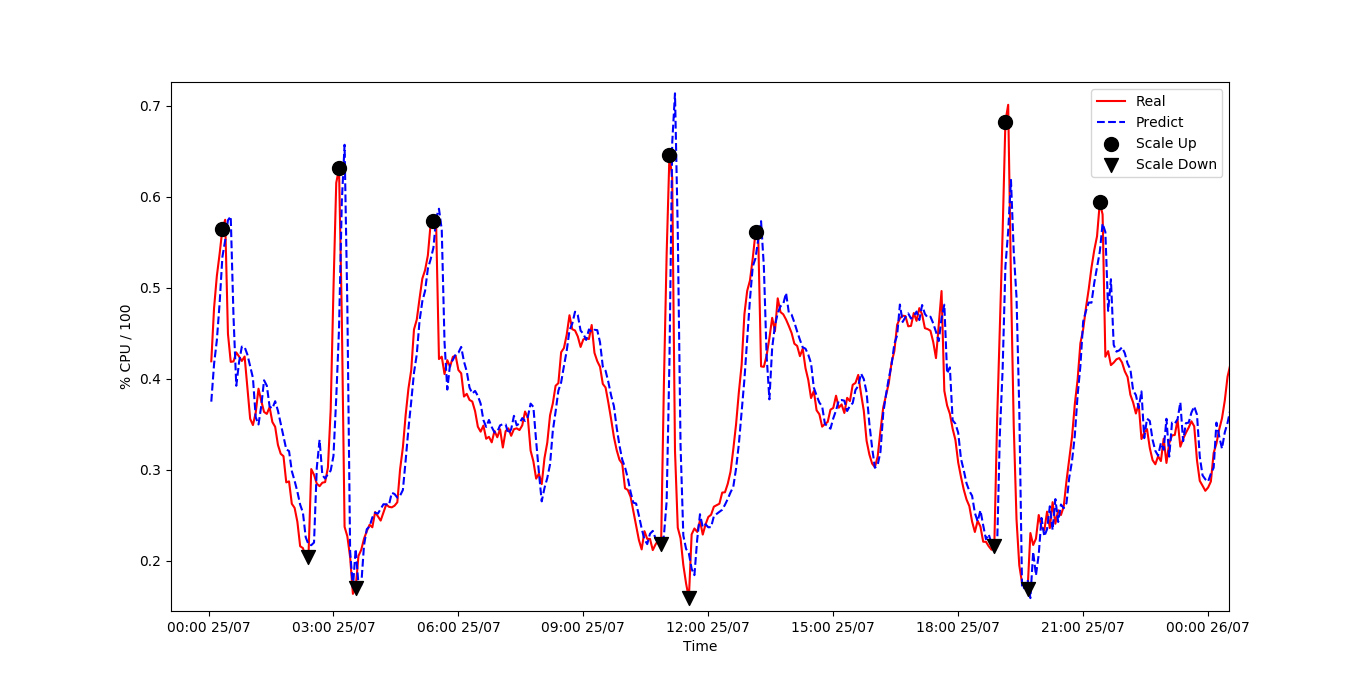
Hình 17: Dữ liệu phần trăm CPU tiêu thụ dùng trong mô hình 2-4, lấy mẫu mỗi 2 phút 1 lần, độ dài khoảng 7 ngày. Trục hoành là điểm dữ liệu, trục tung là phần trăm CPU đưa về khoảng 0-1.

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 18: Kết quả thử nghiệm mô hình 2-4.

*Trục hoành: thời gian. Trục hoành: 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.*

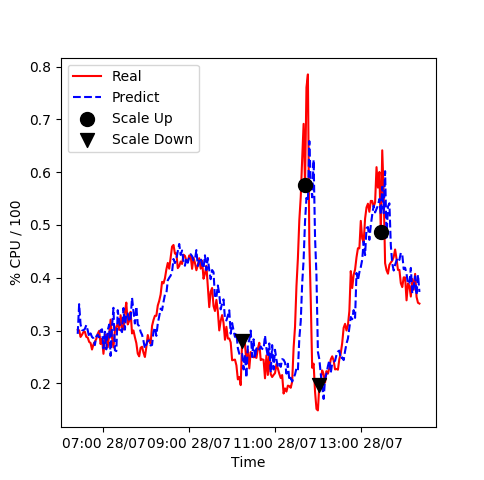
*Hình tam giác đen: Giá trị x: thời điểm phát hiện và xóa bớt máy ảo co tài nguyên.*

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

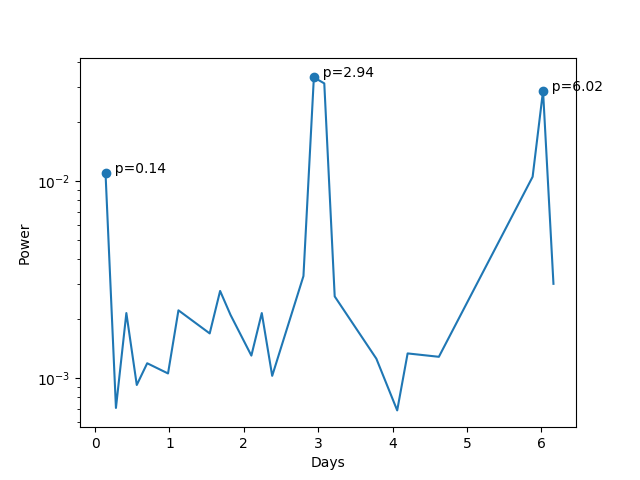
**Cập nhật 1:**

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

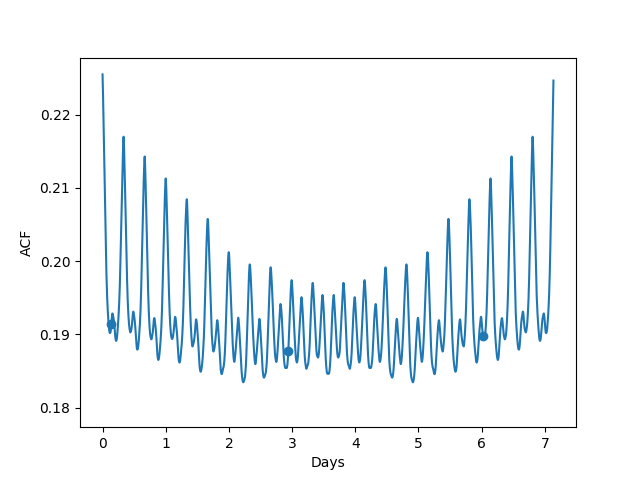
Sau khi chạy thuật toán AUTO PERIOD, các chu kỳ tìm được trên periodogram đều không phù hợp. Dữ liệu không có chu kỳ.

**

Hình 19: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 1.



Hình 20: Periodogram của dữ liệu huấn luyện lần cập nhật 1. Các chu kỳ tìm được p=0.14, p = 2.94, p =6.02

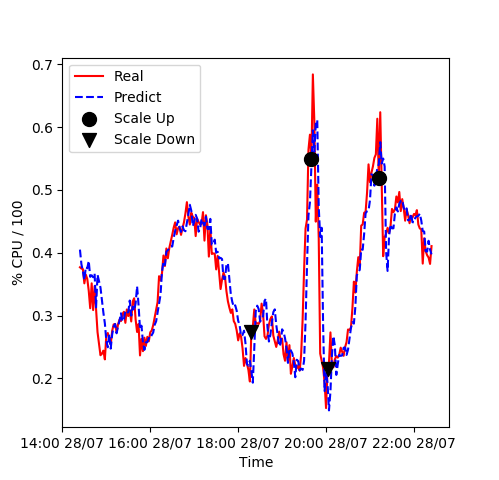


Hình 21: ACF của dữ liệu huấn luyện lần cập nhật 1. Các điểm trên hình là chu kỳ trong hình 20.

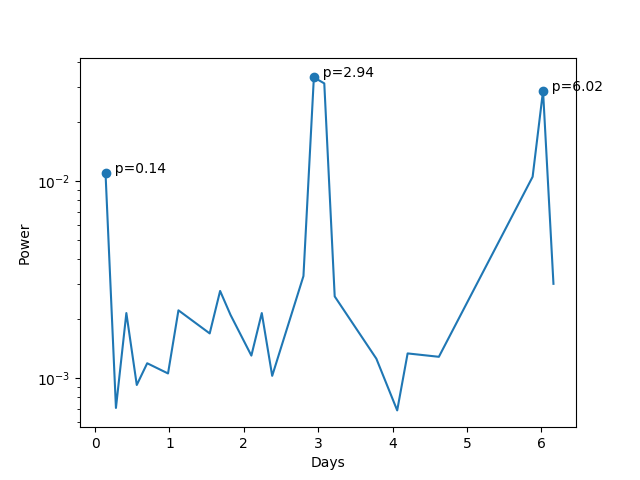
**Cập nhật 2:**

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

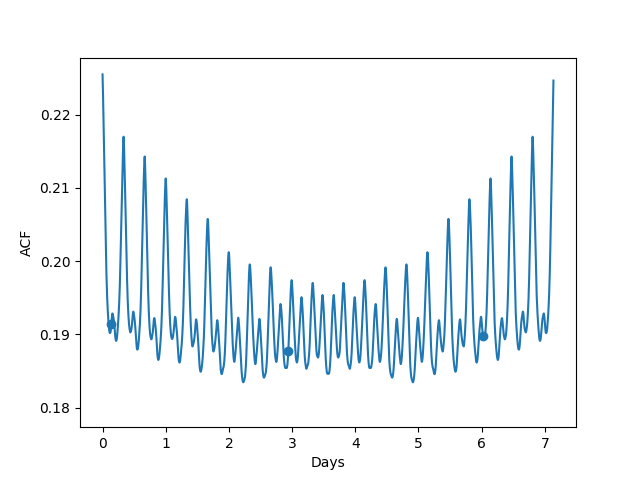
Tương tự, thuật toán AUTO PERIOD không tìm thấy chu kỳ dữ liệu.



Hình 22: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 2.



Hình 23: Periodogram của dữ liệu huấn luyện lần cập nhật 2. Các chu kỳ tìm được p=0.14, p = 2.94, p =6.02

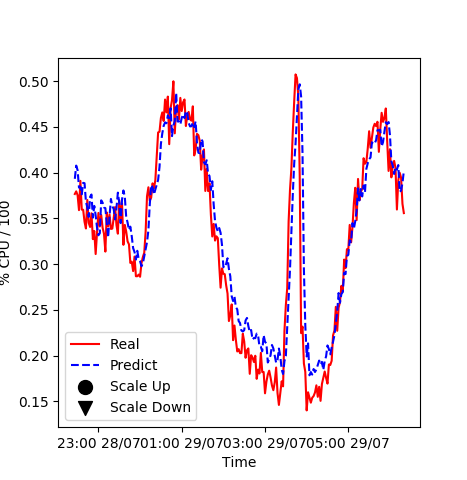


Hình 24: ACF của dữ liệu huấn luyện lần cập nhật 2. Các điểm trên hình là chu kỳ trong hình 23.

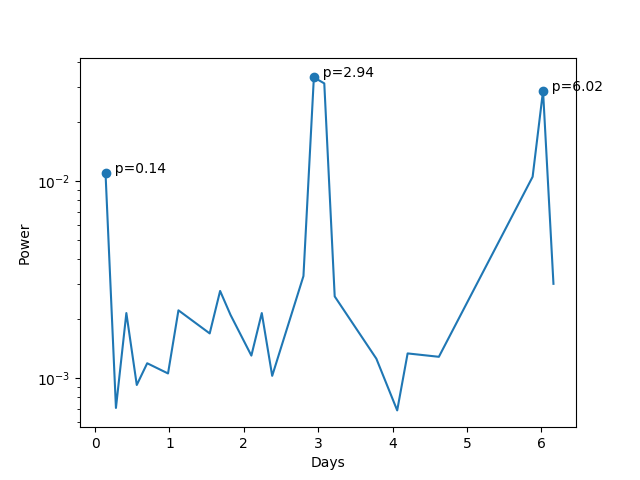
**Cập nhật 3:**

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.

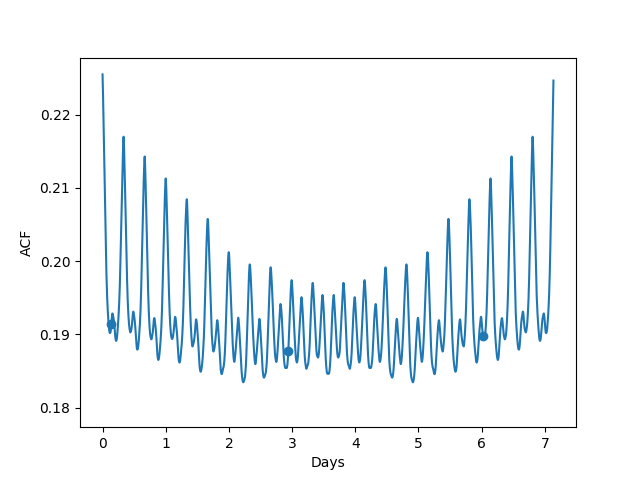
Tương tự, thuật toán AUTO PERIOD không tìm thấy chu kỳ dữ liệu.



Hình 25: Kết quả thử nghiệm mô hình 2-4 lần cập nhật 3.



Hình 26: 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 27: ACF của dữ liệu huấn luyện lần cập nhật 3. Các điểm trên hình là chu kỳ trong hình 26.

**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.

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

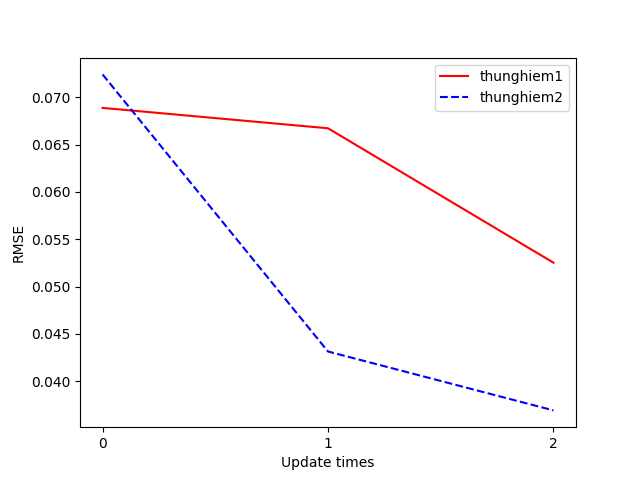
Đô đo được sử dụng để so sánh:

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

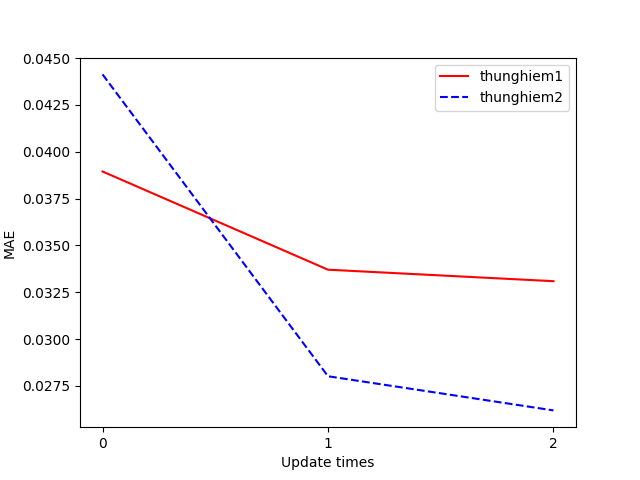
là giá trị thực sự.

là số điểm

So sánh 2 mô hình sau 3 lần cập nhật.



Hình 28: 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 29: 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.

**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:

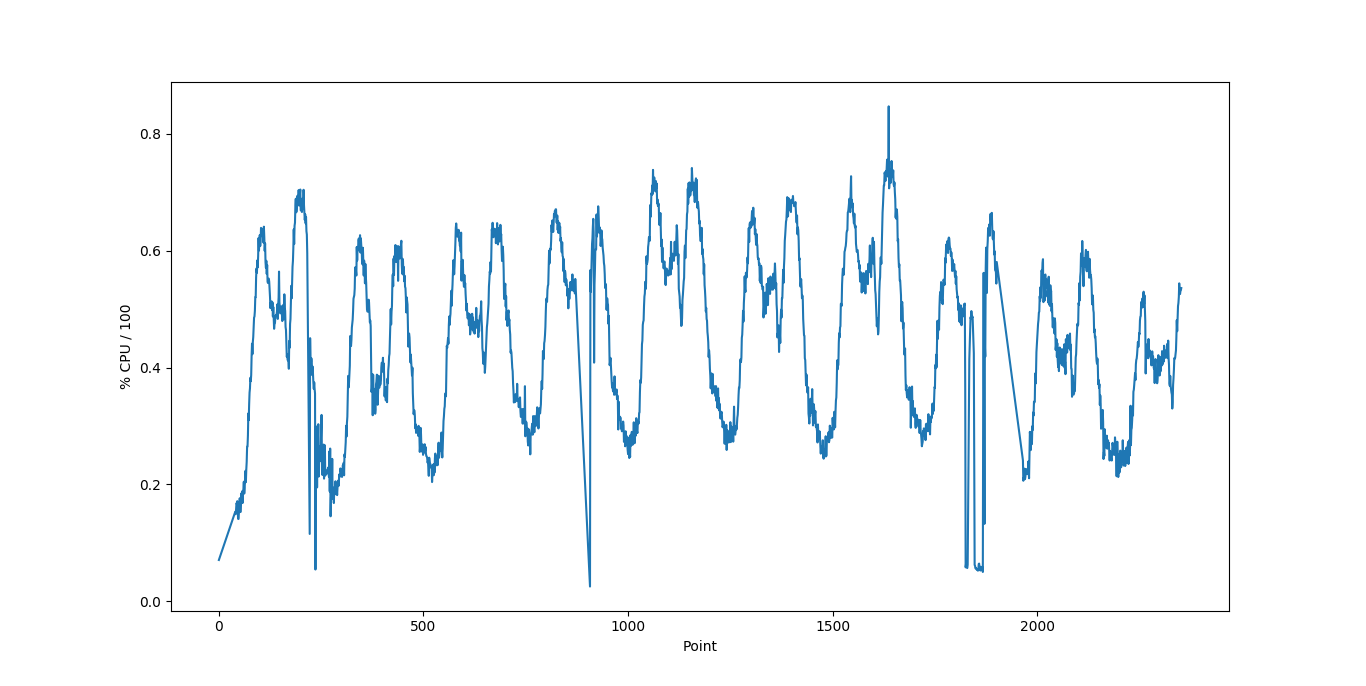
Theo mô hình 2-4, lấy mẫu 2 phút/lần.

Độ dài dữ liệu khoảng hơn 3 ngày.

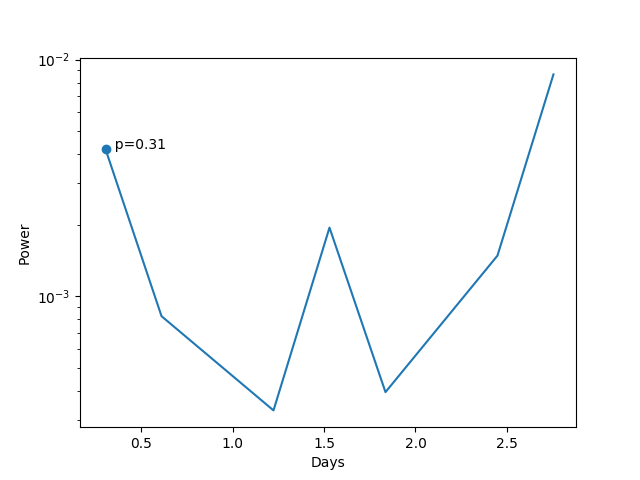
Các tham số của thuật toán AUTO PERIOD:

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

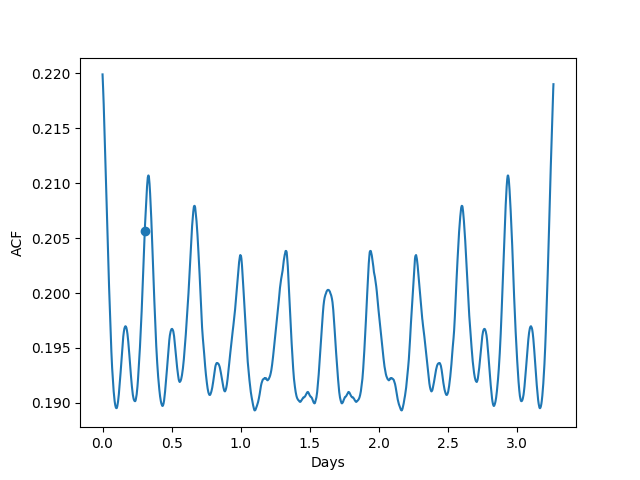
Sau khi chạy thuật toán, chu kỳ tìm được là 0.31 ngày.



Hình 30: 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.

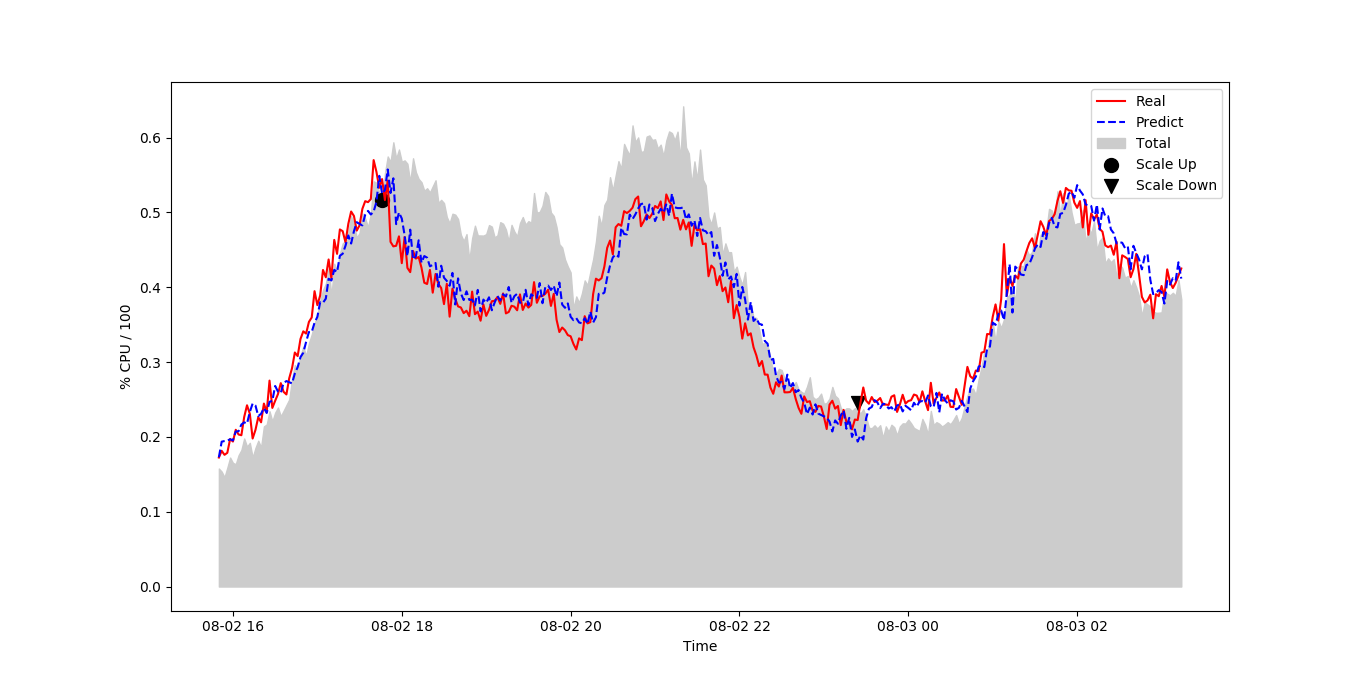


Hình 31: Periodogram của dữ liệu huấn luyện luyện. Chu kỳ tìm được p=0.31



Hình 32: ACF của dữ liệu huấn luyện. Các điểm trên hình là chu kỳ trong hình 31.

Kết quả thử nghiệm:



Hình 33: *Kết quả thử nghiệm:*

*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.

**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.