ĐIỀU CHỈNH TÀI NGUYÊN TỰ ĐỘNG DỰA VÀO THAM SỐ CHẤT LƯỢNG DỊCH VỤ TRONG CLOUD COMPUTING

Nguyễn Hồng Sơn

Học Viện Công Nghệ Bưu Chính Viễn Thông cơ sở tại Thành Phố Hồ Chí Minh

Tóm tắt: Việc cung cấp tính năng điều chỉnh tài nguyên tự động (auto scaling) trên hạ tầng IaaS của nhà cung cấp dịch vụ điện toán đám mây đã giúp thuê bao tránh được tình trạng suy thoái dịch vụ và tiết kiệm chi phí. Cùng với đó là sự xuất hiện các thuật toán auto scaling để tự động hóa việc cung ứng cũng như thu hồi tài nguyên một cách kịp thời. Phương pháp auto scaling dựa vào luật đối sánh giá trị tham số hiện hành với ngưỡng được sử dụng phổ biến trong môi trường công nghiệp. Tính hiệu quả của auto scaling dạng này tùy thuộc vào cách chọn tham số. Trong bài báo này đề xuất sử dụng tham số chất lượng dịch vụ thay cho cách dùng tham số mức độ sử dụng CPU đang dùng phổ biến hiện nay. Bài báo cũng đưa ra mô hình đánh giá tính hiệu quả của auto scaling dạng này và dựa vào đó để kiểm tra tính hiệu quả của phương pháp. Mô hình đã được cài đặt và đã xác định được tính hiệu quả theo cách chọn các tham số dịch vụ.

Từ khóa: điện toán đám mây, điều chỉnh tài nguyên tự động, tham số chất lượng dịch vụ, waiting rate.

I. GIỚI THIỆU

Công nghệ điện toán đám mây đã tạo bước phát triển đột phá trong lĩnh vực công nghệ thông tin. Điện toán đám mây được xem như giải pháp đem đến nhiều đặc tính nổi trội như on-demand self-service, global network access, distributed resource pooling, scalable, và measured service [1]. Công nghệ này tạo điều kiện cho các doanh nghiệp giảm chi phí đầu tư, giảm chi phí vận hành và cho phép các nhà cung cấp dịch vụ đám mây tiếp cận được nhiều khách hàng hơn. Thật vậy, trong thời gian qua điện toán đám mây đã trở thành nguồn lực cung cấp dịch vụ tính toán vô cùng quan trọng cho mọi nhu cầu tính toán từ khắp mọi nơi trên thế giới. Hạ tầng tính toán tại các tổ chức và doanh nghiệp đã dần được thay thế bởi dịch vụ hạ tầng của điện toán đám mây IaaS. Bên cạnh đó điện toán đám mây còn là nguồn cung cấp các dịch vụ mức cao vô cùng phong phú và tiện lợi như PaaS, SaaS, DaaS (Database as a Service), v.v. Ngày nay không chỉ các tổ chức và doanh nghiệp mà các cá nhân đều có thể lấy ngay những dịch vụ cần thiết từ điện toán đám mây. Tất cả đều được cung cấp qua một mô hình thuê bao như các hệ thống cung cấp điện, nước truyền thống. Các nhà cung cấp dịch vụ đám mây nổi tiếng hiện nay như Amazon, Google, RackSpace,
Microsoft Azure đều cung cấp đầy đủ các dịch vụ nói trên. Mở rộng hơn nữa, điện toán đám mây đã thúc đẩy một thị trường kinh doanh và tiêu thụ các dịch vụ công nghệ thông tin đầy tiềm năng và vô cùng phong phú. Nhiều nhà cung cấp sơ cấp và thứ cấp đã xuất hiện và các mô hình kinh doanh trên điện toán đám mây đã và đang phát triển không ngừng.

Ngay từ ý tưởng ban đầu điện toán đám mây cung cấp công suất tính toán tương tự như ngành điện lực cung cấp điện năng cho khách hàng. Tuy nhiên có sự khác nhau về cách tổ chức cung cấp giữa hai hệ thống do đặc thù của dịch vụ. Với hệ thống cung ứng điện năng, khách hàng không cần phải đăng ký trước lượng điện năng tiêu thụ, dùng bao nhiêu thì trả bấy nhiêu. Nhưng với đặc thù của dịch vụ tính toán, thông thường khách hàng sẽ đăng ký lượng tài nguyên tính toán cần dùng và trả chi phí cho lượng tài nguyên thuê bao đó. Điều này có nghĩa là các tài khoản người dùng trên hệ thống dịch vụ điện toán đám mây được cấp một lượng tài nguyên cố định và trả chi phí cho lượng tài nguyên tương ứng. Một doanh nghiệp thuê bao tài nguyên hệ thống đám mây để xây dựng những cổng thông tin kinh doanh của mình cần phải tính toán kỹ lưỡng sao cho vừa đáp ứng tốt hoạt động truy cập từ khách hàng của công ty và vừa tiết kiệm chi phí.Tuy nhiên lượng truy cập là yếu tố ngẫu nhiên có khả năng thay đổi, ví dụ tùy theo mùa hay sự kiện, như hệ thống thương mại điện tử trong mùa cao điểm, cho nên trên thực tế nếu một doanh nghiệp thuê tài nguyên quá dư để dự phòng cho trường hợp lượng truy cập tăng bất thường sẽ lãng phí trong phần lớn thời gian thuê, nhưng thuê vừa đủ thì sẽ bị quá tải trong những thời điểm tải tăng đột biến. Để khắc phục khó khăn này, các nhà cung cấp dịch vụ đám mây như Amazon, Azure, Google đã cung cấp cho khách hàng một giải pháp, cũng là dịch vụ đặc biệt được gọi là dịch vụ bổ sung tài nguyên ngoài gói thuê bao một cách tự động hay dịch vụ auto scaling. Dịch vụ mới này được thực hiện nhờ vào cơ chế auto scaling được bổ sung vào trong hệ thống đám mây. Cơ chế auto scaling tự động bổ sung tài nguyên vào gói thuê bao của khách hàng một cách tạm thời trong

Tác giả liên hệ: Nguyễn Hồng Sơn
Email: ngson@ptithcm.edu.vn
Đến tòa soạn: 5/2020, chỉnh sửa: 6/2020, chấp nhận đăng: 7/2020

trường hợp hệ thống thuê bao của khách hàng bị quá tải do lượng truy cập và nhu cầu xử lý tăng đột biến. Cơ chế này cũng tự động thu hồi lượng tài nguyên bổ sung khi yếu tố gây quá tải không còn. Cơ chế auto scaling không chỉ đem lại lợi ích cho thuê bao mà còn tạo điều kiện cho nhà cung cấp dịch vụ điện toán đám mây tối đa doanh thu khi gia tăng được số thuê bao và có thể tận dụng được các tài nguyên nhàn rỗi một cách linh hoạt.

Auto scaling không chỉ hữu dụng cho hệ thống đám mây dùng máy ảo, trong hệ thống đám mây dùng container xuất hiện trong thời gian gần đây, kỹ thuật auto scaling cũng đóng vai trò rất quan trọng. Các ứng dụng đám mây dựa trên container cần được cung ứng đủ lượng tài nguyên để hoạt động ổn định dưới các điều kiện tải khác nhau và auto scaling đã được sử dụng để điều phối container trong trường hợp này. Ví dụ như trong Kubernettes [7], một mã nguồn mở dàn dựng container nổi tiếng hiện nay, có chứa thành phần autoscaler để bổ sung và thu hồi các pod đáp ứng yêu cầu tài nguyên của các ứng dụng theo các tiêu chí đã được định trước.

Chức năng của auto scaling nhìn thoáng qua thấy cũng đơn giản nhưng khi đặt ra yêu cầu về tính hiệu quả thì trở thành vấn đề phức tạp. Cơ chế được yêu cầu mang tính tự động theo khuynh hướng tự động hóa hiện đại. Cơ chế cần xác định khi nào và bao nhiêu trong việc bổ sung tài nguyên tạm thời để hệ thống thuê bao của khách hàng không bị giảm chất lượng dịch vụ do quá tải và cũng chỉ bổ sung vừa đủ để tiết kiệm chi phí. Mặt khác auto scaling cũng phải xác định thời điểm và thu hồi lượng tài nguyên bổ sung một cách kịp thời để giảm chi phí cho khách hàng. Xác định chính xác thời điểm và lượng bổ sung trong hệ thống vận hành theo thời gian thực là điều không dễ dàng. Các phương pháp auto scaling sẽ phải có cách thức để cảm nhận tình trạng của hệ thống để đưa ra quyết định cung ứng và thu hồi tài nguyên chính xác. Quyết định của auto scaling sẽ ảnh hưởng trực tiếp đến tốc độ đáp ứng của hệ thống dịch vụ và hiệu quả sử dụng tài nguyên. Sự khác biệt giữa các phương pháp auto scaling là độ chính xác trong việc ra quyết định và thi hành trên môi trường công nghiệp.

Căn cứ các phương pháp auto scaling được công bố từ trước đến nay có thể xếp vào ba dạng chính, dạng thứ nhất là auto scaling dựa vào các tham số được đo lường trên hệ thống theo thời gian thực, dạng thứ hai là auto scaling dựa vào công nghệ dự báo và dạng thứ ba là auto scaling dựa vào công nghệ tri thức. Ứng với dạng thứ nhất, các ngưỡng giá trị được thiết lập và phương pháp sẽ đưa ra quyết định trên cơ sở đối sánh giữa giá trị hiện hành của tham số và ngưỡng theo một qui tắc cho trước, vì thế dạng này cũng thường gọi là rule–based auto scaling [2]. Ở dạng thứ hai đặt bài toán auto scaling vào một mô hình lý thuyết hàng đợi hay mô hình lý thuyết điều khiển (có kết hợp dự báo tài nguyên) và cả mô hình dự báo chuỗi thời gian. Dạng thứ ba là auto scaling được đưa vào mô hình ứng dụng máy học, dùng các kỹ thuật học khác nhau như học giám sát, học không giám sát, học tăng cường. Tuy vậy, trong khi auto scaling dạng thứ nhất được áp dụng phổ biến trên hệ thống công nghiệp thì dạng thứ hai và thứ ba vẫn chưa có nhiều công bố về việc áp dụng. Một trong những ưu điểm của auto scaling dạng thứ nhất là đơn giản, dễ dàng cài đặt bởi khách hàng và chi phí thuật toán thấp. Vấn đề còn lại là tính hiệu quả của phương pháp chưa rõ ràng và cần phải được xem xét kỹ lưỡng. Các phương pháp auto scaling khác nhau trong dạng một sẽ có cách chọn tham số đo lường khác nhau để dùng trong thuật toán auto scaling. Tham số đo lường được chọn phổ biến là mức độ sử dụng CPU, mức độ sử dụng RAM cũng được gọi là các tham số đo lường mức thấp. Các tham số đo lường mức cao qua phép thống kê cũng có thể được dùng như cường độ tải, số lượng yêu cầu, thời gian đáp ứng dịch vụ. Trong khuôn khổ bài báo này trước hết sẽ đề xuất mô hình để đánh giá tính hiệu quả của auto scaling dạng thứ nhất, tiếp theo đề xuất sử dụng các tham số chất lượng dịch vụ là thời gian hoàn thành một yêu cầu và tỉ lệ yêu cầu không được đáp ứng, sau cùng sẽ áp dụng mô hình đánh giá để kiểm tra tính hiệu quả của phương pháp này.

Phần tiếp theo của bài báo sẽ gồm có các phần: Phần II sẽ trình bày các giải pháp auto scaling theo cách chọn tham số khác nhau đã được công bố và phương pháp đánh giá tính hiệu quả của auto scaling trong các công bố khác. Nội dung của phần III là mô hình đánh giá được đề xuất. Phần IV là thuật toán để cài đặt với tham số chất lượng dịch vụ được chọn. Thực nghiệm đánh giá qua mô phỏng máy tính được trình bày trong phần V. Bài báo được kết thúc với các các kết luận ở phần VI.

II. CÁC NGHIÊN CỨU LIÊN QUAN

Trong nỗ lực cải tiến auto scaling trên AWS cloud platform, các tác giả trong [3] đã đề xuất giải pháp dùng đại lượng đo lường được tính toán trên cơ sở áp dụng thống kê bậc q-quantile và mean trên thời gian thực thi các yêu cầu thay cho cách dùng đo lường mức sử dụng tài nguyên truyền thống. Phương pháp được thực nghiệm trên bộ công cụ của SmartBear với nhiều tham số giả sử kèm theo, đánh giá cũng cho thấy chi phí giảm so với phương pháp sử dụng đo lường mức CPU. Tuy nhiên việc mở rộng tập tải thử nghiệm cần được thực hiện để đánh giá có tính khách quan hơn.

Thay vì tìm tham số điều khiển thay thế cho giải pháp auto scaling, trong công trình [4] các tác giả đề xuất sử dụng mô hình đa mức ngưỡng bằng cách áp dụng các qui tắc linh động để điều chỉnh mức ngưỡng đáp ứng các dạng tải khác nhau trên hệ thống. Kết quả đánh giá của giải pháp dựa trên phân tích hàm CDF (cumulative distribution function) của thời gian đáp ứng có được dựa vào các mẫu lưu lượng khác nhau. Qua đó cũng chứng tỏ tính hiệu quả của giải pháp chọn ngưỡng linh hoạt. Tuy nhiên, tính phức tạp của giải pháp và chi phí về quá trình quá độ trong thay đổi ngưỡng chưa được đánh giá. Bên cạnh giải pháp dựa vào đo lường và ngưỡng, các đề xuất khác liên quan đến sử dụng công nghệ machine learning, ví dụ như trong [5] các tác giả tập trung auto scaling cho các VNF (virtual network functions) bằng cách xây dựng bộ phân loại học các quyết định điều chỉnh bổ sung trong quá khứ và các động thái của tải theo mùa để đưa ra các quyết định auto scaling trước thời hạn. Giải pháp đã khai thác các thuộc tính của công nghệ ảo hóa nhằm chi phối chất lượng dịch vụ và tiết kiệm chi phí. Tuy nhiên hiệu quả của giải pháp phụ thuộc nhiều vào tập dữ liệu ban đầu và các chi tiết

trong kỹ thuật xử lý trong mô hình học giám sát này.

III. MÔ HÌNH ĐÁNH GIÁ

Để đánh giá hiệu quả sử dụng lượng tài nguyên thuê bao gồm cả tài nguyên máy ảo được bổ sung tự động thông qua dịch vụ auto scaling, trong bài báo này định nghĩa một đại lượng được gọi là máy ảo thời gian VMxTime. Nếu thời gian được tính theo giờ thì gọi là máy ảo giờ (VMxHour), 1 VMxHour có nghĩa là 1 máy ảo được mở trong thời gian 1 giờ. Chi phí tiêu thụ tài nguyên thuê được qui về VM hour bằng cách lấy tích số giữa số máy ảo và thời gian mở tương ứng tính theo giờ. Gọi VM Usage là lượng tiêu thụ máy ảo theo đại lượng VMxTime trong thời gian T, tham số này được tính theo công thức (1) như sau:

$$\text{VM Usage} = \int_0^T \text{NumOfVMs}(t)dt \qquad (1)$$

NumOfVMs(t) là số máy ảo (VM) được mở tại thời điểm t và được tính bởi công thức (2) như sau:

$$\text{NumOfVMs}(t)=\text{currentNumofVMs}(t) + \text{numOfVMsScale}(t) \qquad (2)$$

Dễ dàng nhận thấy nếu VM Usage càng lớn thì tài nguyên của nhà cung cấp dịch vụ bị chiếm dụng càng lớn và chi phí phải trả của người dùng sẽ lớn.

Các thuê bao được bổ sung và thu hồi mỗi một lần là N máy ảo một cách tự động, ta có:

$$\text{numOfVMScale}(t)=N \times \text{scaleEvent}(t) \qquad (3)$$

trong đó numOfVMScale(t) là số máy ảo được bổ sung hay thu hồi vào thời điểm t, scaleEvent(t) lấy giá trị 1, 0 hay -1 tùy vào hàm quyết định, được gọi là D(). Trong trường hợp thuật toán auto scaling được thực hiện theo kiểu giám sát tích cực các đại lượng trạng thái thì autoscaler sẽ theo dõi các tài nguyên còn lại tại thời điểm lấy mẫu, ký hiệu là remainResource(t) và hàm xác định tài nguyên còn lại sẽ được đưa vào như là tham số của hàm quyết định D() cùng với một giá trị ngưỡng được xác lập trước. Như vậy sự kiện điều chỉnh tự động lấy giá trị từ hàm quyết định D() như công thức (4):

$$\text{scaleEvent}(t)=D(\text{controlValue}(t),\text{Threshold}) \qquad (4)$$

$$\text{controlValue}(t)= F(\text{QoS pamameter}(t)) \qquad (5)$$

Với F() là phương pháp tính toán tham số chất lượng dịch vụ. Trong bài báo này sẽ cụ thể bằng tham số tỉ lệ các yêu cầu lần đầu đệ trình vào máy ảo bị từ chối và phải đợi, được giải thích chi tiết trong phần IV.

IV. THUẬT TOÁN AUTO SCALING SỬ DỤNG THAM SỐ CHẤT LƯỢNG DỊCH VỤ

Tham số chất lượng dịch vụ thứ nhất được chọn là thời gian hoàn thành một yêu cầu trên hệ thống. Đây là tham số phản ánh tốc độ đáp ứng dịch vụ của hệ thống, xét một cách định tính thì thời gian này sẽ càng nhỏ khi năng lực tài nguyên của hệ thống càng lớn. Tuy nhiên tham số này không dùng để điều khiển trực tiếp hoạt động auto scaling nhưng sẽ dùng để kiểm tra hiệu quả của thuật toán auto scaling. Tham số thứ hai được chọn làm tham số điều khiển trực tiếp liên quan đến hiện tượng thực tế là khi các yêu cầu gửi đến vào những thời điểm mà năng lực của hệ thống không đáp ứng được sẽ phải vào trạng thái đợi.Tỉ lệ các yêu cầu không được đáp ứng lần đầu là tham số lựa chọn thứ hai. Tham số này được chọn dựa trên phát hiện có những yêu cầu không được đáp ứng ở lần đệ trình đầu tiên trên các hệ thống auto scaling dựa vào mức sử dụng CPU, khi mà mức độ sử dụng này vẫn còn dưới ngưỡng.Thông thường các yêu cầu không được đáp ứng không bị hủy luôn mà được đưa vào hàng đợi của hệ thống để tiếp tục được đệ trình nếu khoảng thời gian timeout vẫn còn. Hệ thống dịch vụ không đáp ứng một yêu cầu nào đó không chỉ vì năng lực CPU bị thiếu, mà còn bởi nhiều tài nguyên khác bị thiếu như bộ nhớ thực thi, bộ nhớ lưu trữ, băng thông của các cổng xuất nhập (I/O) và ngay cả nguyên nhân không từ tài nguyên vật lý mà đến từ tắc nghẽn do lỗi logic của phần mềm ứng dụng. Lợi ích của việc chọn tham số này là bao hàm được nhiều yếu tố, phản ảnh đầy đủ hơn về tình trạng của hệ thống dịch vụ so với trường hợp dùng tham số mức độ sử dụng CPU. Thuật toán auto scaling thay vì sử dụng ngưỡng tỉ lệ sử dụng CPU trong máy ảo sẽ dùng ngưỡng tỉ lệ yêu cầu không được phục vụ ở lần đầu đệ trình vào máy ảo và phải đợi, ở đây gọi vắn tắt là tỉ lệ đợi Waiting rate. Tham số Waiting rate được tính như sau:

$$\text{Waiting rate} = \frac{N_W}{N_T} \times 100\% \qquad (6)$$

N_w là số yêu cầu đệ trình lần đầu bị từ chối phải đợi trong khoảng thời gian lấy mẫu.

N_T là tổng số yêu cầu đệ trình vào máy ảo trong khoảng thời gian lấy mẫu.

Thuật toán auto scaling:

```
1.//Gán ngưỡng a% cho Vth (threshold value);
 Vth = a%;
2. //Gán giá trị b giây cho T1 là thời gian nhàn rỗi của máy ảo trước khi máy ảo bị hủy
T1= b;
3.//Gán giá trị c giây cho T2 là khoảng thời gian lấy mẫu để thống kê và tính toán giá trị tham số điều khiển
T2 = c;
4.//Thực hiện đồng thời trên các máy ảo được đăng ký auto scaling
Parallel:
//Với mỗi máy ảo được thuê VM thu thập và tính tỉ lệ waiting rate trong khoảng thời gian T2
timeout =T2;
Timerstart(timeout);
if (timeoutevent) then {
        Nw=count(waitingList);
        NT=count(submitRequestList);
        wRate=Nw/NT*100%;
```

```
        if (waiting rate > Vth) {
                //tạo máy ảo bổ sung vm
                vm=createVM();
                //đăng ký máy ảo với Loadbalancer
                LBRegister(vm);
                }
        }
   5.//Thực hiện song song trên tất cả các máy ảo
   Parallel:
   //với mỗi máy ảo vm được mở, giám sát thời gian nhàn rỗi
   idletime=count(finishedtime);
   if (idletime>T1) {
            //xóa máy ảo VM khỏi danh sách của Loadbalancer;
                LBDelete(vm);
           //tắt máy ảo VM;
                VMDestroy(vm);
        }
```

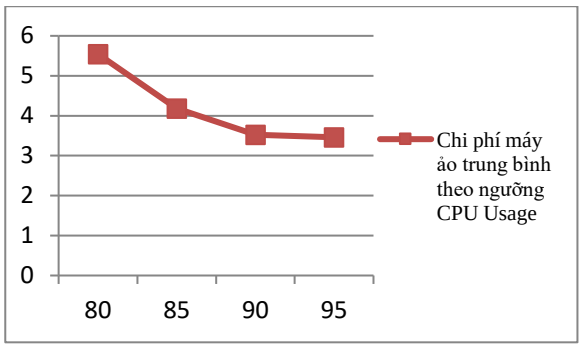
V. MÔ PHỎNG GIẢI PHÁP AUTO SCALING DỰA VÀO TỈ LỆ ĐỆ TRÌNH KHÔNG THÀNH CÔNG

Giải pháp auto scaling dựa vào tỉ lệ đệ trình yêu cầu vào máy ảo lần đầu không thành công như đã đề xuất ở trên được kiểm chứng qua mô phỏng máy tính. Chương trình mô phỏng được xây dựng bằng ngôn ngữ Java sử dụng thư viện Cloudsim [6] chạy trên máy tính Intel Core 2 Dual CPU 2GHz, bộ nhớ RAM 4GB. Mô phỏng sử dụng hệ thống điện toán đám mây có cấu hình như sau:

Data center có 50 host, mỗi host có 32 core, RAM 16GB, Storage 1TB, banwidth 10Gbps. Các host sử dụng time-shared scheduling cho máy ảo. Các máy ảo có cấu hình 1core, 1000 MIPS, 4GB RAM, 60GB Storage và 1Gbps banwidth. Các máy ảo sử dụng time-shared scheduling cho các cloudlet (yêu cầu phục vụ).

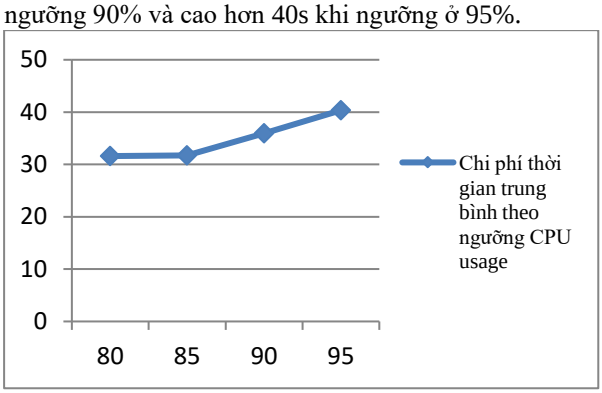
Giả sử khách hành sử dụng gói thuê bao gồm 2 máy ảo VM (virtual machine) và đăng ký sử dụng dịch vụ auto scaling. Sử dụng bộ cân bằng tải phổ biến là AMLB (Active Monitroing Load Balancer). Công việc mô phỏng sẽ dựa vào mô hình đánh giá được trình bày ở phần III, so sánh tính hiệu quả giữa giải pháp auto scaling theo waiting rate được đề xuất ở trên với auto scaling theo CPU usage đang được dùng phổ biến trên các hệ thống điện toán đám mây hiện nay.

Trước tiên, hệ thống điện toán đám mây được chạy với thuật toán auto scaling dùng tham số CPU usage. Các ngưỡng được thay thế lần lượt là 80%, 85%, 90% và 95%. Ứng với mỗi giá trị ngưỡng chi phí máy ảo trung bình hoàn thành một yêu cầu và chi phí thời gian trung bình hoàn thành một yêu cầu được thống kê và tính toán. Kết quả mô phỏng cho trường hợp này được trình bày tương ứng trên hình 1 và hình 2.



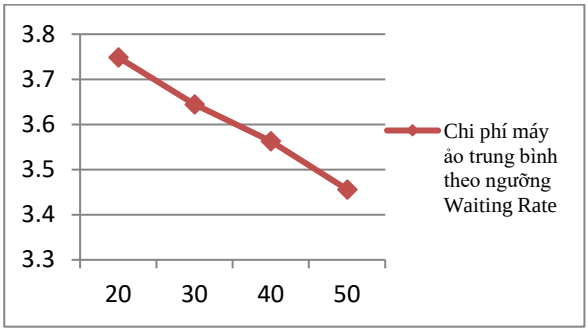
Hình 1. Chi phí máy ảo trung bình hoàn thành một yêu cầu tính theo VM.s tương ứng với các mức ngưỡng dùng CPU.

Trên hình 1 cho thấy khi sử dụng mức ngưỡng thấp 80%, chi phí máy ảo cao hơn 5,5 VM.s/yêu cầu, mức chi phí này giảm dần khi ngưỡng tăng lên. Khi ngưỡng được đặt ở 95% thì chi phí máy ảo vào khoảng 3,5 VM.s. Tuy nhiên, trong khi chi phí máy ảo cao ứng với ngưỡng thấp thì trên hình 2 cho thấy chi phí thời gian trung bình/yêu cầu lại nhỏ hơn, khoảng 31s/yêu cầu ở ngưỡng 80% so với 37s/yêu cầu ở ngưỡng 90% và cao hơn 40s khi ngưỡng ở 95%.



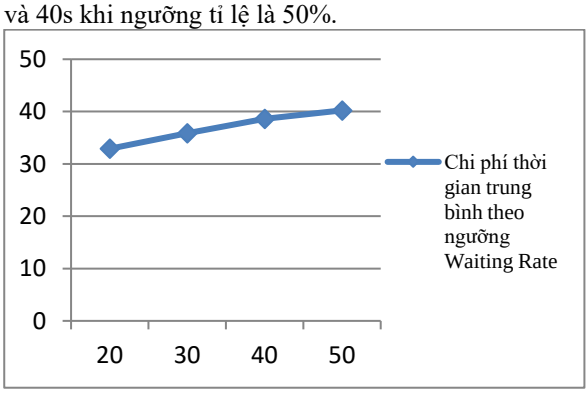
Hình 2. Chi phí thời gian trung bình hoàn thành một yêu cầu tương ứng với các mức ngưỡng dùng CPU.

Tiếp theo hệ thống điện toán đám mây được chạy với thuật toán auto scaling dùng tham số tỉ lệ yêu cầu không được tiếp nhận lần đầu và phải đợi waiting rate. Kịch bản lưu lượng ngõ vào hệ thống điện toán đám mây lần này hoàn toàn giống như lần chạy trong trường hợp dùng ngưỡng CPU uasge ở trên. Các ngưỡng tỉ lệ này được thay đổi lần lượt theo các tỉ lệ 20%, 30%, 40% và 50%. Ứng với mỗi trường hợp ngưỡng khác nhau chi phí máy ảo trung bình hoàn thành một yêu cầu và chi phí thời gian trung bình hoàn thành một yêu cầu được thống kê và tính toán. Kết quả chạy mô phỏng của trường hợp này được trình bày trên hình 3 và hình 4. Hình 3 là mô tả chi phí máy ảo trung bình VM.s/yêu cầu theo các ngưỡng tỉ lệ waiting rate và hình 4 là chi phí thời gian trung bình/yêu cầu theo các ngưỡng tỉ lệ waiting rate.



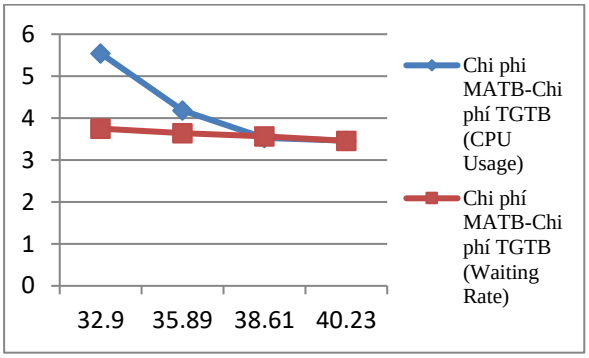
Hình 3. Chi phí máy ảo trung bình hoàn thành một yêu cầu tính theo VM.s tương ứng với các mức ngưỡng dùng waiting rate.

Hình 3 cho thấy với ngưỡng tỉ lệ thấp 20% làm chi phí máy ảo ở mức cao 3.75 VM.s và chi phí giảm dần khi ngưỡng tỉ lệ tăng lên, mỗi yêu cầu tiêu tốn trung bình là 3.45 VM.s khi ngưỡng tỉ lệ được đặt ở 50%. Tuy nhiên, trên hình 4 thì chi phí thời gian trung bình/yêu cầu tăng lên khi ngưỡng tỉ lệ tăng, ở mức 34 s ứng với ngưỡng tỉ lệ 20% và 40s khi ngưỡng tỉ lệ là 50%.



Hình 4. Chi phí thời gian trung bình hoàn thành một yêu cầu tương ứng với các mức ngưỡng dùng waiting rate.

Sau cùng, hình 5 là nhằm so sánh giữa auto scaling dựa vào CPU usage và auto scaling dựa vào waiting rate được đề xuất. Trục tung biểu thị chi phí máy ảo trung bình/yêu cầu và trục hoành là chi phí thời gian trung bình/yêu cầu. Theo đó, với chi phí thời gian trung bình ở mức thấp thì trường hợp dùng waiting rate có chi phí máy ảo thấp hơn nhiều, khoảng 3.7 VM.s so với 5.6 VM.s của trường hợp dùng CPU usage. Chỉ khi chi phí thời gian trung bình/yêu cầu lớn hơn 38.6s thì mức chi phí máy ảo của cả hai phương pháp mới xấp xỉ nhau ,vào khoảng 3.5 VM.s. Điều này chứng tỏ giải pháp auto scaling dùng waiting rate có thể giảm đồng thời cả hai chí phí so với auto scaling dùng CPU usage truyền thống.



Hình 5. Quan hệ giữa chi phí máy ảo trung bình (MATB) và chi phí thời gian trung bình (TGTB) trên một yêu cầu của giải pháp dùng CPU usage và Waiting rate.

VI. KẾT LUẬN

Phương pháp auto scaling sử dụng tỉ lệ giữa yêu cầu không được đáp ứng lần đầu và tổng số yêu cầu được đệ trình đã được trình bày. Phương pháp này được xem như một trong số các phương pháp theo xu hướng custom-metric auto scaling. Với cách dùng tỉ lệ này đã bao hàm tất cả các yếu tố phản ánh năng lực của máy ảo và của hệ thống thuê bao trên điện toán đám mây vào thời điểm xem xét, nhờ đó phản hồi tình trạng máy ảo thực tế hơn, giúp đưa ra quyết định chính xác cho auto scaling. Kết quả mô phỏng dựa vào mô hình đánh giá cho thấy phương pháp auto scaling được đề xuất đã giảm đồng thời cả chi phí máy ảo và chi phí thời gian so với auto scaling dựa vào mức độ sử dụng CPU đang được dùng phổ biến trên các hệ thống điện toán đám mây hiện nay. Điều này đồng nghĩa với chi phí được giảm thiểu nhưng chất lượng dịch vụ vẫn được đảm bảo khi thực hiện auto scaling dùng tỉ lệ đặc biệt này. Nếu các hệ thống giám sát trung tâm của hệ thống điện toán đám mây có cung cấp tỉ lệ này thì người dùng hoàn toàn có thể áp dụng phương pháp cho hệ thống thuê bao của mình một cách nhanh chóng và hiệu quả.

TÀI LIỆU THAM KHẢO

[1] Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing, NIST Special Publication 800-145, September 2011.

[2] Lorido-Botran, T., Miguel-Alonso, J. & Lozano, J.A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. J Grid Computing 12, 559–592 (2014). https://doi.org/10.1007/s10723-014-9314-7

[3]Dariusz Rafal Augustyn, Lukasz Warchal, Metrics-Based Auto Scaling Module for Amazon Web Services Cloud Platform, 2017

[4] Salman Taherizadeh and Vlado Stankovski, Dynamic Multi-level Auto-scaling Rules for Containerized Applications, Computer And Communications Networks and Systems The Computer Journal, Vol. 62 No. 2, 2019

[5] Sabidur Rahman, et al. Auto-Scaling Network Resources using Machine Learning to Improve QoS and Reduce Cost, Networking and Internet Architecture, arXiv:1808.02975v2 [cs.NI], 2019.

[6] CloudSim 3.0 API, The Cloud Computing and Distributed Systems(CLOUDS) Laboratory, The University of Melbourne,available: http://www.cloudbus.org/cloudsim/doc/api/index.html

[7] https://kubernetes.io/

AUTO SCALING BASED ON QUALITY OF SERVICE PARAMETERS IN CLOUD COMPUTING

Abstract: The provision of auto scaling service on the infrastructure of cloud service providers has helped subscribers avoid service degradation and save costs. Along with that is the emergence of auto scaling algorithms to automate the provision and return resources in a timely manner.The auto scaling method, which is based on the rule of matching current parameter values with thresholds, is commonly used in industrial environments. The effectiveness of the auto scaling method depends on how the parameter is selected. In this paper I propose to use quality of service parameters instead of a commonly used CPU usage parameter. The paper also provides a model to evaluate the effectiveness of this type of auto scaling and based on that to test the effectiveness of the method. The model has been installed and effectiveness has been determined according to the choice of quality of service parameters.

Nguyen Hong Son received his B.Sc. in Computer Engineering from Ho Chi Minh City University of Technology, his M.Sc. and PhD in Communication Engineering from the Post and Telecommunication Institute of Technology Hanoi. His current research interests include communication engineering, machine learning, data science, network security, and cloud computing