Kiểm soát đầu vào để lập lịch cho các yêu cầu người dùng trên tính toán đám mây dựa vào ràng buộc QoS

Admission Control to Schedule for User Requirements Based on QoS Constraints in Cloud Computing

Nguyễn Hoàng Hà, Lê Văn Sơn, Nguyễn Mậu Hân

Abstract: *The problem of admission control to schedule for user requirements is NP-complete [1] in cloud computing environment. To solve this problem it is usually to put building heuristic algorithms to form a simple algorithm with complex polynomial. In this paper, we propose an algorithm of admission control and a scheduling algorithm for user requirements based on the use of ACO algorithm (Ant Colony Optimization) and take advantage of validity period between the requirements so that the total cost of the system is minimal but still satisfying QoS (Quality of Service) constraints for the requirements. Two algorithms are set up and run a complete test on CloudSim. The experimental results show the effectiveness and superiority of the proposed algorithm in comparing with sequential and EDF (Earliest Deadline First) algorithms.*

Keyword: *Admission Control, Scheduling Algorithms, QoS Constraint, Resource Allocation.*

I. GIỚI THIỆU

Tính toán đám mây là sự phát triển của tính toán phân tán, tính toán song song và tính toán lưới. Tài nguyên trong môi trường này được cung cấp bởi nhiều nhà cung cấp dịch vụ như, Microsoft Azure [2], IBM [3], Amazon EC2 [4] v.v.. Các nhà cung cấp này cung cấp các dịch vụ cho người dùng bằng cách cho người dùng thuê tài nguyên (phần cứng, phần mềm, tài nguyên lưu trữ, v.v..) thông qua Internet. Người dùng có thể thuê các tài nguyên khác nhau dựa trên yêu cầu của họ và trả chi phí khi họ sử dụng. Thời gian thuê tài nguyên được tính theo giờ, nếu người dùng không sử dụng hết một giờ thì họ cũng phải trả chi phí cho toàn bộ một giờ được thuê. Điều này thúc đẩy nhu cầu tìm kiếm một định vị hiệu quả về chi phí cho tập các yêu cầu của khách hàng.

Tính toán đám mây coi phần mềm (SaaS) và cơ sở hạ tầng (IaaS) như là các dịch vụ. Mục tiêu chính của nhà cung cấp SaaS (Software as a Service) là đem lại lợi nhuận lớn nhất cho họ bằng cách thuê các tài nguyên với chi phí thấp từ nhà cung cấp IaaS (Infrastructure as a Service) nhưng vẫn đảm bảo ràng buộc QoS cho khách hàng. Để đạt được mục tiêu của nhà cung cấp SaaS, bài báo này đề xuất thuật toán vừa kiểm soát đầu vào vừa lập lịch ACACO và thuật toán lập lịch MProfit. Thuật toán ACACO sử dụng ACO (Ant Colony Optimization) [6,8] để tìm kiếm tài nguyên trên các trung tâm dữ liệu với chi phí thấp nhưng vẫn thỏa mãn ràng buộc QoS, sau đó ra quyết định chập nhận hay từ chối yêu cầu của khách hàng. Nếu yêu cầu được chấp nhận thì yêu cầu này sẽ được ánh xạ vào tài nguyên hợp lý. Thuật toán MProfit tiếp tục lập lịch cho các yêu cầu được chấp nhận để tận dụng khoảng thời gian sử dụng chưa hết trong giờ được thuê của các yêu cầu nhằm đem lại chi phí nhỏ nhất cho nhà cung cấp SaaS.

Bài toán kiểm soát đầu vào và lập lịch cho các yêu cầu với các tham số như thời gian đến, deadline, ngân sách, khối lượng công việc, tỉ lệ phạt, v.v. là một bài toán NP-đầy đủ [1]. Do đó, để đưa ra một giải pháp tối ưu thường phải tìm kiếm vét cạn khi đó độ phức tạp sẽ là hàm mũ, nên cách này không thể được áp dụng. Để khắc phục nhược điểm này, người ta thường dùng các phương pháp heuristic để đưa ra một giải pháp gần tối

ưu như phương pháp tối ưu hóa đàn kiến (ACO) [6, 8], kỹ thuật tối ưu hóa đàn ong mờ [9], phương pháp tham lam EDF [10,11], v.v..

Trong môi trường tính toán đám mây, người sử dụng thuê các dịch vụ thông qua Internet và trả phí khi sử dụng. Do đó, các thuật toán lập lịch dựa vào ràng buộc QoS thường được sử dụng. Trong trường hợp này, các tham số của người dùng như thời gian, phí dịch vụ cho người sử dụng, phí dịch vụ cho nhà cung cấp, độ tin cậy, v.v., được ưu tiên xem xét khi lập lịch. Jzau-Sheng Lin và các đồng nghiệp [12] đưa ra mô hình lập lịch cho các yêu cầu trên môi trường tính toán đám mây với mục tiêu đem lại lợi nhuận cao nhất cho nhà cung cấp dịch vụ nhưng chỉ xem xét đến hai tham số ngân sách và deadline của các yêu cầu.

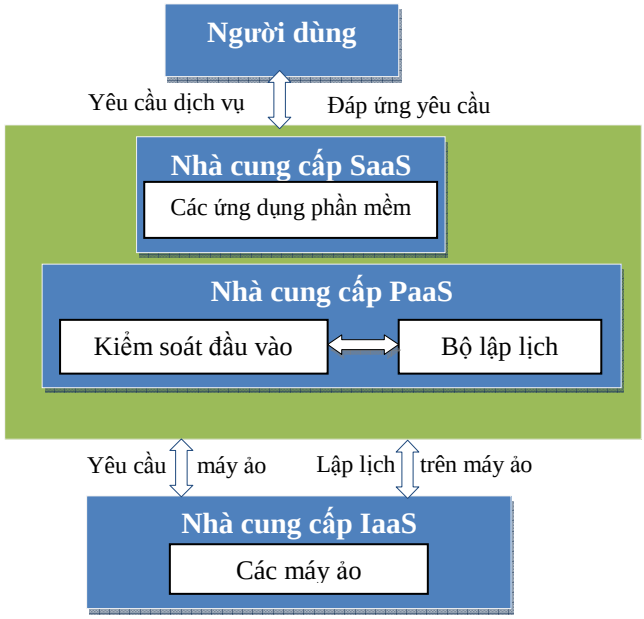
Các nghiên cứu [13,14] lại tập trung vào lập lịch trên các yêu cầu để tiết kiệm điện năng trên các trung tâm dữ liệu. Các nghiên cứu gần đây của Ramkumar N [15] đã lập lịch trên các yêu cầu thời gian thực, sử dụng hàng đợi ưu tiên để ánh xạ yêu cầu vào tài nguyên nhưng chỉ tập trung lập lịch để giải quyết công việc một cách nhanh nhất thỏa mãn deadline của yêu cầu mà không quan tâm đến chi phí và ngân sách. Swarupa Irugurala [16] đưa ra thuật toán lập lịch với mục tiêu đem lại lợi nhuận cao nhất cho nhà cung cấp SaaS nhưng chỉ xem xét giữa hai loại chi phí: chi phí khởi tạo máy ảo và chi phí sử dụng máy ảo đã có để chọn tài nguyên.

Trong bài báo này, chúng tôi đưa ra thuật toán lập lịch cho các yêu cầu với các ràng buộc QoS như chi phí, deadline, ngân sách, khối lượng, tỉ lệ lãi suất phạt, kích cỡ file đầu vào và đầu ra. Sử dụng các máy ảo đã có trên các trung tâm dữ liệu để ánh xạ vào các yêu cầu nhằm mục tiêu làm cho chi phí của hệ thống là nhỏ nhất nhưng vẫn thỏa mãn deadline và ngân sách của các yêu cầu.

Phần còn lại của bài báo bao gồm: Phần II xây dựng mô hình hệ thống, Phần III xây dựng bài toán và đưa ra hai thuật toán ACACO và MProfit, sau đó mô phỏng và đánh giá giữa các thuật toán và Phần IV là kết luận.

II. MÔ HÌNH HỆ THỐNG

Các thành phần của hệ thống bao gồm: người dùng, nhà cung cấp SaaS, PaaS (Platform as a Service) và IaaS. Người dùng gửi các yêu cầu sử dụng phần mềm kèm theo yêu cầu QoS của họ lên nhà cung cấp SaaS. Tại đây, nhà cung cấp PaaS sử dụng thành phần kiểm soát đầu vào để phân tích các tham số QoS và dựa vào khả năng, tính sẵn sàng và giá của máy ảo để quyết định chấp nhận hoặc từ chối các yêu cầu của người dùng. Nếu yêu cầu được chấp nhận thì thành phần lập lịch chịu trách nhiệm để định vị các tài nguyên cho các yêu cầu của người dùng như mô hình trình bày ở Hình 1. Mô hình này bao gồm bốn mô hình thành phần: mô hình người dùng, nhà cung cấp SaaS, IaaS và PaaS. Bài báo này tập trung vào mô hình của nhà cung cấp PaaS.



***Hình 1**. Mô hình tổng quát của các thành phần trong tính toán đám mây.*

II.1. Mô hình người dùng

Người dùng gửi N yêu cầu dịch vụ $(t_1,t_2, \ldots, t_N\}$ đến nhà cung cấp SaaS, mỗi yêu cầu $t_i(a_i,d_i,b_i, \alpha_i,w_i,in_i,out_i)$ bao gồm các thuộc tính và các ràng buộc QoS như sau:

- a_i: Thời gian đến của yêu cầu
- Thời hạn d_i (deadline): Thời gian lớn nhất người

dùng cần để đợi kết quả

- Ngân sách b_i (budget): Chi phí lớn nhất người dùng sẽ trả cho nhà cung cấp dịch vụ

- Tỉ lệ lãi suất phạt (α_i): Một tỷ lệ bồi thường cho người dùng nếu nhà cung cấp SaaS không cung cấp đúng thời hạn

- Khối lượng công việc w_i: Bao nhiêu MI (triệu chỉ thị) được yêu cầu để thực hiện cho yêu cầu.

- Kích cỡ file đầu vào và đầu ra của yêu cầu: in_i và out_i

II.2. Mô hình nhà cung cấp SaaS

Nhà cung cấp SaaS thuê các tài nguyên từ các nhà cung cấp IaaS và nó cho thuê phần mềm như là các dịch vụ cho người dùng. Mục tiêu của các nhà cung cấp SaaS là giảm thiểu chi phí sử dụng tài nguyên từ các nhà cung cấp IaaS để đem lại lợi nhuận cao nhất cho mình.

II.3. Mô hình nhà cung cấp IaaS

Trong môi trường tính toán đám mây có Y nhà cung cấp tài nguyên IaaS $\{x_1,x_{2,\ldots,}x_Y\}$. Mỗi nhà cung cấp IaaS cung cấp M máy ảo $\{vm_1,vm_2,\ldots,vm_M\}$ cho các nhà cung cấp SaaS và chịu trách nhiệm để điều phối các máy ảo chạy trên các tài nguyên vật lý của chúng. Mỗi máy ảo $vm_{jx}(t_{jx},p_{jx},s_{jx},dtp_{jx,}dts_{jx})$ của nhà cung cấp x bao gồm các thuộc tính:

- Thời gian khởi tạo t_{jx}: Thời gian cần thiết để triển khai một máy ảo.

- Giá p_{jx}: Giá tính theo giờ mà nhà cung cấp SaaS phải trả cho nhà cung cấp IaaS khi sử dụng máy ảo.

- s_{jx}: Tốc độ xử lý của máy ảo tính theo MIPS

- dtp_{jx}: Giá tính theo dung lượng truyền của nhà cung cấp SaaS phải trả để chuyển dữ liệu từ nhà cung cấp tài nguyên đến máy địa phương.

- dts_{jx}: Tốc độ chuyển dữ liệu, phụ thuộc vào khoảng cách và hiệu năng mạng.

II.4. Mô hình nhà cung cấp PaaS.

Tất cả các nhà cung cấp tài nguyên IaaS không liên quan với nhau và có thể thực hiện song song. Các nhà cung cấp này được ký hiệu là R. Chúng ta lập lịch cho N yêu cầu độc lập không theo thứ tự ưu tiên (non-preemptive) trên Y nhà cung cấp, các yêu cầu này được ký hiệu là npmtn. Mục đích của chúng ta là tìm ra chi phí thấp nhất để hoàn thành tất cả các yêu cầu nhưng vẫn thỏa mãn thời hạn và ngân sách của các yêu cầu, nghĩa là ta phải tìm ra C_{min}. Vì vậy mô hình của nhà cung cấp PaaS là R| npmtn|C_{min}

Gọi C_{ijx} là chi phí để xử lý yêu cầu i trên máy ảo j của nhà cung cấp tài nguyên x. Khi đó, chi phí C_{ijx} bao gồm các loại chi phí:

- Chi phí xử lý yêu cầu (CP_{ijx}) phụ thuộc vào giá p_{jx}, tốc độ s_{jx} trên máy ảo j của nhà cung cấp tài nguyên x và khối lượng xử lý của yêu cầu w_i:

$$CP_{ijx} = p_{jx} * \frac{w_i}{s_{jx}} \quad (1)$$

- Chi phí truyền tải dữ liệu (CTD_{ijx}) bao gồm chi phí gửi dữ liệu lên và lấy dữ liệu về từ nhà cung cấp tài nguyên phụ thuộc vào kích cỡ file đầu vào in_i và kích cỡ file đầu ra out_i của yêu cầu i, tốc độ chuyển dữ liệu Dts_{jx} và giá để chuyển dữ liệu Dtp_{jx} từ máy ảo j của nhà cung cấp tài nguyên x đến máy máy yêu cầu dịch vụ:

$$CTD_{ijx} = Dtp_{jx} * \frac{in_i + out_i}{Dts_{jx}} \quad (2)$$

- Chi phí khởi tạo máy ảo (CI_{ijx}) phụ thuộc vào thời gian khởi tạo t_{jx} và giá p_{jx} trên máy ảo j của nhà cung cấp tài nguyên x:

$$CI_{ijx} = t_{jx} * p_{jx} \quad (3)$$

- Chi phí nhà cung cấp SaaS phải trả lại cho người nếu không cung cấp yêu cầu đúng thời hạn (CR_{ijx}), phụ thuộc vào tỉ lệ lãi suất phạt (α_i) và khoảng thời gian vượt thời hạn β_{ijx}:

$$CR_{ijx} = \alpha_i * \beta_{ijx} \quad (4)$$

Gọi T_{ijx} là thời gian để xử lý yêu cầu i trên máy ảo j của nhà cung cấp tài nguyên x. Khi đó T_{ijx} bao gồm các loại thời gian:

$$T_{ijx} = \frac{w_i}{s_{jx}} + \frac{in_i + out_i}{Dts_{jx}} + t_{ijx} + \beta_{ijx} \quad (5)$$

Trong đó:

- $\frac{w_i}{s_{jx}}$: Thời gian xử lý yêu cầu
- $\frac{in_i+out_i}{Dts_{jx}}$: Thời gian truyền tải dữ liệu
- t_{ijx} : Thời gian khởi tạo máy ảo
- β_{ijx}: Khoảng thời gian vượt thời hạn

Mục tiêu của các nhà cung cấp SaaS là đem lại lợi nhuận cao nhất. Do đó, ta phải tìm chi phí nhỏ nhất của tất cả các yêu cầu như công thức (6):

$$\underset{i=1..N}{Max}\left(\sum_{x=1}^{Y}\sum_{j=1}^{M}(b_i - C_{ijx})\right) \quad (6)$$

Trong đó:

- Chi phí của yêu cầu i phải thỏa mãn ngân sách của nó tức là:

$$C_{ijx} \le b_i \quad (7)$$

- Thời gian xử lý yêu cầu i phải thỏa mãn ràng buộc:

$$T_{ijx} \le d_i + \beta_{ijx} \quad (8)$$

Như vậy, để đạt được mục tiêu đề ra (6) thì phải thỏa mãn hai ràng buộc (7) và (8).

III. XÂY DỰNG THUẬT TOÁN

Thuật toán đàn kiến dựa trên hành vi của đàn kiến thực được Marco Dorigo giới thiệu vào năm 1996. Nó là một thuật toán sử dụng heuristic và là giải pháp cho các bài toán tối ưu hóa tổ hợp. Thuật toán đàn kiến mô phỏng hành vi của đàn kiến trong tự nhiên nhằm tìm kiếm đường đi ngắn nhất giữa tổ kiến và nguồn thức ăn dựa trên mật độ mùi (pheromone) mà các con kiến để lại trên đường đi. Khi một con kiến tìm kiếm thức ăn, nó sẽ để lại một số lượng mùi trên đường đi. Nếu một con kiến cố gắng để di chuyển từ nơi này sang nơi khác, nó sẽ gặp một dấu vết trước đó đã đặt, kiến có thể phát hiện các dấu vết mùi và nó quyết định chọn đường đi có mật độ mùi cao để đi. Sau khi đi qua con kiến này cũng để lại mùi trên đường và mùi này sẽ mất dần theo thời gian. Do đó, mùi trên con đường ngắn hơn sẽ được tăng lên một cách nhanh chóng. Số lượng mùi trên mỗi con đường sẽ ảnh hưởng đến khả năng con kiến khác để lựa chọn đường đi. Cuối cùng, tất cả những con kiến sẽ chọn con đường ngắn nhất. Để áp dụng thuật toán đàn kiến người ta phải xác định được các thông tin: hàm cực tiểu F, thông tin heuristic η_i, cập nhật lại mùi và xác xuất P [6, 7].

III.1. Hàm cực tiểu F và thông tin heuristic η_i.

Hàm cực tiểu F và thông tin heuristic η_i được sử dụng để tìm ra nhà cung cấp IaaS tốt nhất, phụ thuộc vào chi phí thực hiện (C_{jx}) trên máy ảo j của nhà cung cấp x như sau:

$$F = Max(C_{jx}), j = 1..M, x = 1..Y \quad (9)$$

$$\eta_i = \frac{1}{C_{jx}}, i = 1..N, j = 1..M, x = 1..Y \quad (10)$$

Sử dụng η_i để tìm ra máy ảo j của nhà cung cấp x có độ ưu tiên cao nhất vì chi phí C_{jx} càng nhỏ thì thông tin η_i của yêu cầu i càng cao. Hàm cực tiểu F dùng để tính xác xuất cho yêu cầu i chọn máy ảo j của nhà cung cấp x.

III.2. Cập nhật lại mùi

Mỗi con kiến bắt đầu ngẫu nhiên từ một nhà cung cấp tài nguyên IaaS. Tại mỗi bước lặp các con kiến tính hàm cực tiểu và cập nhật lại mùi (pheromone) như sau:

$$\tau_{ijx} = \rho\tau_{ijx} + \Delta\tau_{ijx} \quad (11)$$

Trong đó:

- $\Delta\tau_{ijx} = \frac{1-\rho}{F_k}$: F_k là hàm cực tiểu của con kiến k, F_k càng nhỏ thì mật độ mùi để lại càng cao.
- τ_{ijx} : Mật độ mùi của yêu cầu i trên máy ảo j của nhà cung cấp x
- $\Delta\tau_{ijx}$: Được thêm vào mật độ mùi
- ρ: Giá trị bay hơi được xác định trong khoảng (0,1)

III.3. Tính xác xuất

Các thuật toán kiểm soát đầu vào được duy trì hai tập yêu cầu. Một tập các yêu cầu đang xử lý và tập khác đi đến và chưa được xử lý. Thuật toán được bắt đầu một cách tự động khi tập các yêu cầu đã xử lý xong và đã chuyển vào thành phần lập lịch. Theo [17] yêu cầu đầu tiên được thực hiện và nó chọn nhà cung

cấp một cách ngẫu nhiên. Yêu cầu kế tiếp được thực hiện và chọn nhà cung cấp tiếp theo với xác suất [17]:

$$P_{ijx} = \frac{\tau_{ijx}\eta_{ijx}}{\sum_{x=1}^{Y}\sum_{j=1}^{M}\tau_{ijx}\eta_{ijx}} \quad (12)$$

Trong đó: η_{ijx}: thông tin heuristic, τ_{ijx}: mật độ mùi để lại khi di chuyển.

P_{ijx} là xác suất để yêu cầu i chọn máy ảo j của nhà cung cấp tài nguyên x phụ thuộc vào sự kết hợp của hai giá trị η_{ijx} và τ_{ijx} được xác định:

$$P_{ijx} = \frac{\left(\rho\tau_{ijx}+\frac{1-\rho}{F_k}\right).\frac{1}{CP_{ijx}+CTD_{ijx}+CI_{ijx}+CR_{ijx}}}{\sum_{x=1}^{Y}\sum_{j=1}^{M}\left(\rho\tau_{ijx}+\frac{1-\rho}{F_k}\right).\frac{1}{CP_{ijx}+CTD_{ijx}+CI_{ijx}+CR_{ijx}}} \quad (13)$$

III.4. Thuật toán ACACO

Đầu vào của thuật toán:

- Khởi tạo giá trị bốc hơi của mùi (pheromone evaporation) ban đầu của ρ là 0,05.

- Giá trị mùi ban đầu (Pheromone deposit) là 0,01.

- Số kiến (k) được sử dụng trong thuật toán đề xuất là 4.

- T={t_1,t_2,…, t_N}: Tập các yêu cầu người dùng gửi đến, mỗi yêu cầu t_i là một bộ 7 <a_i,d_i,b_i,α_i,w_i,in_i, out_i>

- X={ x_1,x_2,…, x_Y },VM_x={ vm_{1x},vm_{2x},…, vm_{Mx} }: tập các nhà cung cấp IaaS và tập các máy ảo của mỗi nhà cung cấp, mỗi máy ảo vm_{jx} là một bộ 5 vm_{jx}(t_{jx},p_{jx},s_{jx},dtp_{jx},dts_{jx}).

Đầu ra của thuật toán: Một lịch trình S chứa tập các ánh xạ của các yêu cầu đã được chấp nhận bởi nhà cung cấp SaaS vào các máy ảo của các nhà cung cấp IaaS.

Mô tả thuật toán:

```
1.  FUNCTION ACACO()
2.      S={};
3.      FOR EACH t_i in T DO
4.          s_i= Test(t_i,X,VM_x);
5.          If(s_i=={})
6.              Thông báo cho người dùng yêu cầu đã
                bị từ chối ;
7.          Else
8.            S=S+{s_i};
9.      END FOR
10.     Return S;
11.   END FUNCTION
```

FUNCTION Test($t_i \in T$,X,VM_x)

12. **FOR EACH con kiến thứ k DO**
13. **FOR EACH nhà cung cấp x in X DO**
14. Tính thông tin heuristic cho yêu cầu t_i trên các máy ảo vm_{jx}

$$\eta_{ijx} = \frac{1}{CP_{ijx} + CTD_{ijx} + CI_{ijx} + CR_{ijx}}$$

15. Tìm ra giá trị của mùi hiện tại: τ_{ijx}
    Tính thời gian thực hiện của t_i trên các máy ảo vm_{jx} như công thức (5)

$$T_{ijx} = \frac{w_i}{s_{jx}} + \frac{in_i + out_i}{Dts_{jx}} + t_{ijx} + \beta_{ijx}$$

16. Cập nhật lại mùi như công thức (11):

$$\tau_{ijx} = \rho\tau_{ijx} + \frac{1-\rho}{F_k}$$

17. Tính xác suất để yêu cầu t_i ánh xạ vào các máy ảo vm_{jx} như công thức (13)
18. **END FOR**
19. **END FOR**
20. Từ xác xuất tính được trên các máy ảo vm_{jx}, tìm ra máy ảo có xác xuất lớn nhất, có chi phí nhỏ hơn ngân sách b_i và thời gian thực hiện nhỏ hơn hoặc bằng $d_i + \beta_{ijx}$ nếu tìm thấy thì ta có được phương án tối ưu s_i={$t_i \rightarrow vm_{jx}$}; return s_i. Ngược lại, nếu không tìm thấy thì return {};

END FUNCTION

III.5. Thuật toán lập lịch MProfit

Sau khi thực hiện thuật toán ACACO, ta có được tập các ánh xạ của các yêu cầu được chấp nhận vào các máy ảo. Mỗi nhà cung cấp x có thể chấp nhận được nhiều yêu cầu, thuật toán tận dụng khoảng thời gian còn hiệu lực trong vòng một giờ được thuê của các yêu cầu nằm trong cùng một nhà cung cấp để đem lại lợi nhuận cao nhất cho nhà cung cấp SaaS.

Chúng tôi gọi khoảng thời gian còn hiệu lực trong vòng một giờ được thuê là thời gian gối đầu của hai yêu cầu. Định nghĩa tập T_i bao gồm các yêu cầu trong cùng một nhà cung cấp với yêu cầu t_i và gối đầu lên yêu cầu t_i, các yêu cầu này có thể chia sẻ cùng một máy ảo:

$$T_i = \{t_l | d_l \geq d_i \text{ và } a_l < d_i\} \quad (14)$$

Sau khi xác định được tập T_i, tiến hành tính khoảng thời gian gối đầu lên nhau. Ta định nghĩa t_{iljx} là thời gian còn hiệu lực để tính toán cho yêu cầu t_l sau khi đã thực hiện xong yêu cầu t_i trên máy ảo j của nhà cung cấp x. Giá trị t_{ijlx} phụ thuộc vào tốc độ của các máy ảo, thời gian đến, deadline và khối lượng công việc của t_i và t_l. T_{iljx} được tính như sau:

$$t_{iljx} = \begin{cases} \min(D - U_{il}, d_l - a_l) & Nếu\ a_l - a_i \geq \frac{w_i}{s_{jx}} \\ D - U_{il} & Nếu\ a_l - a_i < \frac{w_i}{s_{jx}}\ và\ d_l - a_i \geq D \\ d_l - (a_i + U_{il}) & Nếu\ a_l - a_i < \frac{w_i}{s_{jx}}\ và\ d_l - a_i < D \end{cases} \quad (15)$$

Trong đó, D là thời gian thuê nhỏ nhất tính theo giờ, $U_{il} = \frac{w_i}{s_{jx}} + \max(a_l - d_i, 0)$, s_{jx} là tốc độ của máy ảo được ánh xạ vào yêu cầu t_i

Ví dụ: Giả sử D=60, máy ảo vm_{21} của nhà cung cấp 1 có tốc độ s_{21}=20 và yêu cầu t_1 có thời gian đến: 0, deadline: 30, khối lượng: 500 thực hiện trên máy ảo vm_{21} với thời gian: 25 phút. Yêu cầu t_2 có thời gian đến: 30, deadline: 50, khối lượng: 1200, vì $a_l - a_i$=30>$\frac{w_1}{s_{jx}}$=25 nên rơi vào trường hợp đầu tiên của công thức (15). Khi đó U_{il}=25+max(30-30,0)=25 và $min(D - U_{il}, d_l - a_j)$ = min(60-25,50-30)=20, như vậy thời gian còn hiệu lực của máy ảo vm_{21} cho yêu cầu t_2 là 60-25=35 phút, nhưng t_2 chỉ sử dụng được min(35,20)=20 phút. Hai trường hợp còn lại của công thức (15) xét tương tự.

Thuật toán MProfit

Đầu vào:

ST={}: chứa tập ánh xạ của các yêu cầu vào các máy ảo của các nhà cung cấp.

UST = S: trong đó S là lịch trình được tạo ra bởi thuật toán ACACO. Mỗi phần tử trong S là một ánh xạ: $t_i \rightarrow vm_{jx}$, do đó trong thuật toán này chúng tôi gọi ánh xạ t_i thay vì gọi yêu cầu t_i

Đầu ra: Một lịch trình tối ưu ST để ánh xạ các yêu cầu vào các máy ảo

Mô tả thuật toán:

1. Sắp xếp các ánh xạ trong UST theo nhà cung cấp, khi đó các ánh xạ trong cùng 1 nhà cung cấp sẽ nằm trong một nhóm.
2. **FOR EACH nhà cung cấp x in UST DO**
3. PUSH(t_i); *// t_i là ánh xạ đầu tiên của nhà cung cấp x*
4. ST=ST+{t_i}; UST=UST-{t_i};
5. **FOR EACH ánh xạ t_i trong nhà cung cấp x DO**
6. t_i=POP(); *// Lấy ánh xạ t_i từ Stack*
7. Tìm T_i: $T_i = \left\{ t_l \middle| \begin{matrix} d_l \geq d_i\ và\ a_l < d_i \\ và\ t_l\ nằm\ cùng\ một\ nhóm\ với\ t_i \end{matrix} \right\}$
8. Tính t_{iljx} của các ánh xạ trong T_i như công thức (15), t_{iljx} được tính trên máy ảo j của nhà cung cấp x được ánh xạ bởi t_i.
9. Tính max(t_{iljx}), chọn t_l có thời gian gối đầu lớn nhất làm ánh xạ tiếp theo.
10. Dựa vào max(t_{iljx}) để tính lại w_l và cập nhật lại các trạng thái cho t_l.
11. PUSH(t_l);
12. ST=ST+{t_l}; UST=UST-{t_l};
13. **END FOR**
14. **END FOR**
15. Dựa vào ST để đưa ra lịch trình ánh xạ các yêu cầu vào các tài nguyên

III.6. Cơ sở lý luận của hai thuật toán.

- Thomas Stützle, Marco Dorigo [7] đã chứng minh tính hội tụ của thuật toán ACO điều này đảm bảo tính hội tụ của thuật toán ACACO đề xuất.

- Thuật toán ACACO ánh xạ yêu cầu i vào máy ảo j của nhà cung cấp x (vm_{jx}) dựa vào xác suất P_{ijx} như công thức (13). Khi chi phí C_{ijx} càng bé thì hàm cực tiểu F_k và thông tin heuristic η_i càng lớn, điều này dẫn đến mùi để lại và xác xuất chọn máy ảo vm_{jx} càng lớn. Do đó, khi ánh xạ yêu cầu vào máy ảo có chi phí thấp sẽ làm cho tổng chi phí của toàn bộ hệ thống giảm xuống.

- Đối với thuật toán MProfit, sự hình thành của tập UST chắc chắn rằng chỉ có giới hạn vì tập UST nhận các yêu cầu từ thuật toán ACACO. Thuật toán ACACO tiếp nhận các yêu cầu theo lô và theo một chu kỳ tức là ta sử dụng hai tập yêu cầu, cứ tập yêu cầu này đang được tiếp nhận để xử lý thì tập các yêu cầu

kia tiếp tục nhận yêu cầu và lưu vào hàng đợi. Khi tập các yêu cầu này xử lý xong thì hệ thống sẽ xử lý cho tập các yêu cầu lưu ở hàng đợi và quá trình cứ lặp lại.

- Để đảm bảo cho lợi ích của nhà cung cấp dịch vụ SaaS, thuật toán ACACO chấp nhận hoặc từ chối các yêu cầu của khách hàng. Trong các yêu cầu đã chấp nhận thì có thể có nhiều yêu cầu thực hiện không hết thời gian thuê. Do đó, thuật toán MProfit tận dụng khoảng thời gian gian còn hiệu lực này để thực hiện cho yêu cầu tiếp the. Điều này dẫn đến chi phí thực hiện cho cả hệ thống giảm xuống theo đúng mục tiêu đặt ra ở công thức (6).

III.7. Mô phỏng và đánh giá thuật toán

Các thuật toán được cài đặt mô phỏng bằng ngôn ngữ Java (NetBean 7.1.1, JDK 6), gói công cụ CloudSim 2.0 [5] với các thông số sau: Sử dụng 4 Datacenter, 10 host vật lý, số máy ảo thay đổi từ 100 đến 500, 4 PE và 512 RAM trên một máy ảo và số yêu cầu thay đổi từ 1000 đến 5000.

III.7.1. Về phía người dùng

Kế thừa lên lớp Cloudlet để tạo ra các yêu cầu người dùng với các thông số: thời gian đến, khối lượng công việc, ngân sách, tỉ lệ lãi suất phạt và deadline. Các thông số này được xác định như sau: Thời gian đến được lấy ngẫu nhiên từ 1 đến 500, chúng ta phát sinh deadline một cách ngẫu nhiên giữa (d_l,d_u) phút và các giá trị khác nhau d_l và d_u có giới hạn từ 10 đến 1500, deadline phải lớn hơn thời gian đến. Khối lượng công việc được lấy ngẫu nhiên từ $8*10^4$ MI đến 10^5MI, căn cứ vào khối lượng để ước lượng ngân sách cho các yêu cầu, các thông số còn lại được lấy ngầm định như trong CloudSim.

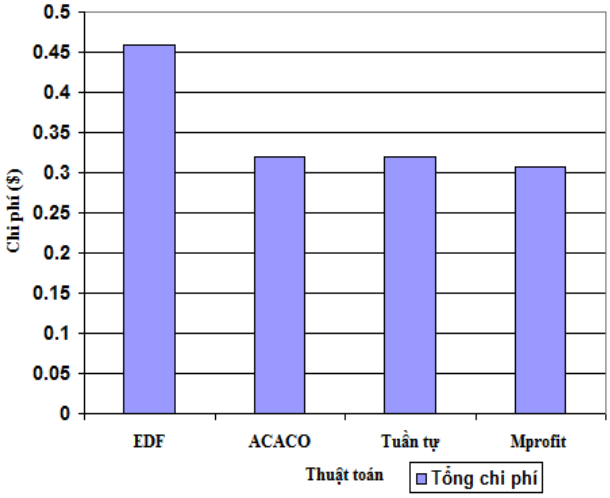
III.7.2. Về phía người cung cấp tài nguyên

Chúng tôi mô phỏng trên bốn nhà cung cấp tài nguyên. Mỗi nhà cung cấp tài nguyên có số máy ảo, chi phí và tốc độ khác nhau. Trong cài đặt, chúng tôi kế thừa lên lớp Vm của CloudSim 2.0 để tạo ra các máy ảo với các thông số tốc độ và chi phí được xác định như sau: tốc độ được lấy ngẫu nhiên từ 10^3 đến $5*10^3$ MIPS tương ứng với chi phí là số thực được lấy ngẫu nhiên từ 0.001 đến 0.01, các thông số khác của máy ảo như thời gian khởi tạo máy ảo, băng thông, v.v., được lấy ngầm định như trong CloudSim.

III.7.3. Kết quả mô phỏng

a) Phân tích tổng chi phí thực hiện khi cố định số yêu cầu

Hình 2 chỉ ra tổng chi phí của bốn thuật toán EDF, ACACO, MProfit và tuần tự khi sử dụng 100 máy ảo và 1000 yêu cầu. Các giá trị trong mô phỏng là kết quả của 5 lần chạy thử và lấy kết quả trung bình.

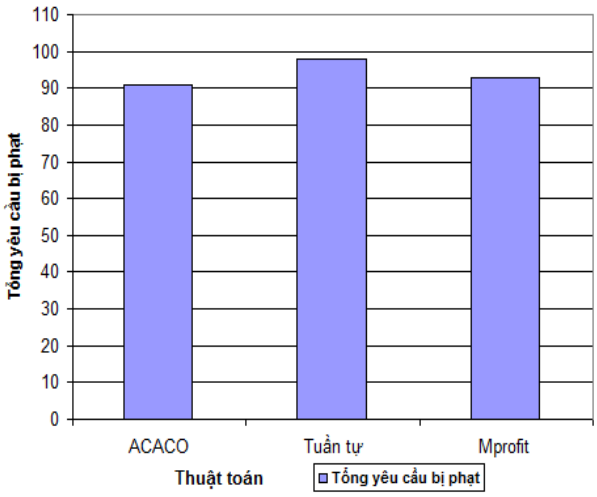


***Hình 2**. Tổng chi phí của thuật toán EDF, ACACO, Tuần tự và MProfit khi cố định số lượng yêu cầu.*

Kết quả mô phỏng cho thấy tổng chi phí của thuật toán MProfit luôn nhỏ hơn thuật toán EDF, ACACO và tuần tự. Bởi vì thuật toán ACACO có nhiệm vụ kiểm soát đầu vào, chấp nhận hay từ chối yêu cầu người dùng. Nếu yêu cầu được chấp nhận, nó sẽ ánh xạ vào máy ảo có chi phí thấp. Sau khi thuật toán ACACO thực hiện xong, chúng ta có được một tập các yêu cầu được chấp nhận với chi phí thấp. Tập yêu cầu này là dữ liệu đầu vào của thuật toán MProfit, MProfit tiếp tục tận dụng khoảng thời thời gian gối đầu của các yêu cầu lên các tài nguyên trong cùng một nhà cung cấp IaaS. Điều này dẫn đến tổng chi phí thực hiện của thuật toán giảm xuống. Thuật toán tuần tự không xem xét khoảng thời gian gối đầu giữa các yêu cầu chỉ dùng thuật toán vét cạn để tìm tài nguyên. Do đó, sẽ có nhiều trường hợp các yêu cầu sử dụng không hết

khoảng thời gian được thuê. Điều này sẽ làm cho chi phí của thuật toán tuần tự tăng lên và mất một khoảng thời gian rất lớn để đưa ra lịch trình. Còn thuật toán EDF chỉ xem xét đến tỉ số sử dụng: $U = \sum_{i=1}^{m} \frac{C_i}{T_i} \leq 1$ (trong đó C_i là thời gian thực hiện và T_i tương ứng với deadline) [15, 16] để ánh xạ các yêu cầu vào các tài nguyên. Do đó thuật toán EDF chỉ đảm bảo các yêu cầu hoàn thành trước deadline của nó chứ không quan tâm đến ngân sách cho các yêu cầu.

b) Phân tích tổng số yêu cầu mà nhà cung cấp SaaS chịu phạt.

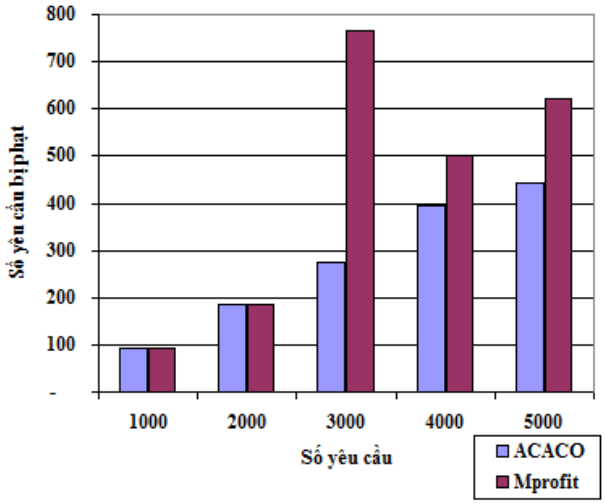


***Hình 3**. Tổng số yêu cầu bị phạt của thuật toán ACACO, Tuần tự và MProfit khi cố định số lượng yêu cầu*

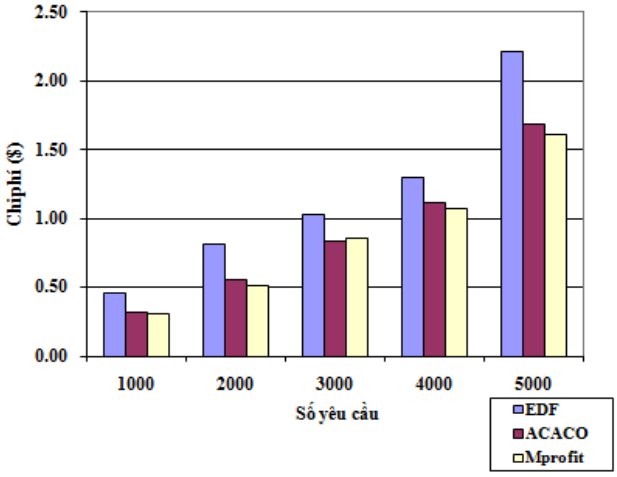
Các yêu cầu trong thuật toán EDF hầu như không bị phạt, vì thuật toán chọn các tài nguyên để hoàn thành trước deadline của các yêu cầu. Còn thuật toán tuần tự, ACACO và MProfit xem xét nếu cộng thêm chi phí bị phạt mà vẫn có lợi cho nhà cung cấp SaaS thì yêu cầu đó vẫn được chấp nhận. Hình 3 trình bày tổng số yêu cầu mà nhà cung cấp bị phạt khi số yêu cầu cố định là 1000.

c) Phân tích tổng chi phí và tổng số yêu cầu mà nhà cung cấp SaaS chịu phạt khi cố định số máy ảo và thay đổi số yêu cầu.

Phần này trình bày kết quả về tổng chi phí và tổng số yêu cầu bị phạt của ba thuật toán khi thay đổi số lượng yêu cầu từ 1000 yêu cầu đến 5000 và cố định số máy ảo là 100 như thể hiện ở Hình 4 và Hình 5.



***Hình 4.** Tổng yêu cầu bị phạt của thuật toán ACACO và MProfit khi thay đổi số lượng yêu cầu.*



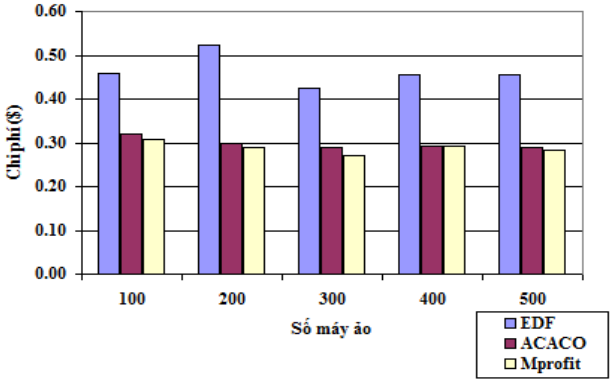
***Hình 5.** Tổng chi phí của thuật toán EDF, ACACO và MProfit khi thay đổi số lượng yêu cầu.*

Thuật toán tuần tự sử dụng thuật toán vét cạn để tìm tài nguyên. Khi số yêu cầu lớn thì thời gian để đưa ra lịch trình của thuật toán tuần tự là rất lớn vì độ phức tạp là hàm mũ. Chính vì vậy, trong phần này chúng tôi không xét đến thuật toán tuần tự. Khi số yêu cầu càng lớn thì tổng chi phí của thuật toán MProfit nhỏ hơn thuật toán EDF và ACACO. Tuy nhiên, trong một số trường hợp khi số yêu cầu bị phạt lớn như trong trường hợp 3000 yêu cầu của Hình 4 thì chi phí phạt mà nhà cung cấp phải trả lớn. Điều này dẫn đến tổng chi phí của hệ thống sẽ lớn như Hình 5. Do đó, trong cài đặt ta so sánh tổng chi phí của hai thuật toán ACACO và MProfit để quyết định chọn thuật toán nào

để tối ưu.

d) Phân tích tổng chi phí khi thay đổi số máy ảo.

Khi tăng số lượng máy ảo thì xác suất để chọn máy ảo có chi phí thấp sẽ cao lên, dẫn đến tổng chi phí của thuật toán MProfit sẽ nhỏ hơn tổng chi phí của thuật toán ACACO và EDF như Hình 6.



Hình 6. *Tổng chi phí của thuật toán EDF, ACACO và MProfit khi thay đổi số lượng máy ảo.*

IV. KẾT LUẬN

Trong bài báo này, chúng tôi tập trung nghiên cứu vấn đề kiểm soát đầu vào và lập lịch cho các yêu cầu của người dùng với các ràng buộc QoS. Trong đó, mỗi yêu cầu được xem xét đến các yếu tố như như thời gian đến, chi phí, deadline, ngân sách, khối lượng, tỉ lệ lãi suất phạt, kích cỡ file đầu vào và đầu ra; mỗi máy ảo bao gồm tốc độ và chi phí khác nhau. Để đem lại lợi nhuận cao nhất cho nhà cung cấp dịch vụ SaaS, bài báo đề xuất hai thuật toán ACACO và MProfit. Cả hai thuật toán này được nghiên cứu để tìm kiếm tài nguyên có chi phí thấp trong nhà cung cấp dịch vụ IaaS để cung cấp cho các yêu cầu người dùng. Thông qua việc phân tích, đánh giá các kết quả thực nghiệm, đối sách trên cùng một bộ mẫu và sử dụng cùng một công cụ mô phỏng CloudSim cho thấy kết quả của thuật toán ACACO và MProfit có sự cải tiến đáng kể về chi phí so với thuật toán tuần tự và EDF đang sử dụng.

TÀI LIỆU THAM KHẢO

[1] P. BRUCKER, *Scheduling Algorithms*, Fifth Edition, Springer Press, 2007.

[2] http://www.microsoft.com/window

[3] http://www.ibm.com/ibm/cloud/ibm_cloud/

[4] http://aws.amazon.com

[5] BUYYA, R., RANJAN, *Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities*, Proceedings of the 7th High Performance Computing and Simulation Conference, Leipzig, Germany, 2009.

[6] MARCO DORIGO and THOMAS STŸTZLE, *Ant Colony Optimization*, A Bradford Book, The MIT Press Cambridge, Massachusetts,London, England, 2004.

[7] THOMAS STŸTZLE, MARCO DORIGO: *A Short Convergence Proof for a Class of Ant Colony Optimization Algorithms*, IEEE transactions on evolutionary computation, Vol. 6, No. 4, August 2002.

[8] KUN LI, GAOCHAO XU, GUANGYU ZHAO, YUSHUANG DONG, DAN WANG, *Cloud Task scheduling based on Load Balancing Ant Colony Optimization*, Sixth Annual ChinaGrid Conference, 2011.

[9] JZAU-SHENG LIN and SHOU-HUNG WU, Fuzzy Artificial Bee Colony System with Cooling Schedule for the Segmentation of Medical Images by Using of Spatial Information, Research Journal of Applied Sciences, Engineering and Technology 4, 2973-2980, 2012

[10]A. BURNS, R.I. DAVIS, P. WANG and F. ZHANG, *Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme*. Department of Computer Science, University of York, UK.

[11] LUKASZ KRUK, JOHN LEHOCZKY, KAVITA RAMANAN And STEVEN SHREVE, *Heavy traffic analysis for EDF queues with reneging*, The Annals of Applied Probability. Vol. 21, No. 2, 484–545, 2011.

[12] JIANGUANG DENG, YUELONG ZHAO, HUAQIANG YUAN, *A Service Revenue-oriented Task Scheduling Model of Cloud Computing*, Journal of Information & Computational Science, July 1, 2013.

[13] MAO, MING and LI, JIE, *Cloud auto-scaling with deadline and budget constraints, Grid Computing*, 11th IEEE/ACM International Conference , 2010.

[14] K. H. KIM et al, *Power-aware provisioning of cloud resources for real-time services.* In International Workshop on Middleware for Grids, Clouds and e-Science , pages 1–6,

2009.

[15] RAMKUMAR N, NIVETHITHA S, *Efficient Resource Utilization Algorithm (ERUA) for Service Request Scheduling in Cloud*, International Journal of Engineering and Technology (IJET), Vol 5 No 2 Apr-May 2013.

[16] SWARUPA IRUGURALA, DR.K.SHAHU CHATRAPATI, *Various Scheduling Algorithms for Resource Allocation In Cloud Computing*, The International Journal Of Engineering And Science (IJES), page 16-24, 2013.

[17] KOUSALYA.K, BALASUBRAMANIE.P: *An Enhanced Ant Algorithm for Grid Scheduling Problem*, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, April 2008.

Nhận bài ngày: 1/10/2014

SƠ LƯỢC VỀ TÁC GIẢ

NGUYỄN HOÀNG HÀ

Sinh năm 1976 tại Thăng Bình, Quảng Nam

Nhận bằng thạc sỹ Tin học, chuyên ngành Khoa học Máy tính tại Trường Đại học Khoa học– Đại học Huế, năm 2005. Đang là NCS tại trường Đại học Khoa học – Đại học Huế, chuyên ngành Khoa học Máy tính.

Hiện công tác tại khoa CNTT, Trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Xử lý song song và phân tán, tính toán lưới và tính toán đám mây.

Điện thoại liên hệ: 0914426033

Email: nhha76@gmail.com

LÊ VĂN SƠN

Sinh năm 1948 tại Điện Bàn, Quảng Nam.

Tốt nghiệp Đại học năm 1978, bảo vệ Tiến sĩ năm 1997 tại trường Đại học Tổng hợp Donesk, Ucraina, công nhận PGS năm 2004.

Hiện công tác tại Khoa Tin học, Đại học Sư phạm, Đại học Đà Nẵng

Lĩnh vực quan tâm : Hệ điều hành, mạng máy tính, hệ phân tán, tính toán đám mây.

E-mail : levansupham2004@yahoo

NGUYỄN MẬU HÂN

Sinh năm 1957 tại Thừa thiên Huế.

Nhận bằng Tiến sĩ tại Viện CNTT.

Hiện là Phó Giáo Sư, giảng viên chính tại khoa CNTT, Trường Đại học Khoa học, Đại học Huế.

Lĩnh vực nghiên cứu: Xử lý song song và phân tán, tính toán lưới và điện toán đám mây.

Email: nmhan2009@gmail.com