TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Ngành khoa học máy tính

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

Sử dụng mạng Bayesian ước lượng tài nguyên khả dụng cho bài toán lập lịch thời gian thực trong môi trường điện toán đám mây

Trương Quang Khánh
khanh.tq170083@sis.hust.edu.vn

Giảng viên hướng dẫn: TS. Nguyễn Bình Minh

Chữ ký của GVHD

Bộ môn : Hệ thống Thông tin
Viện : Công nghệ thông tin và truyền thông

Hà nội, tháng 5 năm 2021

ĐỀ TÀI TỐT NGHIỆP

1. **Thông tin về sinh viên**
   Họ và tên: Trương Quang Khánh
   MSSV: 20170083
   Số điện thoại: 0343863208 Email: khanh.tq170083@gmail.com
   Lớp: KSTN-CNTT K62 Hệ đào tạo: Chính quy
   Đồ án tốt nghiệp được thực hiện tại: Trường ĐHBKHN
   Thời gian làm ĐATN: Từ tháng 2/2021 đến tháng 6/2021

2. **Mục đích nội dung của ĐATN**
   Đề xuất thuật toán lập lịch trong môi trường cloud computing.

3. **Các nhiệm vụ cụ thể của ĐATN**

   - *1.* Nghiên cứu các hạn chế của các thuật toán lập lịch động trong hệ thống Cloud.
   - *2.* Đưa ra mô hình lập lịch cải tiến sai số do độ trễ trong quá trình truyền tin.
   - *3.* Cài đặt và thử nghiệm thuật toán đề xuất.

4. **Lời cam đoan của sinh viên**
   Tôi - *Trương Quang Khánh* - xin cam kết đồ án này là do bản thân tự làm dưới sự hướng dẫn của thầy *Nguyễn Bình Minh*, không sao chép ý tưởng, đạo văn từ bất kỳ nguồn tài liệu nào khác.

   Hà Nội, ngày tháng năm 2021
   Tác giả ĐATN

   Trương Quang Khánh

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 2021
   Giáo viên hướng dẫn

   TS. Nguyễn Bình Minh

LỜI CẢM ƠN

Trước tiên, tôi xin được gửi lời cảm ơn đến gia đình, thầy cô, bạn bè, những người đã luôn bên cạnh tôi trong suốt thời gian qua. Xin được gửi lời cảm ơn và lời chúc tới **Phương**, **Giang**, **Hiếu**, họ là những người bạn gắn liền với thời gian học đại học của tôi.
Xin được cảm ơn anh Nguyễn Đức Thắng vì là người đã lắng nghe ý tưởng và đưa ra những lời khuyên giúp tôi cải thiện năng lực của bản thân.
Cuối cùng, tôi muốn gửi lời cảm ơn chân thành nhất tới thầy **Nguyễn Bình Minh** - người đã luôn tận tình "trải đường" cho sinh viên đến với khoa học, trong đó có bản thân tôi. Trong quá trình làm đồ án, tôi đã được thầy hướng dẫn rất nhiều điều, cả về các tình huống trong nghiên cứu và các kĩ năng mềm trong cuộc sống. Một vài trang giấy khó nói hết, dù vậy, đồ án chính là nỗ lực của bản thân tôi, cũng là lời cảm ơn mà tôi muốn gửi đến thầy.

TÓM TẮT NỘI DUNG

Trong môi trường cloud computing, lập lịch là một phương pháp phân phối tài nguyên cho các công việc được gửi tới, mục đích làm tăng độ hiệu quả và chất lượng dịch vụ, tận dụng tối đa tài nguyên hệ thống. Do đặc thù của hệ thống cloud computing, các công việc được gửi đến hệ thống đến hệ thống liên tiếp và không xác định trước, thuật toán lập lịch cần phải thích nghi với việc trạng thái hệ thống, khối lượng công việc thay đổi liên tục theo thời gian thực. Đồ án này đề xuất sử dụng mô hình đồ thị xác suất để ước lượng thành phần dễ dàng thay đổi trong hệ thống. Mô hình được đề xuất sử dụng mạng Bayesian để tính kỳ vọng cho các thành phần "không chắc chắn" (vd: tài nguyên khả dụng sau 10 giây), sau đó dùng thông tin ước lượng được để lập lịch.

Mục lục

Danh sách hình vẽ

Danh sách bảng

1

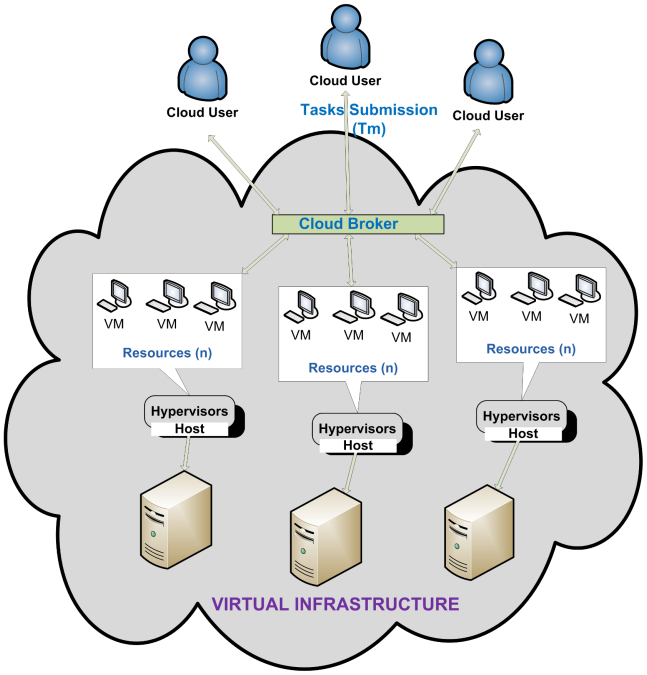
Giới thiệu

1.1 Điện toán đám mây (Cloud computing)

Để có thể hiểu về **điện toán đám mây**, hay còn gọi là **cloud computing** (sau đây ta thống nhất sử dụng thuật ngữ cloud computing), trước tiên ta nên làm quen với các khái niệm sau:

- **Data center**: Là hệ thống các máy chủ trong công ty, tổ chức nhằm phục vụ mục đích lưu trữ và truy cập dữ liệu, cung cấp tiện ích tới người dùng (nhân viên, ...) thông qua mạng nội bộ. Việc xây dựng, bảo trì hệ thống được thực hiện bởi nội bộ công ty.
- **Cloud**: Là phiên bản "từ xa" của Data center, được đặt ở nơi nào đó không phải là ở công ty, tổ chức, người dùng truy cập thông qua internet. Việc quản lý, xây dựng được thực hiện bởi nhà cung cấp dịch vụ cloud.
- **Grid computing**[1]: là mô hình tính toán bằng cách kết hợp khả năng tính toán của nhiều máy tính khác nhau, thích hợp cho các vấn đề đòi hỏi nặng về tính toán như trí tuệ nhân tạo, tin y sinh, ...
- **Utility computing**[2][3]: là mô hình cung cấp các dịch vụ công nghệ thông tin theo nhu cầu của người dùng với các mức phí sử dụng.
- **Virtualization computing**[4]: là công nghệ tạo ra các phiên bản "ảo hóa" của các phiên bản thật, ví dụ như máy tính ảo, mô trường tính toán ảo, ... Ở trong lĩnh vực cloud computing, công nghệ ảo hóa còn được sử dụng để tạo ra các tài nguyên ảo như CPU, ram, ổ đĩa, ... nhằm mục đích dễ dàng phân chia, quản lý tài nguyên.

Cloud computing[2] là sự tích hợp của nhiều công nghệ khác nhau, trong đó bao gồm **grid computing**, **utility computing**, **virtualization computing**, Web 2.0 để cung cấp tài nguyên tính toán, lưu trữ và các dịch vụ công nghệ đến người dùng thông qua internet.



Hình 1.1: Lập lịch trong môi trường cloud computing

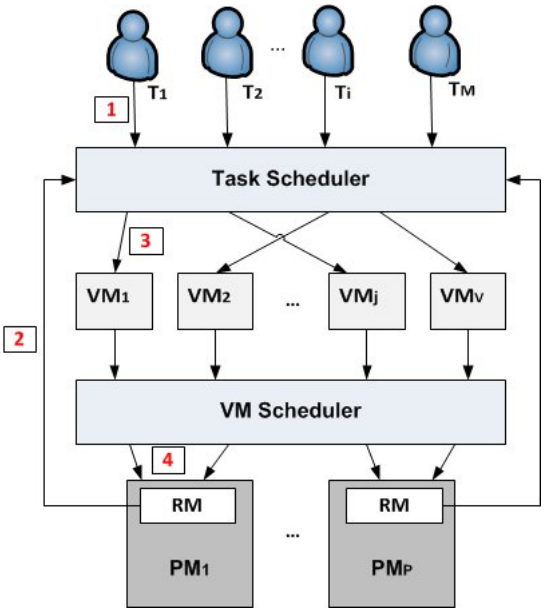
Nguồn: [8]

1.2 Bài toán lập lịch trong môi trường cloud computing

Vấn đề

Có rất nhiều loại hình dịch vụ được cung cấp dưới hình thức cloud computing, ví dụ như hệ sinh thái Google Docs, Slide, Sheet, Google Data Studio, Google Colab ... được cung cấp bởi Google, cũng như nhiều dịch vụ tương tự từ các hãng khác như Amazone, Microsoft, Alibaba, ... Có một đặc điểm chung của các loại hình dịch vụ này là người dùng sẽ thông qua các giao diện hoặc API được cung cấp để gửi yêu cầu thực hiện các công việc của họ. Có nhiều mô hình chi phí cho các loại hình dịch vụ này, trong đó mà đang trở lên rất phổ biến là loại hình "pay as you go"[5], tức là dùng bao nhiêu tài nguyên thì ngân sách bỏ ra là tương ứng. Với mô hình này, nếu có thể sử dụng hiệu quả tài nguyên tính toán của hệ thống thì ta có thể giảm thiểu chi phí sử dụng dịch vụ cho người dùng cũng như tăng doanh thu cho nhà cung cấp. Bài toán tập lịch (task scheduling[6][7]) trong hệ thống cloud computing hướng tới tối ưu tài nguyên sử dụng cho các yêu cầu tính toán được gửi tới. Như trong hình 1.1, các công việc tính toán sẽ được gửi từ người dùng bằng các ứng dụng web, api, ... tới trung tâm máy chủ, hay chính là cloud. **Cloud broker**, là trung gian giữa hệ thống tính toán của máy chủ với các ứng dụng bên ngoài, có nhiệm vụ tiếp nhận, quản lý và phân phối tới các tài nguyên trong cloud. Việc phân phối này sẽ sử dụng bộ lập lịch (scheduler) để tối ưu tài nguyên sử dụng và các thông số đánh giá chất lượng của dịch vụ (qualities of services). Sau đấy các công việc (từ giờ trở đi sẽ được viết theo tên tiếng anh, tasks) sẽ được chuyển tới máy tính ảo tương ứng với kết quả của việc lập lịch, và được thực thi trên đó. Các tài nguyên vật lý sẽ được quản lý bằng cách ảo hóa thành các tài nguyên ảo, nhằm mục đích dễ phân

chia và mở rộng. Như vậy, với công nghệ ảo hóa (virtualization computing), để



Hình 1.2: Các thành phần lập lịch

Nguồn: [9]

thực thi task được gửi từ người dùng thì ta cần có 2 bô lập lịch sau (hình 1.2):

- **Task Scheduler**: là chương trình phân phối tasks đến các máy tính ảo.
- **VM Scheduler**: là chương trình quyết định xem các máy tính ảo được triển khai trên các máy chủ vật lý nào.

Đồ án này nghiên cứu về task scheduler, do đó từ giờ trở đi khi thuật ngữ "scheduler" được nhắc đến mà không ghi cụ thể thì ta hiểu rằng nó là "task scheduler".

Tiêu chí tối ưu

Trước tiên, bài toán lập lịch trong môi trường cloud computing là một bài toán tối ưu. Ta biết rằng mục đích của việc lập lịch là tối ưu việc sử dụng tài nguyên, tăng hiệu năng của hệ thống và xa hơn là tăng doanh thu cho doanh nghiệp. Có thể nói đó là những mục tiêu "vĩ mô", tạo nên nó là sự kết hợp của rất nhiều phần trong doanh nghiệp, và rất khó để đánh giá nếu chỉ tính dựa trên hiệu năng của bộ lập lịch. Do đó, ta cần đưa ra các tiêu chí, metrics mà dựa vào nó ta có thể đánh giá, so sánh giữa những thuật toán lập lịch với nhau. Dưới đây là một số hàm mục tiêu[10], thường được sử dụng trong nghiên cứu và phát triển:

- **makespan**: thời gian hoàn thành một khối lượng các công việc đã được xác định
- **cost**: chi phí để hoàn thành một khối lượng công việc
- **throughput**: lượng công việc hoàn thành được trong một đơn vị thời gian

- **load balancing**: sự cân bằng mật độ dữ liệu tại mọi thời điểm trong hệ thống cloud
- **energy**: mức tiêu thụ năng lượng của hệ thống để hoàn thành một khối lượng công việc
- **execution time**: thời gian thực thi tasks
- **response time**: thời gian chờ đợi phản hồi từ người dùng
- **qualities of service**: được đưa ra trong thỏa thuận cấp độ dịch vụ (SLA – Service Level Agreement), có thể bao gồm các ràng buộc về deadline, makespan, cost, ...

Bài toán lập lịch

Xét hệ thống có M máy tính ảo, cần thực thi N tasks.

- $vms = \{vm_1, vm_2, ..., vm_M\}$ là tập M vector thông tin của máy ảo
- $tasks = \{task_1, task_2, ..., task_N\}$ là tập N vector thông tin của tasks
- $\mathcal{S} = \{task_i \rightarrow vm_j\}$ là cách ghép cặp N tasks tới M máy ảo
- $\mathcal{U} = \{\mathcal{S}\}$, là tập hợp tất cả các cách ghép cặp
- $\mathcal{F}(S)$ là hàm tối ưu (có thể là giá trị vô hướng hoặc là vector trong trường hợp xét nhiều hơn 2 mục tiêu) của một cách ghép cặp.

Không mất tính tổng quát, giả sử bài toán yêu cầu cực tiểu hóa hàm mục tiêu, vì nếu ngược lại thì có thể đổi dấu của nó. Ta cần tìm $\mathcal{S}^*$ sao cho:

$$\mathcal{S}^* = argmin_{\mathcal{S} \in \mathcal{U}} \mathcal{F}(\mathcal{S}) \tag{1.1}$$

1.3 Lập lịch thời gian thực trong môi trường cloud computing

Giống như bài toán lập lịch động được nhắc đến trong 2.1.6, nhiều thông tin của task và môi trường có thể thay đổi trong thời gian thực. Hơn nữa, ở trong môi trường cloud computing, các công việc được gửi tới từ người dùng rất đa dạng, và có thêm mức ưu tiên cho từng công việc, khiến cho việc xác định chính xác một số thông tin của task như thời gian thực thi là không thể. Như ở trong bộ dữ liệu google trace v2[11], không thể xác định thời gian thực thi của tasks khi chúng đến hệ thống, và con số về tài nguyên được yêu cầu với tài nguyên thực tế sử dụng của các tasks là không đồng nhất[12].
Trong hệ thống thực tế, số lượng tasks đến theo thời gian là vô hạn.

- $vms = \{vm_1, vm_2, ..., vm_M\}$ là tập M vector thông tin của máy ảo
- $tasks = \{task_1, task_2, ..., task_N\}$ là tập vector thông tin của tasks với $N \rightarrow \infty$

- $times = \{t_1, t_2, ..., t_N\}$ là tập thời gian đến hệ thống của các task với chỉ số tương ứng
- $constraints = \{r_1, r_2, ..., r_N\}$ là tập ràng buộc tương ứng với các tasks

Trong bài toán lập lịch này, trạng thái tài nguyên của hệ thống sẽ rất khó xác định. Có hai nguyên nhân chính sau:

- tài nguyên được yêu cầu bởi các task với tài nguyên thực tế chúng sử dụng là không đồng nhất
- sự thay đổi trạng thái của task trong quá trình chạy (khi task đang thực thi thì bị dừng lại để nhường tài nguyên cho task có ưu tiên cao hơn, hoặc kết thúc công việc)

Ngoài ra còn có sự thay đổi trong cấu hình của hệ thống. Bên cạnh sự không chắc chắn về trạng thái tài nguyên, các thông tin về thời gian chạy của task cũng không xác định. Điều này làm cho một số thuật toán phát triển trước đó như min-min hay max-min không áp dụng được. Load-balancing cũng trở lên tồi tệ do trạng thái tasks thay đổi quá nhanh làm mất cân bằng giữa các máy tính trong hệ thống.

1.4 Mục tiêu của đồ án nghiên cứu

Như đã trình bày trong 1.3, sự không chắc chắn về trạng thái tài nguyên cùng với việc thiếu thông tin về thời gian thực thi của các tasks khiến cho việc lập lịch không thể tối ưu một vài chỉ số như makespan, throughput, load-balancing. Đồ án này có mục tiêu cải thiện các vấn đề trên, cụ thể:

1. Xác định các nguyên nhân làm cho các thuật toán lập lịch cổ điển không tối ưu cho bài toán lập lịch thời gian thực trong môi trường điện toán đám mây
2. Đưa ra mô hình ước lượng trạng thái tài nguyên trong quá trình hoạt động của hệ thống và cơ chế lập lịch để sự thay đổi trạng thái các tasks không "phá vỡ" load-balancing của hệ thống
3. Cài đặt và đánh giá mô hình đề xuất

Với mục tiêu thứ nhất, nhiều mô hình học máy đã được xem xét, trong đó mô hình đồ thị xác suất Bayesian là một lựa chọn phù hợp để biểu diễn các điều kiện độc lập và phụ thuộc giữa các thành phần trong hệ thống, dễ dàng học "tri thức" từ dữ liệu và đưa ra suy luận trong thời gian chấp nhận được. Tiếp theo, thuật toán **bestfit** kết hợp với các kết quả ước lượng trước đó sẽ đưa ra lời giải mà cải thiện load-balancing của hệ thống.

1.5 Cấu trúc đồ án

Đồ án này được chia thành các phần chính như sau:

- Chương 1 giới thiệu toản quan về bài toán nghiên cứu.
- Chương 2 tóm tắt ngắn gọn về các kiểu lập lịch và những thuật toán đã được nghiên cứu trước đây.
- Chương 3 giới thiệu về đồ thị xác suất Bayesian.
- Chương 4 mô tả mô hình lập lịch đề xuất.
- Chương 5 là thực nghiệm và đánh giá.
- Chương 6 Kết luận và định hướng nghiên cứu.

Phạm vi đồ án bao gồm:

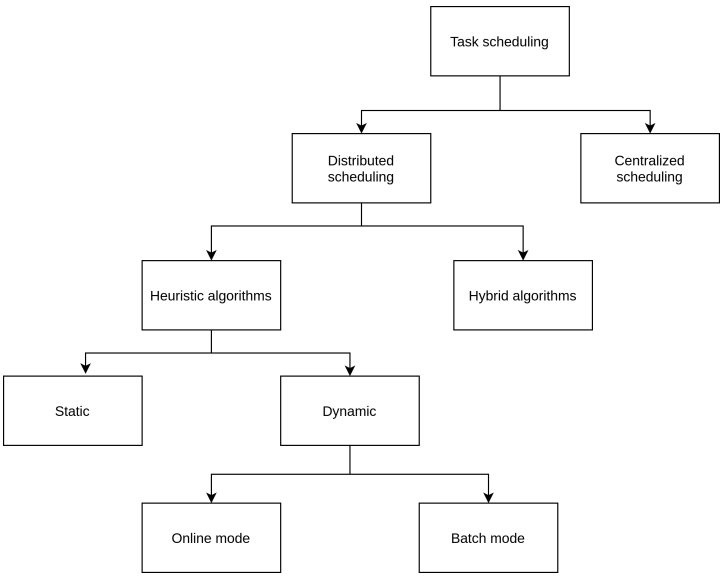
- Mô phỏng môi trường hoạt động Cloud bằng công cụ CloudSim [13]
- Thu thập và trích rút dữ liệu mô phỏng từ API của Google
- Cài đặt thuật toán đề xuất và một vài thuật toán liên quan trong môi trường mô phỏng
- So sánh hiệu quả lập lịch giữa thuật toán đề xuất và các thuật toán khác

2

Các dạng lập lịch và nghiên cứu liên quan

2.1 Các phương pháp lập lịch

Trong [10], tác giả đã phân chia các thuật toán lập lịch dưới dạng cây phân cấp như hình 2.1. Đồ án này tập chung giải quyết bài toán **batch mode**, do đó ở



Hình 2.1: Các phương pháp lập lịch

các phần như **centralized scheduling**, thực tế sẽ được phân thành nhiều phần cụ thể hơn, nhưng do mục đích của ta không phải là nghiên cứu tổng quan giữa các kiểu bài toán lập lịch nên sẽ không làm rõ ở đây.

2.1.1 Centralized scheduling

Centralized scheduling[14] là kiểu bài toán lập lịch khi chỉ có một bộ lập lịch chung cho tất cả các task đến hệ thống. Bộ lập lịch này sẽ quản lý thông tin của toàn bộ cluster, thường các cluster này sẽ ở gần nhau theo vị trí địa lý.

2.1.2 Distributed scheduling

Distributed scheduling[14] là trong trường hợp sẽ có nhiều bộ lập lịch được đặt ở các vị trí chung gian giữa người dùng và các cluster của hệ thống. Các bộ lập lịch này sẽ trao đổi thông tin với toàn bộ cluster, định tuyến và phân chia tài nguyên cho các task. Ưu điểm của mô hình này so với **centralized scheduling** là dễ dàng mở rộng quy mô máy chủ, người dùng, nhưng có hạn chế là do bộ lập lịch cách xa các cluster nên sẽ có độ trễ trong khi trao đổi thông tin.

2.1.3 Hybrid algorithm

Hybrid scheduling algorithms là nhóm các thuật toán mà kết hợp nhiều phương pháp khác nhau để tối ưu nhiều tiêu chí cùng một lúc. Ví dụ như trong [15], tác giả sử dụng "lexi-search" để đưa ra lời giải tối ưu **execution time**, **load balancing**.

2.1.4 Heuristic algorithms

Trong [10][16], **heuristic scheduling** là việc lập lịch dựa vào những giải thuật heuristic như PSO, Genetic, ... Trong môi trường phân tán, các thông tin về máy chủ và task có thể là không đồng nhất, do đó để tìm được lời giải tối ưu tuyệt đối cho bài toán là vô cùng khó, và đôi khi là không cần thiết. Các thuật toán heuristic sẽ tìm cho ta lời giải "chấp nhận được" trong thời gian cho phép.

2.1.5 Static scheduling

Static scheduling, hay là bài toán lập lịch tĩnh, các tasks sẽ đến hệ thống vào cùng một thời điểm, không bị ảnh hưởng bởi các yếu tố liên quan đến hệ thống tài nguyên. Các thông tin về cấu trúc của tasks, thông tin về môi trường, khối lượng công việc đều được xác định. Tất cả các tasks đều sẽ được phân phối tài nguyên trước khi thực thi. Với loại bài toán này, một số giải thuật "cổ điển" như "First Come First Serve", "Round Robin" đã được ứng dụng trong rất nhiều hệ thống. Ưu điểm của các giải thuật này là đơn giản, dễ triển khai, nhưng nhược điểm là không thể tối ưu các tiêu chí được để cập trước đó. Ngoài ra, min-min và max-min là hai thuật toán heuristic cơ bản được áp dụng phổ biến.

- **Min-min**[17] là cách lựa chọn task có execution time nhỏ nhất để gán vào máy tính tương ứng trước tiên. Những task dài sẽ được gán sau cùng, và nếu hệ thống không đủ tài nguyên thì chúng phải chờ đợi đến khi tài nguyên khả dụng. Ta đã biết rằng các tasks dài mới là thành phần quyết định đến chỉ số makespan, do đó thuật toán này chưa tối ưu về makespan, nhưng bù lại, chúng làm tăng throughput, tức là số lượng task hoàn thành trong một đơn vị thời gian. Vấn đề đói tài nguyên cũng sẽ khiến thuật toán này trở lên "tồi tệ", do có thể xảy ra trường hợp task "dài" sẽ luôn trong tình trạng không đủ tài nguyên do chúng luôn được lập lịch sau cùng.
- **Max-min**[18] có một chút ngược lại với min-min. Thuật toán này lựa chọn task có execution time lớn nhất được phân phối tài nguyên trước. Với cách lập lịch này thì các task "ngắn" sẽ được sắp xếp cuối cùng, và cũng có nguy

cơ bị "đói tài nguyên", tức là sau nhiều lần lập lịch vẫn chưa được thực thi do thiếu tài nguyên. Do các task "dài" được sắp xếp trước nên makespan sẽ nhỏ hơn so với min-min, nhưng throughput cũng tương ứng giảm.

2.1.6 Dynamic scheduling

Trong **dynamic scheduling** (lập lịch động), yếu tố **dynamic** được xem xét ở cả tasks và tài nguyên hệ thống. Tasks sẽ đến hệ thống theo thời gian, và có nhiều yếu tố ảnh hưởng đến nó trong quá trình chạy. Ví dụ như task có thể bị fail khi tài nguyên hệ thống không đủ và nhường cho task khác, hay thời gian chạy của task là không thể xác định trước. Hơn nữa, các máy tính có thể bị lỗi, dừng hoạt động trong quá trình chạy, dẫn đến sự thay đổi về tài nguyên khả dụng trong quá trình lập lịch.
Thông thường, các thuật toán lập lịch động được chia làm 2 hướng tiếp cận chính[19] như sau:

- **Online mode**
- **Batch mode**

Với **online mode**, task được lập lịch ngay sau khi nó đến hệ thống, và quyết định lập lịch đó sẽ không bị thay đổi. Cách làm này phù hợp với các trường hợp có tỉ lệ task đến hệ thống thấp[20]. Một số thuật toán lập lịch trong trường hợp này như sau:

- **Minimum completion time (MCT)** lựa chọn máy tính mà có thời gian hoàn thành công việc sớm nhất cho task
- **Minimum execution time (MET)** lựa chọn máy tính có thời gian thực thi task là nhỏ nhất

Với **batch mode**, sau khi đến hệ thống, các task sẽ đến một hàng đợi, sau một khoảng thời gian (interval time) thì hệ thống sẽ lập lịch cho các task trong tập đó. Để lựa chọn thời gian đợi cho hàng đợi, có 2 phương án sau:

- **Regular time interval**: thực hiện lập lịch sau một khoảng thời gian cố định (vd: 100s).
- **Fixed count**: thực hiện lập lịch khi thỏa mãn 1 trong 2 điều kiện sau:
  - Một task mới đến khiến cho lượng task trong tập đang đợi lớn hơn hằng số đặt trước
  - Thời gian chờ đã lớn hơn hoặc bằng hằng số định trước

2.2 Các nghiên cứu liên quan

Đã có nhiều framework được phát triển để giải quyết bài toán lập lịch thời gian thực trong môi trường phân tán.
Với ngữ cảnh các tasks đến hệ thống với n máy ảo theo phân phối Poisson, trong [21], bộ lập lịch chọn ngẫu nhiên d máy ảo và gán task tới máy ảo có số lượng

task đang đợi là ít nhất, *Michael Mitzenmacher* đưa ra kết luận rằng makespan của d = 2 sẽ được cải thiện theo hàm mũ so sánh với d = 1, bên cạnh đó thì d = 3 chỉ tốt hơn theo hàm hằng số so với d = 2. Điểm yếu của thuật toán này là trong trường hợp các tasks được tính toán song song, tức là được thực thi trong nhiều máy tính khác nhau thì việc lựa chọn như trên sẽ làm tăng thời gian hoàn thành công việc.
Trong [22], tác giả đề xuất Sparrow - một mô hình lập lịch hướng tới các hệ thống phân tích dữ liệu lớn, cải tiến nhược điểm trong trường hợp của các tasks cần được song song hóa của phương pháp trước phía trên. Mục tiêu của Sparrow là giảm độ trễ trong quá trình lập lịch cho hệ thống phân tán tính toán song song. Sparrow sử dụng phương pháp được đề xuất trong[23] để lập lịch đảm bảo điều kiện song song hóa giữa các tasks. Sparrow đã được phát triển như một plugin của Apache Spark và tỏ ra hiệu quả so với thuật toán lập lịch nguyên bản của Spark khi số lượng tasks tại một thời điểm lớn nhưng thời gian thực thi ngắn.
Một cách lập lịch phổ biến khác và được áp dụng trong Borg[12][24] - một hệ thống quản lý trung tâm máy chủ của Google, đã bỏ qua phần ước lượng thời gian thực thi trong quá trình lập lịch. Các task đến hệ thống có 3 mức độ ưu tiên là **free**, **production**, **monitor**, lần lược theo mức độ tăng dần. Một máy tính sẽ dừng lại công việc ở mức ưu tiên thấp để dành tài nguyên cho cái có mức ưu tiên cao hơn trong trường hợp không đủ tài nguyên. Cụ thể, việc lập lịch được chia làm 2 giai đoạn:

- **feasible checking**: tìm các máy tính thỏa mãn ràng buộc của task (thường ràng buộc về thư viện) và đủ tài nguyên (tính cả phần tài nguyên được sử dụng cho free tasks)

- **scoring**: tính "điểm" cho các máy tính, dựa trên một số tiêu chuẩn như số lượng tasks bị dừng lại do thiếu tài nguyên, mức độ cân bằng tải giữa các máy tính để lựa chọn máy tính tốt nhất cho task

Quá trình scoring sử dụng một vài biến thể của thuật toán E-PVM[25], mục tiêu là đưa ra một giá trị chi phí mà tối ưu nó sẽ giúp ta đạt được load-balancing của hệ thống. Tuy vậy, việc đảm bảo load-balancing dẫn đến vấn đề phân mảnh bộ nhớ, khiến cho đôi khi tài nguyên dư thừa nhưng không sử dụng được. Ngoài ra, feasible checking và scoring cho từng task rất tốn kém, nên Borg đã sử dụng score caching, tức là lưu lại điểm cho đến khi các các thông tin về task đó hay máy tính tương ứng với nó bị thay đổi. Các tasks cũng được chia thành các nhóm dựa trên ràng buộc của chúng, tức là với 2 task nằm trong cùng một nhóm thì điều kiện để có thể thực thi là như nhau. Như vậy, thay vì tìm tập feasible machines cho từng task, ta sẽ tìm cho nhóm các tasks. Sau khi lập lịch, nếu có trường hợp bị đói tài nguyên thì các task với mức ưu tiên thấp sẽ bị dừng lại, hoặc bị hủy để nhường cho các task ưu tiên cao hơn. Các task bị hoãn đó sẽ quay lại hàng đợi cho lần lập lịch tiếp theo. Cơ chế dừng các task đang chạy để nhường cho các task mức ưu tiên cao hơn giúp cho người dùng dịch vụ trả phí sẽ có trải nghiệm tốt hơn với các dịch vụ khác của Google, nhưng điều này cũng gây ra sự lãng phí tài nguyên khi phải thực hiện cùng một công việc nhiều lần. Một kết quả được đưa ra trong [26] cho thấy rằng việc chạy lại các công việc bị

hoãn chiếm tới 60% tài nguyên CPU của hệ thống.
Trong [27], trong trường hợp số lượng máy tính ảo có thể thay đổi, tác giả đề xuất mô hình lập lịch 2 giai đoạn để lựa chọn cấu hình máy ảo trong hệ thống và phân phối công việc. Cụ thể, trong giai đoạn 1, tasks đến hệ thống sẽ được gán nhãn với các kiểu máy tính dựa theo tiêu chí các yêu cầu của tasks được đáp ứng trong kiểu máy tính đó. Sau đó với mỗi kiểu máy tính, ta sẽ tạo ra một số lượng máy ảo để thực thi các tasks được gán nhãn tương ứng. Giai đoạn 2 chính là phân phối các tasks vào máy tính ảo trùng nhãn với nó. Với mô hình lập lịch này, ta có thể tận dùng tài nguyên hệ thống tối ưu hơn dựa vào việc tự lựa chọn cấu hình cụm máy tính ảo, phân chia phù hợp các kiểu của tasks. Tuy nhiên, tác giải chỉ sử dụng tỉ lệ giữa các kiểu máy tính để đưa ra quyết định số lượng máy tính ảo được tạo ra với các kiểu tương ứng đó. Điều này sẽ khiến cho sự phân phối số lượng máy ảo vào các nhóm trở nên không hợp lý khi số lượng nhóm tăng lên và thông tin của task phức tạp hơn.
Trong [28], nhóm nghiên cứu đề xuất phương pháp mô phỏng Monte Carlo để phân chia số lượng máy tính ảo vào các nhóm kiểu máy tính để cải thiện giai đoạn 1 của thuật toán phía trước, nhưng với giải pháp này thì thời gian là một vấn đề do độ chính xác tỉ lệ thuận với số lần ta mô phỏng. Để không phải lặp lại việc mô phỏng nhiều lần, tác giả đề xuất một độ đo tương đồng giữa hai lần lập lịch, nếu trạng thái hiện tại có độ đo không quá sai khác với trạng thái trước đó thì nó sẽ dùng kết quả của lần lập lịch trước.

Nhận xét

Các mô hình lập lịch trên, tùy từng mô hình mà chúng có mục tiêu tối ưu riêng, nhưng đều hướng tới tăng chất lượng của dịch vụ và hiểu quả việc sử dụng tài nguyên. Tuy vậy, một số vấn đề sau đây vẫn chưa được xem xét cẩn thận khi thực hiện lập lịch trong môi trường cloud:

- Sai số của hệ thống do sự trễ trong việc truyền thông tin.
- Sự tối ưu của các thuật toán nghiên cứu trước đó là tối ưu tại thời điểm lập lịch, không phải là tối ưu trong suốt quá trình chạy. Hệ thống Cloud gồm rất nhiều thành phần phức tạp liên tục thay đổi, điều này sẽ phá vỡ trạng thái tối ưu chấp nhận được - kết quả của việc lập lịch.
- Một số tiêu chí tối ưu có tính chất "tương quan dương" với nhau, tức là nếu chỉ số này tốt hơn thì cái khác cũng tương tự. Ngược lại, một vài tiêu chí tối ưu là "sự đánh đổi", là tối ưu chỉ số này thì cần chấp nhận hạ chất lượng của tiêu chí còn lại. Tuy nhiên, "sự tương quan" và "sự đánh đổi" chưa được đo lường một cách cụ thể.

Hướng tới giải pháp cho hai vấn đề đầu tiên, đồ án này đề xuất mô hình đồ thị xác suất mạng Bayesian để ước lượng kỳ vọng cho tài nguyên khả dụng của hệ thống cloud, từ đó đưa ra quyết định lập lịch.

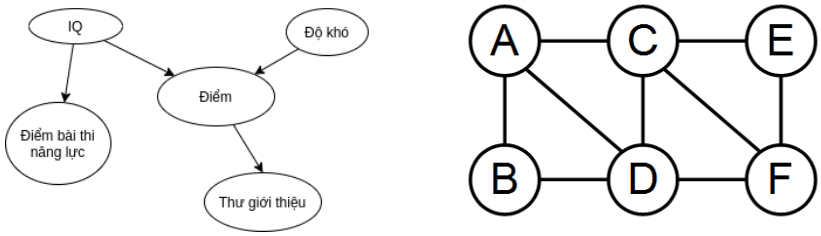
3

Mô hình đồ thị xác suất Bayesian

3.1 Mô hình đồ thị xác suất

Mô hình đồ thị xác suất[29][30] là một dạng mô hình xác suất biểu diễn các điều kiện độc lập giữa các biến ngẫu nhiên dưới dạng đồ thị, là mô hình toán học chứa các giả thuyết xác suất liên quan đến quá trình sinh ra dữ liệu. Về mặt tổng quan, mô hình đồ thị xác suất giúp "phân rã" một phân phối xác suất nhiều chiều phức tạp thành một tập hợp của các phân phối đơn giản hơn với các điều kiện độc lập. Với ý tưởng tương tự như "chia để trị", các phân phối thành phần được tính toán đơn giản hơn và nhờ vậy ta có thể giảm thiểu độ phức tạp của bài toán.

Có hai cách biểu diễn thường dùng nhất với một phân phối, là biểu diễn thành



(a) Đồ thị Bayesian (b) Đồ thị Markov

Hình 3.1: Mô hình đồ thị xác suất

đồ thị có hướng hoặc đồ thị vô hướng. Hai loại đồ thị này còn có tên lần lượt là **đồ thị Bayesian** và **đồ thị Markov**. Việc lựa chọn mô hình đồ thị nào phụ thuộc vào tình huống ta áp dụng. Cụ thể, các điều kiện độc lập giữa các biến ngẫu nhiên sẽ quyết định xem mô hình nào phù hợp với bài toán. Đôi khi cả hai loại đồ thị đều có thể đáp ứng, nhưng phần lớn trường hợp sẽ có một mô hình vượt trội hơn cái còn lại.

Ứng dụng của một mô hình đồ thị xác suất là trả lời các câu hỏi mang tính chất suy diễn khả năng xảy ra của các sự kiện dựa vào một số thông tin được cung cấp. Để làm được điều này, ta cần xây dựng một mô hình phù hợp với bối cảnh

của bài toán, rồi đưa ra cơ chế tính xác suất dựa trên nhưng mô hình đó. Để có thể trả lời chính xác, mô hình của chúng ta cần có "tri thức" - thứ mà được xây dựng từ các các chuyên gia, hoặc là được rút ra từ dữ liệu. Việc trích xuất tri thức từ dữ liệu không còn mới lạ với cộng đồng khoa học hiện nay, và nó cũng trở thành một điểm mạnh của các mô hình đồ thị suy diễn.
Tóm lại, với một mô hình đồ thị xác suất, có ba vấn đề chính ta cần quan tâm:

- *Representation*: cách biểu diễn các điều kiện độc lập dưới dạng đồ thị của các biến ngẫu nhiên (hình dạng của đồ thị)
- *Inference*: cách tính xác suất của một sự kiện dựa trên đồ thị biểu diễn
- *Learning*: cách trích xuất tri thức từ dữ liệu để xây dựng đồ thị

3.2 Đồ thị Bayesian

Trước khi tìm hiểu về mạng Bayesian, ta đề cập lại một số khái niệm của lý thuyết xác suất.
Kí hiệu $A \perp B \mid C$ tức là A độc lập với B khi biết C, ở đây A, B, C đều là các biến ngẫu nhiên.
Ta có

$$A \perp B \mid C \rightarrow P(A, B \mid C) = P(A \mid C)P(B \mid C) \tag{3.1}$$

Phương pháp phân tích chuỗi xác suất Bayesian:

$$P(A_1, A_2, ..., A_n) = P(A_1)P(A_2 \mid A_1)...P(A_n \mid A_1...A_{n-1}) \tag{3.2}$$

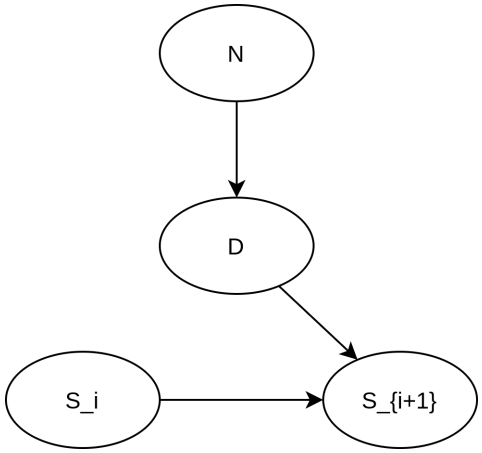
Với cách này, $P(A_1, A_2, ..., A_n)$ được phân tích thành chuỗi các xác suất thành phần, trong đó có $P(A_n \mid A_1...A_{n-1})$. Ta thấy rằng độ phức tạp khi tính $P(A_n \mid A_1...A_{n-1})$ không hề thua kém $P(A_1, A_2, ..., A_n)$, và sẽ tốn kém hơn nếu tính cả các nhân tử khác. Nhưng trong trường hợp giữa các biến tồn tại các điều kiện độc lập, mạng Bayesian có thể giúp làm đơn giản rất nhiều trong việc biểu diễn các phân phối trên.
Hãy xem xét một trường hợp cụ thể như sau. Một hệ thống thực hiện công việc được gửi tới từ người dùng. Việc thực thi các công việc đó làm trạng thái của hệ thống thay đổi. Tại thời điểm t_i, ta quan sát được trạng thái hệ thống là S_i, và dùng thông tin đó để lập lịch cho N công việc đang đợi ở hàng chờ. Thời gian lập lịch cho N công việc đó là D. Sau khi lập lịch xong, tại thời điểm $t_{i+1} = t_i + D$, ta muốn biết trạng thái của hệ thống S_{i+1} có thay đổi hay không. Trong bài toán này, tồn tại một số điều kiện xác suất sau:

- Trạng thái S_{i+1} chỉ phụ thuộc vào trạng thái S_i trước đó và khoảng thời gian giữa 2 lần quan sát
- Thời gian lập lịch D chỉ phụ thuộc vào số lượng công việc cần lập lịch

3.2.1 Representation

Ta sẽ biểu diễn bài toán trên dưới dạng một đồ thị Bayesian. Đồ thị Bayesian là đồ thị có hướng không chu trình có các tính chất sau:

- Mỗi đỉnh biểu diễn cho một biến ngẫu nhiên
- Cạnh biểu diễn cho sự phụ thuộc giữa các biến ngẫu nhiên



Hình 3.2: Mô hình đồ thị Bayesian

Đồ thị Bayesian như trong hình 3.2 giúp ta biểu diễn một số điều kiện độc lập sau:

- $S_{i+1} \perp N \mid D$
- $S_i \perp \{D, N\}$

Từ đó, công thức chuỗi Bayesian cho các biến này được viết gọn như sau:

$$\begin{aligned} P(N, D, S_i, S_{i+1}) &= P(N) \times P(D \mid N) \times P(S_i \mid D, N) \times P(S_{i+1} \mid S_i, D, N) \\ &= P(N) \times P(D \mid N) \times P(S_i) \times P(S_{i+1} \mid S_i, D) \end{aligned}$$

Ta có thể thấy, các phân phối thành phần trở lên đơn giản hơn rất nhiều so với phân phối chung ban đầu, nhờ vậy mà việc tính toán xác suất cũng trở lên dễ dàng hơn.

3.2.2 Inference

Inference là quá trình trả lời các truy vấn liên quan đến xác suất xuất hiện của các sự kiện trong mạng Bayesian. Ví dụ, tập trạng thái của hệ thống là $S = \{s_1, s_2, s_3\}$, ta quan sát được có N = n_0 công việc sẽ được lập lịch, trạng thái hiện tại $S_i = s_1$, xác suất mà hệ thống không đổi trạng thái sau thời gian lập lịch là bao nhiêu?
Với truy vấn trên, ta cần tính:

$$P(S_{i+1} = s_1 \mid S_i = s_1, N = n_0)$$

Gọi $\mathcal{D}$ là không gian sự kiện của biến D. Ta có thể phân tích như sau:

$$\begin{aligned} P(S_{i+1} = s_1 \mid S_i = s_1, N = n_0) &= \sum_{d \in \mathcal{D}} P(S_{i+1} = s_1, D = d \mid S_i = s_1, N = n_0) \\ &= \sum_{d \in \mathcal{D}} P(D = d \mid S_i = s_1, N = n_0) \times P(S_{i+1} = s_1 \mid D = d, S_i = s_1, N = n_0) \\ &= \sum_{d \in \mathcal{D}} P(D = d \mid N = n_0) \times P(S_{i+1} = s_1 \mid D = d, S_i = s_1) \end{aligned}$$

Dựa vào đồ thị, ta có thể tính được $P(D \mid N), P(S_{i+1} \mid S_i, D)$ và từ đó trả lời truy vấn bên trên.
Ngoài ra, còn có thể tìm xem S_{i+1} có khả năng nhận giá trị nào nhất, dựa vào truy vấn MAP (Maximum a Posteriori).

$$MAP(S_{i+1} \mid S_i = s_1, N = n_0) = \underset{s}{argmax} P(S_{i+1} = s \mid S_i = s_1, N = n_0) \quad (3.3)$$

Truy vấn trên đồ thị xác suất còn có một ưu điểm là ta có thể dựa vào cấu trúc dữ liệu đồ thị để giảm độ phức tạp trong quá trình tính toán truy vấn, điển hình như thuật toán Belief Propagation[31].

3.2.3 Learning

Trong nhiều trường hợp, con người không thể cung cấp được "kiến thức" cho đồ thị do khối lượng quá lớn và cần đòi hỏi sự hiểu biết về nhiều lĩnh vực, chuyên ngành. Ngoài ra, các phân phối của các biến có thể thay đổi theo thời gian. Khi đó, một giải pháp hiệu quả là chúng ta cho mô hình đồ thị học, trích rút tri thức từ bộ dữ liệu được sinh ra.
Trong bài toán ví dụ, khi ta muốn tính xác suất $P(N, D, S_i, S_{i+1})$, ta cần biết được giá trị của các thành phần $P(N)$, $P(D \mid N)$, $P(S_i)$, $P(S_{i+1} \mid S_i, D)$, có thể hiểu là các tham số của mạng. Các giá trị này sẽ được cung cấp bởi những chuyên gia, hoặc sẽ được học từ dữ liệu.
Bài toán học trong đồ thị xác suất bản chất giống như một bài toán tối ưu. Xét $\Omega = \{x_1, x_2, ..., x_m\}$ là tập dữ liệu, θ là vector tham số của mạng, Θ là tập giá trị của θ. Ta cần phải tìm giá trị θ^* sao cho likelihood của bộ dữ liệu đạt cực đại, tức là:

$$\theta^* = arg \underset{\theta \in \Theta}{max} P(\Omega \mid \theta) \quad (3.4)$$

Các phương pháp học trong mạng Bayesian đã được trình bày chi tiết trong *Probabilistic Graphical Model: Principles and Techniques*[30], nếu có thời gian xin hãy xem đến cuốn sách này.

4

Mô hình lập lịch thời gian thực sử dụng mạng Bayesian ước lượng tài nguyên

4.1 Bài toán thực tế

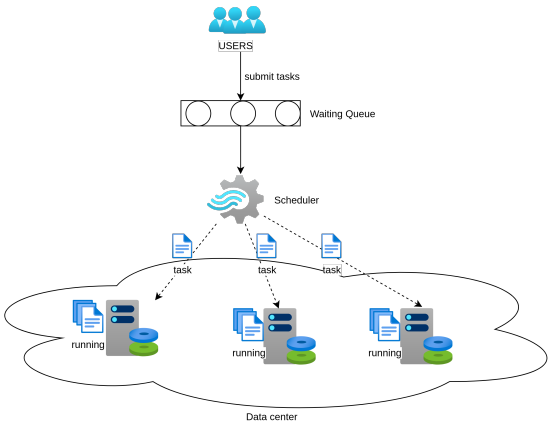
Như đã trình bày trong 1.3, người dùng gửi các tasks đến cloud liên tục theo thời gian. Trong thực tế, người dùng sẽ gửi đến hệ thống các công việc dưới dạng các job, là tập các tasks có thể chạy độc lập với nhau. Do các tasks có thể chạy độc lập, nên ta coi như người dùng gửi các tasks đến hệ thống cùng một thời điểm, và chúng có quan hệ là có chung jobID (mã công việc người dùng gửi đến). Các tasks này sẽ được chia thành 2 loại[12]:

- *long-run*: các công việc mà không có thời điểm kết thúc, luôn chạy để xử lý những nhu cầu ngắn hạn của người dùng. Ví dụ: Gmail, Docs, web search, ...
- *batch-job*: các công việc có thời gian chạy cụ thể, thường từ vài giây đến vài ngày, dao động tùy thuộc vào hiệu năng của hệ thống. Ví dụ: người dùng gửi đến công việc huấn luyện một mô hình học máy, hoặc chạy một script phân tích dữ liệu, ...

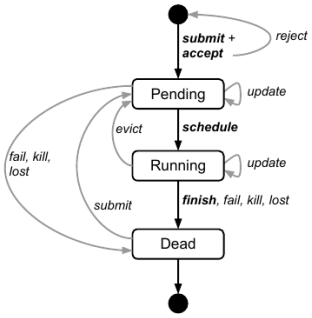
Với kiểu lập lịch batch mode, các tasks phải đợi ở hàng chờ, rồi hệ thống lập lịch chạy cho tasks, và sau đó thì được đưa đến các máy tính ảo tương ứng vào thực thi công việc. Hình 4.1 mô tả quá trình chuyển trạng thái của task trong thực tế. Với các giải thuật lập lịch có sẵn, có 2 vấn đề chưa được khắc phục trong mô hình thực tế, đó là sự sai lệch về các thông số do độ trễ của hệ thống, và sự mất tính load-balancing khi các công việc batch-job kết thúc còn các công việc long-running vẫn đang chạy tập trung trên một vài máy ảo.

4.1.1 Sai lệch về thông tin do độ trễ của hệ thống

Có thể thấy rằng trong khi chạy, task có thể bị fail và được đưa trở về hàng đợi đến lần lập lịch tiếp theo, hoặc cũng có thể kết thúc và rời khỏi máy tính



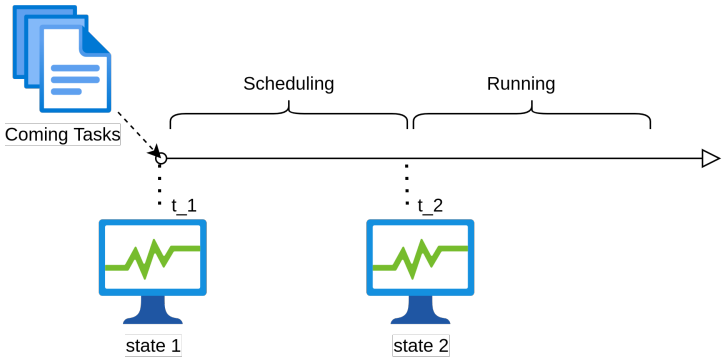
(a) Luồng hoạt động của hệ thống



(b) Trạng thái của task

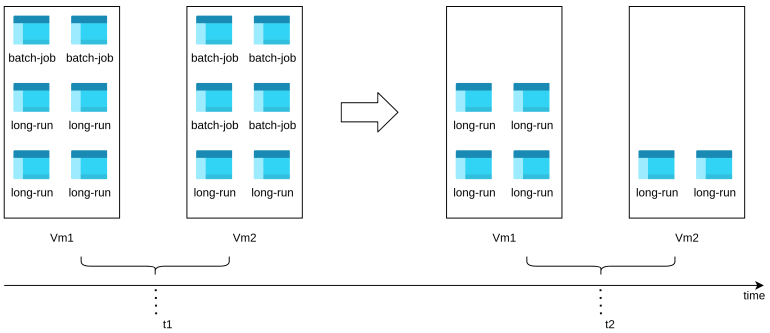
Hình 4.1: Luồng di chuyển của task

ảo. Mỗi khi điều này xảy ra, tài nguyên của hệ thống được giải phóng và có thể phân phối cho các tasks khác. Điều này khiến cho thông tin môi trường sẽ thay đổi liên tục trong khi hoạt động. Như vậy, các thông số của hệ thống mà được sử dụng trong quá trình lập lịch có thể không thống nhất với thời điểm chạy, khiến cho nghiệm tối ưu của bài toán lập lịch không tốt như mong muốn trong quá trình chạy.



Hình 4.2: Trạng thái máy tính tại thời điểm lập lịch và thực thi

Cụ thể hơn, xét tại thời điểm t_1, các tasks trong hàng đợi được mang ra lập lịch, scheduler sẽ lấy các thông số của hệ thống tại thời điểm đó là $state_1$. Quá trình lập lịch trong môi trường phân tán cần lấy thông tin từ các thành phần khác nhau, chạy thuật toán và phân phối tasks đến các máy ảo tương ứng. Điều này khiến cho thời điểm các tasks bắt đầu được thực thi trên máy ảo là t_2 sẽ trễ hơn so với thời điểm t_1 một lượng δ_t đáng kể (hình 4.2). Như vậy, trong thời gian lập lịch thì các task đang chạy trong các máy ảo có thể được hoàn thành, bị fail hoặc bị hoãn, dẫn đến khả năng $state_2 \neq state_1$. Mà quyết định lập lịch là nghiệm bài toán tối ưu thu được dựa trên thông số $state_1$ nên sự chênh lệch này sẽ phá vỡ tính tối ưu của thuật toán lập lịch.



Hình 4.3: Mất cân bằng giữa các task do sự kết thúc của batch-job tasks

4.1.2 Mất cân bằng tải

Với các tasks được gửi từ người dùng, ví dụ như trong hệ thống máy chủ của Google[11], không thể xác định được execution time với các thông tin tại thời điểm lập lịch. Do đó, các thuật toán lập lịch quan tâm chính đến vấn đề tài nguyên khả dụng, mà chưa xem xét đến thời điểm kết thúc của tasks. Điều này có thể gây ra sự mất cân bằng load-balancing trong quá trình hoạt động. Trong hình 4.3, các tasks thuộc dạng batch-job kết thúc trước khiến cho trạng thái load-balancing của hệ thống bị phá vỡ.

4.2 Mô hình đề xuất

$\mathcal{V}$ là tập máy ảo trong hệ thống, $\mathcal{V} = \{vm_1, vm_2, ..., vm_M\}$, với

$$vm_i.v = [v_i^1, v_i^2, v_i^3, v_i^4]$$

là vector mang thông tin tài nguyên của máy ảo, lần lượt là cpu, bộ nhớ chính, băng thông mạng và ổ đĩa. Task đến hệ thống sẽ gồm những thông tin sau:

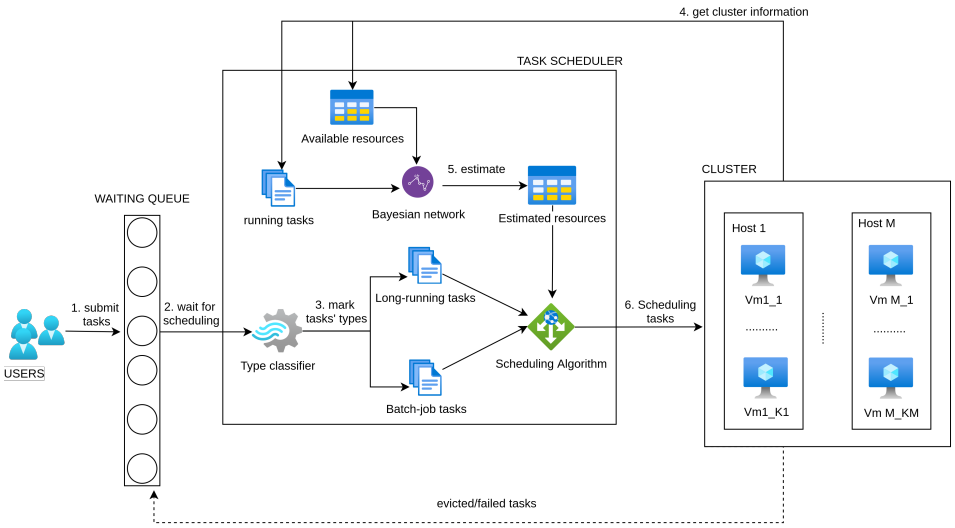
1. $c = [c^1, c^2, c^3, c^4]$ là vector tài nguyên task yêu cầu, lần lượt là cpu, bộ nhớ chính, băng thông mạng và ổ đĩa
2. meta_data bao gồm thông tin về người dùng, application gửi task đến, mức độ ưu tiên
3. lib: rằng buộc về thư viện, môi trường thực thi

Lập lịch là công việc tìm hàm ghép cặp task với máy ảo

$$f : \mathcal{C} \to \mathcal{V} \tag{4.1}$$

với $\mathcal{C}$ là không gian các tasks.
Với mỗi máy tính ảo vm_i, $r_i = \{task_{i1}, ..., task_{ik}\}$ là tập các tasks đang được chạy trong nó. Trong khi lập lịch, ta cần đảm bảo được điều kiện tài nguyên của máy tính phải lớn hơn tổng nhu cầu sử dụng của các tasks:

$$vm_i.v \geq \sum_{j=1}^{k} task_{ij}.c \tag{4.2}$$



Hình 4.4: Mô hình lập lịch

Trong trường hợp tài nguyên không đủ, các tasks có mức ưu tiên thấp hơn sẽ bị dừng lại hoặc hủy và trả về hàng đợi để nhường tài nguyên cho các tasks còn lại. Hình 4.4 là mô hình một lịch được đề xuất, bao gồm:

1. Người dùng gửi các công việc đến hệ thống, được chuyển đến hàng đợi cho việc lập lịch
2. Các tasks chờ đợi đến khi đủ số lượng hoặc vượt quá một khoảng thời gian cố định
3. Dựa vào thông tin metadata của tasks, ta phân loại công việc thành 2 kiểu là long-running và batch-job
4. Bộ lập lịch lấy các thông tin về trạng thái hệ thống, bao gồm các tasks đang được chạy trong hệ thống và thông tin về tài nguyên của các máy tính ảo
5. Sử dụng mạng Bayesian để ước lượng tài nguyên khả dụng tại thời điểm tasks được thực thi trên các máy ảo
6. Lập lịch cho các tasks với thông tin tài nguyên ước lượng

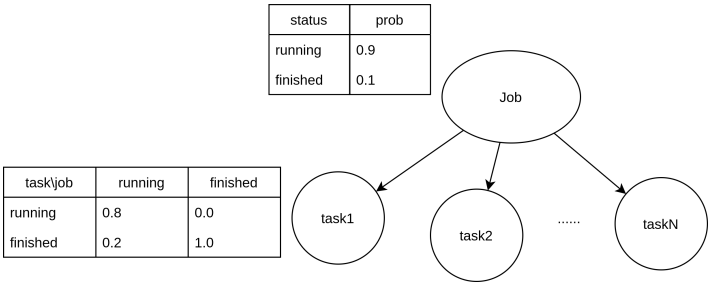
Phân loại tasks

Việc nhận đinh một task thuộc dạng long-running hay batch-job không phải là dễ dàng. Điều quyết định đến thời gian chạy của công việc là người dùng, họ có thể dừng các dịch vụ một cách tùy ý họ muốn, nên đôi khi long-running chưa hẳn thời gian hoạt động đã đủ "long" (dài). Mặc dù vậy, trong [12], tác giả sử dụng thông tin về ứng dụng mà gửi task đến hệ thống để phân loại. Các tasks được gửi từ Gmail, Google Docs, Google Slide, ... sẽ được xem là long-running. Còn với những tasks từ Google Data Analytic, Google Auto ML, ... thì là batch-jobs. Việc phân chia như vậy là dựa vào đặc thù từng ứng dụng, các ứng dụng long-running sẽ là các application tương tác liên tục với người dùng,

còn batch-job thì sau khi gửi công việc thì sẽ ko cần tương tác cho đến khi kết thúc.

4.2.1 Dự đoán tài nguyên khả dụng

Với các thuật toán trước đó, ta mặc định $state_2 = state_1$ (hình 4.2), tức là các tasks đang chạy trong các máy ảo sẽ không thay đổi trạng thái trong quá trình lập lịch. Nhưng trong thực tế không phải như vậy, việc lập lịch kéo dài tới vài giây, hoặc vài phút, tùy theo số lượng tasks và tốc độ truyền tin trong mạng. Một batch-job task có thời gian từ vài giây đến vài ngày [12][11], như vậy việc tasks đang chạy trong hệ thống thay đổi trạng thái trong giai đoạn lập lịch làm $state_2 \neq state_1$ xảy ra rất thường xuyên. Như đã đề cập ở đầu chương, việc sai lệch thông tin này khiến cho lời giải của thuật toán không còn tối ưu như kỳ vọng.
Từ giờ, ta sẽ gọi các tasks đang chạy trong các máy ảo là running-tasks, các tasks đang ở hàng đợi là waiting-tasks. Mục tiêu của ta là ước lượng trạng thái của các running-tasks tại thời điểm các waiting-tasks bắt đầu được thực thi (t_2 trong hình 4.2), và từ đó ước lượng số lượng tài nguyên khả dụng của hệ thống để phục vụ cho việc lập lịch tại t_1.
Phương pháp ước lượng được đề xuất là sử dụng mô hình Bayesian tìm mối quan hệ giữa các tasks thuộc cùng một job để đưa ra phán đoán về trạng thái. Trong hình 4.5, trạng thái mỗi task là một biến ngẫu nhiên được biểu thị bởi



Hình 4.5: Mạng Bayesian thể hiện quan hệ giữa các tasks trong cùng một job

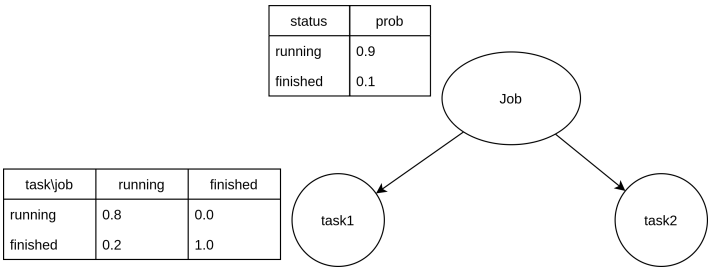
một đỉnh trong đồ thị, với tập giá trị là running (trong trường hợp task đang được thực thi) và finished (trong trường hợp tài nguyên dành cho task được thu hồi). Các tasks trong cùng một job đều phụ thuộc vào job như trong hình 4.5. Trong bảng phân phối xác suất của task, con số 0.8 ý nghĩa

$$P(task = running \mid job = running) = 0.8$$

là xác suất task đang chạy khi biết job vẫn đang chạy.
Với đồ thị trên, ta có:

$$\begin{aligned} P(task) &= \sum_{s \in \{running, finished\}} P(task, job = s) \\ &= \sum_{s \in \{running, finished\}} P(job = s) P(task \mid job = s) \end{aligned}$$

Từ đó, ta tính được P(task = running) = 0.72 và P(task = finished) = 0.28. Sau đây, ta sẽ mô tả cách thức chúng ta dùng đồ thị Bayesian để suy diễn trạng thái của các tasks. Xét với trường hợp job có 2 tasks như hình 4.6.



Hình 4.6: Mạng Bayesian cho trường hợp có 2 tasks

Giả sử ta quan sát được task1 đã kết thúc, tức là trạng thái của nó là finished, khi đó phân phối xác suất của task2 sẽ như thế nào?
Ta sẽ tính

$$\begin{aligned} P(task2 \mid task1 = finished) &= \sum_{job} P(task2, job \mid task1 = finished) \\ &= \sum_{job} \frac{P(task2, job, task1 = finished)}{P(task1 = finished)} \end{aligned}$$

Trong đồ thị Bayesian 4.6, ta phân tích được:

$$P(task2, job, task1) = P(job) \times P(task1 \mid job) \times P(task2 \mid job)$$

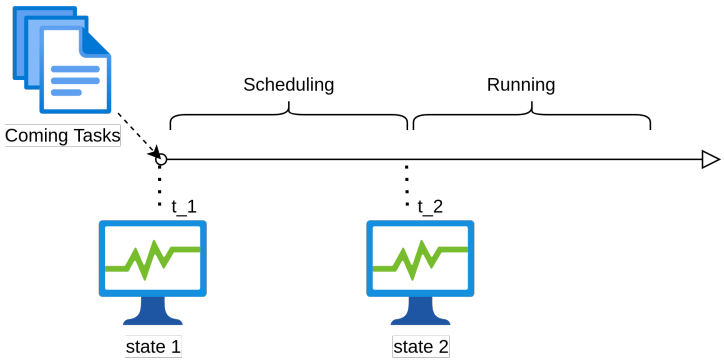
Suy ra,

$$P(task2 \mid task1 = finished) = \sum_{job} \frac{P(job)P(task2 \mid job) \times P(task1 = finished \mid job)}{P(job) \times P(task1 = finished \mid job)}$$

Thay các giá trị vào, tính được:

$$\begin{aligned} P(task2 = running \mid task1 = finished) &= 0.514 \\ P(task2 = finished \mid task1 = finished) &= 0.486 \end{aligned}$$

Chú ý rằng bởi vì quan sát được task1 kết thúc mà xác suất để task2 kết thúc tăng từ 0.28 lên thành 0.486, phù hợp với thực tế rằng khi có một task kết thúc các task còn lại trong job tăng khả năng kết thúc sau đó.
Như vậy, làm thế nào để có được P(task | job) và P(job) tại một thời điểm ngẫu nhiên trong quá trình chạy. Trong đồ án này, ta ghi lại các trạng thái của các thành phần trong hệ thống tại thời điểm đầu lập lịch và kết thúc lập lịch, sau đó tính tỉ lệ giữa các trạng thái. Ta đã biết với phân phối xác suất dạng bảng như trên thì tỉ lệ giữa các giá trị quan sát của biến ngẫu nhiên chính là nghiệm



Hình 4.7: Trạng thái máy tính tại thời điểm lập lịch và thực thi

của hàm tối ưu likelihood trên bộ dữ liệu (trang 719, [30]).

Trong hình 4.7, tại thời điểm t_1 khi bắt đầu lập lịch, ta sẽ ước lượng trạng thái tài nguyên (đã dùng và chưa sử dụng) tại thời điểm kết thúc lập lịch khi mà các tasks được thực thi trên các máy ảo, t_2. Cụ thể, ta sẽ tính kỳ vọng của tài nguyên được sử dụng của từng máy ảo tại thời điểm t_2.

Algorithm 1 Estimate usage of vms

Input: $vms : List < vm >$

Output: $List < vector >$ - list với phần tử là vector tài nguyên khả dụng của các máy ảo

1: $availables \leftarrow List < vector >$
2: **for** $vm \in vms$ **do**
3: $\quad remainder \leftarrow vm.v$
4: $\quad$ **for** $task \in vm.runningTasks$ **do**
5: $\quad\quad prob \leftarrow BayesianNetwork.$**inference**$(task)$
6: $\quad\quad remainder \leftarrow remainder - prob \times task.c$
7: $\quad$ **end for**
8: $\quad availables.add(remainder)$
9: **end for**
10: **return** $availables$

Trong thuật toán 1, ở dòng số 5, ta dùng thuật toán Belief Propagation [31] trên mạng Bayesian để ước lượng P(task is running at t_2), và kỳ vọng tài nguyên sử dụng sẽ là xác suất task còn đang chạy (tức còn chiếm tài nguyên) nhân với tài nguyên task sử dụng. Nên nhớ rằng công việc ước lượng này là một phần của thuật toán lập lịch, nên thời gian chênh lệch

$$\delta_t = t_2 - t_1 = time_{inference} + time_{matching} + network\_delay \tag{4.3}$$

có thể ước lượng được. Trong những trường hợp thời gian chênh lệch không đáng kể, $\delta_t < \epsilon$, ví dụ như số lượng task cần được lập lịch rất ít, ta sẽ không thực hiện ước lượng vì sự sai lệch không đáng kể.

4.2.2 Thuật toán lập lịch đề xuất

Sau khi ước lượng được tài nguyên khả dụng của hệ thống tại thời điểm kết thúc lập lịch, ta cần ghép cặp giữa tasks và máy ảo. Mục tiêu của thuật toán ghép cặp là ổn định tính load-balancing trong quá trình chạy, giải quyết vấn đề được đưa ra trong mục 4.1.2. Ý tưởng tiếp cận là ta sẽ cố gắng cân bằng tài nguyên sử dụng giữa các máy ảo và cân bằng lượng tài nguyên phân phối cho batch-job giữa các máy ảo.
Trước tiên, ta đề xuất thuật toán cân bằng khối lượng công việc giữa các máy ảo 2. Xét

$$\mathcal{L} = \{l_1, l_2, ..., l_M\} \tag{4.4}$$

là tập biểu diễn khối lượng công việc của M máy ảo. Ta coi độ mất cân bằng của hệ thống chính là phương sai tập $\mathcal{L}$:

$$\sigma(\mathcal{L}) = \frac{1}{M} \times \sum_{i=1}^{M} (l_i - \bar{l})^2 \tag{4.5}$$

với

$$\bar{l} = \frac{1}{M} \sum_{i=1}^{M} l_i$$

Đầu vào của thuật toán sẽ là tập các tasks với khối lượng công việc $\mathcal{C} = \{c_1, ..., c_N\}$, ta sẽ đưa ra cách phân phối chúng đến máy ảo sao cho sự mất cân bằng của hệ thống là nhỏ nhất. Trọng số của một cách ghép cặp $task_i \rightarrow vm_j$ là chênh lệch của $\sigma(\mathcal{L})$ trước và sau khi ghép cặp:

$$\Delta_L(task_i \rightarrow vm_j) = \frac{1}{M}[(l_j + c_i - \overline{l'})^2 - (l_i - \bar{l})^2] \tag{4.6}$$

với

$$\overline{l'} = \frac{1}{M}(\sum_{k=1}^{M} l_k + c_i)$$

Trong thuật toán 2, ta thực hiện vòng lập đến khi tất cả các tasks đều được lập lịch. Trong mỗi vòng lặp, ta sẽ tính ma trận Δ, với Δ_{ij} là trọng số của cách ghép $task_i \rightarrow vm_j$ (dòng 3). Ta thực hiện gán cặp task, máy tính ảo sao độ mất cân bằng là nhỏ nhất, sau đó sẽ tăng thêm giá trị khối lượng công việc cho cái máy tính vừa được lập lịch. Đây là một thuật toán heuristic, với độ phức tạp là $O(MN^2)$ do sau mỗi lần ghép cặp thì trong số của tất cả các phép gán tiếp theo sẽ bị thay đổi, nên ta lại phải cập nhật lại.

Algorithm 2 Load-balancing scheduling algorithm

Input:

$\mathcal{L} = \{l_1, l_2, ..., l_M\}$

$\mathcal{C} = \{c_1, ..., c_N\}$

Output:

Cách ghép cặp cho task với vm

1: $solution = \{\}$
2: **while** $|solution| \neq N$ **do**
3: $\quad \Delta \leftarrow$ **calculate_unbalanced_factor**$(\mathcal{L}, \mathcal{C})$
4: $\quad (i, j) = argmin(\Delta)$
5: $\quad solution \rightarrow solution \cup (task_i \rightarrow vm_j)$
6: $\quad C \leftarrow C \setminus \{c_i\}$
7: $\quad l_j \leftarrow l_j + c_i$
8: **end while**
9: **return** *solution*

Quay trở lại bài toán lập lịch thời gian thực, với máy ảo vm_i, vector $vm_i.v = [v_i^1, v_i^2, v_i^3, v_i^4]$ biểu diễn tài nguyên của máy tính, $r_i = \{task_{i1}, ..., task_{ik}\}$ là tập các tasks đang được chạy trong nó.
Với $task_{ij}$,

$$task_{ij}.type = \begin{cases} 0, & \text{nếu } task_{ij} \text{ thuộc kiểu long-running} \\ 1, & \text{nếu } task_{ij} \text{ thuộc kiểu batch-job} \end{cases}$$

Ta xem xét tài nguyên đã được sử dụng là tổng của những phần dành cho long-running tasks với những phần dành cho batch-job tasks. Dựa vào thuật toán 1, ta tính kỳ vọng lượng tài nguyên mà các batch-job tasks sử dụng tại thời điểm kết thúc lập lịch là:

$$vm_i.EshortUsage = \sum_{j=1}^{k} task_{ij}.c \times task_{ij}.type \times prob_{task_{ij}} \tag{4.7}$$

$vm_i.longUsage$ là tài nguyên của máy ảo dành cho long-running tasks.

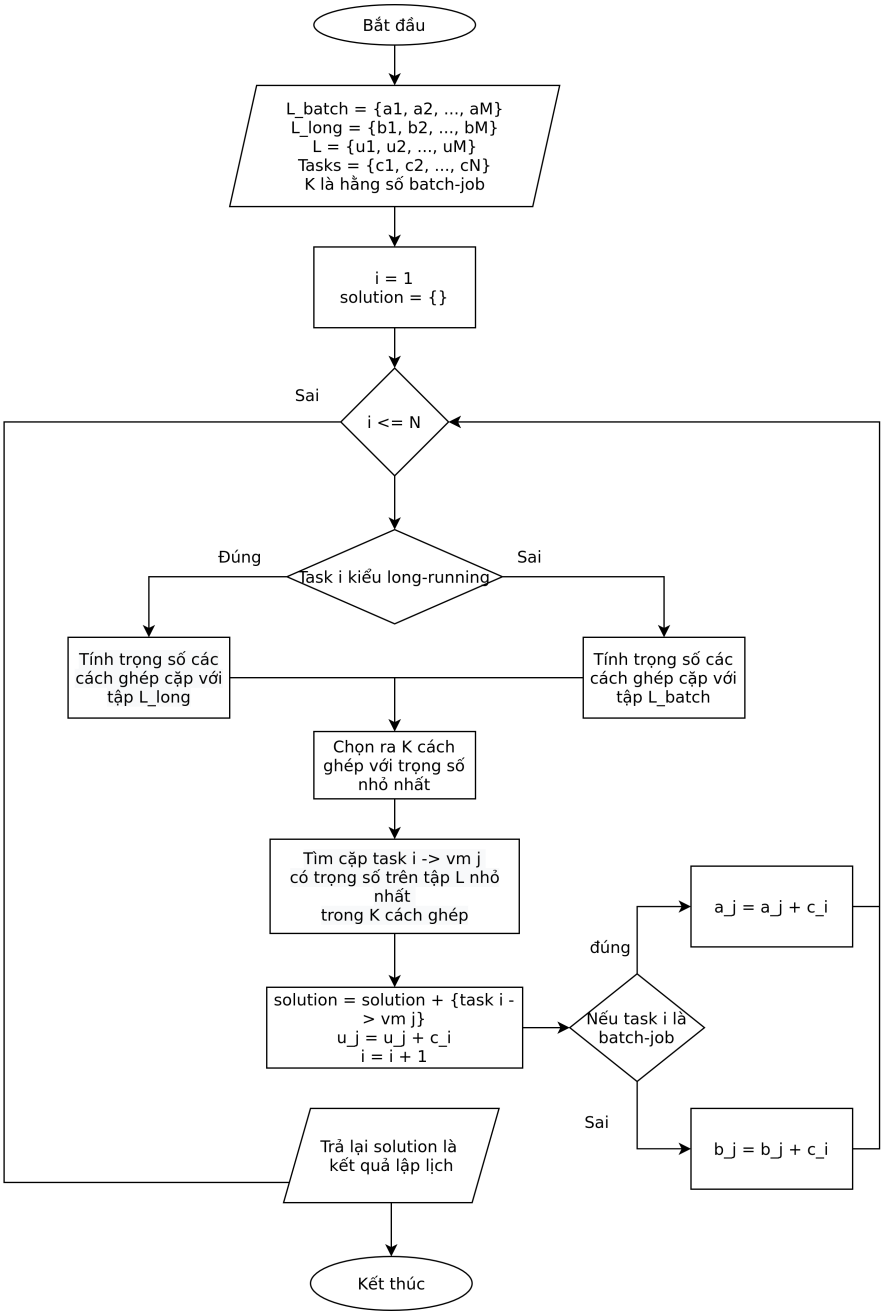
$$vm_i.longUsage = \sum_{j=1}^{k} task_{ij}.c \times (1 - task_{ij}.type) \tag{4.8}$$

$vm_i.usage$ là tài nguyên được sử dụng tại thời điểm kết thúc lập lịch,

$$vm_i.Eusage = vm_i.EshortUsage + vm_i.longUsage \tag{4.9}$$

Ta tính được 2 tập hợp biểu diễn khối lượng công việc giữa các máy tính ảo:

- $\mathcal{L}_{batch} = \{vm_1.EshortUsage, ..., vm_M.EshortUsage\}$ là tập khối lượng tài nguyên được sử dụng bởi batch-job tasks
- $\mathcal{L}_{long} = \{vm_1.longUsage, ..., vm_M.longUsage\}$ là tập khối lượng tài nguyên được sử dụng bởi long-running tasks
- $\mathcal{L} = \{vm_1.Eusage, ..., vm_M.Eusage\}$ là tập khối lượng toàn bộ tài nguyên được sử dụng



Hình 4.8: Thuật toán lập lịch cân bằng giữa long-running và batch-job

Thuật toán đề xuất được mô tả trong hình 4.8, hướng tới việc cân bằng lượng tài nguyên phân phối cho batch-job, đồng thời cân bằng tổng lượng tài nguyên giữa các máy ảo, được mô tả như sau:

Các tasks ở trong hàng đợi được xếp theo thứ tự thời gian đến, ưu tiên các task có thời gian đến sớm sẽ được lựa chọn máy tính trước. Lần lượt duyệt qua các task trong hàng đợi,

1. Nếu task kiểu long-running
    - Tính $\Delta_{\mathcal{L}_{long}}$ là ma trận trọng số với tập $\mathcal{L}_{long}$
    - Chọn ra tập S gồm K cặp $task \rightarrow vm$ có trọng số nhỏ nhất trong $\Delta_{\mathcal{L}_{long}}$
2. Nếu task kiểu batch-job
    - Tính $\Delta_{\mathcal{L}_{batch}}$ là ma trận trọng số với tập $\mathcal{L}_{batch}$
    - Chọn ra tập S gồm K cặp $task \rightarrow vm$ có trọng số nhỏ nhất trong $\Delta_{\mathcal{L}_{batch}}$
3. Tính ma trận trọng số Δ_K của S với $\mathcal{L}$
4. Tìm cặp $task \rightarrow vm$ là $argmin\Delta_K$
5. Thêm cặp $task \rightarrow vm$ vào lời giải và cập nhật lại khối lượng tài nguyên được sử dụng

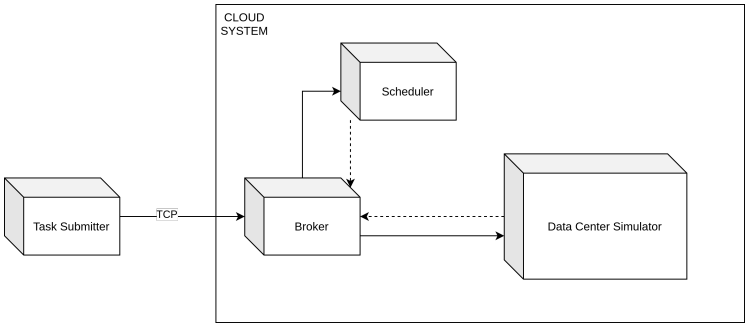
Điểm khác biệt của thuật toán này với thuật toán 2 là ở việc ta cân bằng cho lượng tài nguyên được phân phối cho batch-job tasks.
Thuật toán đề xuất tìm K lời giải vừa tối ưu cho sự cân bằng batch-job tasks, vừa cho long-running tasks giữa các máy ảo, rồi tìm lời giải tốt nhất để cân bằng toàn bộ tài nguyên của hệ thống trong K nghiệm trên.

5

Thực nghiệm và đánh giá

5.1 Mô hình mô phỏng

Để đánh giá hiệu năng của thuật toán, ta sử dụng công cụ mô phỏng CloudSim [13] trên máy tính 4 core, 2.5GHz, 8.0G ram, hệ điều hành Ubuntu 20.04, sử dụng IntelliJ với java 14. CloudSim là công cụ mô phỏng môi trường Cloud được sử dụng rộng rãi trong các nghiên cứu về lĩnh vực này như [32], [33].



Hình 5.1: Mô hình mô phỏng

Các thành phần mô phỏng (hình 5.1) bao gồm:

- **Task Submitter**: thành phần mô phỏng quá trình gửi tasks từ người dùng tới hệ thống cloud. Các tasks sẽ được gửi đi dưới dạng dữ liệu streaming.
- **Broker**: thành phần mang chức năng làm trung gian giữa người dùng với hệ thống máy chủ, tiếp nhận tasks từ người dùng và chuyển chúng tới các máy tính ảo. Trong đồ án này, để phù hợp với xu hướng bùng nổ dữ liệu, Broker sử dụng nền tảng công nghệ Kafka để nhận các tasks được gửi tới, sau này có thể thực hiện thực nghiệm lập lịch phân tán. Để lựa chọn máy tính phù hợp cho tasks, Broker sử dụng bộ Scheduler thực hiện thuật toán lập lịch.
- **Scheduler**: bộ lập lịch có nhiệm vụ ghép cặp giữa các tasks vào máy ảo.

- **Data Center Simulator**: bộ mô phỏng quá trình hoạt động của một hệ thống máy chủ. Được xây dựng trên nền tảng Cloudsim, Data Center Simulator có khả năng mô phỏng thời gian các tasks được thực thi trên các máy tính ảo, đưa ra thời gian thực thi của các tasks cùng với trạng thái tài nguyên của hệ thông trong quá trình mô phỏng.

Các thành phần trên đều được lập trình bằng ngôn ngữ java 14.
Thông số trong quá trình thực nghiệm:

- các tasks được gửi tới hệ thống theo phân phối *Poisson* với trung bình 15 task/second
- thời gian đợi ở hàng chờ tối đa là 20 giây
- chỉ số K trong thuật toán 4.8 bằng 3
- thời gian các tasks được chuyển từ bộ lập lịch đến máy ảo tuân theo phân phối chuẩn với kỳ vọng bằng 5 và độ lệch chuẩn bằng 0.5.

5.2 Dữ liệu mô phỏng

Dưới đây là thông số của 10 máy tính ảo được mô phỏng trong quá trình thực nghiệm.

Bảng 5.1: Bảng mô tả thông tin máy tính ảo

Virtual Machine Information List					
id	core	mips	ram (Mb)	storage (Mb)	bandwidth (Mb/s)
0	4	2000	4096	32768	16
1	4	1000	8012	32768	16
2	4	1000	4096	32768	16
3	4	2000	4096	32768	16
4	4	2000	8012	32768	16
5	4	1000	12288	32768	16
6	4	4000	12288	32768	16
7	4	1000	12288	32768	16
8	4	1000	4096	32768	16
9	4	2000	8012	32768	16

Các dữ liệu về tasks trong quá trình mô phỏng được trích rút ngẫu nhiên từ 300G dữ liệu[11] ghi lại từ hệ thống máy chủ của Google năm 2011. Do bộ dữ liệu của Google không ghi lại các thông số về bandwidth và số lượng core mà task yêu cầu nên ta để mặc định lần lượt là 0 (Mb/s) và 2 (cores).

Bảng 5.2: Tỉ lệ các kiểu công việc

Type percentage	
type	%
long-running	2%
batch-job	98%

Bảng 5.3: Phân phối mức độ ưu tiên của tasks

Priorities percentage	
priority	%
0	90%
1	10%

Bảng 5.4: Tài nguyên yêu cầu

Tasks' description			
stats	cpu request (%)	memory request (Mb)	storage request (Mb)
count	30000	30000	30000
mean	0.051	24.93	11.0
std	0.056	40.35	5.2
min	0.006	1.24	0.1
25%	0.025	20.39	12.6
50%	0.025	26.75	12.6
75%	0.313	26.75	12.6
max	0.251	101.91	63.2

Bảng 5.4 mô tả tài nguyên mà các tasks yêu cầu, bảng 5.2 mô tả phân phối giữa các kiểu của tasks, bảng 5.3 mô tả phân phối về mức độ ưu tiên trong tập dữ liệu.

5.3 Thu thập dữ liệu xây dựng mạng Bayesian

Phân phối xác suất của các đỉnh trong mạng Bayesian sẽ được học từ dữ liệu lịch sử của hệ thống. Ta chia bộ dữ liệu mô phỏng thành 2 phần ngẫu nhiên với tỉ lệ 1:1, gọi là train và test. Công việc thu thập dữ liệu sẽ được thực hiện bằng việc mô phỏng hệ thống với thuật toán Worstfit trên tập train, cụ thể như sau:

1. Task được gửi tới hệ thống theo phân phối possion với trung bình 15 tasks/giây

2. Trong quá trình hoạt động, ghi lại thông tin về các tasks đang chạy trong hệ thống tại thời điểm bắt đầu lập lịch (t_1) và thời điểm gửi các tasks mới tới các máy ảo (t_2).

Tại t_1, ta quan sát một job J có các tasks $\{j_1, ..., j_k\}$ đều kiểu batch-job còn đang chạy. Ta cần tính:

- Xác suất job J còn chạy tại thời điểm t_2
- Xác suất task j_i còn chạy nếu biết J còn chạy tại t_2

Sau khi tính được các tham số cho mạng Bayesian, ta sử dụng nó để chạy thuật toán lập lịch trong quá trình mô phỏng với bộ dữ liệu test.
Trong trường hợp môi trường thực tế, các dữ liệu sẽ được thu thập từ lịch sử hoạt động của hệ thống. Tham số của mạng Bayesian nên được cập nhật sau một khoảng thời gian nhất định để đáp ứng được với sự thay đổi về trạng thái của các tasks trong hệ thống.

5.4 Các tiêu chí đánh giá

Mô hình lập lịch được đề xuất sử dụng mạng Bayesian để ước lượng trạng thái của các tasks trong quá trình lập lịch, từ đó đưa ra quyết định mà thích ứng được với các thay đổi trong quá trình chạy của hệ thống máy chủ. Để đánh giá được hiệu quả của mô hình, ta xem xét các tiêu chí sau:

1. *Số lượng task hoàn thành trong một đơn vị thời gian*: Chỉ số này được dùng nhiều trong các nghiên cứu về thuật toán lập lịch. Với càng nhiều tasks được hoàn thành trong một khoảng thời gian, các chỉ số về thời gian hoàn thành công việc, thời gian chờ từ người dùng cũng sẽ được tối ưu. Bên cạnh đó, một hệ thống chấp nhận được thì số lượng task hoàn thành trong một đơn vị thời gian phải lớn hơn hoặc ngang bằng tỉ lệ task đến hệ thống, do đó, nếu là nhỏ hơn đáng kể thì ta phải xem xét lại độ chính xác của thuật toán hoặc suy nghĩ về việc mở rộng hệ thống.

2. *Độ mất cân bằng khối lượng công việc giữa các máy*: Việc phân phối đều các tasks vào các máy ảo giúp tận dụng được tài nguyên của hệ thống và cân bằng tải trong hoạt hệ thống mạng. Hệ thống mất cân bằng khi phần trăm tài nguyên được sử dụng giữa các máy ảo chênh lệch quá lớn, và trong máy ảo bị sử dụng nhiều thì sẽ xảy ra hiện tượng đói tài nguyên. Việc tính chỉ số này đã được đưa ra trong thuật toán 2. Như trong 4.2.2, việc các tasks thay đổi trạng thái trong quá trình chạy sẽ phá vỡ sự cân bằng khối lượng công việc giữa các máy ảo, do đó, để đánh giá được sự ảnh hưởng của môi trường đến kết quả của thuật toán lập lịch, ta sẽ xem xét chỉ số này thay đổi theo thời gian hệ thống hoạt động.

5.5 Kết quả thực nghiệm

5.5.1 Số lượng task hoàn thành trong một đơn vị thời gian

Kết quả mô phỏng 15000 tasks gửi đến hệ thống trong 1000s được mô mả trong bảng 5.5. Trong 15000 tasks được gửi đến hệ thống, có 300 task là dạng long-running, vì vậy sẽ không kết thúc trong quá trình mô phỏng. Khi số lượng tasks đến lớn hơn số lượng tasks hoàn thành, các tasks chưa hoàn thành sẽ được tích lũy theo thời gian và chiếm tài nguyên của hệ thống. Đến khi tài nguyên không

Bảng 5.5: Kết quả về thời gian chạy của các tasks

Running statistics over 1000s			
stats	FCFS	Worstfit	Resources balancing
count	13214	13925	14235
mean	10.62	6.34	5.42
std	54.24	33.37	27.31
min	0.31	0.12	0.21
25%	3.15	2.09	2.08
50%	5.78	3.52	3.41
75%	8.13	5.61	5.50
max	829.92	616.35	640.39

đủ, các tasks với mức ưu tiên thấp sẽ bị hủy và quay lại hàng chờ để nhường tài nguyên cho các tasks có mức ưu tiên cao. Trong bảng 5.5, ta thấy số lượng tasks hoàn thành của **thuật toán được đề xuất (Resources Balancing)** trong 1000s là cao nhất, 14235 tasks so với 13925 của Worstfit và 13214 của FCFS (First Come First Serve). Thời gian trung bình hoàn thành 1 tasks của Resources Balancing là 5.42 giây, bằng một nửa của FCFS, là giá trị nhỏ nhất trong 3 thuật toán. Trong sự so sánh giữa Resources balancing với Worstfit, ta thấy thống kê min, max thời gian thực thi của thuật toán đề xuất đều lớn hơn, nhưng về mặt trung bình thì lại nhỏ hơn. Điều này là bởi vì Resources balancing vừa cân bằng cho khối lượng batch-job, vừa cho long-running, nên quyết định lập lịch cho các phần tử riêng biệt như task ngắn nhất và dài nhất sẽ không tốt bằng Worstfit - thuật toán tìm máy ảo phù hợp nhất cho cá nhân từng tasks. Nhưng về tổng thể trong suốt quá trình hoạt động, thuật toán đề xuất tỏ ra hiệu quả hơn.

5.5.2 Ảnh hưởng của thời gian trễ tới sai số

Ta đã biết

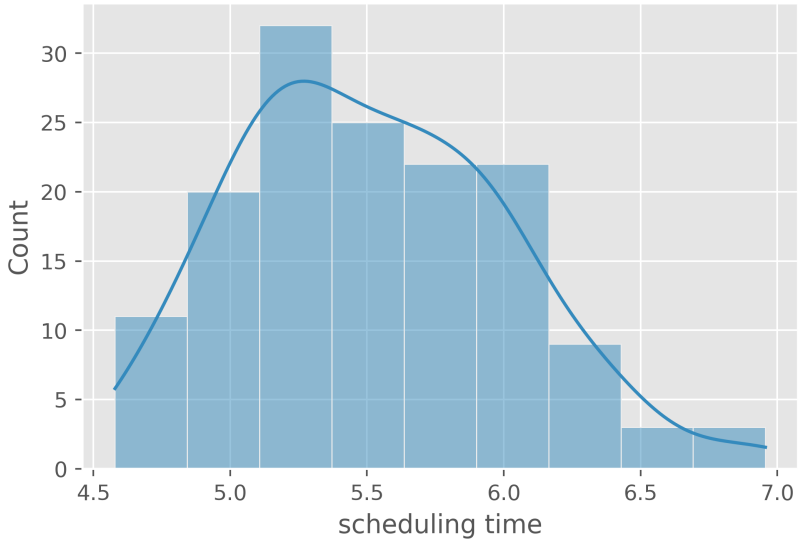
$$\delta_t = t_2 - t_1 = time_{scheduling} + network\_delay$$

là thời gian trễ của thông tin trong quá trình lập lịch, từ thời điểm lập lịch tới thời điểm thực thi. Trong đó:

- $time_{scheduling}$ là thời gian chạy thuật toán lập lịch, phụ thuộc vào độ phức tạp của thuật toán, số lượng máy ảo, số lượng tasks, khả năng tính toán của máy tính thực hiện việc lập lịch.
- $network\_delay$ là thời gian thông tin truyền tin trong mạng, bao gồm việc truyền thông tin trạng thái hệ thống tới bộ lập lịch và việc gửi các tasks đến máy ảo tương ứng để thực thi.

Thông thường, các thuật toán heuristic sẽ có độ phức tạp không quá lớn, nên thời gian lập lịch sẽ không đóng góp nhiều vào tổng thời gian trễ. Tuy nhiên,

trong môi trường dữ liệu lớn, tỉ lệ tasks đến hệ thống được tính bằng hàng triệu tasks một giây, thời gian lập lịch sẽ đóng góp tương đương hoặc hơn so với thời gian truyền tin trong mạng.

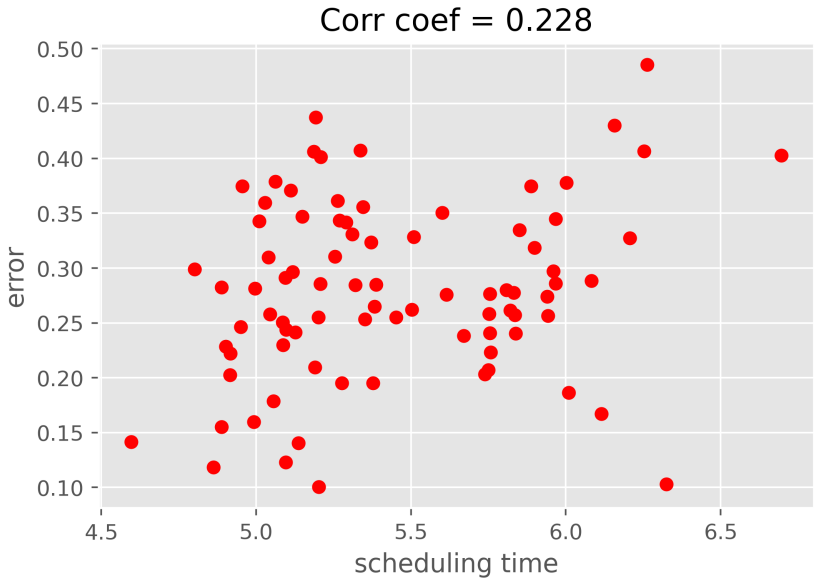


Hình 5.2: Phân phối của độ trễ trong quá trình mô phỏng

Trong hình 5.2, khoảng thời gian trễ từ thời điểm lập lịch với thời điểm thực thi trong quá trình mô phỏng với thuật toán Worstfit được phân bố theo phân phối chuẩn với:

$$E[\delta_t] = 5.5, V[\delta_t] = 0.55^2$$

Với thuật toán Worstfit, ta có hơn 50% tasks có thời gian lập lịch nhỏ hơn 5 giây (trong bảng 5.5), như vậy tại thời điểm lập lịch nếu có N tasks đang chạy trong hệ thống thì sẽ có trung bình lớn hơn N / 2 tasks kết thúc trong quá trình lập lịch.



Hình 5.3: Thời gian trễ ảnh hưởng tới độ sai lệch

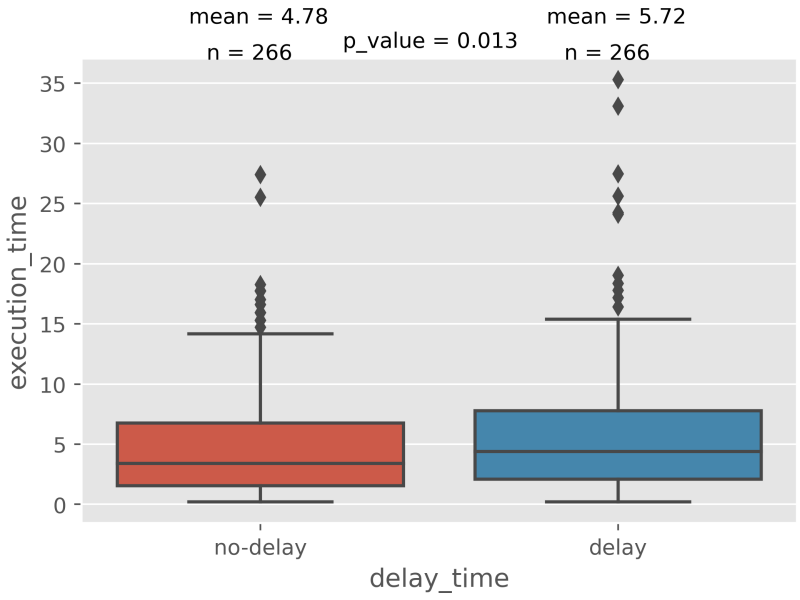
Trong hình 5.3 là thể hiện độ sai lệch của lượng tài nguyên CPU đã sử dụng tại thời điểm lập lịch với thời điểm thực thi. Với hệ số tương quan bằng 0.228, có thể thấy rằng với thời gian trễ càng lâu, sai lệch tại hai thời điểm trên sẽ càng tăng.

5.5.3 Ảnh hưởng của thời gian trễ tới kết quả lập lịch

Liệu rằng thời gian trễ trong quá trình lập lịch có khiến kết quả của việc lập lịch kém hiệu quả hơn không?
Để trả lời cho câu hỏi trên, ta thực hiện thực nghiệm mô phỏng để kiểm chứng xem sự việc dùng thông tin bị trễ để lập lịch có khiến cho thời gian hoàn thành các tasks tăng lên hay không?
Ta sẽ chọn ngẫu nhiên một lượng tasks trong tập dữ liệu trích rút được từ bộ dữ liệu của Google, sau đó tiến hành mô phỏng chúng được lập lịch bằng thuật toán Worstfit với 2 kịch bản:

- Kịch bản 1: Ta giữ thông tin lập lịch không bị trễ bằng cách tạm dừng quá trình mô phỏng trong thời gian lập lịch, sau khi lập lịch xong thì mới tiếp tục mô phỏng. Như vậy, trạng thái của các tasks đang chạy trong hệ thống sẽ không bị thay đổi trong quá trình mô phỏng do chúng bị tạm dừng.
- Kịch bản 2: Quá trình lập lịch sẽ chạy song song với quá trình mô phỏng, như vậy sẽ có sự sai khác giữa thông tin tại thời điểm lập lịch với thời điểm thực thi.

Kết quả thu được trên bộ dữ liệu chọn ngẫu nhiên gồm 266 tasks được mô tả như hình 5.4.



Hình 5.4: Ảnh hưởng của sự trễ với thời gian thực thi

Ta thấy rằng kịch bản thứ nhất (no-delay) có trung bình thời gian thực thi nhỏ hơn so với kịch bản thứ 2 (delay), ứng với số liệu là 4.78 và 5.72. Với giá trị

$pvalue = 0.013$ và mức ý nghĩa $\alpha = 0.05$, ta có thể kết luận rằng việc lập lịch với các thông tin bị trễ khiến cho thời gian thực thi của tasks tăng lên.

5.5.4 Hiệu quả của việc ước lượng tài nguyên

Ta sẽ so sánh sự khác biệt giữa việc sử dụng mạng Bayesian để ước lượng tài nguyên khả dụng với việc dùng thông tin tại thời điểm bắt đầu lập lịch. Xét tại thời điểm lập lịch t_1, với máy tính ảo vm, ta thu được tập các tasks đang chạy là $r = \{task_1, ..., task_k\}$. Tại thời điểm lập lịch hoàn thành và các tasks mới đến được đưa đến vm, t_2, các tasks trong tập r có thể đã hoàn thành và tài nguyên được giải phóng. Gọi $q = \{s_1, ..., s_k\}$ là trạng thái của các tasks trong r tại thời điểm t_2,

$$s_i = \begin{cases} 0, & \text{nếu } task_i \text{ còn chạy tại thời điểm } t_2 \\ 1, & \text{nếu } task_i \text{ kết thúc trong quá trình lập lịch} \end{cases}$$

Với các thuật toán thông thường, khi ta lấy thông tin tại t_1 để lập lịch, tài nguyên khả dụng của máy ảo được tính:

$$hard\_est\_resources = vm.v - \sum_{i=1}^{k} task_i.c \tag{5.1}$$

Nhưng tại thời điểm t_2, lượng tài nguyên thực tế khả dụng là:

$$real\_resources = vm.v - \sum_{i=1}^{k} task_i.c \times s_i \tag{5.2}$$

Do

$$\sum_{i=1}^{k} task_i.c \times s_i \leq \sum_{i=1}^{k} task_i.c$$

nên

$$real\_resources \geq hard\_est\_resources \tag{5.3}$$

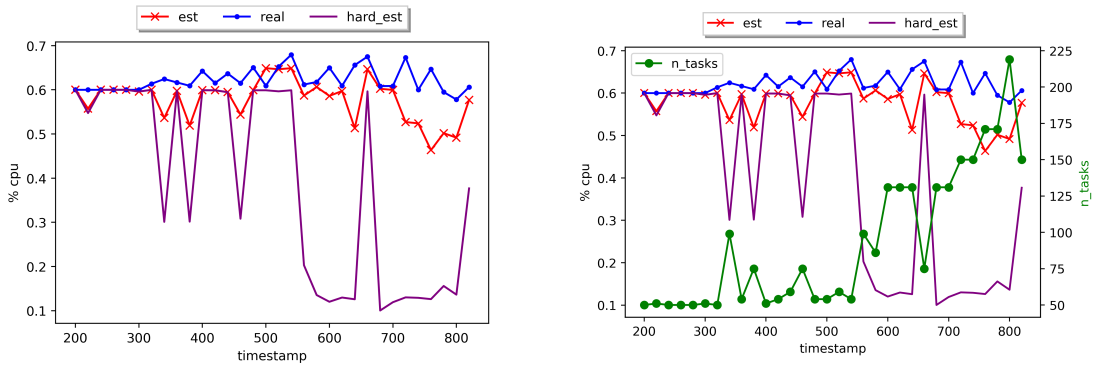
Khi sử dụng mạng Bayesian để tính p_i là xác suất $task_i$ còn chạy tại t_2, tài nguyên ước lượng là:

$$est\_resources = vm.v - \sum_{i=1}^{k} task_i.c \times p_i \tag{5.4}$$

Tương tự, ta có:

$$est\_resources \geq hard\_est\_resources \tag{5.5}$$

Để thấy được rõ ràng sự chênh lệch giữa các thông số tại các thời điểm như trong phương trình 5.3, 5.5, ta ghi lại các kết quả ước lượng được với thông tin thực tế thu được xuyên suốt quá trình mô phỏng. Trong hình 5.5a, đường màu tím biểu diễn cho $hard\_est\_resources$, là thông tin không được ước lượng, đường màu đỏ là thông tin được ước lượng bằng mạng Bayesian, còn đường màu xanh

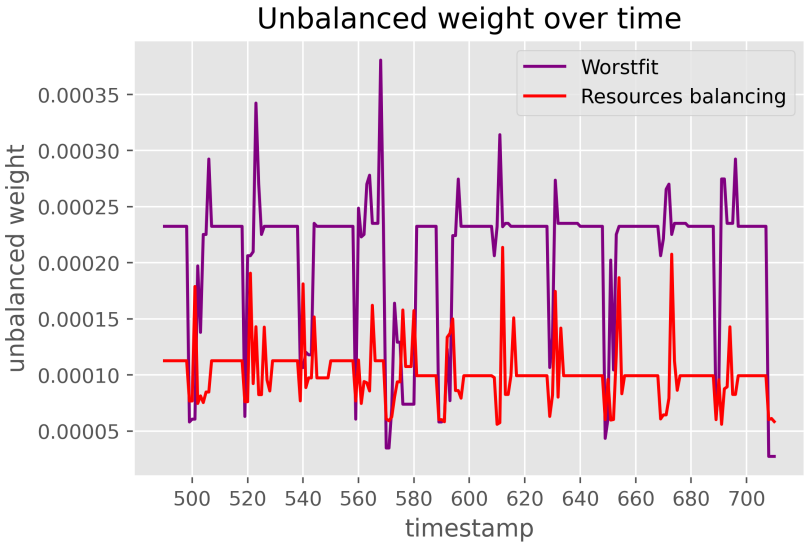


(a) Tài nguyên khả dụng tại thời điểm thực thi (b) Ảnh hưởng của số lượng tasks đang chạy

Hình 5.5: Thông số tại thời điểm kết thúc lập lịch

dương là thông tin tài nguyên thực tế tại thời điểm thực thi. Do thời gian trung bình chạy của các tasks chỉ khoảng 5 giây (bảng 5.5), nên sự thay đổi trạng thái liên tục của các tasks khiến cho việc ước lượng với thực tế bị chênh lệch rất nhiều. Kết quả ước lượng bằng mạng Bayesian dao động xung quanh đường giá trị thực và có sai số nhỏ hơn so với việc dùng thông tin bị trễ.
Mức độ sai lệch của cả hai phương pháp đều phụ thuộc vào số lượng tasks đang chạy trong máy ảo. Hình 5.5b cho thấy khi số lượng tasks càng tăng (đường màu xanh lá cây), sai số do độ trễ của hệ thống cùng với sai số của mạng Bayesian đều tăng lên. Dù vậy, về tổng quan, việc ước lượng cũng có sai số nhưng nhỏ hơn so với trường hợp dùng thông tin bị trễ, và hiệu quả của nó giúp cải thiện sự mất cân bằng của hệ thống do sự thay đổi trạng thái của các tasks trong hệ thống.

5.5.5 Sự mất cân bằng khối lượng công việc giữa các máy trong quá trình hoạt động



Hình 5.6: Mức độ mất cân bằng trong quá trình hoạt động

Như đã được đề cập ở mục 4.1.2, trạng thái các tasks thay đổi trong quá trình

chạy là nguyên nhân dẫn đến sự mất cân bằng tài nguyên sử dụng giữa các máy. Hình 5.6 thể hiện sự thay đổi trọng số mất cân bằng hệ thống (được tính toán trong thuật toán 2 với tài nguyên CPU) theo thời gian của hai thuật toán Resources Balancing và Worstfit. Thuật toán Resources Balancing có sử dụng mạng Bayesian để ước lượng sự thay đổi của tài nguyên trong quá trình chạy để đưa ra quyết định lập lịch có thể thích ứng được với các sự thay đổi đó. Các tasks đến hệ thống và được đưa vào hàng đợi, cách 20 giây lập lịch một lần tại các thời điểm 500, 520, ... Tại các thời điểm lập lịch, ta thấy rằng chỉ số mất cân bằng của thuật toán Worstfit đôi khi tốt hơn so với thuật toán Resources Balancing, nhưng về tổng quan thì Resources Balancing tốt hơn. Bên cạnh đó, độ biến động của đường biểu thị cho Worstfit cũng cao hơn so với Resources Balancing. Điều này đúng với tính chất của thuật toán, do Worstfit chỉ tập trung tối ưu tại thời điểm lập lịch nên chỉ số tại thời điểm đó sẽ tốt hơn, nhưng trong quá trình chạy các tasks thay đổi trạng thái khiến cho sự cân bằng bị phá vỡ. Điều tương tự cũng xảy ra với Resources Balancing, nhưng hiệu quả của việc ước lượng tài nguyên với cân bằng giữa long-running tasks và batch-job tasks đã làm giảm độ biến động của chỉ số mất cân bằng, nên trong quá trình chạy trọng số của thuật toán này nhỏ hơn Worstfit.

6

Kết luận và định hướng nghiên cứu

6.1 Kết quả đạt được

Dựa trên các phân tích thực nghiệm, đồ án này đạt được một số kết quả sau:

- Tăng số lượng tasks được hoàn thành trên đơn vị thời gian so với các thuật toán được sử dụng phổ biến ở các hệ thống máy chủ là Worstfit và FCFS.
- Giảm 50% thời gian thực thi trung bình so với FCFS.
- Cách dùng mạng Bayesian để cải thiện sai số do độ trễ trong quá trình lập lịch đưa ra thông số ước lượng sát với thực tế hơn so với việc dùng thông tin bị trễ.
- Thuật toán đề xuất Resources Balancing giúp cân bằng tài nguyên sử dụng giữa các máy trong hệ thống cả trong quá trình chạy, vượt trội hơn so với việc chỉ cân bằng tại thời điểm lập lịch.

Với kết quả như trên, đồ án đã hoàn thành 3 mục tiêu đặt ra:

1. Đưa ra các hạn chế của các thuật toán lập lịch như Worstfit trong môi trường Cloud
2. Nghiên cứu và đề xuất mô hình lập lịch cải tiến
3. Cài đặt, đánh giá, so sánh với các thuật toán khác

Việc sử dụng các mô hình ước lượng để tiếp cận các thành phần thay đổi liên tục trong hệ thống là một phương pháp khả thi, có thể mở rộng và cải tiến cho nhiều trường hợp phức tạp hơn.

6.2 Định hướng nghiên cứu

Với hiệu quả của phương pháp ước lượng các thành phần lập lịch, có ba hướng phát triển được đưa ra như sau:

1. Trong thực tế, các tasks có thể có quan hệ cha con. Các quan hệ này được biểu diễn dưới dạng cây, kèm theo các rằng buộc giữa chúng. Ta có thể xây dựng đồ thị Bayesian dạng cây tương tự để có thể ước lượng chính xác hơn.

2. Trong thời gian hoạt động, các tasks được gửi đến hệ thống theo dạng streaming. Để thích ứng với sự thay đổi của dữ liệu, mô hình Bayesian cần được học liên tục theo luồng dữ liệu chạy trong hệ thống. Việc cân bằng giữa tri thức học được từ quá khứ với tri thức học được hiện tại sẽ giúp mô hình đưa ra dự đoán chính xác hơn.

3. Ngoài việc tối ưu về thời gian và độ cân bằng sử dụng tài nguyên giữa các máy ảo, ta còn có thể tối ưu năng lượng sử dụng, đưa thêm các ràng buộc về máy ảo, các máy ảo có thể chuyển trạng thái từ bật sang tắt để mở rộng hoặc thu hẹp hệ thống. Trong trường hợp này, nếu có thể ước lượng được tài nguyên cần thiết thì có thể đưa ra số lượng máy tính cần hoạt động nhỏ nhất để tối ưu năng lượng tiêu thụ.

Tài liệu tham khảo

[1] Gurudatt Kulkarni et al. "GRID COMPUTING OVERVIEW". In: May 2013.

[2] Hossein Bidgoli. "Successful Introduction of Cloud Computing into your Organization: A Six-Step Conceptual Model". In: *Journal of International Technology and Information Management* 20.2 (2011).

[3] Kris Bubendorfer and Peter Komisarczuk. "A Position Paper: Towards an Utility Computing and Communications Infrastructure." In: Jan. 2005, pp. 47–53.

[4] Aaqib Rashid and Amit Chaturvedi. "Virtualization and its Role in Cloud Computing Environment". In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* Vol.-7 (Apr. 2019), pp. 1131–1136. DOI: 10.26438/ijcse/v7i4.11311136.

[5] Fernando Pires Barbosa and Andrea Schwertner Charão. "mpact of pay-as-you-go Cloud Platforms on Software Pricing and Development: A Review and Case Study". In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* (2012), pp. 404–417. DOI: 10.1007/978-3-642-31128-4_30.

[6] Mahendra Bhatu Gawali. "Task scheduling and resource allocation in cloud computing using a heuristic approach". In: *Journal of Cloud Computing* 7 (Feb. 2018). DOI: 10.1186/s13677-018-0105-8.

[7] Muhammad Shafie Abd Latiff Syed Hamid Hussain Madni. "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment". In: *PLoS ONE* 12 (Mar. 2017). DOI: 10.6084/m9.figshare.4877438.

[8] Hamid Madni et al. "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment". In: *PLoS ONE* 12 (May 2017), e0176321. DOI: 10.1371/journal.pone.0176321.

[9] Dabiah Alboaneen, Hugo Tianfield, and Yan Zhang. "Glowworm Swarm Optimisation Based Task Scheduling for Cloud Computing". In: Mar. 2017. DOI: 10.1145/3018896.3036395.

[10] Teena Mathew, K. Chandra Sekaran, and John Jose. "Study and analysis of various task scheduling algorithms in the cloud computing environment". In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2014, pp. 658–664. DOI: 10.1109/ICACCI.2014.6968517.

[11] John Wilkes. *More Google cluster data*. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Mountain View, CA, USA, Nov. 2011.

[12] Abhishek Verma et al. "Large-scale cluster management at Google with Borg". In: *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France, 2015.

[13] *CloudSim*. http://www.cloudbus.org/cloudsim/.

[14] Konstantinos (Kostas) Christodoulopoulos et al. "A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks". In: *Computer Communications* 32 (May 2009), pp. 1172–1184. DOI: 10.1016/j.comcom.2009.03.004.

[15] M. Paul, D. Samanta, and G. Sanyal. "Dynamic job Scheduling in Cloud Computing based on horizontal load balancing". In: 2011.

[16] Hesam Izakian, Ajith Abraham, and Vaclav Snasel. "Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments". In: Apr. 2009, pp. 8–12. DOI: 10.1109/CSO.2009.487.

[17] Soheil Anousha and Mahmood Ahmadi. "An Improved Min-Min Task Scheduling Algorithm in Grid Computing". In: May 2013, pp. 103–113. ISBN: 978-3-642-38026-6. DOI: 10.1007/978-3-642-38027-3_11.

[18] Yingchi Mao, Xi Chen, and Xiaofang Li. "Max–Min Task Scheduling Algorithm for Load Balance in Cloud Computing". In: vol. 255. Jan. 2014, pp. 457–465. ISBN: 978-81-322-1758-9. DOI: 10.1007/978-81-322-1759-6_53.

[19] Damanbeer Kaur and Tejinder Sharma. "Scheduling Algorithms in Cloud Computing". In: *International Journal of Computer Applications* 178 (May 2019), pp. 16–21. DOI: 10.5120/ijca2019918801.

[20] Muthucumaru Maheswaran et al. "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems". In: Feb. 1999, pp. 30–44. ISBN: 0-7695-0107-9. DOI: 10.1109/HCW.1999.765094.

[21] M. Mitzenmacher. "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (2001), pp. 1094–1104. DOI: 10.1109/71.963420.

[22] Kay Ousterhout et al. “Sparrow: Distributed, Low Latency Scheduling”. In: New York, NY, USA: Association for Computing Machinery, 2013. ISBN: 9781450323888. DOI: 10.1145/2517349.2522716. URL: https://doi.org/10.1145/2517349.2522716.

[23] Gahyun Park. “A Generalization of Multiple Choice Balls-into-Bins: Tight Bounds”. In: *CoRR* abs/1201.3310 (2012). URL: http://arxiv.org/abs/1201.3310.

[24] Muhammad Tirmazi et al. “Borg: the Next Generation”. In: *EuroSys'20*. Heraklion, Crete, 2020.

[25] Y. Amir et al. “An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster”. In: *IEEE Trans. Parallel Distributed Syst.* 11 (2000), pp. 760–768.

[26] Pascale Minet et al. “Analyzing Traces from a Google Data Center”. In: *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*. 2018, pp. 1167–1172. DOI: 10.1109/IWCMC.2018.8450304.

[27] PeiYun Zhang and MengChu Zhou. “Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy”. In: *IEEE Transactions on Automation Science and Engineering* 15.2 (2018), pp. 772–783. DOI: 10.1109/TASE.2017.2693688.

[28] Fan Zhang et al. “Evolutionary Scheduling of Dynamic Multitasking Workloads for Big-Data Analytics in Elastic Cloud”. In: *IEEE Transactions on Emerging Topics in Computing* 2.3 (2014), pp. 338–351. DOI: 10.1109/TETC.2014.2348196.

[29] Luis Enrique Sucar. *Probabilistic Graphical Models Principles and Applications*. Springer, 2015.

[30] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN: 0262013193.

[31] Amen Ajroud et al. “Loopy Belief Propagation in Bayesian Networks : origin and possibilistic perspectives”. In: *CoRR* abs/1206.0976 (2012). arXiv: 1206.0976. URL: http://arxiv.org/abs/1206.0976.

[32] Yunni Xia et al. “Stochastic Modeling and Quality Evaluation of Infrastructure-as-a-Service Clouds”. In: *IEEE Transactions on Automation Science and Engineering* 12.1 (2015), pp. 162–170. DOI: 10.1109/TASE.2013.2276477.

[33] Ji Wang et al. “FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds”. In: *IEEE Transactions on Computers* 64 (Mar. 2015). DOI: 10.1109/TC.2014.2366751.