HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN XUÂN PHI

NÂNG CAO HIỆU NĂNG CÂN BẰNG TẢI TRÊN ĐIỆN TOÁN ĐÁM MÂY

LUẬN ÁN TIẾN SỸ KỸ THUẬT

HÀ NỘI - 2022

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN XUÂN PHI

NÂNG CAO HIỆU NĂNG CÂN BẰNG TẢI TRÊN ĐIỆN TOÁN ĐÁM MÂY

Chuyên ngành: Hệ thống thông tin

Mã số: 9.48.01.04

LUẬN ÁN TIẾN SỸ KỸ THUẬT

NGƯỜI HƯỚNG DẪN KHOA HỌC:

PGS.TS. TRẦN CÔNG HÙNG

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các kết quả nghiên cứu được viết chung với các tác giả khác đều được sự đồng ý của họ trước khi đưa vào luận án. Các kết quả nêu trong luận án là trung thực và chưa từng được công bố trong các công trình nào khác.

Tác giả

Nguyễn Xuân Phi

LỜI CẢM ƠN

Để hoàn thành được luận án này, tác giả xin được bày tỏ lòng biết ơn sâu sắc đến **PGS.TS. Trần Công Hùng** đã tận tình hướng dẫn, trang bị các phương pháp nghiên cứu, kiến thức khoa học để tôi hoàn thành các nội dung của Luận án.

Hoàn thành Luận án tiến sĩ là một thử thách lớn, đòi hỏi sự kiên trì và tập trung cao độ. Những kết quả đạt được không chỉ là nỗ lực cá nhân, mà còn có sự hỗ trợ và giúp đỡ của nhà trường, bộ môn. Tác giả xin trân trọng cảm ơn Ban Giám đốc Học viện Công nghệ Bưu chính Viễn thông, Hội đồng Khoa học và Đào tạo, Hội đồng Tiến sĩ, Khoa Sau Đại học của Học viện và các Thầy, Cô giáo ngoài Học viện đã góp ý những ý kiến qúy báu và tạo điều kiện thuận lợi giúp tác giả hoàn thiện nội dung nghiên cứu của mình.

Cuối cùng, tôi xin chân thành cảm ơn gia đình, bạn bè và đồng nghiệp đã luôn bên cạnh động viên, khích lệ tinh thần cho tôi hoàn thành mục tiêu nghiên cứu của mình.

Hà Nội, tháng 05 năm 2022

MỤC LỤC

LỜI CAM ĐOAN ..……..……..……..……i
LỜI CÁM ƠN……...…….…...……..……ii
MỤC LỤC……...……...……..……iii
DANH MỤC CHỮ VIẾT TẮT……...……....v
DANH MỤC CÁC KÝ HIỆU……...……....vii
DANH MỤC CÁC BẢNG……...…….…...viii
DANH MỤC CÁC HÌNH……...…….…....ix
MỞ ĐẦU……..……..…...……..……1
1. Lý do chọn đề tài……...…….…...…1
2. Mục tiêu của luận án……...…….…...3
3. Phạm vi, đối tượng và phương pháp nghiên cứu……3
4. Các đóng góp của luận án……...……3
5. Bố cục luận án……...…….…...……4
CHƯƠNG 1. TỔNG QUAN VỀ CÂN BẰNG TẢI TRÊN ĐIỆN TOÁN ĐÁM MÂY……..…….…...…....6
1.1. Cân bằng tải trên điện toán đám mây……...……6
1.1.1. Giới thiệu chung về điện toán đám mây.…….…...……6
1.1.2. Cân bằng tải và hiệu năng cân bằng tải trên điện toán đám mây…….10
1.1.3. Sự cần thiết của cân bằng tải trên điện toán đám mây…….……11
1.1.4. Ảo hóa và quản lý máy ảo trên đám mây……....13
1.1.5. Quản lý và phân bổ tài nguyên trên đám mây……..15
1.2. Bài toán cân bằng tải……...…….…...…17
1.2.1. Phát biểu bài toán và mô hình nghiên cứu……...17
1.2.2. Các yếu tố ảnh hưởng đến cân bằng tải……...….19
1.2.3. Phân loại các thuật toán cân bằng tải……...……23
1.2.4. Đo lường cân bằng tải……...……...26
1.3. Các hướng giải quyết bài toán cân bằng tải……...…….27
1.3.1. Phương pháp xấp xỉ……...……...27
1.3.2. Chiến lược lập lịch phân bổ tài nguyên……...….31
1.3.3. Phương pháp cải tiến các tham số……...……33
1.4. Các vấn đề mà luận án cần giải quyết……...……..38
1.5. Kết luận Chương 1……...…….…...…...39

CHƯƠNG 2. PHÁT TRIỂN MỘT SỐ THUẬT TOÁN CÂN BẰNG TẢI NHẰM CẢI THIỆN THỜI GIAN ĐÁP ỨNG TRÊN ĐIỆN TOÁN ĐÁM MÂY……41
2.1. Đặt vấn đề……41
2.2. Thuật toán LBAIRT……42
2.2.1. Cơ sở lý thuyết……42
2.2.2. Đề xuất thuật toán……45
2.2.3. Kết quả mô phỏng……51
2.3. Thuật toán RRTA……56
2.3.1. Đề xuất thuật toán……56
2.3.2. Thực nghiệm mô phỏng……60
2.4. Kết luận Chương 2……68
CHƯƠNG 3. PHÁT TRIỂN MỘT SỐ THUẬT TOÁN CÂN BẰNG TẢI NHẰM CẢI THIỆN THỜI GIAN XỬ LÝ TRÊN ĐIỆN TOÁN ĐÁM MÂY……70
3.1. Đặt vấn đề……70
3.2. Thuật toán TMA……71
3.2.1. Đề xuất thuật toán……71
3.2.2. Kết quả mô phỏng……73
3.2.3. Đánh giá……81
3.3. Thuật toán MMSIA……81
3.3.1. Giới thiệu thuật toán Max – Min……82
3.3.2. Đề xuất thuật toán MMSIA……84
3.3.3. Kết quả mô phỏng……87
3.4. Kết luận Chương 3……94
KẾT LUẬN……95
I. Những kết quả chính của luận án……95
II. Hướng phát triển của luận án……97
DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CÔNG BỐ……98
TÀI LIỆU THAM KHẢO……100
PHỤ LỤC……109

DANH MỤC CÁC CHỮ VIẾT TẮT

API	Application Programming Interface	Giao diện lập trình ứng dụng
ARIMA	Autoregressive Integrated Moving Average	Thuật toán dự báo
AWS	Amazon Web Services	Dịch vụ đám mây của Amazon
DAIRS	Dynamic and Integrated Resource Scheduling Algorithm	Tên một thuật toán cân bằng tải
DLBS	Dynamic Load Balancing Strategy	Tên một thuật toán cân bằng tải
ELBAA	Enhanced Load Balancing Approach to Avoid Deadlocks in Cloud	Tên một thuật toán cân bằng tải
HDCS	Heterogeneous Distributed Computing System	Tên một thuật toán cân bằng tải
IaaS	Infrastructure as a Service	Dịch vụ cơ sở hạ tầng đám mây
JIQ	Join Idle Queue	Tên một thuật toán cân bằng tải
LBAIRT	Load Balancing Algorithm to Improve Response time on Cloud Computing	Tên thuật toán cân bằng tải đề xuất
LBIMM	Load Balance Improved Min-Min	Tên một thuật toán cân bằng tải
LBVS	Load Balancing Virtual Server	Tên một thuật toán cân bằng tải
MMSIA	Improved Max-Min Scheduling Algorithm for Load Balancing on Cloud Computing	Tên thuật toán cân bằng tải đề xuất

MIPS	Million Instructions Per Second	Số chỉ thị mỗi giây, tốc độ xử lý của máy tính
RRTA	Reduce Response Time Algorithm	Tên thuật toán cân bằng tải đề xuất
SaaS	Software as a Service	Dịch vụ phần mềm đám mây
SHC	Stochastic Hill Climbing	Tên một thuật toán cân bằng tải
SIQ	Stochastic Idle Queue	Tên một thuật toán cân bằng tải
TMA	Throttled Modified Algorithm	Tên thuật toán cân bằng tải đề xuất
VM	Virtual Machine	Máy ảo

DANH MỤC CÁC KÍ HIỆU

$(K_i^{r(t)} + K_i^{d(t)})\mu_i$	Biểu diễn tỉ lệ dịch vụ của cụm C_i
$\xi_i^{(t)}$	Biểu diễn tỉ lệ dịch vụ của cụm C_i
μ_i	Tỉ lệ dịch vụ một cá thể VM của cụm C_i.
$\omega_i^{(t)}.\ \lambda^{(t)}$	Đại điện cho tỉ lệ đến trung bình theo phân phối Possion
$\frac{1}{m}\sum_j t_j$	Ngưỡng dưới của giá trị makespan tối ưu T*
$\frac{1}{m}\sum_k t_k$	Giới hạn dưới của giá trị tối ưu
J	Tập hợp các công việc
m	Số lượng máy chủ
M	Tập các máy chủ
$Max_i\ T_i$	Tải lớn nhất trên máy chủ thứ i
M_i	Máy chủ thứ i
n	Số lượng các công việc
PT	Tổng thời gian xử lý của tất cả máy ảo VMs
PT_i	Thời gian xử lý của máy ảo VM_i
T	Tổng thời gian hoàn thành tất cả các công việc
T*	Thời gian tối ưu hoàn thành tất cả các công việc
T_i	Tổng thời gian hoàn thành tất cả các công việc trên máy thứ i
t_j	Thời gian xử lý của từng công việc

DANH MỤC CÁC BẢNG

Bảng 1.1 Khảo sát các tham số ảnh hưởng đến phương pháp cân bằng tải ……..…21
Bảng 1.2 Ưu điểm và nhược điểm của một số phương pháp cân bằng tải ……24
Bảng 2.1 Thuật toán Throttled ……43
Bảng 2.2 Ưu điểm và nhược điểm của thuật toán Throttled ……45
Bảng 2.3 Giá trị các tham số trong thiết lập đám mây ……51
Bảng 2.4 Cấu hình các VM……52
Bảng 2.5 Thiết lập tham số các Cloudlet ……..53
Bảng 2.6 So sánh kết quả mô phỏng giữa hai thuật toán Throttled và LBAIRT …...56
Bảng 2.7 Thông số cấu hình Datacenter ……61
Bảng 2.8 Cấu hình VM……62
Bảng 2.9 Thông số cấu hình của Request ……62
Bảng 2.10 Kết quả thực nghiệm mô phỏng với 3 VM ……63
Bảng 2.11 Kết quả thực nghiệm mô phỏng với 4 VM ……64
Bảng 2.12 Kết quả thực nghiệm mô phỏng với 5 VM ……65
Bảng 3.1 Thông số cấu hình User base ……74
Bảng 3.2. Số lượng yêu cầu được phân phối tới từng máy ảo (VM)……75
Bảng 3.3 Kết quả mô phỏng trường hợp 20 VM ……76
Bảng 3.4 Số lượng yêu cầu được phân phối tới từng VM ……77
Bảng 3.5 Kết quả mô phỏng trường hợp 2……80
Bảng 3.6 Thông số Datacenter ……88
Bảng 3.7 Cấu hình của VMs ……88
Bảng 3.8 Thông số các Request ……89
Bảng 3.9 Bảng kết quả thời gian xử lý lần 1……90
Bảng 3.10 Bảng kết quả thời gian xử lý lần 2……91
Bảng 3.11 Bảng kết quả thời gian xử lý lần 3……92
Bảng 3.12 Bảng kết quả thời gian xử lý lần 4……92

DANH MỤC CÁC HÌNH

Hình 1.1 Mô hình điện toán đám mây……..……..……7
Hình 1.2 Các dịch vụ điện toán đám mây…….…..…….……9
Hình 1.3 Mức độ cung cấp dịch vụ của các mô hình điện toán đám mây……..…… 10
Hình 1.4 Mô hình phân lớp kiến trúc ảo hóa 03 tầng ……..…….....…….…..14
Hình 1.5 Phân loại các tài nguyên trên điện toán đám mây.…….…..16
Hình 1.6 Mô hình nghiên cứu cân bằng tải trên điện toán đám mây …….19
Hình 1.7 Kết quả chạy thuật toán Greedy-Balance ……...….28
Hình 1.8 Mô hình lập lịch khối lượng công việc cho đám mây……...…..35
Hình 2.1 Sơ đồ nguyên lý của thuật toán Throttled…….…...44
Hình 2.2 Mô hình IaaS điện toán đám mây thông qua thành phần DatacenterBrocker…..46
Hình 2.3 Lưu đồ thuật toán đề xuất LBAIRT……..50
Hình 2.4 Thực nghiệm mô phỏng so sánh thời gian đáp ứng của Throttled và LBAIRT khi thay đổi số lượng cloudlet……..…..55
Hình 2.5 Mô hình thuật toán RRTA…….……58
Hình 2.6 So sánh thời gian đáp ứng dự báo của 3 máy ảo và ngưỡng……....64
Hình 2.7 So sánh thời gian đáp ứng dự báo của 4 VM và ngưỡng……..…65
Hình 2.8 So sánh thời gian đáp ứng dự báo của 5 VM và ngưỡng……...…..66
Hình 2.9. So sánh thời gian đáp ứng dự báo trong các trường hợp 3 máy áo, 4 máy ảo và 5 máy ảo……..……67
Hình 3.1 Sơ đồ hoạt động của thuật toán TMA…….…….73
Hình 3.2 Kết quả mô phỏng trường hợp 20 VM…….…77
Hình 3.3 Kết quả mô phỏng trường hợp 50 VM…….…….81
Hình 3.4 Nguyên lý thuật toán lập lịch Max-Min ……...…82
Hình 3.5 Sơ đồ thuật toán lập lịch Max-Min ...……..…….83
Hình 3.6 Sơ đồ nguyên lý thuật toán MMSIA …….……...…..85
Hình 3.7. Sơ đồ thuật toán MMSIA…….86
Hình 3.8 Biểu đồ so sánh thời gian xử lý lần 1……...……90
Hình 3.9 Biểu đồ so sánh thời gian xử lý lần 2……...……91
Hình 3.10 Biểu đồ so sánh thời gian xử lý lần 3……..……..……92
Hình 3.11 Biểu đồ so sánh thời gian xử lý lần 4…….……..……..93

MỞ ĐẦU

1. Lý do chọn đề tài

Theo các thống kê[1], lượng dữ liệu truyền trên hệ thống mạng toàn cầu nếu lưu trữ trên đĩa DVD thì số lượng đĩa này xếp hàng sẽ có chiều dài bằng 2 lần quãng đường từ trái đất tới mặt trăng, dự báo lượng dữ liệu này sẽ tăng thêm 44 lần vào năm 2020. Theo dự báo của ViettelIDC[2] vào năm 2021 thì điện toán đám mây đóng vai trò quan trọng trong việc đối phó với cuộc khủng hoảng COVID-19. Khi các doanh nghiệp ở khắp mọi nơi cố gắng duy trì hoạt động bằng cách để nhân viên làm việc từ xa, tất cả các nhà cung cấp đám mây về cơ bản đã tăng doanh thu và tiếp tục tăng trưởng với tốc độ chóng mặt. Còn theo Forrester[3], thì năm 2021: Điện toán đám mây đẩy nhanh quá trình chuyển đổi của doanh nghiệp sau đại dịch COVID-19, sự dịch chuyển lên đám mây là bắt buộc, và là giải pháp hữu hiệu giúp doanh nghiệp có thể tồn tại và phát triển; Forrester dự đoán thị trường cơ sở hạ tầng đám mây công cộng toàn cầu sẽ tăng trưởng 35% lên 120 tỷ USD vào năm 2021. Còn theo tờ Thời báo ngân hàng[4] thì cho rằng, thị trường điện toán đám mây đang bùng nổ sau khi doanh nghiệp Việt Nam tham gia vào cuộc cánh mạng 4.0 và Chính phủ ban hành "Chương trình chuyển đổi số quốc gia đến năm 2025, định hướng đến năm 2030", bài báo này cho rằng, theo thống kê của Garner Inc, chi tiêu của người sử dụng cuối đối với dịch vụ điện toán đám mây công cộng trên toàn thế giới năm 2020 là 257,5 tỷ USD. Con số này vào năm 2022 dự báo tăng khoảng 40% đạt 362,2 tỷ USD. Với Việt Nam, theo thông tin từ Bộ Thông tin và Truyền thông, thị trường điện toán đám mây Việt Nam năm 2020 đạt khoảng 3.200 tỷ đồng (tương đương 133 triệu USD), dự báo đến năm 2025, sẽ đạt 500 triệu USD. Một cuộc khảo sát gần đây của Viện Giá trị doanh nghiệp IBM (IBV) cho thấy 56% doanh nghiệp Việt Nam đã và đang sử dụng nền tảng quản lý đám mây.

Với sự bùng nổ đó, đã đặt ra những thách thức rất lớn cho người dùng trên toàn thế giới về các vấn đề: xử lý dữ liệu, an toàn dữ liệu và đặc biệt vấn đề cân bằng tải truy cập. Điện toán đám mây được xem như một trong những nền tảng tốt nhất giúp lưu trữ dữ liệu và cung cấp dịch vụ tính toán với chi phí tối thiểu và truy cập mọi lúc

[1] *http://cloudvps.vn/dien-toan-dam-may-thach-thuc-va-co-hoi/*
[2] *https://viettelidc.com.vn/tin-tuc/du-doan-xu-huong-dien-toan-dam-may-nam-2021*
[3] *https://vticloud.io/forrester-du-doan-nam-2021-dien-toan-dam-may-day-nhanh-qua-trinh-chuyen-doi-cua-doanh-nghiep-sau-dai-dich-covid19/*
[4] *https://thoibaonganhang.vn/thi-truong-dien-toan-dam-may-dang-bung-no-111922.html*

mọi nơi từ Internet. Điện toán đám mây ra đời cho phép các ứng dụng bớt lệ thuộc vào mạng hạ tầng, tiết kiệm cho người dùng khi không quá đầu tư vào hệ thống phần cứng. Điện toán đám mây đang trở thành đích đến của nhiều doanh nghiệp lớn, đặc biệt có liên quan tới mảng trung tâm dữ liệu.

Điện toán đám mây đã trở thành một trong những nền tảng được lựa chọn sử dụng nhiều nhất trong các giải pháp công nghệ hiện đại, bởi vì nó có tính linh động, cung cấp dịch vụ theo nhu cầu sử dụng, và khả năng mở rộng hay thu hẹp một cách dễ dàng. Chính vì vậy, nó luôn đòi hỏi những bước nghiên cứu mới, mang tính tiên phong trong việc nâng cao hiệu quả hoạt động và phải sử dụng tối ưu nhất nguồn tài nguyên nhưng vẫn đảm bảo việc cung ứng dịch vụ là tốt nhất và thời gian nhanh nhất. Trong việc nâng cao hiệu quả hoạt động của điện toán đám mây, cân bằng tải đóng vai trò rất quan trọng và mấu chốt, đây là một cơ chế chủ yếu để giúp cho các giao dịch trên đám mây diễn ra tốt nhất và an toàn nhất. Với sự phát triển nhanh chóng về quy mô cũng như số lượng của các ứng dụng chạy trên nền tảng đám mây thì cân bằng tải phải luôn luôn được cải tiến và nâng cấp cho phù hợp với lượng và chất của sự phát triển đó. Vì thế mà cân bằng tải là một thách thức lớn, luôn được sự quan tâm của các nhà khoa học, nghiên cứu nhằm đáp ứng ngày một tốt hơn cho môi trường điện toán đám mây.

Hiện nay với sự phát triển bùng nổ của Internet thì việc trao đổi dữ liệu, phân bổ tài nguyên máy tính của một tổ chức, doanh nghiệp là vấn đề đặt ra nhiều thách thức. Có nhiều công trình nghiên cứu để nâng cao khả năng cân bằng tải trong môi trường điện toán đám mây [3], [11], [21], [28], [44], [72], [93], [104], [109]. Trong tài liệu [93] tác giả dùng phương pháp cân bằng động với khả năng giám sát tập trung, sử dụng ý tưởng dù môi trường tính toán là phân tán nhưng trạng thái của từng yêu cầu (request) phải được quản lý tập trung nhằm tăng khả năng lập lịch, kiểm tra tình trạng các bộ xử lý trước khi làm nhiệm vụ điều phối tải, cân bằng tải trên các tuyến đường đi. Và vấn đề này thu hút được rất nhiều sự quan tâm của các nhà khoa học và cũng đã đạt được nhiều thành tựu ở các công trình [5], [18], [23], [31], [34], [35],

[42], [83], [84], [108]. Vì vậy, việc nghiên cứu cải tiến hiệu năng cân bằng tải cho điện toán đám mây là một cách tiếp cận của đề tài.

Xuất phát từ những lý do trên, việc nghiên cứu các giải pháp mới nâng cao cân bằng tải là vấn đề cấp thiết. Đề tài ***"Nâng cao hiệu năng cân bằng tải trên điện toán đám mây***" được thực hiện trong khuôn khổ luận án tiến sĩ chuyên ngành Hệ thống thông tin. Kết quả nghiên cứu đóng góp vào việc cải tiến các thuật toán cân bằng tải trên điện toán đám mây. Thách thức đặt ra với đề tài là rất lớn vì với lưu lượng dữ liệu truyền đi trên Internet ngày càng bùng nổ, mỗi loại dữ liệu có đặc thù riêng và có phương pháp xử lý riêng. Việc lựa chọn kỹ thuật để cải tiến cân bằng tải là rất quan trọng, với ý nghĩa to lớn là nâng cao chất lượng dịch vụ đám mây cho người dùng.

2. Mục tiêu của luận án

- Mục tiêu thứ nhất: Nghiên cứu, phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian đáp ứng trên điện toán đám mây.
- Mục tiêu thứ hai: Nghiên cứu, phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian xử lý trên điện toán đám mây.

3. Phạm vi, đối tượng và phương pháp nghiên cứu

- Phạm vi: Các thuật toán cân bằng tải nhằm cải thiện các tham số thời gian đáp ứng, thời gian xử lý trên môi trường điện toán đám.
- Đối tượng nghiên cứu: Các thuật toán cân bằng tải, các tham số ảnh hưởng đến cân bằng tải, các phương pháp đánh giá thuật toán cân bằng tải trên điện toán đám mây.
- Phương pháp nghiên cứu: Xây dựng mô hình; Cài đặt/thử nghiệm mô hình trên các phần mềm mô phỏng; So sánh, đánh giá kết quả so với các thuật toán khác.

4. Các đóng góp của luận án

- Đóng góp thứ nhất: Đề xuất 02 thuật toán cân bằng tải nhằm giảm thời gian đáp ứng trên điện toán đám mây. Các kết quả nghiên cứu công bố trong các công trình (CT4) và (CT7):

- Thuật toán LBAIRT (CT4): Điểm mới của thuật toán LBAIRT là xét thêm số tham số thời gian hoàn thành các yêu cầu dự kiến của mỗi tài nguyên (VM). Kết quả mô phỏng cho thấy LBAIRT có tỉ lệ chấp nhận yêu cầu đầu vào cao hơn, trong khi thời gian tính toán trung bình thấp hơn các công trình liên quan.
- Thuật toán RRTA (CT7): Điểm mới của công trình là sử dụng thuật toán dự báo ARIMA để dự báo thời gian đáp ứng, từ đó đưa ra cách giải quyết phân phối tài nguyên hợp lý.

- Đóng góp thứ hai: Đề xuất 02 thuật toán cân bằng tải với mục đích cải thiện thời gian xử lý trên điện toán đám mây. Các kết quả nghiên cứu công bố trong các công trình (CT5) và (CT6):
  - Thuật toán TMA (CT5): Điểm cải tiến của thuật toán là sử dụng 2 bảng chỉ mục trạng thái của các VM. Các kết quả thu được từ thuật toán đề xuất đã đạt được một số kết quả tốt: giới hạn số lượng yêu cầu được xếp hàng để phân phối, cải thiện thời gian xử lý và thời gian phản hồi của các trung tâm đám mây so với hai thuật toán Round Robin và Throttled.
  - Thuật toán MMSIA (CT6): Điểm mới của thuật toán là nhóm các yêu cầu và các máy ảo theo thời gian hoàn thành dự kiến và thời gian thực hiện hoàn thành tổng thể. Thực nghiệm cho thấy thuật toán đã cải thiện thời gian xử lý các yêu cầu đầu vào.

Các kết quả nghiên cứu của luận án là những đóng góp mới cho lĩnh vực cân bằng tải, đồng thời có thể ứng dụng để giải quyết bài toán nâng cao khả năng cân bằng tải trên điện toán đám mây. Các thuật toán này cũng là tiền đề để tìm ra phương pháp tối ưu giúp cân bằng tải hiệu quả cho các dịch vụ trên đám mây, thông qua đó tăng mức độ hài lòng cho người sử dụng.

5. Bố cục luận án

Nội dung của luận án chia thành 03 chương, cụ thể như sau:

Chương 1 - Tổng quan về cân bằng tải trên điện toán đám mây: Giới thiệu điện toán đám mây, bài toán cân bằng tải trên điện toán đám mây; phân tích ảnh hưởng của các tham số đến cân bằng tải. Chương 1 cũng trình bày tình hình nghiên cứu trên thế giới với các hướng tiếp cận giải quyết bài toán cân bằng tải. Trên cơ sở phân tích, đánh giá về ưu, nhược điểm các công trình đã công bố, xác định rõ hướng nghiên cứu của đề tài là lựa chọn cải tiến các tham số nhằm nâng cao khả năng cân bằng tải trên điện toán đám mây. Các kết quả nghiên cứu được công bố ở các công trình (CT1), (CT2) và (CT3).

Chương 2 – Phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian đáp ứng trên điện toán đám mây: Nội dung chương đề xuất 02 phương pháp nâng cao cân bằng tải dựa trên việc cải thiện tham số thời gian đáp ứng các yêu cầu đầu vào trên điện toán đám mây. Phương pháp thứ nhất là thuật toán LBAIRT (CT4), xét thêm số tham số thời gian hoàn thành các yêu cầu dự kiến của mỗi tài nguyên (VM), việc phân phối tải kết hợp với thời gian hoàn thành dự kiến của các máy ảo cho thấy tỉ lệ chấp nhận các yêu cầu đầu vào cao hơn các công trình đưa ra so sánh. Phương pháp thứ hai là thuật toán RRTA (CT7), dùng kỹ thuật dự báo ARIMA để dự báo thời gian đáp ứng của máy ảo tiếp theo. Thực nghiệm tiến hành trên dữ liệu mô phỏng cho thấy, các thuật toán đề xuất có tỉ lệ chấp nhận yêu cầu cao hơn, cải thiện được thời gian đáp ứng so với các thuật toán Roud Robin, Throttled.

Chương 3 – Phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian xử lý trên điện toán đám mây: Nội dung chương này trình bài 02 cách tiếp cận giải quyết bài toán cân bằng tải dựa trên thời gian xử lý. Thuật toán TMA (CT5) và thuật toán MMSIA (CT6) được đề xuất để cải thiện thời gian xử lý trên điện toán đám mây. Chương 3 cũng trình bày thực nghiệm trên bộ dữ liệu mô phỏng, kết quả thực nghiệm cho thấy các thuật toán đề xuất đã cải thiện được thời gian xử lý so với các thuật toán dùng để so sánh. Thuật toán TMA được so sánh với thuật toán Roud Robin và Throttled. Thuật toán MMSIA được được so sánh với các thuật toán: Max-Min, Min-Min, Roud Robin.

Phần cuối là một số kết luận và hướng phát triển của luận án.

CHƯƠNG 1.
TỔNG QUAN VỀ CÂN BẰNG TẢI TRÊN ĐIỆN TOÁN ĐÁM MÂY

Trong Chương 1, luận án trình bày tổng quan về cân bằng tải trên điện toán đám mây và các tham số ảnh hưởng đến cân bằng tải. Trình bày tình hình nghiên cứu trên thế giới về các thuật toán cân bằng tải trên điện toán đám mây cũng như các hướng tiếp cận giải quyết bài toán cân bằng tải. Phần cuối là đánh giá các ưu, nhược điểm của các công trình liên quan mật thiết đến đề tài từ đó nêu ra định hướng nghiên cứu. Nội dung của Chương 1 được công bố ở các công trình (CT1), (CT2) và (CT3).

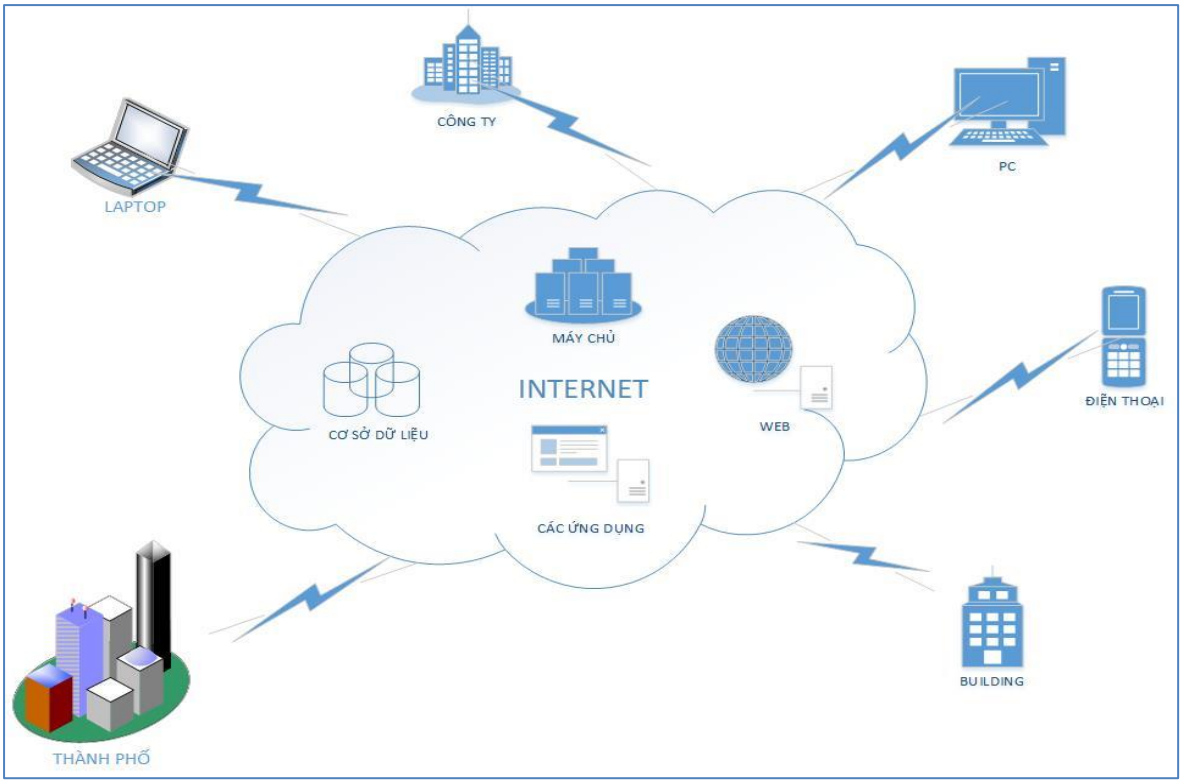
1.1. Cân bằng tải trên điện toán đám mây

1.1.1. Giới thiệu chung về điện toán đám mây

Điện toán đám mây ra đời từ những năm 1950 [66], khi máy chủ tính toán quy mô lớn được triển khai tại một số cơ sở giáo dục và tập đoàn lớn tại Mỹ. Trong những năm 1960 – 1990, xuất hiện luồng ý tưởng coi máy tính hay tài nguyên công nghệ thông tin có thể được tổ chức như hạ tầng dịch vụ công cộng, đây là tiền đề lớn cho sự phát triển sau này của điện toán đám mây. Vào năm 2006, Công ty Amazon cung cấp nền tảng Amazon Web Services (AWS), đánh dấu việc thương mại hóa điện toán đám mây. Từ đầu năm 2008, Eucalyptus được giới thiệu là nền tảng điện toán đám mây mã nguồn mở đầu tiên, tương thích với API của AWS. Tính tới thời điểm hiện tại, có rất nhiều các sản phẩm điện toán đám mây được đưa ra như Google App Engine, Microsoft Azure, Nimbus.

Điện toán đám mây, còn gọi là điện toán máy chủ ảo, là mô hình điện toán sử dụng các công nghệ máy tính và phát triển dựa vào mạng Internet. Thuật ngữ "đám mây" ở đây chỉ độ phức tạp của các cơ sở hạ tầng mạng Internet. Trong mô hình điện toán đám mây, cho phép người sử dụng truy cập các dịch vụ công nghệ từ nhà cung cấp điện toán đám mây mà không cần quan tâm đến hạ tầng mạng. Theo tổ chức IEEE, *"Nó là hình mẫu trong đó thông tin được lưu trữ thường trực tại các máy chủ trên*

Internet và chỉ được lưu trữ tạm thời ở các máy khách, bao gồm máy tính cá nhân, trung tâm giải trí, máy tính trong doanh nghiệp, các phương tiện máy tính cầm tay, ..."



Hình 1.1. Mô hình điện toán đám mây

Theo định nghĩa của Viện Quốc gia Tiêu chuẩn và Công nghệ Mỹ (US NIST) [66]: Điện toán đám mây là mô hình điện toán cho phép truy cập qua mạng để lựa chọn và sử dụng tài nguyên tính toán (ví dụ: mạng, máy chủ, lưu trữ, ứng dụng và dịch vụ) theo nhu cầu một cách thuận tiện và nhanh chóng, đồng thời cho phép kết thúc sử dụng dịch vụ, giải phóng tài nguyên dễ dàng, giảm thiểu các giao tiếp với nhà cung cấp. Những tài nguyên này có thể được cung cấp một cách nhanh chóng hoặc thu hồi với chi phí quản lý tối thiểu hoặc tương tác tối thiểu với nhà cung cấp dịch vụ. Cũng từ đây, NIST đưa ra mô hình điện toán đám mây mang năm đặc điểm cơ bản, ba mô hình dịch vụ và bốn mô hình triển khai cài đặt. Định nghĩa của NIST là một cách nhìn rõ ràng và bao quát về điện toán đám mây. Theo đó, mô hình này cho phép sử dụng dịch vụ theo yêu cầu; cung cấp khả năng truy cập dịch vụ qua mạng rộng rãi từ máy tính để bàn, máy tính xách tay tới thiết bị di động; với tài nguyên tính toán động, phục vụ nhiều

người, năng lực tính toán mềm dẻo, đáp ứng nhanh với nhu cầu thấp tới cao. Mô hình điện toán đám mây cũng đảm bảo việc sử dụng các tài nguyên được "đo" để nâng cấp dịch vụ quản trị và tối ưu được tài nguyên, đồng thời người dùng chỉ phải trả chi phí cho phần tài nguyên đã sử dụng.

Điện toán đám mây đã phát triển nhanh chóng và đạt được nhiều thành tựu to lớn như khả năng mở rộng và tính toán song song trên hệ thống tính toán lưới, từ đó giúp khách hàng chia sẻ tài nguyên triệt để và hữu ích nhất. Đáp ứng được nhu cầu mỗi lúc một lớn của việc lưu trữ dữ liệu và xử lý dữ liệu ngày nay, các trung tâm dữ liệu đã mỗi lúc một khổng lồ hơn và đến hiện nay có thể lên tới hàng triệu máy chủ khắp toàn cầu.

Đặc điểm của điện toán đám mây [66]:

- Khả năng truy cập rộng lớn: Người dùng có thể sử dụng bất kỳ thiết bị nào có kết nối internet như: computer, laptop, thiết bị di động… truy cập vào dịch vụ điện toán đám mây.
- Tự cấp dịch vụ theo nhu cầu: Người dùng được cấp tài nguyên theo nhu cầu một cách tự động, không cần sự can thiệp từ nhà cung cấp dịch vụ.
- Không phụ thuộc vị trí địa lý: Tài nguyên trên dịch vụ điện toán đám mây được điều phối và chia sẻ linh hoạt. Người dùng không biết và không thể điều khiển được vị trí tài nguyên.
- Tính co giãn nhanh: Tài nguyên trên dịch vụ "đám mây" có thể được cấp phát hoặc thu hồi một cách nhanh chóng, linh hoạt và có khả năng thay đổi tài nguyên tăng lên hoặc giảm xuống theo nhu cầu sử dụng.
- Dịch vụ đo lường: Các hệ thống điện toán đám mây có khả năng tự điều khiển, tinh chỉnh và giám sát, đo lường tài nguyên.

Điện toán đám mây có 3 loại mô hình dịch vụ [66]:

SaaS	Nhà cung cấp tạo ra nhiều ứng dụng cơ bản, hoàn chỉnh và triển khai thành dịch vụ để đáp ứng nhu cầu của đại đa số người dùng. Những ứng dụng này được lưu trữ trên máy chủ của nhà cung cấp hoặc máy khách hàng đã có tải ứng dụng xuống thiết bị. Người dùng chỉ việc sử dụng dịch vụ mà không cần quan tâm các vấn đề khác. Ví dụ: Dịch vụ email hay các ứng dụng Zalo, Google Docs, Google Calendar, Google Translate của Google, SmallPDF, OneDrive, Evennote, Facebook, Twitter,…
PaaS	Cung cấp cách thức, tính năng cần thiết cho việc xây dựng ứng dụng trên một nền tảng nào đó. Có 2 dạng hạ tầng được xây dựng phổ biến là hạ tầng trao đổi thông tin ứng dụng (middleware) và nền tảng ứng dụng (application server) với các công cụ và ngôn ngữ lập trình nhất định để xây dựng ứng dụng. Người dùng triển khai ứng dụng mà không cần quan tâm đến chi phí hay thông số phần cứng và phần mềm bên dưới. Ví dụ: Google App Engine, Openshilt, Salesforce, Microsoft Azure.,…
IaaS	Nhà cung cấp triển khai hạ tầng phần cứng (VM, network, vùng lưu trữ) trên các hệ thống phân tán và cung cấp dịch vụ cho người. Người dùng không thể biết thông tin hạ tầng thực tế bên trong "đám mây" nhưng có toàn quyền quản lý và sử dụng tài nguyên được cung cấp, cũng như yêu cầu mở rộng tài nguyên. Ví dụ: Amazon EC2/S3, Elastra (Beta 2.0 2/2009), Nirvanix, AppNexus,…

Hình 1.2. Các dịch vụ điện toán đám mây

	IaaS		PaaS		SaaS	
	Dịch vụ	Người dùng	Dịch vụ	Người dùng	Dịch vụ	Người dùng
Ứng dụng		X		X	X	
Dữ liệu		X		X	X	
Thời gian chạy		X	X		X	
Middleware		X	X		X	
Hệ điều hành	X		X		X	
Ảo hóa	X		X		X	
Máy chủ	X		X		X	
Vùng lưu trữ	X		X		X	
Mạng kết nối	X		X		X	
Ứng dụng	X		X		X	

Hình 1.3. Mức độ cung cấp dịch vụ của các mô hình điện toán đám mây

1.1.2. Cân bằng tải và hiệu năng cân bằng tải trên điện toán đám mây

Theo tài liệu [111] tác giả định nghĩa cân bằng tải là kỹ thuật phân bố lưu lượng mạng qua nhiều máy chủ, đảm bảo rằng không có máy chủ nào bị quá tải. Cân bằng tải làm tăng khả năng đáp ứng của ứng dụng bằng cách phân phối các yêu cầu một cách đồng đều. Bằng cách này, sẽ giúp cho hệ thống của doanh nghiệp giảm thiểu một cách tối đa tình trạng một máy chủ bị quá tải và ngưng hoạt động. Hoặc khi một máy chủ gặp sự cố, chức năng cân bằng tải đám mây sẽ chỉ đạo phân phối công việc của máy chủ đó cho các máy chủ còn lại, cải thiện năng suất hoạt động tổng thể của

hệ thống. Khái niệm cân bằng tải chủ yếu đề cập đến ý tưởng phân phối tải đồng đều trên các tài nguyên công nghệ thông tin có sẵn [112]. Cân bằng tải cũng tập trung vào việc giảm thiểu độ trễ tương ứng với các yêu cầu và do đó cải thiện việc sử dụng tài nguyên, nâng cao hiệu suất hệ thống. Nó cung cấp tính đàn hồi và khả năng thích ứng cho các ứng dụng có kích thước có thể thay đổi theo thời gian và đòi hỏi thêm tài nguyên công nghệ thông tin. Các mục tiêu khác là giảm sử dụng năng lượng và giải phóng các-bon, tránh tắc nghẽn, đáp ứng các yêu cầu về QoS. Theo tài liệu [110] cân bằng tải là một vấn đề phổ biến trong đám mây để duy trì hiệu suất của các ứng dụng thỏa mãn chất lượng dịch vụ (QoS) và tuân theo thỏa thuận mức dịch vụ (SLA) theo yêu cầu từ các nhà cung cấp đám mây cho doanh nghiệp. Các nhà cung cấp đám mây phải vật lộn để phân phối khối lượng công việc đồng đều giữa các máy chủ. Một kỹ thuật cân bằng tải hiệu quả phải tối ưu hóa và đảm bảo sự hài lòng cao của người dùng bằng cách sử dụng tài nguyên của máy ảo một cách hiệu quả.

Các thuật toán cân bằng tải trên điện toán đám mây luôn cố gắng giải quyết một vấn đề cụ thể, bên cạnh đó cũng phải xét đến: bản chất của yêu cầu, quy mô của yêu cầu, sự phụ thuộc của các yêu cầu, khả năng chia nhỏ các yêu cầu, độ phức tạp thuật toán, kiến trúc phần cứng. Việc thiết kế thuật toán cân bằng tải phải xét đến các yêu tố này để nâng cao khả năng cân bằng tải, hay nói cách khác là nâng cao hiệu năng cân bằng tải. Bên cạnh các yếu tố trên, thuật toán cân bằng tải cần cải thiện các tham số ảnh hưởng đến khả năng cân bằng tải, đó là: thời gian đáp ứng, thời gian xử lý, thời gian chờ, độ trễ,…Bài toán cân bằng tải là một bài toán không có lời giải tối ưu, tuy nhiên các thuật toán cân bằng tải khi cải thiện các tham số này thì phải chấp nhận hy sinh tham số khác trong mức độ cho phép. Nâng cao hiệu năng cân bằng tải là nhiệm vụ của các thuật toán cân bằng tải thông qua việc tối ưu các tham số ảnh hưởng đến cân bằng tải, giảm chi phí thời gian phục vụ các truy vấn của người dùng.

1.1.3. Sự cần thiết của cân bằng tải trên điện toán đám mây

Do sự phát triển không ngừng của điện toán đám mây nên vấn đề trao đổi, xử lý, an toàn dữ liệu và đặc biệt vấn đề cân bằng tải truy nhập là những vấn đề cấp thiết

và có ý nghĩa thực tiễn cao, vì vấn đề cơ bản nhất trong truyền thông là những vấn đề liên quan đến dữ liệu. Tuy nhiên, với sự phát triển nhanh chóng về quy mô cũng như số lượng của các ứng dụng thì việc truy nhập này thường bị đe doạ bởi sự quá tải. Bởi vì, các ngành công nghiệp, dịch vụ hiện nay đang gia tăng sử dụng các dịch vụ này để làm giảm cơ sở hạ tầng và chi phí bảo dưỡng, do đó tải trên điện toán đám mây đang tăng lên từng ngày. Có thể hiểu một cách đơn giản là cần có một hoặc nhiều phương pháp để không có một nút bất kỳ nào quá tải và tải được phân phối cân bằng giữa các các nút. Một thuật toán cân bằng tải lý tưởng giúp sử dụng các tài nguyên có sẵn một cách hiệu quả nhất, bằng cách đảm bảo không có nút nào bị quá tải hoặc dưới tải. Các mục tiêu của cân bằng tải liên quan đến [14]:

- Sử dụng tài nguyên tối ưu.
- Điều phối thông lượng tối ưu và đồng đều trên các nút mạng.
- Thời gian đáp ứng, thời gian xử lý yêu cầu tối thiểu.
- Tránh quá tải trên các nút mạng.

Các mô hình và các thuật toán khác nhau để cân đối tài nguyên trong điện toán đám mây tải đã được phát triển với mục đích để người dùng cuối truy cập tài nguyên điện toán đám mây một cách nhanh chóng và thuận tiện. Điện toán đám mây giúp chúng ta chia sẻ dữ liệu và cung cấp nhiều nguồn tài nguyên cho người dùng. Chính vì thế điện toán đám mây lưu trữ dữ liệu và phân phối tài nguyên trong một môi trường mở. Cân bằng tải được xem là quá trình tìm ra các nút mạng bị quá tải và từ đó chuyển sang các nút khác đang có tải ít hơn hoặc không tải.

Cân bằng tải trên điện toán đám mây nhằm tối ưu hóa việc sử dụng các tài nguyên có sẵn, tối đa hóa thông lượng, giảm thiểu thời gian đáp ứng và tránh quá tải cho bất kỳ tài nguyên nào. Thay vì sử dụng một thành phần duy nhất, cân bằng tải tận dụng nhiều tài nguyên, bằng cách tăng độ tin cậy và tính sẵn sàng của kiến trúc mạng. Chính vì vậy, cân bằng tải trên đám mây có rất nhiều tính năng ưu việt, đây cũng là sự khác biệt của hệ thống cân bằng tải trên điện toán đám mây so với các hệ thống cân bằng tải thông thường:

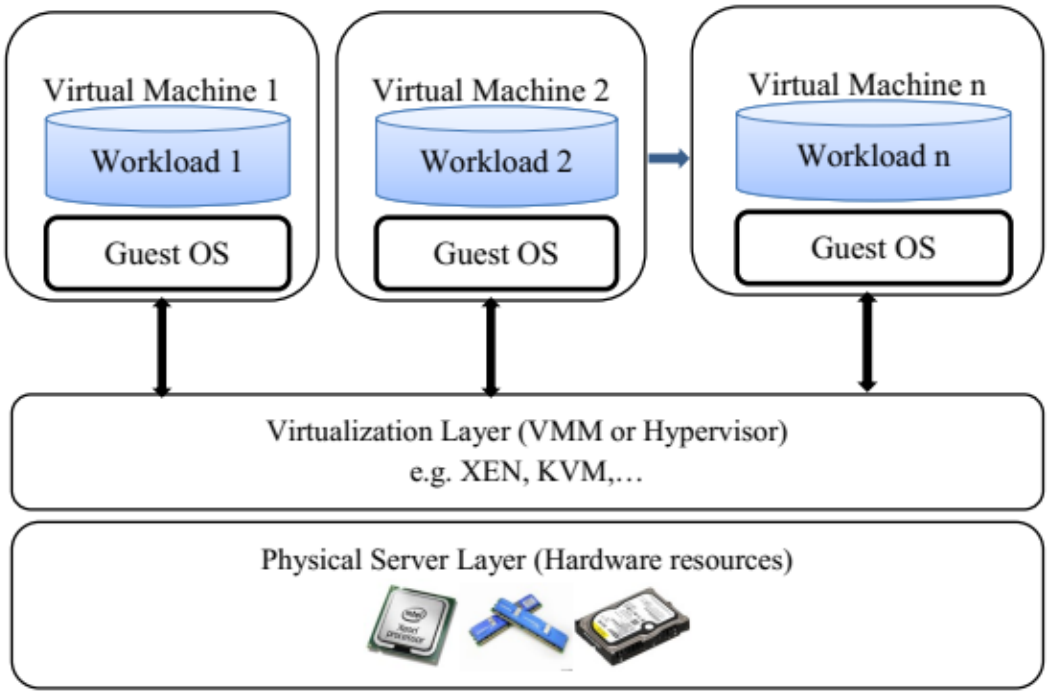
- Tăng thời gian hoạt động của hệ thống
- Tránh lãng phí tài nguyên
- Khả năng chịu tải vượt trội
- Sử dụng các thuật toán tiên tiến
- Hỗ trợ nhiều giao thức: HTTP, HTTPS, LDAP, LDAPS, IMAP, FTP, POP3, POP3S, SMTP.
- Dễ mở rộng mà không làm gián đoạn hệ thống

1.1.4. Ảo hóa và quản lý máy ảo trên đám mây

Công nghệ liên quan mật thiết đến điện toán đám mây là công nghệ ảo hóa, ảo hóa là việc tách hệ điều hành khỏi phần cứng, tạo ra sự di chuyển hệ điều hành và các ứng dụng từ phần cứng này sang phần cứng khác mà mọi thứ vẫn nguyên vẹn. Ảo hóa được thiết kế để tạo ra tầng trung gian giữa hệ thống phần cứng máy tính và phần mềm chạy trên nó. Ý tưởng của ảo hóa máy chủ là từ một máy vật lý đơn lẻ có thể tạo thành nhiều máy ảo độc lập. Mỗi máy ảo đều thiết lập hệ điều hành riêng và các ứng dụng riêng. Ảo hóa có nguồn gốc từ việc phân chia ổ đĩa, chúng phân chia một máy chủ thực thành nhiều máy chủ logic. Khi máy chủ thực được chia, mỗi máy chủ ảo có thể chạy một hệ điều hành và các ứng dụng độc lập. Một máy ảo là một môi trường hoạt động độc lập – phần mềm hoạt động cùng nhưng độc lập với hệ điều hành máy chủ. Nói cách khác, đó là việc cài đặt phần mềm độc lập trên nền của một CPU chạy mã biên dịch. Có rất nhiều loại ảo hóa: ảo hóa mạng, ảo hóa lưu trữ, ảo hóa máy chủ.

Các dịch vụ của điện toán đám mây được sử dụng trong các trung tâm dữ liệu (TTDL) với hàng trăm ngàn máy tính và những TTDL như vậy được xây dựng để phục vụ nhiều người dùng và thực thi nhiều ứng dụng khác nhau. Ý tưởng về sự ảo hóa ra đời từ lâu với mục đích để chia sẻ tài nguyên. Theo tài liệu [13], ảo hóa thông qua nhiều kỹ thuật và các công cụ để quản lý hạ tầng của các TTDL và ảo hóa đã trở thành công nghệ không thể thiếu cho môi trường điện toán đám mây. Sự ảo hóa có thể được định nghĩa như sự trừu tượng của các tài nguyên tính toán gồm: năng lực

tính toán của các bộ xử lý, lưu trữ, bộ nhớ, mạng và I/O. Sự ảo hóa phần cứng cho phép chạy nhiều hệ điều hành và phần mềm trên một nền tảng vật lý đơn. Mô hình phân lớp kiến trúc ảo hóa gồm ba tầng [13]: Tầng tài nguyên phần cứng của máy vật lý (Physical server layer), Tầng ảo hóa (Virtualization layer gồm VMM hay Hypervisor) và Máy ảo (Virtual machine – VM) như Hình 1.4. Tầng ảo hóa là thông qua phần mềm như Virtual Machine Monitor (VMM), còn gọi là Hypervisor điều khiển việc truy cập của các VM vào các tài nguyên phần cứng bên dưới. Hiện nay, các VMM phổ biến cho môi trường điện toán đám mây như VMWare, Xen và KVM. Mỗi máy ảo (VM) có hệ điều hành khách (Guest OS) khác với hệ điều hành của máy vật lý (Host OS), và VM thực thi các ứng dụng khác nhau và độc lập với nhau. Nhiều VM có thể cùng chia sẻ tài nguyên phần cứng bên dưới.



Hình 1.4. Mô hình phân lớp kiến trúc ảo hóa 03 tầng [40]

Công trình của B. Sotomayor và các cộng sự [92] đề xuất mô hình quản lý hạ tầng ảo hóa (Virtual Infrastructure Management - VIM) cho các đám mây riêng (Private cloud) và đám mây hỗn hợp (Hybrid cloud). Tác giả đề xuất OpenNebula

[91], [92] là phần mềm quản lý hạ tầng ảo hóa VIM. OpenNebula khá phổ biến trong việc xây dựng các đám mây riêng. Ý tưởng mà tác giả đề xuất là quản lý máy ảo theo dạng hợp đồng thuê tài nguyên, ông phát triển các giải thuật lập lịch cho máy ảo thành phần mềm Haizea [91], [92]. Các thuật toán lập lịch máy ảo đề xuất quan tâm đến tham số thời gian đợi, thời gian thực thi của các máy ảo. Công trình của R. Buyya và các cộng sự [12] đề xuất kiến trúc Green Cloud bao gồm các thành phần: các máy vật lý, các máy ảo, bộ cấp phát dịch vụ tiết kiệm năng lượng (green service allocator) và khách hàng hoặc các môi giới (brokers). Bài toán phân bổ máy ảo (virtual machine allocation) được đề xuất chia làm hai: giai đoạn đầu là nhận các yêu cầu máy ảo và sắp xếp lên các máy vật lý, giai đoạn sau là tối ưu việc phân bổ hiện tại.

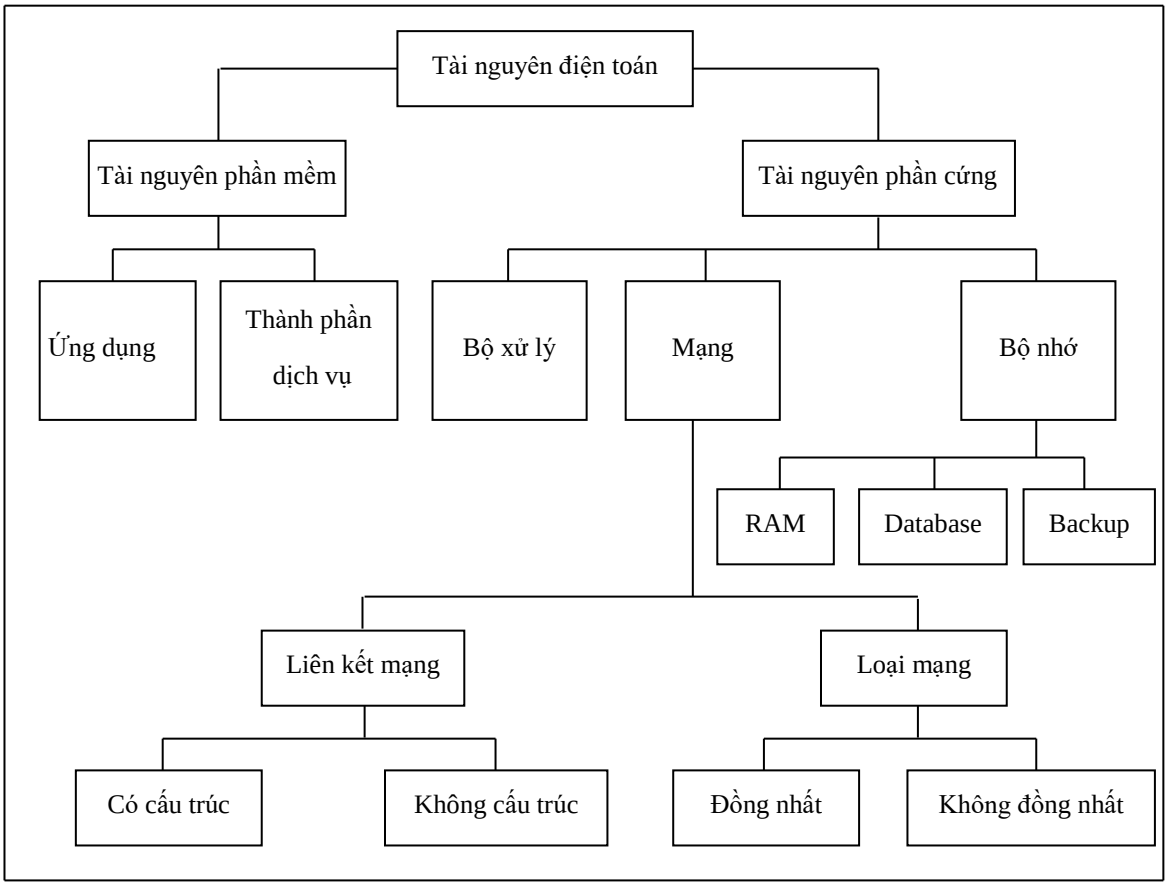
Lợi ích chính của ảo hóa: tiết kiệm năng lượng, thân thiện môi trường; tăng uptime; giảm chi phí, sao lưu dễ dàng; tăng khả năng phục hồi thảm họa.

1.1.5. Quản lý và phân bổ tài nguyên trên điện toán đám mây

Trong điện toán đám mây, phân bổ tài nguyên là quá trình chỉ định động các tài nguyên có sẵn cho các ứng dụng đám mây được yêu cầu. Việc phân bổ tài nguyên xảy ra ở lớp IaaS và tài nguyên sử dụng có thể bao gồm hệ điều hành và ứng dụng cho người dùng. Có thể có hai ứng dụng cố gắng truy cập cùng một tài nguyên cùng một lúc và một số trường hợp xuất hiện khi có tài nguyên hạn chế và nhu cầu tài nguyên cao. Các kỹ thuật phân bổ tài nguyên phải đáp ứng nhiều ứng dụng cần các loại tài nguyên khác nhau như CPU, bộ nhớ, thiết bị I/O [53].

Năm 2016, Sukhpal [95] cùng cộng sự đã tổng hợp các vấn đề và thách thức mà việc phân bổ tài nguyên trên đám mây phải đối mặt. Dựa trên việc khảo sát 110 tài liệu trong tổng số 1206 công trình nghiên cứu về cân bằng tải trên thế giới; đã tổng hợp, mô tả bức tranh tổng thể nhất về cân bằng tải trên đám mây: phân tích hệ thống các phương pháp phân bổ tài nguyên trên đám mây, các thuật toán lập lịch phân bổ tài nguyên và quản lý tài nguyên, các nhóm và ưu điểm của từng nhóm, các chính sách phân bổ tài nguyên. Trong đó bao gồm 13 nhóm thuật toán phân bổ tài nguyên, 8 nhóm chính sách trong việc phân bổ tài nguyên. Tài liệu này nói rõ, việc lập lịch và phân bổ các tải trên đám mây phụ thuộc rất nhiều vào yêu cầu QoS, đồng thời mô tả

một cách tổng quát các phương pháp quản lý tài nguyên trong việc lập lịch và phân bổ tải. Với sự đóng góp này, ta có thể hình dung rất rõ nét về cân bằng tải trên đám mây. Các khảo sát trong nghiên cứu [26] cung cấp một cái nhìn tổng quan ngắn gọn về thuật ngữ và khái niệm cơ bản được sử dụng trong đám mây và khái niệm phân bổ tài nguyên và cân bằng tải trong đám mây. Nghiên cứu này đưa ra mô tả chi tiết về ký hiệu, sơ đồ phân loại đại diện và mô hình về vấn đề lập kế hoạch tài nguyên. Các nghiên cứu của tác giả [1], [60] đã đưa ra các phương pháp lập lịch phân bổ tài nguyên hiệu quả. Trong nghiên cứu [60] đã đề xuất chiến lược lập lịch cho đám mây đa phương tiện và kết quả là tối thiểu hóa thời gian đáp ứng và chi phí tài nguyên trên đám mây. Còn trong nghiên cứu [1] tác giả đề xuất thuật toán lập kế hoạch có xem xét đến độ ưu tiên của các công việc. Thực nghiệm cho thấy thuật toán đề xuất hiệu quả hơn thuật toán FCFS và Round Robin. Theo tài liệu [95] thì có các loại tài nguyên điện toán như sau (Hình 1.5):



Hình 1.5: Phân loại các tài nguyên trên điện toán đám mây [95].

Hình 1.7 cho thấy, có rất nhiều loại tài nguyên cần phải được phân bổ để đảm bảo chất lượng dịch vụ cho phía người dùng. Do đó, cân bằng tải là một trong những thách thức lớn nhất mà điện toán đám mây đang đối mặt, cân bằng tải phải luôn đáp ứng được các yêu cầu người dùng và phân bổ các yêu cầu này một cách linh động nhất giữa các nút hoạt động trên đám mây không được quá tải và không có sự mất cân đối trong việc phân phối tải đến các tài nguyên đám mây.

1.2. Bài toán cân bằng tải

1.2.1. Phát biểu bài toán và mô hình nghiên cứu

Vấn đề cân bằng tải phát sinh khi có nhiều máy chủ ảo xử lý một tập hợp các yêu cầu đầu vào. Giả thiết được đặt ra là tất cả các máy chủ ảo giống hệt nhau về cấu hình và có thể sử dụng để phục vụ bất kỳ yêu cầu nào [43].

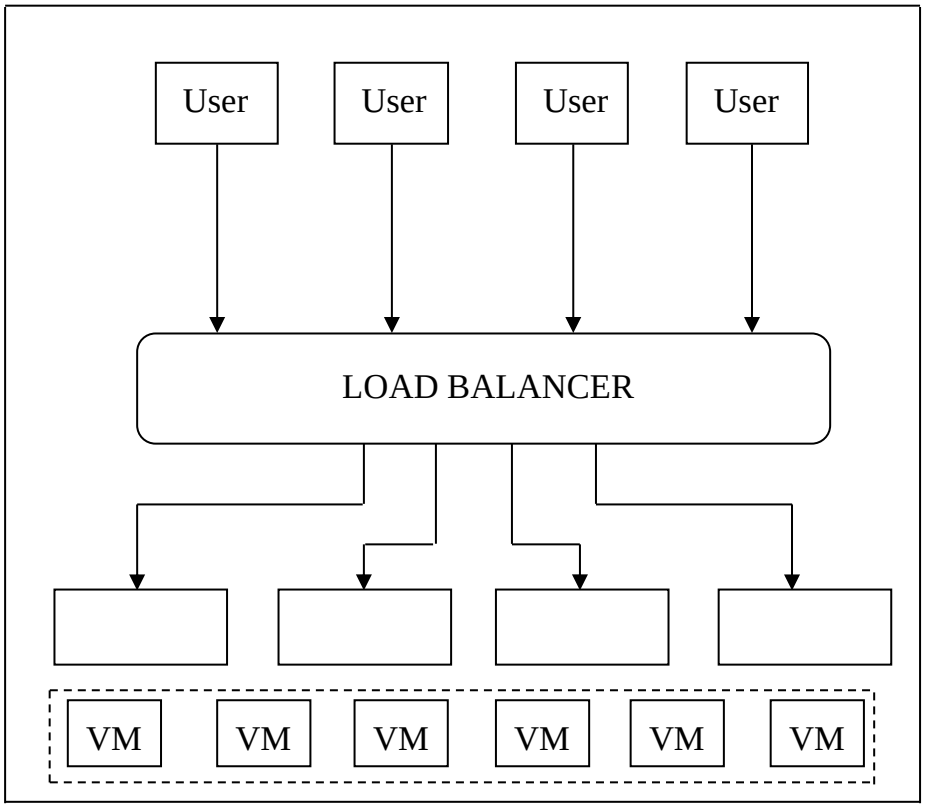
- Đặt vấn đề:
  - Có *m* máy ảo M = {M_1, M_2, M_3,…..,M_m}.
  - Có *n* công việc J = {J_1, J_2, J_3,….., J_n}, với mỗi công việc có thời gian xử lý là $t_j > 0$.
- Yêu cầu đặt ra: Gán tập công việc J cho tập máy ảo M sao cho tải trên tất cả các máy M là đồng đều nhất có thể. Hạn chế tình trạng quá tải trên một máy bất kỳ, trong khi máy khác còn lại thì không phục vụ công việc nào cả.
- Nhiệm vụ chính của luận án: Thông qua phát biểu bài toán như trên, nhận thấy rằng nhiệm vụ chính của đề tài là nghiên cứu phát triển các thuật toán cân bằng tải sao cho giảm được thời gian đáp ứng các yêu cầu đầu vào và giảm thời gian xử lý các yêu cầu. Hay nói cách khác, là tăng hiệu năng của thuật toán cân bằng tải thông qua việc cải thiện các tham số thời gian đáp ứng và thời gian xử lý trên môi trường điện toán đám mây

Đặt A(i) là tập công việc gán cho máy ảo M_i, nên máy M_i cần làm việc trong tổng thời gian [43]:

$$T_i = \sum_{j \in A(i)} t_j \qquad (1.1)$$

Đại lượng T_i là thời gian cần thiết để hoàn thành việc thực hiện tất cả các yêu cầu đầu vào và đó cũng là tải (load) trên các máy M_i [43]. Mục tiêu của bài toán là tối thiểu hóa đại lượng $T = max_i T_i$, với T là tải lớn nhất trên bất kỳ máy nào. Đại lượng T là lượng tải lớn nhất trên bất kỳ máy ảo nào, và T đại diện cho thời gian cần thiết để hoàn thành thực hiện tất cả các yêu cầu đầu vào của bộ cân bằng tải. Do vậy, đại lượng T cần được tối thiểu hoá vì khi tối thiểu hoá được T thì thời gian xử lý các yêu cầu của bộ cân bằng tải sẽ giảm và làm tăng hiệu năng của việc cân bằng tải trên môi trường điện toán đám mây.

Trong môi trường điện toán đám mây, khi một máy ảo quá tải thì các yêu cầu đầu vào phải được gỡ bỏ và gửi đến máy ảo có ít tải để cân bằng tải giữa các máy của cùng một trung tâm dữ liệu [30]. Hình 1.6 biểu diễn mô hình cân bằng tải trên điện toán đám mây như sau [30]. Đây là mô hình tổng quát nghiên cứu cân bằng tải trên điện toán đám mây. Trong đó, yêu cầu của người dùng từ Internet được gửi tới bộ cân bằng tải (Load Balancer) và bộ cân bằng tải này làm nhiệm vụ phân phối các yêu cầu đó đến các máy ảo trong trung tâm dữ liệu một cách đồng đều nhất có thể, nhằm tăng thời gian đáp ứng cũng như phòng tránh hiện tượng quá tải. Hiệu quả của bộ cân bằng tải phụ thuộc rất nhiều vào thuật toán cân bằng tải, mà các thuật toán cân bằng tải chịu sự tác động của các yếu tố ảnh hưởng trực tiếp đến nó. Chính vì thế, trước khi đi sâu nghiên cứu các cách tiếp cận để giải quyết bài toán cân bằng tải, chúng ta phân tích về các yếu ảnh hưởng đến cân bằng tải.



Hình 1.6. Mô hình nghiên cứu cân bằng tải trên điện toán đám mây [30].

1.2.2. Các yếu tố ảnh hưởng đến cân bằng tải

Như đã nói ở trên, việc nghiên cứu các yếu tố ảnh hưởng đến cân bằng tải có ý nghĩa rất to lớn đến quyết định lựa chọn thuật toán cho việc nâng cao cân bằng tải. Có nhiều yếu tố ảnh hưởng đến cân bằng tải trên điện toán đám mây:

- Hạ tầng mạng, lưu lượng mạng, băng thông mạng.
- Cơ chế phân phối tải, cơ chế phòng chống tắc nghẽn.
- Tham số: thời gian đáp ứng, thời gian xử lý, thời gian chờ, mức độ sử dụng tài nguyên, mức độ ưu tiên của các yêu cầu.

Về bản chất, trung tâm dữ liệu đám mây là một hệ thống máy tính kết nối với nhau theo một mô hình mạng cụ thể. Vì vậy, các kết quả nghiên cứu trong công bố trong công trình (CT1) đã nghiên cứu về ảnh hưởng của ma trận lưu lượng đầu vào trong việc quản trị mạng máy tính. Với ma trận lưu lượng đầu vào, ta có thể tính toán để giải quyết các vấn đề của mạng máy tính như: sử dụng băng thông, cân bằng tải,

cải thiện chất lượng mạng. Công trình này cũng đã phân tích các kỹ thuật ước tính và ứng dụng của ma trận lưu lượng vào việc quản trị mạng máy tính.

Cơ chế phân phối tải (CT2), cơ chế phòng chống tắc nghẽn cũng là những tham số ảnh hưởng trực tiếp đến cân bằng tải trên đám mây. Bởi vì, đặc trưng của điện toán đám mây là tính toán phân tán và sử dụng công nghệ ảo hóa nên hiện tượng quá tải có thể xảy ra. Do đó, phân phối tải là vấn đề rất quan trọng để tránh các hiện tượng này, qua đó nâng cao chất lượng dịch vụ, giảm thiểu chi phí để nâng cấp hạ tầng phần cứng, nâng cao hiệu suất của hệ thống và tăng doanh thu của các nhà cung cấp dịch vụ. Tác giả Rashmi. K. S [70] và các cộng sự đã thực hiện một thuật toán cân bằng tải nhằm phòng tránh bế tắc của các máy ảo trong môi trường điện toán đám mây. Ý tưởng của bài báo là chuyển các yêu cầu của người sử dụng sang VM khác mức độ sử dụng thấp hơn nếu xảy ra bế tắc trên một VM nào đó, thuật toán đã làm tăng hiệu quả kinh doanh của các nhà cung cấp dịch vụ đám mây. Bibhudatta Sahoo [10] và nhóm nghiên cứu của ông đã giới thiệu kỹ thuật cân bằng động HDCS trên hệ thống tính toán phân tán không đồng nhất và phân tích tác động của tính không đồng nhất đến năng lực xử lý yêu cầu đến của các nút. Kỹ thuật này thực hiện phân bổ các nguồn lực, tài nguyên một cách có hiệu quả và đã giảm thiểu thời gian tối đa hoàn thành công việc tại mỗi nút. Tác giả Javed Ali [67] đề xuất cơ chế phân loại các yêu cầu, thực hiện phân vùng yêu cầu và cân bằng tải tại các nút đó. Tác giả khẳng định, không có chiến lược phân loại lý tưởng mà tùy thuộc vào từng trường hợp và cơ chế mà nhóm tác giả này đưa ra đã thu được kết quả khả quan, đóng góp vào hướng nghiên cứu hệ thống song song phân tán. Wenhong Tian [106] và các cộng sự đã giới thiệu thuật toán DAIRS tích hợp giữa cân bằng tải động và lập lịch cho trung tâm dữ liệu đám mây, với mục tiêu là tích hợp đo tổng mức độ mất cân bằng của một trung tâm dữ liệu cũng như mức độ mất cân bằng trung bình của mỗi máy chủ. Tác giả Reena Panwar [72] đã nghiên cứu chiến lược phân bổ nguồn lực để đáp ứng toàn bộ các yêu cầu đến các máy ảo. Việc lập lịch cho một hệ thống tính toán phân tán là rất phức tạp, điều khiển các thông số nguồn lực để kiểm soát tình trạng của hệ thống chính xác sẽ nâng cao được hiệu năng hệ thống, giảm được các thông số. Kết quả

nghiên cứu trong công trình (CT2) đã đề xuất cơ chế tích hợp hai hàng đợi cho các yêu cầu đến và mức độ sử dụng của các máy chủ phục vụ cho bộ cân bằng tải điều phối nguồn lực phục vụ khi có yêu cầu đến. Các hàng đợi này được sắp xếp trước khi bộ cân bằng tải sử dụng các thông tin trong đó để ra quyết định chọn máy chủ tương ứng. Thông qua phân tích một số công trình về cơ chế phân phối tải, cho thấy, việc sử dụng cơ chế phân phối tải phù hợp là quan trọng trong việc điều phối tải trên các trung tâm dữ liệu.

Một yếu tố nữa có ảnh hưởng trực tiếp đến cân bằng tải đó là các tham số mà cân bằng tải cần cải thiện như nghiên cứu trong công trình (CT3): thời gian đáp ứng, thời gian xử lý, thời gian chờ, mức độ sử dụng tài nguyên, mức độ ưu tiên của các yêu cầu đầu vào, trạng thái của các nút mạng, băng thông cho máy ảo. Cân bằng tải phụ thuộc vào rất nhiều yếu tố, trong đó có: 1. Khả năng xử lý yêu cầu, khả năng biết trước tình trạng của nút xử lý tiếp theo để tránh tình trạng quá tải, giảm thiểu thời gian xử lý [93]; 2. Thời gian tối đa hoàn thành yêu cầu tại mỗi nút [34], [98]; 3. Phân vùng trong đám mây [21]; 4. Gom cụm trạng thái tải của các máy ảo với cơ chế chia sẻ tải linh hoạt [32]; 5. Thời gian đáp ứng [73]; 6. Tối đa hóa băng thông cho máy ảo [15]. Trong (CT3) nghiên cứu sinh cũng đã tiến hành khảo sát, phân tích ảnh hưởng của các tham số đến các thuật toán cân bằng tải, thể hiện ở Bảng 1.1.

Bảng 1.1. Khảo sát các tham số ảnh hưởng đến phương pháp cân bằng tải

Tác giả	**Phương pháp thực hiện**	**Thông số**	**Kết quả**
A.Govardhan at al [93]	Giải thuật cân bằng động với khả năng giám sát tải tập trung	Trạng thái nút, bộ vi xử lý, yêu cầu công việc	Tối thiểu hóa các lệnh xử lý trên các bộ vi xử lý và giảm lưu lượng mạng

Agraj Sharma et al [2]	Cân bằng tải xét đến thời gian đáp ứng cho mỗi yêu cầu công việc đến	Thời gian đáp ứng, ngưỡng, thời gian đáp ứng dự đoán	Tối đa hóa băng thông mạng
Huankai Chen et al [35]	Giải thuật cân bằng tải Load Balance Improved Min-Min (LBIMM)	Thời gian thực thi tại mỗi nút: lập lịch các yêu cầu; tỉ lệ sử dụng nguồn lực	Giảm thời gian hoàn thành các yêu cầu; cải thiện hiệu năng cân bằng nguồn lực
Dhinesh Babu L.D et al. [15]	Giải thuật HBB-LB (honey bee behavior inspired load balancing)	Mức độ ưu tiên của các yêu cầu, hành vi bầy ong mật	Cân bằng tải có xem xét đến độ ưu tiên của các yêu cầu từ các máy ảo quá tải
Gaochao Xu et al [21]	Mô hình cân bằng tải cho các đám mây công cộng dựa trên khái niệm phân vùng đám mây với một cơ chế chuyển đổi để lựa chọn chiến lược cân bằng hiệu quả	Đám mây công cộng; Phân vùng đám mây	Cải thiện hiệu năng các máy chủ hệ thống khi phân vùng thành các đám mây riêng lẻ
Hiren H. Bhatt et al. [32]	Giải thuật chia sẻ tải linh hoạt FLS (Flexible load sharing algorithm).	Gom nhóm máy chủ; Chia sẻ tải	Phân phối hiệu quả tải cho các máy ảo nhằm giảm thiểu các máy ảo quá tải

Ritu Kapur [73]	Giải thuật LBRS (Load Balanced Resource Scheduling Algorithm)	Thông lượng; Thời gian đáp ứng; Thời gian chờ	Cân bằng tải có xem xét đến tầm quan trọng của việc lập lịch cân bằng nguồn lực
Rashmi. K. S et al [70]	Giải thuật Enhanced Load Balancing Algorithm using Efficient Cloud Management System	Thời gian đáp ứng; Thời gian chờ; Thời gian bế tắc	Đã giảm được hiện tượng tắc nghẽn trong đám mây
Foram Kherani et al [5]	Giải thuật dự báo trạng thái của các nút tiếp theo cho việc phân bổ tải dựa trên phân tích dữ liệu quá khứ	Tải trên các máy chủ; Hiệu suất và hệ số tải trong tương lai (dự báo)	Sử dụng hiệu quả hơn nguồn lực trên đám mây
Reena Panwar et al [16]	Thuật toán quản lý tải động	Phân bổ nguồn lực	Phân phối có hiệu quả toàn bộ các yêu cầu đến các máy ảo

Qua khảo sát ở Bảng 1.1 cho thấy, có rất nhiều yếu tố ảnh hưởng đến hiệu năng của cân bằng tải trên điện toán đám mây như: thuật toán, thời gian đáp ứng, thời gian chờ, thời gian xử lý, thông lượng, mức độ sử dụng tài nguyên. Trong khuôn khổ luận án này chỉ đi sâu nghiên cứu về hai tham số chính là thời gian đáp ứng và thời gian xử lý. Trên cơ sở đó, đề xuất thuật toán cải tiến hai tham số này nhằm nâng cao cân bằng tải trên điện toán đám mây.

1.2.3. Phân loại các thuật toán cân bằng tải

Cân bằng tải trên điện toán đám mây được thực hiện thông qua các thuật toán. Có rất nhiều công trình nghiên cứu cải thiện cân bằng tải và tránh việc sử dụng các nguồn tài nguyên ở mức độ cao bằng việc phát triển các thuật toán cân bằng tải. Thuật toán cân bằng tải được chia thành các loại sau: cân bằng tĩnh, cân bằng động và các thuật toán bổ sung [98]:

- Thuật toán cân bằng tĩnh: không xét đến trạng thái hoặc hành vi trước đó của nút trong khi phân phối tải. Ví dụ: Roud Robin [62], Min- Min [46], Max – Min [69]
- Thuật toán cân bằng động: kiểm tra trạng thái trước đó của một nút trong khi phân phối tải. Ví dụ: DLBS [98], Equally Spread Current Execution [47], Honeybee Foraging Algorithm [100], Biased Random Sampling [61], Active Clustering [102], Throttled Load Balancer [98],
- Thuật toán bổ sung: Cartron [98], LBVS [29].

Bảng 1.2 cho thấy ưu, nhược điểm của một số phương pháp cân bằng tải trên môi trường điện toán đám mây [14].

Bảng 1.2. Ưu, nhược điểm của một số phương pháp cân bằng tải [14].

Thuật toán		**Ưu điểm**	**Nhược điểm**
Cân bằng tĩnh		- Không xem xét tình trạng hiện tại của các nút. - Phù hợp với hệ thống ít phức tạp	- Chỉ giới hạn cho môi trường có tải ít biến đổi
	Round Robin [62]	- Đơn giản, dễ thực hiện - Thực hiện tốt hơn với CPU có chu kỳ thực thi ngắn (short CPU burst)	- Nếu tải lớn thì mất nhiều thời gian. - Các yêu cầu phải tương đương để đạt hiệu suất cao.

	Min-Min [46]	- Thời gian hoàn thành nhỏ nhất - Hiệu quả hơn với các nhiệm vụ có kích thước nhỏ	- Không thể dự báo trước được máy chủ và biến đổi yêu cầu (task)
	Max-Min [69]	- Các yêu cầu được biết trước khi đến, nên hiệu quả hơn	- Tốn thời gian dài để xử lý
	Opportunistic Load Balancing [101]	- Cải thiện hiệu suất - Tận dụng nguồn tài nguyên	- Tốn nhiều thời gian để hoàn thành yêu cầu
Cân bằng động		- Yêu cầu thông tin trạng thái hiện tại của hệ thống - Chịu lỗi	- Cần kiểm tra trạng thái các nút liên tục - Độ phức tạp cao
	Honey Bee [100]	- Tăng thông lượng - Giảm thời gian đáp ứng	- Nhiệm vụ có độ ưu tiên cao không thể làm việc mà không có máy ảo nhàn rỗi
	Ant Colony Optimization [82]	- Tính toán nhanh - Giảm thiểu thời gian xử lý	- Tìm kiếm mất nhiều thời gian - Phức tạp
	Biased Random Sampling [61]	- Cải thiện hiệu suất - Cải thiện việc sử dụng tài nguyên	- Thời gian đáp ứng cao
	Resource Allocation Scheduling [75]	- Tăng hiệu suất - Thời gian thực hiện ngắn hơn	- Khả năng chịu lỗi thấp

1.2.4. Đo lường cân bằng tải

Việc nghiên cứu và cải thiện hiệu năng của điện toán đám mây không thể bỏ qua vấn đề đánh giá hiệu năng từng thuật toán. Hiện nay, có rất nhiều phương pháp để đánh giá hiệu năng một thuật toán: sử dụng mô hình toán học, mô phỏng thực nghiệm, sử dụng các công cụ để đánh giá,…Có rất nhiều các công cụ để mô phỏng các thuật toán trên điện toán đám mây như: CloudSim, Cloud Analyst, GridSim, CSIM for Java, Matlab [22],.. tất cả các công cụ này đã và đang hỗ trợ đắc lực cho việc đánh giá hiệu năng.

Trong công trình [74], tác giả Rodrigo N. Calheiros và các cộng sự đưa ra bộ thư viện CloudSim dùng để hỗ trợ mô hình hóa hệ thống điện toán đám mây. Các thành phần của bộ mô phỏng CloudSim bao gồm: Trung tâm môi giới (Broker), trung tâm dữ liệu, máy ảo, các chính sách cung cấp nguồn tài nguyên và các chính sách quản lý các thành phần khác nhau của hệ thống. Nhờ có các thành phần này mà người dùng có thể đánh giá các chiến lược mới trong việc sử dụng đám mây như các chính sách, thuật toán lập lịch, chính sách cân bằng tải… Bộ thư viện CloudSim dùng để đánh giá hiệu quả của các chiến lược để tăng tốc thời gian thực thi của các ứng dụng. Các lớp thư viện trong CloudSim nhằm mục đích phục vụ cho cho người sử dụng dễ dàng mô phỏng các thuật toán, chiến lược trong cân bằng tải. Tất cả các chi tiết về thông số, mô hình đều được trừu tượng. Sự mô phỏng ở đây được coi như là “chạy một mô hình của một phần mềm trong một mô hình phần cứng”. Thông qua bộ mô phỏng CloudSim đã giải quyết được các vấn đề về việc kiểm tra và thử nghiệm trong điện toán đám mây. Ví dụ như: tiết kiệm năng lượng, thời gian, đánh giá được các thuật toán và ứng dụng cân bằng tải trước khi đưa ra mô trường đám mây thực. Như vậy, bộ mô phỏng CloudSim giúp cho việc kiểm tra hiệu suất cân bằng tải tốt hơn nhờ vào việc tốn ít thời gian để thử nghiệm và tính linh hoạt của nó.

Tác giả Bhathiya Wickremasingle [9] cũng đưa ra bộ công cụ Cloud Analyst để đánh giá môi trường điện toán đám mây. Đây là bộ công cụ mã nguồn mở được phát triển dựa trên nền tảng CloudSim. Nó cho phép mô phỏng và phân tích một môi

trường điện toán đám mây và các ứng dụng quy mô lớn trước khi đưa vào triển khai thực tế. Công trình của tác giả Pericherla S Suryateja [65] so sánh 17 công cụ mô phỏng đám mây dựa trên nhiều tiêu chí cho phép các nhà nghiên cứu chọn một chương trình mô phỏng phù hợp. Có khoảng 47% các chương trình mô phỏng (CloudAnalyst, NetworkCloudSim, EMUSIM, MR-CloudSim, SmartSim, Dynamic CloudSim, CloudSimSDN và CEPSim) được mở rộng từ CloudSim. Có 82% trình mô phỏng (trừ R-CloudSim, secCloudSim, CEPSim) là mã nguồn mở. Đối với hầu hết các trình mô phỏng (76%), ngôn ngữ lập trình được sử dụng là Java. Ngôn ngữ lập trình cơ sở chiếm ưu thế thứ hai là C++ . Chương trình mô phỏng PICS được phát triển bằng Python. Tuy nhiên, công trình [65] khuyến nghị nên sử dụng CloudSim dựa trên các tính năng và mức độ phổ biến của nó trong cộng đồng nghiên cứu.

Các kỹ thuật cân bằng tải trong điện toán đám mây hiện nay xem xét các tham số khác nhau như: hiệu suất, thời gian đáp ứng, thời gian xử lý, khả năng mở rộng, thông lượng, sử dụng tài nguyên, khả năng chịu lỗi, thời gian di trú và chi phí liên quan. Ngoài ra nhằm mục đích tiết kiệm năng lượng thì lượng khí thải cũng được xem xét đến.

1.3. Các hướng giải quyết bài toán cân bằng tải

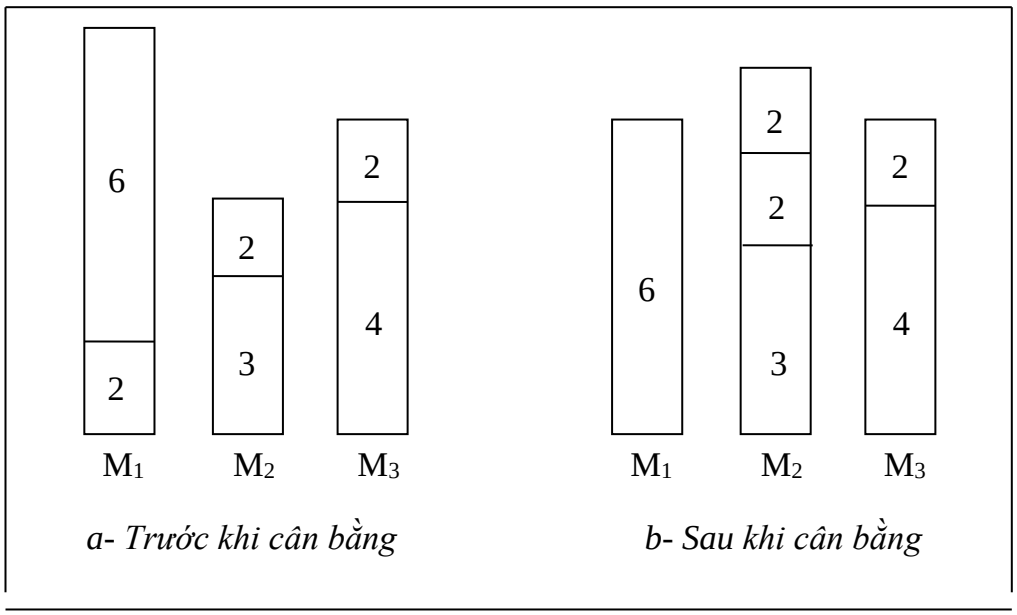
1.3.1. Phương pháp xấp xỉ

Bài toán cân bằng tải là một bài toán NP-Complete [79], hiện chưa có lời giải tối ưu, tuy nhiên có thể giải quyết thông qua các thuật toán xấp xỉ. Phương pháp xấp xỉ là tìm kiếm lời giải cận tối ưu, trong một khoảng thời gian hợp lý. Để thực hiện bài toán cân bằng tải đã phát biểu, thì giả thiết đặt ra là tất cả các máy chủ giống hệt nhau và có thể sử dụng để phục vụ bất kỳ yêu cầu nào. Xét một thuật toán cân bằng tải đơn giản Greedy-Balance [43] để minh họa cho việc giải quyết bài toán cân bằng tải bằng phương pháp xấp xỉ. Thuật toán này gán công việc *j* cho máy có tải nhỏ nhất.

Greedy-Balance:

1. Bắt đầu: Không có công việc nào được gán.
2. Khởi tạo $T_i = 0$ và $A(i) = \emptyset$ cho tất cả các máy Mi.
3. Vòng lặp với j = 1,…,n
4. Đặt M_i là máy có tải tối thiểu ($\min_k T_k$).
5. Gán công việc j cho máy M_i.
6. Đặt $A(i) \leftarrow A(i) \cup \{j\}$
7. Đặt $T_i \leftarrow T_i + t_j$
8. Kết thúc vòng lặp.

Thủ tục Greedy-Balance [43] này chạy cho một chuỗi gồm 6 công việc với kích thước mỗi công việc tương ứng: 2, 3, 4, 6, 2, 2 và có 3 máy ảo M_1, M_2, M_3. Từ Hình 1.7 ta thấy, ban đầu các máy ảo có makespan lần lượt là 8,5,6 (makespan hệ thống = 8) và sau khi chạy thuật toán này thì makespan đã giảm xuống còn 6, 7, 6. Rõ ràng là thuật toán đã có sự cân bằng tải tốt hơn và đã tối tưu được tải hệ thống (makespan hệ thống = 7).



Hình 1.7. Kết quả chạy thuật toán Greedy-Balance

Giả sử T* là giá trị makespan tối ưu, thì nhiệm vụ của các thuật toán cân bằng tải là tìm ra T với kỳ vọng T không lớn hơn nhiều so với T*. Chúng ta không thể biết

được rằng giá trị T* là bao nhiêu nhưng T* luôn có giới hạn dưới [43]. Có nhiều cách xác định giới hạn dưới, công trình [43] đã dựa vào tổng thời gian xử lý $\sum_j t_j$ để tính toán. Một máy ảo phải làm việc ít nhất là $\frac{1}{m}$ công việc trong tổng số các công việc. Do đó, ta có một số nhận xét về T và T* trong thuật toán Greedy-Balance như sau:

- Ngưỡng dưới của giá trị makespan tối ưu T*:

$$T^* \geq \frac{1}{m}\sum_j t_j \tag{1.2}$$

Giá trị T* luôn có ngưỡng dưới, tuy nhiên, trong một số trường hợp thì bổ đề này không đủ mạnh; ví dụ: có một công việc có thời gian xử lý cực kỳ dài so với tổng thời gian xử lý của tất cả các công việc còn lại thì giải pháp hiệu quả là gán công việc đó cho một máy ảo duy nhất nào đó và nó sẽ là máy cuối cùng kết thúc phiên làm việc. Trong trường hợp này, ngưỡng dưới được mở rộng như sau:

- Ngưỡng dưới của giá trị makespan tối ưu T* mở rộng:

$$T^* \geq max_j t_j \tag{1.3}$$

- Thuật toán Greedy-Balance tạo ra sự phân công công việc cho các máy ảo với makepan là:

$$T \leq 2T^*. \tag{1.4}$$

Ta chứng minh công thức (1.4) để hiểu rõ hơn phương pháp xấp xỉ trong cân bằng tải:

Khi phân tích một thuật toán gần đúng, người ta thường so sánh giải pháp thu được với những gì đã biết về giá trị tối ưu, trong trường hợp này là các giới hạn dưới (1.2) và (1.3). Xét một máy M_i đạt tải tối đa T và đòi hỏi xác định công việc cuối cùng j được gán cho M_i là gì? Nếu t_j không quá lớn so với hầu hết các công việc khác, thì giới hạn dưới là công thức (1.2), còn nếu t_j là quá lớn thì giới hạn dưới là công thức (1.3). Khi gán công việc j cho M_i, máy M_i có tải nhỏ nhất so với bất kỳ máy ảo nào; đây là đặc tính chính của thuật toán Greedy-Balance. Tải trọng của M_i ngay trước khi thực hiện phép gán này là $(T_i - t_j)$ và vì đây là tải nhỏ nhất tại thời điểm đó, nên tải trên tất cả các máy ít nhất là bằng $(T_i - t_j)$. Do đó, cộng tải của tất cả các máy ta có:

$$\sum_k T_k \geq m(T_i - t_j) \quad (1.5)$$

Tương đương với: $T_i - t_j \leq \frac{1}{m}\sum_k t_k$ (1.6)

Nhưng giá trị $\sum_k T_k$ chỉ là tổng tải của tất cả các công việc $\sum_j t_j$ (vì mỗi công việc được gán cho chính xác một máy), nên đại lượng $\frac{1}{m}\sum_k t_k$ là giới hạn dưới của giá trị tối ưu (1.2). Như vậy:

$$T_i - t_j \leq T^* \quad (1.7)$$

Và chúng ta tính toán lượng tải còn lại trên M_i, của công việc cuối cùng j. Ở đây thuật toán sử dụng giới hạn dưới là công thức (1.3):

$$t_j \leq T^* \quad (1.8)$$

Cộng 2 bất đẳng thức (1.7) và (1.8) ta được:

$$T_i = (T_i - t_j) + t_j \leq 2\,T^* \quad (1.9)$$

Vì makepan $T = T_i$, nên công thức (1.4) được chứng minh.

Ví dụ: Giả sử có *m* máy và $n = m\,(m - 1) + 1$ công việc. Với $(n - 1) = m(m - 1)$ đầu tiên, mỗi công việc yêu cầu thời gian là $t_j = 1$, còn công việc cuối cùng lớn hơn rất nhiều đòi hỏi thời gian $t_n = m$. Thuật toán Greedy-Balance sẽ cân bằng đồng đều $(n - 1)$ công việc đầu tiên, và sau đó thêm công việc khổng lồ *n* vào một máy còn lại, kết quả tạo ra makespan là $T = 2m - 1$. Trong ví dụ này, giá trị makespan tối ưu là *m* do đó (2m-1)/m = 2- 1/m, khi *m* là rất lớn thì tỉ lệ makespan của thuật toán Greedy-Balance và giá trị tối ưu bằng 2, thì đây chính là trường hợp xấu nhất của thuật toán Greedy-Balance ($T = 2\,T^*$). Trên thực tế, có thể cải tiến thuật toán xấp xỉ để tỉ lệ makespan thu được và makespan tối ưu nhỏ hơn 2, và tỉ lệ này càng nhỏ càng tốt.

Xét thuật toán Sorted-Balance [43], thuật toán này có tỉ lệ nói trên là 1,5.

Sorted-Balance:

1. Bắt đầu: Không có công việc nào được gán.
2. Đặt Ti = 0 và A(i) = ∅ cho tất cả các máy Mi
3. Sắp xếp công việc theo thứ tự giảm thời gian xử lý t_j.

4. Giả sử rằng t1 ≥ t2 ≥. . . ≥ tn
5. Vòng lặp j = 1,…,n
6. Đặt Mi là máy có tải tối thiểu ($\min_k T_k$)
7. Gán công công việc j cho máy M_i
8. Đặt A(i) ← A(i) U {j}
9. Đặt Ti ← Ti + t_j
10. Kết thúc vòng lặp.

Nếu có ít hơn *m* công việc, thì giải pháp Greedy-Balance rõ ràng sẽ là tối ưu, vì nó đặt mỗi công việc vào trên một máy. Và nếu có nhiều hơn *m* công việc, thì có thể sử dụng ràng buộc dưới đây về mức makespan tối ưu:

- Nếu có nhiều hơn *m* công việc, thì:

$$T^* \geq 2t_{m+1} \tag{1.10}$$

- Thuật toán Sorted-Balance tạo ra sự phân công công việc cho các máy có makepan là:

$$T \leq 3/2\ T^* \tag{1.11}$$

Ta nhận thấy, đã có sự cải tiến makspan từ công thức (1.4) và (1.10), tuy không thể tính toán chính xác giá trị của T^*, nhưng rõ ràng đã có thuật toán cân bằng tải để tính toán giá trị T tiệm cận đến giá trị tối ưu T*. Điều đó có nghĩa là có thể dùng thuật toán xấp xỉ để giải quyết bài toán cân bằng tải.

1.3.2. Chiến lược lập lịch phân bổ tài nguyên

Hướng nghiên cứu tiếp theo là cải tiến các chiến lược lập lịch phân bổ tài nguyên nhằm cân bằng nguồn lực trên điện toán đám mây [7], [16], [19], [25], [33], [41], [45], [48], [59], [63], [87], [97], [107]. Công trình [16] tác giả đề xuất một thuật toán quản lý tải động để phân phối có hiệu quả toàn bộ các yêu cầu đến các máy ảo. Giải thuật được mô phỏng bằng công cụ CloudAnalyst dựa trên các thông số khác nhau như thời gian xử lý dữ liệu và thời gian đáp ứng,.. và so sánh kết quả với thuật toán VMAssign. Kết quả mô phỏng đã chứng minh rằng thuật toán phân phối tải thống nhất giữa các máy chủ thông qua việc sử dụng hiệu quả các nguồn tài nguyên. Thông số được quan tâm nhất ở đây là Resource Allocation (phân bổ nguồn lực). Nghiên

cứu của tác giả Mahesh B. Nagpure [54] đã đề ra một chiến lược phân bổ nguồn lực động đã ngăn chặn tình trạng quá tải bằng phương pháp cân bằng tải hiệu quả. Thuật toán đưa ra một khái niệm là độ lệch (skewnees) dùng để đo độ không đồng đều của các nguồn tài nguyên sử dụng trên máy chủ. Bên cạnh đó thuật toán cũng đưa ra được phương pháp dự báo tải trong tương lai để phân phối tải hiệu quả hơn. Các công trình [20], [40] đã đề xuất các chiến lược quản lý lưu tải động nhằm tối đa hóa thông lượng.

Công trình [51], [56], [57], [95] đưa ra phương pháp tối đa hóa việc sử dụng tài nguyên bằng mô hình cân bằng tải, các phương pháp này hỗ trợ trong việc dự báo xu hướng của các tài nguyên, việc cấp phát tài nguyên động và việc giải phóng bộ nhớ của các máy chủ một cách hiệu quả. Trong tài liệu [99], tác giả Syed Hamid tổng hợp các nghiên cứu và các kỹ thuật phân bổ tài nguyên trên đám mây, là bước thực hiện sau khi lập lịch cho tài nguyên. Năm 2016 tác giả V.Krishna [105] đã công bố nghiên cứu về phân tích hiệu năng các thuật toán cân bằng tải trên môi trường đám mây, nghiên cứu này đã phân tích so sánh các nhóm thuật toán: ACO (Ant Colony Optimization), FCFS, GA (Genetic Algorithm), Multi-Objective Model for Scheduling, MOACA (Multi Objective Ant Colony Algorithm). Trong công trình [21] tác giả Gaochao Xu và nhóm nghiên cứu cho rằng cân bằng tải trong môi trường điện toán đám mây có một tác động quan trọng về hiệu suất của hệ thống. Cân bằng tải tốt làm cho điện toán đám mây hiệu quả hơn và cải thiện sự hài lòng của người sử dụng.

Công trình [21] giới thiệu một mô hình cân bằng tải cho các đám mây công cộng (Public Cloud) dựa trên khái niệm phân vùng đám mây (Cloud Partitioning) với một cơ chế chuyển đổi để lựa chọn chiến lược khác nhau cho các tình huống khác nhau. Việc chia đám mây ra thành các phân vùng sẽ góp phần xây dựng được chiến lược cân bằng tải hiệu quả. Qua đây ta thấy, để nâng cao được hiệu quả của việc cân bằng tải trên điện toán đám mây thì việc phân chia thành các phân vùng đám mây riêng lẻ sẽ nâng cao hiệu năng toàn hệ thống máy chủ. Hiren H. Bhatt [32] và cộng sự đã đề xuất một thuật toán tăng cường cân bằng tải trong đó chú ý tới các tham số: nhóm máy ảo, phân loại tải và kỹ thuật chia máy ảo thành các nhóm khác nhau (dưới tải,

cân bằng và quá tải) sẽ linh hoạt trong việc phân phối tải cho các máy ảo do đó sẽ làm giảm các máy ảo quá tải. Những thách thức đối với việc lập lịch trình tài nguyên bao gồm: sự phân tán, tính không chắc chắn và tính không đồng nhất của tài nguyên trong môi trường đám mây [4],[96]. Lập lịch trình tài nguyên bao gồm ba chức năng: Lập bản đồ tài nguyên, Thực thi tài nguyên và giám sát tài nguyên.

Các công trình gần đây nhất [24], [36], [71], [88] cũng đã tập trung nghiên cứu cải tiến các chiến lực phân bổ tài nguyên. Tài liệu [88] đề xuất cơ chế tránh lãng phí tài nguyên, công trình [36] đề xuất cơ chế di trú trực tiếp các yêu cầu hiệu quả, giảm chi phí về thời gian thực hiện. Công trình [24], [71] đề xuất các chiến lược phân bổ tải cho các máy ảo một cách hiệu quả.

1.3.3. Phương pháp cải tiến các tham số

Một hướng nghiên cứu nữa là cải tiến các tham số ảnh hưởng đến khả năng cân bằng tải thu hút rất nhiều tác giả trên thế giới [2], [8], [17], [34], [37-39], [49], [52], [58], [90], [94]. Các tham số cơ bản đó là: thời gian đáp ứng, thời gian xử lý, thời gian hoàn thành, tỉ lệ sử dụng tài nguyên, độ ưu tiên của các nhiệm vụ... Nhóm tác giả Agraj Sharma trong công trình [2] đã cho rằng yếu tố thời gian đáp ứng ảnh hưởng lớn đến hiệu năng cân bằng tải trên điện toán đám mây. Tác giả [2] nêu ra 2 vấn đề còn tồn tại của các giải thuật trước đây là: 1) cân bằng tải chỉ xảy ra sau khi các máy chủ bị quá tải; 2) liên tục truy vấn thông tin tài nguyên sẵn có dẫn đến tăng chi phí tính toán và tiêu thụ băng thông. Vì vậy, tác giả đã đề xuất thuật toán cải tiến thời gian đáp ứng của các yêu cầu để quyết định gán các yêu cầu cho các máy chủ một cách thích hợp, cách tiếp cận của giải thuật này đã giảm được sự truy vấn thông tin về các nguồn lực sẵn có, giảm sự giao tiếp và tính toán trên mỗi máy chủ. Kết quả được mô phỏng với phần mềm ECLIPSE IDE using JAVA 1.6 đã chứng minh được sự đúng đắn của thuật toán đề xuất.

Theo [34] thuật toán Min – Min đã tối thiểu hóa được thời gian hoàn thành công việc ở mỗi nút mạng tuy nhiên giải thuật này vẫn chưa xem xét đến khối lượng công việc của mỗi tài nguyên. Vì vậy, tác giả đưa ra giải thuật Load Balance Improved

Min-Min (LBIMM) để khắc phục điểm yếu này. Sau khi đưa ra thuật toán với việc xem xét khối lượng công việc của mỗi tài nguyên để khắc phục hạn chế của giải thuật Min – Min truyền thống thì giải thuật LBIMM [34] cho kết quả thực nghiệm tốt hơn. Tác giả Dhinesh Babu L.D [15] cho rằng, việc nghiên cứu nhằm tối đa hóa thời gian xử lý, thời gian đáp ứng trên điện toán đám mây cũng được quan tâm, việc cân bằng tải cho các công việc độc lập không ưu tiên là rất quan trọng trong các giải thuật lập lịch trên đám mây. Với việc đề xuất giải thuật HBB-LB với các tham số cơ bản là: thời gian xử lý, thời gian đáp ứng. Tác giả đưa ra tham số là độ lệch chuẩn của tải:

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(PT_i - PT)^2} \quad (1.12)$$

Trong đó:

- PT_i : là thời gian xử lý của máy ảo VM_i
- PT : là tổng thời gian xử lý của tất cả máy ảo VM_s

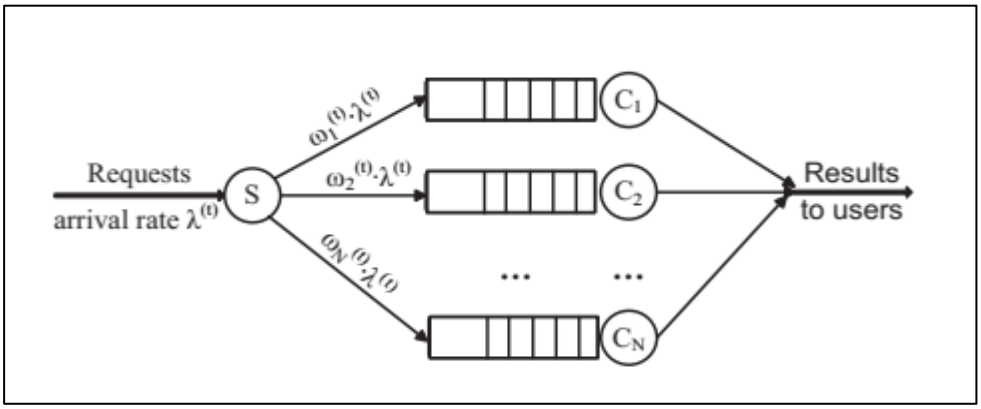
Căn cứ vào giá trị độ lệch chuẩn của tải σ để có chính sách cân bằng tải thích hợp. Nếu độ lệch chuẩn σ nhỏ hơn hoặc bằng ngưỡng $T_s \in [0 - 1]$ thì hệ thống đã cân bằng, nếu không thì hệ thống có thể quá tải hoặc dưới tải. Ở đây các máy ảo được gom thành nhóm dựa trên tải của chúng, phân thành 3 nhóm: nhóm cân bằng (Balanced VMs), nhóm quá tải (Overload VMs) và nhóm dưới tải (Underload VMs). Như vậy, việc gom nhóm trạng thái tải của các máy ảo cũng là một vấn đề nhằm nâng cao hiệu năng của việc cân bằng tải trong đám mây.

Trong công trình [70], nhóm tác giả Rashmi. K.S đã nghiên cứu nâng cao khả năng cân bằng tải tránh tắc nghẽn trong điện toán đám mây, tác giả đưa ra thuật toán Enhanced Load Balancing Algorithm using Efficient Cloud Management System, thuật toán này nhằm khắc phục các hạn chế là đã giảm được hiện tượng tắc nghẽn trong lưu thông. Các tham số được nghiên cứu gồm: thời gian đáp ứng, thời gian chờ, thời gian bế tắc. Foram Kherani và cộng sự [5] đã đề ra một giải thuật dự báo trạng thái của các nút tiếp theo cho việc phân bổ tải dựa trên phân tích dữ liệu quá khứ để

sử dụng nguồn lực tốt hơn cho đám mây. Các tham số được nghiên cứu ở trong công trình này là: tải trên các máy chủ, hiệu suất và hệ số tải trong tương lai (dự báo).

Các công trình gần đây [27], [64], [85], [86], [103] cũng đã công bố những kết quả cải tiến các tham số ảnh hưởng trực tiếp đến cân bằng tải. Bằng các phương pháp khác nhau, các tác giả đã thực hiện cải tiến các tham số chính và đã cho kết quả tốt hơn các phương pháp đã công bố trước đó.

Thời gian đáp ứng là tham số rất quan trọng, nó ảnh hưởng đến hiệu năng của đám mây cũng như là QoS của các nhà cung cấp dịch vụ điện toán. Chính vì vậy, đã có nhiều thuật toán cân bằng tải đề xuất với mục tiêu tối ưu hóa thời gian đáp ứng trên điện toán đám mây [18], [29], [33], [56], [60], [108]. Thuật toán cân bằng tải chỉ xem xét đến thời gian đáp ứng của từng yêu cầu [108], thuật toán này không chỉ mang tính linh động, mà còn làm giảm sự giao tiếp và tính toán thêm trên mỗi máy chủ. Thuật toán lập lịch [60], được xây dựng để giảm thiểu thời gian đáp ứng bằng cách tối ưu hóa trọng số khối lượng công việc trên các khe thời gian t (time slot) của các cụm ảo C_i (virtual cluster) khác nhau.



Hình 1.8. Mô hình lập lịch khối lượng công việc cho đám mây [60].

Với tỉ lệ các yêu cầu đến trung bình của khe thời gian *t* được mô hình hóa theo phân bố Possion là $\lambda^{(t)}$ và $\omega_1^{(t)}$ là trọng số tương ứng [60]. Khi đó thời gian đáp ứng của từng cụm máy ảo là:

$$T_i = \frac{1}{(K_i^{r(t)} + K_i^{d(t)})\mu_i - \omega_i^{(t)}.\ \lambda^{(t)}} \qquad (1.13)$$

Và tổng thời gian đáp ứng là:

$$T = \sum_{i=1}^{N}(\omega_i \ . \ T_i) = \sum_{i=1}^{N} \frac{\omega_i^{(t)}}{(K_i^{r(t)} + K_i^{d(t)})\mu_i - \omega_i^{(t)} . \ \lambda^{(t)}} \qquad (1.14)$$

Trong đó:

- μ_i : là tỉ lệ dịch vụ của một cá thể VM của cụm C_i.
- $(K_i^{r(t)} + K_i^{d(t)})\mu_i$: biểu diễn tỉ lệ dịch vụ của cụm C_i
- $\omega_i^{(t)} . \ \lambda^{(t)}$: đại điện cho tỉ lệ đến trung bình theo phân phối Possion

Về mặt toán học, bài toán tối thiểu hóa thời gian đáp ứng có thể được xây dựng như sau:

- Mục tiêu: $\frac{Minimize}{\{\omega_1,\omega_1,\ldots,\omega_N\}} \sum_{i=1}^{N} \frac{\omega_i^{(t)}}{(K_i^{r(t)} + K_i^{d(t)})\mu_i - \omega_i^{(t)} . \ \lambda^{(t)}}$
- Ràng buộc:
  - Ràng buộc 1: $\omega_i^{(t)} . \ \lambda^{(t)} < (K_i^{r(t)} + K_i^{d(t)})\mu_i, \text{Với } \forall i = 1, \ldots, N$
  - Ràng buộc 2: $\sum_{i=1}^{N} \omega_i = 1, \ \omega_i \geq 0, \text{Với } \forall i = 1, \ldots, N$

Ràng buộc 1 đại diện cho độ ổn định hàng đợi tại cụm C_i, ràng buộc 2 đại diện cho việc lập lịch theo trọng số công việc. Vấn đề tối thiểu hóa thời gian đáp ứng là vấn đề tối ưu hóa lồi [33]. Tác giả [60] sử dụng phương pháp số nhân Lagrange để giải quyết vấn đề và có được giải pháp phân tích tối ưu như sau:

$$\omega_i^{(t)} = \frac{\lambda^{(t)} \cdot \sqrt{\xi_i^{(t)}} + \xi_i^{(t)} \cdot \sum_{j=1}^{N} \sqrt{\xi_j^{(t)}} - \sqrt{\xi_i^{(t)}} \cdot \sum_{j=1}^{N} \xi_j^{(t)}}{\lambda^{(t)} \cdot \sum_{j=1}^{N} \sqrt{\xi_j^{(t)}}} \qquad (1.15)$$

Trong đó: $\xi_i^{(t)} = (K_i^{r(t)} + K_i^{d(t)})\mu_i$ biểu diễn tỉ lệ dịch vụ của cụm C_i

Tác giả sử dụng chiến lược lập lịch heuristic với tỉ lệ dịch vụ được chuẩn hóa làm trọng số lập lịch:

$$\frac{\xi_i^{(t)}}{\sum_{j=0}^{N} \xi_j^{(t)}} \qquad (1.16)$$

Kết quả mô phỏng chứng minh rằng sơ đồ lập lịch khối lượng công việc được đề xuất [81] có thể cân bằng tối ưu khối lượng công việc để đạt được thời gian đáp ứng tối thiểu.

Trong thuật toán cân bằng tải WMaxMin [29], tác giả đã kết hợp hai thuật toán Max – Min và Weighted Round Robin Algorithm để cân bằng tải hiệu quả bằng cách xem xét 2 tham số: thời gian đáp ứng và thời gian chờ. Ý tưởng chủ yếu của WMaxMin là làm sao để không có máy ảo nào quá tải (over loaded) hoặc dưới tải (under loaded), tải được cân bằng giữa tất cả các máy ảo và sẽ được giao cho máy được chỉ định hoặc máy dự định để xử lý yêu cầu, nếu phải chờ đợi bất kỳ yêu cầu nào, nó sẽ lại được giao cho bộ lập lịch để lên lịch yêu cầu theo sự sẵn có của máy ảo. Việc mô phỏng thông qua CloudSim cho thấy, thuật toán đã cải thiện được thời gian đáp ứng.

Trên điện toán đám mây, service broker kiểm soát định tuyến lưu lượng giữa người dùng và trung tâm dữ liệu dựa trên các chính sách môi giới dịch vụ khác nhau. Chính sách định tuyến dựa trên dịch vụ chọn trung tâm dữ liệu gần nhất để định tuyến yêu cầu người dùng. Nếu có nhiều hơn một trung tâm dữ liệu trong cùng một khu vực, nó sẽ chọn ngẫu nhiên mà không xem xét khối lượng công việc, chi phí, thời gian xử lý hoặc các tham số khác. Trung tâm dữ liệu được chọn ngẫu nhiên có xu hướng cho kết quả không đạt yêu cầu. Do đó, tác giả [29] đã đề xuất sửa đổi chính sách vùng lân cận (proximity-based) đó bằng cách áp dụng thuật toán lịch biểu mới để cải thiện thời gian đáp ứng, kiểm soát cân bằng tải. Chính sách này thường áp dụng cho các trung tâm dữ liệu có phân bố địa lý. Trong công trình [14] tác giả kết hợp 2 thuật toán SHC + JIQ để tạo thành thuật toán lai SIQ cung cấp một phương pháp cân bằng tải hiệu quả và thời gian đáp ứng được cải tiến. Hạn chế chính trong thuật toán SHC là quá tải các yêu cầu tại một trung tâm dữ liệu đơn. Do đó, đây là một thuật toán không hiệu quả để cân bằng tải. Vì vậy, SHC được kết hợp với cách tiếp cận JIQ để xây dựng nên thuật toán SIQ nhằm mục đích cải thiện thời gian đáp ứng và sử dụng tài nguyên tốt hơn. Thuật toán SIQ đã cải thiện được thời gian đáp ứng và sử dụng tài nguyên hiệu quả.

Trong bài báo [69] nhóm tác giả đã cải tiến thuật toán Max- Min để tạo ra một thuật toán tối ưu hơn. Bằng việc sử dụng các thông số đầu vào như: số công việc, số lượng nguồn tài nguyên. Khi lượng yêu cầu tới hàng đợi, Bộ xử lý đám mây sẽ tìm yêu cầu có thời gian thực thi lớn nhất sau đó chỉ định cho tài nguyên có thể hoàn thành sớm nhất. Phương pháp cải tiến này có cùng độ phức tạp giống thuật toán Max-Min trước đó nhưng nó đem lại hiệu quả khá tốt thể hiện ở chỗ thời gian thực thi tại mỗi tài nguyên ít hơn và sơ đồ lập lịch khá tin cậy. Thuật toán cải tiến Max-min giúp cân bằng tải tốt hơn các nguồn lực sẵn có và cũng giúp thực hiện đồng thời các nhiệm vụ được đưa ra với xác suất cao hơn so với thuật toán Max-min gốc. Phương pháp này được đánh giá khá tốt, tuy nhiên chỉ tập trung vào nghiên cứu tới hai yếu tố chính là lượng yêu cầu và nguồn tài nguyên. Hơn nữa trong một số trường hợp khối lượng yêu cầu quá nhiều, hoặc bản thân tài nguyên đang xử lý một công việc được yêu cầu thì việc đánh giá khả năng thực hiện công việc kế tiếp sẽ không chuẩn xác và tối ưu.

Có nhiều tham số ảnh hưởng đến khả năng cân bằng tải trên điện toán đám mây, tuy nhiên trong khuôn khổ luận án chỉ nghiên cứu 2 tham số chính phục vụ mục tiên nghiên cứu đó là: thời gian đáp ứng, thời gian xử lý. Các nghiên cứu về thời gian đáp ứng được thực hiện trong Chương 2, nghiên cứu về thời gian xử lý được thực hiện trong Chương 3.

1.4. Các vấn đề mà luận án cần giải quyết

Thông qua khảo sát, đánh giá các công trình nghiên cứu liên quan ở mục 1.2 và 1.3, nghiên cứu sinh rút ra một số nhận xét:

- Công trình [89] là một phương pháp cân bằng tải động sử dụng tài nguyên khá tốt vì thế nên đạt được hiệu năng tương đối tốt. Khi có yêu cầu đến bộ phận quản lý đám mây tìm trong danh sách máy ảo đang sẵn sàng và chọn máy ảo đầu tiên khi nó tìm thấy. Cách này tương đối hợp lý trong việc đảm bảo tính nhanh gọn, nhưng với trường hợp máy ảo đó sẵn sàng nhưng không đủ năng lực thực hiện yêu cầu thì sẽ dẫn đến việc chậm trễ và hao tốn chi phí thời gian. Thuật toán [89] đã không xem xét tới lượng tải hiện tại của máy ảo, vì vậy khi

một máy ảo không đủ năng lực thực hiện yêu cầu nó sẽ phải quay trở lại tìm một máy ảo tiếp theo, hệ thống sẽ tiêu tốn thời gian chờ và thời gian đáp ứng hơn. Vấn đề này được giải quyết trong Chương 2 và kết quả nghiên cứu đã được công bố ở công trình (CT4).

- Phương pháp giảm thời gian xử lý của thuật toán trong công trình [89] tồn tại một vấn đề là: bộ cân bằng tải phải tìm kiếm toàn bộ danh sách các máy ảo để tìm ra máy ảo nào đang sẵn sàng cho việc phân bổ tải, điều này làm tăng thời gian xử lý các công việc. Do đó, nhiệm vụ của đề tài là đưa ra giải pháp khắc phục vấn đề này. Phương pháp giải quyết và kết quả nghiên cứu được thực hiện trong Chương 3 và đã được công bố trong công trình (CT5).
- Cũng nhằm mục đích cải thiện thời gian xử lý, thuật toán Max – Min [69] đã có những kết quả tốt, tuy nhiên điểm hạn chế là bộ cân bằng tải duyệt trên toàn bộ danh sách các máy ảo để thực hiện việc gán yêu cầu cho máy ảo nào thỏa mãn yêu cầu, vấn đề này sẽ làm tăng thời gian xử lý, do các thao tác này lặp đi lặp lại. Do đó, vấn đề đặt ra là phải giảm thiểu các các thao tác lặp lại này để cải thiện thời gian xử lý. Và vấn đề này cũng được giải quyết trong Chương 3, các kết quả nghiên cứu cũng đã được công bố trên công trình (CT6).
- Thuật toán RRTA (CT7): Sử dụng thuật toán dự báo ARIMA để cải thiện thời gian đáp ứng, bằng cách đưa ra cách giải quyết phân phối tài nguyên hợp lý. Vấn đề này được trình bày trong Chương 2 của Luận án.

1.5. Kết luận Chương 1

Chương 1 trình bày tổng quan về cân bằng tải và sự cần thiết của cân bằng tải trên điện toán đám mây. Việc quản lý máy ảo và các chiến lược lập lịch phân bổ tài nguyên trên đám mây cũng được trình bày. Tiếp theo, Chương 1 trình bày bài toán cân bằng tải, mô hình cân bằng tải, đánh giá các tham số ảnh hưởng đến cân bằng tải và trình bày một số phương pháp đánh giá hiệu năng cân bằng tải. Một nội dung nữa được trình bày trong chương này là các phương pháp tiếp cận giải quyết bài toán cân bằng tải, trong luận án trình bày 3 cách tiếp cận để giải quyết vấn đề: phương pháp

xấp xỉ, phương pháp lập lịch phân bổ tài nguyên và phương pháp cải tiến các tham số ảnh hưởng đến cân bằng tải. Bên cạnh đó, Chương 1 cũng trình bày một số ưu, nhược điểm của các phương pháp cân bằng tải thông qua việc xem xét một số tham số ảnh hưởng đến hiệu năng của thuật toán như: thời gian đáp ứng, thời gian xử lý, mức độ sử dụng tài nguyên,...

Chương 1 cũng đã nêu ra các vấn đề mà luận án tập trung giải quyết bao gồm: cải thiện tham số ảnh hưởng đến cân bằng tải (thời gian đáp ứng, thời gian xử lý). Để giải quyết vấn đề còn tồn tại nêu trên, Chương 2 và Chương 3 sẽ tập trung nghiên cứu các vấn đề cụ thể sau:

- Chương 2: Nghiên cứu, phát triển thuật toán cân bằng tải để cải thiện thời gian đáp ứng trên điện toán đám mây.
- Chương 3: Nghiên cứu, phát triển thuật toán cân bằng tải để cải thiện thời gian xử lý trên điện toán đám mây.

Như vậy, nhiệm vụ chính mà luận án cần giải quyết là đề xuất thuật toán cân bằng tải nhằm cải tiến thời gian đáp ứng và thời gian xử lý trên môi trường điện toán đám mây. Cách tiếp cận của luận án là theo hướng cải thiện các tham số ảnh hưởng đến cân bằng tải. Trong Chương 2 và Chương 3 sẽ đề xuất các phương pháp theo cách tiếp cận này.

CHƯƠNG 2.

PHÁT TRIỂN MỘT SỐ THUẬT TOÁN CÂN BẰNG TẢI NHẰM CẢI THIỆN THỜI GIAN ĐÁP ỨNG TRÊN ĐIỆN TOÁN ĐÁM MÂY

Chương này trình bày 02 phương pháp cải tiến thời gian đáp ứng trên điện toán đám mây cùng các kết quả mô phỏng thực nghiệm để chứng minh hiệu quả của thuật toán đề xuất. Phương pháp thứ nhất là thuật toán LBAIRT (CT4) phân phối tải kết hợp với thời gian hoàn thành dự kiến của các máy ảo. Phương pháp thứ hai là thuật toán RRTA (CT7), dùng kỹ thuật dự báo ARIMA để dự báo thời gian đáp ứng của máy ảo tiếp theo. Thực nghiệm tiến hành trên dữ liệu mô phỏng cho thấy, các thuật toán đề xuất có tỉ lệ chấp nhận yêu cầu cao hơn, cải thiện được thời gian đáp ứng so với các thuật toán Roud Robin, Throtted. Nội dung chương 2 được minh chứng bằng các công bố trong các công trình (CT4) và (CT7).

2.1. Đặt vấn đề

Thời gian đáp ứng (Response Time) là tổng thời gian cần thiết để phản hồi yêu cầu dịch vụ. Trong khuôn khổ luận án, không mất đi tính tổng quát, ta có thể bỏ qua thời gian truyền, thì thời gian đáp ứng là tổng của thời gian phục vụ và thời gian chờ. Thời gian phục vụ là thời gian cần thiết để thực hiện công việc được yêu cầu. Nghiên cứu cho rằng thời gian đáp ứng là yếu tố chính có tác động đáng kể đến hiệu suất điện toán đám mây. Để nâng cao hiệu suất phục vụ của các dịch vụ điện toán đám mây thì việc quản lý tài nguyên đối mặt với các vấn đề cơ bản bao gồm phân bổ tài nguyên, đáp ứng tài nguyên, kết nối tới tài nguyên, khám phá tài nguyên chưa sử dụng, ánh xạ các tài nguyên tương ứng, mô hình hóa tài nguyên, cung cấp tài nguyên và lập kế hoạch sử dụng các tài nguyên. Trong đó, việc lập kế hoạch cho sử dụng tài nguyên dựa trên thời gian đáp ứng của dịch vụ là rất quan trọng. Từ đó ta có thể nghiên cứu thời gian đáp ứng để đưa ra giải pháp cho việc phân bố, cân bằng tải của các tài nguyên. Đây là một trong những hướng nghiên cứu nhiều triển vọng giúp cho công nghệ đám mây ngày một hoàn thiện và tiến bộ hơn. Điện toán đám mây là môi trường tính toán phân tán bao gồm nhiều đơn vị tính toán vật lý chia sẻ tải với nhau. Việc

điều phối tải cân bằng giữa các đơn vị tính toán vật lý đã được đặt ra và các cơ chế cân bằng tải đã được đề xuất nhằm giải quyết vấn đề này. Đây là thách thức lớn đối với nhiệm vụ cân bằng tải ở các trung tâm dữ liệu. Mục 2.2 và 2.3 đề xuất hai thuật toán cân bằng tải nhằm bằng cách cải tiến thời gian đáp ứng cho các trung tâm dữ liệu.

2.2. Thuật toán LBAIRT (CT4)

2.2.1. Cơ sở lý thuyết

Tác giả trong công trình [89] đưa ra thuật toán Throttled để cân bằng tải bằng cách duy trì một bảng thông tin cấu hình của các máy ảo và trạng thái của chúng. Khi có yêu cầu cấp phát máy ảo từ trung tâm dữ liệu (Datacenter), bộ phận cân bằng tải (ThrottledVmLoadBalancer - TVLB) sẽ chọn những máy ảo tìm thấy đầu tiên trong bảng thông tin những máy ảo đang sẵn sàng. Nếu tìm được máy ảo sẵn sàng thì sẽ giao yêu cầu đó cho máy ảo. Nếu không tìm được, sẽ trả về trung tâm dữ liệu (Datacentercontroller - DCC) giá trị = -1 sau đó DCC sẽ cho yêu cầu này vào hàng đợi. Trong thuật toán Throttled, danh sách máy ảo sẵn sàng được duy trì cùng với trạng thái của mỗi máy ảo, vì vậy khi có yêu cầu đến thì bộ phận quản lý đám mây chỉ việc đối chiếu và tìm trong danh sách này ra máy ảo sẵn sàng cho việc thực hiện yêu cầu. Đây là một phương pháp cân bằng tải động sử dụng tài nguyên khá tốt nên đạt được hiệu năng tương đối tốt. Mặc dù vậy, thuật toán vẫn còn một số hạn chế. Khi có yêu cầu đến bộ phận quản lý đám mây tìm trong danh sách máy ảo đang sẵn sàng và chọn máy ảo đầu tiên khi nó tìm thấy. Cách này tương đối hợp lý trong việc đảm bảo tính nhanh gọn, nhưng nếu máy ảo đó sẵn sàng nhưng không đủ năng lực thực hiện yêu cầu thì sẽ tốn chi phí về thời gian hơn. Thuật toán đã không xem xét tới lượng tải hiện tại của máy ảo, vì vậy khi một máy ảo không đủ năng lực thực hiện yêu cầu, sẽ phải quay lại tìm một máy ảo tiếp theo, hệ thống sẽ tiêu tốn thời gian chờ và thời gian đáp ứng hơn.

Bảng 2.1. Thuật toán Throttled

Đầu vào:

Danh sách trạng thái VM

Các yêu cầu đến

Đầu ra:

VM ID thỏa mãn để xử lý yêu cầu

Hoặc không có VM ID thỏa mãn

Các bước xử lý:

Khởi tạo: tất cả máy ảo đều ở trạng thái sẵn sàng (sẵn sàng '0' hoặc không sẵn sàng '1')

1. Bộ TVLB duy trì một bảng chỉ mục bao gồm: danh sách VM và trạng thái VM (bận/sẵn sàng) tương ứng.

2. Bộ DCC nhận yêu cầu mới.

3. Bộ DCC truy vấn đến TVLB cho phân bổ kế tiếp.

4. Bộ TVLB duyệt danh sách VM từ trên xuống, tìm ra VM nào đang sẵn sàng:

Trường hợp nếu tìm thấy VM:

- TVLB gửi ID của VM về bộ DCC ⟶ Bộ DCC gửi yêu cầu tới VM ⟶ DCC thông báo tới TVLB một phân bổ mới ⟶ TVLB cập nhật bảng chỉ mục và chờ yêu cầu mới của DCC.

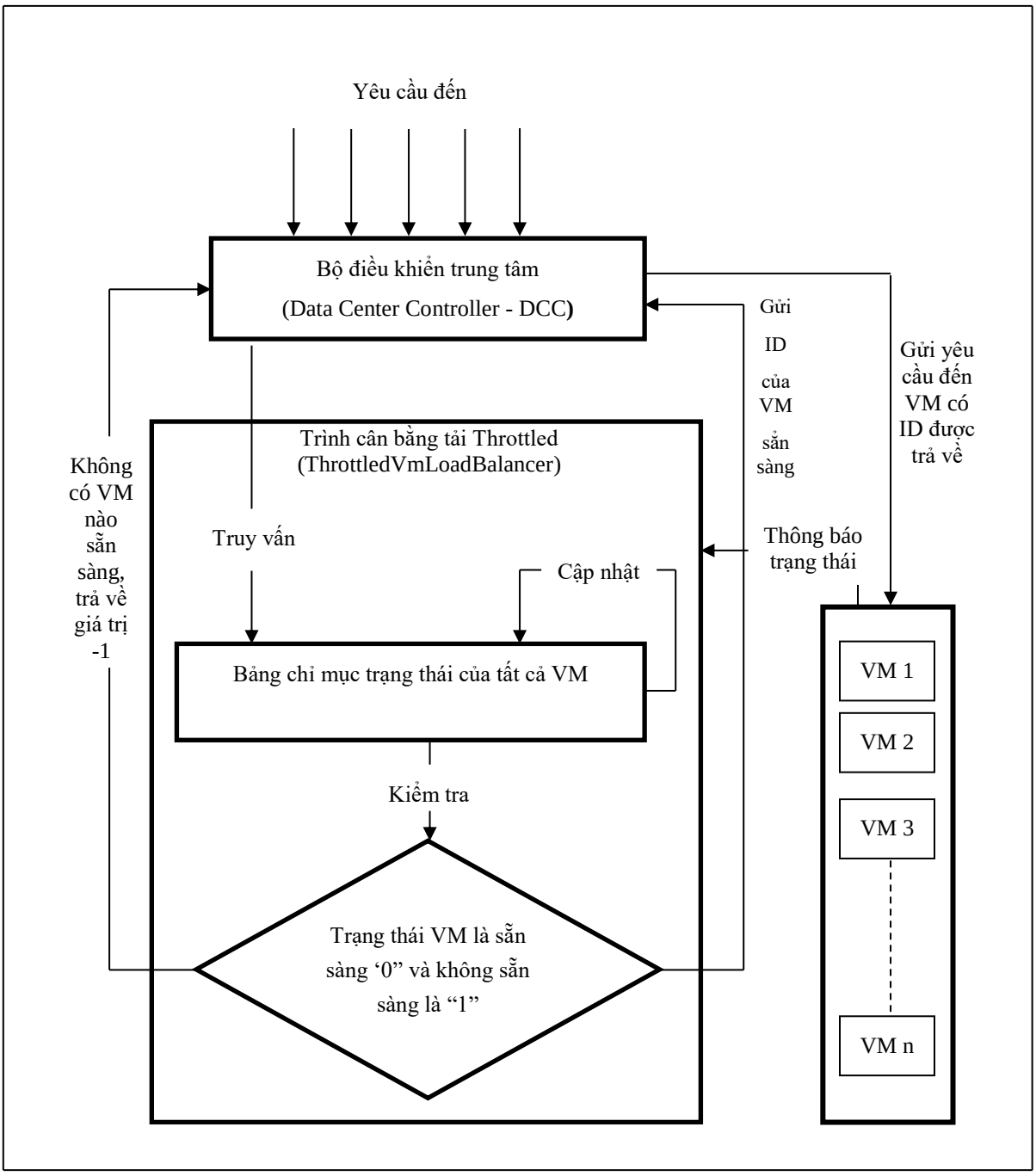
Trường hợp ngược lại, nếu không tìm thấy VM nào:

- Trình cân bằng tải TVLB sẽ trả giá trị về là -1 cho DCC ⟶ DCC sắp xếp các yêu cầu.

5. Khi VM kết thúc xử lý các yêu cầu, thì DCC nhận phản hồi, nó sẽ ghi chú lại việc cấp phát VM của TVLB.

6. DCC kiểm tra hàng đợi. Nếu còn yêu cầu, tiếp tục lặp lại bước 3.

7. Lặp lại bước 2.



Hình 2.1: Sơ đồ nguyên lý của thuật toán Throttled

Bảng 2.2 Ưu điểm và nhược điểm của thuật toán Throttled [89]

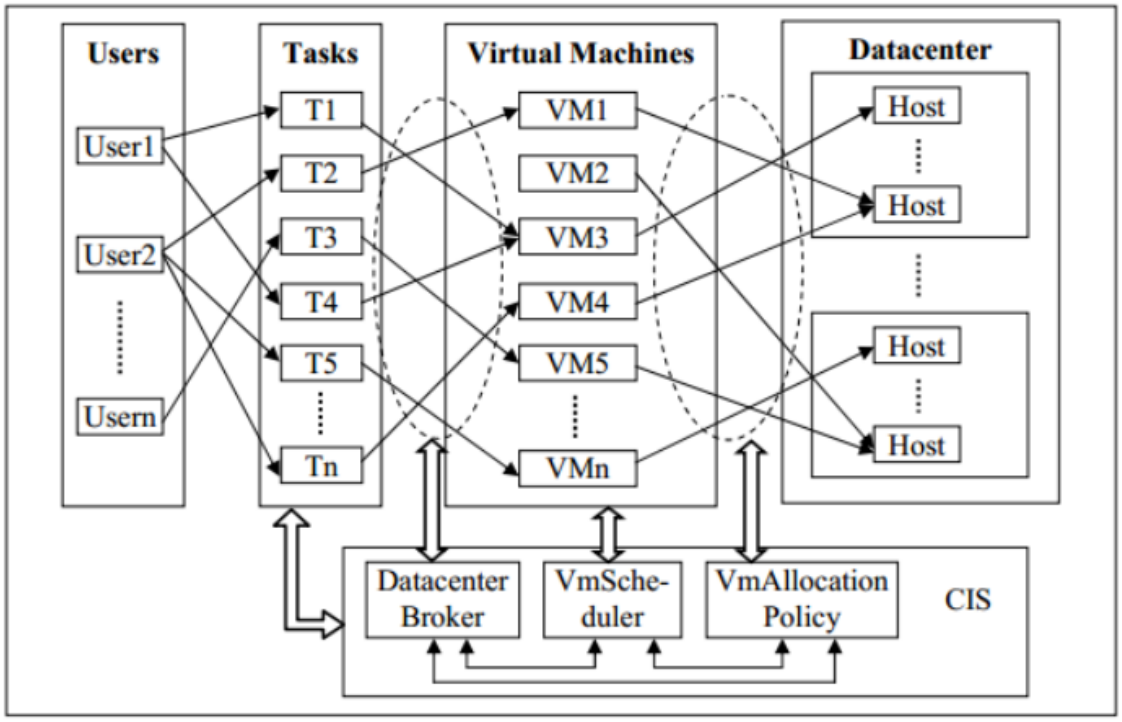
Ưu điểm	Nhược điểm
- Danh sách máy ảo được duy trì cùng với trạng thái các VM. - Hiệu suất tốt; - Sử dụng nguồn tài nguyên hiệu quả.	- Quét toàn bộ máy ảo ngay từ đầu và chọn ra VM sẵn sàng đầu tiên trong danh sách; - Không xem xét tải hiện hành của các VM.

Hạn chế của thuật toán Throttled [89]: Giải thuật này có thời gian đáp ứng nhỏ hơn giải thuật Round Robin, nhưng phải dò tìm VM đang sẵn sàng '0' với toàn bộ kích thước bảng danh sách VM ban đầu do đó làm tăng chi phí thời gian. Phương pháp này cũng chưa xem xét tới lượng tải hiện tại của từng VM, do việc chọn VM đầu tiên trong danh sách các máy ảo sẵn sàng, nên có thể gặp trường hợp một yêu cầu có kích thước lớn sẽ được gán cho máy ảo đang phục vụ một yêu cầu khác và đang sử dụng phần lớn tài nguyên của máy ảo đó, dẫn đến việc các yêu cầu bị tắc nghẽn. Thuật toán LBAIRT được đề xuất nhằm giải quyết vấn đề này.

2.2.2. Đề xuất thuật toán

Thuật toán LBAIRT được công bố trong công trình (CT4). Thuật toán cải tiến dựa trên hai tiêu chí: tải và công suất của VM để đưa ra quyết định VM nào sẽ được chọn phân bổ cho yêu cầu (Cloudlet [74]) tiếp theo. Khi DatacenterBroker phân bổ Cloudlet tới một VM có tải nhỏ, thời gian đáp ứng có thể chưa nhỏ hơn so với VM có lượng tải nhiều hơn, vì điều này còn phụ thuộc vào năng lực xử lý của mỗi VM. Tại thời điểm cụ thể, DatacenterBroker luôn biết được trạng thái hiện tại của mỗi VM, cùng các cloudlet mà nó nắm giữ. Chính vì vậy, ý tưởng của thuật toán đề xuất là tại thời điểm chuẩn bị phân bổ cloudlet tiếp theo, DatacenterBroker sẽ xem xét từng VM trên từng host vật lý, cụ thể là tính tổng chi phí thời gian xử lý các Cloudlet đang ở trong hàng đợi và chi phí thời gian xử lý Cloudlet chuẩn bị phân bổ tiếp theo

trên chính VM đang xét. Khi đó tổng chi phí thời gian xử lý trên VM nào nhỏ nhất thì chọn VM đó để phân bổ Cloudlet đến. Ở đây, công suất thực sự của VM để xử lý các cloudlet là một yêu tố quan trọng được đưa vào tính toán mà không bị bỏ qua như các phương pháp trước đây. Việc xác định công suất xử lý của VM cho Cloudlet trên hạ tầng ảo hóa đám mây là phức tạp, tùy thuộc vào cơ chế lập lịch tài nguyên tính toán trong hệ thống.



Hình 2.2: Mô hình IaaS điện toán đám mây thông qua thành phần DatacenterBrocker [50].

Điểm mới của thuật toán LBAIRT là xét thêm số tham số thời gian hoàn thành công việc dự kiến của mỗi VM khi có các danh sách yêu cầu đến. Thuật toán cân bằng tải dựa vào những thông số đầu vào như: cấu hình các máy ảo, cấu hình các yêu cầu (cloudlet), thời gian đến, thời gian hoàn thành các tác vụ, sau đó dự tính ra thời gian hoàn thành dự kiến của mỗi yêu cầu, thời gian đáp ứng dự kiến. Thời gian đáp ứng là thời gian xử lý cộng thêm chi phí thời gian truyền tải yêu cầu, xếp hàng đợi qua các nút mạng.

- Thời gian đáp ứng dự kiến được tính theo công thức sau [74]:

$$TR_{dk} = F_t - A_t + T_{delay} \quad (2.1)$$

Trong đó:

- TR_{dk}: Thời gian đáp ứng dự kiến.

- F_t: là thời điểm hoàn thành dự kiến xử lý Cloudlet.

- A_t: là thời điểm đến của Cloudlet.

- T_{delay}: là thời gian truyền tải các yêu cầu. Vì thuật toán thực hiện công việc điều phối tải của DatacenterBroker nên mức độ của thuật toán chỉ ảnh hưởng đến thời gian xử lý trong một môi trường mạng nội bộ của một Datacenter. Do đó tham số về độ trễ truyền có thể bỏ qua, nên $T_{delay} = 0$.

Trong khuôn khổ luận án, chỉ nghiên cứu chính sách lập lịch của thuật toán là Spaceshared-Timeshared. Chính sách Space-shared được áp dụng để phân bổ máy ảo tới các VM và chính sách Time-shared hình thành cơ sở để phân bổ các yêu cầu đến core xử lý bên trong một VM. Do đó, trong suốt thời gian sống của VM, tất cả các yêu cầu được phân bổ theo phương pháp động. Cơ sở tính toán cho giải thuật đề xuất sẽ căn cứ vào công thức (2.2) và (2.3). Việc xác định các công thức (2.2) và (2.3) nhằm mục đích tìm ra thời gian đáp ứng dự kiến ở công thức (2.1), khi tìm ra được giá trị thời gian đáp ứng dự kiến của máy ảo thì thuật toán sẽ phân bổ các yêu cầu đầu vào một cách hiệu quả dựa vào giá trị này, điều này đồng nghĩa là nâng cao được hiệu năng của thuật toán cân bằng tải.

- Xác định Ft [74]: Do sử dụng chính sách Timeshared nên F_t của yêu cầu *p* được quản lý bởi VM_i được tính như sau:

$$F_t = ct + \frac{rl(p)}{capacity \times cores(p)} \quad (2.2)$$

Với:

$$capacity = \frac{\sum_{i=1}^{np} cap(i)}{max\left(\sum_{j=1}^{cloudlets} cores(j), np\right)} \quad (2.3)$$

Thời gian thực thi của một Cloudlet được xác định theo công thức sau:

$$Thời\ gian\ thực\ thi\ một\ cloudlet = \frac{rl}{capacity \times cores(p)} \quad (2.4)$$

Trong đó:

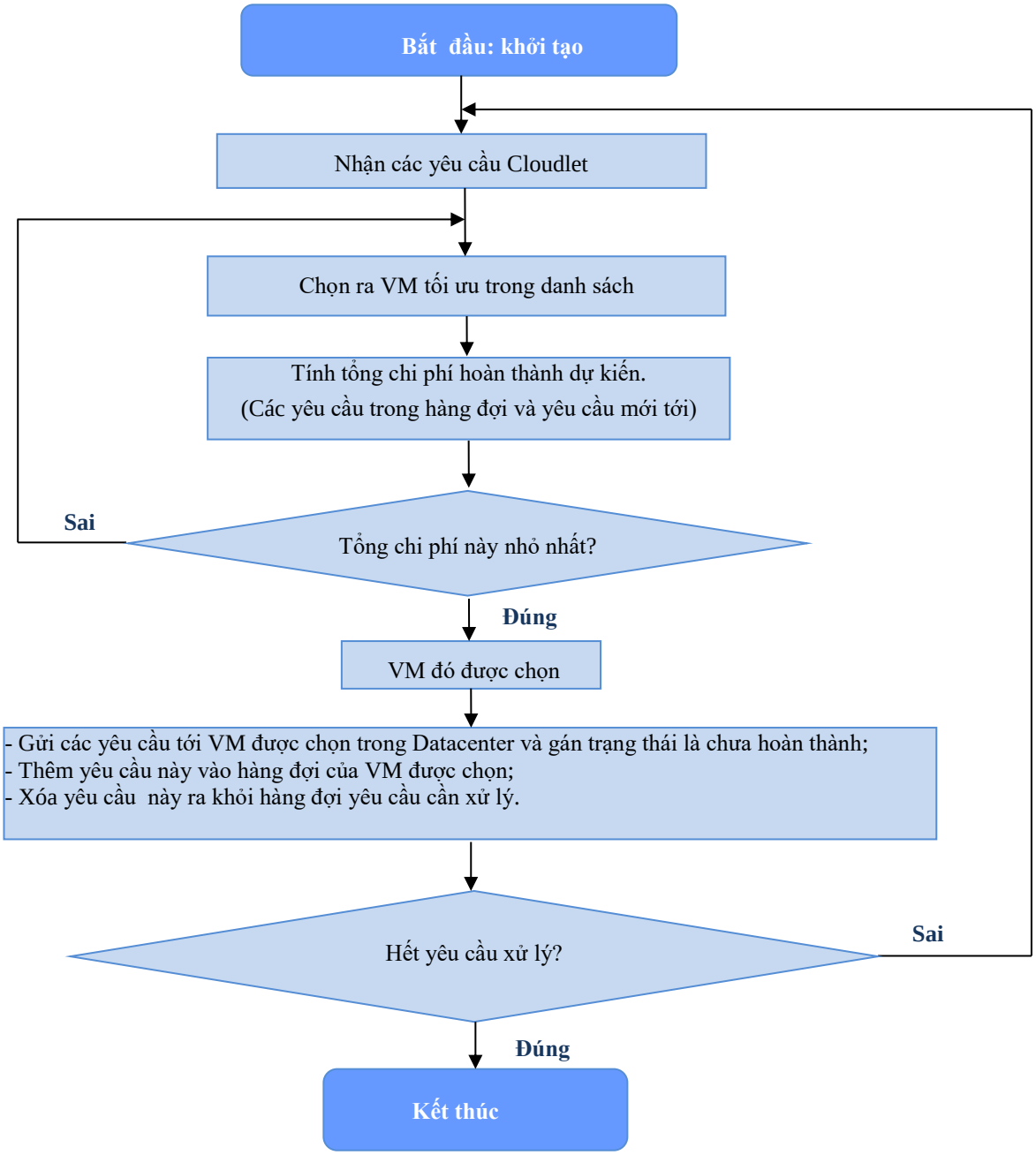
- *rl*: là tổng số lệnh mà Cloudlet *p* cần được thực thi trên một bộ xử lý.
- *capacity*: là năng lực xử lý trung bình (tính theo MIPS) của một core dành cho Cloudlet *p*.
- *ct*: là thời gian mô phỏng hiện tại.
- *cores(p)*: là số lượng core mà Cloudlet p cần.
- *np*: là số lượng core thực mà host đang xét có.
- *cap*: là năng lực xử lý của core.

Tham số Capacity xác định công suất thực sự dành cho xử lý yêu cầu trên mỗi VM. Rõ ràng Capacity tùy thuộc vào chính sách lập lịch tài nguyên tính toán trên hệ thống ảo hóa. Tổng năng lực xử lý trên một host vật lý là không thay đổi và tùy theo số core vật lý và năng lực xử lý của từng core. Tuy nhiên, khi tài nguyên xử lý này được đem chia sẻ cho nhiều yêu cầu đồng thời, mỗi yêu cầu đầu vào cần một số core nhất định và nếu tổng số core đó lớn hơn số core vật lý thì khái niệm core ảo xuất hiện, mỗi core ảo sẽ có năng lực xử lý thấp hơn core vật lý. Nói cách khác, công suất một core ảo dành cho yêu cầu chỉ có thể bằng hoặc nhỏ hơn core vật lý và như thế nào là tùy vào chính sách chia sẻ tài nguyên. Capacity chính là công suất xử lý của một core ảo. Từ phân tích này và căn cứ vào chính sách chia sẻ tài nguyên để phát triển công thức tính cho Capacity. Chính sách chia sẻ tài nguyên được cụ thể hóa qua cơ chế lập lịch trong điện toán đám mây. Ta có hai mức lập lịch: lập lịch VM để chia sẻ tài nguyên host vật lý và lập lịch tác vụ để chia sẻ tài nguyên VM. Có hai cơ chế lập lịch là Timeshared và Spaceshared. Thuật toán này sẽ thực hiện và mô phỏng dựa trên chính sách Spaceshared đối với máy ảo và Timeshared đối với yêu cầu.

Thiết kế thuật toán LBAIRT:

- **Bước 1**: Khởi tạo DatacenterBroker. Bảng chỉ mục trạng thái của máy ảo và trạng thái các Cloudlet hiện tại. Lúc khởi tạo chưa có máy ảo nào được phân bổ Cloudlet.
- **Bước 2**: Khi có một yêu cầu đến, DatacenterBroker phân tích bảng chỉ mục trạng thái. Sau đó sẽ tính tổng chi phí thời gian hoàn thành xử lý tất cả Cloudlet hiện có trong hàng đợi của mỗi máy ảo cộng với thời gian dự kiến hoàn thành của Cloudlet mới đến. Nếu máy ảo nào có thời gian xử lý dự kiến nhỏ nhất thì máy đó được chọn để đệ trình Cloudlet tiếp theo.
- **Bước 3**: Gửi ID của VM được chọn đến DatacenterBroker sau đó DatacenterBroker gửi Cloudlet tới VM đó.
- **Bước 4**: Databroker thông báo về việc phân bổ mới và cập nhật vào bảng trạng thái máy ảo và Cloudlet.
- **Bước 5**: Khi máy ảo hoàn thành yêu cầu xử lý và DatacenterBroker nhận đáp ứng Cloudlet nó sẽ cập nhật vào bảng trạng thái của Cloudlet là đã hoàn thành và giảm đi 1 Cloudlet trong bảng trạng thái.
- **Bước 6**: Quay lại bước 2.

Điểm mới của thuật toán LBAIRT: Đưa vào thời gian hoàn thành dự kiến của mỗi VM cho các cloudlet trong hàng đợi. Dựa trên tham số này, thuật toán sẽ chọn VM với thời gian hoàn thành dự kiến nhỏ nhất và tỷ lệ sử dụng thấp nhất để phân bổ cloudlet. Lượng tải hiện tại trên các VM cũng được xét đến, do đó, việc tìm ra các VM sẵn sàng đồng thời đủ năng lực xử lý các yêu cầu đến đã được tính toán cụ thể hơn so với thuật toán Throttled.



Hình 2.3: Lưu đồ thuật toán đề xuất LBAIRT.

Xác định độ phức tạp tính toán của thuật toán LBAIRT:

Độ phức tạp thời gian (Time complexity) là độ phức tạp tính toán mô tả lượng thời gian cần để chạy một thuật toán. Giả sử số yêu cầu mới là n, số máy ảo là m, kích thước hàng đợi của mỗi máy ảo là q, kích thước của hàng đợi cần xử lý là p.

Vì cứ mỗi yêu cầu đến, thuật toán thực hiện:

- Tính tổng chi phí hoàn thành xử lý tất cả các yêu cầu hiện có trong tất cả các hàng đợi của tất cả các máy ảo. Giả thiết thời gian tính toán cho một yêu cầu là 1 đơn vị, thì độ phức tạp thời gian là: O(mq).
- Tìm máy ảo có tổng thời gian xử lý thấp nhất. Dùng vòng for thì độ phức tạp thời gian là O(m).
- Gửi yêu cầu tới máy ảo được chọn với độ phức tạp thời gian O(1)
- Thêm yêu cầu vào hàng đợi với độ phức tạp thời gian O(1)
- Xóa yêu cầu này ra khỏi hàng đợi cần xử lý với độ phức tạp thời gian O(1).

Như vậy độ phức tạp thời gian cho mỗi yêu cầu mới là: O(mq+m +1+1+1)=O(mq).
Vì có n yêu cầu mới cộng với kích thước hàng đợi cần xử lý là p, nên tổng số yêu cầu cần xử lý thực tế là: O(n+p).
Như vậy, độ phức tạp tính toán của thuật toán là: **O((n+p) mq)**.

2.2.3. Kết quả mô phỏng

Mục tiêu của mô phỏng này là so sánh, phân tích, đánh giá thời gian đáp ứng và thời gian thực hiện của thuật toán Throttled [89] và thuật toán đề xuất LBAIRT. Mô phỏng sử dụng bộ công cụ CloudSim bao gồm 1 trung tâm dữ liệu. Hai thuật toán Throttled và LBAIRT được viết bằng ngôn ngữ Java trên bộ công cụ mô phỏng CloudSim và sử dụng một bộ dữ liệu để so sánh thời gian đáp ứng của đám mây. Giá trị tham số hệ thống được cho trong các bảng Bảng 2.3, Bảng 2.4, Bảng 2.5. Kịch bản mô phỏng sẽ được tiến hành theo chính sách lập lịch cho máy ảo và tác vụ là SpaceShared – TimeShared.

Bảng 2.3: Giá trị các tham số trong thiết lập đám mây

Loại	**Tham số**	**Giá trị**
Datacenter	Số lượng Datacenter	1
	Số lượng Host	3

Host	Số lượng PE trên mỗi Host	1-4
	MIPS của PE	1000-30000 MIPS
	Bộ nhớ Host	5120-10240-12288
	Dung lượng lưu trữ	1024000-1044480 MB (1000-1020 GB)
	Băng thông	10000 MB
Máy ảo	Tổng số máy ảo	3
	Bộ nhớ máy ảo (Ram)	1024-3072
	Băng thông	1024 MB
Cloudlet/tác vụ	Tổng số Cloudlet	10-60
	Chiều dài của Cloudlet	1024-20480
	Số PE yêu cầu	1-3

Bảng 2.4: Cấu hình các VM

ID	**Bộ nhớ (Mb)**	**Băng thông(Mb)**	**Số PE/core**	**Tốc độ PE (MIPS)**
0	4069	1024	2	200000
1	2048	1024	1	100000
2	1024	1024	2	50000

Bảng 2.5: Thiết lập tham số các Cloudlet

ID Cloudlets	Chiều dài Cloudlet	Số PE yêu cầu
0	2000	1
1	3000	2
2	4000	1
3	3000	2
4	2000	2
5	2000	1
6	1000	2
7	3000	1
8	5000	2
9	2000	1
10	2000	1
11	3000	2
12	4000	1
13	3000	2
14	2000	2
15	2000	1
16	1000	2
17	3000	1

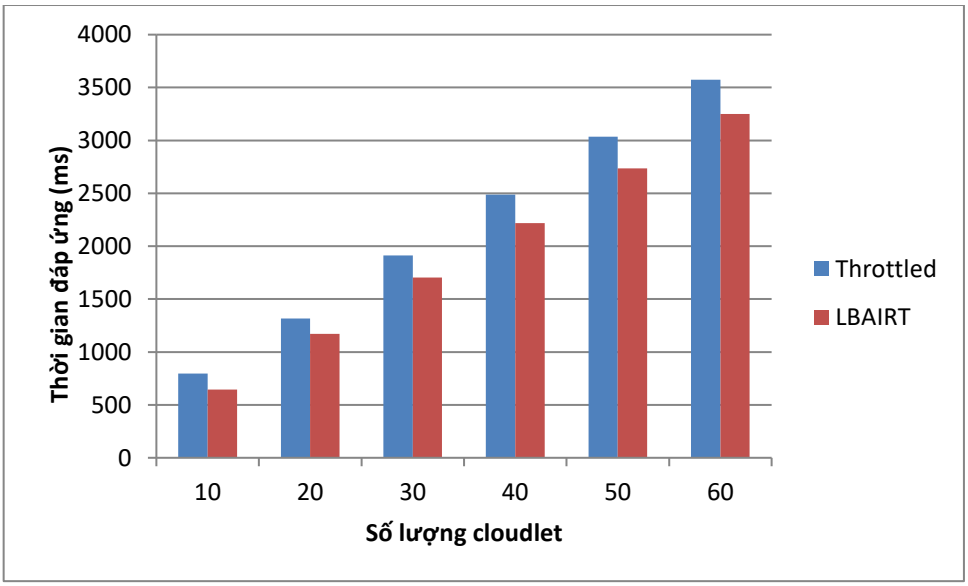
18	5000	2
19	2000	1
20	2000	1
21	3000	2
22	4000	1
23	3000	2
24	2000	2
25	2000	1
26	1000	2
27	3000	1
28	5000	2
29	2000	1

Bảng 2.6 và Hình 2.4 thể hiện kết quả mô phỏng cho các trường hợp tương ứng 10, 20, 30, 40, 50, 60 cloudlet và so sánh thời gian đáp ứng của thuật toán Throtted và thuật toán đề xuất LBAIRT. Kết quả cho thấy thời gian đáp ứng trong thuật toán LBAIRT đã được cải tiến so với thuật toán Throttled.

Bảng 2.6: So sánh kết quả mô phỏng giữa hai thuật toán Throttled và LBAIRT

Số Cloudlet	Thời gian đáp ứng trung bình (ms)	
	Throttled	LBAIRT
10	797.50	644.25
20	1315.65	1171.50
30	1912.75	1704.91
40	2488.37	2217.75
50	3036.65	2734.15
60	3575.08	3250.50

Hình 2.4 cho thấy, khi thay đổi số lượng Cloudlet đầu vào thì thuật toán đề xuất LBAIRT cũng cho thời gian đáp ứng nhỏ hơn so với thuật toán Throttled. Điều đó chứng tỏ thuật toán đề xuất đã cải thiện được thời gian đáp ứng cho đám mây.



Hình 2.4. Thực nghiệm mô phỏng so sánh thời gian đáp ứng của Throttled và LBAIRT khi thay đổi số lượng cloudlet.

Thuật toán LBAIRT được đề xuất dựa trên thuật toán Throttled. Trong thuật toán Throttled, không xét đến lượng tải trên VM. Trong thuật toán đề xuất, ngoài việc xét thời gian hoàn thành dự kiến còn xét đến năng lực xử lý yêu cầu của VM. Trong môi trường đám mây, việc phân phối tải giữa các máy ảo không đồng nhất về khả năng xử lý, do đó mỗi máy ảo có thể có chi phí thời gian xử lý khác nhau. Để cân bằng tải hiệu quả, về nguyên lý là chọn máy ảo nào tốn ít thời gian xử lý nhất để phân công nhiệm vụ. Thuật toán đề xuất đã được thử nghiệm trong môi trường điện toán đám mây CloudSim và được sử dụng trong ngôn ngữ lập trình Java, sử dụng cùng một chế độ lập lịch là Spaceshared - Timeshared với các VM và tác vụ.

2.3. Thuật toán RRTA

2.3.1. Đề xuất thuật toán

Thuật toán RRTA được công bố trong công trình (CT7). Ý tưởng của thuật toán cân bằng tải là giảm thời gian đáp ứng trên điện toán đám mây dựa vào thuật toán dự báo ARIMA [75] để dự báo thời gian đáp ứng, giúp phân bổ hiệu quả các yêu cầu đầu vào. Thuật toán ARIMA được mô tả ở Phụ lục 1

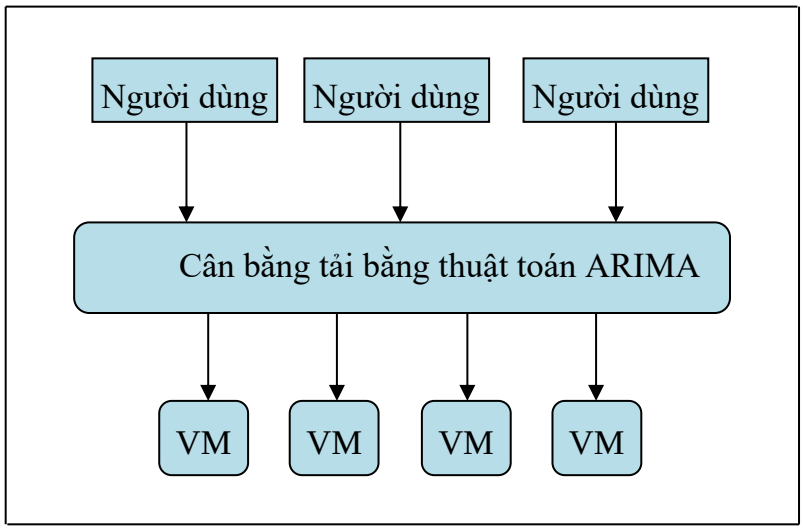
- **Mục tiêu:**
  - Giảm thiểu các quá trình truyền thông tin giữa bộ cân bằng tải và các máy ảo có tài nguyên đang rảnh rỗi.
  - Giảm thời gian đáp ứng các yêu cầu từ phía người dùng
  - Hạn chế tối đa sự mất cân bằng tải giữa các máy ảo, ngăn chặn và cảnh báo trước khi mất cân bằng tải.
  - Dự đoán thời gian đáp ứng tiếp theo từ bất kỳ máy chủ nào cho các yêu cầu đang được xử lý.
- **Giả định:**
  - Bộ cân bằng tải biết trước các dịch vụ nào đang chạy trên các VM vào bất cứ thời điểm nào. Ở đây tập trung vào dịch vụ Web (Web Service), các máy chủ web sẽ biết trước thời gian đáp ứng của từng dịch vụ chạy trên web và trên từng VM.
- **Mô hình nghiên cứu:**

- Bộ cân bằng tải có danh sách các VM và các dịch vụ mà đám mây cung cấp. Bộ cân bằng tải biết trước các dịch vụ chạy trên VM nào và có thể phân bổ dịch vụ mới trên một VM mới theo yêu cầu.
- Mô hình này sử dụng tham số là ngưỡng thời gian, dựa vào ngưỡng này để biết trước được thời gian đáp ứng tiếp theo. Ví dụ: Giả sử VM3 được chọn để xử lý yêu cầu trước đó, yêu cầu mới kế tiếp sẽ được phân bổ vào VM4, và lặp đi lặp lại một cách tự nhiên. VM được chọn có thời gian đáp ứng dự đoán nhỏ hơn ngưỡng được tính toán từ thuật toán ARIMA và việc đối xử với các VM là như nhau trong việc phân bổ các yêu cầu. Nếu không có VM nào trong pool (tập một số VM) đang xét thỏa điều kiện ngưỡng, thì sẽ phân bổ yêu cầu tới các pool VM kế tiếp. VM có thời gian đáp ứng trung bình và dự đoán thấp nhất sẽ được chọn để xử lý yêu cầu tiếp theo.
- Nếu không có VM nào thỏa điều kiện ngưỡng thì xử lý như sau:

  + Nếu có VM (hoặc pool) không tải, có thể sử dụng các máy này để xử lý các yêu cầu, đảm bảo điều kiện thỏa mãn ngưỡng.

  + Nếu không có VM (hoặc pool) không tải, hoặc tất cả đều không thỏa mãn ngưỡng thì phân bổ dịch vụ đang chạy đó tới VM có thời gian đáp ứng dự đoán gần với ngưỡng nhất.

- Thuật toán lên kế hoạch cho các yêu cầu tiếp theo nhằm không bị mất cân bằng tải. Thuật toán sẽ giảm được các liên lạc không cần thiết giữa VM và các nguồn tài nguyên hiện có, tăng băng thông và thông lượng phục vụ cho yêu cầu người dùng.



Hình 2.5. Mô hình thuật toán RRTA

- **Nguyên lý hoạt động của thuật toán:**

Thuật toán hoạt động căn cứ vào dữ liệu đã có về chuỗi thời gian của thời gian đáp ứng, sử dụng thuật toán ARIMA (Phụ lục 1) để dự báo thời gian đáp ứng, từ đó phân bố tài nguyên cho các yêu cầu tiếp theo. Kế thừa công trình đã công bố trong tài liệu [2], thuật toán đề xuất gồm 3 nhóm module chính như sau:

(1) Module tính toán ngưỡng bằng thuật toán ARIMA cho đám mây*:*

Trong thuật toán RRTA sử dụng một tham số là ngưỡng thời gian, được hiểu là ngưỡng thời gian để một máy ảo nhận được yêu cầu mới nếu thời gian đáp ứng trung bình và thời gian đáp ứng trung bình được dự đoán của máy ảo nhỏ hơn thời gian ngưỡng. Ngưỡng là thời gian đáp ứng dự báo của đám mây, được thuật toán ARIMA tính toán và tăng hoặc giảm tùy thuộc thời gian đáp ứng theo dữ liệu chuỗi thời gian. Ngưỡng mới chính là thời gian đáp ứng dự đoán xét trong tập các VM đang xét (trong vòng 100 yêu cầu gần nhất):

$$Ngưỡng\ mới = T_{New} = ARIMA(RT_1, RT_2, \ldots, RT_{100})$$

Trong đó RT_i = là chuỗi thời gian đáp ứng ghi lại được của đám mây (chỉ xét trong vòng 100 Request gần nhất)

(2) Module dự báo thời gian đáp ứng tiếp theo cho từng VM:

Module này sử dụng thuật toán ARIMA để dự báo thời gian đáp ứng tiếp theo của các VM. Việc dự báo thời gian đáp ứng tiếp theo dựa vào dữ liệu chuỗi thời gian đáp ứng trong 50 request gần nhất của VM đang xét thông qua hàm *getPredictedRT()* (Phụ lục 2). Đồng thời cung cấp hàm tính toán giá trị dự báo gần nhất của các VM so với ngưỡng đưa vào thông qua hàm *AllocateRequestToVM(VM, Request)* (Phụ lục 2).

PRT_i = Prediected Response Time = Thời gian đáp ứng dự đoán của VM_i

(3) Module phân bổ các dịch vụ (chọn VM)

Module này có nhiệm vụ phân bổ các yêu cầu đến các VM đạt được điều kiện ngưỡng thời gian. Nếu một yêu cầu được gửi tới VM đang xét và VM này không tải, thì yêu cầu này được chuyển tới trực tiếp VM đó, và lấy được giá trị thời gian đáp ứng. Nếu thời gian đáp ứng dự báo của VM đang xét (được tính toán từ Module 2) nhỏ hơn thời gian đáp ứng tiếp theo của đám mây (tính toán từ module 1) thì yêu cầu này sẽ được xử lý trên VM này. Ngược lại, không có VM nào thỏa điều kiện ngưỡng (thời gian đáp ứng dự báo của VM không nhỏ hơn thời gian đáp ứng dự báo của đám mây) thì yêu cầu sẽ được phân bổ vào VM có dự báo gần với ngưỡng nhất.

Khởi tạo ngưỡng: ban đầu chưa có dữ liệu thời gian, lấy ngưỡng bằng thời gian đáp ứng của yêu cầu đầu tiên.

Khởi tạo ngưỡng*: $T_{initial} = RT_1$*

Các bước của thuật toán:

1. For each Request in CloudRequests
2. T_{new} = ARIMA(RT_i); *// Module 1*
3. isLocated = false;
4. For each VM in VMList
5. If VM.getPredictedRT() < T_{new}
6. AllocateRequestToVM(VM, Request); *// Module 3*
7. isLocated = true;

8. End If
9. End For
10. If (!isLocated)
11. VM = VMList.getMinDistance(T_{new}); *// Module 2*
12. AllocateRequestToVM(VM, Request);
13. End If
14. End For

Theo tài liệu [2], ngưỡng được tính toán chính là thời gian đáp ứng lớn nhất xét trong tập các VM. Vì thế, thuật toán RRTA này sử dụng lại phương pháp chọn ngưỡng này, tuy nhiên sẽ hiệu chỉnh một số thay đổi, hoặc đưa vào các hệ số và tham số, tùy thuộc vào kết quả thực nghiệm.

Xác định độ phức tạp tính toán:

Giả sử số yêu cầu là n và số máy ảo là m. Giả thiết độ phức tạp tính toán là một hàm của n và m.

Đối với mỗi yêu cầu, thuật toán thực hiện:

- Tính toán Tnew dùng thuật toán ARIMA. Độ phức tạp thời gian là O(1).
- Tìm kiếm VM trong VMList sao cho VMgetPredictedRT()<Tnew. Dùng vòng for với độ phức tạp thời gian là O(m). Phân bố yêu cầu cho VM đó nếu tìm được O(1).
- Nếu không tìm được VM thỏa mãn điều kiện trên thì gán VM=VMList.getMinDistance(Tnew) và phân bố yêu cầu cho VM này. Độ phức tạp thời gian O(1).

  Như vậy độ phức tạp thời gian cho mỗi yêu là: O(1+m+1+1)=O(m)

Có tất cả n yêu cầu, vì vậy độ phức tạp tính toán là: **O(nm).**

2.3.2. Thực nghiệm mô phỏng:

Phần này trình bày cách cài đặt mô phỏng thuật toán đề xuất dựa vào thời gian đáp ứng trên điện toán đám mây. Từ kết quả mô phỏng cho thấy, phương pháp sử dụng thuật toán dự báo ARIMA để dự báo thời gian đáp ứng, đã đưa ra quyết định phân phối tài nguyên hiệu quả hơn.

Giả lập môi trường đám mây sử dụng bộ thư viện CloudSim, lập trình trên ngôn ngữ JAVA; Môi trường giả lập đám mây là từ 3 đến 10 VM, và tạo môi trường yêu cầu ngẫu nhiên tới các dịch vụ trên đám mây. Bao gồm dịch vụ cung cấp VM, dịch vụ cung cấp và đáp ứng người dùng của CloudSim để thử nghiệm.

Cài đặt thuật toán RRTA và thuật toán của công trình [2] trên môi trường mô phỏng, thử nghiệm mô phỏng và so sánh kết quả giữa 2 thuật toán.

- **Các tham số mô phỏng:**

Cài đặt thuật toán đề xuất trên ngôn ngữ JAVA, sử dụng NETBEAN IDE để chạy thử và hiển thị kết quả bằng STS IDE với framework SPRING BOOT. Môi trường giả lập với bộ thư viện mã nguồn mở CloudSim 4.0 (được cung cấp bởi http://www.cloudbus.org/).

Môi trường mô phỏng giả lập gồm các thông số: 01 Datacenter với thông số như sau:

Bảng 2.7. Thông số cấu hình Datacenter

Thông tin Datacenter	Thông tin Host trong Datacenter
- Số lượng máy (host) trong datacenter: 5 - Không sử dụng Storage (các ổ SAN) - Kiến trúc (arch): x86 - Hệ điều hành (OS): Linux - Xử lý (VMM): Xen - TimeZone: +7 GMT - Cost: 3.0 - Cost per Memory: 0.05 - Cost per Storage: 0.1	Mỗi host trong Datacenter có cấu hình như sau: - CPU có 4 nhân, mỗi nhân có tốc độ xử lý là 1000 (mips) - Ram: 16384 (MB) - Storage: 1000000 - Bandwidth: 10000

- Cost per Bandwidth: 0.1	

- Các VM có cấu hình giống nhau khi được khởi tạo:

Bảng 2.8: Cấu hình VM

Kích thước (size)	RAM	MIPS	Bandwidth	Số lượng CPU (pes no.)	VMM
10000 MB	512 MB	250	1000	1	Xen

- Các yêu cầu được đại diện bởi Cloudlet trong CloudSim và kích thước của các Cloudlet được khởi tạo một các ngẫu nhiên bằng hàm random của JAVA. Số lượng Cloudlet lần lượt là 100 → 1000.

Bảng 2.9: Thông số cấu hình của Request

Chiều dài (Length)	Kích thước file (File Size)	Kích thước file xuất ra (Output Size)	Số CPU xử lý (PEs)
3000 ~ 1700	5000 ~ 45000	450 ~ 750	1

- Thuật toán đề xuất được xây dựng bằng cách tạo ra lớp *ArimaDatacenterBroker*, kế thừa từ đối tượng *DatacenterBroker*, cập nhật thêm một số phương thức và thuộc tính liên quan tới *PredictedResponseTime*, và điều chỉnh các hàm dựng sẵn để phù hợp với thuật toán đề xuất (trình bày ở Phụ lục 2):
    - *processResourceCharacteristics(SimEvent ev)*
    - *createVmsInDatacenter(int datacenterId)*

- *processVmCreate(SimEvent ev)*
- *processCloudletReturn(SimEvent ev)*

- **Tiêu chí đánh giá:**

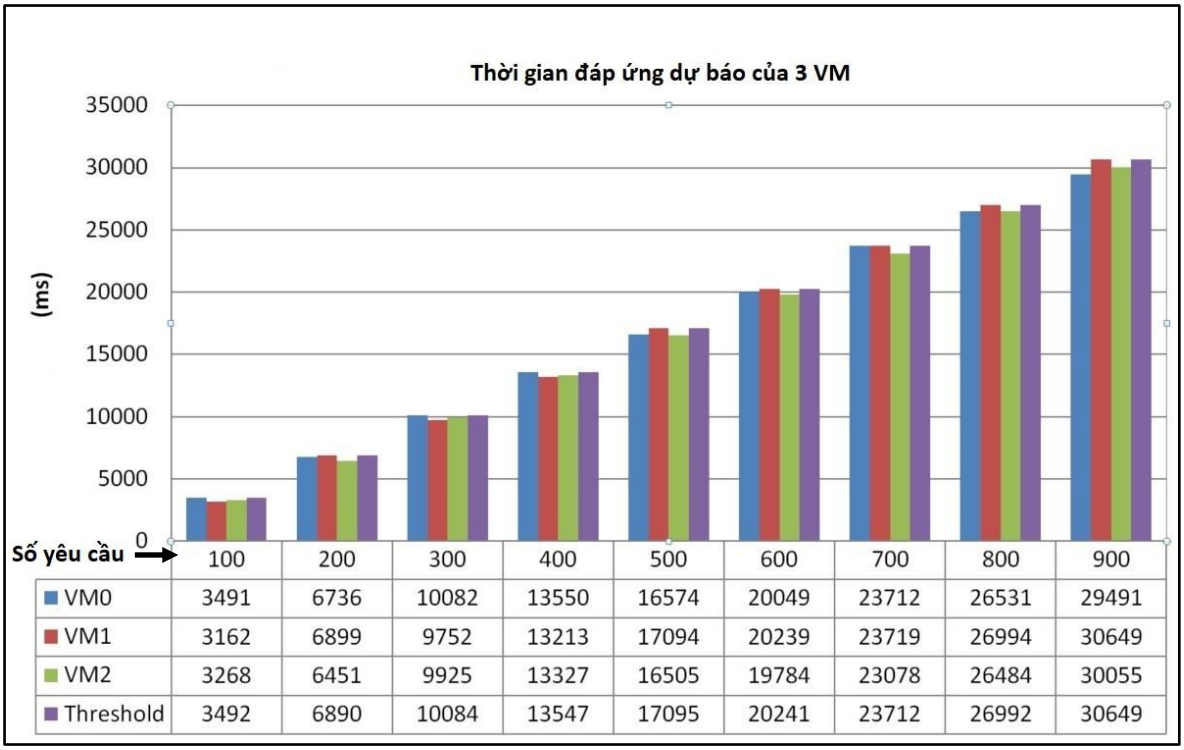
Thực nghiệm mô phỏng đám mây với các tham số như trên, chạy thuật toán cân bằng tải của CloudSim có sẵn, và thuật toán RRTA đề xuất, cùng đầu vào, so sánh kết quả đầu ra, đặc biệt là thông số thời gian đáp ứng. Thời gian đáp ứng dự báo của các VM cũng như thời gian đáp ứng dự đoán của đám mây với sai số càng thấp thì hiệu quả của thuật toán càng tốt.

- **Kết quả mô phỏng**

Kết quả chạy thực nghiệm mô phỏng trên CloudSim với 3 VM được dựng sẵn để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng yêu cầu lần lượt là 100, 200,… đến 900:

Bảng 2.10. Kết quả thực nghiệm mô phỏng với 3 VM

Số yêu cầu	VM0	VM1	VM2	Threshold
100	3491	3162	3268	3492
200	6736	6899	6451	6890
300	10082	9752	9925	10084
400	13550	13213	13327	13547
500	16574	17094	16505	17095
600	20049	20239	19784	20241
700	23712	23719	23078	23712
800	26531	26994	26484	26992
900	29491	30649	30055	30649



Số yêu cầu →	100	200	300	400	500	600	700	800	900
VM0	3491	6736	10082	13550	16574	20049	23712	26531	29491
VM1	3162	6899	9752	13213	17094	20239	23719	26994	30649
VM2	3268	6451	9925	13327	16505	19784	23078	26484	30055
Threshold	3492	6890	10084	13547	17095	20241	23712	26992	30649

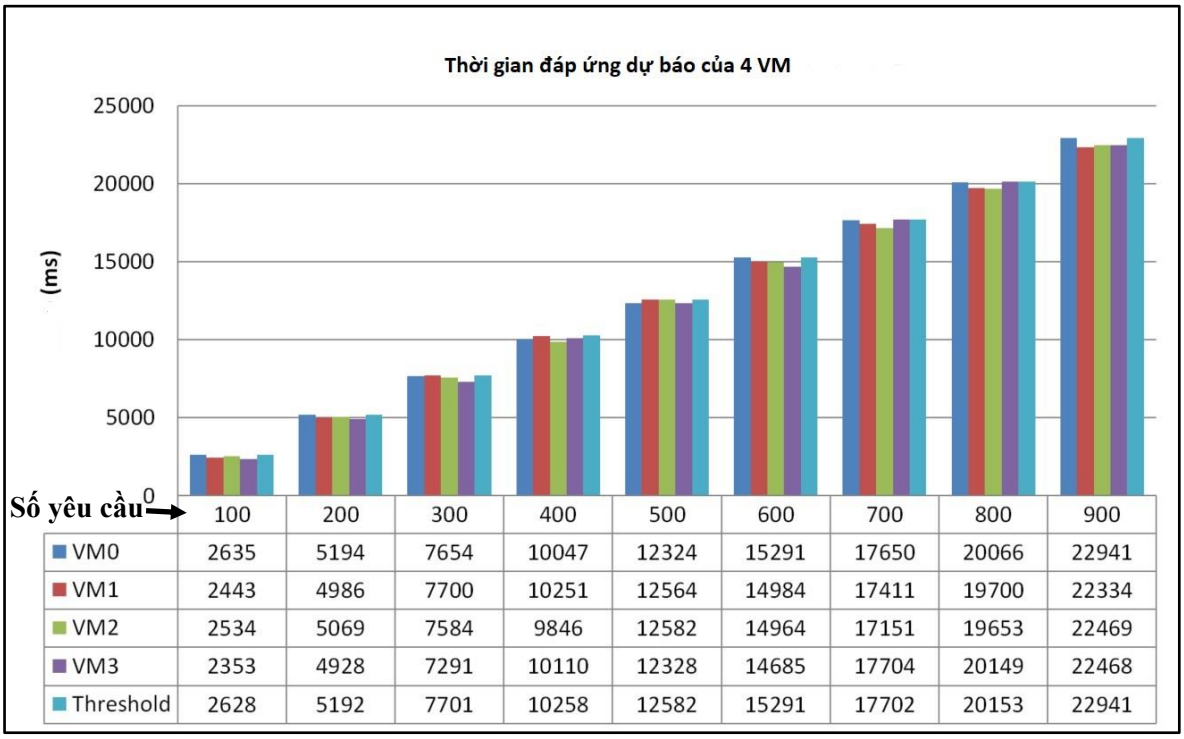
Hình 2.6. So sánh thời gian đáp ứng dự báo của 3 máy ảo và ngưỡng

Kết quả chạy thực nghiệm mô phỏng trên CloudSim với 4 máy ảo được dựng sẵn để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng Request lần lượt là 100, 200,… đến 900:

Bảng 2.11. Kết quả thực nghiệm mô phỏng với 4 VM

Số lần request	**VM0**	**VM1**	**VM2**	**VM3**	**Threshold**
100	2635	2443	2534	2353	2628
200	5194	4986	5069	4928	5192
300	7654	7700	7584	7291	7701
400	10047	10251	9846	10110	10258
500	12324	12564	12582	12328	12582
600	15291	14984	14964	14685	15291

700	17650	17411	17151	17704	17702
800	20066	19700	19653	20149	20153
900	22941	22334	22469	22468	22941



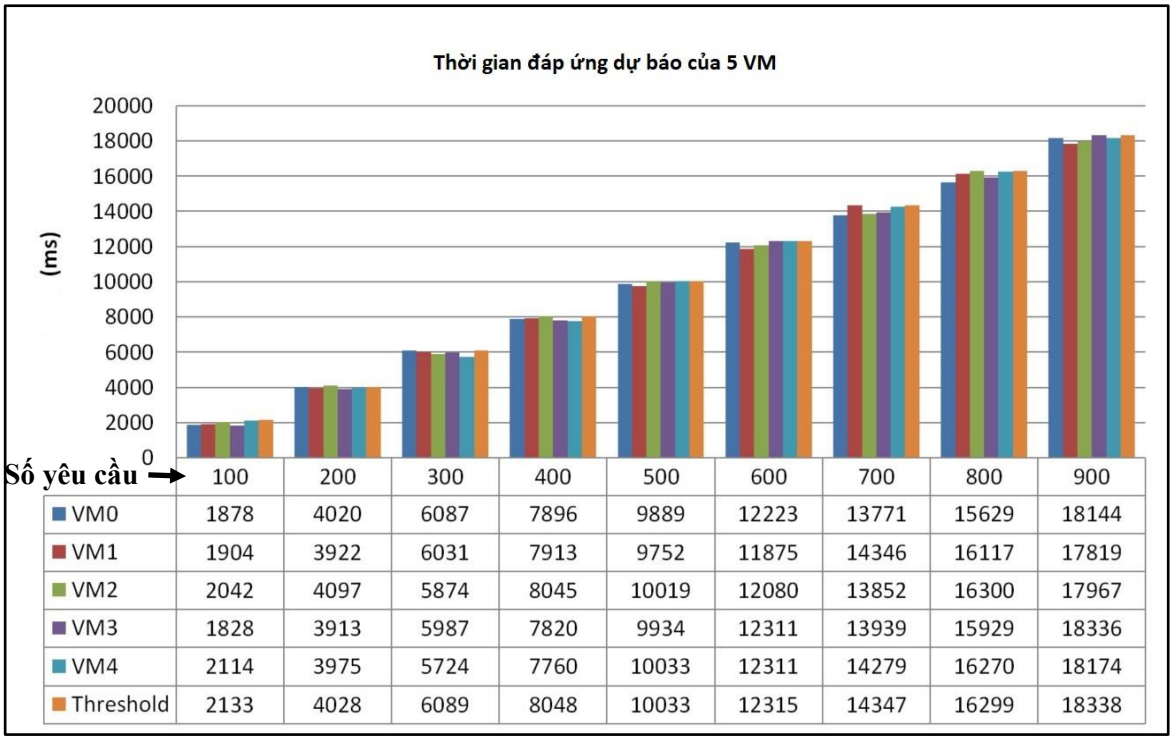
Hình 2.7. So sánh thời gian đáp ứng dự báo của 4 VM và ngưỡng

Kết quả chạy thực nghiệm mô phỏng trên CloudSim với 5 máy ảo được dựng sẵn để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng Request lần lượt là 100, 200,… đến 900:

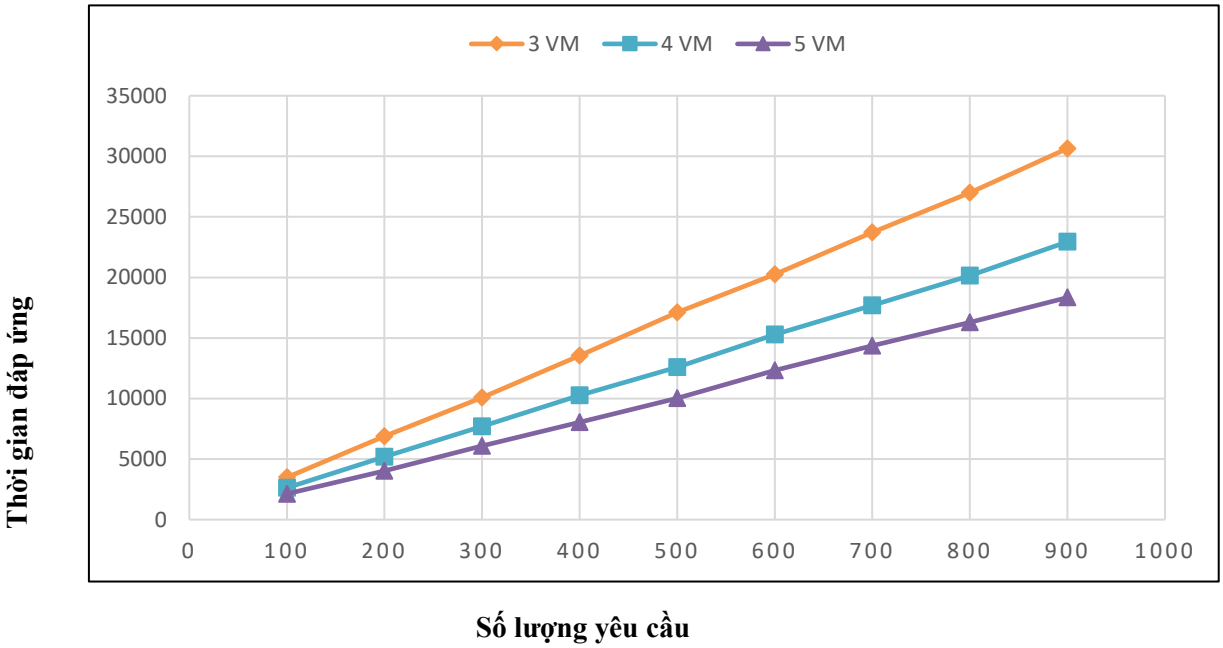
Bảng 2.12. Kết quả thực nghiệm mô phỏng với 5 VM

Số lần request	**VM0**	**VM1**	**VM2**	**VM3**	**VM4**	**Threshold**
100	1878	1904	2042	1828	2114	2133
200	4020	3922	4097	3913	3975	4028

300	6087	6031	5874	5987	5724	6089
400	7896	7913	8045	7820	7760	8048
500	9889	9752	10019	9934	10033	10033
600	12223	11875	12080	12311	12311	12315
700	13771	14346	13852	13939	14279	14347
800	15629	16117	16300	15929	16270	16299
900	18144	17819	17967	18336	18174	18338



Hình 2.8. So sánh thời gian đáp ứng dự báo của 5 VM và ngưỡng



Hình 2.9. So sánh thời gian đáp ứng dự báo trong các trường hợp 3 máy áo, 4 máy ảo và 5 máy ảo.

Thông qua Hình 2.6, Hình 2.7, Hình 2.8 và Hình 2.9, so sánh thời gian đáp ứng dự báo của các VM với ngưỡng tính toán (ứng với trường hợp 3, 4 và 5 VM) có thể thấy sự phân bổ khá ổn định và hợp lý của thuật toán, thời gian đáp ứng dự báo của các VM không quá khác biệt so với thời gian dự báo của đám mây (tức là ngưỡng). Sai số dự báo thấp của thuật toán ARIMA, giúp cho việc phân bổ các yêu cầu tương ứng tới các VM một cách hiệu quả nhất.

Đánh giá thuật toán:

Mô phỏng này chưa tính tới việc mở rộng tập các VM (VM pool) để giảm tải trong trường hợp cần thiết, do giả định nhóm các VM này xử lý tối đa bao nhiêu yêu cầu, nếu vượt quá mới mở rộng pool. Tuy nhiên, việc mô phỏng với lượng request lớn trên 1000 yêu cầu đòi hỏi cấu hình phần cứng máy tính mạnh hơn và bộ xử lý tốt hơn, đây là hạn chế của mô phỏng này. Việc mô phỏng với thông số 3 đến 5 VM, chịu tải từ 100 tới 900 yêu cầu đã cho thấy kết quả tương đối tốt, việc phân bổ các request tới các VM xử lý khá đồng đều và dự đoán với sai số nhỏ.

Thuật toán mới về cân bằng tải trên môi trường đám mây bằng phương pháp dự báo thời gian đáp ứng tiếp theo đã đề xuất (RRTA) được mô phỏng trên mô hình

nhỏ. Dựa trên ý tưởng và các công trình nghiên cứu trước, đưa ra một giải thuật mới ứng dụng thuật toán ARIMA để cân bằng tải dựa vào thời gian đáp ứng. Trong đó, việc tính toán ra thời gian đáp ứng dự báo càng chính xác thì hiệu quả thuật toán càng cao. Tuy nhiên, việc tính toán càng chính xác đòi hỏi tốn nhiều bộ nhớ và bộ xử lý, đồng thời người dùng trên môi trường đám mây có các yêu cầu vô cùng đa dạng và phong phú, nên thời gian đáp ứng cũng biến đổi trên đám mây. Thuật toán RRTA tiếp cận một cách khái quát và phát huy ý tưởng của dự báo và xử lý chuỗi thời gian, điển hình là thuật toán ARIMA. Thuật toán đề xuất có hướng tiếp cận mới trong cân bằng tải ở môi trường đám mây, đồng thời đạt được một số kết quả thực nghiệm mô phỏng khá tích cực, cho thấy hướng phát triển tốt của thuật toán. Phân tích các yếu tố ảnh hưởng đến cân bằng tải, đặc biệt là thời gian đáp ứng là một công việc quan trọng trong việc nâng cao khả năng cân bằng tải trên điện toán đám mây. Các kết quả nghiên cứu về cải tiến thời gian đáp ứng đến hiệu quả cân bằng tải có thể dẫn đến những hiểu biết mới về các hệ thống cân bằng tải góp phần nâng cao khả năng cân bằng tải của điện toán đám mây. Cùng với sự phát triển không ngừng của công nghệ điện toán đám mây dẫn đến lượng dữ liệu trao đổi là khổng lồ, việc đáp ứng các yêu cầu của người dùng thực sự là một thách thức đối với các nhà nghiên cứu.

2.4. Kết luận Chương 2

Nội dung Chương 2 của Luận án trình bày 02 thuật toán nhằm mục đích cải tiến thời gian đáp ứng trên điện toán đám mây cùng các kết quả nghiên cứu thực nghiệm để chứng minh hiệu quả của thuật toán đề xuất.

Thuật toán thứ nhất là LBAIRT (công bố ở công trình CT4): Điểm mới của thuật toán LBAIRT là xét thêm số tham số thời gian hoàn thành công việc dự kiến của mỗi tài nguyên (VM). Kết quả mô phỏng cho thấy LBAIRT có tỉ lệ chấp nhận yêu cầu cao hơn, trong khi thời gian tính toán trung bình thấp hơn dẫn đến thời gian đáp ứng đã được cải thiện so với các công trình liên quan.

Thuật toán thứ hai là thuật toán RRTA (công bố ở công trình CT7), dùng kỹ thuật dự báo ARIMA để dự báo thời gian đáp ứng của VM tiếp theo. Thực nghiệm tiến

hành trên dữ liệu mô phỏng cho thấy, thuật toán RRTA có tỉ lệ chấp nhận yêu cầu cao hơn, cải thiện được thời gian đáp ứng so với các thuật toán Roud Robin, Throtted.

Nội dung Chương 2 được công bố ở các công trình (CT4) và (CT7).

Có nhiều phương pháp để nâng cao hiệu năng cân bằng tải trên điện toán đám mây, trong Chương 3 của Luận án tiếp tục nghiên cứu nâng cao khả năng này dựa trên một trong những tham số quan trọng là thời gian xử lý.

CHƯƠNG 3.

PHÁT TRIỂN MỘT SỐ THUẬT TOÁN CÂN BẰNG TẢI NHẰM CẢI THIỆN THỜI GIAN XỬ LÝ TRÊN ĐIỆN TOÁN ĐÁM MÂY

Nội dung Chương 3 là đề xuất 02 thuật toán cân bằng tải nhằm cải tiến thời gian xử lý trên điện toán đám mây, bao gồm: TMA, MMSIA. Thuật toán TMA, được công bố trong công trình (CT5), với thuật toán đề xuất này sẽ giúp việc dò tìm VM đang sẵn sàng '0' với kích thước bảng "Available Index" thay đổi linh động hơn so với thuật toán Throttled. Thuật toán MMSIA (CT6) đề xuất đã cải tiến thuật toán lập lịch Max-Min giúp cải thiện thời gian xử lý các yêu cầu bằng cách phân nhóm các yêu cầu và các VM theo thời gian hoàn thành dự kiến và thời gian thực hiện hoàn thành tổng thể. Nội dung chương 3 được minh chứng bằng các công bố trong các công trình (CT5) và (CT6).

3.1. Đặt vấn đề

Thời gian xử lý là lượng thời gian mà đơn vị xử lý trung tâm (CPU) sử dụng để xử lý các hướng dẫn của chương trình máy tính hoặc hệ điều hành. Thời gian xử lý (CPU time) của một yêu cầu nhất định được định nghĩa là thời gian dành cho hệ thống thực hiện yêu cầu đó, bao gồm cả thời gian thực hiện các dịch vụ hệ thống. Thời gian CPU là thước đo thực sự của hiệu năng bộ xử lý/bộ nhớ. Thời gian CPU (hoặc thời gian thực thi CPU) là thời gian giữa lúc bắt đầu và kết thúc thực hiện một chương trình nhất định.

Cân bằng tải là cách phân chia và điều phối khối lượng công việc trên nhiều máy tính hoặc một cụm máy tính để các nguồn lực được tận dụng một cách hiệu quả, giảm thiểu được thời gian xử lý và thời gian đáp ứng, tối đa hóa thông lượng, tránh tình trạng quá tải tại một số máy chủ vật lý hay hạn chế số máy chủ nhàn rỗi làm lãng phí tài nguyên. Do đó, các thuật toán cân bằng tải đang cố gắng cải tiến thời gian xử lý các yêu cầu trên điện toán đám mây một cách tối ưu nhằm tăng hiệu năng phục vụ của hệ thống trung tâm dữ liệu đám mây. Vấn đề này hiện nay thực sự là một thử thách lớn, được các nhà nghiên cứu trên thế giới quan tâm rộng rãi và thực chất.

3.2. Thuật toán TMA

3.2.1. Đề xuất thuật toán

Thuật toán được công bố trên công trình (CT5). Thuật toán TMA (Throttled Modified Algorithm) cải thiện thời gian xử lý của trung tâm dữ liệu (Data Center) dựa trên thuật toán gốc Throttled [89]. Phương pháp để cải thiện thời gian xử lý là phân bổ lại các yêu cầu một cách hiệu quả tới các VM, thuật toán TMA được triển khai tại trình cân bằng tải máy chủ ảo (TMAVmLoadBalancer) trong bộ điều khiển trung tâm (Datacenter Controller).

Các bước của thuật toán TMA:

- **Bước 1**: Trình TMAVmLoadBalancer thực hiện cân bằng tải bằng việc cập nhật, duy trì hai bảng chỉ mục.

  - Một bảng chứa thông tin các máy ảo (VM) ở trạng thái sẵn sàng '0'. (Available Index)

  - Một bảng chứa thông tin các máy ảo (VM) ở trạng thái không sẵn sàng '1'. (Busy Index)

  Tại thời điểm bắt đầu, tất cả các máy ảo (VM) đều được cập nhật trong bảng "Available Index" và bảng "Busy Index" là rỗng.
- **Bước 2:** Data Center Controller nhận được một request mới**.**
- **Bước 3:** Data Center Controller truy vấn đến TMAVmLoadBalancer cho phân bổ tiếp theo.
- **Bước 4:** TMAVmLoadBalancer dò tìm và gửi ID máy ảo (VM ID) từ trên xuống trong bảng "Available Index" cho Data Center Controller
  A. Data Center Controller sẽ gửi yêu cầu tới VM xác định bởi ID đó.
  B. Data Center Controller thông báo tới TMAVmLoadBalancer một phân bổ mới.
  C. TMAVmLoadBalancer cập nhật ID máy ảo (VM ID) vừa được gửi vào bảng "Busy Index" và chờ yêu cầu mới từ Data Center Controller

Trường hợp, nếu bảng "Available Index" rỗng (tất cả các VM đang ở trạng thái không sẵn sàng):

A. TMAVmLoadBalancer trả về giá trị -1 cho Data Center Controller.

B. Data Center Controller sắp xếp yêu cầu.

- **Bước 5:** Sau khi xử lý yêu cầu xong, TMAVmLoadBalancer cập nhật lại bảng "Available Index".
- **Bước 6:** Lặp lại Bước 3 cho đến khi bảng "Available Index" rỗng.

Điểm mới của thuật toán TMA: Việc dò tìm VM đang sẵn sàng '0' với kích thước bảng "Available Index" thay đổi linh động hơn so với thuật toán Throttled. Bộ cân bằng tải tốn ít chi phí thời gian do duy trì 2 bảng danh sách các VM "sẵn sàng" và "bận", bộ cân bằng tải chỉ việc gán VM cho các yêu cầu mới đến. Điều này giúp giảm thời gian xử lý các yêu cầu đầu vào của bộ cân bằng tải.

Xác định độ phức tạp tính toán:

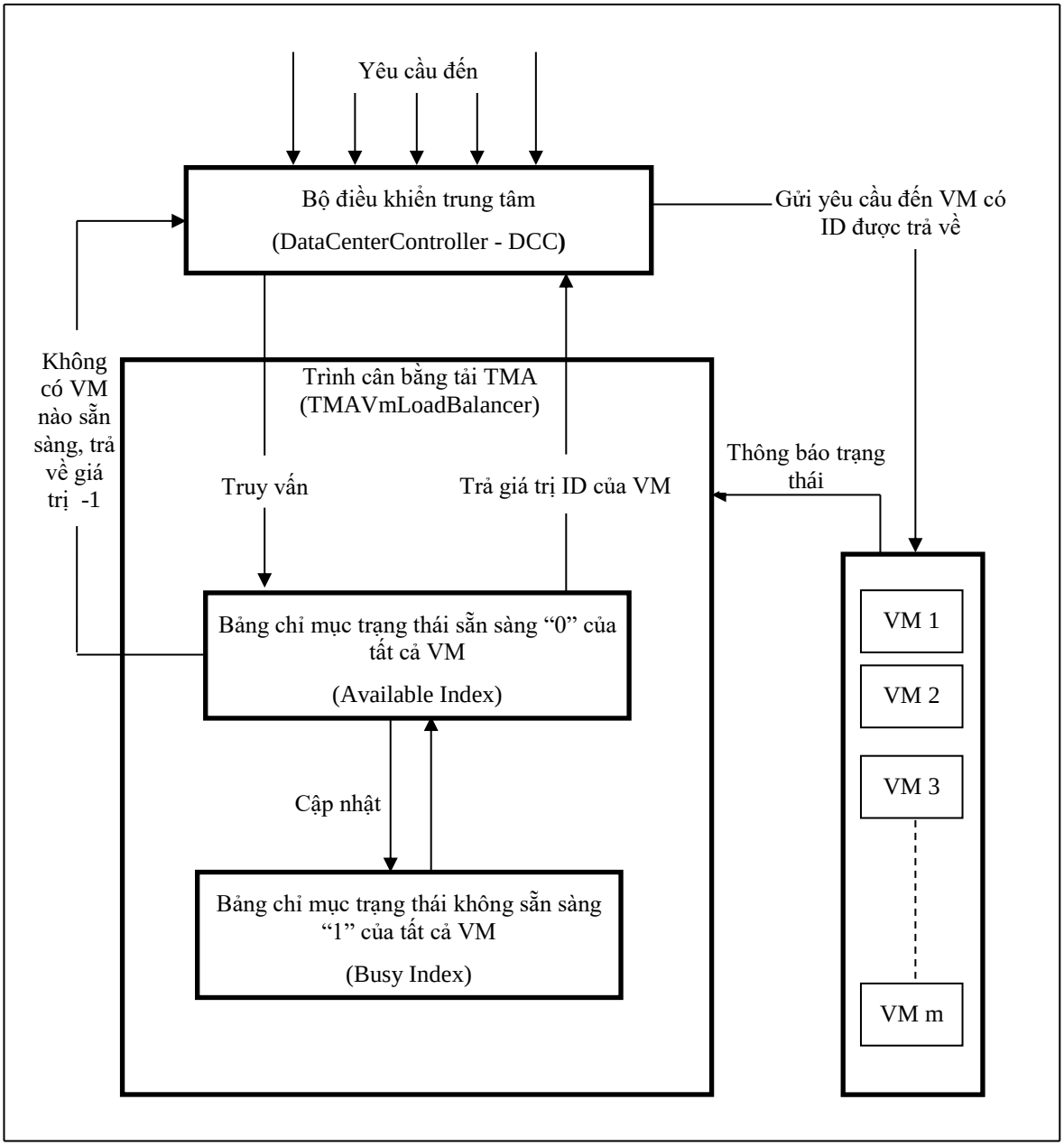
Giả sử số yêu cầu là n và số máy ảo là nm. Đây là bài toán phân bố n requests vào m máy ảo.

Đối với mỗi request:

- DCC truy vấn TMAVmLoadBalancer, độ phức tạp thời gian là O(1)
- TMAVmLoadBalancer tìm kiếm máy ảo rỗi trong Available Index. Đơn giản là dùng vòng for và độ phức tạp thời gian là O(1).
- TMAVmLoadBalancer trả giá trị ID của máy ảo rỗi tới DCC với độ phức tạp thời gian O(1)
- DCC gửi yêu cầu đến máy ảo với ID nhận được từ bước trước O(1)
- Trạng thái của máy ảo được cập nhật ở hai bảng O(1).

Như vậy độ phức tạp thời gian đối với mỗi yêu cầu là O(n).

Có tất cả n yêu cầu, vì vậy độ phức tạp tính toán là: **$O(n^2)$.**



Hình 3.1. Sơ đồ hoạt động của thuật toán TMA

3.2.2 Kết quả mô phỏng

Trong thực nghiệm này, chúng tôi sử dụng bộ công cụ mô phỏng Cloud Analyst để thực hiện mô phỏng và đánh giá thuật toán đề xuất TMA so với 2 thuật toán Round-Robin và Throttled. Trong đó, các thông số như thời gian xử lý của trung

tâm dữ liệu (Data Center Processing Time) và chi phí vận hành được xem xét, so sánh.

Giả lập một mô phỏng có 6 cơ sở người dùng tương ứng với 6 khu vực có múi giờ nhất định. Và giả định là khoảng 5 phút, mỗi người dùng sẽ gửi một yêu cầu mới khi đang online.

Bảng 3.1: Thông số cấu hình User base

User Base	Region	Time Zone	Peak Hour	Simulataneous Online Users During Peak Hrs	Simulataneous Online Users During Off-Peak Hrs
UB1	0	GMT - 6.00	13:00-15:00	400,000	40,000
UB2	1	GMT – 4.00	15:00 – 17:00	100,000	10,000
UB3	2	GMT + 1.00	20:00 – 22:00	300,000	30,000
UB4	3	GMT + 6.00	01:00 – 03:00	150,000	15,000
UB5	4	GMT + 2.00	21:00 – 23:00	50,000	5,000
UB6	5	GMT +10.00	09:00 – 11:00	80,000	8,000

Trong đó:

- Peak Hour: khoảng thời gian cao điểm truy cập
- Simulataneous Online Users During Peak Hrs: số lượng người dùng truy cập trong thời gian cao điểm.
- Simulataneous Online Users During Off-Peak Hrs: số lượng người dùng truy cập trong thời gian thấp điểm.

Chính sách cân bằng tải được áp dụng cho các trung tâm dữ liệu trong việc phân bổ các yêu cầu tới các VM. Ở đây, luận án sẽ mô phỏng 3 lần tương ứng với 3 chính sách khác nhau. Cụ thể:

- Lần thứ 1: áp dụng chính sách Round Robin
- Lần thứ 2: áp dụng chính sách Throtlled
- Lần thứ 3: áp dụng chính sách với thuật toán đề xuất TMA.

Kết quả và phân tích:

Kết quả được phân tích và so sánh trên các thông số chính:

- Số lượng các yêu cầu phải xếp hàng chờ hệ thống phân phối tiếp (-1) do các VM quá tải.

- Thời gian xử lý của trung tâm dữ liệu.

Trường hợp 1: Mô phỏng với số lượng 20 máy ảo.

Bảng 3.2: Số lượng yêu cầu được phân phối tới từng máy ảo (VM)

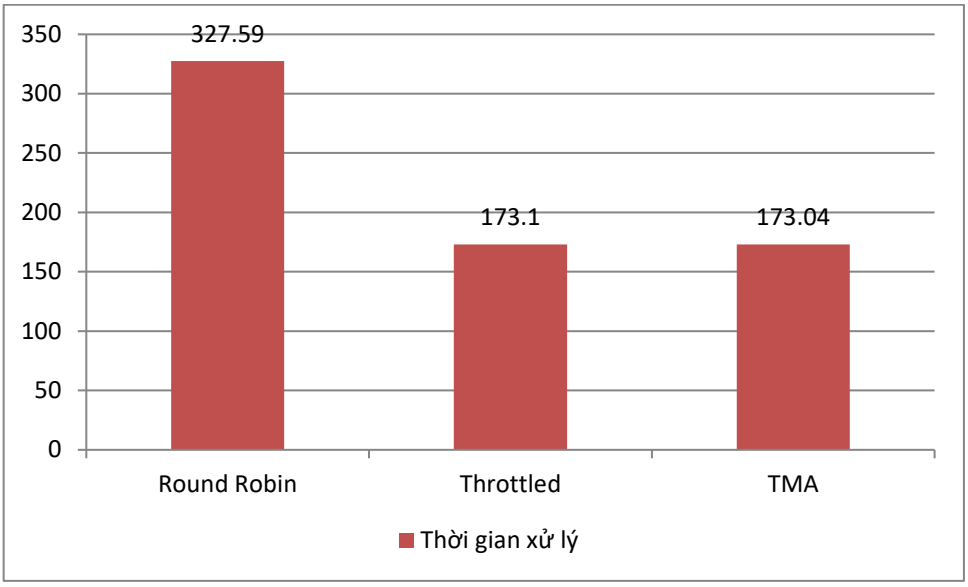
VM ID	Round Robin	Throttled	TMA
0	6532	7959	7978
1	6532	3820	3810
2	6532	3623	3623
3	6531	3509	3523
4	6531	3814	3799
5	6531	3616	3616
6	6531	3540	3540
7	6531	3578	3578
8	6531	3664	3665
9	6531	3388	3389
10	6531	3547	3547
11	6531	3357	3365
12	6531	3435	3435
13	6531	3300	3300
14	6531	3267	3268
15	6531	3230	3230
16	6531	3252	3253

17	6531	3155	3155
18	6531	3134	3134
19	6531	3102	3101
Yêu cầu phải đợi phân phối (-1)	0	57333	0

Bảng 3.2 cho thấy, đối với thuật toán Round Robin các yêu cầu được phân phối đồng đều trên các VM nên không có tình trạng các yêu cầu phải xếp hàng đợi để được phân phối. Với thuật toán Throtlled, việc dò tìm các VM theo phương thức dò tìm từ đầu bảng đến cuối bảng sẽ dẫn đến tình trạng các yêu cầu phải xếp hàng đợi khi hệ thống có số lượng VM lớn. Còn đối với thuật toán TMA, hệ thống chỉ cần phân phối yêu cầu đến các VM sẵn sàng trong bảng chỉ mục Available mà không cần phải dò tìm toàn bộ kích thước bảng. Điều này giúp hạn chế số lượng yêu cầu phải xếp hàng trong hệ thống, cải thiện thời gian xử lý của trung tâm dữ liệu.

Bảng 3.3: Kết quả mô phỏng trường hợp 20 VM

Thuật toán	**Thời gian xử lý trung bình (ms)**
Round Robin	327.59
Throttled	173.10
TMA	173.04



Hình 3.2: Kết quả mô phỏng trường hợp 20 VM.

Thông qua Hình 3.2, ta thấy được đối với thuật toán Round Robin việc phân phối yêu cầu tới VM luân phiên theo vòng tròn mà không có sự sàng lọc tình trạng của VM dẫn đến thời gian xử lý của Data Center và thời gian đáp ứng của hệ thống đến cơ sở người dùng (User base) cao hơn rất nhiều so với 2 thuật toán còn lại. Đối với 2 thuật toán còn lại, thuật toán TMA có thời gian xử lý của Data Center và thời gian phản hồi của hệ thống thấp hơn thuật toán Throttled (mặc dù tỉ lệ này rất nhỏ). Do đó, chúng tôi đã thử tăng số lượng máy ảo (VM) lên 50 máy với cùng các thông số như trên để so sánh một lần nữa.

Trường hợp 2: Mô phỏng với số lượng 50 VM

Bảng 3.4: Số lượng yêu cầu được phân phối tới từng VM

VM ID	Round Robin	Throttled	TMA
0	2614	11432	11432
1	2614	1416	1416
2	2614	1309	1309

3	2613	1210	1210
4	2613	1386	1386
5	2613	1258	1258
6	2613	1235	1234
7	2613	1272	1272
8	2613	1471	1471
9	2613	1137	1137
10	2613	1486	1486
11	2613	1349	1349
12	2613	1438	1438
13	2613	1208	1209
14	2613	1268	1268
15	2613	1157	1157
16	2613	1241	1241
17	2613	1236	1236
18	2613	1091	1091
19	2613	9137	9110
20	2613	1175	1204

21	2613	1087	1087
22	2613	1158	1158
23	2613	1019	1019
24	2613	1243	1242
25	2613	1154	1154
26	2613	1121	1120
27	2613	1047	1047
28	2613	1249	1249
29	2613	1104	1104
30	2613	998	998
31	2613	996	996
32	2613	1110	1109
33	2613	1033	1032
34	2613	1011	1012
35	2613	993	993
36	2613	1045	1045
37	2613	995	995
38	2613	966	966

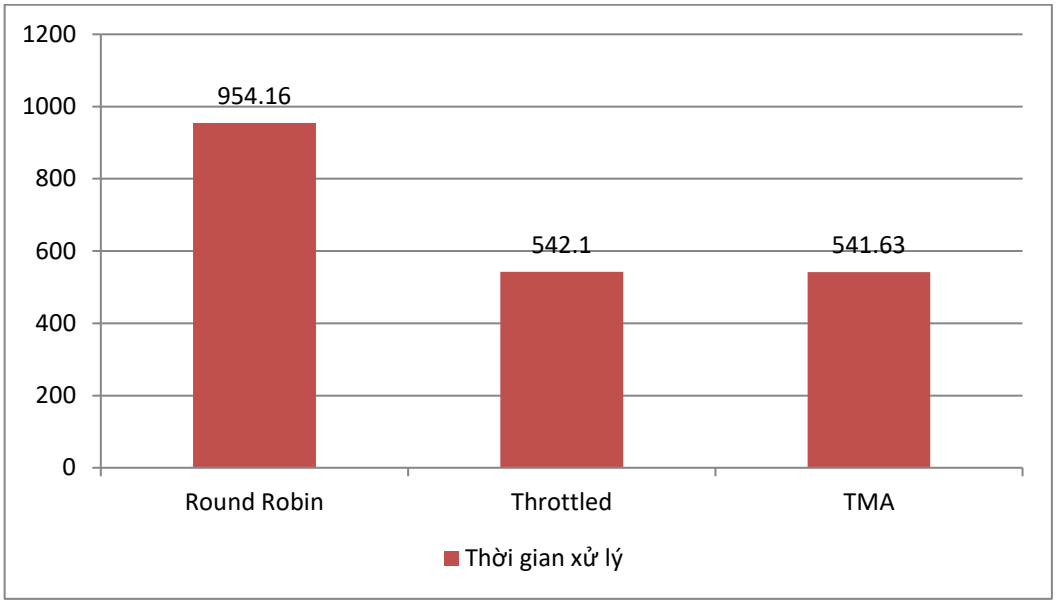
39	2613	8455	8454
40	2613	1038	1038
41	2613	944	945
42	2613	967	967
43	2613	942	943
44	2613	965	965
45	2613	911	910
46	2613	917	917
47	2613	897	897
48	2613	893	893
49	2613	872	872
Yêu cầu phải đợi phân phối (-1)	0	48611	0

Kết quả từ Bảng 3.4 thấy rằng, thuật toán TMA và thuật toán Round Robin có số lượng yêu cầu phải xếp hàng đợi chờ phân phối là 0. Trong khi đối với thuật toán Throttled là lớn.

Bảng 3.5: Kết quả mô phỏng trường hợp 2

Thuật toán	**Thời gian xử lý trung bình (ms)**
Round Robin	954.16
Throttled	542.10

TMA	541.63



Hình 3.3. Kết quả mô phỏng trường hợp 50 VM.

Từ Hình 3.3 cho thấy, thời gian xử lý của Data Center của thuật toán TMA đã giảm so với thuật toán Throttled khi số lượng VM tăng lên. Khi tăng số lượng yêu cầu đầu vào, thì thuật toán đã có sự cải thiến về thời gian xử lý.

3.2.3 Đánh giá:

Thông qua kết quả thực nghiệm mô phỏng cho 2 trường hợp trên, chứng tỏ rằng, với thuật toán TMA thì số lượng yêu cầu phải xếp hàng đợi đã giảm, cũng như thời gian xử lý của Trung tâm dữ liệu. Do đó, thuật toán TMA có khả năng cân bằng tải tốt hơn thuật toán Throttled và Round Robin. Các kết quả thu được từ thuật toán TMA đã đáp ứng các mục tiêu này, chẳng hạn như giới hạn số lượng yêu cầu được xếp hàng để phân phối, cải thiện thời gian xử lý của đám mây so với hai thuật toán cũ.

3.3. Thuật toán MMSIA:

Thuật toán đề xuất MMSIA được công bố trong công trình (CT6). Với ý tưởng cải tiến thuật toán lập lịch Max-Min [69]. Sau đây là mô tả thuật toán Max – Min:

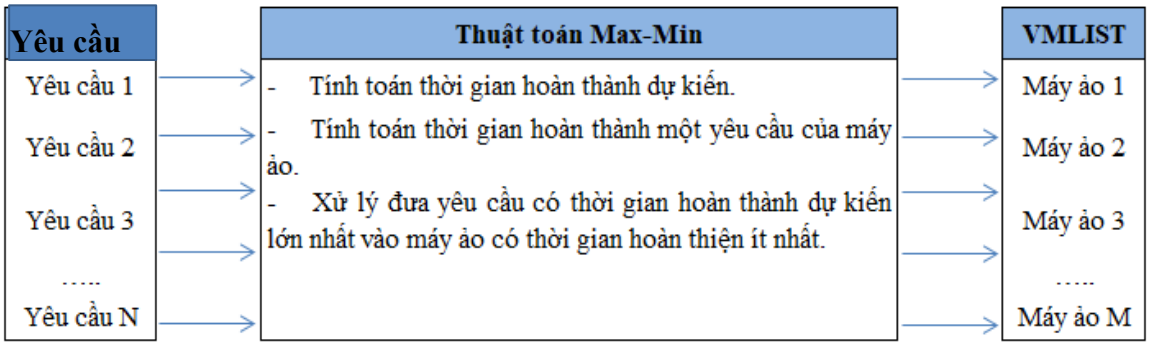
3.3.1 Giới thiệu thuật toán Max - Min

Thuật toán Max-Min [69] chọn yêu cầu với thời gian hoàn thành dự kiến tối đa và gán yêu cầu đó cho máy ảo với thời gian thực hiện tổng thể tối thiểu.

Về mục tiêu:

- Giảm thiểu thời gian "sống" cho các yêu cầu trong điện toán đám mây. Hạn chế tối đa sự mất cân bằng tải giữa các máy ảo.
- Dự báo được thời gian đáp ứng tiếp theo từ bất kỳ máy ảo nào.

Mô hình:



Hình 3.4. Nguyên lý thuật toán lập lịch Max-Min.

Bộ xử lý của thuật toán Max-Min có các hàm tính toán sau:

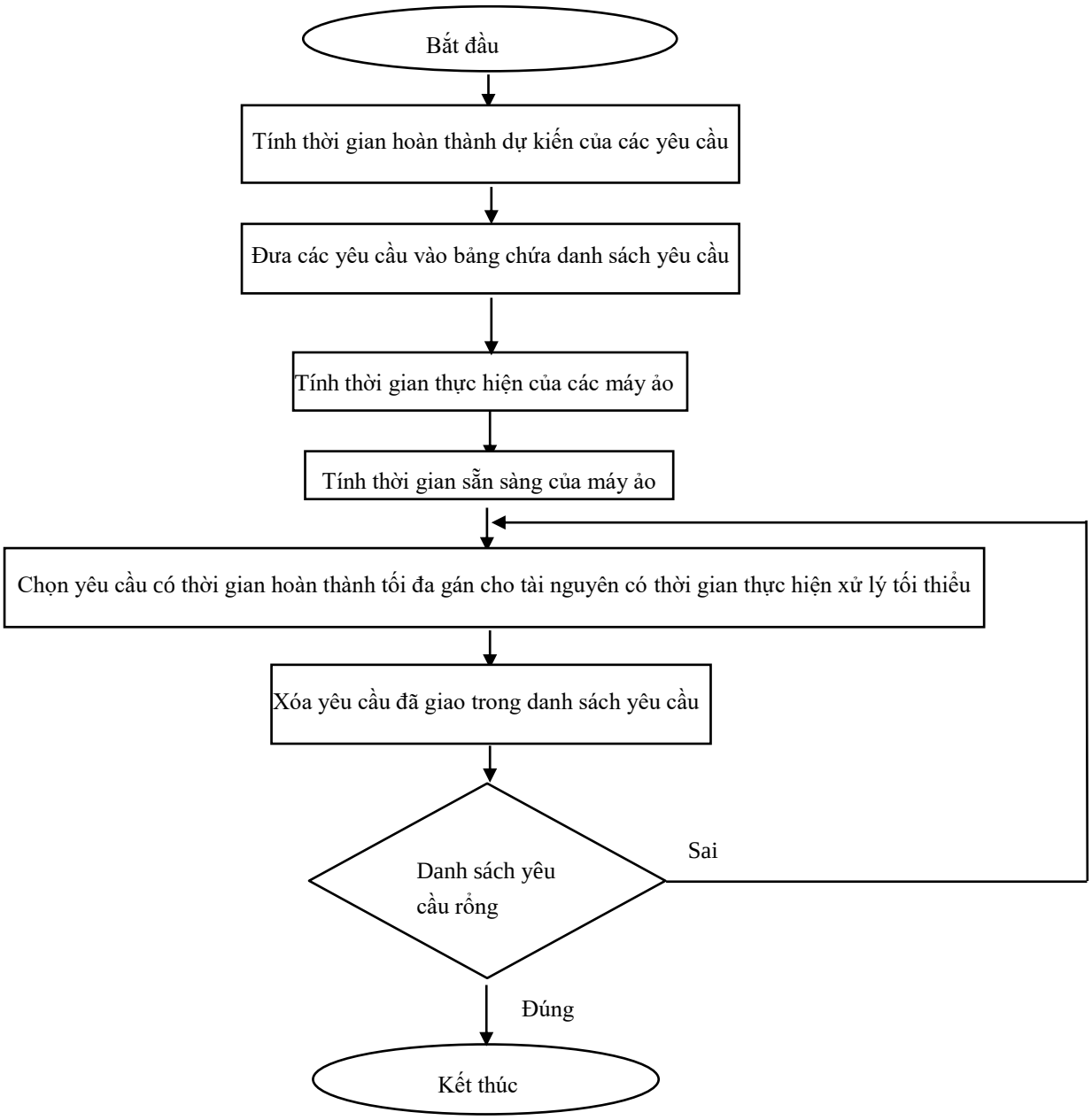
- Hàm tính toán thời gian hoàn thành dự kiến của mỗi yêu cầu từ đó tìm ra các yêu cầu có kích thước lớn nhất và nhỏ nhất.Hàm tính toán thời gian thực hiện tổng thể của các máy ảo từ đó đưa ra các máy ảo có thời gian xử lý lớn nhất và nhỏ nhất.
- Hàm so sánh và gán yêu cầu cho máy ảo phù hợp (công việc có thời gian hoàn thiện lớn nhất gán cho máy ảo có thời gian thực hiện nhỏ nhất)

Nguyên lý hoạt động của thuật toán Max-Min:

- Các yêu cầu sẽ được thu thập và lưu vào bảng chứa danh sách các yêu cầu.
- Từ bảng này hàm tính toán thời gian hoàn thành cho từng yêu cầu và cập nhật lại bảng tương ứng với yêu cầu và thời gian hoàn thành dự kiến.
- Tính toán thời gian thực hiện tổng thể của từng yêu cầu với mỗi máy ảo, đồng thời cũng tính toán thời gian sẵn sàng cho từng máy ảo.

- Thuật toán sẽ tiếp tục tính tổng chi phí cho mỗi yêu cầu với từng máy ảo.
- Gán các yêu cầu có thời gian hoàn thành dự kiến lớn nhất cho các máy ảo có thời gian xử lý nhỏ nhất. Cập nhật danh sách các yêu cầu, dừng chương trình khi hết yêu cầu.

Sơ đồ thuật toán lập lịch Max-Min:



Hình 3.5. Sơ đồ thuật toán lập lịch Max-Min.

Thuật toán Max – Min tính toán thời gian hoàn thành dự kiến của yêu cầu được gửi đến máy ảo, yêu cầu có thời gian thực hiện dự kiến tối đa được gán cho máy ảo có thời gian hoàn thành tối thiểu.

3.3.2. Đề xuất thuật toán MMSIA

Thuật toán đề xuất MMSIA được công bố trong công trình (CT6). Với ý tưởng cải tiến thuật toán lập lịch Max-Min [69], thuật toán MMSIA đã giảm được thời gian xử lý các yêu cầu so với thuật toán Max-Min [69] và giảm thiểu sự mất cân bằng tải trong môi trường điện toán đám mây. Thuật toán lập lịch Max-Min [69] tuy đã cải thiện được thời gian đáp ứng, thời gian hoàn thành của các yêu cầu, giảm thiểu mất cân bằng tải trong hệ thống song thuật toán vẫn còn tồn tại những điểm hạn chế ở việc tính toán thời gian hoàn thành yêu cầu của một máy ảo. Khi đưa một yêu cầu vào, máy ảo phải tính toán lại thời gian hoàn thành dự kiến cho yêu cầu đó (tương tự cho các yêu cầu khác) và sau đó tính toán thời gian hoàn thành dự kiến nhỏ nhất cho yêu cầu đó, dẫn đến việc tính thời gian hoàn thành dự kiến của một máy ảo với 1 yêu cầu sẽ chiếm rất nhiều thời gian và chi phí xử lý. Do đó, thuật toán MMSIA sẽ khắc phục nhược điểm này.

Giả định:

- Bộ cân bằng tải biết trước các yêu cầu đầu vào trong bảng chứa danh sách các yêu cầu, biết danh sách các máy ảo, phần trăm đã sử dụng của máy ảo tại thời điểm hiện tại.

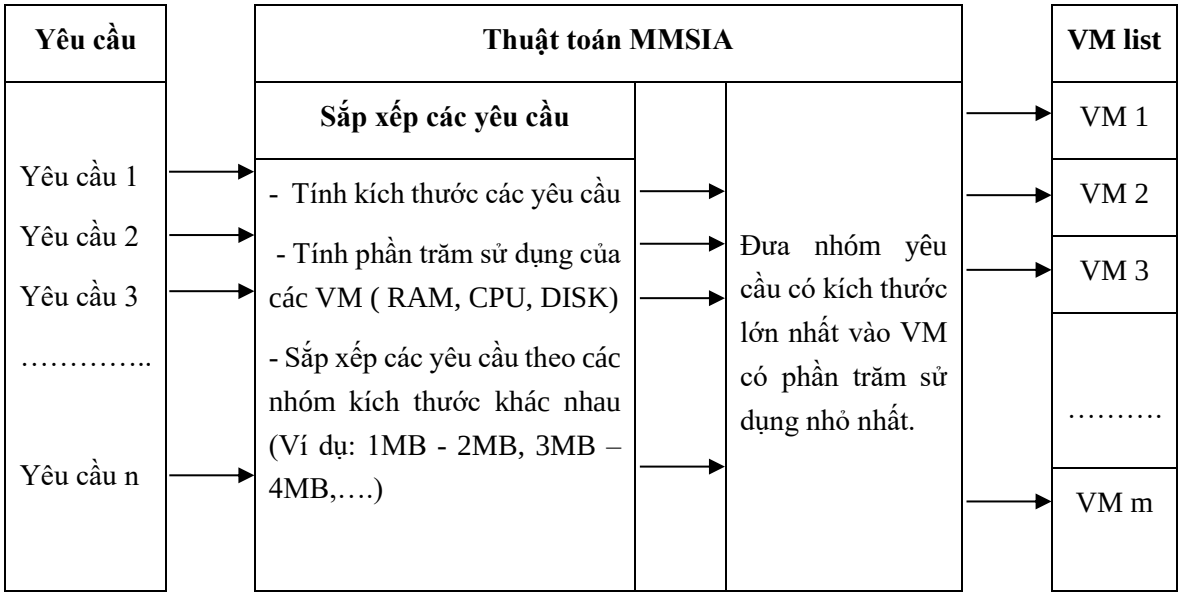
Mục tiêu thuật toán:

- Giảm thời gian xử lý tất cả các yêu cầu vào.
- Tăng độ xử lý yêu cầu của các máy ảo mà không làm mất cân bằng tải.

Ưu điểm thuật toán MMSIA:

So với thuật toán lập lịch Max-Min [69] thì MMSIA tập trung vào tính kích thước của các yêu cầu đầu vào. Đồng thời cũng tính phần trăm sử dụng của máy ảo tại thời điểm đó dựa trên các thông số Ram, CPU và ổ đĩa. MMSIA đã bỏ qua giai đoạn tính thời gian hoàn thành xử lý một yêu cầu của máy ảo, nên đã giảm thời gian xử lý không cần thiết. Mặt khác, khi tính kích thước các yêu cầu đầu vào đồng thời

sẽ phân nhóm thành các kích thước khác nhau, đưa những nhóm có kích thước lớn nhất cho các máy ảo có phần trăm sử dụng thấp nhất để làm giảm thời gian xử lý.
Mô hình:



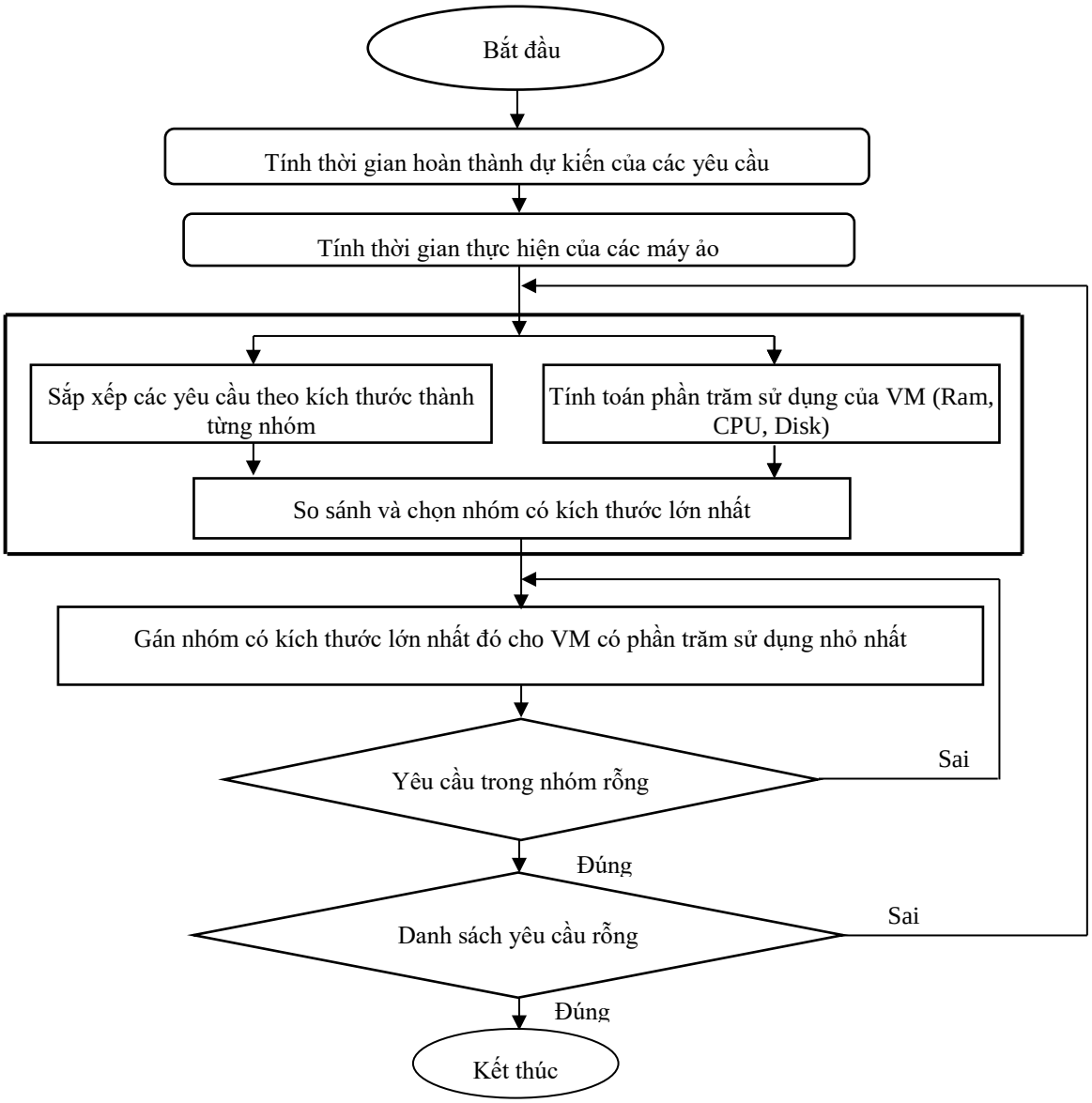
Hình 3.6. Sơ đồ nguyên lý thuật toán MMSIA.

Mô tả:

MMSIA tính kích thước các yêu cầu và tính phần trăm sử dụng của VM theo những nhóm khác nhau từ đó thực hiện 2 hàm sau:

- Hàm so sánh và gán nhóm yêu cầu có kích thước file lớn nhất cho VM có phần trăm dung lượng sử dụng ít nhất.
- Sau khi gán xong sẽ thực hiện xử lý yêu cầu để đưa ra kết quả và tính toán lại phần trăm sử dụng hiện tại của VM.

Thuật toán MMSIA lặp lại đến khi các bảng yêu cầu trống. Các yêu cầu sẽ được xử lý nhanh hơn, giảm mất cân bằng tải cho điện toán đám mây.



Hình 3.7. Sơ đồ thuật toán MMSIA

Thuật toán MMSIA hoạt động trên cơ sở sắp xếp các yêu cầu đầu vào thành nhiều nhóm khác nhau, sau đó sẽ gán cho các VM có phần trăm sử dụng nhỏ nhất (phần trăm sử dụng tính trên CPU, Ram, và Disk) theo cơ chế (Max-Min). Thuật toán này đã cải thiện thời gian xử lý, hạn chế mất cân bằng tải giữa các tài nguyên.

Xác định độ phức tạp tính toán:

Giả sử số requests là n và số máy ảo là m. Độ phức tạp tính toán được tính như sau:

- Tính toán thời gian dự kiến của các yêu cầu O(n)

- Tính thời gian thực hiện của các máy ảo O(m)
- Sắp xếp các yêu cầu theo kích thước thành từng nhóm: dùng vòng for và kiểm tra kích thước từng yêu cầu thuộc nhóm nào rồi gán vào nhóm đó. Độ phức tạp thời gian là: O(n).
- Tính toán phần trăm sử dụng của các VMs: O(m)
- Chọn VM có phần trăm sử dụng nhỏ nhất: chọn VM nào có phần trăm nhỏ nhất và dùng dùng for kiểm tra lại. Độ phức tạp thời gian là: O(m)
- Gán nhóm có kích thước lớn nhất vào VM có phần trăm sử dụng ít nhất O(1).

Như vậy độ phức tạp tính toán cho việc gán yêu cầu với kích thước lớn nhất cho VM có phần trăm sử dụng ít nhất là: O(n+m+n+m+m+1) = O(n+m).

Thực hiện lặp lại n lần để phân bổ hết n yêu cầu, như vậy độ phức tạp tính toán là: **$O(n(n+m)) = O(n^2+nm)$.**

3.3.3. Kết quả mô phỏng

Phần này trình bày về cài đặt, mô phỏng thuật toán đề xuất và từ kết quả thực nghiệm mô phỏng đánh giá phương pháp đề xuất này.

Môi trường mô phỏng:

- Giả lập môi trường đám mây sử dụng bộ thư viện CloudSim và lập trình trên ngôn ngữ JAVA; Môi trường giả lập đám mây là từ 3 đến 10 máy ảo, và tạo môi trường yêu cầu ngẫu nhiên tới các dịch vụ trên đám mây này. Bao gồm dịch vụ cung cấp máy ảo, dịch vụ cung cấp cho người dùng của CloudSim để thử nghiệm.
- Cài đặt thuật toán lập lịch Max-Min, và thuật toán MMSIA trên môi trường mô phỏng, và so sánh kết quả.

Các tham số mô phỏng:

Thực nghiệm mô phỏng cài đặt trên ngôn ngữ JAVA và sử dụng Eclipse Jee Photon để chạy thử và hiển thị kết quả. Môi trường giả lập với bộ thư viện mã nguồn mở CloudSim 4.0 (được cung cấp bởi http://www.cloudbus.org/).

Môi trường mô phỏng giả lập gồm các thông số sau:

- 01 Datacenter với thông số như sau:

Bảng 3.6. Thông số Datacenter

Thông tin Datacenter	**Thông tin Host trong Datacenter**
Số lượng máy (host) trong datacenter: 5 - Không sử dụng Storage (các ổ SAN) - Kiến trúc (arch): x86 - Hệ điều hành (OS): Linux - Xử lý (VMM): Xen - TimeZone: +7 GMT - Chi phí xử lý: 3.0 - Chi phí sử dụng bộ nhớ: 0.05 - Chi phí sử dụng dung lượng: 0.1 - Chi phí sử dụng băng thông: 0.1	Mỗi host trong Datacenter có cấu hình như sau: - CPU có 4 nhân, mỗi nhân có tốc độ xử lý là 1000 (mips) - Ram: 2048 (MB) - Storage: 1000000 - Bandwidth: 10000

- Các máy ảo có cấu hình giống nhau khi được khởi tạo

Bảng 3.7. Cấu hình của VMs

Dung lượng	**Ram**	**Mips**	**Bandwidth**	**Số lượng CPU (PEs no.)**	**VMM**
10000 MB	512 MB	250	1000	1	Xen

Các yêu cầu (các request chạy trên web, WebRequest) được đại diện bởi Cloudlet trong CloudSim và kích thước của Cloudlet được khởi tạo một cách ngẫu nhiên bằng hàm random của JAVA. Số lượng Cloudlet lần lượt 25, 50, 100, 1000.

Bảng 3.8 Thông số các Request.

Chiều dài (Length)	Kích thước file (File Size)	Kích thước file xuất ra (Output Size)	Số CPU xử lý (PEs)
3000 ~ 1700	5000 ~ 45000	450 ~ 750	1

Thuật toán đề xuất được xây dựng bằng cách sử dụng các hàm tổng hợp giá trị Cloudlet để đánh giá:

```
public double getAverageSize()
  {
   double res=0;
   res = this.getCloudletLength() *0.5 + this.getCloudletOutputSize()*0.2 +
     this.getCloudletFileSize()*0.3;
   return res;
  }
```

//Sử dụng thuật toán tính phần trăm trung bình của VM:

```
public double getAverageRequestedResources()
  {
   double res =0;
   double per_mips=(this.getCurrentRequestedTotalMips() / this.getMips())*100 ;
   double per_ram = (this.getCurrentRequestedRam() / this.getRam() ) * 100;
   double per_storage = (this.getCurrentAllocatedSize()/this.getSize()) * 100;
   res = (per_mips + per_ram + per_storage)/3;
   return res;
  }
```

Mô phỏng với các tham số như trên, chạy thuật toán cân bằng tải có sẵn và thuật toán đề xuất cùng đầu vào, so sánh kết quả đầu ra.

Kết quả mô phỏng

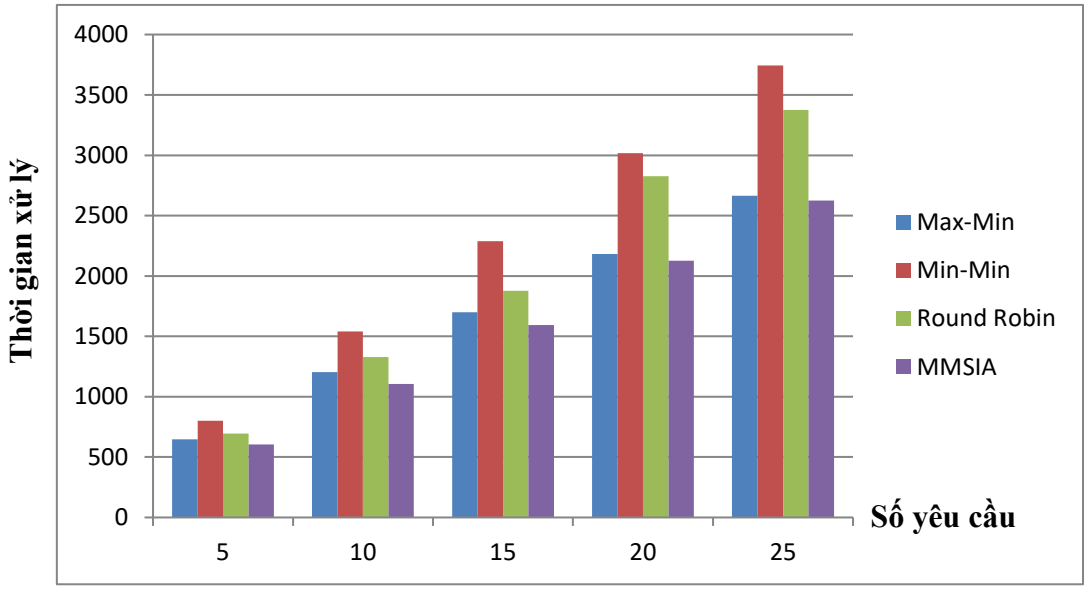
- Quá trình thực nghiệm mô phỏng sẽ được thực hiện trong khoảng 500 yêu cầu, với 4 lần và mỗi lần sẽ có 5 VM và số lượng yêu cầu lần lượt là 25, 50, 100 và 500.

- Thực nghiệm lần 1:

Kết quả chạy mô phỏng trên CloudSim với 5 VM để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng yêu cầu là 25, chạy cho 4 thuật toán như Bảng 3.9.

Bảng 3.9. Bảng kết qủa thời gian xử lý lần 1

Số lượng yêu cầu	**Thời gian xử lý**			
	Max-Min	**Min-Min**	**Round Robin**	**MMSIA**
5	49.93	27.79	46.93	25.78
10	77.13	142.04	120.62	67.02
15	120.94	226.01	180.64	101.75
20	166.08	248.41	210.32	121.44
25	210.59	338.44	276.82	185.74



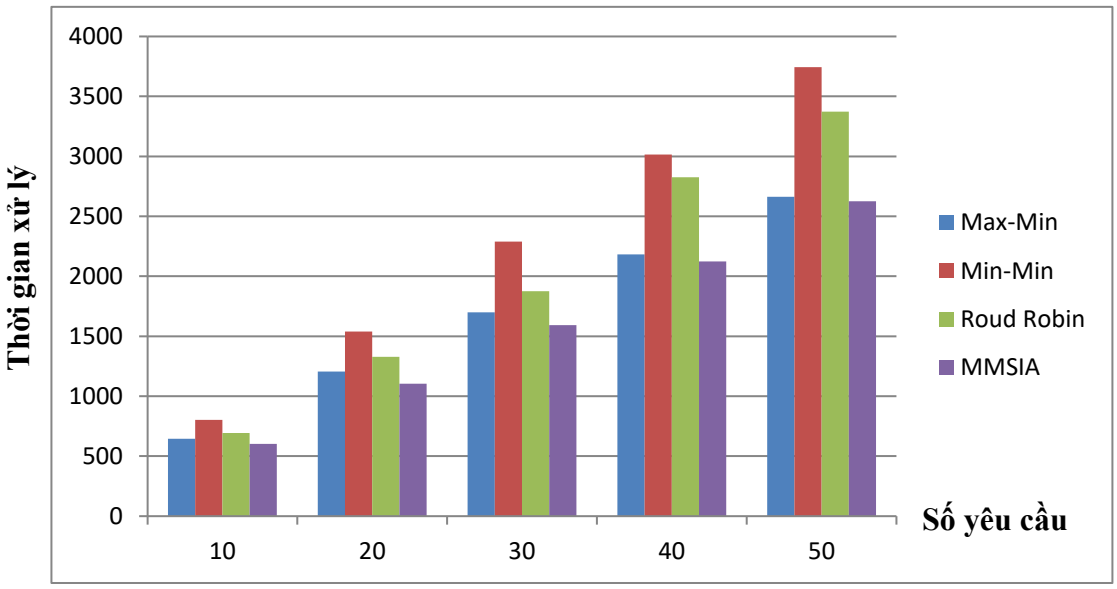
Hình 3.8. Biểu đồ so sánh thời gian xử lý lần 1

- Thực nghiệm lần 2:

Kết quả chạy mô phỏng trên CloudSim với 5 VM để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng Request lần lượt là 50.

Bảng 3.10. Bảng kết qủa thời gian xử lý lần 2

Số lượng yêu cầu	Thời gian xử lý			
	Max-Min	Min-Min	Round Robin	MMSIA
10	40.8	75.7	59.04	30.44
20	72.16	130.36	91.3	56.55
30	103.77	543.67	152.94	96.59
40	142.84	967.41	306.15	120.68
50	175.8	1380.55	476.69	165.63



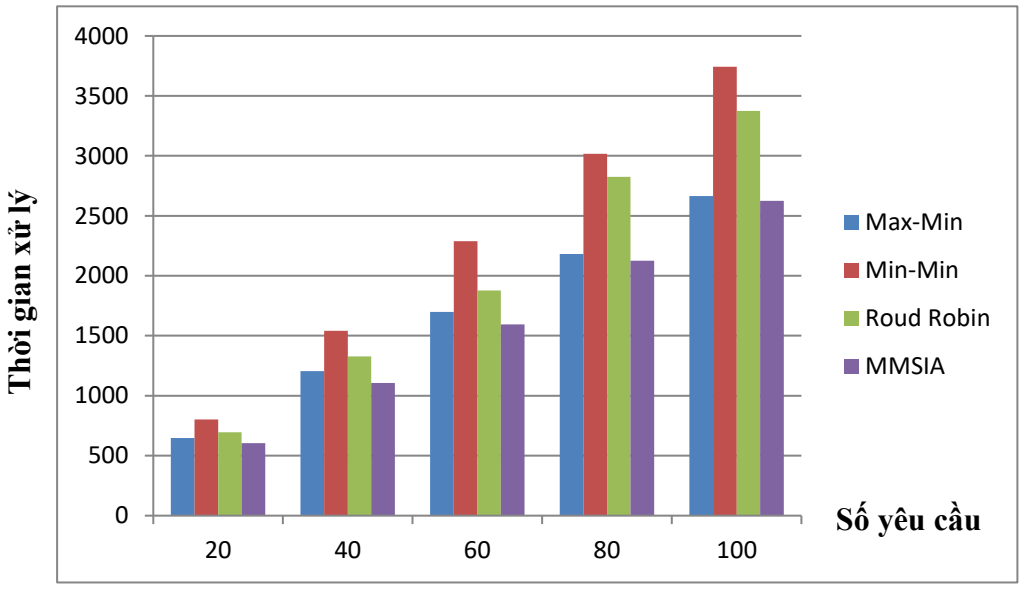
Hình 3.9. Biểu đồ so sánh thời gian xử lý lần 2

- Thực nghiệm lần 3:

  Kết quả mô phỏng trên CloudSim với *5 máy ảo* được dựng sẵn để đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng Request lần lượt là 100.

Bảng 3.11. Bảng kết qủa thời gian xử lý lần 3

Số lượng yêu cầu	Thời gian xử lý			
	Max-Min	Min- Min	Round Robin	MMSIA
20	104.67	110.1	106.53	90.21
40	191.56	177.6	187.09	163.82
60	267.74	250.04	260.56	234.24
80	1552.81	390.56	389.41	346.14
100	2268.1	623.59	597.29	537.79



Hình 3.10 Biểu đồ so sánh thời gian xử lý lần 3

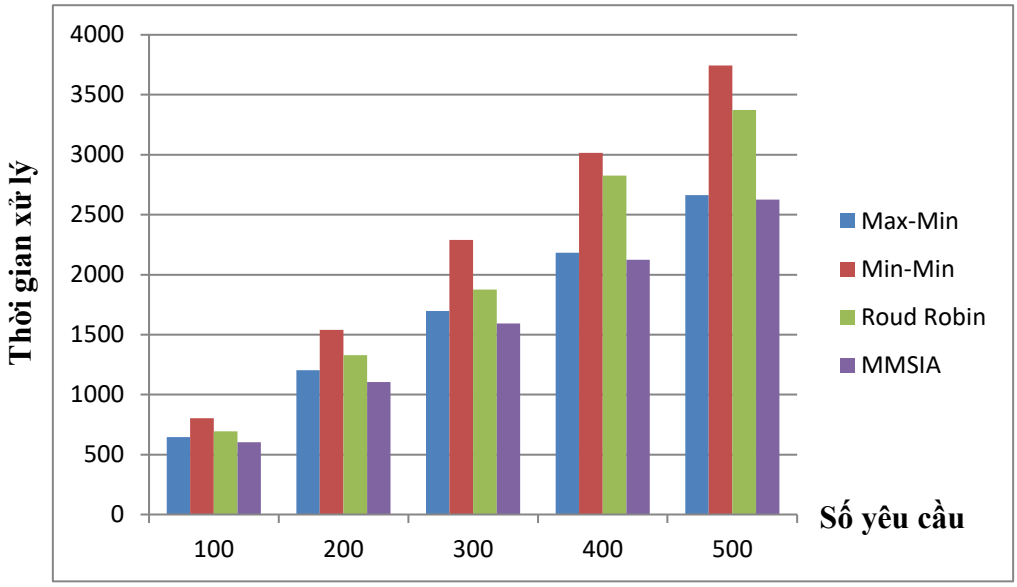
- Thực nghiệm lần 4:

  Kết quả mô phỏng trên CloudSim với 5 VM đáp ứng các yêu cầu, các yêu cầu được khởi tạo với chiều dài và kích thước ngẫu nhiên, số lượng Request lần lượt là 500.

Bảng 3.12. Bảng kết qủa thời gian xử lý lần 4

Số lượng yêu cầu	Thời gian xử lý			
	Max-Min	Min-Min	RoundRobin	MMSIA
100	646.08	802	693.88	603.52

200	1204.61	1540.3	1328.05	1105.17
300	1698.41	2288.48	1876.75	1593.66
400	2181.55	3016.68	2825.5	2125.26
500	2664.22	3743.97	3374.22	2625.95



Hình 3.11. Biểu đồ so sánh thời gian xử lý lần 4

Kết quả của Hình 3.8, Hình 3.9, Hình 3.10, Hình 3.11 cho thấy thời gian xử lý của các VM trên thuật toán MMSIA đã được cải thiện so với các thuật toán được đề tài đưa ra so sánh.

Thực nghiệm này mô phỏng nhóm các VM, chưa tính tới việc mở rộng tập các máy ảo để giảm tải trong trường hợp cần thiết, vì giả định là nhóm các máy ảo này xử lý tối đa bao nhiêu yêu cầu (request), nếu vượt quá mới mở rộng và việc mô phỏng này thực hiện ở những mô hình nhỏ và số lượng yêu cầu ít. Với việc gom nhóm yêu cầu theo kích thước file giúp VM xử lý nhanh hơn, đồng thời làm cho hệ thống phân loại những yêu cầu, từ đó đưa vào những VM có phần trăm xử lý thấp nhất để xử lý. Thực nghiệm cho thấy thuật toán đề xuất đã giảm thiểu thời gian xử lý các yêu cầu. Các thông số cũng như kịch bản đưa ra dựa vào quá trình request của các browser trên môi trường đám mây. Từ đó, ghi nhận các thông số về kích thước file của các

yêu cầu và trung bình phần trăm của VM. Thực nghiệm với 5 máy ảo, số lượng từ 25 đến 500 request đã cho thấy kết quả tương đối tốt, việc phân bổ các request tới các máy ảo xử lý khá đồng đều và kết quả xử lý có sự sai lệch không quá lớn.

3.3. Kết luận Chương 3

Chương 3 đã đề xuất 02 thuật toán cân bằng tải nhằm cải tiến thời gian xử lý trên điện toán đám mây, bao gồm: TMA, MMSIA. Thuật toán TMA (được công bố trong công trình CT5), với thuật toán đề xuất này sẽ giúp việc dò tìm VM đang sẵn sàng ‘0’ với kích thước bảng “Available Index” thay đổi linh động hơn so với thuật toán Throttled. Thuật toán MMSIA (được công bố trong công trình CT6) đề xuất đã cải tiến thuật toán lập lịch Max-Min giúp cải thiện thời gian xử lý các yêu cầu bằng cách phân nhóm các yêu cầu và các VM theo thời gian hoàn thành dự kiến và thời gian thực hiện hoàn thành tổng thể.

Các thuật toán này được so sánh, đánh giá với các thuật toán liên quan Round Robin, Throttled, Max-Min, Min-Min. Thử nghiệm được tiến hành trên các kịch bản khác nhau với số lượng các yêu cầu và số lượng VM khác nhau. Kết quả mô phỏng đã chứng tỏ các thuật toán đề xuất đã cải thiện được thời gian xử lý các yêu cầu.

KẾT LUẬN

Luận án hướng tới một chủ đề có ý nghĩa về lý thuyết và thực tiễn được cộng đồng nghiên cứu quan tâm, là nghiên cứu nâng cao hiệu năng cân bằng tải trên điện toán đám mây. Với mục tiêu nghiên cứu là: phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian đáp ứng và thời gian xử lý trên điện toán đám mây, luận án đã thực hiện được các nội dung đề ra ở phần mục tiêu ban đầu.

I. Những kết quả chính của luận án:

1. Nghiên cứu phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian đáp ứng trên điện toán đám mây:

   - **Đề xuất thuật toán LBAIRT (CT4):** cải tiến thuật toán Throttled với đóng góp chính là việc phân bổ yêu cầu đầu vào đến các máy ảo dựa trên thời gian đáp ứng nhỏ nhất và bằng cách xem xét tham số thời gian hoàn thành các yêu cầu công việc dự kiến của mỗi tài nguyên. Thuật toán đưa vào thời gian hoàn thành dự kiến của mỗi VM cho các yêu cầu trong hàng đợi. Dựa trên tham số này, thuật toán sẽ chọn VM với thời gian hoàn thành dự kiến nhỏ nhất và tỷ lệ sử dụng thấp nhất để phân bổ yêu cầu.

   - **Đề xuất thuật toán RRTA (CT7)**: ứng dụng thuật toán ARIMA để dự đoán ngưỡng thời gian đáp ứng chung của hệ thống và dự đoán thời gian đáp ứng của các máy ảo dựa trên tập yêu cầu tương tự trước đó nhằm đưa ra cách phân phối tài nguyên hợp lý. Thuật toán RRTA tiếp cận một cách khái quát và phát huy ý tưởng của dự báo và xử lý chuỗi thời gian, điển hình là thuật toán ARIMA. Thuật toán đề xuất có hướng tiếp cận mới trong cân bằng tải ở môi trường đám mây, đồng thời đạt được một số kết quả thực nghiệm mô phỏng khá tích cực, cho thấy hướng phát triển tốt của thuật toán

Các kết quả nghiên cứu thực nghiệm dựa trên bộ dữ liệu mô phỏng đã chứng minh hiệu quả và tính đúng đắn của 02 thuật toán đề xuất. Qua đó, giúp cho các nhà cung

cấp dịch vụ điện toán đám mây nâng cao chất lượng dịch vụ cho người dùng trong thực tế

2. Nghiên cứu phát triển một số thuật toán cân bằng tải nhằm cải thiện thời gian xử lý trên điện toán đám mây:

- **Đề xuất thuật toán TMA (CT5):** Thuật toán TMA cải tiến thuật toán Throttled bằng cách chia bảng chứa thông tin máy ảo chung thành hai bảng máy ảo ở trạng thái sẵn sàng và trạng thái không sẵn sàng nhằm giảm thời gian tìm kiếm máy ảo sẵn sàng cho mỗi yêu cầu đầu vào. Điểm mới của thuật toán TMA: Việc dò tìm VM đang sẵn sàng '0' với kích thước bảng "Available Index" thay đổi linh động hơn so với thuật toán Throttled. Bộ cân bằng tải tốt ít chi phí thời gian do duy trì 2 bảng danh sách các VM "sẵn sàng" và "bận", bộ cân bằng tải chỉ việc lấy gán VM cho các yêu cầu mới đến. Điều này giúp tăng hiệu suất xử lý cho hệ thống đồng nghĩa với giảm thời gian xử lý các yêu cầu đầu vào
- **Đề xuất thuật toán MMSIA (CT6):** Thuật toán MMSIA cải tiến thuật toán lập lịch Min – Max bằng cách nhóm các yêu cầu và máy áo theo thời gian hoàn thành dự kiến và thời gian thực hiện hoàn thành tổng thể. Thuật toán MMSIA hoạt động trên cơ sở sắp xếp các yêu cầu đầu vào thành nhiều nhóm khác nhau, sau đó sẽ gán cho các VM có phần trăm sử dụng nhỏ nhất (phần trăm sử dụng tính trên CPU, Ram, và Disk) theo cơ chế (Max-Min). Thuật toán này đã cải thiện thời gian xử lý, hạn chế mất cân bằng tải giữa các tài nguyên

Thông qua thực nghiệm với nhiều kịch bản mô phỏng đã chứng minh tính hiệu quả và tính đúng đắn của 02 thuật toán đề xuất. Đây cũng là cơ sở lý luận cho các nhà phát triển dịch vụ điện toán đám mây nâng cao chất lượng dịch vụ và hiệu năng phục vụ của trung tâm dữ liệu đám mây.

Về mặt thực tiễn: kết quả của luận án đã được thực nghiệm trên các bộ dữ liệu mô phỏng trong các kịch bản khác nhau, kết quả thực nghiệm của phương pháp đề xuất được đánh giá là có hiệu quả hơn các phương pháp đã công bố trong đa số trường hợp, đồng thời là cơ sở khoa học để chế tạo ra các bộ cân bằng tải ứng dụng vào các trung

tâm dữ thực tế. Đây là cơ sở cho thấy, có thể áp dụng kết quả nghiên cứu của đề tài trong việc triển khai các hệ thống cân bằng tải nhằm đối phó với sự bùng nổ trao đổi dữ liệu đám mây hiện nay ở đa dạng các lĩnh vực. Các thuật toán đề xuất được mô phỏng để đánh giá tính hiệu quả so với các thuật toán gốc đã được công bố trước đó.

Phạm vi ứng dụng của các thuật toán đề xuất: Các thuật toán đề xuất được định hướng cho các bộ cân bằng tải (Load Balaner) trong các trung tâm dữ liệu của các nhà cung cấp dịch vụ đám mây, do tính hiệu quả của nó đã được chứng minh thông qua cơ sở lý luận cũng như mô hình thực nghiệm trong luận án. Áp dụng các thuật toán để cải thiện thời gian đáp ứng, thời gian xử lý các yêu cầu từ phía người dùng truy cập đến trung tâm điện toán đám mây.

II. Hướng phát triển của luận án:

1. Luận án có thể được phát triển theo hướng xây dựng mô hình cơ sở dựa vào công nghệ trí tuệ nhân tạo (AI) để nhận diện theo đặc tính riêng lẻ của các yêu cầu đầu vào nhằm đánh giá hiệu năng của hệ thống điện toán đám mây. Từ đó có được mô hình lý thuyết đầy đủ hỗ trợ hoạt động nghiên cứu và triển khai hệ thống điện toán đám mây trong thực tế.
2. Ngoài ra, luận án có thể được phát triển theo hướng cải thiện đồng thời hai tham số: thời gian đáp ứng và thời gian xử lý trên môi trường điện toán đám mây. Đây cũng là một cách tiếp cận rất thiết thực trong bối cảnh bùng nổ trao đổi dữ liệu trên môi trường điện toán đám mây hiện nay.
3. Nghiên cứu cân bằng tải trên mạng lưới vạn vật kết nối (IoT) cũng có thể là một hướng phát triển của luận án khi mà cuộc cách mạng công nghệ 4.0 đang làm thay đổi mọi lĩnh vực trong đời sống hàng ngày, hàng giờ.

DANH MỤC CÁC CÔNG TRÌNH CÔNG BỐ

(CT1)	Tran Cong Hung, Nguyen Khoi, **Nguyen Xuan Phi** (2013), *"Survey traffic matrix for optimizing network performance"*, Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT), October Edition, 2013 Volume 3, Issue 10, ISSN 1925-2676, pages 29-35, October 2013, Canada. Website: http://www.cyberjournals.com/Sep2013.html
(CT2)	**Nguyễn Xuân Phi**, Trần Công Hùng (2015), "*Giải Thuật Phòng Tránh Tình Trạng Quá Tải Trong Điện Toán Đám Mây"*, Proceedings of The 2015 National Conference on Electronics, Communications and Information Technology **ECIT** 2015, pages 66-70, ISBN: 978-604-67-0635-9, December, 10-11, 2015, Ho Chi Minh City, Viet Nam.
(CT3)	**Nguyen Xuan Phi**, Tran Cong Hung (2016), *"Study the Effect of Parameters to Load Balancing in Cloud Computing*", International Journal of Computer Networks & Communications (IJCNC) Vol.8, No.3, May 2016.ISSN:0974-9322 [Online]; 0975-2293 [Print], DOI: 10.5121/ijcnc.2016.8303, pp.33-45, **SCOPUS**, the Australian Research Council (ARC) Journal Ranking, http://airccse.org/journal/ijc2016.html, http://aircconline.com/ijcnc/V8N3/8316cnc03.pdf
(CT4)	**Nguyen Xuan Phi**, Tran Cong Hung (2017), *"Load Balancing Algorithm to Improve Response time on Cloud Computing"*, International Journal on Cloud Computing: Services and Architecture (IJCCSA) Vol. 7, No. 6, December 2017, DOI: 10.5121/ijccsa.2017.7601, pp.1-12, http://airccse.org/journal/ijccsa/current2017.html, http://aircconline.com/ijccsa/V7N6/7617ijccsa01.pdf
(CT5)	**Nguyen Xuan Phi**, Cao Trung Tin, Luu Nguyen Ky Thu, Tran Cong Hung (2018), *"Proposed Load Balancing Algorithm to Reduce Response time and Processing time on Cloud Computing*", International Journal of Computer Networks & Communications (IJCNC) Vol.10, No.3, May 2018, DOI:

	10.5121/ijcnc.2018.10307, pp.87-98, ISSN 0974-9322 (Online), 0975- 2293 (Print), **SCOPUS**, http://airccse.org/journal/ijc2018.html, http://aircconline.com/ijcnc/V10N3/10318cnc07.pdf
(CT6)	Tran Cong Hung, Phan Thanh Hy, Le Ngoc Hieu, **Nguyen Xuan Phi**, *"MMSIA: Improved Max-Min Scheduling Algorithm for Load Balancing on Cloud Computing*", ICMLSC 2019 (Proceedings of The 3rd International Conference on Machine Learning and Soft Computing), pp.60-64 ACM New York, NY, USA @2019 (ISBN: 978-1-4503-6612-0), indexed by Ei Compendex, **SCOPUS**, Da Lat, Vietnam, January 25-28, 2019, https://dl.acm.org/citation.cfm?id=3311017
(CT7)	**Nguyễn Xuân Phi**, Lê Ngọc Hiếu, Trần Công Hùng (2019), *"Thuật toán cân bằng tải nhằm giảm thời gian đáp ứng dựa vào ngưỡng thời gian trên điện toán đám mây"*, Tạp chí Khoa học và Công nghệ về Thông tin và Truyền thông (JSTIC- Journal of Science & Technology on Information and Communications, ISSN: 2525-2224, pp.43-48, 04(CS.01)2018, PTIT, 01/2019

TÀI LIỆU THAM KHẢO

[1] Agarwal A., Jain S. (2014), Efficient Optimal Algorithm and Task Scheduling in Cloud Computing Environment, *International Journal of Computer Trends and Technology (IJCTT)*, vol. 9, pp. 344-349.

[2] Agraj Sharma, Peddoju Sateesh K. (2014), Response Time Based Load Balancing in Cloud Computing, *International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT).*

[3] Anant Kumar Jaiswal, Smriti Srivastava (2013), Clustering based Load Balanced Gateway Placement Approach, *International Journal of Computer Applications (0975 – 8887)* , Volume 63– No.5.

[4] Aruna M, D.Bhanu and S.Karthik (2017), An improved load balanced metaheuristic scheduling in cloud, *Cluster Computing*, DOI: 10.1007/s10586-017-1213-9

[5] Ashis Talukder, Sarder Fakhrul Abedin, Md. Shirajum Munir, and Choong Seon Hong (2017), Dual Threshold Load Balancing in SDN Environment Using Process Migration, *International Conference on Information Networking (ICOIN)*, DOI: 10.1109/ICOIN.2018.8343226, Publisher: IEEE

[6] Atyaf Dhari and Khaldun I. Arif (2017), An Efcient Load Balancing Scheme for Cloud Computing, *Indian Journal of Science and Technology*, Vol 10(11), DOI: 10.17485/ijst/2017/v10i11/110107.

[7] Bahman Keshanchia, Alireza Souri, Nima Jafari Navimipour (2016), An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing, *Journal of Systems and Software 124*, DOI: 10.1016/j.jss.2016.07.006

[8] Bharat Khatavkar, Prabadevi Boopathy (2017), Efficient WMaxMin Static Algorithm For Load Balancing In Cloud Computation, *International Conference on Innovations in Power and Advanced Computing Technologies [i-PACT2017]*, DOI: 10.1109/IPACT.2017.8245166.

[9] Bhathiya Wickremasingle (2010), Cloud Analyst: A CloudSim- based Tool for Modelling and Analysis of Large Scale Cloud Computing Environments, *MEDC Project Report, in 433-659 Distributed Computing Project, CSSE Dept, University of Melbourne*.

[10] Bibhudatta Sahoo, Dilip Kumar and Sanjay Kumar Jena (2013), Analysing the Impact of Heterogeneity with Greedy Resource Allocation Algorithms for Dynamic Load Balancing in Heterogeneous Distributed Computing System, *International Journal of Computer Applications (0975 – 8887)*, Volume 62– No.19.

[11] Branko Radojevic, Mario Zagar (2011), Analysis of Issues with Load Balancing Algorithms in Hosted (Cloud) Environments, *MIPRO 2011*, Opatija, Croatia.

[12] Buyya R, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic (2009), Cloud computing and emerging IT platforms: Vision, hype, and reality for

delivering computing as the 5th utility, *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616.

[13] Buyya R, Broberg J and Goscinski A (2011), Cloud Computing: Principles and Paradigms. *Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011*.

[14] Deepa T, Dhanaraj Cheelu (2017), Load Balancing Algorithms in Cloud Computing: A Comparative Study, *International Journal of Innovations & Advancement in Computer Science IJIACS*, ISSN 2347 – 8616,Volume 6, Issue 1.

[15] Dhinesh Babu,Venkata Krishna P. (2013), Honey bee behavior inspired load balancing of tasks in cloud computing environments, *Elsevier- Journal of Applied Soft Computing*, no-l3, 2013, pp-2292-2303.

[16] Divya Sree A., Bhanu Prakash M. (2016), Load Balancing in Cloud Computing using Dynamic Load Management Algorithm, *International Journal of Advanced Technology and Innovative Research*, Volume. 08, IssueNo.24, Pages: 4740-4744.

[17] Durgesh Patel, Anand S Rajawat (2015), Efficient Throttled Load Balancing Algorithm in Cloud Environment, *International Journal of Modern Trends in Engineering and Research, Scientific Journal Impact Factor (SJIF): 1.711*.

[18] Einollah Jafarnejad Ghomia, Amir Masoud Rahmania, and Nooruldeen Nasih Qader (2017), Load-balancing algorithms in cloud computing: A survey, *Journal of Network and Computer Applications 88 (2017) 50–71, Publisher: Elsevier*

[19] Farzana Sadia, Nusrat Jahan, Lamisha Rawshan, Madina Tul Jeba and Touhid Bhuiyan (2017), A Priority Based Dynamic Resource Mapping Algorithm For Load Balancing In Cloud, *International Conference on Advances in Electrical Engineering (ICAEE)*, DOI: 10.1109/ICAEE.2017.8255349

[20] Feilong Tang, Laurence T. Yang, Can Tang, Jie Li and Minyi Guo (2016), A Dynamical and Load-Balanced Flow Scheduling Approach for Big Data Centers in Clouds, *IEEE TRANSACTIONS ON CLOUD COMPUTING 2016*, DOI 10.1109/TCC.2016.2543722.

[21] Gaochao Xu, Junjie Pang, and Xiaodong Fu (2013), A Load Balancing Model Based on Cloud Partitioning for the Public Cloud, *Tsinghua Science and Technology* , ISSN 1007-0214 04/12, pp 34-39, Volume 18, Number I.

[22] Garima Rastogi, Rama Sushil (2015), Analytical Literature Survey on Existing Load Balancing Schemes in Cloud Computing, *International Conference on Green Computing and Internet of Things (ICGCIoT)*.

[23] Geeta, Santosh Gupta, Shiva Prakash (2019), Qos and Load Balancing in Cloud Computingan Access for Performance Enhancement using Agent Based Software, *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, ISSN: 2278-3075, Volume-8, Issue-11S

[24] Geetha Megharaj, Mohan Kabadi (2018), Run Time Virtual Machine Task Migration Technique for Load Balancing in Cloud, *International Journal of Intelligent Engineering and Systems*, Vol.11, No.5, DOI: 10.22266/ijies2018.1031.25

[25] Guilin Shao, Jiming Chen (2016), A Load Balancing Strategy Based on Data Correlation in Cloud Computing, *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*.

[26] Habibi Farhad, Farnaz Barzinpour and Seyed Jafar Sadjadi (2018), Resource-constrained project scheduling problem: review of past and recent developments, *Journal of Project Management 3 (2018) 55–88*, DOI: 10.5267/j.jpm.2018.1.005

[27] Hadi Khani, Hamed Khanmirza. (2019), Randomized routing of virtual machines in IaaS data centers, *PeerJ Computer Science 5(13):e211*. DOI: 10.7717/peerj-cs.211

[28] Hamed Mahdizadeh. (2017), Designing a Smart Method for Load Balancing in Cloud Computing, *International Journal of Mechatronics, Electrical and Computer Technology (IJMEC) Universal Scientific Organization*, www.aeuso.org PISSN: 2411-6173, EISSN: 2305-0543

[29] Hao Liu, Shijun Liu, Xiangxu Meng, Chengwei Yang, Yong Zhang (2010), LBVS: A Load Balancing Strategy for Virtual Storage, *International Conference on Service Science, IEEE*, DOI: 10.1109/ICSS.2010.27

[30] Harshit Gupta, Kalicharan Sahu (2014), Honey Bee Behavior Based Load Balancing of Tasks in Cloud Computing, *International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064 Impact Factor (2012): 3.358*, Volume 3 Issue 6.

[31] He-Sheng WU, Chong-Jun WANG and Jun-Yuan XIE (2013), TeraScaler ELB-an Algorithm of Prediction-based Elastic Load Balancing Resource Management in Cloud Computing, *International Conference on Advanced Information Networking and Applications Workshops*, DOI: 10.1109/WAINA.2013.79, Publisher: IEEE

[32] Hiren H. Bhatt and Bheda Hitesh A. (2015), Enhance Load Balancing using Flexible Load Sharing in Cloud Computing, *International Conference on Next Generation Computing Technologies (NGCT-2015)*, IEEE.

[33] Hu J, Jianhua Gu, Guofei Sun, Tianhai Zhao (2010), A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud computing Environment, *Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*.

[34] Huahui Lyu, Ping Li, Ruihong Yan, Anum Masood, Bin Sheng,Yaoying Luo (2016), Load Forecast of Resource Scheduler in Cloud Architecture, *International Conference on Progress in Informatics and Computing (PIC)*, DOI: 10.1109/PIC.2016.7949553, Publisher: IEEE

[35] Huankai Chen, Frank Wang, Na Helian, Gbola Akanmu (2013), User-Priority Guided Min-Min Scheduling Algorithm For Load Balancing in Cloud Computing, *Parallel Computing Technologies (PARCOMPTECH)*, National Conference IEEE.

[36] Jananta Permata Putra, Supeno Mardi Susiki Nugroho, Istas Pratomo (2017), Live Migration Based on Cloud Computing to Increase Load Balancing, *International Seminar on Intelligent Technology and Its Application*, DOI: 10.1109/ISITIA.2017.8124096.

[37] Jasmin James, Bhupendra Verma (2012), Efficient VM Load Balancing Algorithm for the Cloud Computing Environment, *International Journal on Computer Science and Engineering (IJCSE)*, ISSN : 0975-3397 Vol. 4 No. 09.

[38] Jean Pepe Buanga Mapetu, Zhen Chen and Lingfu Kong (2018), Heuristic Cloudlet Allocation Approach Based on Optimal Completion Time and Earliest Finish Time, *Published in IEEE Access 2018, DOI:10.1109/access.2018.2876033*.

[39] Jie Cui, Qinghe Lu, Hong Zhong, Miaomiao Tian, and Lu Liu (2018), A Load-balancing Mechanism for Distributed SDN Control Plane Using Response Time, *IEEE Transactions on Network and Service Management*, Volume: 15 , Issue: 4, DOI: 10.1109/TNSM.2018.2876369

[40] Jing V. Wang, Nuwan Ganganath, Chi-Tsun Cheng, and Chi K. Tse (2017), A Heuristics-based VM Allocation Mechanism for Cloud Data Centers, *IEEE International Symposium on Circuits and Systems (ISCAS)*, DOI: 10.1109/ISCAS.2017.8050470

[41] Jinhua Hu, Jianhua Gu, Guofei Sun, Tianhai Zhao (2010), A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment, *International Symposium on Parallel Architectures*, DOI: 10.1109/PAAP.2010.65.

[42] Jitendra Singh (2014), Study of Response Time in Cloud Computing, I.J. Information Engineering and Electronic Business, *Published Online October 2014 in MECS (http://www.mecs-press.org/*), DOI: 10.5815/ijieeb.2014.05.06.

[43] Jon Kleinberg, Eva Tardos (2006), Algorithm Design, *Conell University, Copyright 2006 by Pearsion Education Inc*, ISBN 0-321-29535-8, Publisher: Addison-Wesley.

[44] Jun Duan and Yuanyuan Yang (2016), A Data Center Virtualization Framework towards Load Balancing and Multi-tenancy, *IEEE 17th International Conference on High Performance Switching and Routing*, DOI: 10.1109/HPSR.2016.7525633.

[45] Keng-Mao Cho, Pang-Wei Tsai, Chun-Wei Tsai (2015), A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing, *Neural Comput & Applic Neural Comput & Applic*, DOI 10.1007/s00521-014-1804-9.

[46] Kokilavani T., George Amalarethinam D. I. (2011), Load Balanced Min-Min Algorithm for Static Meta Task Scheduling in Grid computing, *International Journal of Computer Applications*, Vol-20, No.2.

[47] Komalpreet Kaur, Rohit Mahajan (2018), Equally Spread Current Execution Load Algorithm - A Novel Approach for Improving Data Centre's Performance in Cloud Computing, *International Journal on Future Revolution in Computer Science & Communication Engineering*, ISSN: 2454-4248 Volume: 4 Issue: 8. IJFRCSCE.

[48] Konjaang. J. Kok, Fahrul Hakim Ayob, Abdullah Muhammed (2017), An Optimized MaxMin Scheduling Algorithm Cloud Computing, *Journal of Theoretical and Applied Information Technology*, ISSN: 1992-8645, E-ISSN: 1817-3195.

[49] Kripa Sekaran, Kosala Devi. K. R. (2017), SIQ Algorithm for Efficient Load Balancing In Cloud, *International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*

[50] Kun Li, GaochaoXu, Guangyu Zhao, Yushuang Dong, Dan Wang (2011), Cloud Task scheduling based on Load Balancing Ant Colony Optimization, *Sixth Annual ChinaGrid Conference*, DOI 10.1109/ChinaGrid.2011.17, IEEE Computer Society

[51] Leszek Sliwko and Vladimir Getov (2015), A Meta-Heuristic Load Balancer for Cloud Computing Systems, *IEEE 39th Annual International Computers, Software & Applications Conference*, DOI 10.1109/COMPSAC.2015.223.

[52] Louai Sheikhani, Yaohui Chang, Chunhua Gu, Fei Luo (2017), Modifying Broker Policy for Better Response Time in Datacenters, *IEEE International Conference on Computer and Communications (ICCC)*, DOI: 10.1109/CompComm.2017.8322977

[53] Magesh Kumar B., Ramesh C (2015), Green Computing Approach in Dynamic Resource Allocation for VM Environment, *JETIR*, ISSN-2349-5162), Volume 2, Issue 4

[54] Mahesh B. Nagpure, Prashant Dahiwale, Punam Marbate (2015), An Efficient Dynamic Resource Allocation Strategy for VM Environment in Cloud, *International Conference on Pervasive Computing (ICPC), DOI: 10.1109/PERVASIVE.2015.7087186*

[55] Mallikarjuna B., Arun Kumar Reddy D. (2019), The Role of Load Balancing Algorithms in Next Generation of Cloud Computing, *Jour of Adv Research in Dynamical & Control Systems*, Vol. 11, 07-Special Issue.

[56] Manisha Malhotra, Aarti Singh (2015), Adaptive Framework for Load Balancing to Improve the Performance of Cloud Environment, *IEEE International Conference on Computational Intelligence & Communication Technology*, DOI 10.1109/CICT.2015.11

[57] Mao-Lun Chiang, Hui-Ching Hsieh, Wen-Chung Tsai, Ming-Ching Ke (2017), An Improved Task Scheduling and Load Balancing Algorithm under the Heterogeneous Cloud Computing Network, *IEEE 8th International Conference on Awareness Science and Technology (iCAST 2017).*

[58] Mark van der Boor, Sem Borst, and Johan van Leeuwaarden (2017), Load Balancing in Large-Scale Systems with Multiple Dispatchers, *IEEE Conference on Computer Communications*, DOI: 10.1109/INFOCOM.2017.8057012

[59] Mohammad Riyaz Belgaum, Safeeullah Soomro, Zainab Alansari, Muhammad Alam (2018), Load Balancing with preemptive and non-preemptive task scheduling in Cloud Computing, *IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS).*

[60] Nan X, Yifeng He and Ling Guan (2013), Optimization of Workload Scheduling for Multimedia Cloud Computing, *Proc. IEEE InternationalSymposium on Circuits and Systems (ISCAS)*, pp. 1–4.

[61] Nayandeep Sran, Navdeep Kaur (2013), Comparative Analysis of Existing Load balancing techniques in cloud computing, *International Journal of Engineering Science Invention*, Vol-2, Issue-1.

[62] Nidhi Jain Kansal, Inderveer Chana (2012), Existing Load balancing Techniques in cloud computing: A systematic review, *Journal of Information system and communication*, Vol-3, Issue-1.

[63] Padmavathi M., Mahaboob Basha Shaik (2017), Dynamic And Elasticity ACO Load Balancing Algorithm for Cloud Computing, *International Conference on Intelligent Computing and Control Systems (ICICCS)*.

[64] Pawan Kumar, Rakesh Kumar (2019), Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey, *ACM Computing Surveys*, Vol. 51, No. 6, DOI: 10.1145/3281010.

[65] Pericherla S Suryateja (2016), A Comparative Analysis of Cloud Simulators, *International Journal of Modern Education and Computer Science*, DOI: 10.5815/ijmecs.2016.04.08

[66] Peter Mell, Timothy Grance (2011), The NIST Definition of Cloud Computing, *NIST Special Publication 800-145*.

[67] Rafiqul Zaman Khan, Javed Ali (2012), Classification of Task Partitioning and Load Balancing Strategies in Distributed Parallel Computing Systems, *International Journal of Computer Applications (0975 – 8887)*, Volume 60– No.17.

[68] Rajwinder Kaur, Pawan Luthra (2014), Load Balancing in Cloud Computing, *Proc. of Int. Conf. on Recent Trends in Information, Telecommunication and Computing*, ITC.

[69] Rajwinder Kaur, Pawan Luthra (2014), Load Balancing in Cloud System using Max Min and Min Min Algorithm, *Proceedings on National Conference on Emerging Trends in Computer Technology (NCETCT- Number 1)*, pp.31-34.

[70] Rashmi. K. S, Suma. V, Vaidehi. M (2012), Enhanced Load Balancing Approach to Avoid Deadlocks in Cloud, *Special Issue of International Journal of Computer Applications (0975 – 8887) on Advanced Computing and Communication Technologies for HPC Applications - ACCTHPCA*.

[71] Rasim Alguliyev, Y. N. Imamverdiyeva, and F. J. Abdullayeva (2019), PSO-based Load Balancing Method in Cloud Computing, *Automatic Control and Computer Sciences 53(1):45-55*. DOI: 10.3103/S0146411619010024

[72] Reena Panwar, Bhawna Mallick (2015), Load Balancing in Cloud Computing Using Dynamic Load Management Algorithm, *International Conference on Green Computing and Internet of Things (ICGCIoT)*, DOI:10.1109/ICGCIoT.2015.7380567

[73] Ritu Kapur (2015), A Workload Balanced Approach for Resource Scheduling in Cloud Computing, *Eighth International Conference on Contemporary Computing (IC3)*, DOI: 10.1109/IC3.2015.7346649

[74] Rodrigo N. Calheiros, Rajiv Ranjan2, Anton Beloglazov1, Cesar A. F. De Rose (2010), CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Software: Practice and Experience (SPE)*, Volume 41 Number 1, pp.23-50.

[75] Ross Ihaka (2005), Time Series Analysis, *Lecture Notes for 475.726, Statistics Department, University of Auckland.*

[76] Roy A. (2013), Dynamic load balancing: improve efficiency in cloud computing, *International Journal of Emerging Research in Management &Technology*, vol. 9359, no. 4, pp. 78–82, 2013.

[77] Saher Manaseer, Metib Alzghoul, Mazen Mohmad (2019), An Advanced Algorithm for Load Balancing in Cloud Computing using MEMA Technique, *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, ISSN: 2278, 3075, Volume 8 Issue3.

[78] Sajjan R.S, Biradar Rekha Yashwantrao (2017), Load Balancing and its Algorithms in Cloud Computing: A Survey, *International Journal of Computer Sciences and Engineering*, Volume-5, Issue-1 E-ISSN: 2347-2693

[79] Sambit Kumar Mishra, Bibhudatta Sahoo, Priti Paramita Parida (2018), Load Balancing in Cloud Computing: A big Picture, *Journal of King Saud University - Computer and Information Sciences*.

[80] Sambit Kumar Mishra, Md Akram Khan, Bibhudatta Sahoo, Deepak Puthal and Mohammad S. Obaidat, and KF Hsiao (2017), Time Efficient Dynamic Threshold-based load balancing technique for cloud computing, *International Conference on Computer, Information and Telecommunication Systems (CITS)*, DOI: 10.1109/CITS.2017.8035327, Publisher: IEEE.

[81] Saranya D., Sankara Maheswari L. (2015), Load Balancing Algorithms in Cloud Computing: A Review, *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 5, Issue 7, ISSN: 2277 128X.

[82] Shagufta Khan, Niresh Sharma (2013), Ant Colony Optimization for Effective Load Balancing In Cloud Computing, *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, Volume 2, Issue 6, ISSN 2278-6856.

[83] Shalu Mall, Sharma A.K. (2018), Analyzing Load on Cloud: A Review, *Second International Conference on Computing Methodologies and Communication (ICCMC 2018)* ; IEEE Xplore ISBN:978-1-5386-3452-3.

[84] Shikha Gupta, Suman Sanghwan (2015), Load Balancing in Cloud Computing: A Review, *International Journal of Science, Engineering and Technology Research (IJSETR),* Volume 4, Issue 6.

[85] Shivani Dubey, Mamta Dahiya and Sunayana Jain (2019), Implementation of Load Balancing Algorithm with Cloud Collaboration for Logistics, *Journal of Engineering and Applied Sciences 14 (2): 507-515.*

[86] Shivangi Mayur, Nidhi Chaudhary (2019), Enhanced Weighted Round Robin Load Balancing Algorithm in Cloud Computing, *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, ISSN: 2278-3075, Volume-8, Issue- 9S2.

[87] Shubham Sidana, Neha Tiwari (2016), NBST Algorithm: A load balancing algorithm in cloud computing, *International Conference on Computing, Communication and Automation*, DOI: 10.1109/CCAA.2016.7813914.

[88] Simranjit Kaur, Tejinder Sharma (2018), Efficient Load Balancing using Improved Central Load Balancing Technique, *2nd International Conference on Inventive Systems and Control (ICISC),* DOI: 10.1109/ICISC.2018.8398857.

[89] Hafiz Jabr Younis (2015), "Efficient Load Balancing Algorithm in Cloud Computing", *slamic University Gaza Deanery of Post Graduate Studies Faculty Of Information Technology.*

[90] SomulaRamasubbareddy, **T**. AdityaSaiSrinivas, K. Govinda, S.S. Manivannan and E. Swetha (2019), Analysis of Load Balancing Algorithms using Cloud Analyst, *International Journal of Recent Technology and Engineering (IJRTE)*, ISSN: 2277-3878, Volume-7 Issue-6S2.

[91] Sotomayor B, Ruben Santiago Montero, Ignacio Martın Llorente, and Ian Foster (2008), Capacity Leasing in Cloud Systems using the OpenNebula Engine, *Workshop on Cloud Computing and its Applications 2008 (CCA08).*

[92] Sotomayor B, Ruben S. Montero, and Ian Foster (2009), Virtual Infrastructure Management in Private and Hybrid Clouds, *IEEE Internet Comput.*, vol. 13, no. 5, pp. 14–22.

[93] Srinivasa Rao P., Govardhan A. (2013), Dynamic Load Balancing With Central Monitoring of Distributed Job Processing System, *International Journal of Computer Applications (0975 – 8887)* ,Volume 65– No.21.

[94] Subasish Mohapatra, Ishan Aryendu, Anshuman Panda, Aswini Kumar Padhi (2018), A modern approach for load balancing sing forest optimazation algorithm, *Second International Conference on Computing Methodologies and Communication (ICCMC),* EEE Xplore ISBN:978-1-5386-3452-3.

[95] Sukhpal Singh, Inderveer Chana (2016), A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges, *J Grid Computing (2016) 14:217–264*.

[96] Sukhpal Singh, Inderveer Chana (2015), QRSF: QoS-aware resource scheduling framework in cloud computing, *Supercomput (2015) 71:241–292*, DOI: 10.1007/s11227-014-1295-6

[97] Sutha K., Kadhar Nawaz G.M. (2016), Research Perspective of Job Scheduling in Cloud Computing, *IEEE Eighth International Conference on Advanced Computing (ICoAC).*

[98] Swaroop S. M, Rajadeepan D. Ramesh and Digamber Powoar (2013), Analysis of load balancers in cloud computing, *International Journal of Computer Science and Engineering (IJCSE)*, vol. 2, no. 2, pp. 101–108.

[99] Syed Hamid Hussain Madni, Muhammad Shafie Abd Latiff, Yahaya Coulibaly, Shafi'i Muhammad Abdulhamid (2017), Recent advancements in resource allocation techniques for cloud computing environment: a systematic review, *Cluster Computing*, DOI: 10.1007/s10586-016-0684-4

[100] Tanvi Gupta, SS.Handa, Supriya Panda (2017), A Survey on Honey Bee Foraging Behavior and Its Improvised Load Balancing Technique, *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887.

[101] Tejinder Sharma, Vijay Kumar Banga (2013), Efficient and Enhanced Algorithm in Cloud Computing, *International Journal of Soft Computing and Engineering (IJSCE),* ISSN: 2231-2307, Volume-3, Issue-1.

[102] Uma J., Ramasamy V., Kaleeswaran A. (2014), Load Balancing Algorithms in Cloud Computing Environment - A Methodical Comparison, *International Journal of Engineering Research & Technology (IJERT),* ISSN: 2278-0181, Vol. 3, Issue 2.

[103] Varsha Soni, Nemi Chand Barwar (2018), Performance Analysis of Enhanced Max-Min and Min-Min Task Scheduling Algorithms in Cloud Computing Environment, *International Conference on Emerging Trends in Science, Engineering and Management (ICETSEM-2018)*

[104] Vivek Gehlot, S.P. Singh, Akash Saxena (2019), Analyzing Task Scheduling Algorithms and Load Balancing Approach to Enhance the Performance of Cloud Computing, *ICAESMT 2019.*

[105] Vuyyuru Krishna Reddy (2016), Performance Analysis of Load Balancing Algorithms in Cloud Computing Environment, *Indian Journal of Science and Technology,* Vol 9(18), DOI: 10.17485/ijst/2016/v9i18/90697,

[106] Wenhong Tian, Xianrong Liu, Chen Jin, Yuanliang Zhong (2013), LIF: A Dynamic Scheduling Algorithm for Cloud Data Centers Considering Multi-dimensional Resources, *Journal of Information & Computational Science*, 3925–3937.

[107] Xiaoming Nan, Yifeng He and Ling Guan (2013), Optimization of Workload Scheduling for Multimedia Cloud Computing, *IEEE International Symposium on Circuits and Systems (ISCAS2013),* DOI: 10.1109/ISCAS.2013.6572478

[108] Yingchi Mao, Daoning Ren, Xi Chen (2013), Adaptive Load Balancing Algorithm Based on Prediction Model in Cloud Computing, *Proceedings of the Second International Conference on Innovative Computing and Cloud Computing (ICCC 13),* Pages 165–170, DOI:10.1145/2556871.2556907.

[109] Youssef FAHIM, Elhabib BEN LAHMAR, El houssine LABRlJI, Ahmed EDDAOUI (2014), The load balancing based on the estimated fnish time of tasks in cloud computing, *Second World Conference on Complex Systems (WCCS),* DOI: 10.1109/ICoCS.2014.7060891, Publisher: IEEE

[110] Dalia Abdulkareem Shafiq, N.Z. Jhanjhi, Azween Abdullah (2021), Load balancing techniques in cloud computing environment: A review, *Journal of King Saud University – Computer and Information Sciences*, The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University

[111] Yelchuri Venkata Sai Harsha, Nagaraj G Cholli (2021), Load Balancing in Cloud Computing, *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 10, Issue 5.

[112] K.Balaji, P.Sai Kiran, M.Sunil Kumar (2021), Load balancing in Cloud Computing: Issues and Challenges, *Turkish Journal of Computer and Mathematics Education (TURCOMAT),* Vol.12 No.2 (2021), 3077 – 3084

[113] Roy Batchelor (1976), Box-Jenkins Analysis. *Cass Business School, City of Lodon*

PHỤ LỤC

PHỤ LỤC 1. TỔNG QUAN VỀ THUẬT TOÁN ARIMA

Theo [75], ARIMA là thuật toán dựa trên thống kê, là thuật toán tự hồi quy tích hợp trung bình trượt (Auto Regression Integrated Moving Average), được phát triển từ mô hình hồi quy ARMA (Auto Regression Moving Avera). Đây là mô hình phát triển trên số liệu chuỗi thời gian đã biết và dự báo số liệu trong tương lai gần.

1. Dữ liệu chuỗi thời gian

Dữ liệu thời gian thực hay chuỗi thời gian là một chuỗi các giá trị của một đại lượng nào đó được ghi nhận là thời gian. Bất cứ dữ liệu chuỗi thời gian nào cũng được tạo ra bằng một quá trình ngẫu nhiên.

Các giá trị của chuỗi thời gian của đại lượng X được kí hiệu là X_1, X_2, X_3,…, X_t,… , X_n với X là giá trị của X tại thời điểm t.

Một dãy số liệu thực tế cụ thể như giá bắp cải từng tháng là kết quả của một quá trình ngẫu nhiên. Đối với dữ liệu chuỗi thời gian, chúng ta có những khái niệm về tổng thể và mẫu như sau:

- Quá trình ngẫu nhiên là một tổng thể
- Số liệu thực tế sinh ra từ quá trình ngẫu nhiên là mẫu.

Chuỗi thời gian bao gồm 4 thành phần:

- Thành phần xu hướng dài hạn (long –term trend component): Thành phần này dùng để chỉ xu hướng tăng hay giảm của đại lượng X trong thời gian dài. Về mặt đồ thị thành phần này có thể biểu diễn bởi một đường thẳng hay một đường cong trơn
- Thành phần mùa (seasional component): Thành phần này dùng để chỉ xu hướng tăng hay giảm của đại lượng X tính theo mùa trong năm (có thể tính theo tháng trong năm) Ví dụ : Lượng tiêu thụ chất đốt sẽ tăng vào mùa đông và giảm vào mùa hè, ngược lại, lượng tiêu thụ xăng sẽ tăng vào mùa hè và giảm vào mùa đông.
- Thành phần chu kỳ (cyclical component) : Thành phần này chỉ sự thay đổi của đại lượng X theo chu kỳ. Thành phần này khác thành phần mùa ở chỗ chu

kỳ của đại lượng X kéo dài hơn 1 năm. Để đánh giá thành phần này các giá trị của chuỗi thời gian được quan sát hàng năm. Ví dụ, Lượng dòng chảy đến hồ Trị An từ năm 1959 – 1985

- Thành phần bất thường (irregular component): Thành phần này dùng để chỉ sự thay đổi bất thường của các giá trị trong chuỗi thời gian. Sự thay đổi này không thể dự đoán bằng các số liệu kinh nghiệm trong quá khứ, về mặt bản chất thành phần này không có tính chu kỳ.

2. Tính dừng và tính mùa vụ

a. Tính dừng của dữ liệu (Stationary)

Nếu mỗi chuỗi thời gian gọi là dừng thì trung bình, phương sai, đồng phương sai (tại các độ trễ khác nhau) sẽ giữ nguyên không đổi dù cho chúng được xác định vào thời điểm nào đi nữa.

Trung bình: $E(Y_t)$=const (Kỳ vọng không đổi theo thời gian)

Phương sai: $Var(Y_t)$=const (Phương sai không đổi theo thời gian)

Đồng phương sai: $Covar(Y_t,Y_{t-k})=g_k$ (Đồng phương sai chỉ phụ thuộc khoảng cách của độ trễ mà không phụ thuộc thời điểm tính đồng phương sai đó, không phụ thuộc t)

Để xem một chuỗi thời gian có dừng hay không, ta có thể sử dụng Đồ thị của Yt theo thời gian, Đồ thị tự tương quan mẫu (Sample Auto Correlation), hay kiểm định bước ngẫu nhiên (kiểm định Dickey-Fuller)

Nếu chuỗi Y_t không dừng, ta có thể lấy sai phân bậc 1. Khi đó chuỗi sai phân bậc 1 (Wt) sẽ có thể dừng. Sai phân bậc 1: $W_t=Y_t-Y_{t-1}$

Nếu chuỗi sai phân bậc 1 (W_t) không dừng, ta có thể lấy sai phân bậc 2. Khi đó chuỗi sai phân bậc 2 có thể dừng. Sai phân bậc 2: $V_t=W_t-W_{t-1}$

b. Hàm tự tương quan và hàm tự tương quan mẫu

Hàm tự tương quan(ACF) ở độ trễ k được ký hiệu là ρ_k được định nghĩa như sau:

Trong đó:

- pk không có thứ nguyên
- pk có giá trị từ -1 đến 1

c. Tính mùa vụ

Nếu sai phân bậc 2 mà chưa dừng, có thể chuỗi Y_t có yếu tố mùa vụ. (Nếu có yếu tố mùa vụ, tức là chuỗi vẫn chưa dừng).

Nếu cứ sau *m* thời đoạn, SAC lại có giá trị cao. Khi đó Y_t có tính mùa vụ với chu kỳ *m* thời đoạn. Phương pháp đơn giản nhất để khử tính mùa vụ là lấy sai phân thứ m

$Z_t=Y_t-Y_{t-m}$

3. Nhận dạng mô hình

Mô hình ARIMA (hay còn gọi là phương pháp Box-Jenkin)

Nhận dạng mô hình tức là xác định p, d, q trong ARIMA(p,d,q), Do mô hình Box-Jenkins chỉ mô tả chuỗi dừng hoặc những chuỗi đã sai phân hóa, nên mô hình ARIMA(p,d,q) thể hiện những chuỗi dữ liệu không dừng, đã được sai phân (ở đây, d chỉ mức độ sai phân).

Khi chuỗi thời gian dừng được lựa chọn (hàm tự tương quan ACF giảm đột ngột hoặc giảm đều nhanh), chúng ta có thể chỉ ra một mô hình dự định bằng cách nghiên cứu xu hướng của hàm tự tương quan ACF và hàm tự tương quan từng phần PACF. Theo lý thuyết, nếu hàm tự tương quan ACF giảm đột biến và hàm tự tương quan từng phần PACF giảm mạnh thì chúng ta có mô hình tự tượng quan. Nếu hàm tự tương quan ACF và hàm tự tương quan từng phần PACF đều giảm đột ngột thì chúng ta có mô hình hỗn hợp.

Về mặt lý thuyết, không có trường hợp hàm tự tương quan ACF và hàm tự tương quan từng phần cùng giảm đột ngột. Trong thực tế, hàm tự tương quan ACF và hàm tự tương quan từng phần PACF giảm đột biến khá nhanh. Trong trường hợp này, chúng ta nên phân biệt hàm nào giảm đột biến nhanh hơn, hàm còn lại được xem là giảm đều. Do đôi lúc sẽ có trường hợp giảm đột biến đồng thời khi quan sát biểu đồ hàm tự tương quan ACF và hàm tự tương quan từng phần PACF, biện pháp khắc phục là tìm vài dạng hàm dự định khác nhau cho chuỗi thời gian dừng. Sau đó, kiểm tra độ chính xác mô hình tốt nhất.

p: dựa vào SPAC

q: dựa vào SAC

d: dựa vào số lần lấy sai phân để làm cho chuỗi dừng

Mô hình ***ARIMA(1, 1, 1)*** : $y(t) - y(t-1) = a_0 + a_1(y(t-1) - y(t-2) + e(t) + b_1e(t-1))$

Hoặc $z(t) = a_0 + a_1z(t-1) + e(t) + b_1e(t-1)$,

Với $z(t) = y(t) - y(t-1)$ ở sai phân đầu tiên : $d = 1$.

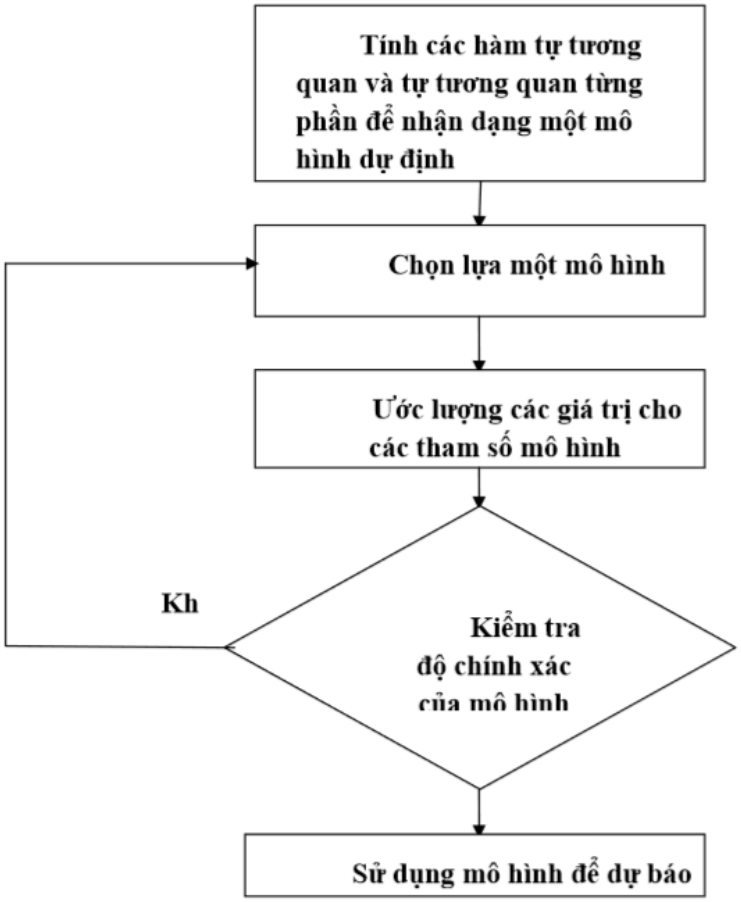
Tương tự ***ARIMA(1,2,1)*** : $h(t) = a_0 + a_1z(t-1) + e(t) + b_1e(t-1)$,

Với $h(t) = z(t) - z(t-1)$ ở sai phân thứ hai : $d = 2$. (d lớn hơn 2 rất ít được sử dụng)

4. Kiểm tra chuẩn đoán mô hình

Mô hình ARIMA tốt có RMSE nhỏ và sai số là nhiễu trắng: Sai số có phân phối chuẩn, và đồ thị SAC giảm nhanh về 0

Tìm kiếm mô hình ARIMA phù hợp là một quá trình thử và sai.



Hình 1. Sơ đồ mô phỏng mô hình Box-Jenkins .

5. Các bước phát triển mô hình ARIMA

Phương pháp Box – Jenkins bao gồm các bước chung:

- *Xác định mô hình :* Mô hình ARIMA chỉ được áp dụng đối với chuỗi dừng Mô hình có thể trình bày theo dạng AR, MA hay ARMA. Phương pháp xác định mô hình thường được thực hiện qua nghiên cứu chiều hướng biến đổi của hàm tự tương quan ACF hay hàm tự tương quan từng phần PACF.
- *Chuỗi ARIMA không dừng :* cần phải được chuyển đổi thành chuỗi dừng trước khi tính ước lượng tham số bình phương tối thiểu. Việc chuyển đổi này được thực hiện bằng cách tính sai phân giữa các giá trị quan sát dựa vào giả định các phần khác nhau của các chuỗi thời gian đều được xem xét tương tự, ngoại trừ các khác biệt ở giá trị trung bình. Nếu việc chuyển đổi này không thành công, sẽ áp dụng tiếp các kiểu chuyển đổi khác (chuyển đổi logarithm chẳng hạn).
- *Ước lượng tham số :* tính những ước lượng khởi đầu cho các tham số a_0, a_1, …, a_p, b_1, …, b_q của mô hình dự định. Sau đó xây dựng những ước lượng sau cùng bằng một quá trình lặp.
- *Kiểm định độ chính xác :* Sau khi các tham số của mô hình tổng quát đã xây dựng, ta kiểm tra mức độ chính xác và phù hợp của mô hình với dữ liệu. Chúng ta kiểm định phần dư (Y_t –Y^t) và có ý nghĩa cũng như mối quan hệ các tham số. Nếu bất cứ kiểm định nào không thỏa mãn, mô hình sẽ nhận dạng lại các bước trên được thực hiện lại.
- *Dự báo:* Khi mô hình thích hợp với dữ liệu đã tìm được, ta sẽ thực hiện dự báo tại thời điểm tiếp theo t. Do đó, mô hình ARMA(p,q) :

$$y(t+1) = a_0 + a_1y(t) + \dots + a_py(t - p + 1) + e(t+1) + b_1e(t) + \dots + b_qe(t - q + 1)$$

PHỤ LỤC 2. CÁC HÀM/PHƯƠNG THỨC ĐƯỢC CÀI ĐẶT BỔ SUNG VÀO CLOUDSIM

```
public class VmExt extends Vm {
   private List<Double> LastRT;
   private double PredictedRT;
   public VmExt(
         int id,
         int userId,
         double mips,
         int numberOfPes,
         int ram,
         long bw,
         long size,
         String vmm,
         CloudletScheduler cloudletScheduler) {
      super(id, userId, mips, numberOfPes, ram, bw, size, vmm, cloudletScheduler);
      LastRT = new ArrayList<Double>();
   }
   public double getPredictedRT() {
      List<Double> arraylist = getLastRT();
      if(arraylist.size()>15){
      double[] dataArray = new double[arraylist.size() - 1];
      for (int i = 0; i < arraylist.size() - 1; i++) {
         dataArray[i] = arraylist.get(i);
      }
      ARIMA arima = new ARIMA(dataArray);
      int[] model = arima.getARIMAmodel();
      this.PredictedRT = arima.aftDeal(arima.predictValue(model[0], model[1]));
      }
      else
      {
         this.PredictedRT =arraylist.get(arraylist.size()-1);
      }
      return this.PredictedRT;
```

```
    }

    public List<Double> getLastRT() {
        return this.LastRT;
    }
    public void addLastestRT(Double RT) {
        LastRT.add(RT);
        if (LastRT.size() > 50) {
            LastRT.remove(0);
        }
    }
}
public class CloudletExt extends Cloudlet {
    private double responseTime;
    public CloudletExt(final int cloudletId,
            final long cloudletLength,
            final int pesNumber,
            final long cloudletFileSize,
            final long cloudletOutputSize,
            final UtilizationModel utilizationModelCpu,
            final UtilizationModel utilizationModelRam,
            final UtilizationModel utilizationModelBw) {
        super(
                cloudletId,
                cloudletLength,
                pesNumber,
                cloudletFileSize,
                cloudletOutputSize,
                utilizationModelCpu,
                utilizationModelRam,
                utilizationModelBw,
                false);
    }
```

```
        public double getResponseTime() {
            return this.getFinishTime() - this.getExecStartTime();
        }
    }
    public class ArimaDatacenterBroker extends SimEntity {
        private static List<Double> LastRT;
private double PredictedRT;
…
public double getPredictedRT() {
        List<Double> arraylist = getLastRT();
        if(arraylist.size()>50){
        double[] dataArray = new double[arraylist.size() - 1];
        for (int i = 0; i < arraylist.size() - 1; i++) {
            dataArray[i] = arraylist.get(i);
        }
        ARIMA arima = new ARIMA(dataArray);
        int[] model = arima.getARIMAmodel();
        this.PredictedRT = arima.aftDeal(arima.predictValue(model[0], model[1]));
        }
        else
        {
            this.PredictedRT =arraylist.get(arraylist.size()-1);
        }
        return this.PredictedRT;
    }

    public List<Double> getLastRT() {
        return this.LastRT;
    }
    public void addLastestRT(Double RT) {
        LastRT.add(RT);
        if (LastRT.size() > 200) {
            LastRT.remove(0);
        }
```

```
    }
    protected VmExt getReliableVm()
    {
        VmExt res = null;
        double[] DistanceList  = new  double[vmsCreatedList.size()-1];
        int i=0;
        double predicted = getPredictedRT();
        for(VmExt vm: vmsCreatedList)
        {
            DistanceList[i] = Math.abs(vm.getPredictedRT() - predicted);
            i++;
            if(vm.getPredictedRT() < predicted)
            {
                return vm;
                //break;
            }
        }
        int index =getMin(DistanceList);
        return vmsCreatedList.get(index);
    }
        public static int getMin(double[] inputArray){
    double minValue = inputArray[0];
    int index = 0;
    for(int i=1;i<inputArray.length;i++){
      if(inputArray[i] < minValue){
        minValue = inputArray[i];
        index = i;
      }
    }
    return index;
  }
protected void processCloudletReturn(SimEvent ev) {
        CloudletExt cloudlet = (CloudletExt) ev.getData();
        VmExt vm = VmList.getById(vmsCreatedList, cloudlet.vmId);
```

```
            vm.addLastestRT(cloudlet.getResponseTime());
            addLastestRT(cloudlet.getResponseTime());
            getCloudletReceivedList().add(cloudlet);
            Log.printConcatLine(CloudSim.clock(), ": ", getName(), ": Cloudlet ",
cloudlet.getCloudletId(),
                    " received");
            cloudletsSubmitted--;
            if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed
                Log.printConcatLine(CloudSim.clock(), ": ", getName(), ": All Cloudlets executed.
Finishing...");
                clearDatacenters();
                finishExecution();
            } else { // some cloudlets haven't finished yet
                if (getCloudletList().size() > 0 && cloudletsSubmitted == 0) {
                    // all the cloudlets sent finished. It means that some bount
                    // cloudlet is waiting its VM be created
                    clearDatacenters();
                    createVmsInDatacenter(0);
                }
            }
        }
    protected void submitCloudlets() {
            int vmIndex = 0;
            List<CloudletExt> successfullySubmitted = new ArrayList<CloudletExt>();
            for (CloudletExt cloudlet : getCloudletList()) {
                VmExt vm;
                // if user didn't bind this cloudlet and it has not been executed yet
                if (cloudlet.getVmId() == -1) {
                    vm = getVmsCreatedList().get(vmIndex);
                } else { // submit to the specific vm
                    vm = getReliableVm();
                    if (vm == null) { // vm was not created
                        if (!Log.isDisabled()) {
```

```
                    Log.printConcatLine(CloudSim.clock(), ": ", getName(), ": Postponing execution
of cloudlet ",
                            cloudlet.getCloudletId(), ": bount VM not available");
                }
                continue;
            }
        }
        if (!Log.isDisabled()) {
            Log.printConcatLine(CloudSim.clock(), ": ", getName(), ": Sending cloudlet ",
                cloudlet.getCloudletId(), " to VM #", vm.getId());
        }
        cloudlet.setVmId(vm.getId());
        sendNow(getVmsToDatacentersMap().get(vm.getId()),
CloudSimTags.CLOUDLET_SUBMIT, cloudlet);
        cloudletsSubmitted++;
        vmIndex = (vmIndex + 1) % getVmsCreatedList().size();
        getCloudletSubmittedList().add(cloudlet);
        successfullySubmitted.add(cloudlet);
      }
      // remove submitted cloudlets from waiting list
      getCloudletList().removeAll(successfullySubmitted);
    }
```